ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αυτοματοποίηση δέσμευσης πόρων για εκτέλεση εφαρμογών μεγάλης κλίμακας σε περιβάλλοντα υπολογιστικών νεφών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Θανάς Λεράι

Επιβλέπων:
Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2021

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Αυτοματοποίηση δέσμευσης πόρων για εκτέλεση εφαρμογών μεγάλης κλίμακας σε περιβάλλοντα υπολογιστικών νεφών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Λεράι Θανάς

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 15η Ιουνίου 2021.

(Υπογραφή)	(Υπογραφή)	(Υπογραφή)
.....	.....	.....
Νεκτάριος Κοζύρης	Ιωάννης Κωνσταντίνου	Νικόλαος Παπασπύρου
Καθηγητής	Επικ. Καθηγητής	Καθηγητής
Ε.Μ.Π.	Παν. Θεσσαλίας	Ε.Μ.Π.

Αθήνα, Ιούνιος 2021

(Υπογραφή)

.....

ΛΕΡΑΙ ΘΑΝΑΣ
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η ανάγκη για επεξεργασία μεγάλου όγκου δεδομένων (Big Data) γίνεται ολοένα και πιο μεγάλη τα τελευταία χρόνια. Λόγω της ανάγκης αυτής, αναπτύχθηκαν νέες τεχνολογίες οι οποίες στις περισσότερες περιπτώσεις φιλοξενούνται σε διαμοιραζόμενους υπολογιστικούς πόρους, στα λεγόμενα υπολογιστικά νέφη.

Η πλειοψηφία των οργανισμών που διαχειρίζονται μεγάλο όγκο δεδομένων επιλέγουν τη μεταφορά των εγκαταστάσεών τους από τοπικούς υπολογιστές και servers του οργανισμού, σε υπολογιστικά νέφη. Τα πλεονεκτήματα από αυτήν την μεταφορά είναι η εξοικονόμηση κερδών, η δυνατότητα κλιμάκωσης, η ασφάλεια των δεδομένων τους και η απαραίτητη τεχνική υποστήριξη η οποία διατίθεται από τον πάροχο ανά πάσα στιγμή.

Μία από τις υπηρεσίες που παρουσίασαν ραγδαία άνοδο τα τελευταία χρόνια είναι τα Spot Instances της Amazon. Με την υπηρεσία αυτή η Amazon διαθέτει σε χαμηλές τιμές αχρησιμοποίητες εικονικές μηχανές της. Οι χρήστες λαμβάνουν τις μηχανές πραγματοποιώντας προσφορές. Σε περίπτωση που η προσφορά κάποιου νέου χρήστη είναι υψηλότερη από την τρέχουσα για τον ίδιο υπολογιστικό πόρο, τότε ο παλιός χρήστης ειδοποιείται και χάνει την εικονική μηχανή του. Μερικές εταιρείες που χρησιμοποιούν τη συγκεκριμένη υπηρεσία είναι: AOL, BloomReach, Mapbox, Moovit, Zillow, Novartis κ.α.

Στα πλαίσια της παρούσας διπλωματικής εργασίας περιγράφουμε και αναλύουμε μια μέθοδο η οποία έχει ως στόχο την ελαχιστοποίηση του οικονομικού κόστους χρήσης της υπηρεσίας των Spot Instances για χρήστες οι οποίοι δεν γνωρίζουν εκ των προτέρων το μέγεθος του υπολογιστικού φόρτου των εργασιών που επιθυμούν να εκτελέσουν.

Για την επίλυση του προβλήματος βασιζόμαστε σε κάποιες βασικές αλγοριθμικές τεχνικές για την ανάπτυξη αλγορίθμων που ειδικεύονται σε προβλήματα βελτιστοποίησης, ενώ για τις προβλέψεις των άγνωστων παραμέτρων, οι οποίες στο πρόβλημα μας είναι ο φόρτος εργασιών και η αγοραία τιμή των υπολογιστικών πόρων, χρησιμοποιούμε τεχνικές μηχανικής μάθησης, και συγκεκριμένα τα νευρωνικά δίκτυα.

Οι προσεγγίσεις και οι προσομοιώσεις που πραγματοποιήσαμε βασίζονται εξ ολοκλήρου σε πραγματικά δεδομένα. Παρουσιάζουμε τα αποτελέσματα των πειραμάτων μας, αναλύουμε ποιοτικά τα αποτελέσματα αυτά και ολοκληρώνουμε με μία σύνοψη των πεπραγμένων μας και την αναφορά σε πιθανές μελλοντικές επεκτάσεις της δουλειάς μας.

Λέξεις Κλειδιά

Υπολογιστικό Νέφος, Spot Instances, Αλγοριθμικές Τεχνικές, Πρόβλεψη Παραμέτρων, Μηχανική Μάθηση, Νευρωνικά Δίκτυα

Abstract

The need for Big Data processing has witnessed a severe increment over the last years. Due to this need, new technologies were developed which in most cases are hosted on shared computing resources, in the so-called Cloud Services.

The vast majority of corporations that manipulate Big Data rely on shared computing resources instead of their own local servers and personal devices. The advantages of this migration are the cost savings, the possibility of scaling, the security of their data and the necessary technical support which is available from the cloud provider at any time.

One of the cloud services that has rapidly risen in recent years is Amazon Spot Instances. In this cloud service, Amazon sells unused virtual machines at low prices. Users with high enough bids for these machines occupate them. In the case where, a new user bids a larger amount of money for the virtual machine of an old user, then the old user is warned about an interruption and its virtual machine is granted to the new user. Examples of modern enterprises that use Spot Instances are: AOL, BloomReach, Mapbox, Moovit, Zillow, Novartis and others.

In this specific diploma thesis, we describe and analyze a method which aims to minimize the financial cost of using the Spot Instances service for users who do not know in advance the size of the computational workload of the tasks they wish to execute.

To solve the problem we rely on some basic algorithmic techniques for the development of algorithms specializing in optimization problems, while for the predictions of the unknown parameters, which in our problem are the workload and the price of the virtual machines, we use machine learning techniques, and neural networks in particular.

The approaches and simulations we performed are based entirely on real data. We present the results of our experiments, we qualitatively analyze these results and we conclude with a summary of our thesis and the reference to possible future extensions of our work.

Key Words

Cloud Computing, Spot Instances, Algorithmic Techniques, Parameters Prediction, Machine Learning, Neural Networks

Ευχαριστίες

Η παρούσα διπλωματική εργασία σηματοδοτεί το τέλος ενός πολύ όμορφου κύκλου της ζωής μου, αυτού των προπτυχιακών μου σπουδών. Έχοντας φτάσει στο σημείο αυτό, νιώθω την ανάγκη να ευχαριστήσω μερικούς ανθρώπους, οι οποίοι εμπλούτισαν και έκαναν πιο συναρπαστικό αυτό το ταξίδι.

Αρχικά, θα ήθελα να ευχαριστήσω τον καθηγητή κ. Νεκτάριο Κοζύρη, ο οποίος μου έδωσε την ευκαιρία να μελετήσω υπό την επίβλεψή του ένα τόσο ενδιαφέρον θέμα που συνδυάζει πολλαπλούς επιστημονικούς κλάδους. Επίσης θα ήθελα να ευχαριστήσω τον καθηγητή κ. Ιωάννη Κωνσταντίνου για την καθοδήγηση και όλη την υποστήριξη που μου παρείχε. Θέλω επίσης να ευχαριστήσω τον υποψήφιο διδάκτορα Κωνσταντίνο Μπιτσάκο ο οποίος ήταν εκεί για να λύνει οποιαδήποτε απορία μου και να μου προτείνει συνεχώς νέες ιδέες.

Μέσα από όλη την εμπειρία των φοιτητικών μου χρόνων, γνώρισα μερικούς ανθρώπους οι οποίοι πέραν του ότι είναι εξαιρετικοί επιστήμονες, είναι και πολύ ξεχωριστοί μου φίλοι. Θα ήθελα λοιπόν να ευχαριστήσω τον Νίκο, ο οποίος αποτελούσε πάντα την πρώτη επιλογή συνεργάτη σε όλες τις εργασίες, τον Αστέριο, ο οποίος με παρότρυνε συνεχώς να γίνομαι καλύτερος φοιτητής, και τον Τόλη, που με τις συζητήσεις μας με έκανε συνεχώς να αναρωτιέμαι για το τι θα ήθελα να κάνω στο μέλλον.

Θα ήθελα επίσης να ευχαριστήσω ξεχωριστά τον πολύ καλό μου φίλο Αλέξη, που μέσα από τις αμέτρητες βόλτες και τις κουβέντες μας διαμόρφωσε ένα πολύ σημαντικό κομμάτι του χαρακτήρα μου και με έκανε να βλέπω τα πράγματα από μια διαφορετική οπτική γωνία.

Ένα ιδιαίτερα μεγάλο ευχαριστώ πηγαίνει στην μεγάλη μου αδελφή Ελβίρα, η οποία μου έκανε πάντα όλα τα χατίρια και ήταν πάντοτε εκεί για να ακούει όλα μου τα προβλήματα και να μου παρέχει την απαραίτητη υποστήριξη.

Το μεγαλύτερο ευχαριστώ όμως πηγαίνει στους γονείς μου, Μαρία και Γρηγόρη, προτίστως για τις αρχές και τις αξίες που μου μετέδωσαν και με διαμόρφωσαν ως άνθρωπο, αλλά κυρίως για την συνεχή ενθάρρυνση και παρότρυνση να κυνηγάω τα όνειρά μου και να προσπαθώ πάντοτε για το καλύτερο.

Θανάς Λεράι
$15^{η}$ Ιουνίου 2021

Στους γονείς μου, που θυσίασαν αυτά των οποίων
τη θυσία εγώ δεν διαπραγματεύομαι.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Αντικείμενο της εργασίας

Η συγκεκριμένη διπλωματική εργασία εστιάζει στην οικονομική δέσμευση υπολογιστικών πόρων για χρήστες οι οποίοι εκτελούν εργασίες χρησιμοποιώντας την υπηρεσία Spot Instances της Amazon. Το πρίσμα υπό το οποίο πραγματοποιήθηκε η μελέτη μας αφορά το σενάριο όπου οι χρήστες δεν γνωρίζουν εκ των προτέρων το φόρτο των προς εκτέλεση εργασιών. Για την κατανόηση των προσεγγίσεων απαιτούνται γνώσεις από διαφορετικούς επιστημονικούς κλάδους.

Αρχικά, θα πρέπει να γίνει μία πλήρης εισαγωγή στο σύστημα Spot Instances, στις βασικές έννοιές του και στην αρχιτεκτονική του. Γίνεται μια παρουσίαση κάποιων στρατηγικών χρήσης του συστήματος. Στο πλαίσιο της διπλωματικής εργασίας, επικεντρωνόμαστε στην περιγραφή και την ανάλυση μιας μεθόδου που στοχεύει στη βέλτιστη χρήση μιας από τις εν λόγω στρατηγικές.

Μιλώντας όμως για οικονομική δέσμευση υπολογιστικών πόρων, δηλαδή για ελαχιστοποίηση του χρηματικού κόστους χρήσης της υπηρεσίας των Spot Instances, καταλαβαίνει κανείς πως αναφερόμαστε σε ένα πρόβλημα βελτιστοποίησης. Η συνήθης μεθοδολογία για την επίλυση ενός προβλήματος βελτιστοποίησης είναι η κατασκευή ενός αλγορίθμου. Μελετάμε λοιπόν δύο συγκεκριμένες αλγοριθμικές τεχνικές για την επίλυση προβλημάτων βελτιστοποίησης, τους Άπληστους Αλγορίθμους και το Δυναμικό Προγραμματισμό.

Υπάρχουν όμως κάποιοι παράγοντες που δημιουργούν προκλήσεις στην προσπάθειά μας για την βελτιστοποίηση του κόστους χρήσης της υπηρεσίας αυτής. Οι δύο αλγοριθμικές τεχνικές που αναφέραμε στηρίζονται στην εκ των προτέρων γνώση δύο συγκεκριμένων παραμέτρων: το φόρτο εργασιών που πρόκειται να εκτελέσουμε και την τιμή των υπολογιστικών πόρων που επιθυμούμε να αγοράσουμε προκειμένου να εκτελέσουμε τις εργασίες αυτές. Όμως, και οι δύο παράμετροι είναι άγνωστες. Το κλειδί βρίσκεται λοιπόν στην πρόβλεψη αυτών των δύο παραμέτρων.

Θα χρησιμοποιήσουμε τεχνικές μηχανικής μάθησης, και πιο συγκεκριμένα των Νευρωνικών Δικτύων, με τα οποία θα είμαστε σε θέση να κάνουμε προβλέψεις για τις άγνωστες παραμέτρους, προκειμένου να συνδυαστούν και με την χρήση των προαναφερθέντων αλγοριθμικών τεχνικών να φτάσουμε στον στόχο μας: την ελαχιστοποίηση του κόστους χρήσης της υπηρεσίας.

Η ανάλυση δεδομένων και οι προσομοιώσεις πραγματοποιήθηκαν σε γλώσσα προγραμματισμού Python. Οι παράμετροι καθώς και τα σύνολα δεδομένων που χρησιμοποιήθηκαν είναι επιλεγμένα με βάση διεθνείς εργασίες με σκοπό τις πλέον ρεαλιστικές προσομοιώσεις.

1.2 Οργάνωση του Τόμου

Η παρούσα διπλωματική εργασία χωρίζεται σε 6 κεφάλαια.

Στο **Κεφάλαιο** 1 πραγματοποιούμε μία εισαγωγή στο αντικείμενο της εργασίας και παρουσιάζεται η συνολική δομή της διπλωματικής.

Στο **Κεφάλαιο** 2 παρουσιάζεται το σύστημα Spot Instances το οποίο αποτέλεσε κίνητρο για την παρούσα διπλωματική εργασία. Γίνεται μια εισαγωγή στο σύστημα και τις βασικές έννοιές του, καθώς και τις κυρίαρχες στρατηγικές χρήσης του εν λόγω και άλλων παρόμοιων υπηρεσιών. Τέλος περιγράφεται η δική μας σκοπιά μελέτης του συστήματος και οι ιδέες για την αξιοποίησή του.

Στο **Κεφάλαιο** 3 παρουσιάζονται κάποιες βασικές αλγοριθμικές τεχνικές οι οποίες χρησιμοποιήθηκαν για την επίλυση του προβλήματος βέλτιστης της δέσμευσης πόρων. Πρόκειται για τους Άπληστους Αλγορίθμους (Greedy Algorithms) και το Δυναμικό Προγραμματισμό (Dynamic Programming). Για κάθε μια από τις τεχνικές αυτές γίνεται μια περιγραφή της βασικής της ιδέας και παρουσιάζεται και από ένα παράδειγμα που στοχεύει στην καλύτερη κατανόησή της.

Στο **Κεφάλαιο** 4 γίνεται μια παρουσίαση των Νευρωνικών Δικτύων. Αρχικά αναφερόμαστε στο Perceptron και στη χρησιμότητά του στο να ταξινομεί δεδομένα τα οποία ανήκουν σε γραμμικά διαχωρίσιμες κλάσεις. Στη συνέχεια αναφερόμαστε στη δομή και την χρησιμότητα των νευρωνικών δικτύων πολλαπλών στρωμάτων και αναλύουμε τον αλγόριθμο εκπαίδευσής τους, τον Back Propagation. Έπειτα, παρουσιάζουμε κάποια παραδείγματα αλγορίθμων βελτιστοποίησης Gradient Descent. Τέλος, γίνεται μια παρουσίαση της δομής και της λειτουργίας μιας συγκεκριμένης κατηγορίας νευρωνικών δικτύων, τα Αναδρομικά Νευρωνικά Δίκτυα, και συγκεκριμένα τα δίκτυα τύπου LSTM, τα οποία παίζουν καθοριστικό ρόλο στην πρόβλεψη χρονοσειρών.

Στο **Κεφάλαιο** 5 παρουσιάζεται η θεωρητική θεμελίωση πάνω στην οποία στηριχθήκαμε για να πραγματοποιήσουμε τα πειράματα και τις προσομοιώσεις μας. Εισάγουμε τις απραίτητες έννοιες και ορισμούς που πλαισιώνουν το πρόβλημα που έχουμε θέσει ως στόχο, απαριθμούμε τις παραμέτρους που απαιτούνται για την επίλυσή του και αναφέρουμε τις παραδοχές υπό τις οποίες υλοποιήθηκε η μελέτη μας. Στη συνέχεια, διατυπώνουμε δύο αλγορίθμους οι οποίοι επιλύουν βέλτιστα το πρόβλημα μας. Τέλος, γίνεται μια περιγραφή της αρχιτεκτονικής των νευρωνικών δικτύων που χρησιμοποιήσαμε για την πρόβλεψη των άγνωστων παραμέτρων τις οποίες δίνουμε ως είσοδο στους αλγορίθμους μας και αναλύουμε την διαδικασία που ακολουθήθηκε για την κατασκευή δεδομένων εκπαίδευσης των νευρωνικών δικτύων.

Στο **Κεφάλαιο** 6 παρουσιάζουμε τα πειραματικά αποτελέσματα των προσεγγίσεών μας για την επίλυση του προβλήματος της οικονομικής δέσμευσης πόρων. Αρχικά, παρουσιάζουμε το σύνολο των δεδομένων εκπαίδευσης που χρησιμοποιήσαμε για την εκπαίδευση του νευρωνικού δικτύου για την πρόβλεψη του φόρτου εργασιών και παρουσιάζονται τα αποτελέσματα για όλες τις προσεγγίσεις που χρησιμοποιήσαμε. Στη συνέχεια, επαναλαμβάνουμε την ίδια διαδικασία για το νευρωνικό δίκτυο για την πρόβλεψη της τιμής των υπολογιστικών πόρων της υπηρεσίας. Τέλος, παρουσιάζουμε τα αποτελέσματα από τις προσομοιώσεις μας για την οικονομικά βέλτιστη δέσμευση πόρων με την χρήση των αλγοριθμικών τεχνικών που αναφέραμε προηγουμένως και γίνεται μια ποιοτική ανάλυση πάνω στα αποτελέσματα αυτά.

Στο **Κεφάλαιο** 7 πραγματοποιείται ο επίλογος της παρούσας διπλωματικής εργασίας. Γίνεται μία σύνοψη των αποτελεσμάτων και παραθέτονται κάποια βασικά συμπεράσματα. Τέλος, παρουσιάζονται πιθανές μελλοντικές κατευθύνσεις που θα έχουν ως έναυσμα την παρούσα εργασία.

Κεφάλαιο 2

Το σύστημα Spot Instances

2.1 Περιγραφή του Συστήματος

Στην ενότητα αυτή θα παρουσιάσουμε το σύστημα Spot Instances [1] της Amazon, το οποίο αποτέλεσε έναυσμα για την συγκεκριμένη διπλωματική εργασία.

2.1.1 Η ιδέα

Η ανάγκη για τη χρήση των εικονικών μηχανών της Amazon, οι οποίες δεν έχουν δοθεί ακόμα σε κάποιον χρήστη, οδήγησε στην ιδέα των Spot Instances. Ένα Spot Instance είναι μια εικονική μηχανή που δεν χρησιμοποιείται. Η Amazon λοιπόν αποφάσισε για αυτόν τον λόγο να διαθέσει αυτές τις εικονικές μηχανές σε χαμηλότερες τιμές από ότι οι κανονικές (On Demand) εικονικές μηχανές.

Η τιμή του Spot Instance αλλάζει ανά μία ώρα, ενώ μπορεί να διαφέρει ανάλογα με τη γεωγραφική ζώνη. Στο διάστημα της μίας ώρας η τιμή του Spot Instance δεν μεταβάλλεται. Για να περάσει ένα Spot Instance στην κυριότητα ενός χρήστη, θα πρέπει να προσφέρει τιμή μεγαλύτερη ή ίση από την τιμή του Spot Instance εκείνη τη στιγμή. Εαν η προσφορά του κάθε χρήστη γίνει μικρότερη από την τιμή του Spot Instance, τότε η Amazon δικαιούται να πάρει πίσω μέσα σε πολύ μικρό χρονικό διάστημα το Spot Instance από το χρήστη και να το διαθέσει αλλού. Αυτό μπορεί να συμβεί εαν για παράδειγμα υπάρχουν μεγαλύτερες προσφορές ή έχει μειωθεί το πλήθος των διαθέσιμων Spot Instances. Ο υπολογισμός της τιμής αυτής γίνεται σταδιακά και βασίζεται στη μακροπρόθεσμη ζήτηση και προσφορά για Spot Instances.

Τα Spot Instances είναι σε γενικές γραμμές μία οικονομική προσέγγιση για χρήστες οι οποίοι τρέχουν εφαρμογές οι οποίες δεν επεξεργάζονται κρίσιμο φορτίο και μπορούν να διακοπούν. Τέτοιες εφαρμογές μπορεί να είναι οι εργασίες ανάλυσης δεδομένων, τα batch jobs, το background processing, τα optional tasks κ.λ.π.

2.1.2 Βασικές έννοιες

Παρουσιάζονται παρακάτω οι βασικές έννοιες των Spot Instances:

- ***Spot Instance Pool***: Ένα σύνολο αχρησιμοποίητων εικονικών μηχανών της Amazon που είναι του ίδιου τύπου, έχουν ίδιο λογισμικό, ανήκουν στην ίδια γεωγραφική ζώνη και χρησιμοποιούν την ίδια πλατφόρμα δικτύου.

- ***Spot Price***: Η τιμή ενός Spot Instance για την τρέχουσα ώρα.

- ***Spot Instance Request***: Με αυτό παρέχεται η μέγιστη τιμή που κάποιος επιθυμεί να δώσει ανά ώρα για ένα Spot Instance. Αν η τιμή αυτή δεν προσδιοριστεί από τον χρήστη, τότε η Amazon ορίζει ως μέγιστη τιμή την τρέχουσα τιμή των κανονικών εικονικών μηχανών που διαθέτει. Αν η μέγιστη τιμή ξεπεράσει την Spot Price και εφόσον υπάρχει διαθέσιμη χωρητικότητα, τότε η Amazon καταχωρεί το Spot Instance στο χρήστη. Ένα Spot Instance Request μπορεί να είναι μίας φοράς (one-time) ή διαρκές (persistent). Αν είναι διαρκές, τότε μόλις σταματήσει το Spot Instance το οποίο είναι συνδεδεμένο με το Request, η Amazon αυτόματα πραγματοποιεί ξανά το Request. Ένα Spot Instance Request μπορεί να περιέχει διάρκεια για τη χρήση των Spot Instances.

- ***Spot Fleet***: Ένα σύνολο από Spot Instances το οποίο ξεκινάει και πληροί τα κριτήρια που έχει ορίσει ο χρήστης. Το Spot Fleet είναι υπεύθυνο για την επιλογή των Spot Instance Pools που ικανοποιούν τις ανάγκες του χρήστη καθώς και για την έναρξή τους. Τα Spot Fleets ορίζονται προκειμένου να διατηρούν τις απαιτήσεις του χρήστη σε υπολογιστικούς πόρους αντικαθιστώντας τα Spot Instances τα οποία τερματίζονται με άλλα ισοδύναμα. Ένα Spot Fleet μπορεί να ζητηθεί σε one-time Request, το οποίο αποτελείται από την στοχευμένη χωρητικότητα, μια ή περισσότερες προδιαγραφές εκκίνησης, την μέγιστη τιμή που ο χρήστης είναι διατεθειμένος να πληρώσει και προαιρετικά μια ποσότητα από On Demand εικονικές μηχανές.

- ***Spot Instance Interruption***: Η Amazon μπορεί να διακόψει (terminate), σταματήσει (stop) και πάψει (hibernate) ένα Spot Instance όταν το Spot Price ξεπεράσει τη μέγιστη προσφορά του χρήστη ή όταν δεν υπάρχει διαθέσιμη εικονική μηχανή. Παρέχει βέβαια ένα μήνυμα διακοπής 2 λεπτά πριν τερματιστεί το Spot Instance.

2.1.3 Διαφορές Μεταξύ Spot Instances και Κανονικών Εικονικών Μηχανών

Στον παρακάτω πίνακα παρουσιάζονται οι βασικές διαφορές μεταξύ των Spot Instances και των κανονικών (On-Demand) εικονικών μηχανών που παρέχει η Amazon.

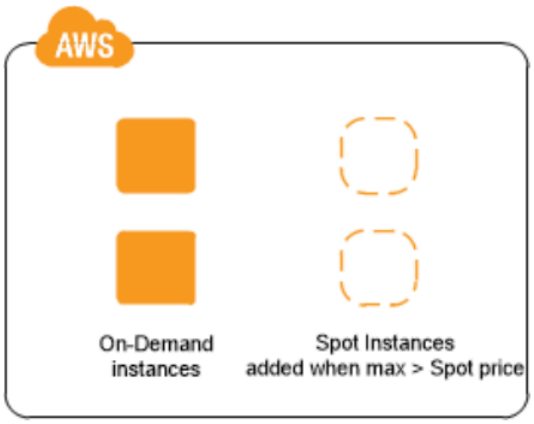
	Spot Instances	**On Demand Instances**
Launch Time	Μπορούν να ξεκινήσουν άμεσα μόνο αν το Spot Request είναι ενεργό και υπάρχει διαθεσιμότητα.	Μπορούν να ξεκινήσουν άμεσα αν πραγματοποιηθεί manual αίτημα εκκίνησης και υπάρχει διαθεσιμότητα.
Available Capacity	Αν δεν υπάρχει διαθεσιμότητα το Spot Request συνεχίζει αυτόματα να πραγματοποιεί launch request μέχρι να υπάρξει διαθέσιμη χωρητικότητα.	Αν δεν υπάρχει διαθεσιμότητα όταν γίνεται launch request, τότε ο χρήστης λαμβάνει error.
Hourly Price	Η τιμή για ένα Spot Instance αλλάζει ανάλογα με τη ζήτηση.	Η τιμή για ένα On-Demand Instance είναι στατική.
Instance Interruption	Ο χρήστης δε μπορεί να σταματήσει και να ξεκινήσει ένα Spot Instance. Αυτό μπορεί να το κάνει μόνο η Amazon αν δεν υπάρχει διαθεσιμότητα, αν η Spot Price ξεπερνά τη μέγιστη προσφορά του χρήστη ή αν αυξηθεί η ζήτηση για Spot Instances.	Ο χρήστης καθορίζει πότε θα διακόψει ένα On-Demand Instance.

Πίνακας 2.1: Διαφορές Μεταξύ Spot Instances και On-Demand Εικονικών Μηχανών

2.1.4 Στρατηγικές Χρήστης και Αρχιτεκτονική Συστήματος

Παρακάτω παρουσιάζονται δύο συνήθεις στρατηγικές χρήσης που έχουν παρατηρηθεί για την υπηρεσία Spot Instances.

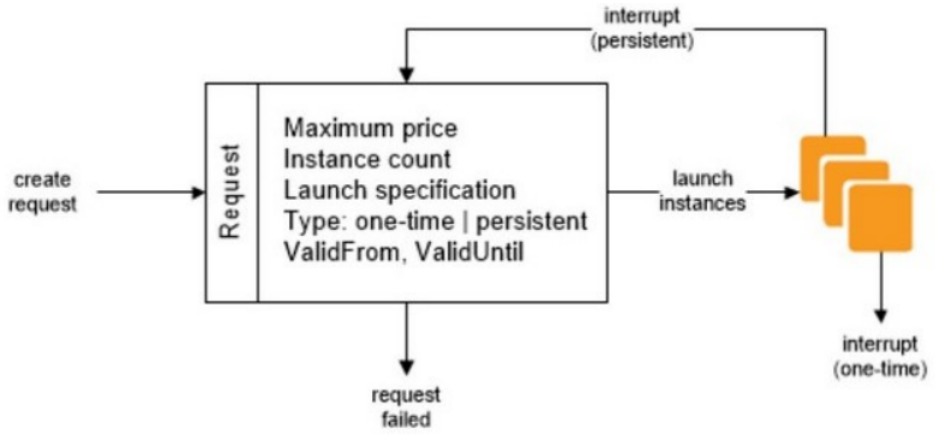
Η φιλοσοφία πίσω από την πρώτη στρατηγική αφορά τη διατήρηση ενός ελαχίστου επιπέδου υπολογιστικών πόρων από On-Demand Istances προκειμένου να τρέχουν οι εφαρμογές του χρήστη και τον εμπλουτισμό του με Spot Instances κάθε φορά όπου η μέγιστη προσφορά του χρήστη είναι μεγαλύτερη από το Spot Price. Στο παρακάτω σχήμα παρουσιάζεται η ιδέα αυτής της τεχνικής.



Εικόνα 2.1: Πρώτη Στρατηγική Χρήσης της Υπηρεσίας Spot Instances

Η ιδέα της δεύτερης στρατηγικής αφορά τη χρήση των Spot Instances για ένα συγκεκριμένο χρονικό διάστημα (Spot Block) με σκοπό να μη διακόπτονται και να τρέχουν συνεχόμενα για τη διάρκεια που έχει επιλέξει ο χρήστης. Είναι πολύ σπάνιο για ένα Spot Block να διακοπεί λόγω έλλειψης διαθεσιμότητας. Σε μια τέτοια περίπτωση ο χρήστης λαμβάνει μήνυμα δύο λεπτά πριν τον τερματισμό και δε χρεώνεται ακόμα και αν χρησιμοποίησε τα Spot Instances.

Στο παρακάτω σχήμα απεικονίζεται η συνολική αρχιτεκτονική του συστήματος.



Εικόνα 2.2: Η Αρχιτεκτονική της Υπηρεσίας Spot Instances

Στο πλαίσιο της διπλωματικής εργασίας εστιάζουμε στην βελτιστοποίηση της χρήσης της πρώτης στρατηγικής, και ειδικότερα στο κομμάτι της βέλτιστης δέσμευσης Spot Instances. Πιο συγκεκριμένα, θεωρούμε ότι ο χρήστης πραγματοποιεί πάντα μια αρκετά υψηλή προσφορά για να διασφαλίσει τα Spot Instances που χρειάζεται, και επικεντρωνόμαστε στον καθορισμό του οικονομικά πιο συμφέροντος πλήθους από Spot Instances για την εκτέλεση των επιθυμητών εργασιών.

2.1.5 Παρόμοιες Υπηρεσίες

Μια παρόμοια υπηρεσία αποτελεί η Preemptible VM (PVM) της Google [1]. Στην εν λόγω υπηρεσία, ο χρήστης μπορεί να αποκτήσει μία εικονική μηχανή σε χαμηλή τιμή της οποίας όμως η λειτουργία θα διακοπεί σε μόλις 24 ώρες μετά την απόκτησή της. Η Google έχει τη δυνατότητα να αποσύρει την εικονική μηχανή για λόγους διαθεσιμότητας. Τα PVMs χρησιμοποιούνται για λόγους testing, για σύντομες batch jobs και για fault tolerant εφαρμογές. Με τα PVMs γίνεται εξοικονόμηση μέχρι και 80% για το χρήστη σε σχέση με μία κανονική εικονική μηχανή της Google. Η βασική διαφορά τους με τα Spot Instances είναι πως η τιμή των PVMs παραμένει σταθερή, ενώ η τιμή των Spot Instances (Spot Price) αλλάζει δυναμικά. Επίσης τα PVMs πάντα διακόπτονται και πρέπει να επανεκκινηθούν manually ενώ τα Spot Instances απλά σταματούν και μπορούν να επανεκκινηθούν αυτόματα όταν υπάρξει διαθεσιμότητα ή όταν η προσφορά του χρήστη επαρκεί για την απόκτησή τους.

Μία άλλη υπηρεσία που παρέχει οικονομικές εικονικές μηχανές είναι η υπηρεσία Droplets της DigitalOcean η οποία λειτουργεί με τη λογική των On-Demand Instances της Amazon. Παρόλα αυτά, οι τιμές της είναι αρκετά χαμηλές, γεγονός που την κάνει να ξεχωρίζει σε σχέση με τους ανταγωνιστές. Όμως και σε αυτή την υπηρεσία η τιμή των Droplets παραμένει σταθερή, σε αντίθεση με αυτήν των Spot Instances.

[1] https://cloud.google.com/compute/docs/instances/preemptible

2.2 Η Ιδέα μας

Παρατηρώντας όλα τα παραπάνω και λαμβάνοντας υπόψιν τη ραγδαία άνοδο της υπηρεσίας παγκοσμίως θεωρήσαμε πως μέσα από τη συγκεκριμένη εργασία θα μπορούσαμε να αξιοποιήσουμε την υπηρεσία εκμεταλλευόμενοι κάποια χαρακτηριστικά της.

Πιο συγκεκριμένα, στοχεύσαμε στην ανάπτυξη μιας μεθόδου η οποία απευθύνεται σε χρήστες που χρησιμοποιούν την υπηρεσία των Spot Instances και επιθυμούν να εκτελέσουν κάποιες εργασίες εντός ενός συγκεκριμένου χρονικού διαστήματος με το ελάχιστο δυνατό κόστος. Η ιδέα αυτή, αν και ελκυστική, παρουσιάζει δύο βασικές δυσκολίες:

- Ο χρήστης δεν μπορεί να είναι πάντα βέβαιος για το μέγεθος του φόρτου εργασίας που έχει να εκτελέσει. Αν για παράδειγμα χρησιμοποιεί την υπηρεσία των Spot Instances για την διεκπεραίωση requests σε κάποιο backend, τότε δεν μπορεί να ξέρει εκ των προτέρων πόσα τέτοια requests θα φτάσουν.
- Ο χρήστης δεν μπορεί να ξέρει εκ των προτέρων την Spot Price. Η γνώση αυτή είναι πολύ σημαντική, καθώς αν ήξερε πως θα κινηθεί η τιμή του Spot Instance, θα μπορούσε να λαμβάνει καλύτερα τις αποφάσεις για το πότε τον συμφέρει να αγοράσει ή όχι Spot Instances, λαμβάνοντας υπόψιν βέβαια και την προθεσμία για την περάτωση των εργασιών του.

Καταλαβαίνει λοιπόν κανείς την σημασία ενός μηχανισμού που θα παρέχει έγκυρες και αξιόπιστες προβλέψεις για τις δύο αυτές άγνωστες παραμέτρους: το φόρτο εργασίας και το Spot Price. Τη λύση και στα δύο αυτά προβλήματα έρχονται να δώσουν τα Νευρωνικά Δίκτυα, τα οποία θα αναλύσουμε με πιο μεγάλη λεπτομέρεια στο κεφάλαιο 4.

Το ερώτημα που προκύπτει τώρα είναι το εξής: Έχοντας μια πρόβλεψη για το μέγεθος του φόρτου εργασίας και του Spot Price, πώς μπορούμε να αξιοποιήσουμε την πληροφορία αυτή για να αποφασίσουμε αν τη συγκεκριμένη χρονική στιγμή μας συμφέρει να αγοράσουμε Spot Instances; Με άλλα λόγια, πώς θα συνδυάσουμε αυτές τις δύο προβλέψεις για να λαμβάνουμε καλύτερες αποφάσεις σχετικά με την αγορά Spot Instances; Για να απαντήσουμε στο ερώτημα αυτό, θα χρησιμοποιήσουμε δύο γνωστές αλγοριθμικές τεχνικές για την επίλυση προβλημάτων βελτιστοποίησης: τους Άπληστους Αλγορίθμους και τον Δυναμικό Προγραμματισμό. Περισσότερες λεπτομέρειες για τις τεχνικές αυτές θα δοθούν στο κεφάλαιο 3.

Κεφάλαιο 3

Αλγοριθμικές Τεχνικές

3.1 Άπληστοι Αλγόριθμοι

Οι αλγόριθμοι για την επίλυση προβλημάτων βελτιστοποίησης κατά κανόνα εκτελούν μια ακολουθία βημάτων καθένα από τα οποία χαρακτηρίζεται από ένα σύνολο επιλογών. Ένας ***άπληστος αλγόριθμος*** κάνει πάντοτε την επιλογή που φαίνεται καλύτερη τη δεδομένη στιγμή. Με άλλα λόγια, κάνει μια τοπικά βέλτιστη επιλογή με την ελπίδα ότι η επιλογή αυτή θα οδηγήσει σε μια καθολικά βέλτιστη λύση. Στο κεφάλαιο αυτό, θα μελετήσουμε ένα πρόβλημα βελτιστοποίησης για το οποίο μπορεί να βρεθεί βέλτιστη λύση με άπληστους αλγορίθμους.

Οι άπληστοι αλγόριθμοι είναι σε θέση να προσδιορίσουν βέλτιστες λύσεις σε πολλά προβλήματα, όχι όμως σε όλα. Θα εξετάσουμε ένα απλό αλλά μη τετριμμένο πρόβλημα [3], το πρόβλημα της επιλογής δραστηριοτήτων, το οποίο μπορεί να λυθεί με βέλτιστο τρόπο με έναν άπληστο αλγόριθμο. Θα καταστρώσουμε τον αλγόριθμο αυτό δείχνοντας ότι μπορούμε να καταλήξουμε σε μια βέλτιστη λύση κάνοντας πάντοτε άπληστες επιλογές. Επίσης θα περιγράψουμε συνοπτικά τα βασικά στοιχεία της άπληστης μεθοδολογίας, αναπτύσσοντας μια άμεση προσέγγιση της απόδειξης της ορθότητας άπληστων αλγορίθμων.

3.1.1 Το πρόβλημα της επιλογής δραστηριοτήτων

Το παράδειγμα που θα μελετήσουμε αφορά το πρόβλημα του χρονοπρογραμματισμού διαφόρων ανταγωνιστικών δραστηριοτήτων οι οποίες απαιτούν αποκλειστική χρήση ενός κοινού πόρου, όπου το ζητούμενο είναι η εύρεση ενός μέγιστου συνόλου αμοιβαία συμβατών δραστηριοτήτων. Έστω ότι μας δίνεται κάποιο σύνολο

$$\mathcal{S} = \Big\{\alpha_1, \alpha_2, \cdots, \alpha_n\Big\}$$

από n προτεινόμενες ***δραστηριότητες*** που απαιτούν τη χρήση ενός, όπως π.χ. μιας αίθουσας διαλέξεων, ο οποίος μπορεί να χρησιμοποιείται μόνο από μία δραστηριότητα κάθε φορά. Η κάθε δραστηριότητα α_i έχει κάποιον ***χρόνο έναρξης*** s_i και κάποιον ***χρόνο λήξης*** f_i, όπου

$$0 \leq s_i < f_i < \infty$$

Θα πρέπει να σημειωθεί ότι ο όρος *χρόνος λήξης* αναφέρεται στη *χρονική στιγμή της λήξης* μιας δραστηριότητας και όχι στη διάρκειά της. Επομένως, λέγοντας φερ' ειπείν ότι κάποια δραστηριότητα έχει μικρότερο χρόνο λήξης από κάποια άλλη εννοούμε ότι η πρώτη λήγει σε προγενέστερη χρονική στιγμή από τη δεύτερη, και όχι ότι έχει απαραιτήτως μικρότερη διάρκεια.

Εφόσον επιλεγεί, η δραστηριότητα α_i πραγματοποιείται στη διάρκεια του ημιανοικτού χρονικού διαστήματος $[s_i, f_i)$. Οι δραστηριότητες α_i και α_j χαρακτηρίζονται ***συμβατές*** εαν τα διαστήματα $[s_i, f_i)$ και $[s_j, f_j)$ δεν επικαλύπτονται. Δηλαδή, οι α_i και α_j είναι συμβατές εαν $s_i \geq f_j$ ή $s_j \geq f_i$. Το ***πρόβλημα της επιλογής δραστηριοτήτων*** έγκειται στην εύρεση ενός μέγιστου συνόλου αμοιβαία συμβατών δραστηριοτήτων. Υποθέτουμε ότι οι δραστηριότητες είναι διατεταγμένες κατά μονότονα αύξουσα σειρά ως προς τον χρόνο λήξης, δηλαδή

$$f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_{n-1} \leq f_n$$

Το πλεονέκτημα που προσφέρει αυτή η παραδοχή θα φανεί στη συνέχεια. Για παράδειγμα, έστω ότι έχουμε το ακόλουθο σύνολο δραστηριοτήτων $\mathcal{S}$

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	14	16

Στο συγκεκριμένο παράδειγμα, ένα υποσύνολο αμοιβαία συμβατών δραστηριοτήτων είναι το $\{\alpha_3, \alpha_9, \alpha_{11}\}$. Ωστόσο, το υποσύνολο αυτό δεν είναι μέγιστο, αφού το $\{\alpha_1, \alpha_4, \alpha_8, \alpha_{11}\}$ είναι μεγαλύτερο, διότι περιέχει πιο πολλές δραστηριότητες. Στην πραγματικότητα, το $\{\alpha_1, \alpha_4, \alpha_8, \alpha_{11}\}$ είναι ***ένα*** μέγιστο υποσύνολο αμοιβαία συμβατών δραστηριοτήτων. Ένα άλλο τέτοιο υποσύνολο είναι το $\{\alpha_2, \alpha_4, \alpha_9, \alpha_{11}\}$.

3.1.2 Η βέλτιστη υποδομή του προβλήματος επιλογής δραστηριοτήτων

Λέμε πως ένα πρόβλημα βελτιστοποίησης παρουσιάζει **βέλτιστη υποδομή** όταν οι βέλτιστες λύσεις σε ένα υποπρόβλημα εμπεριέχουν βέλτιστες λύσεις σε συναφή υποπροβλήματα, τα οποία μπορούμε να λύσουμε ανεξάρτητα.

Μπορούμε να επιβεβαιώσουμε εύκολα ότι το πρόβλημα της επιλογής δραστηριοτήτων παρουσιάζει βέλτιστη υποδομή. Έστω $\mathcal{S}_{ij}$ το σύνολο των δραστηριοτήτων που ξεκινούν μετά τη λήξη της α_i και λήγουν πριν την έναρξη της α_j. Ας υποθέσουμε ότι θέλουμε να βρούμε ένα μέγιστο σύνολο από αμοιβαία συμβατές δραστηριότητες στο $\mathcal{S}_{ij}$, και ότι επιπλέον ένα τέτοιο σύνολο είναι το $\mathcal{A}_{ij}$, το οποίο περιλαμβάνει κάποια δραστηριότητα α_k. Όταν συμπεριλαμβάνεται σε μια βέλτιστη λύση η α_k, μας απομένουν δύο υποπροβλήματα: να βρούμε αμοιβαία συμβατές δραστηριότητες στο σύνολο $\mathcal{S}_{ik}$ (δραστηριότητες που ξεκινούν μετά τη λήξη της α_i και λήγουν πριν από την έναρξη της α_k) και να βρούμε αμοιβαία συμβατές δραστηριότητες στο σύνολο $\mathcal{S}_{kj}$ (δραστηριότητες που ξεκινούν μετά τη λήξη της α_k και λήγουν πριν από την έναρξη της α_j). Έστω $\mathcal{A}_{ik} = \mathcal{A}_{ij} \cap \mathcal{S}_{ik}$ και $\mathcal{A}_{kj} = \mathcal{A}_{ij} \cap \mathcal{S}_{kj}$, οπότε το $\mathcal{A}_{ik}$ περιέχει τις δραστηριότητες του $\mathcal{A}_{ij}$ που λήγουν πριν από την έναρξη της α_k και το $\mathcal{A}_{kj}$ περιέχει τις δραστηριότητες του $\mathcal{A}_{ij}$ που ξεκινούν μετά τη λήξη της α_k. Επομένως, έχουμε

$$\mathcal{A}_{ij} = \mathcal{A}_{ik} \cup \left\{\alpha_k\right\} \cup \mathcal{A}_{kj},$$

και άρα το πλήθος των δραστηριοτήτων στο μέγιστο σύνολο $\mathcal{A}_{ij}$ αμοιβαία συμβατών δραστηριοτήτων του $\mathcal{S}_{ij}$ θα είναι

$$\left|\mathcal{A}_{ij}\right| = \left|\mathcal{A}_{ik}\right| + \left|\mathcal{A}_{kj}\right| + 1$$

Όπως προκύπτει με το σύνηθες σκεπτικό της αποκοπής και επικόλλησης, η βέλτιστη λύση $\mathcal{A}_{ij}$ θα πρέπει να περιλαμβάνει επίσης βέλτιστες λύσεις στα δύο υποπροβλήματα για $\mathcal{S}_{ik}$ και $\mathcal{S}_{kj}$. Αν μπορούσαμε να βρούμε ένα σύνολο $\mathcal{A}'_{kj}$ αμοιβαία συμβατών δραστηριοτήτων του $\mathcal{S}_{kj}$ όπου

$$\left|\mathcal{A}'_{kj}\right| > \left|\mathcal{A}_{kj}\right|,$$

θα μπορούσαμε να χρησιμοποιήσουμε το $\mathcal{A}'_{kj}$ αντί του $\mathcal{A}_{kj}$ σε μια λύση του υποπροβλήματος για το $\mathcal{S}_{ij}$. Θα κατασκευάζαμε ένα σύνολο από

$$\left|\mathcal{A}_{ik}\right| + \left|\mathcal{A}'_{kj}\right| + 1 > \left|\mathcal{A}_{ik}\right| + \left|\mathcal{A}_{kj}\right| + 1 = \left|\mathcal{A}_{ij}\right|$$

αμοιβαία συμβατές δραστηριότητες, γεγονός που αντιφάσκει με την παραδοχή ότι το $\mathcal{A}_{ij}$ είναι βέλτιστη λύση. Συμμετρικό σκεπτικό ισχύει και για τις δραστηριότητες του $\mathcal{S}_{ik}$.

3.1.3 Η άπληστη επιλογή

Τι σημαίνει *άπληστη επιλογή* για το πρόβλημα της επιλογής δραστηριοτήτων; Η διαίσθησή μας υπαγορεύει ότι θα έπρεπε να επιλέξουμε μια δραστηριότητα που αφήνει τον διεκδικούμενο πόρο διαθέσιμο για όσο το δυνατόν περισσότερες από τις υπόλοιπες δραστηριότητες. Από τις δραστηριότητες που τελικά θα επιλέξουμε, όμως, κάποια θα είναι αναγκαστικά η πρώτη που θα ολοκληρωθεί. Επομένως, η διαίσθηση μας υπαγορεύει να επιλέξουμε τη δραστηριότητα του $\mathcal{S}$ με τον μικρότερο χρόνο λήξης, διότι με αυτόν τον τρόπο ο πόρος θα έμενε διαθέσιμος για όσο το δυνατόν περισσότερες από τις επόμενες δραστηριότητες. Αν υπάρχουν στο $\mathcal{S}$ περισσότερες από μία δραστηριότητες με τον μικρότερο χρόνο λήξης, τότε μπορούμε να επιλέξουμε οποιαδήποτε τέτοια δραστηριότητα. Με άλλα λόγια, με δεδομένο ότι οι δραστηριότητες είναι ταξινομημένες κατά μονότονα αύξουσα σειρά ως προς τον χρόνο λήξης τους, η άπληστη επιλογή είναι η δραστηριότητα α_1.

Εαν κάνουμε την άπληστη επιλογή, μας απομένει να λύσουμε μόνο ένα υποπρόβλημα: την εύρεση δραστηριοτήτων που ξεκινούν μετά τη λήξη της α_1. Γιατί δεν χρειάζεται να εξετάσουμε δραστηριότητες που λήγουν πριν την έναρξη της α_1; Δεδομένου ότι $s_1 < f_1$, και ότι ο f_1 είναι ο μικρότερος χρόνος λήξης για οποιαδήποτε δραστηριότητα, καμία δραστηριότητα δεν μπορεί να έχει χρόνο λήξης μικρότερο ή ίσο του s_1. Συνεπώς, όλες οι δραστηριότητες που είναι συμβατές με την α_1 θα πρέπει να ξεκινούν μετά τη λήξη της α_1.

Επιπλέον, έχουμε ήδη δείξει ότι το πρόβλημα της επιλογής δραστηριοτήτων εμφανίζει βέλτιστη υποδομή. Έστω

$$\mathcal{S}_k = \left\{\alpha_i \in \mathcal{S} \mid s_i \geq f_k\right\}$$

το σύνολο των δραστηριοτήτων που ξεκινούν μετά τη λήξη της δραστηριότητας α_k. Εαν κάνουμε την άπληστη επιλογή της α_1, τότε απομένει ως μοναδικό υποπρόβλημα για επίλυση το $\mathcal{S}_1$. Η βέλτιστη υποδομή σημαίνει ότι αν η α_1 συμπεριλαμβάνεται στη βέλτιστη λύση, τότε μια βέλτιστη λύση στο αρχικό πρόβλημα αποτελείται από τη δραστηριότητα α_1 και όλες τις δραστηριότητες που συμπεριλαμβάνονται σε μια βέλτιστη λύση στο υποπρόβλημα $\mathcal{S}_1$.

Παραμένει όμως ένα μεγάλο ερώτημα: είναι σωστή η διαίσθησή μας; Είναι όντως η άπληστη επιλογή -όπου επιλέγεται η δραστηριότητα που τελειώνει πρώτη- πάντοτε μέρος κάποιας βέλτιστης λύσης; Όπως δείχνει το παρακάτω θεώρημα, η απάντηση είναι καταφατική.

Θεώρημα 1

Για οποιοδήποτε μη κενό υποπρόβλημα $\mathcal{S}_k$, αν α_m είναι μια δραστηριότητα του $\mathcal{S}_k$ με τον μικρότερο χρόνο λήξης, τότε η α_m περιλαμβάνεται σε κάποιο μέγιστο υποσύνολο αμοιβαία συμβατών δραστηριοτήτων του $\mathcal{S}_k$.

Απόδειξη: Έστω $\mathcal{A}_k$ ένα μέγιστο υποσύνολο αμοιβαία συμβατών δραστηριοτήτων του $\mathcal{S}_k$, και α_j η δραστηριότητα του $\mathcal{A}_k$ με τον μικρότερο χρόνο λήξης. Εαν $\alpha_j = \alpha_m$, η απόδειξη έχει ολοκληρωθεί, αφού έχουμε δείξει ότι η α_m περιλαμβάνεται σε κάποιο μέγιστο υποσύνολο αμοιβαία συμβατών δραστηριοτήτων του $\mathcal{S}_k$. Εαν $\alpha_j \neq \alpha_m$, έστω το σύνολο

$$\mathcal{A}'_k = \mathcal{A}_k - \left\{\alpha_j\right\} \cup \left\{\alpha_m\right\}$$

που είναι το $\mathcal{A}_k$ αλλά με το α_m στη θέση του α_j. Οι δραστηριότητες του $\mathcal{A}'_k$ είναι ξένες. Αυτό έπεται από το γεγονός ότι οι δραστηριότητες του $\mathcal{A}_k$ είναι ξένες, η α_j είναι η δραστηριότητα του $\mathcal{A}_k$ που λήγει πρώτη, και $f_m \leq f_j$. Δεδομένου ότι

$$\left|\mathcal{A}'_k\right| = \left|\mathcal{A}_k\right|$$

συμπεραίνουμε ότι το $\mathcal{A}'_k$ είναι ένα μέγιστο υποσύνολο αμοιβαία συμβατών δραστηριοτήτων του $\mathcal{S}_k$, και περιλαμβάνει την α_m. ■

3.1.4 Ένας επαναληπτικός άπληστος αλγόριθμος

Παρακάτω παρουσιάζουμε έναν επαναληπτικό αλγόριθμο για την επίλυση του προβλήματος της επιλογής δραστηριοτήτων:

Αλγόριθμος 1 Επαναληπτικός άπληστος αλγόριθμος επιλογής δραστηριοτήτων

```
1: function GREEDY-SELECTION(s, f)
2:     n = s.length
3:     A = {α_1}
4:     k = 1
5:     for m = 2 to n do
6:         if s[m] ≥ f[k] then
7:             A = A ∪ {α_m}
8:             k = m
9:     return A
```

Η διαδικασία λειτουργεί ως εξής. Η μεταβλητή k εμπεριέχει τον αύξοντα αριθμό της πιο πρόσφατης προσθήκης στο σύνολο $\mathcal{A}$. Δεδομένου ότι οι δραστηριότητες εξετάζονται μονότονα κατά αύξοντα χρόνο λήξης, ο χρόνος f_k είναι ανα πάσα στιγμή ο μέγιστος χρόνος λήξης όλων των δραστηριοτήτων του συνόλου $\mathcal{A}$. Δηλαδή

$$f_k = \max\left\{ f_i \;\middle|\; a_i \in \mathcal{A} \right\} \tag{3.1}$$

Στις γραμμές 3-4 επιλέγεται η δραστηριότητα α_1, ορίζεται ως αρχικό περιεχόμενο του $\mathcal{A}$ μόνο αυτή η δραστηριότητα, και αποδίδεται στον μετρητή k ως αρχική τιμή ο αύξων αριθμός αυτής της δραστηριότητας. Ο βρόχος **for** στις γραμμές 5-8 εντοπίζει τη δραστηριότητα με τον μικρότερο χρόνο λήξης μεταξύ των δραστηριοτήτων του συνόλου $\mathcal{S}_k$. Ο βρόχος εξετάζει την κάθε δραστηριότητα α_m, και εφόσον είναι συμβατή με όλες τις ήδη επιλεγμένες δραστηριότητες την προσθέτει στο σύνολο $\mathcal{A}$. Η προστιθέμενη δραστηριότητα είναι εκείνη με τον μικρότερο χρόνο λήξης μεταξύ των δραστηριοτήτων του $\mathcal{S}_k$. Για να διαπιστώσουμε αν η δραστηριότητα α_m είναι συμβατή με όλες όσες περιέχονται τη δεδομένη στιγμή στο σύνολο $\mathcal{A}$, αρκεί, σύμφωνα με την εξίσωση (3.1), να ελέγξουμε (γραμμή 6) αν ο χρόνος έναρξης της s_m είναι μεγαλύτερος ή ίσος του χρόνου λήξης f_k της τελευταίας δραστηριότητας που προστέθηκε στο $\mathcal{A}$. Εαν η δραστηριότητα α_m είναι συμβατή, τότε προστίθεται στο $\mathcal{A}$ και αποδίδεται στον μετρητή k η τιμή m (γραμμές 7-8).

Η παραπάνω διαδικασία επεξεργάζεται n δραστηριότητες σε χρόνο $\Theta(n)$, υπό την προϋπόθεση ότι οι δραστηριότητες αυτές είναι ήδη διατεταγμένες κατά αύξουσα σειρά ως προς τον χρόνο λήξης. Αν η προϋπόθεση αυτή δεν ισχύει, τότε θα πρέπει να ξοδέψουμε $\Theta(n \log n)$ χρόνο για την ταξινόμηση των δραστηριοτήτων σε αύξουσα σειρά ως προς τον χρόνο λήξης.

3.2 Δυναμικός Προγραμματισμός

Ο δυναμικός προγραμματισμός [3] είναι μια μέθοδος επίλυσης προβλημάτων μέσω του συνδυασμού των λύσεων κάποιων υποπροβλημάτων τους, όπως η μέθοδος διαίρει και βασίλευε. Θα πρέπει να τονίσουμε πως ο όρος *προγραμματισμός* αναφέρεται σε μια μέθοδο τήρησης στοιχείων μέσω πινάκων, και όχι στη σύνταξη κώδικα.

Οι αλγόριθμοι τύπου διαίρει και βασίλευε αναλύουν το πρόβλημα σε ξένα υποπροβλήματα, επιλύουν τα υποπροβλήματα αυτά αναδρομικά, και στη συνέχεια συνδυάζουν τις λύσεις ώστε να συνθέσουν μια λύση του αρχικού προβλήματος. Ο δυναμικός προγραμματισμός, από την άλλη πλευρά, εφαρμόζεται όταν τα διάφορα υποπροβλήματα επικαλύπτονται, δηλαδή όταν έχουν με τη σειρά τους (κάποια) κοινά υπο-υποπροβλήματα. Στην περίπτωση αυτή, ένας αλγόριθμος τύπου διαίρει και βασίλευε θα εκτελούσε περιττές εργασίες, καθώς θα επέλυε κατ' επανάληψη τα ίδια υπο-υποπροβλήματα. Ένας αλγόριθμος δυναμικού προγραμματισμού επιλύει το κάθε υπο-υποπρόβλημα μόνο μια φορά, και στη συνέχεια αποθηκεύει τη λύση σε έναν πίνακα, αποφεύγοντας με τον τρόπο αυτό τον εκ νέου υπολογισμό της απάντησης κάθε φορά που επιλύει το κάθε υπο-υποπρόβλημα.

Ο δυναμικός προγραμματισμός εφαρμόζεται κατά κανόνα σε ***προβλήματα βελτιστοποίησης***. Τέτοιου είδους προβλήματα είναι δυνατόν να έχουν πολλές διαφορετικές λύσεις. Η κάθε λύση έχει μια συγκεκριμένη τιμή, και το ζητούμενο είναι να βρεθεί η λύση με τη βέλτιστη (την ελάχιστη ή τη μέγιστη) τιμή. Μια λύση που εξασφαλίζει τη βέλτιστη τιμή χαρακτηρίζεται ως *μια* βέλτιστη, σε αντιδιαστολή με *τη* βέλτιστη λύση, δεδομένου ότι μπορεί να υπάρχουν διάφορες λύσεις που επιτυγχάνουν τη βέλτιστη τιμή.

Για να αναπτύξουμε έναν αλγόριθμο δυναμικού προγραμματισμού, ακολουθούμε τα παρακάτω τέσσερα βήματα:

1. Χαρακτηρίζουμε τη δομή μιας βέλτιστης λύσης.
2. Ορίζουμε αναδρομικά την τιμή μιας βέλτιστης λύσης.
3. Υπολογίζουμε την τιμή μιας βέλτιστης λύσης εργαζόμενοι κατά κανόνα *αναβιβαστικά* (*bottom-up* δηλαδή *από κάτω προς τα πάνω*).
4. Κατασκευάζουμε μια βέλτιστη λύση από τα δεδομένα που έχουμε υπολογίσει

Τα βήματα 1-3 αποτελούν τα βασικά στάδια της επίλυσης ενός προβλήματος με τη μέθοδο του δυναμικού προγραμματισμού. Αν χρειαζόμαστε μόνο την τιμή μιας βέλτιστης λύσης, και όχι την ίδια τη λύση, μπορούμε να παραλείψουμε το βήμα 4. Σε περίπτωση που απαιτείται και το βήμα 4, μερικές φορές τηρούμε επιπλέον πληροφορίες κατά τη διάρκεια του βήματος 3, ώστε να μπορούμε να κατασκευάσουμε εύκολα μια βέλτιστη λύση.

Στις ενότητες που ακολουθούν, θα παρουσιάσουμε την επίλυση ενός προβλήματος βελτιστοποίησης με τη μέθοδο του δυναμικού προγραμματισμού.

3.2.1 Πολλαπλασιασμός αλληλουχίας πινάκων

Το παράδειγμα δυναμικού προγραμματισμού που θα εξετάσουμε αφορά έναν αλγόριθμο που επιλύει το πρόβλημα του πολλαπλασιασμού μιας ακολουθίας πινάκων. Έστω ότι μας δίνεται μια αλληλουχία n πινάκων $\langle A_1, A_2, \cdots, A_n \rangle$ και μας ζητείται να υπολογίσουμε το γινόμενο

$$A_1 A_2 \cdots A_n \tag{3.2}$$

Από τη στιγμή που θα ομαδοποιήσουμε παρενθετικά τους όρους της έκφρασης (3.2) ώστε να προσδιορίσουμε μονοσήμαντα τη σειρά των πολλαπλασιασμών μεταξύ τους, μπορούμε να υπολογίσουμε την έκφραση αυτή χρησιμοποιώντας τον τυπικό αλγόριθμο πολλαπλασιασμού δύο πινάκων ως υποπρόγραμμα. Ο πολλαπλασιασμός πινάκων είναι προσεταιριστική πράξη, και συνεπώς όλες οι παρενθετικές ομαδοποιήσεις δίνουν το ίδιο γινόμενο. Ένα γινόμενο πινάκων είναι εκφρασμένο σε ***πλήρως παρενθετική μορφή*** εαν είναι είτε ένας και μόνος πίνακας είτε γινόμενο δύο γινομένων πινάκων τα οποία είναι εκφρασμένα σε πλήρως παρενθετική μορφή

και περικλείονται σε παρενθέσεις. Παραδείγματος χάριν, αν η αλληλουχία των πινάκων είναι $\langle A_1, A_2, A_3, A_4\rangle$, το γινόμενο $A_1A_2A_3A_4$ μπορεί να εκφραστεί σε πλήρως παρενθετική μορφή με πέντε διαφορετικούς τρόπους:

$$\Big(A_1\big(A_2(A_3A_4)\big)\Big),\ \Big(A_1\big((A_2A_3)A_4\big)\Big),\ \Big(\big((A_1A_2)A_3\big)A_4\Big),\ \big((A_1A_2)(A_3A_4)\big),\ \Big(\big(A_1(A_2A_3)\big)A_4\Big)$$

Η παρενθετική ομαδοποίηση που χρησιμοποιούμε στον υπολογισμό του γινομένου μιας αλληλουχίας πινάκων μπορεί να έχει σημαντικότατη επίδραση στο κόστος του υπολογισμού. Ας εξετάσουμε κατ' αρχάς το κόστος του πολλαπλασιασμού δύο πινάκων. Ο τυπικός αλγόριθμος του πολλαπλασιασμού αυτού δίνεται από τον ακόλουθο ψευδοκώδικα. Τα πεδία *rows* και *columns* αντιπροσωπεύουν το πλήθος των γραμμών και των στηλών ενός πίνακα, αντίστοιχα.

Αλγόριθμος 2 Πολλαπλασιασμός δύο πινάκων A και B

```
1:  function MATRIX-MULTIPLY(A, B)
2:      if A.columns ≠ B.rows then
3:          throw error "incompatible dimensions"
4:      let C be a new A.rows × B.columns matrix
5:      for i = 1 to A.rows do
6:          for j = 1 to B.columns do
7:              c_ij = 0
8:              for k = 1 to A.columns do
9:                  c_ij = c_ij + a_ik b_kj
10:     return C
```

Για να πολλαπλασιάσουμε δύο πίνακες A και B, θα πρέπει αυτοί να είναι ***συμβατοί***: ο αριθμός των στηλών του A θα πρέπει να ισούται με τον αριθμό των γραμμών του B. Εάν ο A είναι ένας πίνακας $p \times q$ και ο B είναι ένας πίνακας $q \times r$, το γινόμενό τους είναι ένας πίνακας C διαστάσεων $p \times r$. Ο χρόνος υπολογισμού του C καθορίζεται κατά βάση από το πλήθος των βαθμωτών πολλαπλασιασμών στη γραμμή 9, το οποίο ισούται με pqr. Εφεξής, το κόστος τέτοιων υπολογισμών θα εκφράζεται συναρτήσει του πλήθους των βαθμωτών πολλαπλασιασμών.

Στο παρακάτω παράδειγμα αναδεικνύεται η διαφοροποίηση του κόστους που επισύρει η κάθε διαφορετική παρενθετική ομαδοποίηση ενός γινομένου πινάκων.

Παράδειγμα 3.1: Ελάχιστο κόστος γινομένου της αλληλουχίας $\langle A_1, A_2, A_3\rangle$

Έστω ότι μας δίνεται η αλληλουχία τριών πινάκων $\langle A_1, A_2, A_3\rangle$ με διαστάσεις 10×100, 100×5 και 5×50, αντίστοιχα. Θα υπολογίσουμε το κόστος του γινομένου $A_1A_2A_3$ για κάθε μια από τις πλήρως παρενθετικές μορφές του.

- $\big((A_1A_2)A_3\big)$: Εκτελούμε $10 \cdot 100 \cdot 5 = 5000$ βαθμωτούς πολλαπλασιασμούς για να υπολογίσουμε το διαστάσεων 10×5 γινόμενο πινάκων A_1A_2, συν άλλους $10 \cdot 5 \cdot 50 = 2500$ βαθμωτούς πολλαπλασιασμούς για να πολλαπλασιάσουμε αυτόν τον τελευταίο πίνακα με τον A_3. Συνολικά έχουμε 7500 βαθμωτούς πολλαπλασιασμούς.
- $\big(A_1(A_2A_3)\big)$: Εκτελούμε $100 \cdot 5 \cdot 50 = 25000$ βαθμωτούς πολλαπλασιασμούς για να υπολογίσουμε το διαστάσεων 100×50 γινόμενο πινάκων A_2A_3, συν άλλους $10 \cdot 100 \cdot 50 = 50000$ βαθμωτούς πολλαπλασιασμούς για να πολλαπλασιάσουμε αυτόν τον τελευταίο πίνακα με τον A_1. Συνολικά έχουμε 75000 βαθμωτούς πολλαπλασιασμούς.

Επομένως, ο υλοποιισμός του γινομένου σύμφωνα με την $\big((A_1A_2)A_3\big)$ ομαδοποίηση είναι 10 φορές ταχύτερος

Το ***πρόβλημα του πολλαπλασιασμού αλληλουχίας πινάκων*** μπορεί να διατυπωθεί ως εξής: για μια δεδομένη ακολουθία n πινάκων $\langle A_1, A_2, \cdots, A_n \rangle$, όπου για κάθε $i = 1, 2, \cdots, n$ ο πίνακας A_i έχει διαστάσεις $p_{i-1} \times p_i$, εκφράστε το γινόμενο $A_1 A_2 \cdots A_n$ σε πλήρως παρενθετική μορφή τέτοια ώστε να ελαχιστοποιείται το πλήθος των βαθμωτών πολλαπλασιασμών.

Να σημειωθεί ότι το πρόβλημα του πολλαπλασιασμού αλληλουχίας πινάκων δεν αφορά την καθαυτό πράξη του πολλαπλασιασμού των πινάκων. Το ζητούμενο του προβλήματος είναι να προσδιοριστεί μια διάταξη πολλαπλασιασμού των πινάκων η οποία να δίνει το ελάχιστο δυνατό κόστος. Κατά κανόνα, ο χρόνος που αναλώνεται για τον προσδιορισμό αυτής της βέλτιστης διάταξης υπεραντισταθμίζεται από τον χρόνο ο οποίος εξοικονομείται αργότερα κατά τον καθαυτό πολλαπλασιασμό των πινάκων. Στο προηγούμενο παράδειγμα, το χρονικό αυτό όφελος ισούται με τη διαφορά χρόνου μεταξύ των 7500 και των 75000 βαθμωτών πολλαπλασιασμών.

3.2.2 Απαρίθμηση του πλήθους των παρενθετικών ομαδοποιήσεων

Προτού στραφούμε στην επίλυση του προβλήματος του πολλαπλασιασμού αλληλουχίας πινάκων με τη μέθοδο του δυναμικού προγραμματισμού, ας βεβαιωθούμε ότι ο εξαντλητικός έλεγχος όλων των δυνατών ομαδοποιήσεων είναι αδύνατον να δώσει αποδοτικό αλγόριθμο. Ας συμβολίσουμε το πλήθος των εναλλακτικών ομαδοποιήσεων μιας αλληλουχίας n πινάκων με $P(n)$. Για $n = 1$, υπάρχει μόνο ένας πίνακας, και επομένως μόνο ένας τρόπος να εκφραστεί το γινόμενο σε πλήρως παρενθετική μορφή. Για $n \geq 2$, ένα γινόμενο πινάκων εκφρασμένο σε πλήρως παρενθετική μορφή είναι το γινόμενο δύο υπογινομένων πινάκων εκφρασμένων σε πλήρως παρενθετική μορφή, και η *διαχωριστική γραμμή* μεταξύ των δύο υπογινομένων θα βρίσκεται μεταξύ του k-οστου και του $(k+1)$-οστου πίνακα, όπου το k μπορεί να πάρει οποιαδήποτε από τις τιμές $k = 1, 2, \cdots, n-1$.

$$\underbrace{A_1 A_2 A_3 \cdots A_{k-1} A_k}_{P(k)} \Big| \underbrace{A_{k+1} \cdots A_{n-1} A_n}_{P(n-k)}$$

Ως εκ τούτου, παίρνουμε την αναδρομική σχέση

$$P(n) = \begin{cases} 1 & \text{εάν } n = 1, \\ \displaystyle\sum_{k=1}^{n-1} P(k) P(n-k) & \text{εάν } n \geq 2. \end{cases} \tag{3.3}$$

Χρησιμοποιώντας τη μέθοδο της ισχυρής μαθηματικής επαγωγής και υποθέτοντας ότι $P(m) \geq c \cdot 2^m$ για κάθε $m = 2, 3, \cdots, n-1$, προκύπτει:

$$\begin{aligned} P(n) &= \sum_{k=1}^{n-1} P(k) P(n-k) \\ &\geq \sum_{k=1}^{n-1} c \cdot 2^k \cdot c \cdot 2^{n-k} \\ &= \sum_{k=1}^{n-1} c^2 \cdot 2^n \\ &= c^2 (n-1) 2^n \\ &\geq c \cdot 2^n \end{aligned}$$

Άρα, η λύση της (3.3) είναι $\Omega(2^n)$. Επομένως, το πλήθος των ομαδοποιήσεων αυξάνεται εκθετικά συναρτήσει του n, και συνεπώς η μέθοδος της εξαντλητικής αναζήτησης δεν αποτελεί ικανοποιητική στρατηγική για τον προσδιορισμό της βέλτιστης παρενθετικής ομαδοποίησης μιας αλληλουχίας πινάκων.

3.2.3 Εφαρμογή του δυναμικού προγραμματισμού

Για να προσδιορίσουμε τη βέλτιστη παρενθετική ομαδοποίηση μιας αλληλουχίας πινάκων θα χρησιμοποιήσουμε τη μέθοδο του δυναμικού προγραμματισμού. Στο πλαίσιο αυτό, θα ακολουθήσουμε την ακολουθία των τεσσάρων βημάτων που παραθέσαμε στην αρχή του κεφαλαίου:

1. Χαρακτηρίζουμε τη δομή μιας βέλτιστης λύσης.
2. Ορίζουμε αναδρομικά την τιμή μιας βέλτιστης λύσης.
3. Υπολογίζουμε την τιμή μιας βέλτιστης λύσης.
4. Κατασκευάζουμε μια βέλτιστη λύση από τα δεδομένα τα οποία έχουμε υπολογίσει

Θα ακολουθήσουμε αυτά τα βήματα με τη σειρά, δείχνοντας με σαφήνεια πώς εφαρμόζεται το καθένα από αυτά στο πρόβλημα.

Βήμα 1: Η δομή μιας βέλτιστης παρενθετικής ομαδοποίησης

Το πρώτο βήμα του υποδείγματος του δυναμικού προγραμματισμού είναι η εύρεση της βέλτιστης υποδομής και στη συνέχεια η κατασκευή (με τη βοήθεια της βέλτιστης υποδομής) μιας βέλτιστης λύσης στο πρόβλημα από βέλτιστες λύσεις υποπροβλημάτων. Κατ' αρχάς, για να διευκολύνουμε την ανάλυσή μας, θα υιοθετήσουμε τον ακόλουθο συμβολισμό

$$A_{i..j} = A_i A_{i+1} \cdots A_j, \text{ όπου } 1 \leq i \leq j \leq n$$

Με άλλα λόγια, ο $A_{i..j}$ είναι ο πίνακας που προκύπτει από τον πολλαπλασιασμό πινάκων $A_i A_{i+1} \cdots A_j$. Παρατηρούμε ότι αν το πρόβλημα είναι μη τετριμμένο, δηλαδή αν $i < j$, τότε για να ομαδοποιήσουμε το γινόμενο $A_i A_{i+1} \cdots A_j$, θα πρέπει να το διαχωρήσουμε στο *σύνορο* ανάμεσα στον πίνακα A_k και τον A_{k+1}, για κάποιον ακέραιο αριθμό k στο διάστημα $i \leq k < j$. Με άλλα λόγια, για κάποια τιμή του k, υπολογίζουμε αρχικά τους πίνακες $A_{i..k}$ και $A_{k+1..j}$, και στη συνέχεια πολλαπλασιάζουμε τους πίνακες αυτούς για να πάρουμε το τελικό γινόμενο $A_{i..j}$. Επομένως, το κόστος αυτής της ομαδοποίησης ισούται με το κόστος του υπολογισμού του πίνακα $A_{i..k}$, συν το κόστος του υπολογισμού του πίνακα $A_{k+1..j}$, συν το κόστος του πολλαπλασιασμού των δύο αυτών πινάκων.

Η βέλτιστη υποδομή του προβλήματος αυτού έχει ως εξής. Έστω ότι για να ομαδοποιήσουμε με βέλτιστο τρόπο την αλληλουχία $A_i A_{i+1} \cdots A_j$, διαχωρίζουμε το γινόμενο στο σύνορο ανάμεσα στον πίνακα A_k και τον A_{k+1}. Στην περίπτωση αυτή, η ομαδοποίηση της προθηματικής υποαλληλουχίας $A_i A_{i+1} \cdots A_k$ εντός αυτής της βέλτιστης ομαδοποίησης της $A_i A_{i+1} \cdots A_j$ θα πρέπει να είναι μια βέλτιστη ομαδοποίηση της $A_i A_{i+1} \cdots A_k$. Γιατί; Αν υπήρχε πιο οικονομική ομαδοποίηση της $A_i A_{i+1} \cdots A_k$, τότε αντικαθιστώντας αυτήν την ομαδοποίηση στη βέλτιστη ομαδοποίηση της αλληλουχίας $A_i A_{i+1} \cdots A_j$ θα παίρναμε μια άλλη ομαδοποίηση της $A_i A_{i+1} \cdots A_j$ με κόστος μικρότερο από το βέλτιστο, όπερ άτοπο. Το ίδιο ισχύει και για την ομαδοποίηση της υποαλληλουχίας $A_{k+1} A_{k+2} \cdots A_j$ στη βέλτιστη ομαδοποίηση της $A_i A_{i+1} \cdots A_j$: θα πρέπει να είναι μια βέλτιστη ομαδοποίηση της $A_{k+1} A_{k+2} \cdots A_j$.

Εν συνεχεία, θα χρησιμοποιήσουμε τη βέλτιστη υποδομή μας για να δείξουμε ότι μπορούμε να κατασκευάσουμε μια βέλτιστη λύση στο πρόβλημα από βέλτιστες λύσεις υποπροβλημάτων. Όπως έχουμε δει, οποιαδήποτε λύση σε ένα μη τετριμμένο στιγμιότυπο του προβλήματος του πολλαπλασιασμού αλληλουχίας πινάκων προϋποθέτει τον διαχωρισμό του γινομένου, και οποιαδήποτε βέλτιστη λύση εμπεριέχει βέλτιστες λύσεις σε στιγμιότυπα υποπροβλημάτων. Άρα, μπορούμε να κατασκευάσουμε μια βέλτιστη λύση σε ένα στιγμιότυπο του προβλήματος του πολλαπλασιασμού αλληλουχίας πινάκων διαχωρίζοντας το πρόβλημα σε δύο υποπροβλήματα (την εύρεση της βέλτιστης ομαδοποίησης των $A_i A_{i+1} \cdots A_k$ και $A_{k+1} A_{k+2} \cdots A_j$), προσδιορίζοντας βέλτιστες λύσεις στα στιγμιότυπα των υποπροβλημάτων, και εν συνεχεία συνδυάζοντας αυτές τις βέλτιστες λύσεις των υποπροβλημάτων. Θα πρέπει να διασφαλίσουμε ότι όταν αναζητούμε το σωστό σημείο τομής του γινομένου, έχουμε λάβει υπ' όψιν όλα τα δυνατά σημεία, και επομένως είναι βέβαιο ότι έχουμε εξετάσει το βέλτιστο.

Βήμα 2: Μια αναδρομική λύση

Στη συνέχεια, ορίζουμε το κόστος μιας βέλτιστης λύσης αναδρομικά συναρτήσει των βέλτιστων λύσεων υποπροβλημάτων. Για το πρόβλημα του πολλαπλασιασμού αλληλουχίας πινάκων, επιλέγουμε ως υποπροβλήματα τα προβλήματα του προσδιορισμού του ελάχιστου κόστους ομαδοποίησης των γινομένων $A_iA_{i+1}\cdots A_j$ για $1 \leq i \leq j \leq n$. Έστω $m[i,j]$ το ελάχιστο πλήθος βαθμωτών πολλαπλασιασμών που απαιτούνται για τον υπολογισμό του γινομένου $A_{i..j}$. Επομένως, για το πλήρες πρόβλημα, το κόστος του πιο οικονομικού τρόπου υπολογισμού του γινομένου $A_{1..n}$ θα είναι $m[1,n]$.

Μπορούμε να ορίσουμε το $m[i,j]$ αναδρομικά ως εξής. Εάν $i = j$, το πρόβλημα είναι τετριμμένο, καθώς η αλληλουχία αποτελείται από έναν μόνο πίνακα $A_{i..i} = A_i$, και επομένως δεν απαιτείται κανένας βαθμωτός πολλαπλασιασμός για τον υπολογισμό του γινομένου. Κατά συνέπεια,

$$m[i,i] = 0 \text{ για } i = 1, 2, \cdots, n$$

Για να υπολογίσουμε το $m[i,j]$ όταν $i < j$, εκμεταλλευόμαστε τη δομή μιας βέλτιστης λύσης από το βήμα 1. Έστω ότι η βέλτιστη ομαδοποίηση διαχωρίζει το γινόμενο $A_iA_{i+1}\cdots A_j$ στο σύνορο ανάμεσα στον πίνακα A_k και τον A_{k+1}, όπου $i \leq k < j$. Στην περίπτωση αυτή, το $m[i,j]$ ισούται με το ελάχιστο κόστος για τον υπολογισμό των υπογινομένων $A_{i..k}$ και $A_{k+1..j}$, συν το κόστος του πολλαπλασιασμού αυτών των δύο πινάκων. Δεδομένου ότι κάθε πίνακας A_i έχει διαστάσεις $p_{i-1} \times p_i$, ο υπολογισμός του γινομένου πινάκων $A_{i..k}A_{k+1..j}$ απαιτεί $p_{i-1}p_kp_j$ βαθμωτούς πολλαπλασιασμούς. Κατά συνέπεια έχουμε

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$$

Αυτή η αναδρομική εξίσωση προϋποθέτει ότι γνωρίζουμε την τιμή του k, την οποία αγνοούμε. Ωστόσο, υπάρχουν μόνο $j-i$ πιθανές τιμές του k, συγκεκριμένα οι $k = i, i+1, \cdots, j-1$. Εφόσον η βέλτιστη ομαδοποίηση θα πρέπει να αντιστοιχεί σε κάποια από αυτές τις τιμές, αρκεί να τις εξετάσουμε όλες για να εντοπίσουμε την καλύτερη. Επομένως, ο αναδρομικός ορισμός μας για το ελάχιστο κόστος ομαδοποίησης του γινομένου $A_iA_{i+1}\cdots A_j$ παίρνει τη μορφή

$$m[i,j] = \begin{cases} 0 & \text{εάν } i = j, \\ \min\limits_{i \leq k < j} \Big\{ m[i,k] + m[k+1,j] + p_{i-1}p_kp_j \Big\} & \text{εάν } i < j. \end{cases} \tag{3.4}$$

Οι τιμές $m[i,j]$ δίνουν τα κόστη βέλτιστων λύσεων των υποπροβλημάτων, αλλά δεν παρέχουν όλες τις πληροφορίες που χρειαζόμαστε για την κατασκευή μια βέλτιστης λύσης. Για να διευκολύνουμε αυτήν την κατασκευή, ορίζουμε ως $s[i,j]$ μια τιμή k στην οποία διαχωρίζεται το γινόμενο $A_iA_{i+1}\cdots A_j$ σε μια βέλτιστη ομαδοποίηση. Με άλλα λόγια, το στοιχείο $s[i,j]$ ισούται με μια τιμή k τέτοια ώστε $m[i,j] = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$.

Βήμα 3: Υπολογισμός του ελάχιστου κόστους

Στο σημείο αυτό, θα μπορούσαμε να κατασκευάσουμε εύκολα έναν αναδρομικό αλγόριθμο με βάση την αναδρομική εξίσωση (3.4) για να προσδιορίσουμε το ελάχιστο κόστος $m[1,n]$ για τον υπολογισμό του γινομένου $A_1A_2\cdots A_n$. Ωστόσο, ο αναδρομικός αλγόριθμος έχει εκθετικό χρόνο εκτέλεσης, δηλαδή ίδιας τάξεως με τη μέθοδο του εξαντλητικού ελέγχου όλων των δυνατών ομαδοποιήσεων του γινομένου.

Όπως μπορούμε να παρατηρήσουμε, έχουμε σχετικά μικρό πλήθος διαφορετικών υποπροβλημάτων: ένα υποπρόβλημα για κάθε ζεύγος των i και j το οποίο ικανοποιεί τη σχέση $1 \leq i \leq j \leq n$, δηλαδή συνολικά

$$\underbrace{\binom{n}{2}}_{i<j} + \underbrace{n}_{i=j} = \Theta(n^2)$$

Ένας αναδρομικός αλγόριθμος πιθανόν να συναντά το κάθε υποπρόβλημα πολλές φορές, σε διαφορετικούς κλάδους του δένδρου αναδρομής. Η ιδιότητα αυτή της επικάλυψης των υποπροβλημάτων αποτελεί το δεύτερο

κριτήριο εφαρμοσιμότητας του δυναμικού προγραμματισμού (το πρώτο είναι η βέλτιστη υποδομή).

Αντί να υπολογίσουμε ανδρομικά τη λύση της αναδρομικής εξίσωσης (3.4), θα υπολογίσουμε το ελάχιστο κόστος χρησιμοποιώντας μια αναβιβαστική προσέγγιση με χρήση πινάκων.

Η αναβιβαστική μέθοδος με τη χρήση πινάκων υλοποιείται με τη συνάρτηση MATRIX-CHAIN-ORDER, που παρουσιάζεται παρακάτω.

Αλγόριθμος 3 Βέλτιστη ομαδοποίηση γινομένου $A_1 A_2 \cdots A_n$

```
1:  function MATRIX-CHAIN-ORDER(p)
2:      n = p.length
3:      let m[1..n, 1..n] and s[1..n − 1, 2..n] be two new tables
4:      for i = 1 to n do
5:          m[i, i] = 0
6:      for l = 2 to n do
7:          for i = 1 to n − l + 1 do
8:              j = i + l − 1
9:              m[i, j] = ∞
10:             for k = i to j − 1 do
11:                 q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
12:                 if q < m[i, j] then
13:                     m[i, j] = q
14:                     s[i, j] = k
15:     return m, s
```

Η συνάρτηση προϋποθέτει ότι ο πίνακας A_i έχει διαστάσεις $p_{i-1} \times p_i$ για $i = 1, 2, \cdots, n$. Η είσοδός της είναι μια ακολουθία $p = \langle p_0, p_1, \cdots, p_n \rangle$, όπου $p.length = n + 1$. Η συνάρτηση χρησιμοποιεί έναν βοηθητικό πίνακα $m[1..n, 1..n]$ για την αποθήκευση των ποσοτήτων $m[i, j]$ και έναν άλλο βοηθητικό πίνακα $s[1..n-1, 2..n]$ στον οποίο καταγράφεται η τιμή του k για την οποία επιτυγχάνεται το ελάχιστο κόστος στον υπολογισμό του του $m[i, j]$. Ο πίνακας s θα χρησιμοποιηθεί για την κατασκευή μιας βέλτιστης λύσης.

Για να υλοποιήσουμε την αναβιβαστική προσέγγιση, θα πρέπει να προσδιορίσουμε σε ποια στοιχεία του πίνακα ανατρέχουμε όταν υπολογίζουμε το $m[i, j]$. Όπως φαίνεται από την εξίσωση (3.4), το κόστος $m[i, j]$ του υπολογισμού ενός γινομένου μιας αλληλουχίας $j - i + 1$ πινάκων εξαρτάται μόνο από τα κόστη υπολογισμού κάποιων γινομένων αλληλουχιών οι οποίες περιλαμβάνουν λιγότερους από $j - i + 1$ πίνακες. Με άλλα λόγια, για $k = i, i + 1, \cdots, j - 1$, ο πίνακας $A_{i..k}$ είναι ένα γινόμενο $k - i + 1 < j - i + 1$ πινάκων, και ο πίνακας $A_{k+1..j}$ είναι ένα γινόμενο $j - k < j - i + 1$ πινάκων. Επομένως, ο αλγόριθμος θα πρέπει να συμπληρώνει τον πίνακα m με τρόπο που να ανταποκρίνεται στην επίλυση του προβλήματος της ομαδοποίησης για αλληλουχίες αύξοντος μήκους. Για το υποπρόβλημα της βέλτιστης ομαδοποίησης της αλληλουχίας $A_i A_{i+1} \cdots A_j$, θεωρούμε ότι το μέγεθος υποπροβλήματος είναι το μήκος $j - i + 1$ της αλληλουχίας.

Ο αλγόριθμος θέτει αρχικά $m[i, i] = 0$ για $i = 1, 2, \cdots, n$, δηλαδή υπολογίζει τα ελάχιστα κόστη για αλληλουχίες μοναδιαίου μήκους, στις γραμμές 4-5. Στη συνέχεια, κατά τη πρώτη εκτέλεση του βρόχου **for** στις γραμμές 6-14 υπολογίζει με βάση την αναδρομική σχέση (3.4) τα $m[i, i + 1]$ για $i = 1, 2, \cdots, n - 1$, δηλαδή τα ελάχιστα κόστη για αλληλουχίες μήκους $l = 2$. Τη δεύτερη φορά που διατρέχει τον βρόχο, υπολογίζει τα $m[i, i + 2]$ για $i = 1, 2, \cdots, n - 2$, δηλαδή τα ελάχιστα κόστη για αλληλουχίες μήκους $l = 3$, κ.ο.κ. Σε κάθε βήμα, το κόστος $m[i, j]$ που υπολογίζεται στις γραμμές 11-14 εξαρτάται μόνο από τα στοιχεία $m[i, k]$ και $m[k + 1, j]$ του πίνακα, τα οποία έχουν ήδη υπολογιστεί.

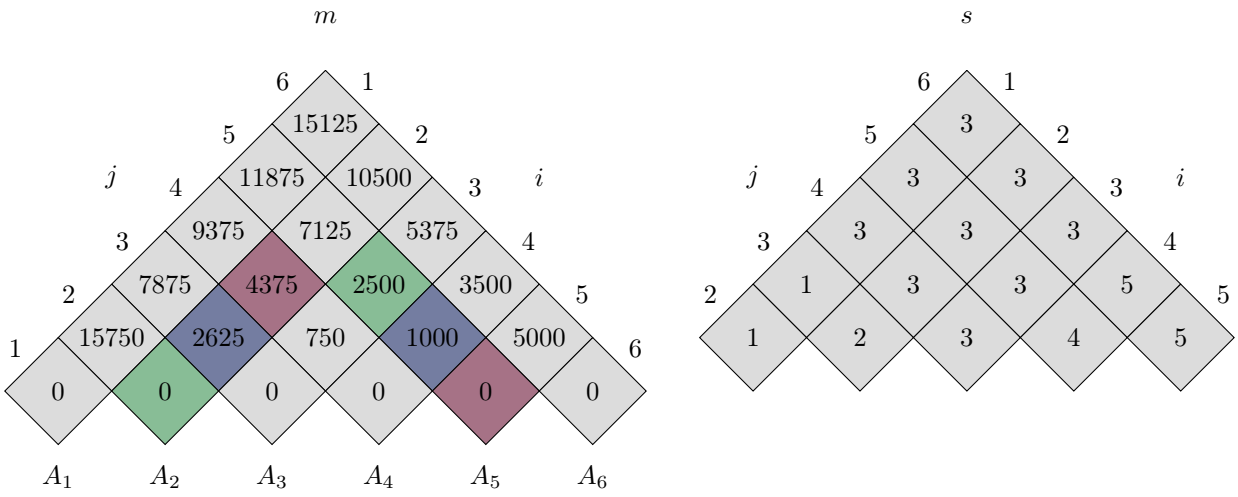
Το σχήμα 3.1 αναπαριστά γραφικά αυτήν την διαδικασία για μια αλληλουχία $n = 6$ πινάκων με τις παρακάτω διαστάσεις

πίνακας	A_1	A_2	A_3	A_4	A_5	A_6
διαστάσεις	30×35	35×15	15×5	5×10	10×20	20×25

Δεδομένου ότι έχουμε ορίσει τα στοιχεία $m[i, j]$ μόνο για $i \leq j$, χρησιμοποιείται μόνο το τμήμα του πίνακα m που κείται στην κύρια διαγώνιο ή επάνω από αυτή. Στο σχήμα ο πίνακας απεικονίζεται στραμμένος έτσι ώστε η κύρια διαγώνιος να εκτείνεται κατά την οριζόντια διεύθυνση. Η αλληλουχία των πινάκων παρατίθεται κατά μήκος της βάσης. Μέσω αυτής της διάταξης, το ελάχιστο κόστος $m[i, j]$ για τον πολλαπλασιασμό μιας αλληλουχίας $A_i A_{i+1} \cdots A_j$ πινάκων μπορεί να βρεθεί στο σημείο τομής των γραμμών που εκτείνονται προς τα επάνω και δεξιά από τον A_i και προς τα πάνω και αριστερά από τον A_j. Τα στοιχεία που περιέχονται σε κάθε οριζόντια γραμμή του πίνακα αντιστοιχούν σε ισομήκεις αλληλουχίες πινάκων. Η συνάρτηση MATRIX-CHAIN-ORDER υπολογίζει τις γραμμές από τη βάση προς την κορυφή και από αριστερά προς τα δεξιά εντός κάθε γραμμής. Για τον υπολογισμό ενός στοιχείου $m[i, j]$ χρησιμοποιούνται τα γινόμενα $p_{i-1} p_k p_j$ για $k = i, i+1, \cdots, j-1$ και όλα τα στοιχεία κάτω αριστερά και κάτω δεξιά από το $m[i, j]$.

Στο σχήμα 3.1, τα στοιχεία που έχουν την ίδια σκίαση συνδυάζονται μεταξύ τους ανά δύο στη γραμμή 11 κατά τον υπολογισμό του στοιχείου

$$m[2,5] = \min \left\{ \begin{array}{l} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13000 \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125 \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11375 \end{array} \right\} = 7125$$



Εικόνα 3.1: Οι πίνακες m και s που υπολογίζονται από τη διαδικασία MATRIX-CHAIN-ORDER για $n = 6$.

Με απλή εποπτεία της δομής των αλληλένθετων βρόχων της συνάρτησης MATRIX-CHAIN-ORDER, συμπεραίνουμε ότι ο χρόνος εκτέλεσης του αλγορίθμου είναι $O(n^3)$. Η δομή αυτή έχει βάθος τριών βρόχων, ενώ ο μετρητής του κάθε βρόχου (l, i και k) λαμβάνει το πολύ $n - 1$ τιμές. Ο αλγόριθμος απαιτεί χώρο $\Theta(n^2)$ για την αποθήκευση των πινάκων m και s. Επομένως, η συνάρτηση MATRIX-CHAIN-ORDER είναι πολύ πιο αποδοτική από την εκθετικού χρόνου μέθοδο της απαρίθμησης όλων των δυνατών ομαδοποιήσεων και του ελέγχου καθεμιάς.

Βήμα 4: Κατασκευή μιας βέλτιστης λύσης

Αν και η συνάρτηση MATRIX-CHAIN-ORDER προσδιορίζει το ελάχιστο πλήθος βαθμωτών πολλαπλασιασμών που απαιτούνται για τον υπολογισμό του γινομένου μιας αλληλουχίας πινάκων, δεν προσδιορίζει άμεσα τον τρόπο πολλαπλασιασμού των πινάκων αυτών. Ο πίνακας $s[1..n-1, 2..n]$ μας δίνει τις πληροφορίες που απαιτούνται γι' αυτό τον σκοπό. Η τιμή του κάθε στοιχείου $s[i, j]$ αντιστοιχεί σε ένα σημείο διαχωρισμού k μεταξύ των πινάκων A_k και A_{k+1} για μια βέλτιστη ομαδοποίηση της αλληλουχίας $A_i A_{i+1} \cdots A_j$. Επομένως, γνωρίζουμε ότι ο τελικός πολλαπλασιασμός πινάκων για τον υπολογισμό του γινομένου $A_{1..n}$ με τον βέλτιστο τρόπο θα είναι $A_{1..s[1,n]} A_{s[1,n]+1..n}$. Οι προηγούμενοι πολλαπλασιασμοί πινάκων μπορούν να υπολογιστούν αναδρομικά, δεδομένου ότι το στοιχείο $s\big[1, s[1, n]\big]$ προσδιορίζει τον τελευταίο πολλαπλασιασμό πινάκων στον υπολογισμό του γινομένου $A_{1..s[1,n]}$, και το $s\big[s[1, n]+1, n\big]$ προσδιορίζει τον τελευταίο πολλαπλασιασμό πινάκων στον τον υπολογισμό του γινομένου $A_{s[1,n]+1..n}$. Η αναδρομική διαδικασία που ακολουθεί εκτυπώνει μια βέλτιστη ομαδοποίηση της αλληλουχίας $\langle A_i, A_{i+1}, \cdots, A_j \rangle$, λαμβάνοντας ως είσοδο τον πίνακα s που υπολογίζεται από τη MATRIX-CHAIN-ORDER και τις τιμές των i και j. Αν η διαδικασία κληθεί αρχικά με είσοδο PRINT-OPTIMAL-PARENS$(s, 1, n)$, θα εκτυπώσει μια βέλτιστη ομαδοποίηση της πλήρους αλληλουχίας $\langle A_1, A_2, \cdots, A_n \rangle$.

Αλγόριθμος 4 Εκτύπωση βέλτιστης ομαδοποίησης αλληλουχίας $\langle A_i, A_{i+1}, \cdots, A_j \rangle$

```
1: procedure PRINT-OPTIMAL-PARENS(s, i, j)
2:     if i = j then
3:         print "A"_i
4:     else
5:         print "("
6:         PRINT-OPTIMAL-PARENS(s, i, s[i, j])
7:         PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
8:         print ")"
```

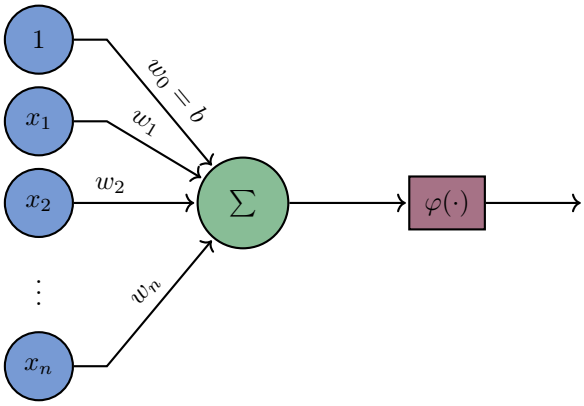
Για το παράδειγμα που αναφέρεται στο σχήμα 3.1, η PRINT-OPTIMAL-PARENS$(s, 1, 6)$ θα εκτυπώσει την ομαδοποίηση $((A_1(A_2A_3))((A_4A_5)A_6))$.

Κεφάλαιο 4

Νευρωνικά Δίκτυα

4.1 Το Perceptron

Το perceptron [6] [9] [10] αποτελεί την απλούστερη δυνατή μορφή ενός νευρωνικού δικτύου και χρησιμοποιείται για την ταξινόμηση προτύπων τα οποία λέγεται πως είναι γραμμικά διαχωρίσιμα. Αποτελείται από ένα μεμονωμένο νευρώνα με προσαρμόσιμα συναπτικά βάρη και προδιάθεση (πόλωση, bias). Το μοντέλο αυτό προτάθηκε το 1958 από τον Rosenblat ως πρώτο μοντέλο μάθησης με τη συνδρομή ενός εκπαιδευτή (επιβλεπόμενη μάθηση). Στην παρακάτω εικόνα δίνεται μια σχηματική απεικόνιση του perceptron του Rosenblat.



Εικόνα 4.1: Το perceptron του Rosenblat

Το παραπάνω μοντέλο διαθέτει τρία βασικά χαρακτηριστικά.

- Ένα σύνολο συνάψεων κάθε μια εκ των οποίων διαθέτει το δικό της βάρος w_i. Ένα σήμα x_j στην είσοδο της σύναψης j που συνδέεται με τον νευρώνα πολλαπλασιάζεται πρώτα με το συναπτικό βάρος w_j.
- Έναν αθροιστή για την άθροιση των σημάτων εισόδου πραγματοποιώντας λειτουργίες γραμμικού συνδυαστή.
- Μία συνάρτηση ενεργοποίησης (activation function) για τον περιορισμό του πλάτους του σήματος εξόδου ενός νευρώνα. Υπάρχουν διάφορα είδη συναρτήσεων ενεργοποίησης τα οποία θα αναλυθούν παρακάτω.

Θα πρέπει να επισημάνουμε την ύπαρξη της εξωτερικά εφαρμοζόμενης πόλωσης b η οποία στο παραπάνω σχήμα περιγράφεται ως είσοδος μεγέθους 1 σταθμισμένη με βάρος w_0.

Έστω $\mathbf{x} = [1, x_1, x_2, \cdots, x_n]^\top$ το διάνυσμα εισόδων και $\mathbf{w} = [b, w_1, w_2, \cdots, w_n]^\top$ το διάνυσμα συναπτικών βαρών. Η έξοδος v του γραμμικού συνδυαστή δίνεται από τη σχέση:

$$v = \sum_{i=0}^{n} w_i x_i = \mathbf{w}^\top \mathbf{x}$$

Η εξίσωση $\mathbf{w}^\top \mathbf{x} = 0$ αποτελεί ένα υπερεπίπεδο του $\mathbb{R}^n$, το οποίο λειτουργεί ως διαχωριστική επιφάνεια απόφασης μεταξύ δύο διαφορετικών κλάσεων εισόδων $\mathcal{C}_1$ και $\mathcal{C}_2$.

Προκειμένου το perceptron να λειτουργήσει σωστά θα πρέπει οι δύο αυτές κλάσεις να είναι γραμμικά διαχωρίσιμες, δηλαδή τα πρότυπα προς ταξινόμηση να είναι επαρκώς διαχωρισμένα μεταξύ τους για να διασφαλιστεί ότι η επιφάνεια απόφασης αποτελείται από ένα υπερεπίπεδο. Δίνοντας ως είσοδο διανύσματα και από τις δύο κλάσεις για την εκπαίδευση του ταξινομητή, η διαδικασία εκπαίδευσης απαιτεί την προσαρμογή του διανύσματος βαρών $\mathbf{w}$ με τέτοιο τρόπο ώστε οι δύο κλάσεις να είναι γραμμικά διαχωρίσιμες. Δηλαδή, να βρεθεί διάνυσμα βαρών $\mathbf{w}$ τέτοιο ώστε:

$$\mathbf{w}^\top \mathbf{x} \geq 0, \ \forall \mathbf{x} \in \mathcal{C}_1$$
$$\mathbf{w}^\top \mathbf{x} < 0, \ \forall \mathbf{x} \in \mathcal{C}_2$$

Στην συγκεκριμένη περίπτωση, η συνάρτηση ενεργοποίησης είναι η συνάρτηση προσήμου, η οποία δίνεται από τον τύπο

$$\varphi(v) = \begin{cases} 1 & \text{εάν } v \geq 0 \\ -1 & \text{εάν } v < 0 \end{cases}$$

Στον παρακάτω πίνακα παρουσιάζονται μερικές χρήσιμες συναρτήσεις ενεργοποίησης.

Όνομα	**Τύπος**	**Ομαλότητα**
Binary Step	$\varphi(x) = \begin{cases} 1 & \text{εάν } v \geq 0 \\ -1 & \text{εάν } v < 0 \end{cases}$	C^{-1}
Rectifier	$\varphi(x) = \max\{x, 0\}$	C^{-1}
Logistic (Sigmoid)	$\varphi(x) = \frac{1}{1+e^{-x}}$	C^{∞}
TanH	$\varphi(x) = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	C^{∞}
ArcTan	$\varphi(x) = \tan^{-1}(x)$	C^{∞}

Πίνακας 4.1: Βασικές συναρτήσεις ενεργοποίησης

Αποδεικνύεται πως το perceptron μετά το πέρας της εκπαίδευσής του μπορεί να ταξινομήσει σωστά πρότυπα από δύο γραμμικά διαχωρίσιμες κλάσεις. Για να συμβεί αυτό θα πρέπει να δοθεί κατάλληλος αριθμός εποχών. Σε κάθε εποχή το perceptron δέχεται όλα τα πρότυπα εισόδου. Αν σε μία εποχή δεν υπάρξει ούτε μία λανθασμένη ταξινόμηση, τότε η διαδικασία εκπαίδευσης έχει ολοκληρωθεί. Η διαδικασία εκπαίδευσης του perceptron παρουσιάζεται παρακάτω.

Επιλέγοντας κατάλληλο αριθμό εποχών και ρυθμό μάθησης η, το perceptron παράγει ένα σωστό σύνορο απόφασης για γραμμικά διαχωρίσιμες κλάσεις. Ωστόσο, μία τόσο απλή κατανομή, με πρότυπα πάνω και κάτω ενός υπερεπιπέδου, δεν παρατηρείται σε πραγματικά προβλήματα. Αντίθετα, χαλαρώνεται η απαίτηση για μηδενικό λάθος και στόχος γίνεται η ελαχιστοποίησή του.

Για την εκμάθηση σύνθετων κατανομών, επεκτείνουμε το μοντέλο του απλού perceptron σε πολλά επίπεδα διατάσσοντας στο καθένα πολυάριθμα perceptrons. Οι νευρώνες που λαμβάνουν το ίδιο διάνυσμα εισόδου βρίσκονται στο ίδιο *επίπεδο*. Κατά συνέπεια, κάθε επίπεδο με n-διάστατο διάνυσμα εισόδου και m εξόδους ορίζει μία απεικόνιση $\varphi : \mathbb{R}^n \to \mathbb{R}^m$. Το διάνυσμα εξόδου $\varphi(\mathbf{x})$ αποτελεί είσοδο στο επόμενο επίπεδο και η διαδικασία επαναλαμβάνεται έως το επίπεδο εξόδου που, συνήθως, χαρακτηρίζεται από μία έξοδο $y \in \mathbb{R}$. Τα ενδιάμεσα επίπεδα, δηλαδή όλα εκτός της εισόδου και της εξόδου, λέγονται *κρυφά*. Στις επόμενες ενότητες θα ακολουθήσει πιο λεπτομερής ανάλυση.

Αλγόριθμος 5: Εκπαίδευση perceptron

Μεταβλητές και παράμετροι:

- $\mathbf{x}(n) = [1, x_1(n), x_2(n), \cdots, x_m(n)]^\top$: διάνυσμα εισόδων $(m+1) \times 1$
- $\mathbf{w}(n) = [b, w_1(n), w_2(n), \cdots, w_m(n)]^\top$: διάνυσμα συναπτικών βαρών $(m+1) \times 1$
- b: πόλωση,
- η: παράμετρος ρυθμού μάθησης, μια θετική σταθερά μικρότερη της μονάδας
- $y(n)$: πραγματική απόκριση
- $d(n)$: επιθυμητή απόκριση
- Όρισε μέγιστο αριθμό εποχών

Τα βήματα που πραγματοποιούνται είναι τα ακόλουθα:

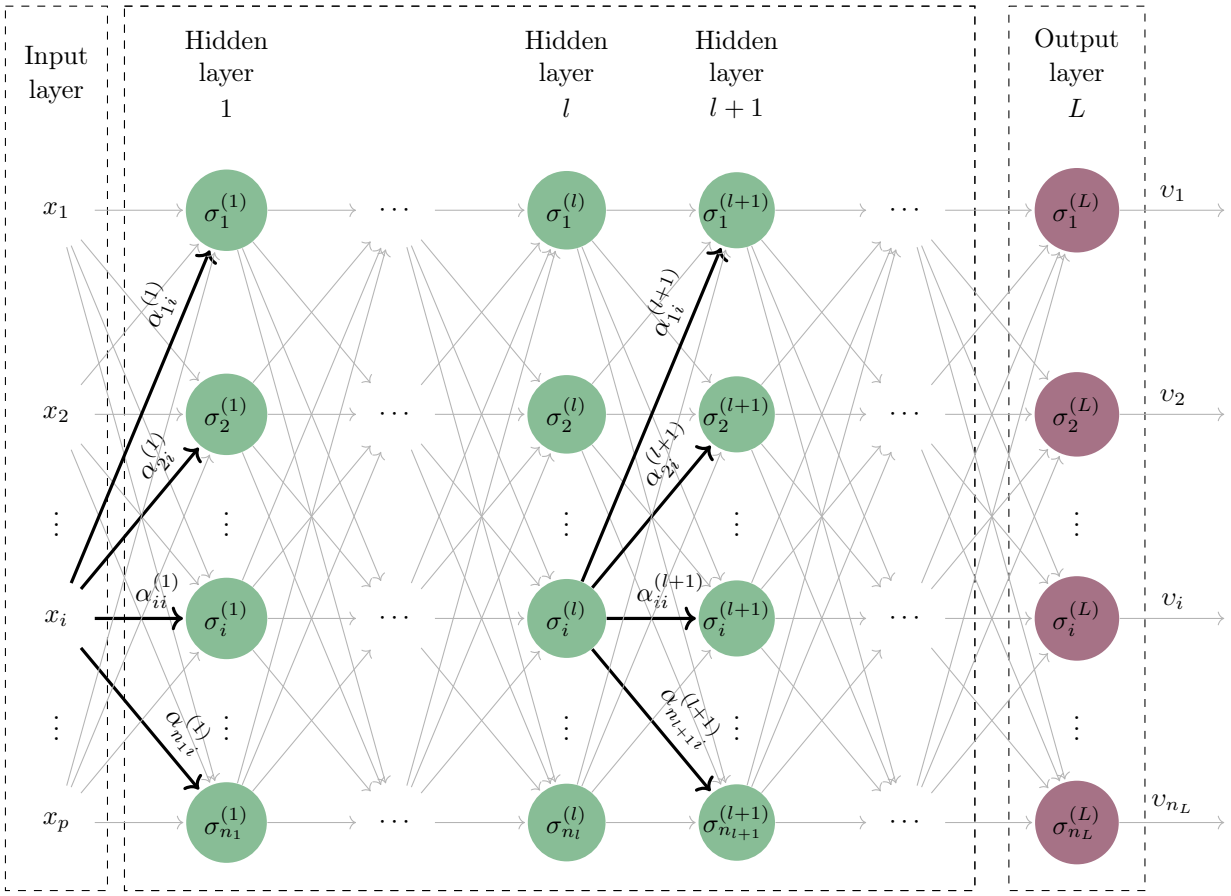
1. Αρχικοποίησε με τυχαίες τιμές το διάνυσμα $\mathbf{w}(n)$. Στη συνέχεια, εκτέλεσε τα ακόλουθα βήματα για χρονικές στιγμές $n = 1, 2, ...$
2. Επανάλαβε έως ότου δεν γίνει καμία αλλαγή βαρών ή έχει συμπληρωθεί ο μέγιστος αριθμός εποχών. Τη χρονική στιγμή n, ενεργοποίησε το perceptron εφαρμόζοντας το διάνυσμα εισόδων $\mathbf{x}(n)$ και την επιθυμητή απόκριση $d(n)$.
3. Υπολόγισε την πραγματική απόκριση του perceptron ως $y(n) = sgn[\mathbf{w}^\top(n)\mathbf{x}(n)]$
4. Ενημέρωσε το διάνυσμα βαρών ως εξής: $\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$, όπου

$$d(n) = \begin{cases} 1 & \text{εάν } \mathbf{x}(n) \in \mathcal{C}_1 \\ -1 & \text{εάν } \mathbf{x}(n) \in \mathcal{C}_2 \end{cases}$$

5. Αν έχει συμπληρωθεί ο μέγιστος αριθμός εποχών επίστρεψε σφάλμα, αλλιώς επίστρεψε τα βάρη.

4.2 Νευρωνικά Δίκτυα Πολλαπλών Στρωμάτων

Στην ενότητα αυτή θα μιλήσουμε για perceptron πολλαπλών στρωμάτων-επιπέδων. Στην εικόνα 4.2 φαίνεται ένα πλήρως συνδεδεμένο νευρωνικό δίκτυο πολλαπλών στρωμάτων. Τα δίκτυα στο οποία θα αναφερθούμε είναι όλα πλήρως συνδεδεμένα, εννοώντας πως ένας νευρώνας σε οποιοδήποτε επίπεδο του δικτύου συνδέεται με όλους τους νευρώνες του προηγούμενου επιπέδου.



Εικόνα 4.2: Feed-forward Νευρωνικό Δίκτυο

Σε ένα perceptron πολλαπλών στρωμάτων-επιπέδων παρατηρούμε δύο είδη σημάτων. Το πρώτο είδος είναι τα λειτουργικά σήματα (function signal) δηλαδή σήματα εισόδου τα οποία διαδίδονται προς τα εμπρός και εμφανίζονται τελικά στην έξοδο του δικτύου. Το σήμα εισόδου εκτελεί μία χρήσιμη λειτουργία στην έξοδο του δικτύου και σε κάθε νευρώνα του ενδιάμεσου δικτύου το σήμα αυτό υπολογίζεται ως συνάρτηση των εισόδων και των σχετιζομένων βαρών που εφαρμόζονται σε αυτόν τον νευρώνα.Η δεύτερη κατηγορία αφορά τα σήματα σφάλματος (error signal). Ένα τέτοιο σήμα προέρχεται από ένα νευρώνα εξόδου του δικτύου και διαδίδεται προς τα πίσω διαμέσου του δικτύου. Αποκαλείται σήμα σφάλματος επειδή ο υπολογισμός του απαιτεί τη χρήση μίας σχετικής συνάρτησης σφάλματος.

Οι κρυφοί νευρώνες απαρτίζουν τα κρυφά επίπεδα του δικτύου. Δεν ανήκουν ούτε στο επίπεδο εισόδου ούτε στο επίπεδο εξόδου. Για το λόγο αυτό ονομάζονται και κρυφοί. Δρουν ως *ανιχνευτές χαρακτηριστικών* και διαδραματίζουν κρίσιμο ρόλο στη λειτουργία του νευρωνικού δικτύου. Κατά την πρόοδο της διαδικασίας μάθησης οι κρυφοί νευρώνες αρχίζουν σταδιακά να ανακαλύπτουν τα εξέχοντα χαρακτηριστικά που χαρακτηρίζουν τα δεδομένα εκπαίδευσης.

4.3 Εκπαίδευση Νευρωνικού Δικτύου Πολλαπλών Στρωμάτων

Έστω $\{\mathbf{x}(n), \mathbf{d}(n)\}_{n=1}^{N}$ η ακολουθία των δειγμάτων εκπαίδευσης. Έστω $y_j(n)$ η έξοδος του νευρώνα j στο επίπεδο εξόδου. Τότε, το σήμα σφάλματος που παράγεται στην έξοδο του νευρώνα j θα είναι:

$$e_j(n) = d_j(n) - y_j(n)$$

όπου d_j το j-οστό στοιχείο του διανύσματος επιθυμητών αποκρίσεων $\mathbf{d}(n)$. Η στιγμιαία ενέργεια σφάλματος του νευρώνα ορίζεται ως:

$$E_j(n) = \frac{1}{2} e_j^2(n)$$

ενώ η συνολική στιγμιαία ενέργεια σφάλματος του δικτύου ως

$$E(n) = \frac{1}{2} \sum_{j \in \mathcal{C}} e_j^2(n)$$

όπου το $\mathcal{C}$ περιλαμβάνει όλους τους νευρώνες στο επίπεδο εξόδου. Τέλος, καθώς η ακολουθία αποτελείται από N δείγματα εκπαίδευσης, η μέση ενέργεια σφάλματος ορίζεται ως:

$$E_{\text{av}} = \frac{1}{N} \sum_{n=1}^{N} E(n) = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in \mathcal{C}} e_j^2(n)$$

Ένας από τους πιο σημαντικούς αλγορίθμους για την εκπαίδευση ενός νευρωνικού δικτύου είναι ο αλγόριθμος **Back Propagation**. Πριν περάσουμε στην περιγραφή του Back Propagation κάνουμε μία σύντομη περιγραφή του αλγορίθμου **Gradient Descent**.

Ο Gradient Descent είναι ένας επαναληπτικός αλγόριθμος βελτιστοποίησης για την εύρεση του ελαχίστου μίας συνάρτησης. Η ιδέα του βασίζεται στην πραγματοποίηση βημάτων προς το αρνητικό της κλίσης σε ένα συγκεκριμένο σημείο της συνάρτησης προς βελτιστοποίηση.

Έστω λοιπόν μία αντικειμενική συνάρτηση $\mathbf{F}(\mathbf{x})$ πολλαπλών μεταβλητών και έστω ένα σημείο $\mathbf{a}$ στο οποίο η συνάρτηση $\mathbf{F}(\mathbf{x})$ ορίζεται και είναι παραγωγίσιμη σε μία γειτονιά του σημείου αυτού. Η αντικειμενική συνάρτηση $\mathbf{F}(\mathbf{x})$ μειώνεται γρηγορότερα αν μετακινηθεί από το σημείο $\mathbf{a}$ σε σημείο αντίθετα της κατεύθυνσης της κλίσης της $\mathbf{F}(\mathbf{x})$ στο σημείο αυτό. Προκύπτει πως για $\gamma \in \mathbb{R}_+$ αρκετά μικρό, το σημείο

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla \mathbf{F}(\mathbf{a}_n)$$

μας οδηγεί στη σχέση $\mathbf{F}(\mathbf{a}_n) \geq \mathbf{F}(\mathbf{a}_{n+1})$. Αφαιρώντας τον όρο $\gamma \nabla \mathbf{F}(\mathbf{a}_n)$ από το $\mathbf{a}$ κινούμαστε προς το ελάχιστο της συνάρτησης. Επομένως, θεωρώντας αρχικό σημείο $\mathbf{x}_0$ τοπικού ελαχίστου, λαμβάνουμε τα σημεία $\mathbf{x}_0, \mathbf{x}_1, \cdots$ από τη σχέση

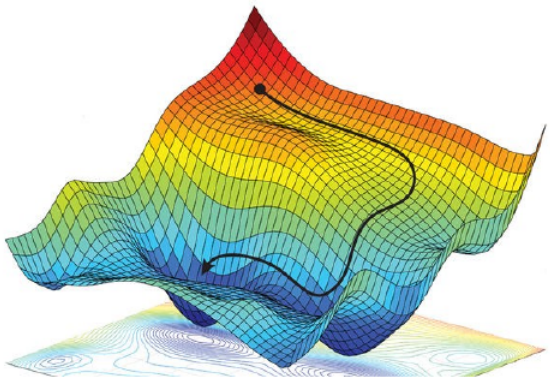
$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla \mathbf{F}(\mathbf{x}_n)$$

για τα οποία ισχύει

$$\mathbf{F}(\mathbf{x}_0) \geq \mathbf{F}(\mathbf{x}_1) \geq \cdots$$

η οποία πιστεύουμε πως θα μας οδηγήσει σε τοπικό ελάχιστο. Η σταθερά γ μπορεί να αλλάζει σε κάθε επανάληψη. Αποδεικνύεται πως αν γνωρίζουμε κάποιες συνθήκες για τη συνάρτηση $\mathbf{F}(\mathbf{x})$ (π.χ. η $\mathbf{F}(\mathbf{x})$ είναι κυρτή και η $\nabla \mathbf{F}(\mathbf{x})$ είναι συνεχής κατά Lipschitz), τότε υπάρχει γ τέτοιο, ώστε η $\mathbf{F}(\mathbf{x})$ να συγκλίνει σίγουρα σε τοπικό ελάχιστο. Αν μάλιστα η $\mathbf{F}(\mathbf{x})$ είναι κυρτή τότε το τοπικό ελάχιστο είναι και ολικό.

Παρόλα αυτά, ο αλγόριθμος Gradient Descent παρουσιάζει κάποιους περιορισμούς. Είναι αργός όταν πλησιάζει το ελάχιστο, ενώ παράλληλα για ορισμένα προβλήματα είναι πιθανό να παρουσιάσει κινήσεις τύπου *ζικ-ζακ* γύρω από το ελάχιστο. Επιπλέον, δεν ορίζεται για μη διαφορίσιμες συναρτήσεις. Για να γίνει άρση κάποιων περιορισμών έχουν δημιουργηθεί διάφορες παραλλαγές του αλγορίθμου που θα αναλυθούν στην επόμενη ενότητα. Παρακάτω παρουσιάζεται σχηματικά η πορεία του αλγορίθμου Gradient Descent σε μία τρισδιάστατη κυρτή συνάρτηση.



Εικόνα 4.3: Πορεία αλγορίθμου Gradient Descent σε τρισδιάστατη κυρτή συνάρτηση

Προχωράμε τώρα στην ανάλυση του αλγορίθμου Back Propagation. Ο αλγόριθμος Back Propagation αποτελείται από δύο περάσματα. Το πρώτο γίνεται με κατεύθυνση προς τα εμπρός και το δεύτερο με κατεύθυνση προς τα πίσω.

Στο πέρασμα με κατεύθυνση προς τα εμπρός, τα συναπτικά βάρη παραμένουν αμετάβλητα σε όλη την έκταση του δικτύου και τα λειτουργικά σήματα του δικτύου υπολογίζονται νευρώνα προς νευρώνα έως ότου φτάσουμε στους κόμβους εξόδου. Έστω τυχαίος νευρώνας j. Το τοπικό πεδίο v_j που παράγεται στην είσοδο της συνάρτησης ενεργοποίησης που σχετίζεται με το νευρώνα j είναι

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$$

όπου m ο συνολικός αριθμός εισόδων που εφαρμόζονται στον νευρώνα j. Το συναπτικό βάθος w_{j0} ισούται με την πόλωση b_j που εφαρμόζεται στον νευρώνα j. Επομένως, το λειτουργικό σήμα $y_j(n)$ που εμφανίζεται στην έξοδο του νευρώνα j κατά την n-οστή επανάληψη είναι

$$y_j(n) = \varphi_j(v_j(n))$$

όπου $\varphi_j(\cdot)$ η συνάρτηση ενεργοποίησης.

Στο πέρασμα με κατεύθυνση προς τα πίσω, ξεκινάμε από το επίπεδο εξόδου στέλνοντας τα σήματα σφάλματος προς τα αριστερά, σε όλα τα επίπεδα του δικτύου, επίπεδο προς επίπεδο, υπολογίζοντας την τοπική κλίση για κάθε νευρώνα μετατρέποντας με τον τρόπο αυτό κατάλληλα τα συναπτικά βάρη του δικτύου. Πιο συγκεκριμένα, εφαρμόζεται μία διόρθωση $\Delta w_{ji}(n)$ στο συναπτικό βάρος $w_{ji}(n)$, η οποία είναι ανάλογη με τη μερική παράγωγο της συνολικής στιγμιαίας ενέργειας σφάλματος του δικτύου ως προς το βάρος $w_{ji}(n)$. Πιο συγκεκριμένα, από τον κανόνα αλυσίδας έχουμε:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

Προκύπτει πως

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n),\ \frac{\partial e_j(n)}{\partial y_j(n)} = -1,\ \frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)) \text{ και } \frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n)$$

Η χρήση των παραπάνω σχέσεων δίνει:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi'_j(v_j(n))y_i(n)$$

Επομένως, η διόρθωση $\Delta w_{ji}(n)$ που εφαρμόζεται στο βάρος $w_{ji}(n)$ ορίζεται από τον *κανόνα Δέλτα*

$$\Delta w_{ji}(n) = -\eta\frac{\partial E(n)}{\partial w_{ji}(n)} = \eta\delta_j(n)y_i(n)$$

όπου η η παράμετρος μάθησης του αλγορίθμου Back Propagation και $\delta_j(n)$ η τοπική κλίση που ορίζεται ως

$$\delta_j(n) = \frac{\partial E(n)}{\partial v_j(n)} = \frac{\partial E(n)}{\partial e_j(n)}\frac{\partial e_j(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n)\varphi'_j(v_j(n))$$

Η παράμετρος η είναι πρακτικά η παράμετρος γ του αλγορίθμου Gradient Descent και η χρήση του αρνητικού προσήμου υποδηλώνει τη χρήση του αλγορίθμου αυτού.

Όταν ο υπό μελέτη νευρώνας j είναι ένας κόμβος εξόδου, τότε τροφοδοτείται με την δική του επιθυμητή απόκριση και το είναι εύκολο να υπολογίσουμε το σήμα σφάλματος. Όταν ο νευρώνας j είναι ένας κρυφός νευρώνας, τότε δεν υπάρχει καθορισμένη επιθυμητή απόκριση για αυτόν. Συνεπώς, το σήμα σφάλματος για έναν κρυφό νευρώνα πρέπει να καθοριστεί αναδρομικά, δουλεύοντας προς τα πίσω. Επαναπροσδιορίζοντας την τοπική κλίση και χρησιμοποιώντας τους τύπους ενέργειας που ορίσαμε παραπάνω προκύπτει πως

$$\delta_j(n) = -\frac{\partial E(n)}{\partial y_j(n)}\varphi'_j(v_j(n)),\ \text{όπου}$$

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k(n)\frac{\partial e_k(n)}{\partial y_j(n)} = \sum_k e_k(n)\frac{\partial e_k(n)}{\partial v_k(n)}\frac{\partial v_k(n)}{\partial y_j(n)}$$

και k κόμβος εξόδου. Όμως, $e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k(v_k(n))$, επομένως

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi'_k(v_k(n))$$

Για το νευρώνα k το τοπικό πεδίο δίνεται από τη σχέση

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n)y_j(n)$$

όπου m ο συνολικός αριθμός εισόδων που εφαρμόζονται στον νευρώνα k. Παραγωγίζοντας ως προς $y_j(n)$ λαμβάνουμε

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n)$$

και χρησιμοποιώντας τις πιο πάνω εξισώσεις καταλήγουμε πως

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_k \delta_k(n)w_{kj}(n)$$

Τέλος λαμβάνουμε τον τύπο οπισθοδιάδοσης για την τοπική κλίση $\delta_j(n)$ ο οποίος περιγράφεται από την

$$\delta_j(n) = \varphi'_j(v_j(n))\sum_k \delta_k(n)w_{kj}(n)$$

όπου ο νευρώνας j είναι κρυφός.

Ο εξωτερικός παράγοντας $\varphi'(v_j(n))$ που χρησιμοποιείται στον υπολογισμό της τοπικής κλίσης $\delta_j(n)$ εξαρτάται αποκλειστικά τη συνάρτηση ενεργοποίησης που σχετίζεται με τον κρυφό νευρώνα j. Ο άλλος παράγοντας για τον υπολογισμό αυτό εξαρτάται από δύο σύνολα όρων. Το πρώτο σύνολο όρων, $\delta_k(n)$, απαιτεί γνώση των σημάτων σφάλματος $e_k(n)$ για όλους τους νευρώνες που βρίσκονται στο αμέσως δεξιότερο επίπεδο του κρυφού νευρώνα j και συνδέονται με τον νευρώνα j. Το δεύτερο σύνολο όρων, $w_{kj}(n)$, αποτελείται από τα συναπτικά βάρη που σχετίζονται με αυτές τις συνδέσεις.

Συνοπτικά λοιπόν έχουμε πως η διόρθωση $\Delta w_{ji}(n)$ που εφαρμόζεται στο συναπτικό βάρος που συνδέει το νευρώνα j με το νευρώνα i ορίζεται από τον παρακάτω κανόνα

$$\text{Διόρθωση βάρους } \Delta w_{ji}(n) =$$
$$\left(\text{παράμετρος ρυθμού μάθησης } \eta\right) \times \left(\text{τοπική κλίση } \delta_j(n)\right) \times \left(\text{σήμα εισόδου νευρώνα } j \; y_i(n)\right)$$

και πως η τοπική κλίση $\delta_j(n)$ εξαρτάται από το εάν ο νευρώνας j είναι ένας νευρώνας εξόδου ή ένας κρυφός κόμβος.

1. Αν ο νευρώνας j είναι ένας κόμβος εξόδου, το $\delta_j(n)$ ισούται με το γινόμενο της παραγώγου $\varphi'(v_j(n))$ και του σήματος σφάλματος $e_j(n)$, αμφότερα εκ των οποίων σχετίζονται με το νευρώνα j.

2. Αν ο νευρώνας j είναι ένας κρυφός κόμβος, το $\delta_j(n)$ ισούται με γινόμενο της σχετιζόμενης παραγώγου $\varphi'(v_j(n))$ και του σταθμισμένου αθροίσματος των δ που υπολογίζονται για τους νευρώνες του επόμενου κρυφού επιπέδου ή επιπέδου εξόδου και οι οποίοι συνδέονται με τον νευρώνα j.

4.4 Περιγραφή Αλγορίθμων Βελτιστοποίησης Gradient Descent

Στην προηγούμενη ενότητα αναφερθήκαμε στον αλγόριθμο του Back Propagation κάνοντας αναφορά και στο γενικό αλγόριθμο του Gradient Descent. Στην ενότητα αυτή, θα περιγράψουμε διαφορετικούς αλγορίθμους Gradient Descent [11] αναλύοντας τη συμπεριφορά τους. Ξεκινάμε κάνοντας μία σύντομη υπενθύμιση της βασικής θεωρίας χρησιμοποιώντας λίγο διαφορετικό συμβολισμό.

Έστω μία αντικειμενική συνάρτηση $\mathbf{J}(\theta)$ πολλαπλών μεταβλητών και έστω ένα σημείο θ στο οποίο η συνάρτηση $\mathbf{J}(\theta)$ ορίζεται και είναι παραγωγίσιμη σε μία γειτονιά του σημείου αυτού. Η αντικειμενική συνάρτηση $\mathbf{J}(\theta)$ μειώνεται γρηγορότερα αν μετακινηθεί από το σημείο θ σε σημείο αντίθετα της κατεύθυνσης της κλίσης της $\mathbf{J}(\theta)$ στο σημείο αυτό. Ο ρυθμός μάθησης η καθορίζει το πλήθος των βημάτων που πραγματοποιούνται μέχρι να φτάσουμε σε (τοπικό ελάχιστο).

Παρακάτω παρουσιάζουμε τις τρεις παραλλαγές του αλγορίθμου Gradient Descent.

4.4.1 Batch Gradient Descent

Ο Batch Gradient Descent υπολογίζει την κλίση της αντικειμενικής συνάρτησης για ολόκληρο το διαθέσιμο σύνολο δεδομένων εκπαίδευσης. Τυπικά αυτό περιγράφεται ως εξής:

$$\theta = \theta - \eta \cdot \nabla_\theta \mathbf{J}(\theta)$$

Στην περίπτωση του Batch Gradient Descent για να ενημερώσουμε τις παραμέτρους μία φορά θα πρέπει να υπολογίσουμε τις κλίσεις για όλο το σύνολο δεδομένων εκπαίδευσης. Η διαδικασία αυτή είναι πολύ αργή για μεγάλα σύνολα δεδομένων εκπαίδευσης, ενώ υπάρχει και η δυνατότητα τα πολύ μεγάλα σύνολα δεδομένων εκπαίδευσης να μη χωρούν ολόκληρα στη μνήμη του υπολογιστή.

4.4.2 Stohastic Gradient Descent

Σε αντίθεση με τον Batch Gradient Descent, ο αλγόριθμος Stohastic Gradient Descent υπολογίζει την κλίση της αντικειμενικής συνάρτησης για κάθε ένα δείγμα εκπαίδευσης $x^{(i)}$ με ετικέτα $y^{(i)}$ από το διαθέσιμο σύνολο δεδομένων εκπαίδευσης ξεχωριστά κάθε φορά. Τυπικά αυτό περιγράφεται ως εξής:

$$\theta = \theta - \eta \cdot \nabla_\theta \mathbf{J}(\theta; x^{(i)}; y^{(i)})$$

Σε αντίθεση με τον Batch Gradient Descent, ο αλγόριθμος Stohastic Gradient Descent είναι αρκετά πιο γρήγορος και υπολογιστικά εφικτός. Επιπρόσθετα, το γεγονός ότι υπολογίζει την κλίση της αντικειμενικής συνάρτησης για κάθε ένα δείγμα εκπαίδευσης ξεχωριστά οδηγεί σε συχνές ενημερώσεις υψηλής διακύμανσης. Αυτό σημαίνει πως μπορεί να μας οδηγήσει σε νέα ελάχιστα. Παρόλα αυτά, η συγκεκριμένη ιδιαιτερότητα δυσχεραίνει τη γρήγορη σύγκλιση. Έχει αποδειχθεί πως αν μειώνουμε το ρυθμό μάθησης σταδιακά ο Stohastic Gradient Descent συγκλίνει σχεδόν πάντα σε ελάχιστο, δείχνοντας παρόμοια συμπεριφορά με τον Batch Gradient Descent.

4.4.3 Mini-Batch Gradient Descent

Ο αλγόριθμος Mini-Batch Gradient Descent είναι ένας υβριδικός συνδυασμός των πιο πάνω αλγορίθμων, καθώς υπολογίζει την κλίση και ενημερώνει τις παραμέτρους για n δείγματα και ετικέτες από το διαθέσιμο σύνολο δεδομένων εκπαίδευσης. Τυπικά αυτό περιγράφεται ως εξής:

$$\Theta = \theta - \eta \cdot \nabla_\theta \mathbf{J}(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

Με τον τρόπο αυτό επιτυγχάνεται μικρότερη διακύμανση, σταθερότερη σύγκλιση και ευκολία στην υπολογιστική διαχείριση. Πολλές state-of-the-art deep learning βιβλιοθήκες χρησιμοποιούν τον αλγόριθμο αυτό. Τυπικά μεγέθη για batch size είναι μεταξύ του 50 και του 256.

Παρόλα αυτά υπάρχουν ορισμένα ζητήματα τα οποία πρέπει να αναφερθούν. Για παράδειγμα, η επιλογή ρυθμού μάθησης είναι δύσκολη, καθώς μικρός ρυθμός μάθησης οδηγεί σε αργή σύγκλιση, ενώ υψηλός ρυθμός μάθησης αυξάνει την διακύμανση και την ταλάντωση γύρω από το ελάχιστο. Επίσης, είναι δύσκολο να προγραμματίσεις εκ των προτέρων πώς θα πρέπει να αλλάξει κατά τη διάρκεια εκπαίδευσης ο ρυθμός μάθησης, καθώς κάθε φορά θα πρέπει ο ρυθμός μάθησης να προσαρμόζεται στο εκάστοτε σύνολο δεδομένων εκπαίδευσης. Επίσης, λόγω του ότι ο ίδιος ρυθμός μάθησης εφαρμόζεται σε όλες τις παραμέτρους, μπορεί πολλές φορές τα χαρακτηριστικά του συνόλου δεδομένων εκπαίδευσης να μην εμφανίζονται στην ίδια συχνότητα και για το λόγο αυτό να μη θέλουμε να τα ενημερώσουμε όλα στον ίδιο βαθμό. Τέλος, δεν πρέπει να ξεχνάμε πως σε μία μη κυρτή συνάρτηση είναι πολύ πιθανό το νευρωνικό μας δίκτυο να παγιδευτεί σε υπο-βέλτιστα τοπικά ελάχιστα, ιδιαίτερα μάλιστα σε σημεία σέλας (saddle points).

Για τους παραπάνω λόγους, υπάρχουν διάφορες τροποποιήσεις που αντιμετωπίζουν κάποια από τα προβλήματα αυτά. Στο πλαίσιο της παρούσας διπλωματικής εργασίας χρησιμοποιούμε τη βελτιστοποίηση Adaptive Moment Estimation για την εκπαίδευση των νευρωνικών μας δικτύων.

4.4.4 Adaptive Moment Estimation (Adam)

Η μέθοδος Adaptive Moment Estimation (Adam) υπολογίζει προσαρμοστικούς ρυθμούς μάθησης για κάθε μία από τις παραμέτρους του νευρωνικού μας δικτύου. Η μέθοδος διατηρεί έναν εκθετικά μειούμενο μέσο όρο προηγούμενων κλίσεων. Ορίζουμε ως $g_{t,i}$ την κλίση της αντικειμενικής συνάρτησης ως προς την παράμετρο θ_i τη χρονική στιγμή t.

$$g_{t,i} = \nabla_{\theta_t} \mathbf{J}(\theta_{t,i})$$

Ορίζουμε ως g_t το διάνυσμα που αποτελείται από όλα τα $g_{t,i}$. Επιπλέον, ορίζουμε ως m_t και v_t τις εκτιμήσεις της πρώτης ροπής (μέση τιμή) και της δεύτερης ροπής (διακύμανσης) των κλίσεων. Πρόκειται για διανύσματα που διατηρούν πληροφορία για κάθε χαρακτηριστικό και είναι αρχικοποιημένα στο 0. Τυπικά περιγράφονται ως:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

όπου β_1 και β_2 θετικές σταθερές. Οι δημιουργοί της μεθόδου παρατήρησαν ότι οι παραπάνω ποσότητες δεν είναι αμερόληπτοι εκτιμητές και για το λόγο τις τροποποίησαν όπως φαίνεται παρακάτω

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t}$$

Ο κανόνας εκπαίδευσης περιγράφεται τυπικά ως:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v_t}} + \varepsilon} \hat{m_t}$$

Οι δημιουργοί της μεθόδου προτείνουν τις τιμές $\beta_1 = 0.9$, $\beta_2 = 0.999$ και $\varepsilon = 10^{-8}$ καθώς εμπειρικές μέθοδοι έχουν δείξει πως οι τιμές αυτές δίνουν καλά αποτελέσματα στην πράξη.

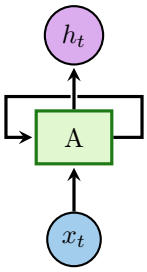
4.5 Αναδρομικά Νευρωνικά Δίκτυα

Στην ενότητα αυτή θα περιγράψουμε μια συγκεκριμένη κατηγορία νευρωνικών δικτύων, τα οποία είναι ιδιαίτερα χρήσιμα για την πρόβλεψη χρονοσειρών: τα Αναδρομικά Νευρωνικά Δίκτυα (Recurrent Neural Networks).

Οι άνθρωποι δεν ξεκινούν τη σκέψη τους από το μηδέν κάθε δευτερόλεπτο. Καθώς διαβάζετε αυτήν την εργασία, καταλαβαίνετε κάθε λέξη με βάση την κατανόηση των προηγούμενων λέξεων. Δεν ξεχνάτε τα πάντα και αρχίζετε να σκέφτεστε ξανά από το μηδέν. Οι σκέψεις σας έχουν διάρκεια.

Τα παραδοσιακά νευρωνικά δίκτυα δεν μπορούν να το κάνουν αυτό και αυτό μοιάζει να είναι ένα μεγάλο μειονέκτημα. Για παράδειγμα, φανταστείτε ότι θέλετε να ταξινομήσετε τι είδους συμβάν συμβαίνει σε κάθε σημείο μιας ταινίας. Δεν είναι σαφές πώς ένα παραδοσιακό νευρωνικό δίκτυο θα μπορούσε να χρησιμοποιήσει το σκεπτικό του για προηγούμενα γεγονότα στην ταινία, προκειμένου να ενημερώσει τα επόμενα.

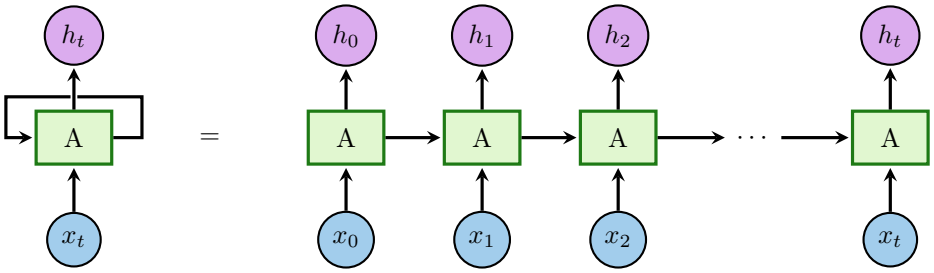
Τα αναδρομικά νευρωνικά δίκτυα αντιμετωπίζουν αυτό το ζήτημα. Είναι δίκτυα με βρόχους, που επιτρέπουν τη διατήρηση της πληροφορίας.



Εικόνα 4.4: Τα Αναδρομικά Νευρωνικά Δίκτυα παρουσιάζουν βρόχους

Στο παραπάνω σχήμα, ένα νευρωνικό δίκτυο A δέχεται μια είσοδο x_t και εξάγει μια τιμή h_t. Ένας βρόχος επιτρέπει στην πληροφορία να περάσει από ένα στάδιο του δικτύου στο επόμενο.

Αυτοί οι βρόχοι κάνουν τα αναδρομικά νευρωνικά δίκτυα να μοιάζουν δυσνόητα. Όμως, με μια πιο προσεκτική ματιά, αποδεικνύεται ότι δεν διαφέρουν καθόλου από τα κλασικά νευρωνικά δίκτυα. Ένα αναδρομικό νευρωνικό δίκτυο μπορεί να θεωρηθεί ως πολλαπλά αντίγραφα του ίδιου δικτύου, καθένα από τα οποία περνά την πληροφορία στο επόμενό του. Αυτή η ιδέα φαίνεται στο παρακάτω σχήμα.



Εικόνα 4.5: Ένα εκτυλισμένο αναδρομικό νευρωνικό δίκτυο

Αυτή η αλυσιδωτή φύση αποκαλύπτει ότι τα αναδρομικά νευρωνικά δίκτυα σχετίζονται στενά με ακολουθίες. Είναι η φυσική αρχιτεκτονική νευρωνικού δικτύου που χρησιμοποιείται για τέτοια δεδομένα.

Τα τελευταία χρόνια, υπήρξε μεγάλη επιτυχία με την εφαρμογή των RNN σε πληθώρα προβλημάτων, όπως η αναγνώριση φωνής, η μοντελοποίηση γλωσσών, η μετάφραση, η αναγνώριση εικόνων, κ.α.

Πολύ σημαντική σε αυτές τις επιτυχίες είναι η χρήση των LSTM, μια ειδική κατηγορία αναδρομικών νευρωνικών δικτύων, τα οποία λειτουργούν, για πολλά προβλήματα, πολύ καλύτερα από την τυπική αρχιτεκτονική RNN. Σχεδόν όλες οι επιτυχίες των αναδρομικών νευρωνικών δικτύων προέρχονται από τη χρήση των LSTM.

4.5.1 Το πρόβλημα των μακροπρόθεσμων εξαρτήσεων

Αυτό που κάνει τα RNN ελκυστικά, είναι η ιδέα ότι μπορεί να είναι σε θέση να συνδέσουν προηγούμενες πληροφορίες με την παρούσα εργασία, όπως η χρήση προηγούμενων καρέ βίντεο μπορεί να ενημερώσει την κατανόηση του τρέχοντος καρέ.

Μερικές φορές χρειαζόμαστε μόνο κάποια πρόσφατη πληροφορία για να εκτελέσουμε μια εργασία. Για παράδειγμα, φανταστείτε ένα γλωσσικό μοντέλο το οποίο προσπαθεί να προβλέψει την επόμενη λέξη βασιζόμενο σε προηγούμενες λέξεις. Αν προσπαθούμε να προβλέψουμε την τελευταία λέξη στην πρόταση "*τα σύννεφα είναι στον* ***ουρανό***", δεν χρειαζόμαστε περαιτέρω στοιχεία - είναι αρκετά προφανές ότι η επόμενη λέξη είναι η λέξη "***ουρανός***". Σε τέτοιες περιπτώσεις, όπου το χάσμα μεταξύ των σχετικών πληροφοριών και του σημείου που χρειάζονται είναι μικρό, τα RNN μπορούν να μάθουν να χρησιμοποιούν παρελθοντικές πληροφορίες.

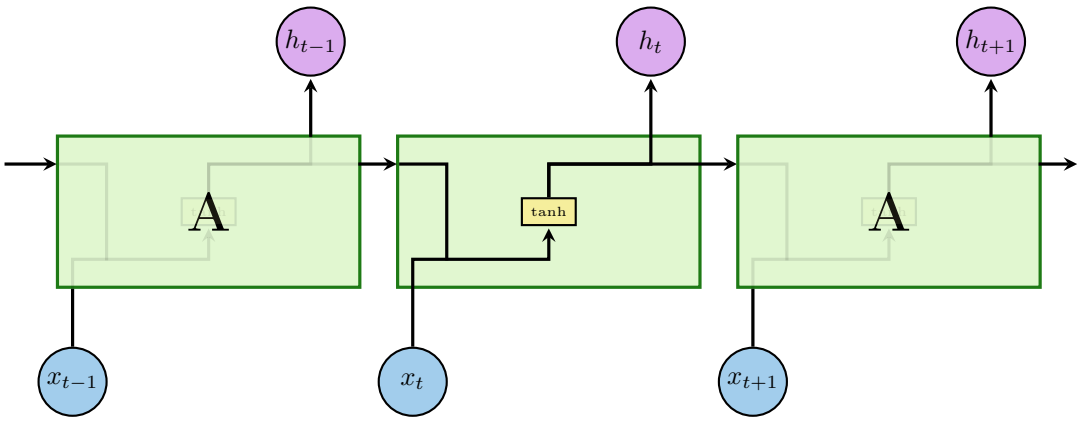
Υπάρχουν όμως περιπτώσεις όπου χρειαζόμαστε περισσότερα στοιχεία. Σκεφτείτε ότι θέλουμε να προβλέψουμε την τελευταία λέξη στο κείμενο "*μεγάλωσα στην Ελλάδα... μιλάω άπταιστα* ***ελληνικά***". Η πρόσφατη πληροφορία δείχνει ότι η επόμενη λέξη θα είναι πιθανότατα το όνομα της γλώσσας, αλλά αν θέλουμε περιορίσουμε τις πιθανές γλώσσες, θα πρέπει να ανατρέξουμε στην πληροφορία ότι ο ομιλητής κατάγεται από την Ελλάδα, η οποία δόθηκε αρκετά πιο πίσω στο παρελθόν. Είναι αρκετά πιθανό το χάσμα μεταξύ των σχετικών πληροφοριών και του σημείου που χρειάζονται να γίνει πολύ μεγάλο. Καθώς αυτό το χάσμα μεγαλώνει, τα RNN αντιμετωπίζουν όλο και μεγαλύτερη δυσκολία στο να μάθουν να συνδέουν πληροφορίες. Το πρόβλημα αυτό έρχονται να λύσουν τα LSTM τα οποία θα παρουσιάσουμε στην επόμενη ενότητα.

4.5.2 Δίκτυα LSTM

Τα δίκτυα Long Short Term Memory [7], που για συντομία αναφέρονται ως LSTM, αποτελούν μια ειδική κατηγορία αναδρομικών νευρωνικών δικτύων, τα οποία είναι σε θέση να μαθαίνουν μακροπρόθεσμες εξαρτήσεις. Προτάθηκαν από τους Hochreiter και Schmidhuber το 1997. Λειτουργούν εξαιρετικά καλά σε μια μεγάλη ποικιλία προβλημάτων και χρησιμοποιούνται πλέον ευρέως.

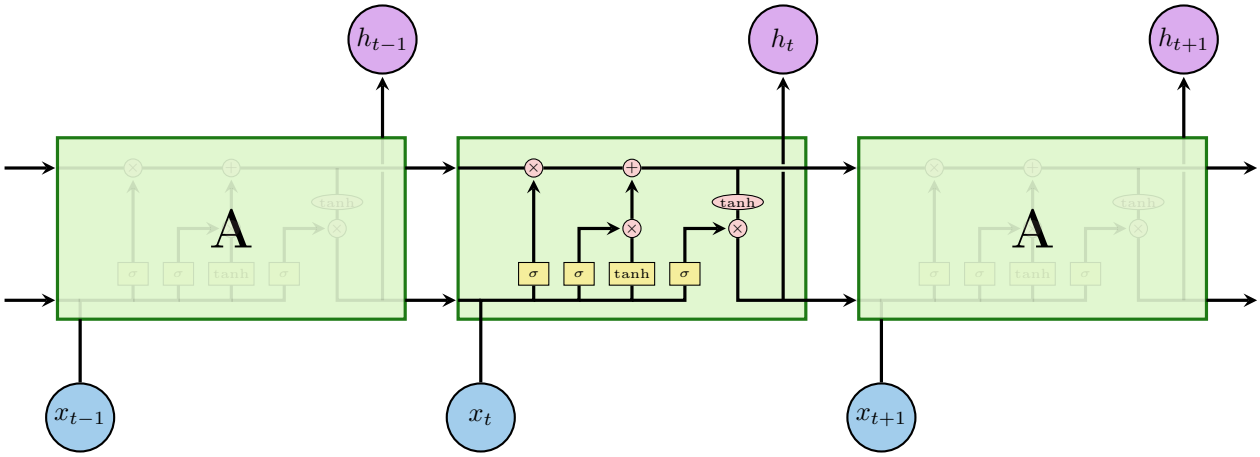
Τα LSTM έχουν σχεδιαστεί ρητά για την αποφυγή του προβλήματος των μακροπρόθεσμων εξαρτήσεων. Η απομνημόνευση πληροφοριών για μεγάλα χρονικά διαστήματα είναι ουσιαστικά η προεπιλεγμένη συμπεριφορά τους, όχι κάτι που δυσκολεύονται να μάθουν.

Όλα τα αναδρομικά νευρωνικά δίκτυα έχουν τη μορφή μιας αλυσίδας επαναλαμβανόμενων μονάδων νευρωνικού δικτύου. Στα τυπικά RNN, αυτή η επαναλαμβανόμενη μονάδα έχει μια πολύ απλή δομή, όπως ένα μονό επίπεδο tanh.



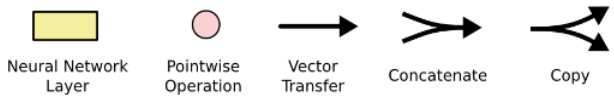
Εικόνα 4.6: Η επαναλαμβανόμενη μονάδα σε ένα τυπικό RNN περιέχει ένα μόνο επίπεδο

Τα LSTM έχουν και αυτά μορφή μιας αλυσίδας, αλλά οι επαναλαμβανόμενες μονάδες έχουν διαφορετική δομή. Αντί για ένα απλό επίπεδο νευρωνικού δικτύου, έχουν τέσσερα επίπεδα τα οποία αλληλεπιδρούν με έναν ιδιαίτερο τρόπο.



Εικόνα 4.7: Η επαναλαμβανόμενη μονάδα σε ένα LSTM περιέχει τέσσερα αλληλεπιδρώντα επίπεδα

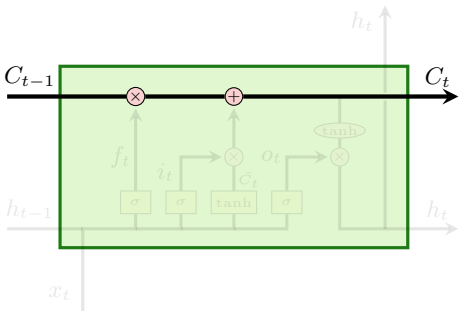
Στο παραπάνω διάγραμμα, κάθε γραμμή φέρει ένα διάνυσμα, από την έξοδο ενός κόμβου εως τις εισόδους των άλλων. Οι ροζ κύκλοι αντιπροσωπεύουν τις σημείο προς σημείο πράξεις, όπως άθροιση διανυσμάτων, ενώ τα κίτρινα κουτιά αποτελούν επίπεδα νερωνικού δικτύου. Οι γραμμές που ενώνονται αντιπροσωπεύουν συνένωση διανυσμάτων ενώ οι γραμμές που διαχωρίζονται αντιπροσωπεύουν αντίγραφα ενός διανύσματος τα οποία πάνε σε διαφορετικούς προορισμούς. Στην παρακάτω εικόνα παρουσιάζεται συνοπτικά ο συμβολισμός που χρησιμοποιούμε.



Εικόνα 4.8: Ο συμβολισμός που χρησιμοποιείται για την περιγραφή των LSTM

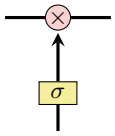
4.5.3 Η βασική ιδέα πίσω από τα LSTM

Το κλειδί για τα LSTM είναι η κατάσταση κελιού, η οριζόντια γραμμή που διατρέχει την κορυφή του διαγράμματος. Η κατάσταση κελιού είναι σαν μια μεταφορική ταινία. Διατρέχει ολόκληρη την αλυσίδα, με μόνο μερικές μικρές γραμμικές αλληλεπιδράσεις. Είναι πολύ εύκολο για τις πληροφορίες να ρέουν αμετάβλητα.



Εικόνα 4.9: Η κατάσταση κελιού ενός LSTM

Τα LSTM έχουν τη δυνατότητα να αφαιρούν ή να προσθέτουν πληροφορίες στην κατάσταση κελιού, η οποία ρυθμίζεται από δομές που ονομάζονται πύλες. Οι πύλες είναι ένας τρόπος για το ελεγχόμενο πέρασμα των πληροφοριών. Αποτελούνται από ένα σιγμοειδές νευρωνικό επίπεδο και έναν στοιχείο προς στοιχείο πολλαπλασιαστή.



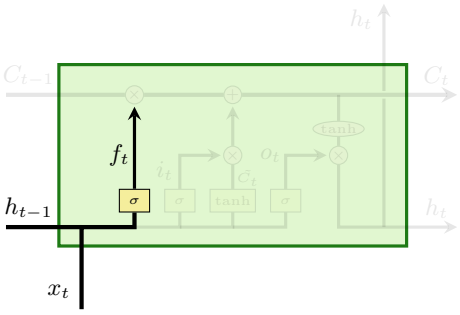
Εικόνα 4.10: Πύλη εντός μιας μονάδας ενός δικτύου LSTM

Το σιγμοειδές επίπεδο εξάγει αριθμούς στο διάστημα $[0, 1]$, περιγράφοντας τι ποσοστό της πληροφορίας θέλουμε να περάσει στα επόμενα στάδια. Μια τιμή κοντά στο 0 σημαίνει ότι περνά πολύ μικρό μέρος της πληροφορίας, ενώ μια τιμή κοντά στο 1 σημαίνει ότι περνά ένα αρκετά μεγάλο μέρος της πληροφορίας. Ένα LSTM διαθέτει τρεις τέτοιες πύλες για την προστασία και τον έλεγχο της κατάστασης κελιού.

4.5.4 Βήμα προς βήμα περιγραφή των LSTM

Το πρώτο βήμα της ανάλυσης των LSTM δικτύων αφορά την απόφαση σχετικά με το τι πληροφορία θα απορρίψουμε από την κατάσταση κελιού. Αυτή η απόφαση λαμβάνεται από ένα σιμοειδές επίπεδο, την *πύλη απόρριψης*. Δέχεται ως είσοδο την συνένωση των h_{t-1} και x_t και εξάγει έναν αριθμό στο διάστημα $[0, 1]$, για κάθε αριθμό στην κατάσταση κελιού C_{t-1} Το 1 αντιπροσωπεύει την εξ ολοκήρου διατήρηση της πληροφορίας ενώ το 0 αντιπροσωπεύει την εξ ολοκλήρου απόρριψη της πληροφορίας.

Ας θυμηθούμε το παράδειγμα με το γλωσσικό μοντέλο το οποίο προσπαθεί να προβλέψει την επόμενη λέξη βασιζόμενο σε όλες τις προηγούμενες λέξεις. Σε ένα τέτοιο πρόβλημα, η κατάσταση του κελιού μπορεί να περιλαμβάνει το φύλο του παρόντος υποκειμένου, έτσι ώστε να μπορούν να χρησιμοποιηθούν οι σωστές αντωνυμίες. Όταν βλέπουμε ένα νέο υποκείμενο, θέλουμε να ξεχάσουμε το φύλο του παλιού υποκειμένου.

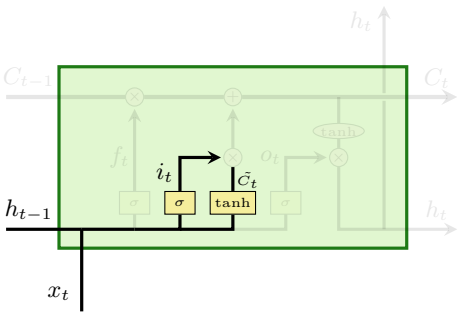


$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big)$$

Εικόνα 4.11: Η πύλη απόρριψης ενός LSTM

Το επόμενο βήμα αφορά την απόφαση σχετικά με το ποιες νέες πληροφορίες πρόκειται να αποθηκεύσουμε στην κατάσταση κελιού. Αυτή η απόφαση αποτελείται από δύο σκέλη. Πρώτα, ένα σιγμοειδές επίπεδο, η *πύλη εισόδου*, αποφασίζει ποιες τιμές θα ενημερώσουμε. Ύστερα, ένα tanh επίπεδο δημιουργεί ένα διάνυσμα με νέες υποψήφιες τιμές, $\tilde{C}_t$, το οποίο μπορεί να προστεθεί στην κατάσταση κελιού. Στο επόμενο βήμα, θα συνδυάσουμε αυτά τα δύο για να ενημερώσουμε την κατάσταση κελιού.

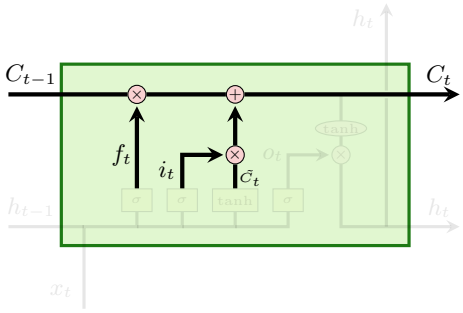
Στο παράδειγμα του γλωσσικού μοντέλου μας, θα θέλαμε να προσθέσουμε το φύλο του νέου υποκειμένου στην κατάσταση κελιού, για να αντικαταστήσουμε το παλιό.



$$i_t = \sigma\big(W_i \cdot [h_{t-1}, x_t] + b_i\big)$$
$$\tilde{C}_t = \tanh\big(W_C \cdot [h_{t-1}, x_t] + b_C\big)$$

Εικόνα 4.12: Η διαδικασία παραγωγής των διανυσμάτων i_t και $\tilde{C}_t$

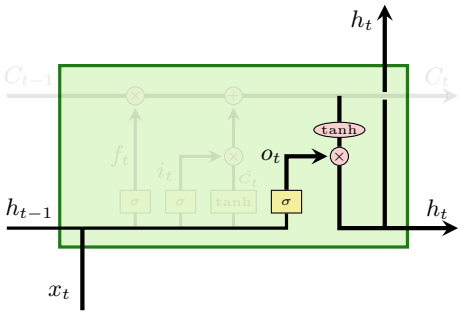
Θα πρέπει τώρα να ενημερώσουμε την παλιά κατάσταση κελιού C_{t-1}, ώστε να πάρουμε τη νέα κατάσταση κελιού C_t. Πολλαπλασιάζουμε την παλιά κατάσταση με το f_t, ώστε να απορρίψουμε την πληροφορία που επιθυμούμε να απορρίψουμε. Στη συνέχεια προσθέτουμε το $i_t * \tilde{C}_t$, το οποίο αποτελεί τις νέες υποψήφιες τιμές, οι οποίες κλιμακώνονται από το πόσο πολύ αποφασίσαμε να ενημερώσουμε την κάθε τιμή. Στην περίπτωση του γλωσσικού μοντέλου, εδώ θα απορρίψουμε τις πληροφορίες σχετικά με το φύλο του παλιού υποκείμενου και θα προσθέσουμε τις νέες πληροφορίες, όπως αποφασίσαμε στα προηγούμενα βήματα. Να σημειώσουμε στο σημείο αυτό, ότι με $*$ συμβολίζουμε τον στοιχείο προς στοιχείο πολλαπλασιασμό διανυσμάτων.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Εικόνα 4.13: Η διαδικασία παραγωγής της νέας κατάστασης κελιού C_t

Τέλος, θα πρέπει να αποφασίσουμε ποια θα είναι η έξοδος. Η έξοδος αυτή θα βασιστεί στην νέα κατάσταση κελιού, αλλά θα είναι μια φιλτραρισμένη εκδοχή του. Αρχικά, εφαρμόζουμε ένα σιγμοειδές επίπεδο, το οποίο αποφασίζει ποια μέρη της κατάστασης κελιού θα εξάγουμε. Στη συνέχεια, εφαρμόζουμε την κατάσταση κελιού σε ένα tanh επίπεδο, ώστε οι τιμές να μετακινηθούν στο διάστημα $[-1, 1]$, και το πολλαπλασιάζουμε με την έξοδο της σιγμοειδούς πύλης, ώστε να εξάγουμε μόνο τα μέρη της κατάστασης κελιού που επιθυμούμε.



$$o_t = \sigma\big(W_o \cdot [h_{t-1}, x_t] + b_o\big)$$
$$h_t = o_t * \tanh(C_t)$$

Εικόνα 4.14: Η διαδικασία παραγωγής της εξόδου του LSTM

Κεφάλαιο 5

Προσεγγίσεις για την Επίλυση του Προβλήματος

5.1 Δέσμευση Πόρων

Όπως αναφέρθηκε και στην ενότητα 2.2, στοχεύσαμε στη δημιουργία ενός συστήματος το οποίο απευθύνεται σε χρήστες που χρησιμοποιούν την υπηρεσία των Spot Instances και επιθυμούν να εκτελέσουν κάποιες εργασίες εντός ενός συγκεκριμένου χρονικού διαστήματος με το ελάχιστο δυνατό κόστος. Οι δύο άγνωστες παράμετροι οι οποίες δυσκολεύουν αυτήν την προσπάθεια ελαχιστοποίησης του κόστους εκτέλεσης των εργασιών είναι το μέγεθος του φόρτου εργασιών και η τιμή των Spot Instances. Αξίζει στο σημείο αυτό να δούμε πώς η εκ των προτέρων γνώση αυτών των δύο παραμέτρων θα μας βοηθούσε σε αυτήν την προσπάθεια.

Στις επόμενες ενότητες πραγματοποιείται μια λεπτομερής ανάλυση, παρουσιάζοντας τον απαραίτητο συμβολισμό και τις παραδοχές μας, προκειμένου να γίνει σαφής ο τρόπος με τον οποίο αξιοποιούμε την εκ των προτέρων γνώση των προαναφερθέντων παραμέτρων.

5.1.1 Στρατηγική Χρέωσης

Υποθέτουμε ότι ο πάροχος της υπηρεσίας μπορεί να ανακοινώσει ανά πάσα στιγμή μια νέα *αγοραία τιμή* για κάθε τύπο μηχανήματος. Η αγοραία τιμή παραμένει η ίδια μέχρι νεωτέρας, αλλά δεν υπάρχει καμία εγγύηση για το πότε και πόσο θα αλλάξει ξανά. Οι χρήστες μπορούν να αποκτήσουν μηχανήματα ανά πάσα στιγμή υποβάλλοντας μια *προσφορά*, η οποία αποτελείται από μια τιμή προσφοράς, καθώς και τον αριθμό και τον τύπο των απαιτούμενων μηχανημάτων. Προκειμένου μια προσφορά να είναι επιτυχής, η τιμή προσφοράς δεν πρέπει να είναι μικρότερη από την τρέχουσα αγοραία τιμή για τον τύπο του μηχανήματος που ζητείται. Μόλις μια προσφορά υποβληθεί, δεν μπορεί να τροποποιηθεί.

Ο πάροχος της υπηρεσίας χρεώνει κάθε μηχάνημα που χρησιμοποιείται σε χρονικές προσαυξήσεις μεγέθους τ_{charge}. Συγκεκριμένα, ο χρόνος χρήσης στρογγυλοποιείται προς τα πάνω σε ένα πολλαπλάσιο του τ_{charge}, και κάθε περίοδος μεγέθους τ_{charge} χρεώνεται με την αγοραία τιμή στην έναρξη της περιόδου. Για την Amazon, $\tau_{\text{charge}} = 3600s$ (μια ώρα), αν και η τελευταία μερική ώρα είναι δωρεάν αν το μηχάνημα τερματιστεί από την Amazon. Για την Google, $\tau_{\text{charge}} = 60s$ (ένα λεπτό).

5.1.2 Διαμόρφωση Συστάδων

Για την εκτέλεση των εργασιών, ο χρήστης διαμορφώνει συστάδες υπολογιστών, δηλαδή σύνολα δύο ή περισσότερων μηχανημάτων τα οποία λειτουργούν παράλληλα για την εκτέλεση μιας εργασίας. Για λόγους απλότητας, υποθέτουμε ότι ο χρήστης διαμορφώνει μια *ομογενή* συστάδα, δηλαδή όλα τα μηχανήματα προέρχονται από την ίδια αγορά και έχουν τον ίδιο τύπο. Αυτά τα μηχανήματα μπορούν, φυσικά, να αγοραστούν και να τερματιστούν σε διαφορετικούς χρόνους. Στο υπόλοιπο της συγκεκριμένης διπλωματικής εργασίας, επικεντρωνόμαστε στο πρόβλημα της *βέλτιστης προσαρμογής του μεγέθους της υπολογιστικής συστάδας*, δεδομένου του τύπου μηχανήματος.

5.1.3 Ροή Εργασιών και Συναρτήσεις Επιτάχυνσης

Ας υποθέσουμε ότι έχουμε να εκτελέσουμε μια ροή εργασιών η οποία αποτελείται από J εργασίες. Συμβολίζουμε με $T^{(j)}(n)$ τον χρόνο εκτέλεσης της εργασίας j, για $j \in \{1, 2, \cdots, J\}$, χρησιμοποιώντας n μηχανήματα, υποθέτωντας ότι δεν υπάρχουν διακοπές. Με $w^{(j)} = T^{(j)}(1)$ συμβολίζουμε το συνολικό έργο στην εργασία j, η μονάδα μέτρησης του οποίου είναι το έργο που εκτελείται από ένα μόνο μηχάνημα στη μονάδα χρόνου. Το συνολικό έργο της ροής εργασιών συμβολίζεται με W και ισχύει

$$W = \sum_{j=1}^{J} w^{(j)} \tag{5.1}$$

Η *συνάρτηση επιτάχυνσης* της εργασίας j δίνεται από τον τύπο

$$g^{(j)}(n) = \frac{T^{(j)}(1)}{T^{(j)}(n)} \tag{5.2}$$

Γενικότερα, η παραλληλοποίηση εργασιών παράγει φθίνουσες αποδόσεις λόγω της υπολογιστικής της επιβάρυνσης και οποιουδήποτε εγγενώς σειριακού τμήματος του υπολογισμού. Στην πράξη, ακόμη και για εξαιρετικά παραλληλοποιήσιμες εργασίες όπως ο πολλαπλασιασμός πινάκων, παρατηρούμε ότι η $g^{(j)}(n)$ είναι αύξουσα, κοίλη και ανήκει στην κλάση $o(n)$, με $g^{(j)}(1) = 1$ και $g^{(j)}(n) \leq n$. Λόγω αυτής της φθίνουσας απόδοσης, θέτουμε ένα μέγιστο όριο στον αριθμό των μηχανημάτων που χρησιμοποιούνται ταυτόχρονα στη συστάδα, καθώς μια μεγαλύτερη συστάδα δεν θα είναι πλεον οικονομικά αποδοτική. Συμβολίζουμε αυτό το όριο με N.

5.1.4 Πλάνο, Ενέργειες και Υπολογιστική Επιβάρυνση

Δεδομένης μια προθεσμίας, αναζητούμε ένα *πλάνο* το οποίο κωδικοποιεί με την πάροδο του χρόνου την ακολουθία των ενεργειών, προκειμένου να ολοκληρωθεί η ροή εργασιών εντός της προθεσμίας. Μια *ενέργεια* μπορεί να είναι προσφορά για περισσότερα μηχανήματα σε κάποια τιμή ή αποδέσμευση ενός υποσυνόλου μηχανημάτων. Κατά την διατύπωση και την επίλυση του προβλήματος βελτιστοποίησης, κάνουμε δύο απλοποιήσεις, οι οποίες δεν μας περιορίζουν στην πράξη:

- Διακριτοποιούμε τον χρόνο σε βήματα μεγέθους τ_{opt}, και εξετάζουμε ενέργειες μόνο στην αρχή κάθε χρονικού βήματος. Η επιλογή του τ_{opt} αντικατοπτρίζει τον συμβιβασμό μεταξύ χρόνου βελτιστοποίησης και ποιότητας λύσης.
- Αγνοούμε το πρόβλημα του καθορισμού τιμής προσφοράς, και υποθέτουμε ότι κάνουμε μια αρκετά υψηλή προσφορά. Για τον σκοπό της βελτιστοποίησης, θέτουμε απλώς τιμή προσφοράς ίση με ∞, που σημαίνει ότι τα μηχανήματα δεν θα τερματιστούν από τον πάροχο της υπηρεσίας, αλλά μπορούν και πάλι να αποδεσμευτούν όποτε εμείς θελήσουμε.

Επομένως, με τις παραπάνω απλοποιήσεις, υποθέτοντας ότι η τρέχουσα χρονική στιγμή είναι 0 και δεδομένης της προθεσμίας d (σε μονάδες τ_{opt}), θα πρέπει απλώς να επιλέξουμε την ακολουθία $\langle n_t : 0 \leq t \leq d-1 \rangle$, όπου με n_t συμβολίζουμε το επιλεγμένο μέγεθος της συστάδας την χρονική στιγμή t. Όπου είναι σαφές, θα συμβολίζουμε το πλάνο ως $\langle n_t \rangle$.

Οποιαδήποτε μεταβολή στο μέγεθος της συστάδας κατά τον χρόνο εκτέλεσης επηρεάζει την κατανομή σε εργασίες του εναπομένοντος έργου. Αυτό το φαινόμενο γίνεται εμφανές και από τις συναρτήσεις επιτάχυνσης που ορίσαμε νωρίτερα. Στην πραγματικότητα, η αλλαγή του μεγέθους της συστάδας προκαλεί και διάφορες υπολογιστικές επιβαρύνσεις. Για παράδειγμα, όταν αποκτώνται νέα μηχανήματα, απαιτείται χρόνος για την αρχικοποίησή τους, χρόνος κατά τον οποίο δεν παράγουν χρήσιμο έργο αλλά χρεώνονται κανονικά ωστόσο. Επίσης, όταν μηχανήματα τερματίζονται ή αποδεσμεύονται, ένα μέρος του έργου σε εκκρεμούσες εργασίες μπορεί να χαθεί, το οποίο θα πρέπει να αναπληρωθεί από άλλα μηχανήματα, με αποτέλεσμα την καθυστέρηση της προόδου εκτέλεσης. Συμβολίζουμε με $\texttt{Overhead}(\varpi, n, n')$ την *επιβάρυνση προσαρμογής* που αφορά το μέγεθος μη χρήσιμου ή σπαταλημένου έργου που πραγματοποιήθηκε, όπου ϖ το μέγεθος του έργου που απομένει στη ροή εργασιών την χρονική στιγμή που το μέγεθος της συστάδας αλλάζει από n σε n'. Μπορούμε εύλογα να υποθέσουμε η διευθέτηση αυτής της επιβάρυνσης μιας ενέργειας προσαρμογής του μεγέθους της συστάδας δεν διαρκεί παραπάνω από ένα χρονικό βήμα τ_{opt}, διαφορετικά θα έπρεπε να επιλέξουμε ένα μεγαλύτερο τ_{opt} για να αποφύγουμε την πλεονάζουσα προσαρμογή.

5.1.5 Πρόοδος Πλάνου

Παρουσιάζουμε τώρα πώς μπορούμε να μετρήσουμε την πρόοδο εκτέλεσης σύμφωνα με ένα πλάνο. Δεδομένης μιας ροής εργασιών με J εργασίες και συνολικό έργο W (5.1), συμβολίζουμε με $\texttt{Progress}(\varpi, n, n')$ την πρόοδο που σημειώνεται από μια συστάδα σε ένα χρονικό βήμα. Συγκεκριμένα, με ϖ συμβολίζουμε το μέγεθος του έργου που απομένει κατά την έναρξη του χρονικού βήματος. Τα n και n' είναι τα μεγέθη της συστάδας στο προηγούμενο και στο τρέχον χρονικό βήμα, αντίστοιχα. Η $\texttt{Progress}(\varpi, n, n')$ επιστρέφει το μέγεθος του έργου που απομένει στη ροή εργασιών στο τέλος του τρέχοντος χρονικού βήματος.

Παρακάτω δίνεται ο ορισμός της $\texttt{Progress}(\varpi, n, n')$:

Ορισμός 1

Αν $n' = 0$, τότε $\texttt{Progress}(\varpi, n, n') = \varpi$. Διαφορετικά, ορίζουμε τα παρακάτω μεγέθη:

- Η εργασία κατά τη διάρκεια της οποίας ξεκινά το τρέχον χρονικό βήμα

$$j_{\vdash} = 1 + \max\left\{ j \;\middle|\; \sum_{k=1}^{j} w^{(k)} \leq W - \varpi \right\}$$

- Η ποσότητα έργου που απομένει στην εργασία $j_{\vdash}$ κατά την έναρξη του τρέχοντος χρονικού βήματος

$$\varpi_{\vdash} = \left(\sum_{k=1}^{j_{\vdash}} w^{(k)} \right) - (W - \varpi)$$

- Η εργασία κατά τη διάρκεια της οποίας τελειώνει το τρέχον χρονικό βήμα

$$j_{\dashv} = 1 + \max\left\{ j \;\middle|\; \frac{\varpi_{\vdash} + \texttt{Overhead}(\varpi, n, n')}{g^{(j_{\vdash})}(n')} + \sum_{k=j_{\vdash}+1}^{j} \frac{w^{(k)}}{g^{(k)}(n')} < \tau_{\text{opt}} \right\}$$

- Η ποσότητα έργου που απομένει στην εργασία $j_{\dashv}$ στο τέλος του τρέχοντος χρονικού βήματος

$$\varpi_{\dashv} = g^{(j_{\dashv})}(n') \cdot \left[\frac{\varpi_{\vdash} + \texttt{Overhead}(\varpi, n, n')}{g^{(j_{\vdash})}(n')} + \left(\sum_{k=j_{\vdash}+1}^{j_{\dashv}} \frac{w^{(k)}}{g^{(k)}(n')} \right) - \tau_{\text{opt}} \right], \text{ αν } j_{\dashv} \neq J + 1$$

ή

$$\varpi_{\dashv} = 0, \text{ αν } j_{\dashv} = J + 1$$

Έχοντας ορίσει όλα τα παραπάνω μεγέθη, μπορούμε πλεον να ορίσουμε την $\texttt{Progress}(\varpi, n, n')$ ως:

$$\texttt{Progress}(\varpi, n, n') = \varpi_{\dashv} + \sum_{k=j_{\dashv}+1}^{J} w^{(k)}$$

Έχοντας ορίσει την `Progress`, μπορούμε να ορίσουμε την $\texttt{Workleft}(\langle n_i : i \leq t \rangle)$, δηλαδή την ποσότητα έργου που απομένει στη ροή εργασιών μετά από χρόνο t ακολουθώντας το πλάνο.

Ορισμός 2

Η παρακάτω αναδρομή ορίζει την `Workleft`:

- $\texttt{Workleft}(\langle\rangle) = W$
- $\texttt{Workleft}(\langle n_i : i \leq t \rangle) = \texttt{Progress}(\texttt{Workleft}(\langle n_i : i \leq t-1 \rangle), n_{t-1}, n_t)$,

όπου $n_{-1} = 0$ χάριν ευκολίας συμβολισμού.

5.1.6 Κόστος Πλάνου

Έστω $p(\tau)$ η αγοραία τιμή την χρονική στιγμή τ. Θυμηθείτε ότι αυτή μπορεί να αλλάξει σε οποιαδήποτε στιγμή. Η αναλυτική εξαγωγή του $\texttt{Cost}(\langle n_t \rangle)$, δηλαδή του χρηματικού κόστους εκτέλεσης του πλάνου, μπορεί να γίνει πολύ δύσκολη, λόγω της ασυμφωνίας μεταξύ των τ_{opt} και τ_{charge}. Το τ_{charge} ορίζεται από τον πάροχο της υπηρεσίας, ενώ το τ_{opt} μπορούμε να το επιλέξουμε εμείς. Περιορίζουμε τις επιλογές του τ_{opt} σε δύο περιπτώσεις: $\tau_{\text{charge}} | \tau_{\text{opt}}$, ή $\tau_{\text{opt}} | \tau_{\text{charge}}$.

Ξεκινάμε με την πιο απλή περίπτωση, όταν $\tau_{\text{charge}} | \tau_{\text{opt}}$. Κατά τη διάρκεια του χρονικού βήματος t, τα n_t μηχανήματα θα χρεωθούν $\lambda = \tau_{\text{opt}} / \tau_{\text{charge}}$ φορές. Επομένως, θα έχουμε:

$$\texttt{Cost}(\langle n_t \rangle) = \sum_{t=0}^{d-1} n_t \sum_{i=0}^{\lambda-1} p(t\tau_{\text{opt}} + i\tau_{\text{charge}})$$

Στην ειδική περίπτωση που $\tau_{\text{opt}} = \tau_{\text{charge}}$ η παραπάνω έκφραση απλοποιείται σε:

$$\texttt{Cost}(\langle n_t \rangle) = \sum_{t=0}^{d-1} n_t p(t\tau_{\text{opt}})$$

Μελετάμε τώρα την περίπτωση όπου $\tau_{\text{opt}} | \tau_{\text{charge}}$. Η περίπτωση αυτή είναι πιο περίπλοκη, καθώς στην έναρξη του χρονικού βήματος t, δεν θα χρεωθούν όλα εκ των n_t μηχανημάτων, καθώς μερικά από αυτά έχουν ήδη χρεωθεί σε προηγούμενα χρονικά βήματα. Για να κάνουμε τον υπολογισμό του `Cost` πιο εύκολο, εξάγουμε από την ακολουθία $\langle n_t \rangle$ την ακολουθία $\langle m_t \rangle$, όπου m_t ο αριθμός των μηχανημάτων που πραγματικά χρεώνονται στο χρονικό βήμα t. Έστω $\kappa = \tau_{\text{charge}} / \tau_{\text{opt}}$. Σημειώστε ότι μόλις ένα μηχάνημα χρεωθεί για άλλη μια χρονική περίοδο τ_{charge}, δεν υπάρχει λόγος να το τερματίσουμε πριν από κ χρονικά βήματα, καθώς έχουμε ήδη πληρώσει για την χρήση του. Επομένως, για το χρονικό βήμα t, θα πρέπει να διατηρήσουμε όλα εκ των $\sum_{i=1}^{\kappa-1} m_{t-i}$ μηχανήματα που χρεώθηκαν στα προηγούμενα $\kappa - 1$ βήματα. Επιπρόσθετα, θα πρέπει να επιλέξουμε $m_t = n_t - \sum_{i=1}^{\kappa-1} m_{t-i}$ μηχανήματα για να χρεωθούν στο χρονικό βήμα t. Αυτά τα m_t μηχανήματα θα πρέπει όσο το δυνατόν περισσότερο να προέρχονται από τα $m_{t-\kappa}$ για τα οποία πληρώσαμε στο χρονικό βήμα $t - \kappa$. Μπορούμε δραστικά να τερματίσουμε κάποια αν $m_t < m_{t-k}$, ή να κάνουμε μια προσφορά για περισσότερα αν $m_t > m_{t-\kappa}$. Οπότε, η ακολουθία $\langle m_t \rangle$ μπορεί να εξαχθεί από την $\langle n_t \rangle$ χρησιμοποιώντας την παρακάτω αναδρομή:

$$m_t = n_t - \sum_{1 \leq i < \kappa,\ t-i \geq 0} m_{t-i}$$

Εναλλακτικά, μπορούμε να εξάγουμε την $\langle n_t \rangle$ από την $\langle m_t \rangle$ ως εξής:

$$n_t = \sum_{0 \leq i < \kappa,\ t-i \geq 0} m_{t-i}$$

Έχοντας ορίσει την ακολουθία $\langle m_t \rangle$, μπορούμε να εξάγουμε το κόστος στην περίπτωση όπου $\tau_{\text{opt}} | \tau_{\text{charge}}$ ως

$$\texttt{Cost}(\langle n_t \rangle) = \sum_{t=0}^{d-1} m_t p(t\tau_{\text{opt}})$$

Τέλος, κάνουμε μια ενοποίηση των ορισμών του $\texttt{Cost}(\langle n_t \rangle)$ για τις δύο παραπάνω περιπτώσεις.

Ορισμός 3

Ορίζουμε το κόστος εκτέλεσης του πλάνου $\langle n_t \rangle$ ως $\texttt{Cost}(\langle n_t \rangle) = \sum_{t=0}^{d-1} m_t \cdot \texttt{Charge}(t)$, όπου:

- $\kappa = \lceil \tau_{\text{charge}} / \tau_{\text{opt}} \rceil$
- $m_t = n_t - \sum_{1 \le i < \kappa,\ t-i \ge 0} m_{t-i}$, για $t = 0, \cdots, d-1$
- $\texttt{Charge}(t) = \sum_{i=0}^{\lceil \tau_{\text{opt}} / \tau_{\text{charge}} \rceil - 1} p(t\tau_{\text{opt}} + i\tau_{\text{charge}})$

Παρατηρήστε ότι αν $\tau_{\text{charge}} | \tau_{\text{opt}}$, τότε σύμφωνα με τον παραπάνω ορισμό θα έχουμε $\kappa = 1$ και $\langle m_t \rangle = \langle n_t \rangle$.

5.1.7 Βελτιστοποίηση Κόστους

Έχοντας ορίσει όλα τα απαραίτητα μεγέθη, το πρόβλημα βελτιστοποίησης μπορεί να διατυπωθεί ως εξής: δεδομένης μια προθεσμίας d, επιλέξτε την ακολουθία $\langle n_t : 0 \le t \le d-1 \rangle$ που ελαχιστοποιεί το κόστος $\texttt{Cost}(\langle n_t \rangle)$ υπό τη συνθήκη $\texttt{Workleft}(\langle n_t \rangle) = 0$, δηλαδή η ροή εργασιών ολοκληρώνεται πριν από την προθεσμία. Παρουσιάζουμε δύο αλγορίθμους:

Ένας Απλός Άπληστος Αλγόριθμος

Ας υποθέσουμε ότι $\tau_{\text{charge}} | \tau_{\text{opt}}$ και ότι όλες οι εργασίες έχουν την ίδια συνάρτηση επιτάχυνσης, δηλαδή $g^{(j)}(n) = g(n),\ \forall j \in \{1, 2, \cdots, J\}$. Στην περίπτωση αυτή, μπορούμε να υιοθετήσουμε μια επαναληπτική άπληστη προσέγγιση όπου κάθε φορά δεσμεύουμε ένα μηχάνημα σε ένα επιλεγμένο χρονικό βήμα. Ξεκινώντας με $n_t = 0\ \forall t$, επιλέγουμε το χρονικό βήμα t που μεγιστοποιεί την ποσότητα $(g(n_t + 1) - g(n_t))/\texttt{Charge}(t)$ και αυξάνουμε το n_t κατά ένα. Εφόσον οι μελλοντικές αγοραίες τιμές είναι γνωστές, τα κόστη $\texttt{Charge}(t)$ είναι επίσης γνωστά. Διαισθητικά, επιλέγουμε πάντα τη δέσμευση μηχανημάτων με τον υψηλότερο λόγο έργου προς κόστος. Επαναλαμβάνουμε αυτή τη διαδικασία εως ότου $\texttt{Workleft}(\langle n_t \rangle) = 0$.

Παρακάτω αποδεικνύουμε την βελτιστότητα του άπληστου αυτού αλγορίθμου, με την υπόθεση ότι η $g(n)$ είναι κοίλη και πως δεν υπάρχει επιβάρυνση προσαρμογής, δηλαδή το $\texttt{Overhead}(\varpi, n, n')$ είναι πάντα 0.

Απόδειξη: Στην ειδική περίπτωση όπου $\texttt{Overhead}(\varpi, n, n') = 0$ και $g^{(j)}(n) = g(n)\ \forall j$ θα έχουμε

$$\begin{cases} \texttt{Progress}(\varpi, n, n') = \max\left(0, \varpi - g(n')\tau_{\text{opt}}\right) \\ \texttt{Workleft}(\langle n_i : i \le t \rangle) = W - \sum_{i=0}^{t} g(n_i)\tau_{\text{opt}} \end{cases}$$

Υποθέτοντας ότι $\tau_{\text{charge}} | \tau_{\text{opt}}$ και ντετερμινιστική αγοραία τιμή, θα έχουμε επίσης τα $m_t = n_t$ και $\texttt{Charge}(t)$ ως γνωστές σταθερές. Το πρόβλημα βελτιστοποίησης απλοποιείται ως εξής: ελαχιστοποίηση του $\sum_{t=0}^{d-1} n_t \texttt{Charge}(t)$, υπό τη συνθήκη $\sum_{t=0}^{d-1} g(n_t)\tau_{\text{opt}} \ge W$. Θα αποδείξουμε χρησιμοποιώντας τη μέθοδο της μαθηματικής επαγωγής, ότι σε κάθε βήμα του άπληστου αλγορίθμου, ο οποίος αυξάνει το n_t κατά ένα στο χρονικό βήμα t που μεγιστοποιεί την ποσότητα $(g(n_t + 1) - g(n_t))/\texttt{Charge}(t)$, το τρέχον σύνολο $\langle n_t : 0 \le t \le d-1 \rangle$ είναι η βέλτιστη λύση για το υποπρόβλημα μεγέθους $W = \sum_{t=0}^{d-1} g(n_t)\tau_{\text{opt}}$.

Για την βάση $\langle n_t = 0 : 0 \le t \le d-1 \rangle$ της επαγωγής, ο ισχυρισμός είναι ξεκάθαρα αληθής. Στη συνέχεια, υποθέτουμε ότι ο αλγόριθμος έχει φτάσει σε ένα σύνολο $\langle n_t \ge 0 \rangle$, και ότι $V = \sum_{t=0}^{d-1} n_t \texttt{Charge}(t)$ είναι το ελάχιστο δυνατό κόστος για το πρόβλημα μεγέθους $W = \sum_{t=0}^{d-1} g(n_t)\tau_{\text{opt}}$. Χωρίς βλάβη της γενικότητας, υποθέτουμε ότι στο επόμενο βήμα ο αλγόριθμος επιλέγει το n_0 και το αυξάνει σε $n_0 + 1$. Τώρα με

$$W' = [g(n_0 + 1) + g(n_1) + \cdots + g(n_{d-1})]\tau_{\text{opt}}$$

έχουμε νέο κόστος

$$\begin{aligned} V' &= (n_0+1)\texttt{Charge}(0) + n_1\texttt{Charge}(1) + \cdots + n_{d-1}\texttt{Charge}(d-1) \\ &= V + \texttt{Charge}(0) \end{aligned}$$

Εφόσον το n_0 επιλέγεται στο τρέχον χρονικό βήμα, θα έχουμε $\forall t$:

$$g(n_t+1) - g(n_t) \leq \frac{\texttt{Charge}(t)}{\texttt{Charge}(0)}[g(n_0+1) - g(n_0)]$$

Ας υποθέσουμε ότι υπάρχει ένα άλλο σύνολο $\langle n'_t \geq 0 \rangle$ τέτοιο, ώστε $\sum_{t=0}^{d-1} g(n'_t)\tau_{\text{opt}} \geq W'$. Ορίζουμε $\Delta n_t = n'_t - n_t$. Αρκεί απλώς να δείξουμε ότι $\sum_{t=0}^{d-1} n'_t\texttt{Charge}(t) \geq V'$, ή ισοδύναμα $\sum_{t=0}^{d-1} \Delta n_t\texttt{Charge}(t) \geq \texttt{Charge}(0)$.

Εφόσον η συνάρτηση επιτάχυνσης $g(n_t)$ είναι κοίλη, θα έχουμε:

$$g(n_t + \Delta n_t) - g(n_t) \leq \Delta n_t[g(n_t+1) - g(n_t)]$$

Θα είναι:

$$\begin{aligned} g(n_0+1) - g(n_0) &= (W' - W)/\tau_{\text{opt}} \\ &\leq \sum_{t=0}^{d-1} g(n_t + \Delta n_t) - g(n_t) \\ &\leq \sum_{t=0}^{d-1} \Delta n_t[g(n_t+1) - g(n_t)] \\ &\leq \sum_{t=0}^{d-1} \Delta n_t \frac{\texttt{Charge}(t)}{\texttt{Charge}(0)}[g(n_0+1) - g(n_0)] \end{aligned}$$

Εφόσον $g(n_0+1) - g(n_0) > 0$, θα έχουμε $\sum_{t=0}^{d-1} \Delta n_t\texttt{Charge}(t) \geq \texttt{Charge}(0)$. ■

Ένας Αλγόριθμος Δυναμικού Προγραμματισμού

Σε γενικές γραμμές, οι επιλογές του μεγέθους της συστάδας με την πάροδο του χρόνου δεν είναι ασυσχέτιστες και η άπληστη στρατηγική δεν είναι βέλτιστη. Αντ' αυτού, λύνουμε το πρόβλημα με τη χρήση δυναμικού προγραμματισμού. Έστω $C(t, \varpi, \langle m_{t-\kappa}, \cdots, m_{t-1}\rangle)$ το ελάχιστο κόστος ολοκλήρωσης ϖ εναπομένοντος έργου ξεκινώντας από τη χρονική στιγμή t, όταν ο αριθμός των τελευταίων μηχανημάτων που χρεώθηκαν στην έναρξη των προηγούμενων κ χρονικών βημάτων δίνεται από την ακολουθία $\langle m_{t-\kappa}, \cdots, m_{t-1}\rangle$. Σημειώστε ότι για τα $m_{t-\kappa}$ μηχανήματα, ξεκινά μια νέα περίοδος χρέωσης στο χρονικό βήμα t. Δεν υπάρχει λόγος να μοντελοποιήσουμε τα m_i για $i < t-\kappa$, καθώς τα μηχανήματα που χρεώθηκαν πριν από το χρονικό βήμα $t-\kappa$ είτε έχουν αποδεσμευτεί είτε επαναχρεώθηκαν στη διάρκεια $[t-\kappa, t-1]$.

Έχουμε την ακόλουθη συνάρτηση μετάβασης κατάστασης δυναμικού προγραμματισμού (με $m_i = 0$ όταν $i < 0$ χάριν ευκολίας συμβολισμού):

$$C(t, \varpi, \langle m_{t-\kappa}, \cdots, m_{t-1}\rangle) =$$
$$\min_{m_t \geq 0} \left\{ m_t \texttt{Charge}(t) + C(t+1, \texttt{Progress}(\varpi, n_{t-1}, n_t), \langle m_{t-\kappa+1}, \cdots, m_{t-1}, m_t\rangle) \right\}$$

όπου $n_{t-1} = \sum_{i=t-\kappa}^{t-1} m_i$ και $n_t = \sum_{i=t-\kappa+1}^{t} m_i$. Θυμηθείτε ότι η διάσταση του χρόνου t διακριτοποιείται σε d βήματα, και ότι ο μέγιστος αριθμός μηχανημάτων που χρησιμοποιούνται ταυτόχρονα στην συστάδα είναι N. Διακριτοποιούμε επίσης και την διάσταση του εναπομένοντος έργου ϖ. Όσο πιο λεπτομερής είναι η διακριτοποίηση αυτής της διάστασης, τόσο πιο κοστοβόρα προκύπτει και η λύση του δυναμικού προγραμματισμού αλλά και τόσο πιο καλή η ποιότητα της λύσης. Ο αλγόριθμος δυναμικού προγραμματισμού εργάζεται προς τα πίσω στον χρόνο, ξεκινώντας με $C(d, 0, \langle m_{d-\kappa}, \cdots, m_{d-1}\rangle) = 0$ για όλους τους συνδυασμούς $m_{d-\kappa}, \cdots, m_{d-1}$ τέτοιους, ώστε $0 \leq m_{d-\kappa} + \cdots + m_{d-1} \leq N$. Η χρονική πολυπλοκότητα του αλγορίθμου είναι $O(dWN\binom{N+\kappa}{\kappa})$, ή απλούστερα $O(dWN^{\kappa+1})$, όπου το W διακριτοποιείται ως ακέραιος, όπως αναφέρθηκε προηγουμένως.

Στα πλαίσια της συγκεκριμένης διπλωματικής εργασίας, πραγματοποιήσαμε προσομοιώσεις για την δέσμευση πόρων με χρήση του αλγορίθμου Δυναμικού Προγραμματισμού, καθώς ο Άπληστος Αλγόριθμος κάνει κάποιες μη ρεαλιστικές παραδοχές, όπως ότι οι επιλογές του μεγέθους της συστάδας με την πάροδο του χρόνου είναι ασυσχέτιστες, και επίσης θέλαμε να εξετάσουμε την επίδραση του παράγοντα $\texttt{Overhead}(\varpi, n, n')$.

5.2 Πρόβλεψη Χρονοσειρών

Στη ενότητα αυτή θα αναλύσουμε τη μέθοδο που ακολουθήσαμε για την πρόβλεψη των δύο άγνωστων παραμέτρων του προβλήματος της ελαχιστοποίησης του κόστους εκτέλεσης μιας ροής εργασιών εντός μιας δοθείσης προθεσμίας: τον *φόρτο εργασίας* και το *Spot Price*. Η προσέγγισή μας χρησιμοποιεί νευρωνικά δίκτυα τύπου LSTM, η λειτουργία των οποίων αναλύθηκε στην ενότητα 4.5.2. Χρησιμοποιήσαμε την αρχιτεκτονική LSTM ενός κρυφού επιπέδου (`vanillaLSTM`). Η αρχιτεκτονική αυτή εκπαιδεύεται στην πρόβλεψη χρονοσειρών. Συγκεκριμένα, για το πρόβλημά μας, οι χρονοσειρές αυτές περιγράφουν ποσοστά χρήσης CPU (CPU utilization) και Spot Prices. Με άλλα λόγια, τα δίκτυα αυτά δέχονται ως είσοδο μια χρονοσειρά και στην έξοδο δίνουν μια πρόβλεψη της επόμενης τιμής της χρονοσειράς αυτής. Πιο απλά, χρησιμοποιούμε τα δίκτυα LSTM για την επίλυση ενός προβλήματος παλινδρόμησης (regression).

Όπως αναφέρθηκε προηγουμένως, για την εκπαίδευση των νευρωνικών μας δικτύων χρησιμοποιήσαμε χρονοσειρές από μετρήσεις ποσοστών χρήσης CPU και Spot Price. Οι χρονοσειρές αυτές έχουν την μορφή ακολουθιών, τις οποίες εδώ θα συμβολίσουμε ως διανύσματα

$$[u_1, u_2, \cdots, u_{i-1}, u_i, u_{i+1}, \cdots, u_n]$$

Θα πρέπει λοιπόν από αυτές τις χρονοσειρές να κατασκευάσουμε δεδομένα εκπαίδευσης, προκειμένου να τροφοδοτήσουμε τα νευρωνικά μας δίκτυα με αυτά. Ο τρόπος με τον οποίο πραγματοποιήθηκε αυτή η μετατροπή έχει ως εξής: Επιλέγουμε έναν ακέραιο $k < n$ και κατασκευάζουμε δείγματα εκπαίδευσης της μορφής:

$$\mathbf{x}_i = [u_{i-k}, u_{i-k+1}, \cdots, u_{i-1}],\ y_i = u_i,\ \text{για } k < i \leq n$$

Παρουσιάζουμε παρακάτω ένα παράδειγμα μιας τέτοιας μετατροπής:

$$[10, 20, 30, 40, 50, 60, 70, 80, 90] \xrightarrow{k=3} \underbrace{\begin{bmatrix} 10 & 20 & 30 \\ 20 & 30 & 40 \\ 30 & 40 & 50 \\ 40 & 50 & 60 \\ 50 & 60 & 70 \\ 60 & 70 & 80 \end{bmatrix}}_{\mathbf{X}}, \underbrace{\begin{bmatrix} 40 \\ 50 \\ 60 \\ 70 \\ 80 \\ 90 \end{bmatrix}}_{\mathbf{y}}$$

Η παράμετρος k συγκαταλέγεται στο σύνολο των παραμέτρων για την επίλυση του προβλήματος πρόβλεψης χρονοσειρών.

Τα δύο νευρωνικά δίκτυα που κατασκευάσαμε στο πλαίσιο της συγκεκριμένης διπλωματικής εργασίας, παρέχουν προβλέψεις για το ποσοστό χρήσης CPU και το Spot Price αντίστοιχα.

Θα παρατηρήσατε ότι αναφερόμαστε σε προβλέψεις ποσοστών χρήσης CPU. Το εύλογο ερώτημα που γεννιέται εδώ είναι το εξής: πώς από μια πρόβλεψη του CPU utilization περνάμε σε μια ροή εργασιών, όπως αυτή ορίστηκε στην προηγούμενη ενότητα; Για να απαντήσουμε στο ερώτημα αυτό, κάνουμε την εξής παραδοχή: όλες οι εργασίες που τρέχουν απαιτούν σταθερό ποσοστό CPU για την εκτέλεσή τους και το συνολικό τους έργο w είναι επίσης το ίδιο για όλες. Εισάγουμε λοιπόν δύο ακόμη παραμέτρους στο πρόβλημά μας, c_0 και w_0, δηλαδή το ποσοστό χρήσης CPU για κάθε εργασία και το συνολικό έργο κάθε εργασίας (θυμίζουμε ότι η μονάδα μέτρησης του w_0 είναι το έργο που εκτελείται από ένα μόνο μηχάνημα στη μονάδα χρόνου), αντίστοιχα. Για παράδειγμα, αν κάποια χρονική στιγμή η πρόβλεψη του νευρωνικού δικτύου για το ποσοστό χρήσης CPU είναι c_t, τότε η προς εκτέλεση ροή εργασιών αποτελείται από $J = \lceil c_t/c_0 \rceil$ εργασίες, όπου $w^{(j)} = w_0,\ \forall\ j \in \{1, 2, \cdots, J\}$. Η προθεσμία για την εκτέλεση αυτής της ροής εργασιών είναι ο χρόνος που μεσολαβεί για την επόμενη μέτρηση του CPU utilization, c_{t+1} (ο οποίος όπως θα δούμε παρακάτω είναι γνωστός). Παρατηρούμε ότι η μετατροπή αυτή διατηρεί την κατανομή του ποσοστού CPU utilization, δηλαδή υψηλότερο ποσοστό CPU utilization οδηγεί σε μεγαλύτερο πλήθος εργασιών και μεγαλύτερο συνολικό έργο W, και αντιστρόφως, μια ροή εργασιών με περισσότερες στο πλήθος και πιο διαρκείς εργασίες συνεπάγεται μεγαλύτερο CPU utilization.

Κεφάλαιο 6

Πειραματική Αξιολόγηση

6.1 Σύνολα Δεδομένων

Στην ενότητα αυτή θα πραγματοποιηθεί μια παρουσίαση των συνόλων των δεδομένων εκπαίδευσης που χρησιμοποιήσαμε για την εκπαίδευση των νευρωνικών μας δικτύων. Χρησιμοποιήσαμε δύο διαφορετικά σύνολα δεδομένων εκπαίδευσης, ένα για κάθε ένα εκ των νευρωνικών στα οποία έγινε αναφορά στο προηγούμενο κεφάλαιο. Θα παρουσιάσουμε τη δομή των δεδομένων εκπαίδευσης και θα αναλύσουμε τα στάδια προεπεξεργασίας που ακολουθήσαμε προκειμένου να μετατρέψουμε τα δεδομένα αυτά σε μορφή αξιοποιήσιμη για τα νευρωνικά μας δίκτυα.

6.1.1 Φόρτος Εργασιών

Για το φόρτο εργασιών, χρησιμοποιήσαμε το σύνολο δεδομένων `AzurePublicDatasetV2` [4], το οποίο περιέχει δεδομένα για το φόρτο εργασιών των εικονικών μηχανών τύπου first-party της Azure σε μια γεωγραφική περιοχή για μια χρονική περίοδο διάρκειας τριάντα ημερών. Το συγκεκριμένο trace file αποτελείται από πολλαπλά datasets και κάθε dataset αποτελείται από ένα σχεσιακό πίνακα, οποίος περιέχει ένα primary key. Πριν προχωρήσουμε στην περιγραφή των πινάκων, θα πρέπει να τονίσουμε πως τα περισσότερα δεδομένα είναι κρυπτογραφημένα ή παρουσιάζονται κανονικοποιημένα για λόγους προσωπικών δικαιωμάτων. Οι πίνακες οι οποίοι χρησιμοποιήθηκαν είναι οι ακόλουθοι:

- `vmtable`
- `vm_cpu_readings`

Ο πίνακας `vmtable` περιέχει στοιχεία για κάθε εικονική μηχανή που δημιουργήθηκε στην γεωγραφική αυτή περιοχή για το χρονικό διάστημα στο οποίο πραγματοιήθηκαν οι μετρήσεις. Η μορφή του έχει ως εξής:

vmtable
vm id 🔑
subscription id
deployment id
timestamp vm created
timestamp vm deleted
max cpu
avg cpu
p95 max cpu
vm category
vm virtual core count bucket
vm memory (gb) bucket

Πίνακας 6.1: Ο πίνακας `vmtable`

Ο πίνακας `vm_cpu_readings` περιέχει μετρήσεις για τα ποσοστά χρήσης CPU κάθε εικονικής μηχανής. Οι μετρήσεις αυτές είναι κανονικοποιημένες, με μέγιστη τιμή το 1 (100%) και ελάχιστη τιμή το 0 (0%). Μια μέτρηση πραγματοποιείται κάθε πέντε λεπτά. Η μορφή του πίνακα έχει ως εξής:

vm_cpu_readings
timestamp 🔑
vm id
min cpu
max cpu
avg cpu

Πίνακας 6.2: Ο πίνακας `vm_cpu_readings`

Παρουσιάζουμε τώρα την προεπεξεργασία που ακολουθήσαμε για να μετατρέψουμε τα δεδομένα από τους δύο αυτούς πίνακες σε αξιοποιήσιμη μορφή. Οι εν λόγω εικονικές μηχανές δεσμεύονται από διάφορους χρήστες οι οποίοι χρησιμοποιούν την υπηρεσία της Azure στο συγκεκριμένο χρονικό διάστημα. Κάθε ένας από τους χρήστες διακρίνεται από ένα μοναδικό `subscription id` (εγγραφή του πίνακα `vmtable`). Κατά τη διάρκεια χρήσης της υπηρεσίας, κάθε χρήστης δημιουργεί ένα σύνολο από παρατάξεις εικονικών μηχανών, προκειμένου να ομαδοποιήσει και να διαχειριστεί λογικά συνδεδεμένες εικονικές μηχανές εντός μια συνδρομής (subscription). Κάθε μια από τις παρατάξεις αυτές διακρίνεται από ένα μοναδικό `deployment id` (εγγραφή του πίνακα `vmtable`). Στα πλαίσια της συγκεκριμένης διπλωματικής εργασίας, θεωρήσαμε πως κάθε ένα απο αυτά τα deployments αποτελεί και μια υπολογιστική συστάδα, όπως αυτή ορίστηκε στο προηγούμενο κεφάλαιο.

Από όλους αυτούς τους χρήστες, εμείς απομονώσαμε εκείνον που είχε τις περισσότερες εικονικές μηχανές στην κυριότητά του για το χρονικό διάστημα στο οποίο πραγματοιήθηκαν οι μετρήσεις. Βρήκαμε δηλαδή την εγγραφή `subscription id` η οποία εμφανίζεται τις περισσότερες φορές στον πίνακα `vmtable`. Στη συνέχεια, για τον συγκεκριμένο χρήστη, ομαδοποιήσαμε τις εικονικές μηχανές του με βάση τις διαφορετικές εγγραφές `deployment id`. Για να γίνει λίγο πιο ξεκάθαρη αυτή η διαδικασία, παρουσιάζουμε παρακάτω ένα παράδειγμα:

vm id	subscription id	deployment id	···
1	1	1	···
2	1	1	···
3	1	2	···
4	2	4	···
5	2	4	···
6	3	5	···
7	3	5	···
8	1	2	···
9	1	3	···
10	1	3	···

vm id	subscription id	deployment id	···
1	1	1	···
2	1	1	···
3	1	2	···
8	1	2	···
9	1	3	···
10	1	3	···

vm id	subscription id	deployment id	···
1	1	1	···
2	1	1	···

vm id	subscription id	deployment id	···
3	1	2	···
8	1	2	···

vm id	subscription id	deployment id	···
9	1	3	···
10	1	3	···

Έστω $\mathcal{D}_i$ το σύνολο των εικονικών μηχανών στην κυριότητα του εν λόγω χρήστη που ανήκουν στην παράταξη με `deployment id` ίσο με i. Για κάθε ένα από τα σύνολα αυτά, διατρέξαμε τον πίνακα `vm_cpu_readings` και κρατήσαμε μόνο τις εγγραφές για τις οποίες ισχύει `vm id` $\in \mathcal{D}_i$. Με άλλα λόγια, απομονώσαμε τις μετρήσεις για τα ποσοστά χρήσης CPU των εικονικών μηχανών που ανήκουν στην παράταξη $\mathcal{D}_i$. Στη συνέχεια, από αυτές τις μετρήσεις, κατασκευάσαμε και από έναν πίνακα `deployment_cpu_readings` για κάθε παράταξη $\mathcal{D}_i$, ο οποίος έχει την παρακάτω μορφή:

deployment_cpu_readings
timestamp 🔑
cluster size
avg min cpu
avg max cpu
avg mean cpu
max cpu
total memory
avg memory

Πίνακας 6.3: Ο πίνακας `deployment_cpu_readings`

Οι στήλες του παραπάνω πίνακα προκύπτουν ως εξής:

- `cluster size`: Ο αριθμός των εικονικών μηχανών που ανήκουν στο $\mathcal{D}_i$ και ικανοποιούν την ιδιότητα `timestamp vm created` $\leq$ `timestamp` $\leq$ `timestamp vm deleted`
- `avg min cpu`: Ο μέσος όρος των εγγραφών `min cpu` για τις εικονικές μηχανές που ανήκουν στο $\mathcal{D}_i$
- `avg max cpu`: Ο μέσος όρος των εγγραφών `max cpu` για τις εικονικές μηχανές που ανήκουν στο $\mathcal{D}_i$
- `avg mean cpu`: Ο μέσος όρος των εγγραφών `avg cpu` για τις εικονικές μηχανές που ανήκουν στο $\mathcal{D}_i$
- `max cpu`: Η μέγιστη τιμή των εγγραφών `max cpu` για τις εικονικές μηχανές που ανήκουν στο $\mathcal{D}_i$
- `total memory`: Το άθροισμα των εγγραφών `vm memory (gb) bucket` για τις εικονικές μηχανές που ανήκουν στο $\mathcal{D}_i$
- `avg memory`: Ο μέσος όρος των εγγραφών `vm memory (gb) bucket` για τις εικονικές μηχανές που ανήκουν στο $\mathcal{D}_i$

Στα πλαίσια της συγκεκριμένης διπλωματικής εργασίας ασχοληθήκαμε με την πρόβλεψη της χρονοσειράς που προκύπτει από την εγγραφή `avg mean cpu` του πίνακα `deployment_cpu_readings`, από την οποία κατασκευάσαμε δεδομένα εκπαίδευσης, όπως περιγράφεται στην ενότητα 5.2.

6.1.2 Spot Price

Για το Spot Price, χρησιμοποιήσαμε ιστορικά δεδομένα από την υπηρεσία AWS της Amazon. Τα δεδομένα που χρησιμοποιήσαμε αφορούν την εικονική μηχανή Amazon EC2 Instance Type `c5.2xlarge`, η οποία έχει τα παρακάτω χαρακτηριστικά:

Χαρακτηριστικό	**Τιμή**
Πλήθος Πυρήνων	8
Αρχιτεκτονική	x86_64
Μνήμη RAM	16 GiB
Απόδοση Δικτύου	Εως και 10 Gigabit

Πίνακας 6.4: Χαρακτηριστικά Amazon EC2 Instance Type `c5.2xlarge`

Τα δεδομένα αυτά αντιστοιχούν στη χρονική περίοδο από `2020-12-28T16:38:08+00:00` εως `2021-03-27T17:55:24+00:00`, στη ζώνη `us.east.2a`, με περιγραφή προϊόντος `Linux/UNIX`. Για να λάβουμε τα δεδομένα αυτά, χρησιμοποιήσαμε το εργαλείο CloudShell της υπηρεσίας και την εντολή `describe-spot-price-history`. Δίνεται παρακάτω ένα παράδειγμα χρήσης της εντολής αυτής:

```
$ aws ec2 describe-spot-price-history --instance-types c5.2xlarge --availability-zone
us-east-2a --product-descriptions Linux/UNIX --start-time 2020-12-28T16:38:08+00:00 --end-time
2021-03-27T17:55:24+00:00
```

Τα δεδομένα που επιστρέφει η εντολή αυτή είναι σε μορφότυπο JSON και έχουν την ακόλουθη μορφή:

```
{
  "SpotPriceHistory": [
    {
      "AvailabilityZone": "us-east-2a",
      "InstanceType": "c5.2xlarge",
      "ProductDescription": "Linux/UNIX",
      "SpotPrice": "0.098600",
      "Timestamp": "2021-03-27T17:55:24+00:00"
    },
    {
      "AvailabilityZone": "us-east-2a",
      "InstanceType": "c5.2xlarge",
      "ProductDescription": "Linux/UNIX",
      "SpotPrice": "0.097900",
      "Timestamp": "2021-03-27T12:12:24+00:00"
    },
    ...
  ]
}
```

Στη συνέχεια μετατρέψαμε τα δεδομένα αυτά σε έναν σχεσιακό πίνακα με την παρακάτω μορφή:

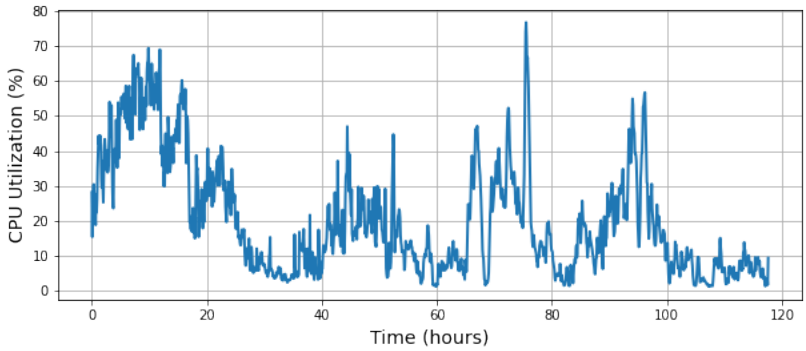
spot_price
timestamp 🔑
spot price

Πίνακας 6.5: Ο πίνακας `spot_price`

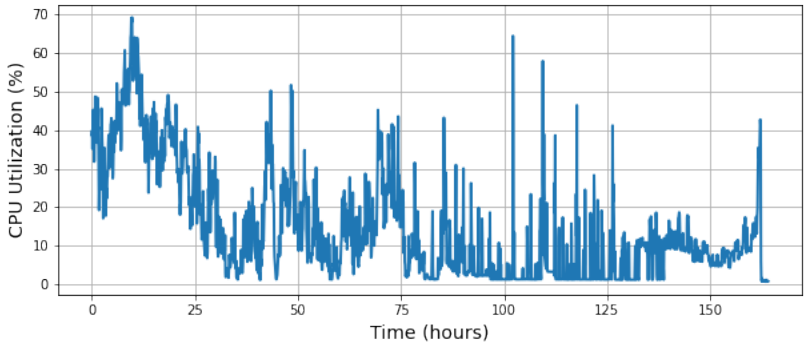
Οι μετρήσεις του Spot Price που λαμβάνουμε, δεν απέχουν μεταξύ τους κάποιο σταθερό χρονικό διάστημα. Προκειμένου να λύσουμε το πρόβλημα αυτό, πραγματοποιήσαμε το μοναδικό στάδιο προεπεξεργασίας στα δεδομένα αυτά, το οποίο αφορά μια παλινδρόμιση, ωστέ να έχουμε μια μέτρηση του Spot Price **κάθε λεπτό**. Στη συνέχεια, χρησιμοποιήσαμε την χρονοσειρά που προκύπτει από αυτή την παλινδρόμιση για να κατασκευάσουμε δεδομένα εκπαίδευσης, όπως περιγράφεται στην ενότητα 5.2.

6.2 Αποτελέσματα Πρόβλεψης Φόρτου Εργασιών

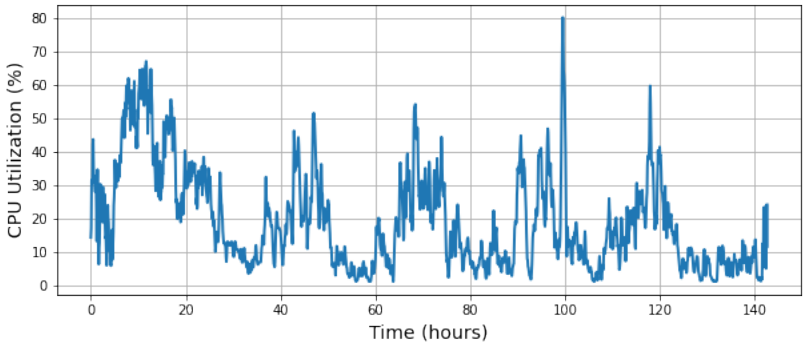
Στην ενότητα αυτή θα παραθέσουμε τα αποτελέσματα των προβλέψεων του LSTM νευρωνικού δικτύου για την χρονοσειρά `avg mean cpu`, η οποία όπως αναφέραμε προηγουμένως, περιγράφει τη μέση τιμή του ποσοστού χρήσης CPU μιας παράταξης (deployment). Από τις παρατάξεις του χρήστη που απομονώσαμε, επιλέξαμε για την εκπαίδευση του νευρωνικού δικτύου εκείνες που είχαν διάρκεια τουλάχιστον τεσσάρων ημερών και παρουσίαζαν την μεγαλύτερη μεταβλητότητα. Με άλλα λόγια, επιλέξαμε τις παρατάξεις οι οποίες θα κάνουν πιο δύσκολο το έργο του νευρωνικού για εκπαίδευση και πρόβλεψη. Αυτές είναι οι παρατάξεις 0, 3, 4, 10 και 12. Παρουσιάζουμε παρακάτω τις γραφικές παραστάσεις του `avg mean cpu` για όλες τις παρατάξεις που επιλέξαμε για εκπαίδευση:



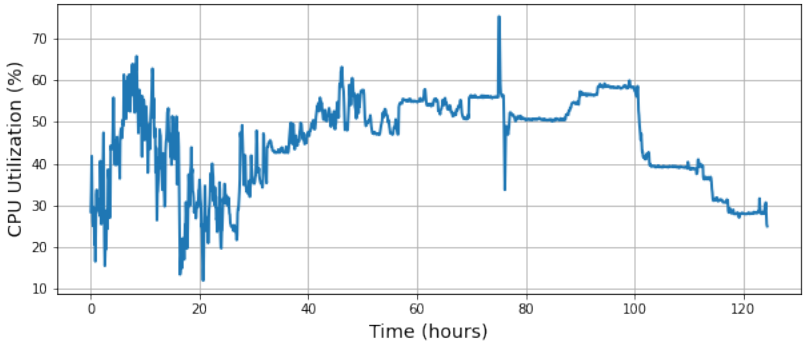
Εικόνα 6.1: Η γραφική παράσταση του `avg mean cpu` για το deployment 0



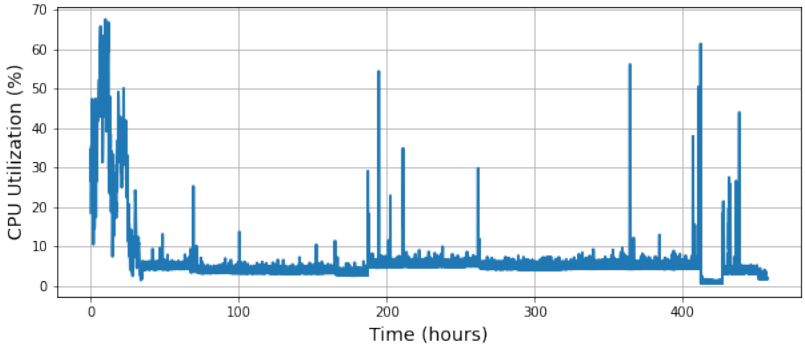
Εικόνα 6.2: Η γραφική παράσταση του `avg mean cpu` για το deployment 3



Εικόνα 6.3: Η γραφική παράσταση του `avg mean cpu` για το deployment 4



Εικόνα 6.4: Η γραφική παράσταση του `avg mean cpu` για το deployment 10



Εικόνα 6.5: Η γραφική παράσταση του `avg mean cpu` για το deployment 12

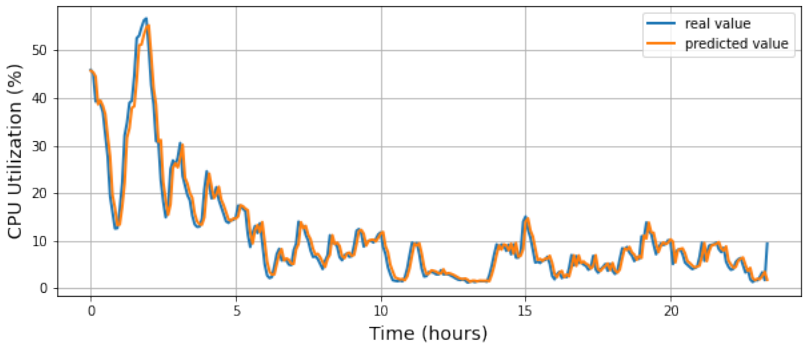
Η αναλογία train/test δεδομένων ορίστηκε ως 80%/20%. Οι παράμετροι με τις οποίες πειραματιστήκαμε είναι ο αριθμός n των LSTM μονάδων στο κρυφό επίπεδο του νευρωνικού δικτύου και ο αριθμός k των παρελθοντικών τιμών της χρονοσειράς για την κατασκευή δεδομένων εκπαίδευσης που παρουσιάστηκε στην ενότητα 5.2. Ως συνάρτηση ενεργοποίησης χρησιμοποιήσαμε την ReLU, ενώ ως αλγόριθμο βελτιστοποίησης επιλέχθηκε ο Adam. Ο αριθμός των εποχών εκπαίδευσης ορίστηκε στις 210. Η μετρική που χρησιμοποιήσαμε είναι το μέσο τετραγωνικό σφάλμα (mean squared error). Παρουσιάζουμε στον παρακάτω πίνακα τα αποτελέσματα για το μέσο τετραγωνικό σφάλμα πρόβλεψης για τις διάφορες τιμές που επιλέχθηκαν για τις παραμέτρους n, k για όλες τις επιλεγμένες παρατάξεις:

αριθμός παρελθοντικών βημάτων (k)	**αριθμός LSTM μονάδων** (n)	**deployment**				
		0	3	4	10	12
10	20	0.000684	0.000950	0.001428	0.000146	0.001308
	30	0.000700	0.001221	0.001637	0.000137	0.001301
	40	0.000609	0.001154	0.001587	0.000080	0.001318
	50	0.000585	0.000986	0.001585	0.000068	0.001371
	60	0.000576	**0.000927**	0.001559	0.000084	0.001292
20	20	0.000662	0.001078	0.001390	0.000055	0.001309
	30	0.000549	0.001147	0.001369	0.000056	0.001308
	40	0.000672	0.001035	**0.001259**	0.000058	**0.001288**
	50	0.000602	0.001010	0.001343	0.000064	0.001402
	60	0.000551	0.001127	0.001469	0.000165	0.001354
30	20	0.000621	0.001061	0.001448	0.000058	0.001651
	30	**0.000547**	0.001055	0.001436	**0.000047**	0.001505
	40	0.000602	0.000966	0.001478	0.000050	0.001721
	50	0.000581	0.001041	0.001329	0.000048	0.001604
	60	0.000696	0.001104	0.001447	0.000067	0.001628

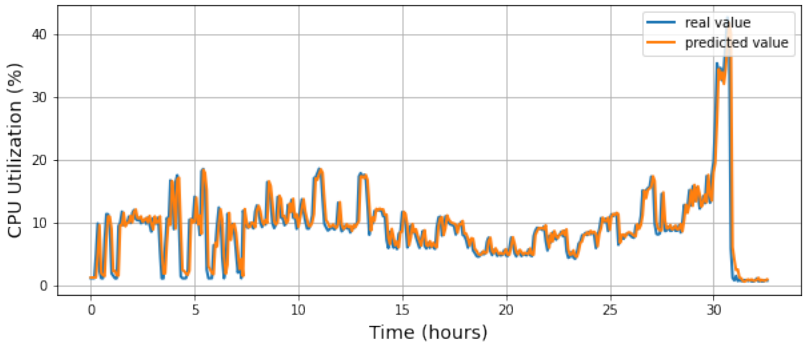
Πίνακας 6.6: Μέσο τετραγωνικό σφάλμα πρόβλεψης για τις διάφορες τιμές των παραμέτρων n, k

Σχολιασμός: Παρατηρούμε πως όλες οι προσεγγίσεις δίνουν ένα ικανοποιητικά μικρό μέσο τετραγωνικό σφάλμα. Για παράδειγμα, το μέγιστο σφάλμα που παρατηρήθηκε (deployment 12, $k = 30$, $n = 20$) είναι ίσο με 0.001651, το οποίο αντιστοιχεί σε σφάλμα $\pm 4.06\%$ του ποσοστού χρήσης CPU. Επίσης, η αύξηση του αριθμού των LSTM μονάδων στο κρυφό επίπεδο του νευρωνικού δικτύου στα περισσότερα deployments, επιφέρει μια μείωση στο σφάλμα, αλλά από ένα σημείο η περαιτέρω αύξηση του n οδηγεί σε αύξηση του σφάλματος, χωρίς αυτή η αύξηση να φαίνεται σημαντικά μεγάλη. Τέλος, η αύξηση του αριθμού των παρελθοντικών βημάτων k, με σταθερή την παράμετρο n, φαίνεται να οδηγεί σε μια ελαφρά μείωση του σφάλματος στα περισσότερα deployments, το οποίο είναι λογικό, καθώς το νευρωνικό επεξεργάζεται περισσότερες τιμές του παρελθόντος για να πραγματοποιήσει μια πρόβλεψη.

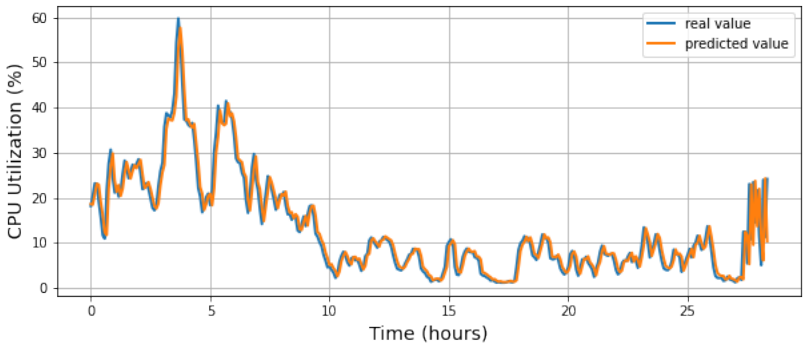
Παραθέτουμε παρακάτω τις γραφικές παραστάσεις των προβλέψεων του νευρωνικού σε σχέση με την πραγματική χρονοσειρά του `avg mean cpu` για όλες τις παρατάξεις, με $n = 40$ και $k = 10$:



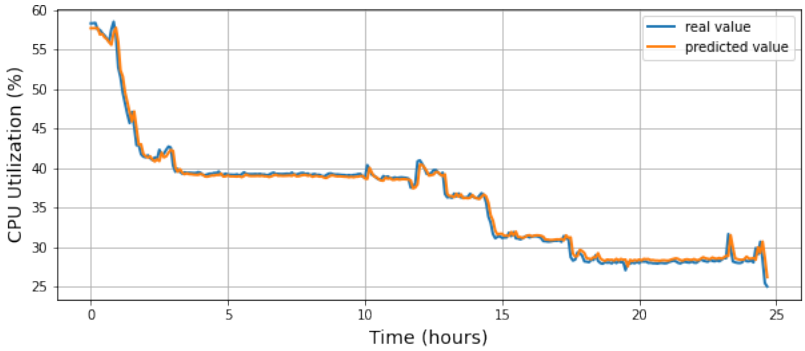
Εικόνα 6.6: Πρόβλεψη του `avg mean cpu` για το deployment 0



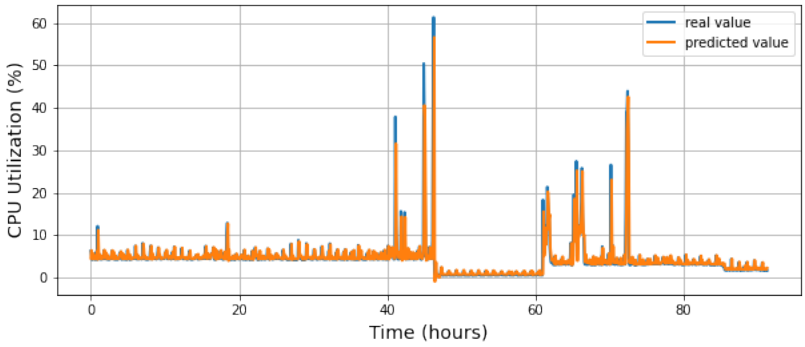
Εικόνα 6.7: Πρόβλεψη του `avg mean cpu` για το deployment 3



Εικόνα 6.8: Πρόβλεψη του `avg mean cpu` για το deployment 4



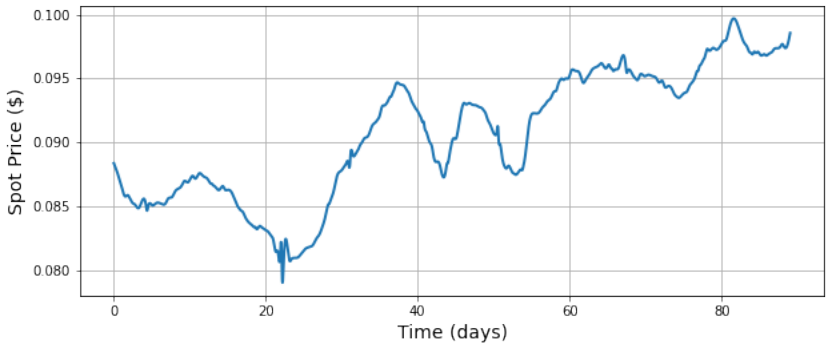
Εικόνα 6.9: Πρόβλεψη του `avg mean cpu` για το deployment 10



Εικόνα 6.10: Πρόβλεψη του `avg mean cpu` για το deployment 12

6.3 Αποτελέσματα Πρόβλεψης Spot Price

Στην ενότητα αυτή θα παραθέσουμε τα αποτελέσματα των προβλέψεων του LSTM νευρωνικού δικτύου για την χρονοσειρά του Spot Price. Παρουσιάζουμε παρακάτω τη γραφική παράσταση του Spot Price για το μηχάνημα `c5.2xlarge` για χρονική την περίοδο από `2020-12-28T16:38:08+00:00` εως `2021-03-27T17:55:24+00:00` στην ζώνη `us.east.2a` με περιγραφή προϊόντος `Linux/UNIX`:



Εικόνα 6.11: Η γραφική παράσταση του Spot Price

Όπως και στην πρόβλεψη του φόρτου εργασιών, η αναλογία train/test δεδομένων ορίστηκε ως 80%/20%. Οι παράμετροι με τις οποίες πειραματιστήκαμε είναι ο αριθμός n των LSTM μονάδων στο κρυφό επίπεδο του νευρωνικού δικτύου και ο αριθμός k των παρελθοντικών τιμών της χρονοσειράς για την κατασκευή δεδομένων εκπαίδευσης που παρουσιάστηκε στην ενότητα 5.2. Ως συνάρτηση ενεργοποίησης χρησιμοποιήσαμε την ReLU, ενώ ως αλγόριθμο βελτιστοποίησης επιλέχθηκε ο Adam. Ο αριθμός των εποχών εκπαίδευσης ορίστηκε στις 210. Η μετρική που χρησιμοποιήσαμε είναι το μέσο τετραγωνικό σφάλμα (mean squared error). Παρουσιάζουμε στον παρακάτω πίνακα τα αποτελέσματα για το μέσο τετραγωνικό σφάλμα πρόβλεψης για τις διάφορες τιμές που επιλέχθηκαν για τις παραμέτρους n, k:

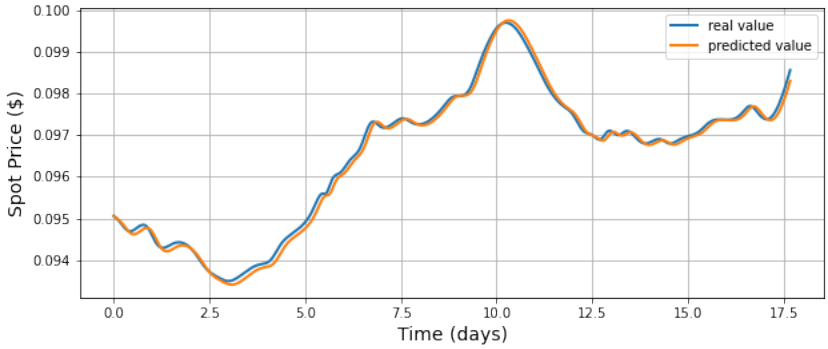
αριθμός παρελθοντικών βημάτων (k)	**αριθμός LSTM μονάδων** (n)	**μέσο τετραγωνικό σφάλμα**
5	20	0.00000425
	30	0.00000290
	40	0.00000276
	50	0.00000334
	60	0.00000539
10	20	0.00000444
	30	0.00000480
	40	0.00000355
	50	0.00000366

	60	0.00000343
15	20	**0.00000113**
	30	0.00000393
	40	0.00000163
	50	0.00000234
	60	0.00000297

Πίνακας 6.7: Μέσο τετραγωνικό σφάλμα πρόβλεψης για τις διάφορες τιμές των παραμέτρων n, k

Σχολιασμός: Παρατηρούμε πως όλες οι προσεγγίσεις δίνουν ένα ικανοποιητικά μικρό μέσο τετραγωνικό σφάλμα. Το μέγιστο σφάλμα που παρατηρήθηκε ($k = 5$, $n = 60$) είναι ίσο με 0.00000539, το οποίο αντιστοιχεί σε σφάλμα ± 0.0002 \$. Οι ελάχιστες τιμές του σφάλματος προκύπτουν για τη μέγιστη τιμή του k με την οποία πειραματιστήκαμε, δηλαδή $k = 15$, χωρις όμως αυτό να δείχνει ότι η περαιτέρω αύξηση του k θα μας δώσει καλύτερα αποτελέσματα, καθώς βλέπουμε πως η αύξηση του k από 5 σε 10 οδηγεί σε μεγαλύτερα σφάλματα. Παρατηρούμε επίσης ότι η αύξηση στον αριθμό των LSTM μονάδων στο κρυφό επίπεδο (n) δεν οδηγεί σε κάποια δραματική πτώση του σφάλματος καθώς παρατηρούμε παραπλήσιες τιμές του σφάλματος για τις διαφορετικές τιμές του n.

Παραθέτουμε παρακάτω τις γραφικές παραστάσεις των προβλέψεων του νευρωνικού σε σχέση με την πραγματική χρονοσειρά του Spot Price για την βέλτιστη προσέγγιση, με $n = 20$ και $k = 15$:



Εικόνα 6.12: Γραφική παράσταση της πρόβλεψης του Spot Price για $k = 15$ και $n = 20$.

6.4 Δέσμευση Πόρων με Χρήση Δυναμικού Προγραμματισμού

Σε αυτήν την ενότητα θα παραθέσουμε τα αποτελέσματα που προκύπτουν από τις προσομοιώσεις για την δέσμευση υπολογιστικών πόρων με χρήση του αλγορίθμου δυναμικού προγραμματισμού που περιγράψαμε στην ενότητα 5.1.7. Προτού γίνει αυτό, θα παρουσιάσουμε στον παρακάτω συγκεντρωτικό πίνακα τις απαραίτητες παραμέτρους για τις προσομοιώσεις μας και τις τιμές που τους δώσαμε:

Παράμετρος	**Περιγραφή**	**Τιμή**
τ_{charge}	Πόσο συχνά ο πάροχος χρεώνει τα μηχανήματα της συστάδας	$60sec$
τ_{opt}	Πόσο συχνά αποφασίζουμε για το μέγεθος της συστάδας	$20sec, 30sec, 60sec$
d	Η προθεσμία για την εκτέλεση της ροής εργασιών εκφρασμένη ως πολλαπλάσιο του τ_{opt}	$300sec/\tau_{\text{opt}}$
c_0	Ποσοστό χρήσης CPU για κάθε εργασία	0.1%
w_0	Συνολικό έργο κάθε εργασίας	4sec
$g(n)$	Συνάρτηση επιτάχυνσης για κάθε εργασία	$1 + \log_2(n)$
N	Μέγιστος αριθμός μηχανημάτων που χρησιμοποιούνται ταυτόχρονα στη συστάδα	10
$\texttt{Overhead}(\varpi, n, n')$	Υπολογιστική επιβάρυνση όταν το μέγεθος της συστάδας αλλάζει απο n σε n'	$\tau_{\text{opt}} \cdot \lvert n - n' \rvert / N$

Πίνακας 6.8: Παράμετροι προσομοίωσης

Θα μελετήσουμε πρώτα την περίπτωση όπου $\tau_{\text{opt}} = \tau_{\text{charge}} = 60sec$. Στην περίπτωση αυτή, η συνάρτηση μετάβασης κατάστασης δυναμικού προγραμματισμού που ορίσαμε στην ενότητα 5.1.7 απλοποιείται στην παρακάτω έκφραση:

$$C(t, \varpi, n_{t-1}) = \min_{n_t \geq 0} \left\{ n_t \texttt{Charge}(t) + C(t+1, \texttt{Progress}(\varpi, n_{t-1}, n_t), n_t) \right\}$$

Θυμίζουμε ότι η παραπάνω έκφραση αναπαριστά το ελάχιστο κόστος ολοκλήρωσης ϖ εναπομένοντος έργου ξεκινώντας από τη χρονική στιγμή t, όταν ο αριθμός των μηχανημάτων που χρεώθηκαν στην έναρξη του προηγούμενου χρονικού βήματος είναι n_{t-1}. Επιλέγουμε μια σχετικά ελαφριά ροή εργασιών που αντιστοιχεί σε 15.83% `avg mean cpu`, δηλαδή, σύμφωνα με την μετατροπή που περιγράψαμε στην ενότητα 5.2, μια ροή εργασιών που αποτελείται από $J = \lceil 15.83\%/c_0 \rceil = 159$ εργασίες με συνολικό έργο $W = 159 \cdot w_0 = 636sec$. Η προθεσμία είναι $d = 300sec/60sec = 5$ βήματα τ_{opt} και η ακολουθία του Spot Price που προκύπτει από την πρόβλεψη είναι

$$[0.097278, 0.097306, 0.097328, 0.097345, 0.097356]$$

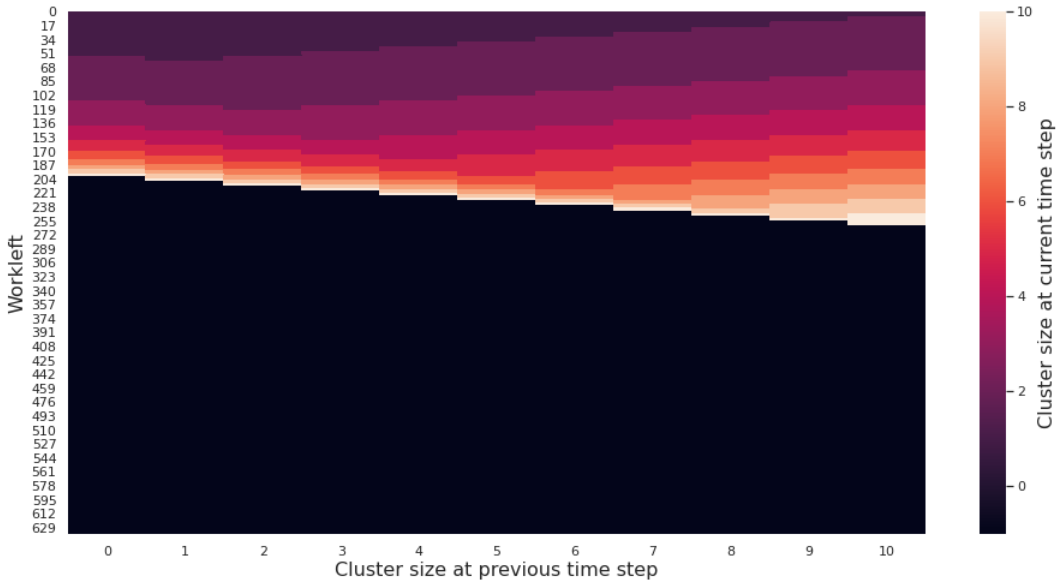
Από την εκτέλεση του αλγορίθμου προκύπτει πως το ελάχιστο χρηματικό κόστος για την εκτέλεση της ροής εργασιών είναι $C(0, W, 0) = 1.16781\$$ ενώ η βέλτιστη ακολουθία μεγεθών της συστάδας είναι

$$[3, 3, 2, 2, 2]$$

πράγμα που δικαιολογεί και την τιμή του ελαχίστου κόστους, καθώς:

$$3 \cdot 0.097278 + 3 \cdot 0.097306 + 2 \cdot 0.097328 + 2 \cdot 0.097345 + 2 \cdot 0.097356 = 1.16781$$

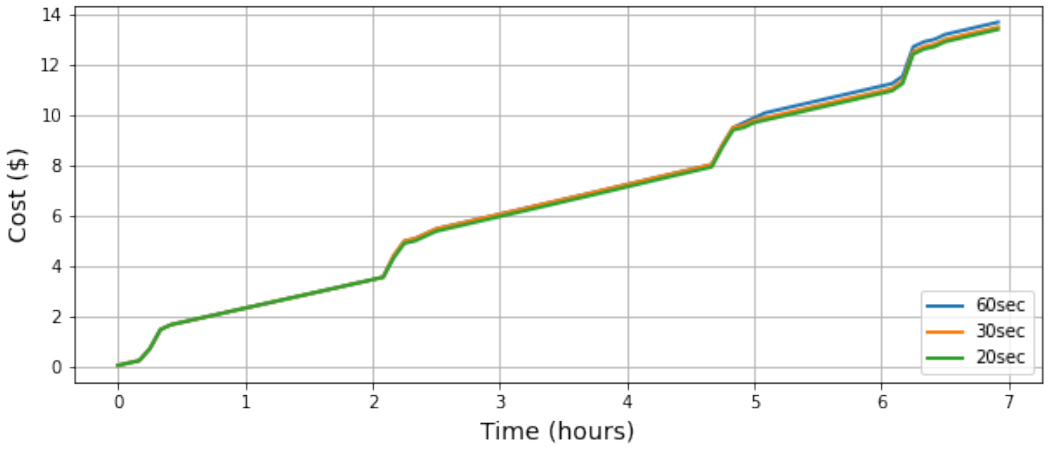
Για το συγκεκριμένο παράδειγμα, παρουσιάζουμε στο παρακάτω διάγραμμα την κατανομή του n_{d-1}, δηλαδή το μέγεθος της συστάδας ένα χρονικό βήμα πριν την προθεσμία, για να παρατηρήσουμε τι έχει να προτείνει ο αλγόριθμος όταν στενεύουν τα χρονικά περιθώρια:



Εικόνα 6.13: Κατανομή του μεγέθους της συστάδας ένα χρονικό βήμα πριν την λήξη της προθεσμίας

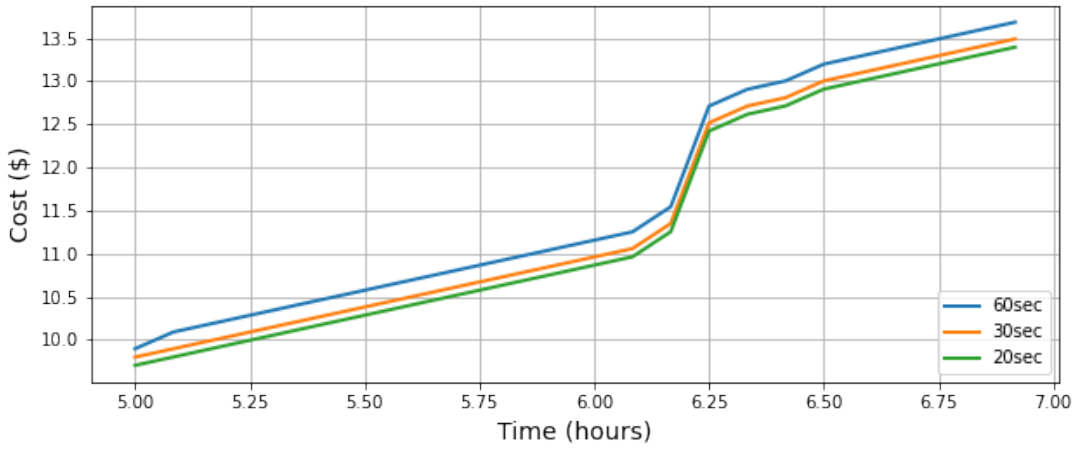
Σχολιασμός: Ο οριζόντιος άξονας αναπαριστά το μέγεθος της συστάδας στο προηγούμενο χρονικό βήμα, n_{d-2}, ενώ ο κατακόρυφος άξονας αναπαριστά το έργο που απομένει στη ροή εργασιών ϖ. Η μάυρη περιοχή του διαγράμματος αφορά τους συνδυασμούς n_{d-2} και ϖ στους οποίους χάνουμε την προθεσμία έτσι κι αλλιώς, δηλαδή όσα μηχανήματα και να δεσμεύσουμε στο τρέχον χρονικό βήμα, δεν πρόκειται να καταφέρουμε να εκτελέσουμε τη ροή εργασιών κατά τη διάρκεια του τρέχοντος χρονικού βήματος. Παρατηρούμε πως όσο κινούμαστε προς τα κάτω στο γράφημα, δηλαδή όσο μεγαλύτερο είναι το έργο που απομένει στη ροή εργασιών κατά την έναρξη του τρέχοντος χρονικού βήματος, τόσο μεγαλύτερη γίνεται η ανάγκη να δεσμεύσουμε περισσότερα μηχανήματα προκειμένου να ολοκληρώσουμε τη ροή εργασιών εντός της προθεσμίας. Επίσης, για μια σταθερή τιμή του ϖ, όσο κινούμαστε προς τα δεξιά στο γράφημα παρατηρούμε μια σταθερότητα στο μέγεθος της συστάδας, καθώς οι τιμές των n_{d-2} και n_{d-1} δεν διαφέρουν κατά πολύ. Το αποτέλεσμα αυτό είναι λογικό, καθώς μια μεγάλη μεταβολή στο μέγεθος της συστάδας, θα σήμαινε μεγάλη υπολογιστική επιβάρυνση, κάτι το οποίο είναι ανεπιθύμητο λίγο πριν την λήξη της προθεσμίας.

Αξίζει στο σημείο αυτό να εξετάσουμε πώς η επιλογή του τ_{opt} επηρεάζει το χρηματικό κόστος εκτέλεσης του πλάνου. Στο παρακάτω διάγραμμα παρουσιάζουμε το αθροιστικό κόστος εκτέλεσης του ίδιου πλάνου με την πάροδο του χρόνου για τις τρεις διαφορετικές τιμές του χρονικού βήματος τ_{opt} που επιλέξαμε ($20sec, 30sec$, $60sec$):



Εικόνα 6.14: Αθροιστικό κόστος εκτέλεσης πλάνου

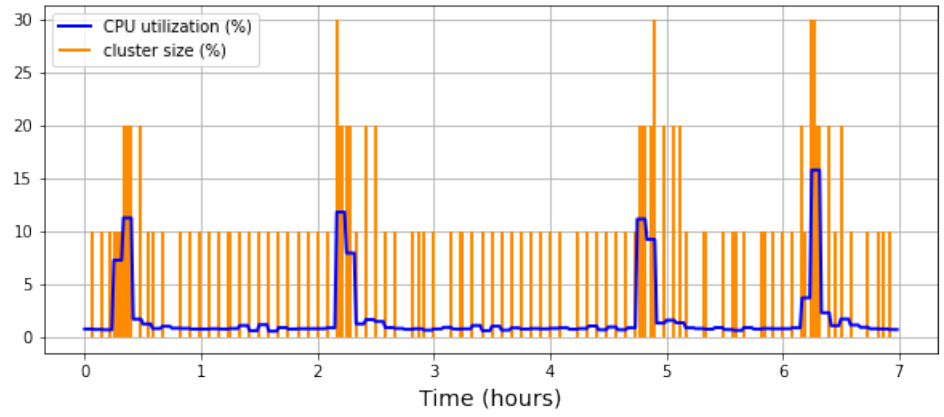
Στο παρακάτω διάγραμμα εστιάζουμε λίγο περισσότερο στην περιοχή του γραφήματος που αντιστοιχεί μετά τις πέντε πρώτες ώρες:



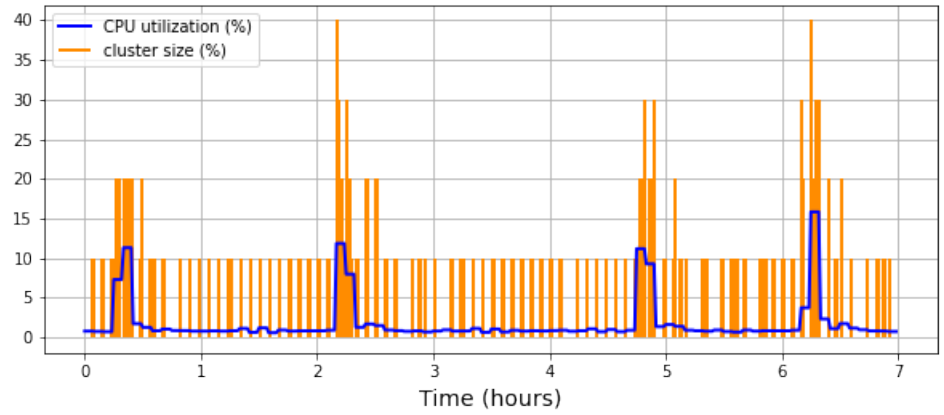
Εικόνα 6.15: Αθροιστικό κόστος εκτέλεσης πλάνου μετά τις πρώτες πέντε ώρες

Σχολιασμός: Παρατηρούμε πως η μείωση του χρονικού βήματος τ_{opt}, δηλαδή η αύξηση της συχνότητας λήψης αποφάσεων για το μέγεθος της υπολογιστικής συστάδας, οδηγεί σε μείωση του χρηματικού κόστους εκτέλεσης των ροών εργασιών που έχουμε επιλέξει. Το αποτέλεσμα αυτό είναι λογικό, αν σκεφτεί κανείς πως με την αύξηση της συχνότητας λήψης αποφάσεων, δίνουμε περισσότερες ευκαιρίες στον αλγόριθμο για να εξετάσει πιο οικονομικές επιλογές για το μέγεθος της συστάδας.

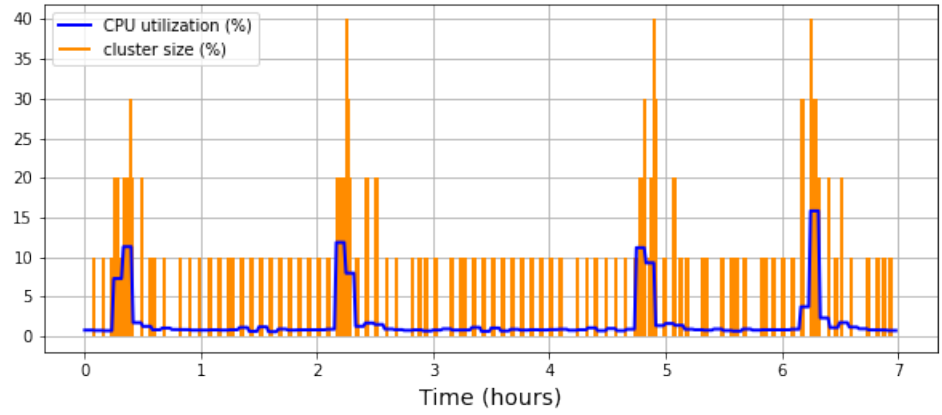
Τέλος, εξετάζουμε πώς επηρεάζεται το μέγεθος της συστάδας από με τον προβλεπόμενο φόρτο εργασιών. Στα παρακάτω γραφήματα παρουσιάζουμε το μέγεθος της συστάδας σε σχέση με το ποσοστό χρήσης CPU για όλες τις επιλογές τ_{opt}. Σημειώνεται πως το μέγεθος της συστάδας απεικονίζεται ως ποσοστό του N.



Εικόνα 6.16: Μέγεθος συστάδας σε σχέση με το ποσοστό χρήσης CPU για $\tau_{\text{opt}} = 60sec$



Εικόνα 6.17: Μέγεθος συστάδας σε σχέση με το ποσοστό χρήσης CPU για $\tau_{\text{opt}} = 30sec$



Εικόνα 6.18: Μέγεθος συστάδας σε σχέση με το ποσοστό χρήσης CPU για $\tau_{\text{opt}} = 20sec$

Σχολιασμός: Παρατηρούμε ότι ο αλγόριθμος διατηρεί ένα ελάχιστο πλήθος από μηχανήματα στη συστάδα και δεσμεύει περισσότερα μόνο όταν είναι αναγκαίο, δηλαδή όταν βλέπει πως υπάρχει αύξηση στον φόρτο εργασιών. Θα χαρακτηρίζαμε λοιπόν των αλγόριθμο ως αντιδραστικό (reactive) και όχι προληπτικό (proactive), καθώς υιοθετεί μια *βλέποντας και κάνοντας* στρατηγική.

Κεφάλαιο 7

Επίλογος

7.1 Σύνοψη και συμπεράσματα

Το ζήτημα της δυναμικής δέσμευσης υπολογιστικών πόρων με οικονομικά βέλτιστο τρόπο είναι ιδιαίτερα σημαντικό για τους χρήστες που χρησιμοποιούν υπηρεσίες Cloud Computing. Η ραγδαία ανάπτυξη που παρουσιάζεται σε αυτές τις υπηρεσίες αποτελεί έναυσμα για την εξέλιξη νέων τεχνικών που θα εξυπηρετούν τους χρήστες στην εκτέλεση των εργασιών που επιθυμούν με το ελάχιστο δυνατό οικονομικό κόστος.

Ο στόχος της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη μιας μεθόδου που εξυπηρετεί τον παραπάνω σκοπό. Για την επίτευξη του στόχου αυτού, χρησιμοποιήσαμε τεχνικές μηχανικής μάθησης για την πρόβλεψη των άγνωστων παραμέτρων του προβλήματος και στη συνέχεια κατασκευάσαμε αλγορίθμους οι οποίοι συνδυάζουν αυτές τις προβλέψεις για να προτείνουν δεσμεύσεις υπολογιστικών πόρων που ελαχιστοποιούν το οικονομικό κόστος εκτέλεσης των επιθυμητών εργασιών.

Η πρόβλεψη των άγνωστων παραμέτρων επετεύχθη με την χρήση Νευρωνικών Δικτύων, και συγκεκριμένα των LSTM δικτύων. Σύμφωνα με τη βιβλιογραφία, τα δίκτυα αυτά ειδικεύονται στην πρόβλεψη χρονοσειρών λόγω της ικανότητάς τους να αναγνωρίζουν μακροπρόθεσμες εξαρτήσεις. Όπως έδειξε και η πειραματική αξιολόγηση, αυτό είναι απόλυτα αληθές, καθώς όπως είδαμε, τα δίκτυα που αναπτύξαμε παρέχουν ικανοποιητικές προβλέψεις για χρονοσειρές μεγάλου χρονικού εύρους.

Ο πρώτος αλγόριθμος δέσμευσης πόρων που παρουσιάσαμε κατασκευάστηκε με την χρήση της αλγοριθμικής τεχνικής των Άπληστων Αλγορίθμων. Αν και ο αλγόριθμος αυτός δεν είναι βέλτιστος, καθώς κάνει κάποιες μη ρεαλιστικές παραδοχές, αποτελεί μια γρήγορη και απλοποιημένη προσέγγιση για την επίλυση του προβλήματός μας.

Ο δεύτερος αλγόριθμος δέσμευσης πόρων που παρουσιάσαμε κατασκευάστηκε με την χρήση της αλγοριθμικής τεχνικής του Δυναμικού Προγραμματισμού. Πρόκειται για μια πιο περίπλοκη προσέγγιση, καθώς πραγματοποιεί λιγότερες απλοποιήσεις από τον προηγούμενο και στηρίζεται σε ένα πιο ρεαλιστικό σενάριο, όπου υπάρχει εξάρτηση μεταξύ του μεγέθους των υπολογιστικών πόρων που δεσμεύονται σε διαδοχικές χρονικές στιγμές.

Χρησιμοποιήθηκαν εξ ολοκλήρου πραγματικά δεδομένα τόσο για την εκπαίδευση των νευρωνικών μας δικτύων, όσο και για τις προσομοιώσεις δέσμευσης πόρων με χρήση του παραπάνω αλγορίθμου. Όλες οι αρχιτεκτονικές νευρωνικών δικτύων που παρουσιάσαμε δίνουν ικανοποιητικά μικρά σφάλματα πρόβλεψης, ενώ για τον αλγόριθμο παρατηρήσαμε πως η αύξηση της συχνότητας λήψης αποφάσεων για το μέγεθος της υπολογιστικής συστάδας, που συνιστάται από τα μηχανήματα που δεσμεύουμε, οδηγεί σε πιο χαμηλά οικονομικά κόστη.

Συνολικά, μέσα από αυτή τη διπλωματική εργασία ήρθαμε σε επαφή και κατανοήσαμε διαφορετικές τεχνικές ανάπτυξης αλγορίθμων βελτιστοποίησης, όπως οι Άπληστοι Αλγόριθμοι και ο Δυναμικός Προγραμματισμός, ενώ παράλληλα μελετήσαμε και αναπτύξαμε νευρωνικά δίκτυα ικανά για την πρόβλεψη χρονικά εξελισσόμενων παραμέτρων.

7.2 Μελλοντική Εργασία

Κλείνουμε την παρούσα εργασία παρουσιάζοντας πιθανές κατευθύνσεις μελλοντικής εργασίας.

- Αρχικά, στην προσέγγισή μας, δεν εστιάζουμε στο πρόβλημα του καθορισμού τιμής προσφοράς για τους υπολογιστικούς πόρους, καθώς θεωρούμε ότι ο χρήστης πραγματοποιεί πάντα μια αρκετά υψηλή προσφορά, και συγκεκριμένα ίση με ∞. Σε μια πιθανή λοιπόν επέκταση της ιδέας μας, ο χρήστης θα μπορεί να δίνει ως επιπλέον είσοδο ένα συνολικό budget, το οποίο δεν πρέπει με τίποτα να ξεπεραστεί.
- Μία άλλη προσέγγιση, θα ήταν να κατασκευαστεί ένα πλήρες end-to-end σύστημα στο οποίο ο πάροχος θα χρησιμοποιεί ένα "έξυπνο σύστημα" τιμολόγησης πόρων [13], ενώ ο χρήστης θα χρησιμοποιεί έναν "έξυπνο πράκτορα" [2] στη θέση του, ο οποίος θα δέχεται ως είσοδο ένα συνολικό budget και θα ανταγωνίζεται ενάντια στους υπόλοιπους χρήστες προκειμένου να καλύψει τις υπολογιστικές ανάγκες του χρήστη.
- Επιπλέον, στην προσέγγισή μας θεωρήσαμε πως ο χρήστης δημιουργεί ομογενείς υπολογιστικές συστάδες, δηλαδή συστάδες που αποτελούνται από μηχανήματα του ίδιου τύπου. Μια πιθανή επέκταση θα ήταν η χρήση μηχανημάτων διαφορετικών τύπων.
- Επιπλέον, με τη βοήθεια των κατάλληλων υπολογιστικών πόρων και πλήθους δεδομένων εκπαίδευσης θα μπορούσαμε να πραγματοποιήσουμε ένα fine tuning στα νευρωνικά μας δίκτυα για την επίτευξη ακόμη καλύτερα προσεγγίσεων των άγνωστων παραμέτρων.

Βιβλιογραφία

[1] Amazon Elastic Compute Cloud - User Guide for Linux Instances. page 1476.

[2] Constantinos Bitsakos, Ioannis Konstantinou, and Nectarios Koziris. DERP: A Deep Reinforcement Learning Cloud System for Elastic Resource Provisioning. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 21–29, Nicosia, December 2018. IEEE.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[4] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles - SOSP '17*, pages 153–167, Shanghai, China, 2017. ACM Press.

[5] Bingqian Du, Chuan Wu, and Zhiyi Huang. Learning Resource Allocation and Pricing for Cloud Profit Maximization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:7570–7577, July 2019.

[6] Simon S. Haykin. *Neural networks and learning machines*. Pearson Education, Upper Saddle River, NJ, third edition, 2009.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[8] Botong Huang and Jun Yang. Cümülön-D: data analytics in a dynamic spot market. *Proceedings of the VLDB Endowment*, 10(8):865–876, April 2017.

[9] Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[10] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[11] Sebastian Ruder. An overview of gradient descent optimization algorithms., 2016. cite arxiv:1609.04747Comment: Added derivations of AdaMax and Nadam.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018.

[13] Αστέριος Τσιούρβας. A Mechanism Design and Deep Learning Approach for Revenue Maximization on Cloud Dynamic Resource Provisioning Systems. October 2019.