ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΜΣ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία
Μεταπτυχιακού Διπλώματος Ειδίκευσης

Variational Autoencoders & Applications

Φοιτητής:
Χρήστος ΚΟΡΜΑΡΗΣ
ΕΥ1617

Επιβλέπων Καθηγητής:
Δρ. Μιχάλης ΤΙΤΣΙΑΣ

ΑΘΗΝΑ, 3 ΜΑΙΟΥ 2018

Δήλωση Συγγραφής

Εγώ, ο Χρήστος Κορμαρής, δηλώνω ότι αυτή η διατριβή με τον τίτλο "Variational Autoencoders" που παρουσιάζεται, είναι δικό μου έργο και επιβεβαιώνω ότι:

- Το έργο αυτό έγινε εξ' ολοκλήρου κατά τη διαδικασία υποψηφιότητας μου, για ερευνητικό πτυχίο στο Οικονομικό Πανεπιστήμιο Αθηνών.
- Οποιοδήποτε σημείο της παρούσας διατριβής έχει χρησιμοποιηθεί προηγουμένως για παρόμοιο πτυχίο στο Οικονομικό Πανεπιστήμιο Αθηνών ή σε οποιοδήποτε άλλο ίδρυμα, αυτό έχει αναφερθεί σαφώς.
- Στα σημεία που έχω συμβουλευτεί τη δημοσιευμένη εργασία άλλων ερευνητών, αυτό διευκρινίζεται πάντα σαφώς.
- Όπου έχω αναφερθεί σε δημοσιεύσεις άλλων ερευνητών, η πηγή αναφέρεται με ευκρίνεια. Με εξαίρεση τις αναφορές αυτές, η εργασία είναι εξ' ολοκλήρου δικό μου έργο.
- Έχω αναφέρει όλες τις κύριες πηγές βοηθείας.
- Όπου το έργο μου βασίζεται σε συνεργασία που έχω κάνει από κοινού με άλλους, αυτό αναφέρεται με ακρίβεια.

Αρχαίο Κείμενο

“ ξεῖν’, ἦ τοι μὲν ὄνειροι ἀμήχανοι ἀκριτόμυθοι
γίγνοντ’, οὐδέ τι πάντα τελείεται ἀνθρώποισι.
δοιαὶ γάρ τε πύλαι ἀμενηνῶν εἰσίν ὀνείρων·
αἱ μὲν γὰρ κεράεσσι τετεύχαται, αἱ δ’ ἐλέφαντι·
τῶν οἳ μέν κ’ ἔλθωσι διὰ πριστοῦ ἐλέφαντος,
οἵ ῥ’ ἐλεφαίρονται, ἔπε’ ἀκράαντα φέροντες·
οἳ δὲ διὰ ξεστῶν κεράων ἔλθωσι θύραζε,
οἵ ῥ’ ἔτυμα κραίνουσι, βροτῶν ὅτε κέν τις ἴδηται. “

Ὁμήρου Ὀδύσσεια, Ῥαψῳδία τ, στ. 562 (Πρωτότυπο)

Απόδοση στην Καθαρεύουσα

“Ξένε, τά ὄνειρα βέβαια ἀμήχανα καί ἀκριτόμυθοι εἶναι, οὐδέ πάντα τελοῦνται στούς ἀνθρώπους. Γιατί δύο οἱ πύλες των ἀδυνάμων εἶναι ὀνείρων· ἡ μία ἀπό κέρατα ἔχει φτιαχθῆ, κι ἡ ἄλλη ἀπό ἐλεφαντόδοντο. Καί ὅσα διέλθουν ἀπό το πριονιστό ἐλεφαντόδοντο μᾶς ἀπατοῦν, ἔπη ἀνεκτέλεστα φέροντα· καί ὅσα διά ξυστῶν κεράτων ἔλθουν πρός τά ἔξω, αὐτά τ’ ἀληθινά κραίνουν, σέ ὅποιον ἀπό τούς βροτούς τά ἰδῆ.“

Ομήρου Οδύσσεια, Ραψωδία τ, στ. 562 (Μετάφραση Κώστα Δούκα)

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

Προοίμιο *(προ + οἶμος = δρόμος)*

Σχολή Επιστημών & Τεχνολογίας της Πληροφορίας
Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών στην Επιστήμη των Υπολογιστών

Variational Autoencoders & Applications

Φοιτητής: Χρήστος Κορμαρής. Επιβλέπων Καθηγητής: Διδάκτωρ Μιχάλης Τίτσιας

Ο variational autoencoder είναι μια μέθοδος που μπορεί να παράγει τεχνητά δεδομένα που θα μοιάζουν με ένα δοσμένο σύνολο πραγματικών δεδομένων. Για παράδειγμα, αν θέλουμε να δημιουργήσουμε καινούργιες τεχνητές φωτογραφίες από γάτες, μπορούμε να χρησιμοποιήσουμε ένα variational autoencoder αλγόριθμο για να τα καταφέρουμε, αφού τον εκπαιδεύσουμε σε ένα μεγάλο σύνολο δεδομένων από γάτες. Τα δεδομένα εισόδου είναι μη κατηγοριοποιημένα διότι δεν ενδιαφερόμαστε να τα κατηγοριοποιήσουμε σε κάποια κλάση, αλλά επιθυμούμε να είμαστε σε θέση να μάθουμε τα πιο σημαντικά χαρακτηριστικά ή τις ομοιότητες μεταξύ τους. Εφόσον τα δεδομένα μας δεν έχουν ετικέτες-κατηγορίες, ο variational autoencoder χαρακτηρίζεται ως αλγόριθμος μη επιβλεπόμενης μάθησης. Όσον αφορά το παράδειγμα των εικόνων από γάτες, ο αλγόριθμος μπορεί να μάθει να αναγνωρίζει ότι μια γάτα πρέπει να έχει δύο αυτιά, μια μύτη, μουστάκια, τέσσερα πόδια, μια ουρά και μια ποικιλία από χρώματα. Ο αλγόριθμος χρησιμοποιεί δύο νευρωνικά δίκτυα, έναν κωδικοποιητή και έναν αποκωδικοποιητή, τα οποία εκπαιδεύονται ταυτόχρονα. Ένας variational autoencoder θα έχει καλές εφαρμογές σε περιπτώσεις όπου θα επιθυμούσαμε να παράγουμε μεγαλύτερο σύνολο δεδομένων, για καλύτερη εκπαίδευση σε διάφορα νευρωνικά δίκτυα. Επίσης, κάνει ελάττωση διάστασης των δεδομένων, συμπιέζοντάς τα σε λανθάνουσες μεταβλητές. Τρέχουμε υλοποιήσεις διαφορετικών variational autoencoder σε διάφορα dataset, όπως MNIST, Binarized MNIST, CIFAR-10, OMNIGLOT, YALE Faces, ORL Face Database, MovieLens, υλοποιημένες σε Python 3, χρησιμοποιώντας τρεις διαφορετικές βιβλιοθήκες κάθε φορά, TensorFlow, PyTorch και Keras και παρουσιάζουμε τα αποτελέσματα. Συστήνουμε έναν απλό αλγόριθμο συμπλήρωσης ελλιπών τιμών που χρησιμοποιεί K-NN συνεργατικό φιλτράρισμα για να κάνει προβλέψεις (πχ. για ελλιπή pixel). Τέλος, χρησιμοποιούμε το variational autoencoder σε αλγόριθμο συμπλήρωσης ελλιπών τιμών για να προβλέψουμε τις ελλιπείς τιμές σε διάφορα dataset. Ο K-NN αλγόριθμος τα πήγε σπουδαία στις προβλέψεις και το variational autoencoder σύστημα συμπλήρωσης επέφερε πολύ ικανοποιητικά αποτελέσματα. Έχει επίσης υλοποιηθεί GUI.

Ευχαριστίες

Ευχαριστώ τον καθηγητή *Δρ. Μιχάλη Τίτσια* για τη βοήθεια και τις πολύτιμες συμβουλές του. Επίσης, θέλω να ευχαριστήσω τον υπεύθυνο του προγράμματος, τον κύριο καθηγητή *Δρ. Γεώργιο Πολύζο*, για την κατανόηση και την ενθάρρυνσή του κατά τη διάρκεια του μεταπτυχιακού. Τέλος, θέλω να ευχαριστήσω ξεχωριστά τους καθηγητές του μεταπτυχιακού, κύριο *Δρ. Γεώργιο Σταμούλη*, *Δρ. Ιωάννη Κωτίδη*, *Δρ. Βάνα Καλογεράκη*, *Δρ. Ευάγγελο Μαρκάκη*, *Δρ. Σταύρο Τουμπή*, *Δρ. Ίων Ανδρουτσόπουλο*, *Δρ. Γεώργιο Παπαϊωάννου*, *Δρ. Μιχάλη Βαζιργιάννη*, *Δρ. Βασίλειο Σύρη* και *Δρ. Σοφία Δημέλη* για όλη τη γνώση που μου προσέφεραν.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κατάλογος Κώδικα

Συντομογραφίες

CF	**C**ollaborative **F**iltering (Συνεργατικό Φιλτράρισμα)
CIFAR dataset	**C**anadian **I**nstitute **f**or **A**dvanced **R**esearch dataset
ELBO	**E**vidence **L**ower **Bo**und (Ελάχιστο Φράγμα Απόδειξης)
EM algorithm	**E**xpectation **M**aximization algorithm (αλγόριθμος Μεγιστοποίησης της Προσδοκίας)
KL divergence	**K**ullback-**L**eibler divergence (KL απόκλιση)
K-NN	**K** **N**earest **N**eighbors (K Κοντινότεροι Γείτονες)
MNIST dataset	**M**odified **N**ational **I**nstitute of **S**tandards & **T**echnology dataset
VAE	**V**ariational **A**uto**e**ncoder
VB methods	**V**ariational **B**ayes methods (μέθοδοι Variational Bayes)
TM	**Τ**υχαία **Μ**εταβλητή

Σύμβολα

$D_{KL}(P \,\|\| \, Q)$	**K**ullback-**L**eibler Απόκλιση μεταξύ δύο κατανομών **P** και **Q**
E	Μέση τιμή
L	Μεταβλητό Ελάχιστο Φράγμα (ELBO)
$N(\mu, \sigma^2)$	Κανονική (**N**ormal) κατανομή με Μέση τιμή μ και Διασπορά σ^2
$P(X)$	Κατανομή πιθανότητας τυχαίας μεταβλητής X
x	Ένα δεδομένο εισόδου
X	Πολλά δεδομένα εισόδου
z	Μία λανθάνουσα μεταβλητή
Z	Πολλές λανθάνουσες μεταβλητές
D	Διάσταση δεδομένων εισόδου / Αριθμός των pixel
M_1	Αριθμός νευρόνων στον κωδικοποιητή
M_2	Αριθμός νευρόνων στον αποκωδικοποιητή
Z_{dim}	Διάσταση λανθανουσών μεταβλητών
$\epsilon \sim N(\mu, \sigma^2)$	Η Τ.Μ. (τυχαία μεταβλητή) ϵ ακολουθεί κανονική κατανομή
θ	Παράμετροι του αποκωδικοποιητή (decoder)
μ	Μία μεταβλητή μέσης τιμής
M	Πολλές μεταβλητές μέσης τιμής
σ	Μία μεταβλητή τυπικής απόκλισης
Σ	Πολλές μεταβλητές τυπικής απόκλισης
σ^2	Διασπορά
Σ^2	Διασπορές
ϕ	Παράμετροι του κωδικοποιητή (encoder)
@	Τελεστής πολλαπλασιασμού πινάκων σε PyTorch
$.*$	Τελεστής πολλαπλασιασμού πινάκων στοιχείο προς στοιχείο (element-wise)

Αφιερώνω τη διπλωματική μου εργασία στους γονείς μου.

Κεφάλαιο 1

Εισαγωγή

1.1 Προκαταρκτικά

Έστω X τα δεδομένα εκπαίδευσης. Σκοπεύουμε να μεγιστοποιήσουμε την πιθανότητα κάθε δεδομένου εκπαίδευσης x, από τα δεδομένα εκπαίδευσης X, σύμφωνα με τον τύπο:

$$P(X) = \int P(X, z)\, dz = \int P(X \mid z, \phi) \cdot P(z)\, dz$$

όπου Z είναι συνεχής και ΟΧΙ διακριτή κατανομή και κάθε z είναι δεδομένο του Z. Συνεπώς, παίρνουμε ολοκλήρωμα από κοινού κατανομών και όχι άθροισμα, ώστε να υπολογίσουμε την κατανομή X. Η κατανομή X μπορεί να είναι είτε συνεχής είτε διακριτή. Για παράδειγμα, αν το dataset περιλαμβάνει εικόνες, η X μπορεί να είναι μια Bernoulli διακριτή κατανομή με διακριτές τιμές, 1 ή 0, 1 για άσπρα pixel και 0 για μαύρα pixel. Αντιθέτως, αν η X παίρνει πραγματικές τιμές μεταξύ του διαστήματος $[0, 1]$, τότε είναι συνεχής κατανομή. Με μια συνεχή κατανομή, το dataset μπορεί να αναπαραστήσει παραπάνω από δύο χρώματα. (Carl Doersch 2016. [2])

1.2 Αναπαράσταση του Μοντέλου

Τα δεδομένα εισόδου του μοντέλου μας είναι κυρίως εικόνες. Οι εικόνες εισόδου αναπαρίστανται ως μια μεταβλητή μεγέθους N x D. Το N συμβολίζεται ως το μέγεθος των εικόνων και το D συμβολίζει τον αριθμό των pixel. Ένα καλό παράδειγμα είναι το *MNIST* dataset, που περιλαμβάνει εικόνες από μονοψήφιους αριθμούς, από το *0-9*. Άλλα παραδείγματα από dataset με εικόνες είναι το *Binarized MNIST* dataset, τα *CIFAR-10 / CIFAR-100* dataset, το *OMNIGLOT* dataset, το *YALE Faces* dataset και το *The Database of Faces* dataset. Επίσης, θα εξετάσουμε το *MovieLens* dataset, το οποίο περιλαμβάνει βαθμολογίες που έδωσαν πολλοί χρήστες σε συγκεκριμένες ταινίες.

Ο variational autoencoder είναι μια ακολουθία από δύο νευρωνικά δίκτυα, το ένα μετά το άλλο. Το πρώτο νευρωνικό δίκτυο ονομάζεται κωδικοποιητής (encoder) και το δεύτερο νευρωνικό δίκτυο ονομάζεται αποκωδικοποιητής (decoder). Επίσης, ο κωδικοποιητής και ο αποκωδικοποιητής εκπαιδεύονται ταυτόχρονα. Ο ρόλος του κωδικοποιητή είναι να μάθει πως να αναπαριστά τα κρυμμένα

χαρακτηριστικά του dataset, αποθηκεύοντάς τα σε λανθάνουσες μεταβλητές ελαττωμένης διάστασης. Από την άλλη πλευρά, ο αποκωδικοποιητής, κατασκευάζει τεχνητά δεδομένα, από τις λανθάνουσες μεταβλητές που έχουμε μάθει. Τα τεχνητά δεδομένα πρέπει να είναι παρόμοια με τα πρωτότυπα δεδομένα εισόδου, αλλά ΟΧΙ πανομοιότυπα. Διαφορετικά, η διαδικασία απέτυχε. Όταν ο αποκωδικοποιητής τελειώσει, ο στόχος μας έχει ολοκληρωθεί.

1.3 Variational Συμπέρασμα (Variational Inference)

Πρώτα θέλουμε να υπολογίσουμε τον κωδικοποιητή, δηλαδή θέλουμε να υπολογίσουμε:

$$P(Z \mid X) = \frac{P(X \mid Z) \cdot P(Z)}{P(X)} = \frac{P(X \mid Z) \cdot P(Z)}{\int P(X, Z)\, dz}$$

Ο κάθε όρος μπορεί να αναπαρασταθεί ως:

$$\textbf{posterior} = \frac{\textbf{likelihood} \cdot \textbf{prior}}{\textbf{normalizing constant}}$$

Όπου posterior: *P(Z|X)*, likelihood: *P(X|Z)*, prior: *P(Z)*, normalizing constant: *P(X)*.
Ο παρανομαστής *P(X)*, εκτός από *σταθερά κανονικοποίησης (normalizing constant)*, ονομάζεται και *απόδειξη (evidence)*. Μπορούμε να τον υπολογίσουμε περιθωριοποιώντας ως προς τις λανθάνουσες μεταβλητές Z, ως εξής:

$$P(X) = \int P(X \mid Z, \theta) \cdot P(Z)\, dz = \int P(X, Z, \theta)\, dz$$

Ωστόσο, υπολογίζοντας αυτό το ολοκλήρωμα απαιτεί εκθετικό χρόνο, επειδή η κατανομή των λανθανουσών μεταβλητών Z είναι συνεχής. Ο όρος $P(X \mid Z, \theta)$ είναι μια πολύπλοκη συνάρτηση πιθανοφάνειας, λόγω της μη γραμμικότητας των κρυφών επιπέδων. Το πρόβλημα μεγιστοποίησης του όρου *logP(Z|X)*, μέσω του κανόνα του Bayes, ανάγεται στο ακόλουθο:

$$logP(Z \mid X) = log\frac{P(X \mid Z) \cdot P(Z)}{P(X)} = logP(X \mid Z) + logP(Z) - logP(X)$$

Εφόσον ο όρος $P(X)$ είναι μη υπολογίσημος, ο όρος $P(Z \mid X)$ είναι επίσης μη υπολογίσημος, μέσω του κανόνα του Bayes. Για να υπολογίσουμε το $P(Z \mid X)$, θα χρησιμοποιήσουμε μια άλλη μέθοδο που λέγεται **variational συμπέρασμα (variational inference)**, χρησιμοποιώντας μια οικογένεια κατανομών που ονομάζουμε $Q_\phi(Z \mid X)$, όπου $\boldsymbol{\varphi}$ είναι μια παράμετρος. Μαθαίνουμε την παράμετρο $\boldsymbol{\varphi}$ μέσω stochastic (ή mini-batch) αλγορίθμου ανοδικής (ή καθοδικής) κλίσης. Σε κάθε επανάληψη,

υπολογίζουμε τη συνάρτηση κόστους ή την πιθανοφάνεια, η οποία είναι το ελάχιστο φράγμα του όρου $logP(X)$. Θέλουμε να μεγιστοποιήσουμε αυτόν τον όρο, επομένως θέλουμε να μεγιστοποιήσουμε το ελάχιστο φράγμα. Θα αναλύσουμε αυτή τη διαδικασία περαιτέρω στο Κεφάλαιο 2.

Χρησιμοποιώντας το variational συμπέρασμα (variational inference) καταφέραμε να κάνουμε τον υπολογισμό του όρου $P(Z \mid X)$ εφικτό. Στο δεύτερο σκέλος του variational autoencoder, τον αποκωδικοποιητή, θέλουμε να υπολογίσουμε τον όρο $P_\theta(X \mid Z)$. Θα χρησιμοποιήσουμε stochastic (ή mini-batch) αλγόριθμο ανοδικής (ή καθοδικής) κλίσης για να μάθουμε τις
παραμέτρους $\boldsymbol{\vartheta}$.

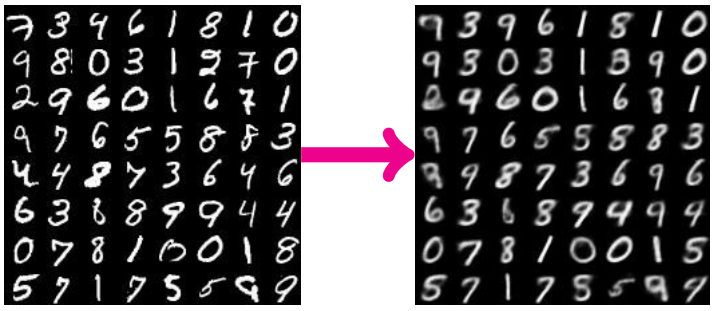
1.3.1 Πλεονεκτήματα του Variational Inference

(Max Welling, Diederik P. Kingma 2013. [3])

1. **Διαχείριση Μη Υπολογισιμότητας (Intractability):** Όπως εξηγήσαμε παραπάνω, ο υπολογισμός του όρου $P(X)$ είναι αδύνατος και ως αποτέλεσμα ισχύει το ίδιο και για τον όρο $P(Z \mid X)$. Συνεπώς, ο αλγόριθμος **Expectation-Maximization (EM)** και άλλοι αλγόριθμοι τύπου Variational Bayes δεν μπορούν να χρησιμοποιηθούν. Το variational συμπέρασμα (variational inference) όμως, που δεν απαιτεί το υπολογισμό αυτού του όρου, μπορεί να δώσει τη λύση.
2. **Large datasets:** Αλγόριθμοι βελτιστοποίησης που εκπαιδεύονται σε όλο το σύνολο των δεδομένων σε κάθε επανάληψη (πχ. batch gradient descent) είναι πολύ δαπανηροί. Χάρη στο variational inference οι παράμετροι ενημερώνονται χρησιμοποιώντας μικρές δέσμες δεδομένων (mini-batches) ή ακόμα μεμονομένα στιγμιότυπα (πχ. stochastic gradient descent). Λύσεις που βασίζονται σε δειγματοληψία, όπως Monte Carlo EM είναι πολύ αργές γιατί εμπλέκουν ακριβούς βρόχους δειγματοληψίας για κάθε δεδομένο ξεχωριστά.

1.3.2 Μειονεκτήματα του Variational Inference

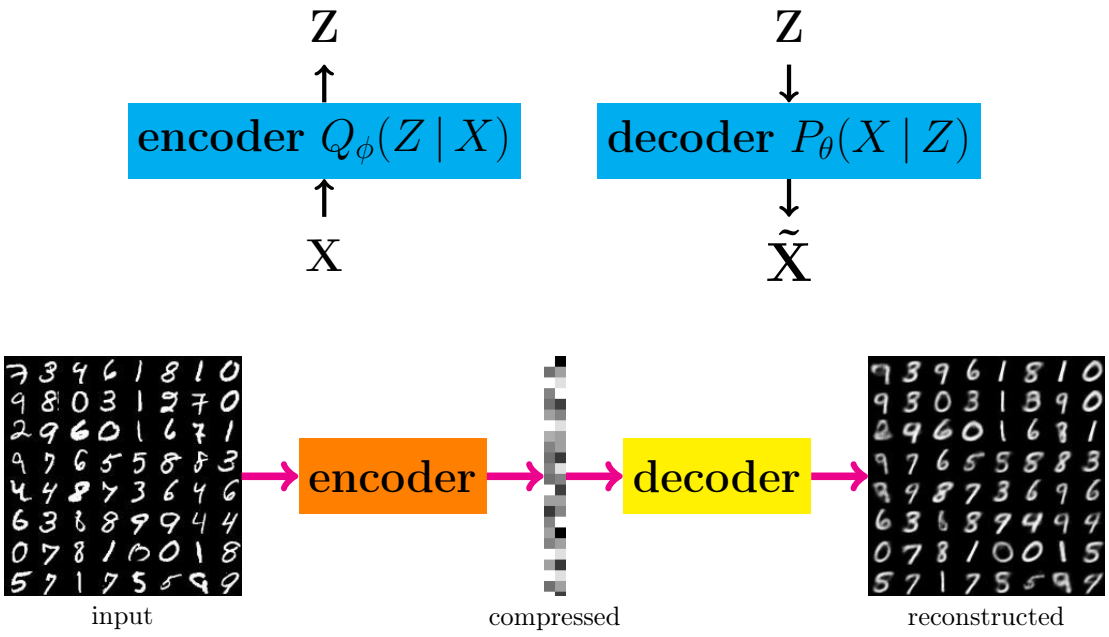
(Aaron Courville, Ian Goodfellow, Yoshua Bengio 2016. [4]) Η προσέγγιση του variational autoencoder είναι κομψή, θεωρητικά ευχάριστη και απλή στην υλοποίηση. Επίσης, επιφέρει σχεδόν τέλεια αποτελέσματα και είναι μέσα στις τεχνολογίες τελευταίας λέξης που αφορούν προσεγγίσεις σε γεννητική μοντελοποίηση (generative modelling). Το κύριο μειονέκτημα είναι ότι δείγματα από variational autoencoder εκπαιδευμένα σε εικόνες τείνουν να είναι μερικώς θολά. Οι αιτίες αυτού του φαινομένου δεν είναι ακόμα γνωστές και παραμένει να βρεθούν. Μια πιθανή εξήγηση είναι ότι η θολότητα είναι ένα εγγενές αποτέλεσμα της μέγιστης πιθανοφάνειας, που ελαχιστοποιεί την απόκλιση $D_{KL}(P(Z|X) \,||\, Q(Z|X))$. Αυτό σημαίνει ότι το μοντέλο θα αναθέσει μεγάλες πιθανότητες σε δεδομένα που ανήκουν στο σύνολο εκπαίδευσης, αλλά μπορεί επίσης να αναθέσει μεγάλες πιθανότητες και σε υπόλοιπα δεδομένα. Τα υπόλοιπα δεδομένα μπορεί να περιλαμβάνουν θολές εικόνες.



Εικονα 1.1: Παράδειγμα δεδομένων εισόδου και εξόδου ενός VAE. Όπως παρατηρούμε, τα τελικά αποτελέσματα είναι μερικώς θολά.

1.3.3 Κύρια Βήματα του Variational Autoencoder

1. Λαμβάνουμε τα δεδομένα (εικόνες) εισόδου X, ως μεταβλητή μεγέθους N x D.
2. Εκπαιδεύουμε τον κωδικοποιητή, που συμβολίζεται ως $Q_\phi(Z \mid X)$, χρησιμοποιώντας batch gradient descent (όπου $\boldsymbol{\varphi}$ είναι οι παράμετροι του κωδικοποιητή και Z είναι οι λανθάνουσες μεταβλητές ελαττωμένης διάστασης).
3. Ταυτόχρονα με τον κωδικοποιητή, εκπαιδεύουμε τον αποκωδικοποιητή, που συμβολίζεται ως $P_\theta(X \mid Z)$, χρησιμοποιώντας batch gradient descent (όπου $\boldsymbol{\vartheta}$ είναι οι παράμετροι του αποκωδικοποιητή).
4. Είμαστε έτοιμοι να δείξουμε και να χρησιμοποιήσουμε τα καινούρια μας τεχνητά δεδομένα (εικόνες) $\tilde{\mathbf{X}}$, τα οποία πρέπει να μοιάζουν με τα αρχικά δεδομένα.



Εικονα 1.2: Σχήμα Variational Autoencoder.

Κεφάλαιο 2

Υπολογίζοντας το Variational Ελάχιστο Φράγμα (ELBO)

2.1 Χρησιμοποιώντας την Ανισότητα του Jensen για να Υπολογίσουμε το Variational Ελάχιστο Φράγμα (ELBO)

Εφόσον ο όρος $P(Z|X)$ είναι μη υπολογίσιμος, όπως εξηγήσαμε στο Κεφάλαιο 1, χρησιμοποιούμε μια αυθαίρετη κατανομή $Q(Z)$ για να προσεγγίσουμε την πραγματική εκ των υστέρων κατανομή $P(Z|X)$.

$$logP(X) = log \int P(X,z)\,dz = log \int P(X,z) \cdot \frac{Q(Z)}{Q(Z)}\,dz = log(E_q[\frac{P(X,z)}{Q(Z)}])$$

Από την ανισότητα του Jensen για κοίλες συναρτήσεις έχουμε $f(E[X]) \geq E[f(X)]$. Εφόσον η λογαριθμική συνάρτηση είναι κοίλη, έχουμε:

$$logP(X) \geq E_q[logP(X,z)] - E_q[logQ(Z)]$$

Ορίζουμε το ELBO ως εξής:

$$ELBO = L(X,Q) = E_q[logP(X,z)] - E_q[logQ(Z)]$$

Συμπερασματικά, είναι προφανές ότι το ELBO είναι το ελάχιστο φράγμα των λογαριθμικών πιθανοτήτων των παρατηρήσεων ($logP(X)$). Συνεπώς, αν σε ορισμένες περιπτώσεις επιθυμούμε να μεγιστοποιήσουμε την περιθώρια πιθανότητα, μπορούμε εναλλακτικά να μεγιστοποιήσουμε το variational ελάχιστο φράγμα (ELBO).

2.2 Χρησιμοποιώντας την KL Απόκλιση για να Υπολογίσουμε το ELBO

(Carl Doersch 2016. [2]) Στην περίπτωση των Variational Autoencoder, η KL-απόκλιση (Kullback-Leibler απόκλιση) είναι η πιθανοφάνεια μεταξύ της πραγματικής κατανομής των λανθανουσών μεταβλητών Z, δεδομένου των X, $P(Z|X)$, και της προσεγγιστικής κατανομής των λανθανουσών μεταβλητών Z, δεδομένου των X, $Q_\phi(Z|X)$. Για το δεύτερο όρο, $Q_\phi(Z|X)$, ισχύει η ακόλουθη εξίσωση:

$$Q_\phi(Z|X) \approx Q_\phi(Z)$$

Η KL απόκλιση μεταξύ δύο κατανομών, παίρνει την ακόλουθη μορφή:

$$D_{KL}[Q_\phi(Z)\,||\,P(Z|X)] = E_{Z\sim Q}[\frac{logQ_\phi(Z)}{logP(Z|X)}] \Rightarrow$$

$$D_{KL}[Q_\phi(Z)\,||\,P(Z|X)] = E_{Z\sim Q}[logQ_\phi(Z) - logP(Z|X)] \Rightarrow$$

όπου το D συμβολίζει την KL-απόκλιση μεταξύ δύο κατανομών.

Αφού εφαρμόσουμε τον κανόνα του Bayes στον δεύτερο όρο, έχουμε:

$$D_{KL}[Q_\phi(Z)\,||\,P(Z|X)] = E_{Z\sim Q}[logQ_\phi(Z) - log[\frac{P_\theta(X|Z)\cdot P(Z)}{P(X)}]] \Rightarrow$$

$$D_{KL}[Q_\phi(Z)\,||\,P(Z|X)] = E_{Z\sim Q}[logQ_\phi(Z) - logP_\theta(X|Z) - logP(Z) + logP(X)] \Rightarrow$$

$$logP(X) - D_{KL}[Q_\phi(Z)\,||\,P(Z|X)] = E_{Z\sim Q}[logP_\theta(X|Z)] - D_{KL}[Q_\phi(Z)\,||\,P(Z)] \Rightarrow$$

$$logP(X) - D_{KL}[Q_\phi(Z|X)\,||\,P(Z|X)] = E_{Z\sim Q}[logP_\theta(X|Z)] - D_{KL}[Q_\phi(Z|X)\,||\,P(Z)]$$

Η τελευταία εξίσωση είναι το variational ελάχιστο φράγμα, το οποίο ονομάζουμε **ELBO** από δω και στο εξής. Το αριστερό σκέλος της εξίσωσης έχει τον όρο που θέλουμε να μεγιστοποιήσουμε, $P(X)$, συν έναν όρο σφάλματος. Ο όρος σφάλματος είναι η Kullback-Leibler απόκλιση μεταξύ των $Q_\phi(Z|X) \approx Q_\phi(Z)$ και $P(Z|X)$, που οδηγεί την κατανομή Q να παράγει λανθάνουσες μεταβλητές Z, δεδομένου των μεταβλητών εισόδου X. Θέλουμε να ελαχιστοποιήσουμε την Kullback-Leibler απόκλιση μεταξύ των δύο κατανομών. Το πρόβλημα ανάγεται στη μεγιστοποίηση του όρου ELBO. Αν η κατανομή Q έχει προσεγγιστεί με υψηλή ακρίβεια, τότε ο όρος του σφάλματος γίνεται μικρός.

Συνοπτικά, το ELBO προκύπτει από τον παρακάτω τύπο:

$$ELBO = L(X,Q) = logP(X) - D_{KL}[Q_\phi(Z|X)\,||\,P(Z|X)] \Rightarrow$$

$$logP(X) = L(X,Q) + D_{KL}[Q_\phi(Z|X)\,||\,P(Z|X)]$$

εφόσον η KL-απόκλιση είναι μη αρνητική, έχουμε:

$$logP(X) \geq L(X,Q)$$

γι' αυτό ονομάζουμε το L, ELBO ή variational ελάχιστο φράγμα (Max Welling, Diederik P. Kingma 2013. [3]). Επίσης, το ELBO είναι ίσο με:

$$ELBO = L(X,Q) = E_{Z\sim Q}[logP_\theta(X|Z)] - D_{KL}[Q_\phi(Z|X)\,||\,P(Z)] \Longrightarrow_{Q_\phi(Z|X)\approx Q_\phi(Z)}$$

$$\mathbf{L(X,Q)} = \mathbf{E_{Z\sim Q}[logP_\theta(X|Z)]} - \mathbf{D_{KL}[Q_\phi(Z)\,||\,P(Z)]}$$

Ο όρος $E_{Z\sim\ Q}[logP_\theta(X|Z)]$ ονομάζεται **κόστος ανακατασκευής** (reconstruction cost).
Ο όρος $D_{KL}[Q_\phi(Z)||P(Z)]$ ονομάζεται **ποινή** (penalty) ή **όρος κανονικοποίησης** (regularization term). Ο όρος ποινής εξασφαλίζει ότι η εξήγηση των δεδομένων, $Q_\phi(Z|X) \approx Q_\phi(Z)$ δεν αποκλίνει πολύ από τον όρο των παρατηρήσεων $P(Z)$. Επιπλέον, ο όρος ποινής μας βοηθά να εφαρμόσουμε το **Ξυράφι του Occam** στον κωδικοποιητή μας (inference model). Ο όρος αυτός είναι πάντα μεγαλύτερος ή ίσος του 0 και άρα μπορεί να παραλειφθεί.

Μπορούμε να εκπαιδεύσουμε ταυτόχρονα τον κωδικοποιητή και τον αποκωδικοποιητή.

Εικονα 2.1: Το Elbo Room είναι ένα bar που βρίσκεται στην ιστορική περιοχή Mission District του San Francisco στην οδό Valencia Street μεταξύ 17th Street και 18th Street.

Κεφάλαιο 3

Αλγόριθμος Προς τα Πίσω Διάδοσης (Back-Propagation)

3.1 Βελτιστοποιώντας τη Συνάρτηση Κόστους (ELBO)

Όπως αναφέραμε στο προηγούμενο κεφάλαιο, εκπαιδεύουμε ταυτόχρονα τον κωδικοποιητή - (inference model) $Q_\phi(Z|X)$, και τον αποκωδικοποιητή - (generative model), $P_\theta(X|Z)$, βελτιστοποιώντας το variational lower bound (ELBO), χρησιμοποιώντας αλγόριθμο **προς τα πίσω διάδοσης** (gradient back-propagation). Καταλήξαμε λοιπόν στον ακόλουθο τύπο για το ELBO:

$$L(X, Q) = E_{Z\sim Q}[logP_\theta(X|Z)] - D_{KL}[Q_\phi(Z)\,||\,P(Z)]$$

Οι κανόνες ενημέρωσης καθορίζονται βασισμένοι στον επιλεγμένο αλγόριθμο προς τα πίσω διάδοσης (back-propagation).

Τώρα, θα προσπαθήσουμε να βρούμε έναν κατάλληλο τύπο, για την KL-απόκλιση μεταξύ της κατανομής $P(Z|X)$ και της κατανομής $Q(Z|X)$. (Carl Doersch 2016. [2])

έχουμε:

$$\mathbf{Q}_\phi(\mathbf{Z}) = N_1 = N(Z|\mu_1, \sigma_1^2) = N(Z|M, \Sigma^2), \text{ όπου: } \mu_1 = M \text{ και } \sigma_1 = \Sigma$$

$$\mathbf{P}(\mathbf{Z}) = N_2 = N(Z|\mu_2, \sigma_2^2) = N(Z|0, I), \text{ όπου: } \mu_2 = 0 \text{ και } \sigma_2 = I$$

ακόμα έχουμε:

$$\int \mathbf{Q}_\phi(\mathbf{Z}) \cdot \log \mathbf{P}(\mathbf{Z})\, \mathbf{dz} = \int N(Z|M, \Sigma^2) \cdot \log N(Z|0, I)\, dz = -\frac{J}{2}\log 2\pi - \frac{1}{2}\cdot\sum_{j=1}^{J}(\mu_j^2 + \sigma_j^2) \Rightarrow$$

$$\int \mathbf{Q}_\phi(\mathbf{Z}) \cdot \log \mathbf{P}(\mathbf{Z})\, \mathbf{dz} = -\frac{J}{2}\log 2\pi - \frac{1}{2}\cdot(M^2 + \Sigma^2)\ (\mathbf{1})$$

και:

$$\int \mathbf{Q}_\phi(\mathbf{Z}) \cdot \log \mathbf{Q}_\phi(\mathbf{Z})\, \mathbf{dz} = \int N(Z|M,\Sigma^2) \cdot \log N(Z|M,\Sigma^2)\, dz = -\frac{J}{2}\log 2\pi - \frac{1}{2}\cdot\sum_{j=1}^{J}(1+\log\sigma_j^2) \Rightarrow$$

$$\int \mathbf{Q}_\phi(\mathbf{Z}) \cdot \log \mathbf{Q}_\phi(\mathbf{Z})\, \mathbf{dz} = -\frac{J}{2}\log 2\pi - \frac{1}{2}\cdot(1+\log\Sigma^2)\ (\mathbf{2})$$

όπου J είναι η διάσταση των λανθανουσών μεταβλητών Z. Οι μέσες τιμές M και οι διασπορές Σ ορίζονται ως εξής:

$$M = \begin{bmatrix} M_{11} & M_{12} & M_{13} & \dots & M_{1J} \\ M_{21} & M_{22} & M_{23} & \dots & M_{2J} \\ & \dots\dots\dots\dots & & & \\ M_{N1} & M_{N2} & M_{N3} & \dots & M_{NJ} \end{bmatrix}, \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} & \Sigma_{13} & \dots & \Sigma_{1J} \\ \Sigma_{21} & \Sigma_{22} & \Sigma_{23} & \dots & \Sigma_{2J} \\ & \dots\dots\dots\dots & & & \\ \Sigma_{N1} & \Sigma_{N2} & \Sigma_{N3} & \dots & \Sigma_{NJ} \end{bmatrix}$$

όπου N είναι ο αριθμός των pixel

Συνεχίζοντας, μπορούμε να υπολογίσουμε την KL-απόκλιση μεταξύ των κατανομών P και Q από τον τύπο του ELBO, ως εξής:

$$D_{KL}[Q_\phi(Z|X)\,||\,P(Z|X)] = D_{KL}[N_1\,||\,N_2] = D_{KL}[N(Z|\mu_1,\sigma_1^2)\,||\,N(Z|\mu_2,\sigma_2^2)] \Rightarrow$$

$$D_{KL}[Q_\phi(Z|X)\,||\,P(Z|X)] = D_{KL}[N(Z|\mu_1,\sigma_1^2)\,||\,N(Z|0,I)] \Rightarrow$$

$$D_{KL}[Q_\phi(Z|X)\,||\,P(Z|X)] = \int Q_\phi(Z)\cdot\log\frac{P(Z)}{Q_\phi(Z)}\,dz \Rightarrow$$

$$D_{KL}[Q_\phi(Z|X)\,||\,P(Z|X)] = \int Q_\phi(Z)\cdot(\log P(Z) - \log Q_\phi(Z))\,dz \Rightarrow$$

$$D_{KL}[Q_\phi(Z|X)\,||\,P(Z|X)] = \int Q_\phi(Z)\cdot\log P(Z) - Q_\phi(Z)\cdot\log Q_\phi\,dz \Longrightarrow_{(\mathbf{1}),(\mathbf{2})}$$

$$D_{KL}[Q_\phi(Z|X)\,||\,P(Z|X)] = -\frac{J}{2}\log 2\cdot\pi - \frac{1}{2}\cdot\sum_{j=1}^{J}(\mu_j^2+\sigma_j^2) - (-\frac{J}{2}\log 2\cdot\pi - \frac{1}{2}\cdot\sum_{j=1}^{J}(1+\log\sigma_j^2)) \Rightarrow$$

$$D_{KL}[Q_\phi(Z|X)\,||\,P(Z|X)] = -\frac{J}{2}\log 2\cdot\pi + \frac{J}{2}\cdot\log 2\cdot\pi - \frac{1}{2}\cdot\sum_{j=1}^{J}(\mu_j^2+\sigma_j^2) + \frac{1}{2}\cdot\sum_{j=1}^{J}(1+\log\sigma_j^2) \Rightarrow$$

$$D_{KL}[Q_\phi(Z|X)\,||\,P(Z|X)] = \frac{1}{2}\cdot\sum_{j=1}^{J}(1+\log\sigma_j^2-\mu_j^2-\sigma_j^2) \Rightarrow$$

$$D_{KL}[Q_\phi(Z|X)\,||\,P(Z|X)] = \frac{1}{2}\cdot(J+\log\Sigma^2-M^2-\Sigma^2)$$

αν η διάσταση της παραμέτρου $J = 1$, των λανθανουσών μεταβλητών Z, αυτό σημαίνει ότι έχουμε ενιαίες (univariate) κατανομές Gauss και καταλήγουμε στον εξής τύπο:

$$\mathbf{D_{KL}[Q_\phi(Z|X)\,||\,P(Z|X)] = \frac{1}{2}\cdot(1+\log\Sigma^2-M^2-\Sigma^2)}$$

Ας θυμηθούμε ότι ο όρος της KL-απόκλισης έχει αρνητικό πρόσημο στον τύπο του variational lower bound (ELBO), επομένως θέλουμε να τον ελαχιστοποιήσουμε.

3.2 To Reparameterization Trick

(Carl Doersch 2016. [2]) Έχοντας καταλήξει σε έναν κατάλληλο τύπο για τον όρο της KL-απόκλισης, μένει ακόμα να τρέξουμε τον αλγόριθμο στοχαστικής (ή mini-batch) καθοδικής κλίσης (gradient descent) για διάφορες τιμές του X, παίρνοντας δείγματα από ένα dataset, D. Η πλήρης εξίσωση που θέλουμε να βελτιστοποιήσουμε είναι η εξής:

$$E_{X\sim D}[E_{Z\sim Q}[logP_\theta(X|Z)] - D_{KL}[Q_\phi(Z)\,||\,P(Z)]]\ (\mathbf{1})$$

Θέλουμε να βρούμε την παράγωγο αυτής της εξίσωσης. Το σύμβολο της παραγώγου μπορεί να μετακινηθεί εντός των μέσων τιμών. Επομένως, μπορούμε να δειγματοληπτήσουμε μία μόνο τιμή του X και μία μόνο τιμή του Z από την κατανομή $Q(Z|X)$, και να υπολογίσουμε την παράγωγο της εξίσωσης:

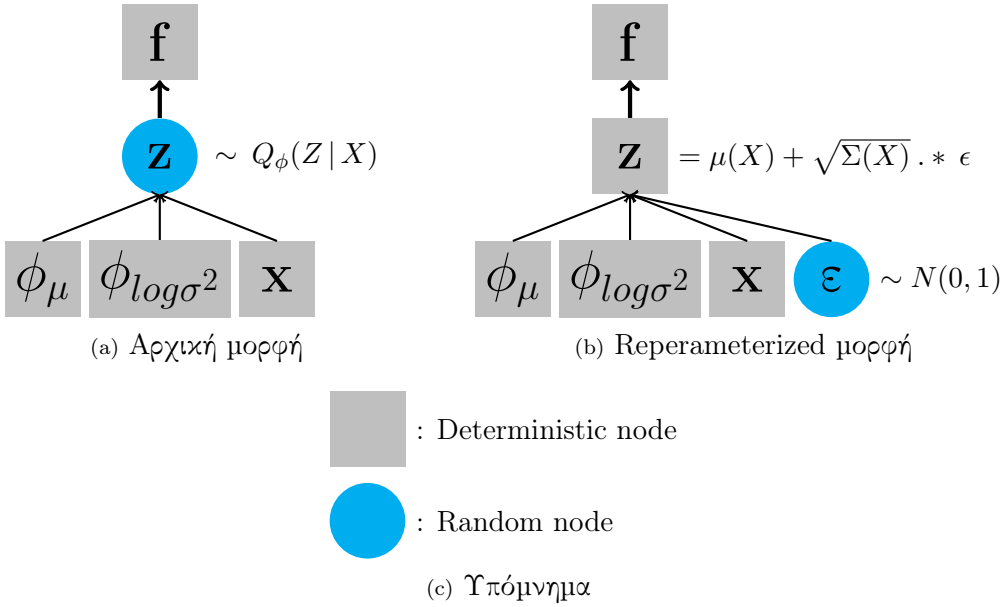
$$logP_\theta(X|Z) - D_{KL}[Q_\phi(Z)\,||\,P(Z)](\mathbf{2})$$

Στη συνέχεια, μπορούμε να πάρουμε τη μέση τιμή της παραγώγου αυτής της συνάρτησης, για αυθαίρετα πολλά δείγματα X και Z και το αποτέλεσμα θα συγκλίνει στην παράγωγο της Συνάρτησης 1. Ωστόσο, υπάρχει ένα σημαντικό πρόβλημα με την Εξίσωση 2. Ο όρος $E_{Z\sim Q}[logP_\theta(X|Z)]$ δεν εξαρτάται μόνο

από τις παραμέτρους P, αλλά και από τις παραμέτρους Q. Εν τούτοις, στην Εξίσωση 2, η εξάρτηση αυτή έχει εξαφανιστεί! Για να δουλέψουν οι VAE, είναι ουσιώδες να οδηγήσουμε την κατανομή Q να παράγει κωδικοποιήσεις για την X, τις οποίες η P μπορεί αξιόπιστα να αποκωδικοποιήσει. Το προς τα εμπρός πέρασμα του δικτύου (forward pass) δουλεύει σωστά και παράγει τη σωστή μέση τιμή, αν η έξοδος (output) υπολογίζεται κατά μέσο όρο πολλών δειγμάτων X και Z. Ωστόσο, πρέπει να προωθήσουμε προς τα πίσω (back-propagate) το σφάλμα μέσω ενός επιπέδου που δειγματοληπτεί τη Z μέσω της κατανομής $Q(Z|X)$, η οποία είναι μη συνεχής διαδικασία και δεν έχει παράγωγο. Ο στοχαστικός αλγόριθμος καθοδικής κλίσης (stochastic gradient descent) μέσω της προς τα πίσω διάδοσης (back-propagation) μπορεί να χειριστεί στοχαστικές εισόδους (inputs), αλλά δεν μπορεί να χειριστεί μονάδες εντός του επιπέδου εισόδου (input layer). Δεδομένου των $\mu(X)$ και $\Sigma(X)$ - η μέση τιμή και η συνδιακύμανση της κατανομής $Q(Z|X)$ - μπορούμε να δειγματοληπτήσουμε από την κανονική κατανομή $N(\mu(X), \Sigma(X))$, δειγματοληπτώντας πρώτα από την $\epsilon \sim N(0, I)$. Τελικά υπολογίζουμε τη μεταβλητή $Z = \mu(X) + \sqrt{\Sigma(X)} \,.*\, \epsilon$, που ακολουθεί κανονική κατανομή, $Z \sim N(\mu(X), \Sigma(X))$, εφόσον, κάθε γραμμικός μετασχηματισμός μιας Gaussian T.M. (τυχαίας μεταβλητής) είναι πάλι Gaussian (συμβουλευτείτε το Παράρτημα B, Κατανομή Gauss (ή Κανονική Κατανομή)). Επομένως, η εξίσωση της οποίας υπολογίζουμε την παράγωγο είναι η εξής:

$$E_{X\sim D}[E_{Z\sim Q}[logP_\theta(X|Z = \mu(X) + \sqrt{\Sigma(X)} \cdot \epsilon)] - D_{KL}[Q_\phi(Z) \,||\, P(Z)]]$$

Είναι σημαντικό να προσέξουμε ότι η κατανομή $Q(Z|X)$ (συνεπώς και η $P(Z)$) πρέπει να είναι συνεχής! Το reparameterization trick μας επιτρέπει να υπολογίσουμε την παράγωγο της μέσης τιμής του ELBO, έτσι ώστε το back-propagation να μπορεί να εφαρμοστεί.



(a) Αρχική μορφή

(b) Reperameterized μορφή

(c) Υπόμνημα

Εικονα 3.1: Το δεξιά δίκτυο χρησιμοποιεί το reparameterization trick. Το back-propagation μπορεί να εφαρμοστεί μόνο σε αυτό το δίκτυο.

3.3 Υπολογίζοντας τους κανόνες ενημέρωσης βαρών του κωδικοποιητή και του αποκωδικοποιητή

Στις υλοποιήσεις που παρουσιάζονται σε αυτή τη διατριβή, οι κανόνες ενημέρωσης των βαρών (Phis) του κωδικοποιητή και τα βάρη (Thetas) του αποκωδικοποιητή υπολογίζονται αυτόματα. Οι βιβλιοθήκες TensorFlow και PyTorch περιλαμβάνουν ενσωματωμένους αλγορίθμους προς τα πίσω διάδοσης (back-propagation) που υπολογίζουν τις παραγώγους για τα βάρη που χρησιμοποιούνται για τη διαδικασία της μάθησης. Οι κανόνες ενημέρωσης, επίσης περιλαμβάνονται ως ενσωματωμένες συναρτήσεις και εφαρμόζονται στη διαδικασία χρησιμοποιώντας τις παραγώγους που υπολογίστηκαν προηγουμένως.

Για να κάνουμε τη διαδικασία πιο ξεκάθαρη, ακολουθεί παράδειγμα κλήσης του ενσωματωμένου αλγορίθμου προς τα πίσω διάδοσης και της συνάρτησης ενημέρωσης βαρών σε TensorFlow:

```
var_list  # the weights and the biases of the variational autoencoder
elbo  # the loss function of the variational autoencoder to be minimized
lr  # learning rate for the weights and the biases updates

# Adam Optimizer #
grads_and_vars = tf.train.AdamOptimizer(learning_rate=lr).
        compute_gradients(loss=elbo, var_list=var_list)
apply_updates = tf.train.AdamOptimizer(learning_rate=lr).
        apply_gradients(grads_and_vars=grads_and_vars)
```

ΚΩΔΙΚΑΣ 3.1: προς τα πίσω διάδοση σε TensorFlow

Ακολουθεί παράδειγμα κλήσης του ενσωματωμένου αλγορίθμου προς τα πίσω διάδοσης και της συνάρτησης ενημέρωσης βαρών σε PyTorch. Ας σημειώσουμε ότι και οι δύο διεργασίες εκτελούνται με την ίδια εντολή:

```
params  # the weights and the biases of the variational autoencoder
lr  # learning rate for the weights and the biases updates
solver = optim.Adam(params, lr=lr)  # Adam Optimizer #
elbo_loss.backward()  # Backward #
solver.step()  # Update #
for p in params:  # Housekeeping #
    # initialize the parameter gradients of the next epoch as zeros
    p.grad.data.zero_()
```

ΚΩΔΙΚΑΣ 3.2: προς τα πίσω διάδοση σε PyTorch

Για οδηγίες σχετικά με το πως εφαρμόζονται οι κανόνες ενημέρωσης, συμβουλευτείτε το Παράρτημα Α, στην ενότητα A.11 Κανόνες Ενημέρωσης Βαρών των Νευρωνικών Δικτύων.

ΣΗΜΕΙΩΣΗ: Όπως εξηγήσαμε στο προηγούμενο κεφάλαιο, θέλουμε να μεγιστοποιήσουμε το variational ελάχιστο φράγμα (ELBO). Ωστόσο, στις υλοποιήσεις που παρουσιάζονται σε αυτή τη διπλωματική εργασία, το ELBO γίνεται όλο και μικρότερο σε κάθε εποχή (epoch). Ο λόγος που αυτό συμβαίνει είναι επειδή χρησιμοποιούμε αλγόριθμο καθοδικής κλίσης (gradient descent) αντί για αλγόριθμο ανοδικής κλίσης (gradient ascent).

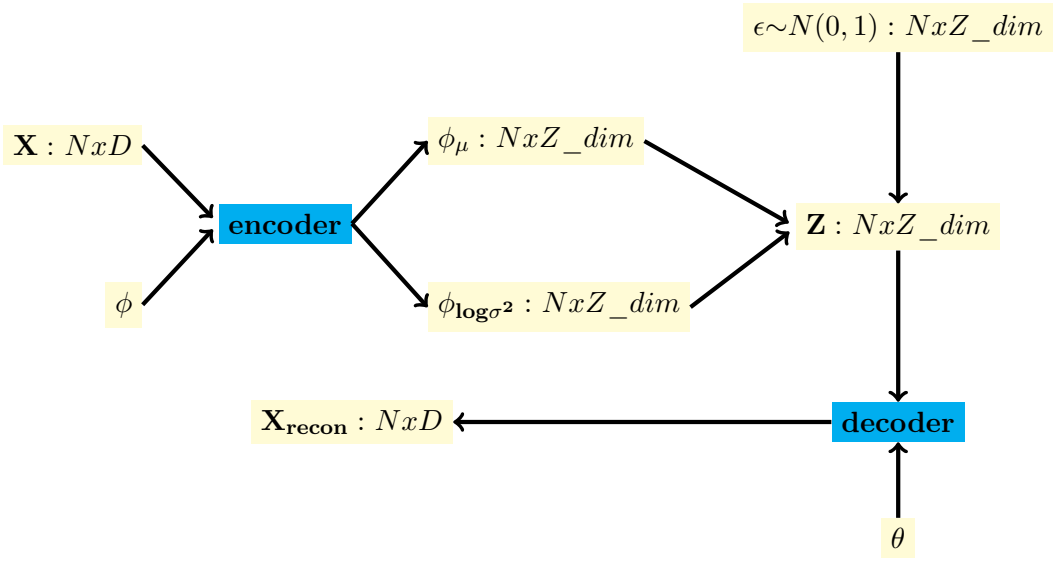
Κεφάλαιο 4

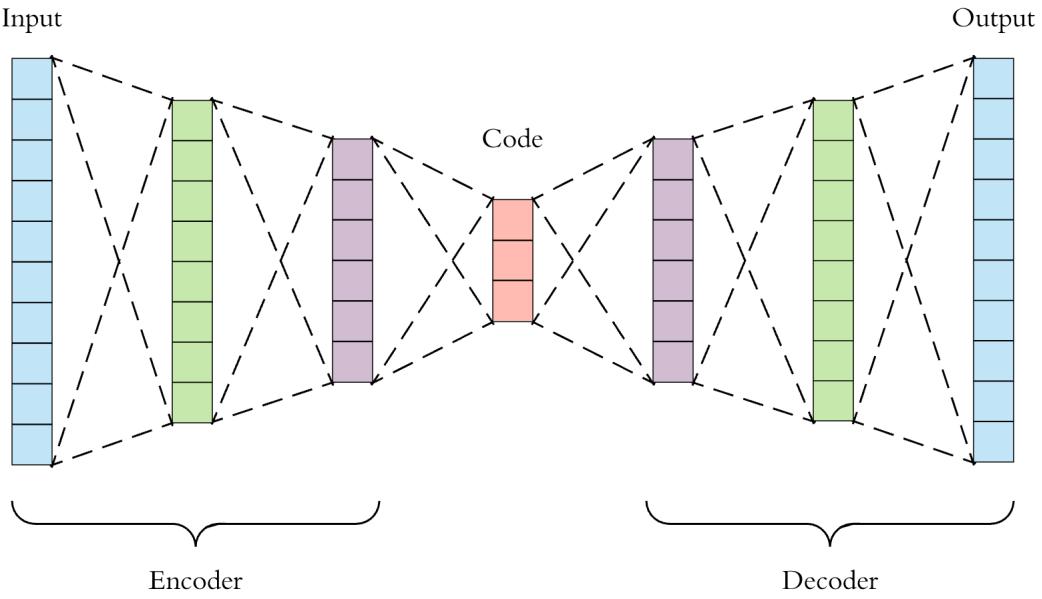
Δομή του Variational Autoencoder & Αποτελέσματα Πειραμάτων

4.1 Δομή του Variational Autoencoder

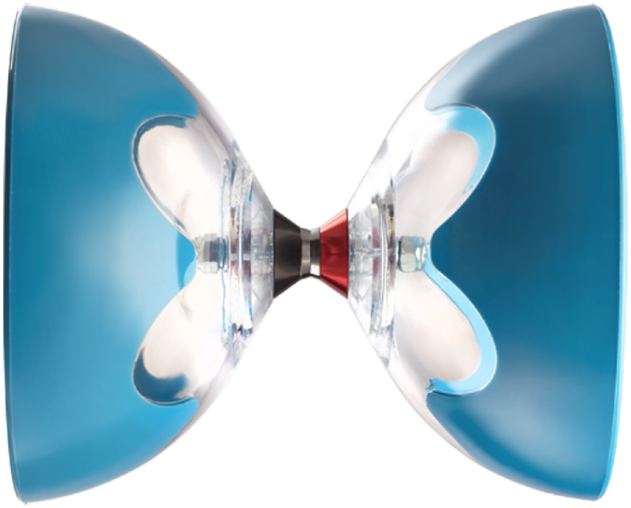
Συνήθως, ο variational autoencoder χρειάζεται πολλές εποχές (epochs) για να εκπαιδευτεί. Ωστόσο, υπάρχουν περιπτώσεις όπου πάρα πολλές εποχές μπορεί να επιφέρουν μη επιθυμητά αποτελέσματα, πχ. θολές εικόνες. Όταν το ελάχιστο φράγμα (ELBO) γίνει πολύ μικρό, ο VAE θα πρέπει να σταματήσει την εκπαίδευση, γιατί στην επόμενη εποχή ή επανάληψη, η τιμή του ELBO θα γίνει τελικά πολύ μεγάλη. Σε έναν autoencoder, ο αριθμός των νευρώνων του κωδικοποιητή (encoder), τον οποίο συμβολίζουμε ως M_1, πρέπει να είναι ίσος με τον αριθό των νευρώνων του αποκωδικοποιητή (decoder), τον οποίο συμβολίζουμε ως M_2.

Το παρακάτω διάγραμμα απεικονίζει τα βήματα που ο variational autoencoder ακολουθεί για να κατασκευάσει τα τεχνητά δεδομένα, $\mathbf{X_{recon}}$, από τα πρωτότυπα δεδομένα, $\mathbf{X}$. Μπορούμε να εξετάσουμε πώς τα αρχικά δεδομένα, $\mathbf{X}$, μετατρέπονται στα λανθάνοντα δεδομένα, $\mathbf{Z}$, τα οποία είναι μια αναπαράσταση των μεταβλητών $\mathbf{X}$ στην ελαττωμένη διάσταση $\mathbf{Z_{dim}}$:





(a) Δομή του autoencoder



(b) Ένα ζογκλερικό παιχνίδι diabolo

Εικονα 4.1: Ο autoencoder ονομάζεται και δίκτυο Diabolo λόγω της εμφάνισης της δομής του. (Rumelhart et al., 1986a; Bourlard & Kamp, 1988; Hinton & Zemel, 1994; Schwenk & Milgram, 1995; Japkowicz, Hanson, & Gluck, 2000, Yoshua Bengio 2009. [1])

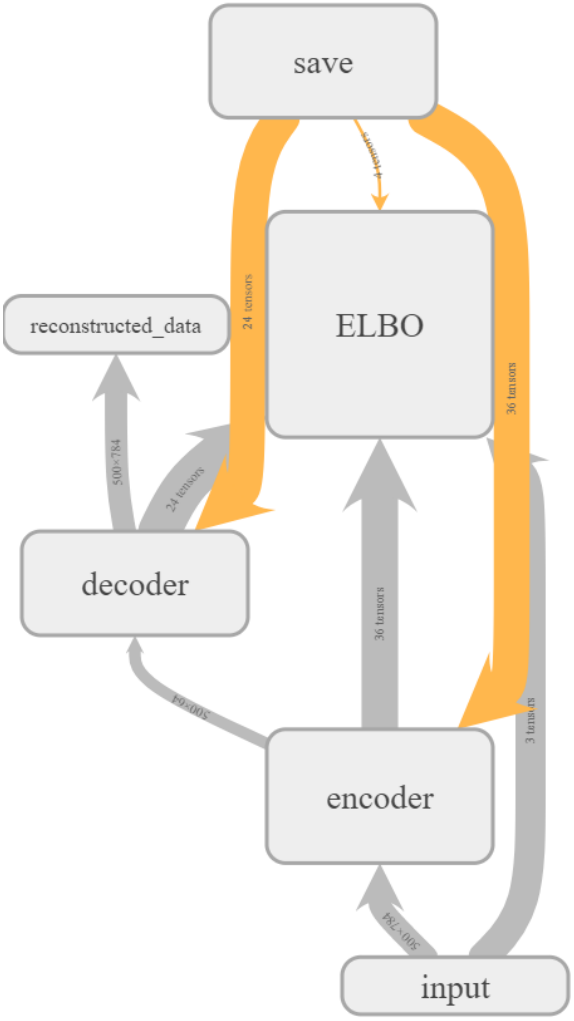
4.2 Το TensorBoard

Για προγραμματιστικές υλοποιήσεις των variational autoencoder σε TensorFlow, PyTorch και Keras δείτε το Παράρτημα Γ. Ανοίξτε ένα τερματικό (terminal), για Unix/Linux, ή τη γραμμή εντολών (command prompt), για Windows, και τρέξτε την ακόλουθη εντολή:

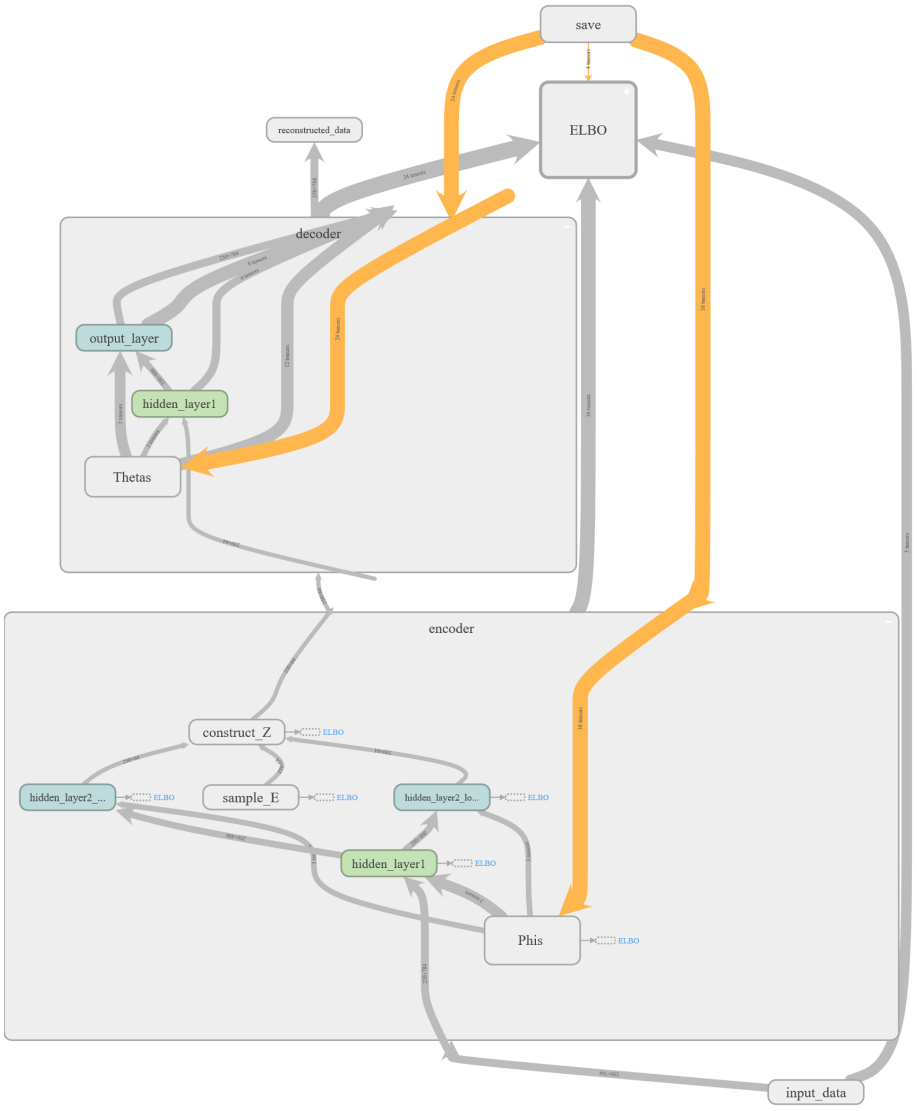
```
tensorboard --logdir = "path_to_the_graph"
```

ΚΩΔΙΚΑΣ 4.1: εντολή για έναρξη του TensorBoard

Μετά, ένα μήνυμα θα πρέπει να εμφανιστεί που θα αναφέρεται στο ακόλουθο URL: http://localhost:6006. Ανοίξτε ένα φυλλομετρητή (πχ. Firefox) και πλοηγηθείτε στο URL. Το TensorFlow τώρα τρέχει ως εφαρμογή δικτύου (web), από προεπιλογή στο port 6006.



ΕΙΚΟΝΑ 4.2: Αυτή είναι απεικόνιση της υλοποίησης σε TensorFlow, με το TensorBoard.



Εικονα 4.3: Αυτή είναι διευρυμένη απεικόνιση της υλοποίησης σε TensorFlow, με το TensorBoard.

ΣΗΜΕΙΩΣΗ: Το port "6006" στο URL `http://localhost:6006` είναι το "goog" ανάποδα.

4.3 Σύνολα Δεδομένων (Datasets)

Ακολουθεί η κατάσταση όλων των συνόλων δεδομένων (dataset) που χρησιμοποιήθηκαν σε αυτή τη διπλωματική διατριβή:

Dataset	# TRAIN	# TEST	# VALIDATION
MNIST [5]	55000	10000	5000
Binarized MNIST	50000	10000	5000
CIFAR-10 [6]	50000	10000	-
OMNIGLOT	390 [En] / 360 [Gr]	130 [En] / 120 [Gr]	-
Cropped YALE Faces [7]	2442	-	-
ORL Face Database	400	-	-
MovieLens 100k	90570	9430	-

Πινακας 4.1: Λεπτομέρειες των dataset.

Dataset	# Classes	Dimensions	Type of Values
MNIST [5]	10	28x28 pixels	*real values in* $[0, 1]$
Binarized MNIST	10	28x28 pixels	$\{0, 1\}$
CIFAR-10 [6]	10	32x32x3 [RGB] / 32x32x1 [Grayscaled] pixels	*real values in* $[0, 1]$
OMNIGLOT	26 [En] / 24 [Gr]	32x32 pixels	*real values in* $[0, 1]$
Cropped YALE Faces [7]	38	168x192 pixels	*real values in* $[0, 255]$
ORL Face Database	40	92x112 pixels	*real values in* $[0, 255]$
MovieLens 100k	943 users	1682 movies	$\{1, 2, 3, 4, 5\}$

Πινακας 4.2: Πλήθος κατηγοριών, διαστάσεις και είδη τιμών των dataset.

Dataset	URL
MNIST [5]	http://yann.lecun.com/exdb/mnist
Binarized MNIST	https://github.com/yburda/iwae/tree/master/datasets/BinaryMNIST
CIFAR-10 [6]	https://www.cs.toronto.edu/~kriz/cifar.html
OMNIGLOT	https://github.com/yburda/iwae/tree/master/datasets/OMNIGLOT
Cropped YALE Faces [7]	http://vision.ucsd.edu/extyaleb/CroppedYaleBZip/CroppedYale.zip
ORL Face Database	http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
MovieLens 100k	https://grouplens.org/datasets/movielens

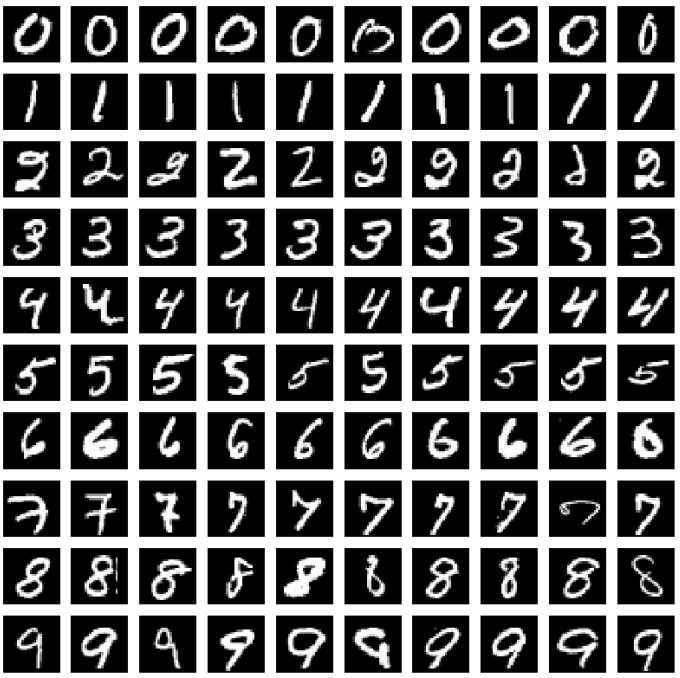
Πινακας 4.3: Σύνδεσμοι για τα dataset.

ΣΗΜΕΙΩΣΕΙΣ:

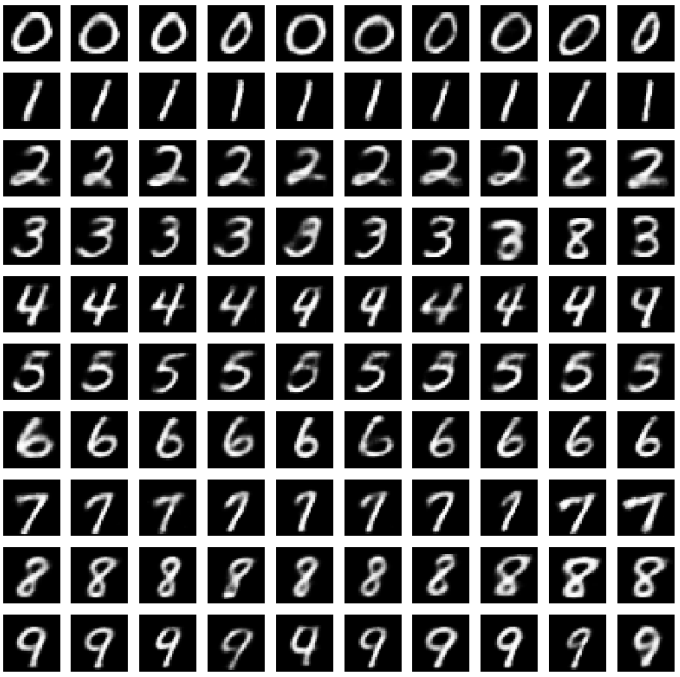
- Ο σύνδεσμος για το dataset YALE faces πιθανόν να μην είναι έγκυρος.
- Το dataset "ORL Face Database" περιλαμβάνει πολύ λίγα παραδείγματα και ως εκ τούτου τα αποτελέσματα είναι ανακριβή. Δεν συνιστούμε αυτό το dataset στα πειράματά μας. Ωστόσο, το dataset υπόσχεται καλά αποτελέσματα μόνο στον K-NN αλγόριθμο συμπλήρωσης ελλιπών τιμών.
- Όλα τα dataset που παίρνουν τιμές στο διάστημα $[0, 255]$ κανονικοποιούνται στο διάστημα $[0, 1]$, για ευκολότερους υπολογισμούς στη διαδικασία εκπαίδευσης.

4.4 Αποτελέσματα Πειραμάτων

Ακολουθούν τα αποτελέσματα του VAE αλγορίθμου, με χρήση **TensorFlow**, στο **MNIST** dataset:



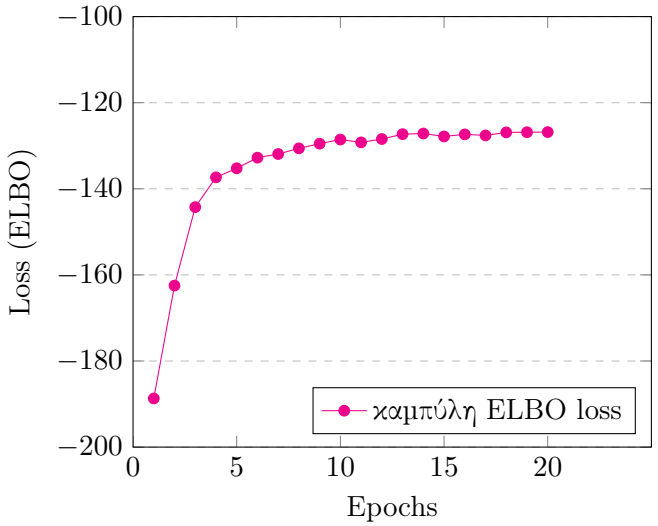
(a) Αρχικά Δεδομένα



(b) VAE Εποχή 20

Εικονα 4.4: MNIST VAE σε TensorFlow.
root mean squared error: 0.142598

ELBO σε κάθε εποχή (epoch), VAE στο MNIST dataset σε TensorFlow

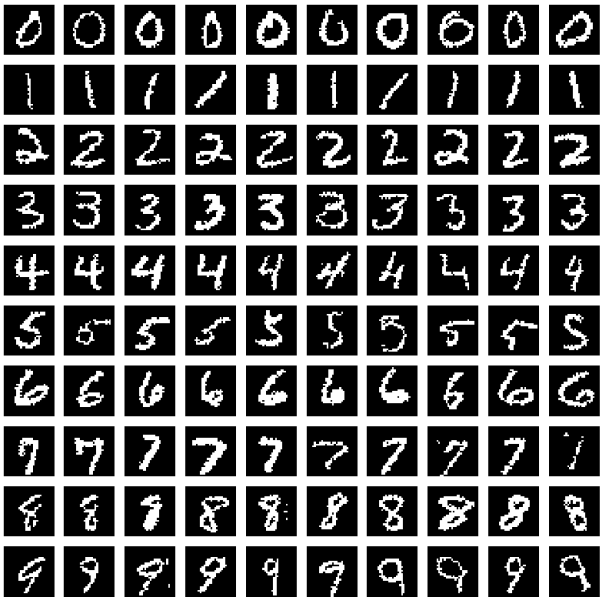


Metric	Value
last epoch loss (ELBO)	-126.83
root mean squared error (RMSE)	0.17121448655010216
mean absolute error (MAE)	0.07299246278044173

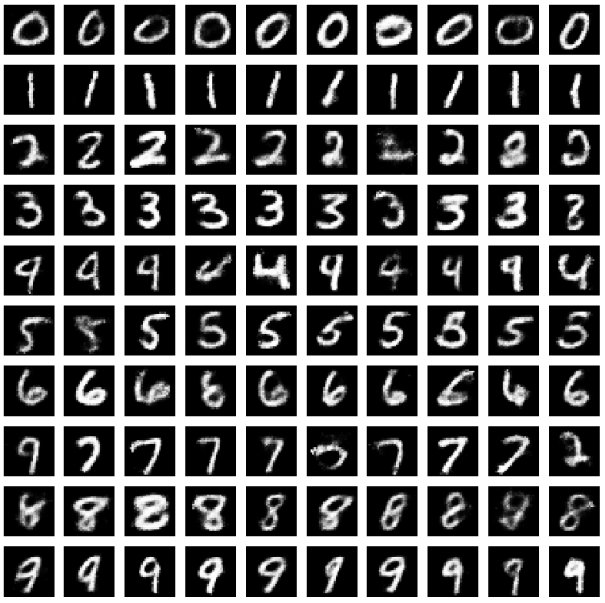
Πινακας 4.4: VAE στο MNIST dataset σε TensorFlow.

Όπως εξηγήσαμε στο Κεφάλαιο 3, Ενότητα 2, το ELBO ελαχιστοποιείται σε κάθε εποχή αντί να μεγιστοποιείται, επειδή χρησιμοποιήσαμε αλγόριθμο καθοδικής κλίσης (gradient descent), αντί για αλγόριθμο ανοδικής κλίσης (gradient ascent). Επίσης, μπορούμε να συμπεράνουμε ότι το MAE είναι καλύτερος δείκτης από το RMSE, επειδή το RMSE είναι μικρότερο του 1. Αυτό σημαίνει ότι το τετράγωνό του (δηλαδή το MSE) είναι μικρότερο από το RMSE και επίσης μικρότερο από 1.

Ακολουθούν αποτελέσματα του VAE αλγορίθμου, χρησιμοποιώντας **TensorFlow**, στο **Binarized MNIST** dataset:



(a) Αρχικά Δεδομένα



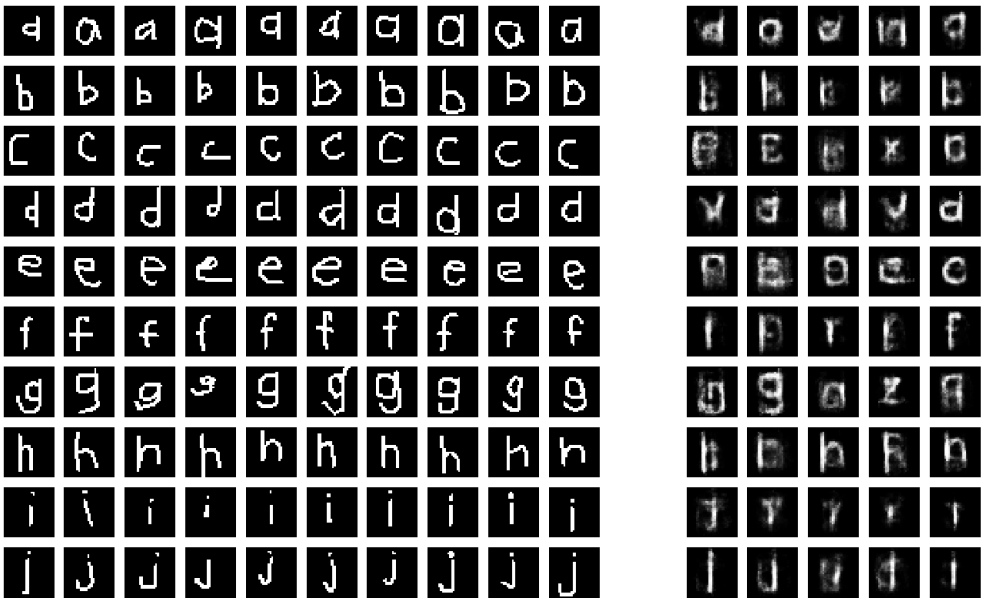
(b) VAE Εποχή 50

Εικονα 4.5: MNIST VAE σε TensorFlow.

Metric	Value
last epoch loss (ELBO)	-109.82
root mean squared error (RMSE)	0.186880795395
mean absolute error (MAE)	0.0685835682327

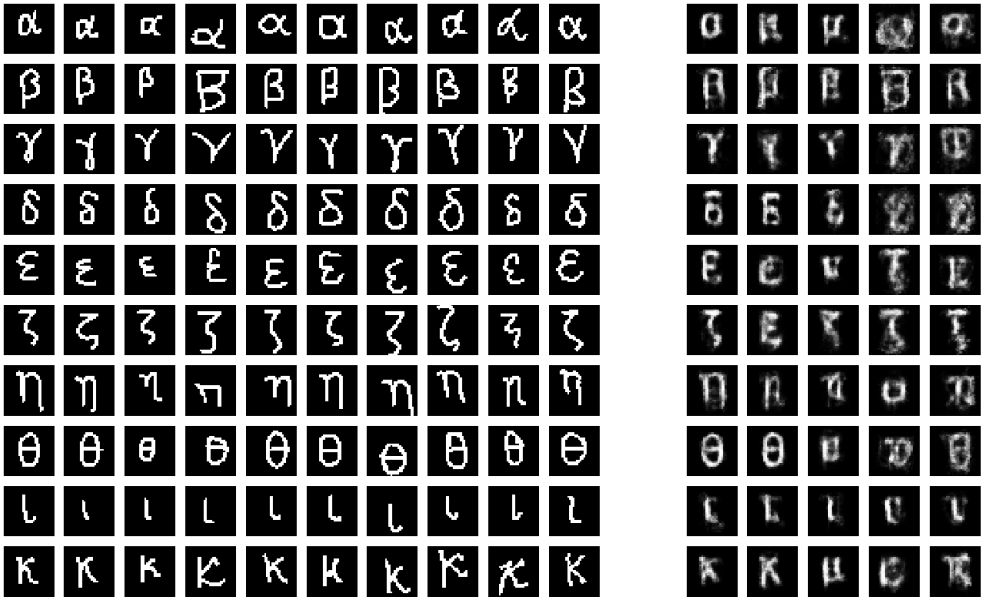
Πινακας 4.5: VAE στο Binarized MNIST dataset σε TensorFlow.

Ακολουθούν αποτελέσματα του VAE αλγορίθμου, με χρήση **PyTorch**, στο **OMNIGLOT** dataset:



(a) Αρχικά Δεδομένα (b) VAE Εποχή 100

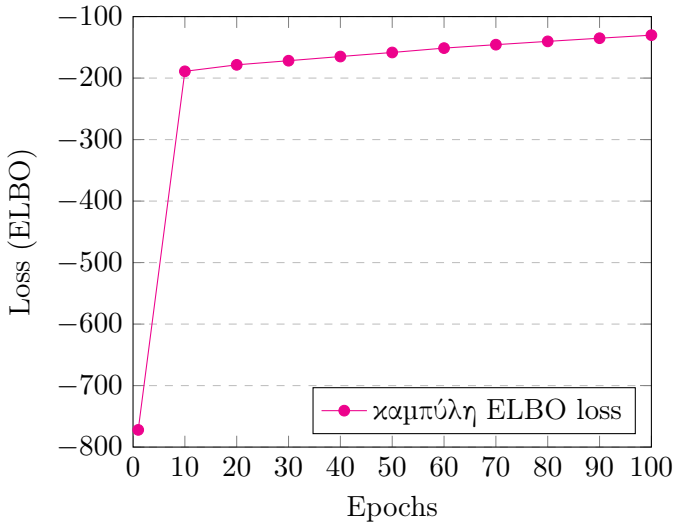
Εικονα 4.6: OMNIGLOT Αγγλικό αλφάβητο,
χαρακτήρες 1-10, αρχικοί και ανακατασκευασμένοι.



(a) Αρχικά Δεδομένα (b) VAE Εποχή 100

Εικονα 4.7: OMNIGLOT Ελληνικό αλφάβητο,
χαρακτήρες 1-10, αρχικοί και ανακατασκευασμένοι.

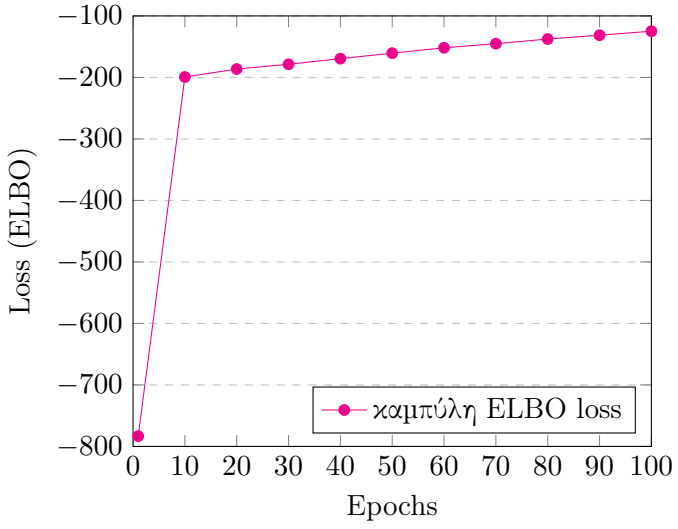
ELBO σε κάθε εποχή (epoch), VAE στο OMNIGLOT Αγγλικό dataset, σε PyTorch



Metric	Value
last epoch loss (ELBO)	-130.22
root mean squared error (RMSE)	0.2134686176531402
mean absolute error (MAE)	0.09105986614619989

Πινακας 4.6: VAE στο OMNIGLOT Αγγλικό dataset σε PyTorch.

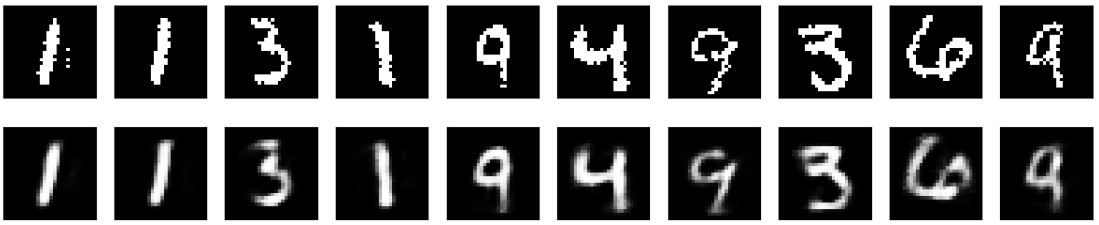
ELBO σε κάθε εποχή (epoch), VAE στο OMNIGLOT Ελληνικό dataset, σε PyTorch



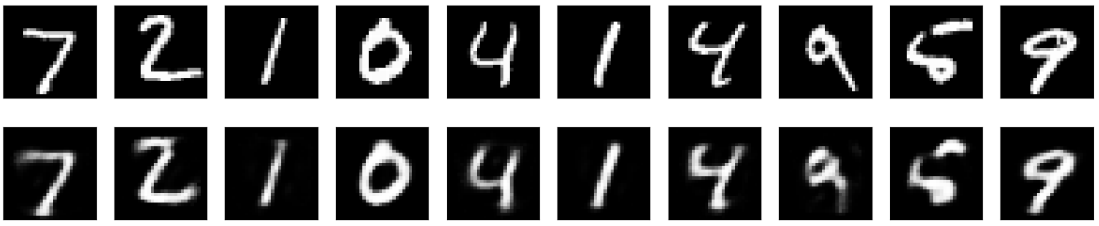
Metric	Value
last epoch loss (ELBO)	-124.85
root mean squared error (RMSE)	0.20157381435292054
mean absolute error (MAE)	0.08379960438030741

Πινακας 4.7: VAE στο OMNIGLOT Ελληνικό dataset σε PyTorch.

Ακολουθούν αποτελέσματα του VAE αλγορίθμου, με χρήση του **Keras**, πάνω στα **Binarized MNIST**, **MNIST**, **CIFAR-10** και **OMNIGLOT** dataset:

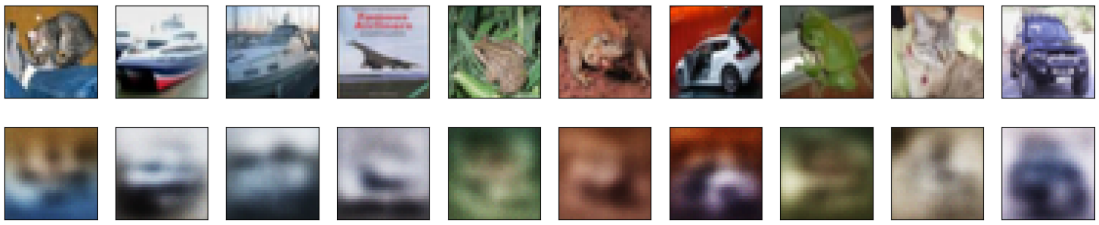


(a) Binarized MNIST

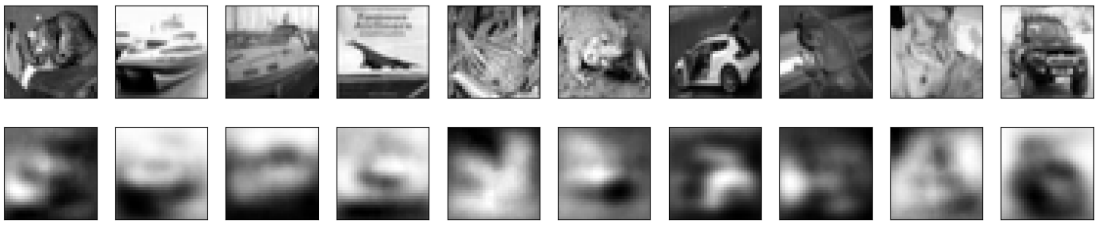


(b) MNIST

Εικονα 4.8: Binarized MNIST και (Real-valued) MNIST ψηφία, αρχικά και ανακατασκευασμένα.

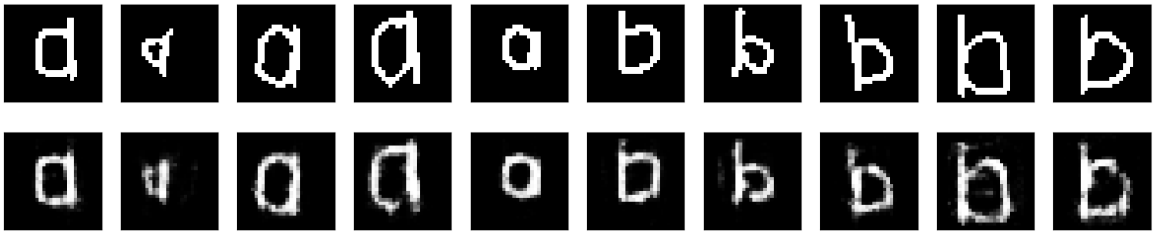


(a) CIFAR-10 RGB

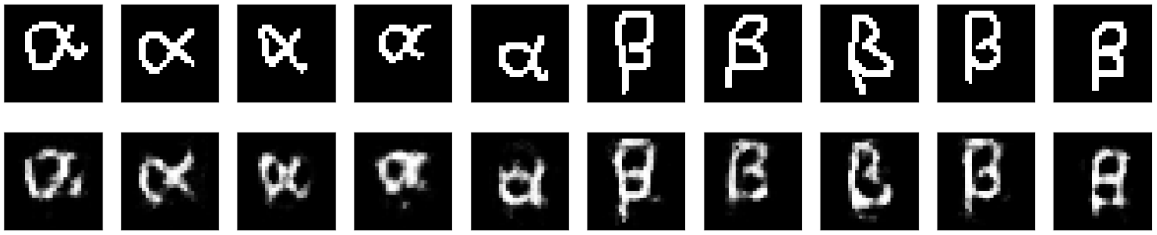


(b) CIFAR-10 Grayscale

Εικονα 4.9: CIFAR-10 εικόνες, RGB και grayscale, αρχικές και ανακατασκευασμένες.



(a) OMNIGLOT Αγγλικό αλφάβητο



(b) OMNIGLOT Ελληνικό αλφάβητο

Εικονα 4.10: OMNIGLOT Αγγλικοί και Ελληνικοί χαρακτήρες, αρχικοί και ανακατασκευασμένοι.

ΣΗΜΕΙΩΣΗ: Οι εικόνες στο CIFAR-10 RGB dataset περιλαμβάνουν pixel με τρία κανάλια (κόκκινο, πράσινο και μπλε) τα οποία παίρνουν πραγματικές τιμές, επομένως ο VAE δεν αναμένεται να έχει ικανοποιητικά αποτελέσματα σε αυτές.

Metric	Value
last epoch loss (ELBO)	0.0635
root mean squared error (RMSE)	0.1361456340713906
mean absolute error (MAE)	0.03993093576275391

Πινακας 4.8: VAE στο Binarized MNIST dataset, σε Keras.

Metric	Value
last epoch loss (ELBO)	0.0040
root mean squared error (RMSE)	0.00102286
mean absolute error (MAE)	0.00063948194

Πινακας 4.9: VAE στο MNIST dataset, σε Keras.

Metric	Value
last epoch loss (ELBO)	0.6198
root mean squared error (RMSE)	0.17352526
mean absolute error (MAE)	0.1368697

Πινακας 4.10: VAE στο CIFAR-10 RGB dataset, σε Keras.

Metric	Value
last epoch loss (ELBO)	0.6166
root mean squared error (RMSE)	0.14963366
mean absolute error (MAE)	0.11517575

Πινακας 4.11: VAE στο CIFAR-10 Grayscale dataset, σε Keras.

Metric	Value
last epoch loss (ELBO)	0.1764
root mean squared error (RMSE)	0.23234108227525616
mean absolute error (MAE)	0.11663868188603484

Πινακας 4.12: VAE στο OMNIGLOT Αγγλικό dataset, σε Keras.

Metric	Value
last epoch loss (ELBO)	0.1840
root mean squared error (RMSE)	0.23350568189372273
mean absolute error (MAE)	0.1205574349168227

Πινακας 4.13: VAE στο OMNIGLOT Ελληνικό dataset, σε Keras.

ΠΑΡΑΤΗΡΗΣΕΙΣ:

- Τα ELBO loss που υπολογίστηκαν από το VAE σε Keras είναι σημαντικά χαμηλότερες από τις απώλειες που επέφεραν οι υλοποιήσεις σε TensorFlow και PyTorch.
- Οι εικόνες που ανακατασκευάστηκαν από το VAE με το MNIST dataset είναι πιο κοντά στα αρχικά δεδομένα από τις εικόνες που ανακατασκευάστηκαν με το Binarized MNIST dataset. Ο δείκτης σφάλματος (RMSE, MAE) όπως και το ELBO για το MNIST dataet είναι πολύ κοντά στο 0, πράγμα που σημαίνει ότι ο VAE έχει κάνει overfit στα δεδομένα εκπαίδευσης. Αυτό το αποτέλεσμα δεν ήταν αναμενόμενο καθώς ο VAE θα έπρεπε να συμπεριφέρεται καλύτερα στα δυαδικά δεδομένα. Η εξήγηση γι' αυτό πρέπει να αναζητηθεί στη λειτουργία της βιβλιοθήκης Keras και στους αλγόριθμους που χρησιμοποιεί για να υπολογίσει τη συνάρτηση απώλειας.
- Τα αποτελέσματα στο CIFAR-10 dataset είναι καλύτερα για τις Grayscaled εικόνες αντί των RGB εικόνων.
- Τα αποτελέσματα στο OMNIGLOT dataset είναι ελαφρώς καλύτερα για το dataset της Αγγλικής γλώσσας έναντι του dataset της Ελληνικής γλώσσας, κυρίως λόγω έλλειψης σε δεδομένα. Υπάρχουν 520 εικόνες Αγγλικών χαρακτήρων και 480 εικόνες Ελληνικών χαρακτήρων.

Κεφάλαιο 5

Αλγόριθμοι Συμπλήρωσης Ελλιπών Τιμών & Variational Autoencoders

5.1 Ένας Απλός Αλγόριθμος Συμπλήρωσης Ελλιπών Τιμών Χρησιμοποιώντας K-NN Συνεργατικό Φιλτράρισμα (CF)

Ο παραδοσιακός αλγόριθμος για ένα σύστημα συστάσεων (recommendation system) είναι το συνεργατικό φιλτράρισμα (collaborative filtering). Μια πολύ συνηθισμένη εφαρμογή για αυτή τη μέθοδο είναι η πρόβλεψη βαθμολογίας ενός χρήστη για ταινίες που δεν έχει ακόμα βαθμολογήσει, με βάση την ομοιότητα μεταξύ των βαθμολογιών του χρήστη για άλλες ταινίες και τις βαθμολογίες άλλων χρηστών. Η ομοιότητα συνημιτόνων (cosine similarity) είναι ένας τρόπος να μετρήσουμε την ομοιότητα μεταξύ χρηστών. Στον αλγόριθμο αυτό, θα χρησιμοποιήσουμε Ευκλίδειες αποστάσεις. Το **MovieLens** dataset είναι το πιο δημοφιλές σε σχετικές περιπτώσεις. Το **MNIST** dataset μπορεί επίσης να είναι μια κατάλληλη εφαρμογή του αλγορίθμου, με μικρές τροποποιήσεις.

Μπορούμε να εφαρμόσουμε συνεργατικό φιλτράρισμα στο **MNIST** dataset, χρησιμοποιώντας μια παραλλαγή του K-NN (K Nearest Neighbors) αλγορίθμου για regression, ως εξής:

1. Αποθηκεύουμε το MNIST dataset **X_train** με εικόνες από ψηφία στη μνήμη. Οι διαστάσεις των δεδομένων εκπαίδευσης, **X_train** είναι NxD, όπου N είναι ο αριθμός των εικόνων εκπαίδευσης και D είναι ο αριθμός των pixel σε κάθε εικόνα. Για το MNIST dataset: $D = 784$. Επίσης, αποθηκεύουμε τα δεδομένα ελέγχου σε μια μεταβλητή, **X_test**.
2. Τροποποιούμε τα δεδομένα εκπαίδευσης (train) και ελέγχου (test), εισάγοντας ελλιπείς τιμές (missing values). Σκεφτήκαμε δύο τρόπους να επιλέξουμε και να κατασκευάσουμε τις ελλιπείς τιμές. Ένας τρόπος είναι να αντικαταστήσουμε όλα τα δεξιά, αριστερά, επάνω και κάτω pixel ή καθόλου pixel σε κάθε εικόνα με τιμές που δείχνουν ότι τα pixel είναι ελλιπή. Ένας δεύτερος τρόπος είναι να διαλέξουμε τυχαία pixel από κάθε εικόνα και να τα αντικαταστήσουμε με την προκαθορισμένη τιμή για ελλιπή pixel. Αν τα pixel παίρνουν τιμές στο διάστημα $[0, 1]$ (1 για μαύρο pixel και 0 άσπρο pixel), μια καλή επιλογή για την τιμή που θα αναπαραστήσει τα ελλιπή δεδομένα είναι το *0.5*, που αντιστοιχεί σε pixel με γκρι χρώμα.
3. Για τα δεδομένα ελέγχου, διαλέγουμε τα κοντινότερα **K** δεδομένα (εικόνες) τα οποία ο αλγόριθμος θα λάβει υπόψη. Ο αλγόριθμος ενός κοντινότερου γείτονα αντιστοιχεί σε **K=1**.

4. Για κάθε δεδομένο ελέγχου υπολογίζουμε τις Ευκλείδιες αποστάσεις προς κάθε δεδομένο εκπαίδευσης, βρίσκουμε τα **K** κοντινότερα και αποθηκεύουμε τα ευρετήρια (indices) των K κοντινότερων δεδομένων εκπαίδευσης σε μια μεταβλητή. Πρέπει όμως να προσέξουμε να μην προτιμήσουμε κοντινότερους γείτονες με ελλιπείς τιμές. Για να προσπεράσουμε αυτό το πρόβλημα, προτείνουμε την παρακάτω διαδικασία:
Κάνουμε τα pixel των δεδομένων εκπαίδευσης ίσα με την ελλιπή τιμή (missing value), στα σημεία όπου το τρέχον δεδομένο ελέγχου (test instance) έχει ελλιπείς τιμές. Αποθηκεύουμε το αποτέλεσμα στη μεταβλητή με όνομα **X_train_common**. Έτσι, η
διαφορά **X_train_common - X_test_common**, στα σημεία όπου το τρέχον δεδομένο ελέγχου έχει ελλιπείς τιμές, θα είναι 0. Η μεταβλητή **X_test_common** θα κατασκευαστεί αμέσως μετά.

```
X_test_i  # the current test instance
s = np.where(X_test_i == missing_value)
X_train_common = np.array(X_train)
X_train_common[:, s] = missing_value
```

ΚΩΔΙΚΑΣ 5.1: X_train_common

Επαναλαμβάνουμε το τρέχον δεδομένο ελέγχου Ntrain φορές και κάνουμε τις τιμές των δεδομένων ελέγχου ίσες με τον μέσο όρο όλων των δεδομένων εκπαίδευσης, εκεί όπου το τρέχον δεδομένο εκπαίδευσης κάθε φορά έχει ελλιπείς τιμές.
Αποθηκεύουμε το αποτέλεσμα σε μια μεταβλητή με όνομα **X_test_common**.

```
Ntrain  # number of train examples
D  # number of pixels (aka dimensions)
X_test_i  # the current test instance
X_test_common = np.zeros((Ntrain, D))
mean_values = np.mean(X_train, axis=0)  # 1 x D array
for k in range(Ntrain):
    X_test_common[k, :] = X_test_i
    s = np.where(X_train[k, :] == missing_value)
    if len(s[0]) != 0:
        X_test_common[k, s] = mean_values[s]
```

ΚΩΔΙΚΑΣ 5.2: X_test_common

5. Εξάγουμε τα δεδομένα (εικόνες) των οποίων τα ευρετήρια (indices) αντιστοιχούν στα σημεία των K κοντινότερων δεδομένων εκπαίδευσης, από τα δεδομένα εκπαίδευσης **X_train** και τα αποθηκεύουμε σε μια μεταβλητή.
6. Προτείνουμε δύο λύσεις για να προχωρήσουμε περαιτέρω.

   - Μπορούμε να χρησιμοποιήσουμε συντελεστές βαρών, ανάλογα με τις αποστάσεις μεταξύ του τρέχοντος παραδείγματος ελέγχου και τα K κοντινότερα δεδομένα εκπαίδευσης. Το κοντινότερο δεδομένο παίρνει το βάρος με τη μεγαλύτερη τιμή. Κάθε pixel του *k-οστού* δεδομένου εκπαίδευσης πολλαπλασιάζεται με το αντίστοιχο βάρος του w_k. Οι τιμές των

βαρών ανατίθενται ως εξής:

$$\mathbf{w_k} = \mathbf{softmax}(-\mathbf{d_k}) = \frac{\mathbf{e^{-d_k}}}{\sum_{\mathbf{i=1}}^{\mathbf{K}} \mathbf{e^{-d_i}}},$$

$$\text{όπου: } w_1 \geq w_2 \geq w_3 \geq ... \geq w_K,$$

$$\text{και } w_1 + w_2 + w_3 + ... + w_K = \sum_{j=1}^{K} w_j = \sum_{j=1}^{K} \frac{e^{-d_j}}{\sum_{i=1}^{K} e^{-d_i}} = 1$$

όπου d_kσυμβολίζει την απόσταση από το k-οστό κοντινότερο δεδομένο εκπαίδευσης

- Εναλλακτικά, μπορούμε να πάρουμε το άθροισμα όλων των σειρών από την εξαγόμενη μεταβλητή (κάθε σειρά αναπαριστά ένα δεδομένο εκπαίδευσης). Έχοντας υπολογίσει το άθροισμα των K κοντινότερων δεδομένων εκπαίδευσης σε κάθε pixel, μπορούμε να διαιρέσουμε το διάνυσμα των αθροισμάτων με το **K**. Το αποτέλεσμα θα είναι μια μέση πρόβλεψη στη τιμή του κάθε pixel του τρέχοντος δεδομένου ελέγχου. Αυτή η μέθοδος θα ήταν ισοδύναμη με την προηγούμενη, αν είχαμε αναθέσει σε όλα τα βάρη ίσες τιμές: $w_k = \frac{1}{K}$, για κάθε k.

7. Αναθέτουμε τις τιμές του διανύσματος με τις προβλεπόμενες τιμές, μόνο στα αντίστοιχα σημεία (indices) με ελλιπείς τιμές για το τρέχον δεδομένο.
8. Επαναλαμβάνουμε τα βήματα 3-7 για όλα τα δεδομένα ελέγχου (test data).
9. Υπολογίζουμε τη ρίζα της μέσης τιμής του τετραγώνου του σφάλματος (RMSE) μεταξύ των προβλεπόμενων δεδομένων ελέγχου και των πραγματικών δεδομένων ελέγχου, έτσι ώστε να εκτιμήσουμε την απόδοση του αλγορίθμου, με τον ακόλουθο τρόπο:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{N} \sum_{j=1}^{D} (X_{i,j} - \tilde{X}_{i,j})^2}$$

όπου X είναι τα αρχικά δεδομένα και $\tilde{X}$ είναι τα δεδομένα εκπαίδευσης με τις προβλεπόμενες ελλιπείς τιμές. Όσο πιο κοντά είναι αυτός ο δείκτης στο 0, τόσο πιο αποδοτικός είναι ο αλγόριθμος.

Η συνολική διαδικασία της πρόβλεψης των pixel μπορεί να αναπαρασταθεί με χρήση πινάκων ως εξής:

$$closest\_data_{ij} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \dots & p_{1D} \\ p_{21} & p_{22} & p_{23} & \dots & p_{2D} \\ & \dots\dots\dots\dots & & & \\ p_{k1} & p_{k2} & p_{k3} & \dots & p_{kD} \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ . \\ w_K \end{bmatrix} = \begin{bmatrix} p_{11} \cdot w_1 & p_{12} \cdot w_1 & \dots & p_{1D} \cdot w_1 \\ p_{21} \cdot w_2 & p_{22} \cdot w_2 & \dots & p_{2D} \cdot w_2 \\ & \dots\dots\dots\dots\dots & & \\ p_{K1} \cdot w_K & p_{K2} \cdot w_K & \dots & p_{KD \cdot w_K} \end{bmatrix}$$

*όπου **p** σημαίνει pixel, **K** είναι ο αριθμός των κοντινότερων δεδομένων εκπαίδευσης που θα λάβουμε υπόψη και D είναι ο αριθμός των pixel.* Επομένως, το διάνυσμα των προβλεπόμενων pixel για το

τρέχον δεδομένο ελέγχου θα είναι το ακόλουθο:

$$predicted\_pixels = \sum_{i=1}^{K} closest\_data_{ij}$$

Από αυτό το διάνυσμα θα αναθέσουμε τιμές μόνο στα ελλιπή pixel του τρέχοντος δεδομένου ελέγχου. Συνοψίζοντας, ο K-NN αλγόριθμος χρησιμοποιείται κυρίως για κατηγοριοποίηση. Ωστόσο, εδώ δείξαμε πως μπορούμε να τροποποιήσουμε τον αλγόριθμο για να εφαρμόσουμε regression, όπως για παράδειγμα αν θέλουμε να κάνουμε πρόβλεψη πραγματικών τιμών για pixel.

5.2 Ενας Προτεινόμενος Αλγόριθμος Συμπλήρωσης Ελλιπών Τιμών Χρησιμοποιώντας Variational Autoencoder

Μπορούμε να χρησιμοποιήσουμε variational autoencoders για να προβλέψουμε ελλιπείς τιμές σε εικόνες εκπαίδευσης. Η κύρια ιδέα είναι να τροποποιήσουμε τον αλγόριθμο VAE έτσι ώστε να κρατάει ακέραια τα δεδομένα εκεί όπου οι τιμές δεν είναι ελλιπείς και να αλλάζει μόνο τα σημεία με ελλιπείς τιμές, σε κάθε επανάληψη.

Πρώτον, κατασκευάζουμε το dataset με ελλιπείς τιμές, $X\_train\_missing$, από το αρχικό dataset X_train. Αφήνουμε το $\frac{1}{5}$ του dataset όπως ήταν. Από τα υπόλοιπα $\frac{4}{5}$ των δεδομένων, αντικαθιστούμε το μισό κομμάτι κάθε εικόνας, με την "ελλιπή τιμή" (missing value). Η "ελλιπής τιμή" θα μπορούσε να οριστεί ίση με το μέσο του διαστήματος $[lowest\_value, highest\_value]$, όπου $lowest\_value$ και $highest\_value$ είναι η χαμηλότερη και η υψηλότερη τιμή που εμφανίζεται στο dataset, αντίστοιχα. Για παράδειγμα, αυτό το διάστημα, στο MNIST dataset είναι $[0, 1]$, άρα η "ελλιπής τιμή" είναι ίση με 0.5. Το κομμάτι που επιλέγουμε να διαγράψουμε και να αντικαταστήσουμε με ελλιπείς τιμές μπορεί να είναι το επάνω, κάτω, αριστερά ή δεξιά μισό της εικόνας. Μια εναλλακτική επιλογή είναι να διαλέξουμε τυχαία pixel, τα μισά από κάθε εικόνα και να τα αντικαταστήσουμε με την ελλιπή τιμή. Αφού κατασκευάσουμε το dataset με τις ελλιπείς τιμές, $X\_train\_missing$, χρειαζόμαστε να κατασκευάσουμε ένα πίνακα με δυαδικές τιμές (0 ή 1), ο οποίος θα αποθηκεύει την πληροφορία σχετικά με το ποια σημεία των εικόνων αντικαταστήσαμε με ελλιπείς τιμές. Ονομάζουμε αυτόν τον πίνακα X_train_masked. Αν κάποιο pixel δεν αντικαταστάθηκε, το αντίστοιχο pixel στον πίνακα X_train_masked θα είναι 1. Αντίθετα, αν ένα pixel σε μια εικόνα αντικαταστάθηκε από την ελλιπή τιμή, το αντίστοιχο pixel στον πίνακα X_train_masked θα είναι 0. Επιπλέον, πρέπει να καθορίσουμε τον πίνακα που θα περιλαμβάνει τις προβλεπόμενες τιμές των ελλιπών pixel. Ονομάζουμε αυτόν τον πίνακα X_filled. Αρχικοποιούμε τον πίνακα X_filled να είναι ίδιος με τον πίνακα $X\_train\_missing$.

Τελικά, τρέχουμε μια τροποποιημένη έκδοση του VAE αλγορίθμου. Όπως αναφέραμε νωρίτερα, σε κάθε επανάληψη, πρέπει να αντικαθιστούμε τα pixel μόνο εκεί όπου υπήρχαν ελλιπείς τιμές, με τις

προβλεπόμενες. Την ίδια στιγμή, πρέπει να διατηρούμε ακέραια τα pixel με τις μη ελλιπείς τιμές. Για αυτό το σκοπό, θα φανεί χρήσιμος ο πίνακας X_train_masked.

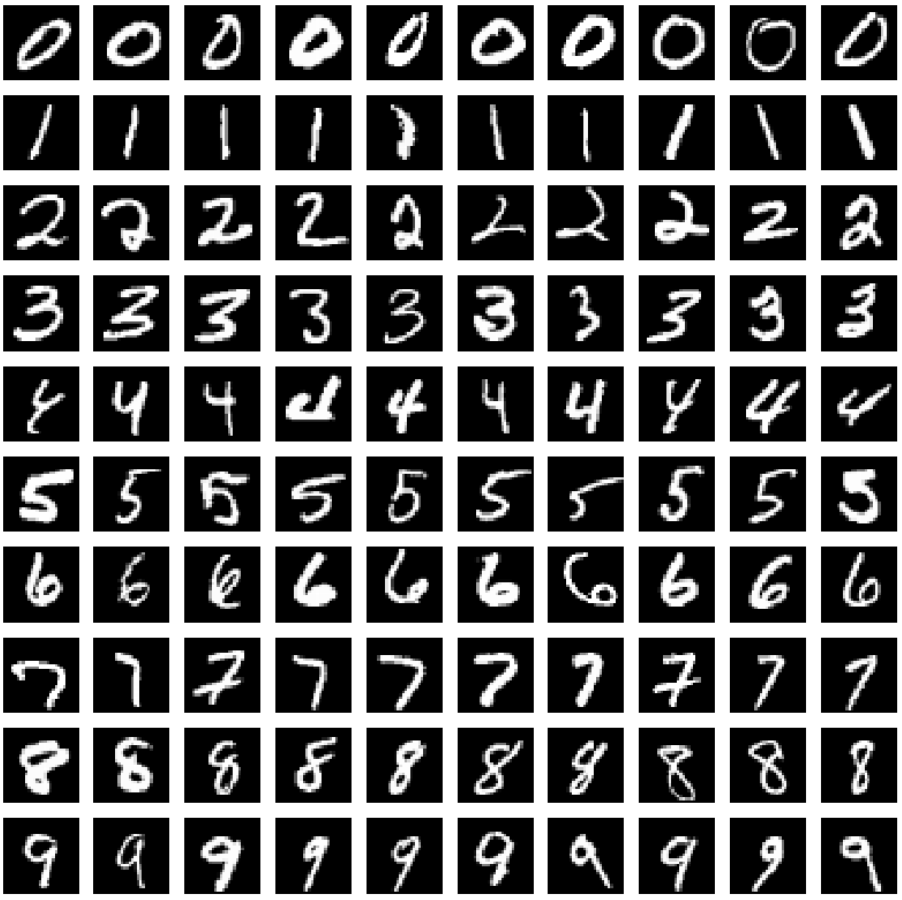
Ακολουθεί ψευδοκώδικας για όλη τη διαδικασία που περιγράψαμε παραπάνω:

```
for epoch = 0 to epochs - 1
    iterations = N / batch_size

    for i = 0 to iterations - 1
        start_index = i * batch_size
        end_index = (i + 1) * batch_size

        // fetch the batch data, labels and masked batch data
        batch_data = X_filled(start_index:end_index, :)
        batch_labels = y_train(start_index:end_index)
        masked_batch_data = X_train_masked(start_index:end_index, :)

        // train the batch data using the VAE process
        cur_samples = train(batch_data, params)

        // ".*" denotes element-wise multiplication
        // The "cur_samples" will take values from the "batch_data"
        // where the pixels are observed (with masked values=1)
        // and will keep intact its values from the VAE training
        // where the pixels are missing (with masked values=0).
        cur_samples = masked_batch_data .* batch_data +
                      (1 - masked_batch_data) .* cur_samples
        X_filled(start_index:end_index, :) = cur_samples
```

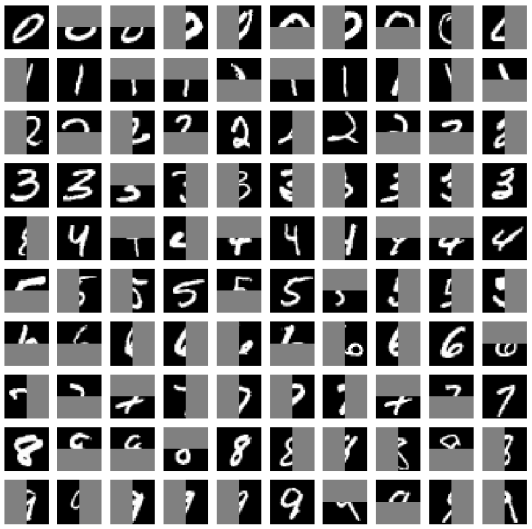
ΚΩΔΙΚΑΣ 5.3: ψευδοκώδικας VAE αλγορίθμου συμπλήρωσης ελλιπών τιμών

5.3 Αποτελέσματα Πειραμάτων του K-NN Αλγορίθμου Συμπλήρωσης Ελλιπών Τιμών και Αλγορίθμων Συμπλήρωσης Ελλιπών Τιμών με Χρήση VAE

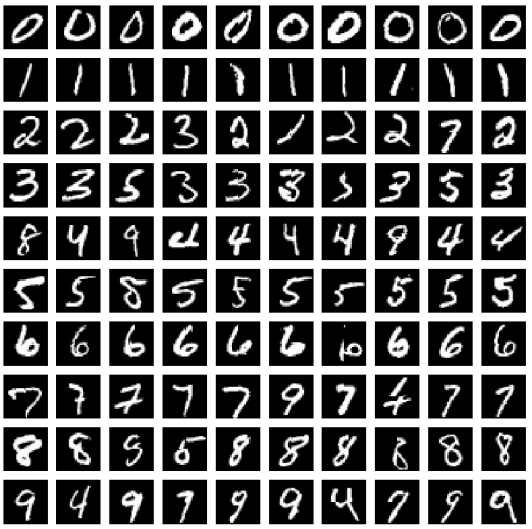
Μετά από εκτελέσεις του αλγορίθμου συνεργατικού φιλτραρίσματος στο **MNIST** dataset με ελλιπείς τιμές, για διάφορες τιμές του K, και εκτελέσεις του VAE αλγορίθμου αλλιπών τιμών σε PyTorch, καταλήξαμε στα παρακάτω αποτελέσματα. Όπως πρόκειται να δούμε, κάποιες εικόνες ελέγχου (test) των ψηφίων *3*, *5*, *8* ανακατασκευάστηκαν με το άλλο μισό των ψηφίων *3* ή *5* ή *8*. Αυτό το ατυχές συμβάν είναι λογικό, γιατί τα ψηφία *3*, *5* και *8* μοιάζουν μεταξύ τους, εν αντιθέσει με τα άλλα ψηφία.



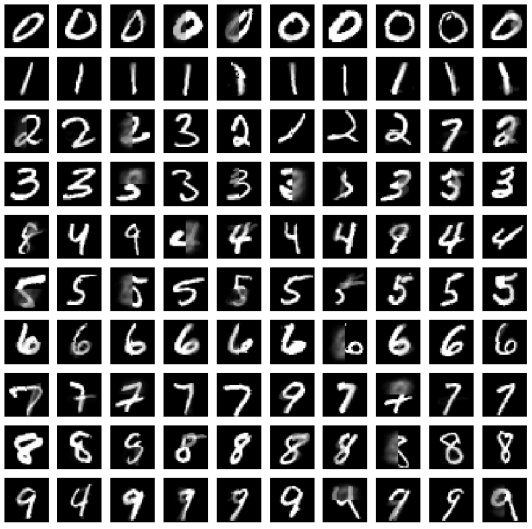
Εικονα 5.1: Αρχικά MNIST Δεδομένα Ελέγχου (Test)



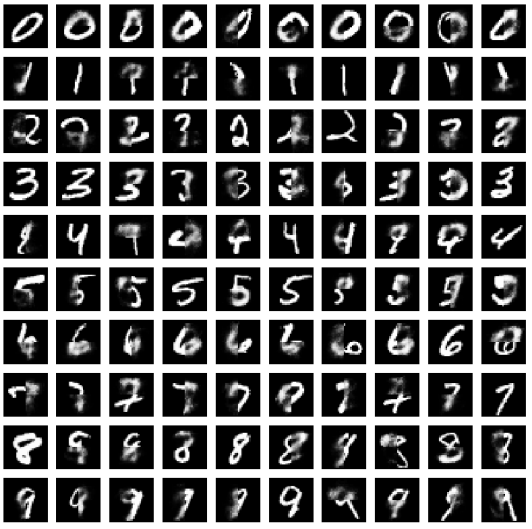
(a) Δεδομένα Εκπαίδευσης με Δομημένες (Structured) Ελλιπείς Τιμές



(b) 1-NN



(c) 100-NN



(d) VAE σε PyTorch Εποχή 200

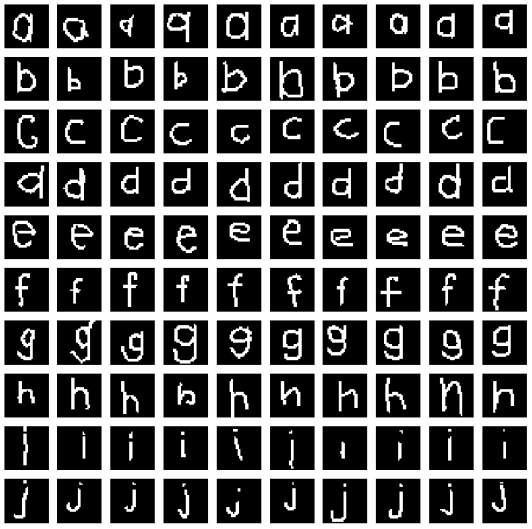
Εικονα 5.2: MNIST **1-NN**, **100-NN** & **VAE** αλγόριθμοι ελλιπών τιμών.

Το ελάχιστο root mean squared error (RMSE) και το ελάχιστο mean absolute error (MAE) που επετεύχθησαν ήταν για **K=100**, αλλά ο αλγόριθμος ήταν ο πιο χρονοβόρος συγκριτικά με τους άλλους.

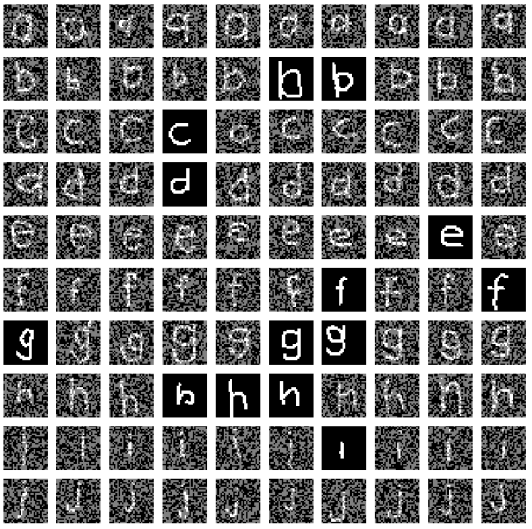
Method	RMSE	MAE	Time
1-NN	0.171569	0.0406822	190.26 sec
100-NN	0.148223	0.0396628	233.19 sec
VAE	0.168031	0.0499518	178.09 sec

Πινακας 5.1: Αλγόριθμοι συμπλήρωσης ελλιπών τιμών στο MNIST dataset.

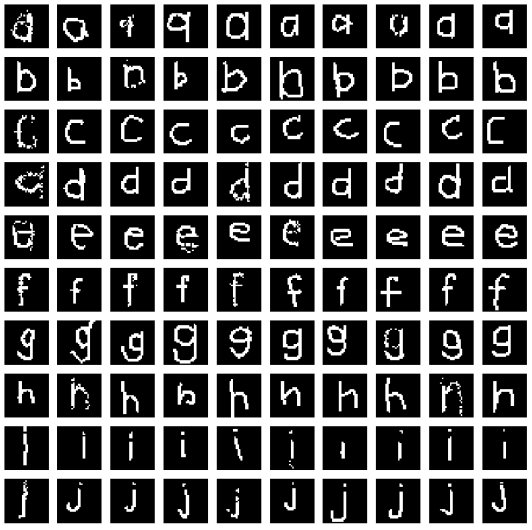
Ακολουθούν τα αποτελέσματα των αλγορίθμων συμπλήρωσης ελλιπών τιμών, στο **OMNIGLOT** Αγγλικό dataset: **OMNIGLOT** English dataset.



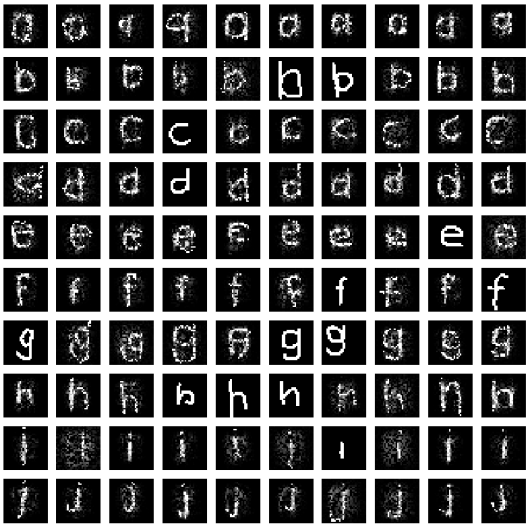
(a) Αρχικά Δεδομένα



(b) Ελλιπή Δεδομένα



(c) 1-NN



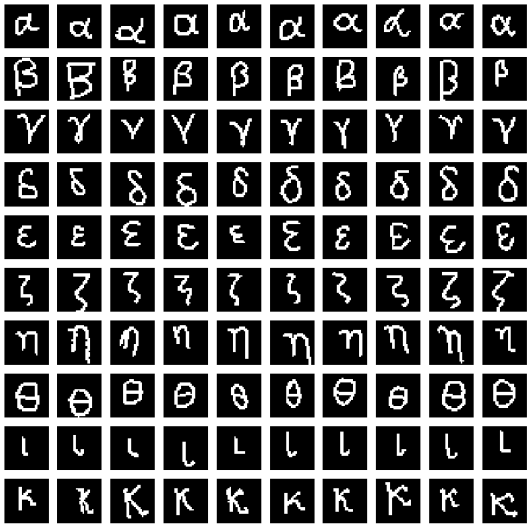
(d) VAE σε PyTorch Εποχή 100

Εικονα 5.3: **1-NN** & **VAE** αλγόριθμοι ελλιπών τιμών στο OMNIGLOT Αγγλικό αλφάβητο, χαρακτήρες 1-10, αρχικές, τυχαίες ελλιπείς τιμές και ανακατασκευασμένες.

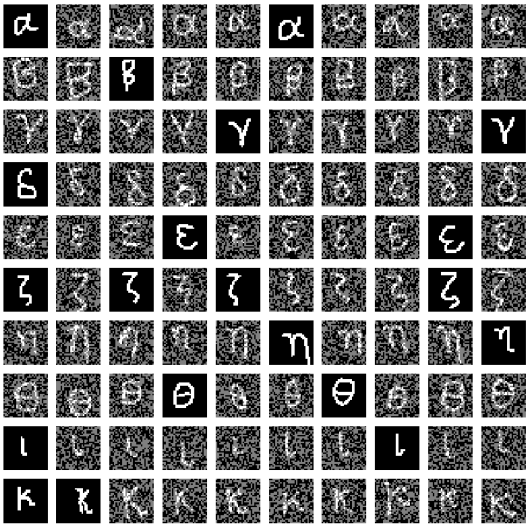
Method	RMSE	MAE	Time
1-NN	0.0890025270986	0.00792144982993	8.64 sec
VAE	0.168031	0.0499518	178.09 sec

Πινακας 5.2: Αλγόριθμος συμπλήρωσης ελλιπών τιμών στο OMNIGLOT English dataset.

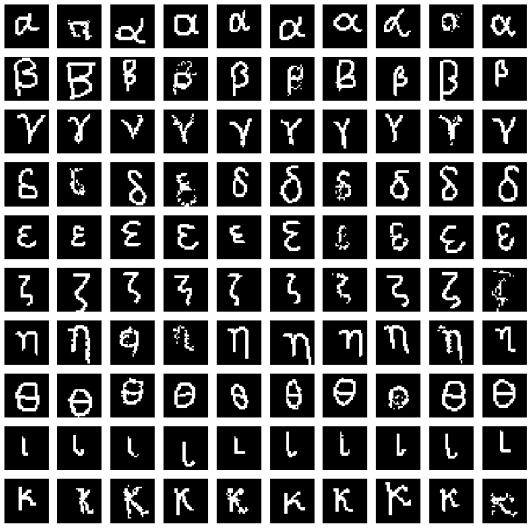
Ακολουθούν τα αποτελέσματα των αλγορίθμων συμπλήρωσης ελλιπών τιμών, με χρήση **PyTorch**, στο **OMNIGLOT** Ελληνικό dataset.



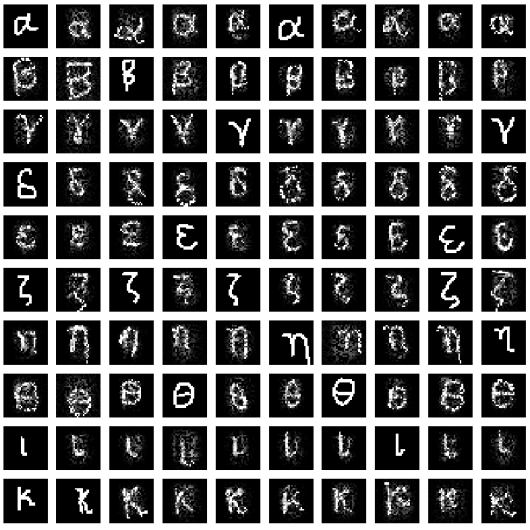
(a) Αρχικά Δεδομένα



(b) Ελλιπή Δεδομένα



(c) 1-NN



(d) VAE σε PyTorch Εποχή 100

Εικονα 5.4: **1-NN** & **VAE** αλγόριθμοι ελλιπών τιμών στο OMNIGLOT Ελληνικό αλφάβητο, χαρακτήρες 1-10, αρχικές, τυχαίες ελλιπείς τιμές και ανακατασκευασμένες.

Method	RMSE	MAE	Time
1-NN	0.0846893853663	0.00717229199372	9.37 sec
VAE	0.162737553068	0.0524327812139	245.66 sec

Πινακας 5.3: Αλγόριθμος συμπλήρωσης ελλιπών τιμών στο OMNIGLOT Ελληνικό dataset.

ΣΗΜΕΙΩΣΗ: Ο δείκτης του χρόνου (time) για τους K-NN αλγορίθμους δείχνει τη διάρκεια των υπολογισμών των αποστάσεων και των προβλέψεων, ενώ ο δείκτης του χρόνου για τους VAE αλγορίθμους δείχνει τις διάρκειες των βρόχων εκπαίδευσης των VAE.

5.4 K-NN vs VAE Αλγόριθμος Συμπλήρωσης Ελλιπών Τιμών στο MovieLens Dataset

Τρέξαμε τους K-NN και VAE σε TensorFlow αλγορίθμους συμπλήρωσης ελλιπών τιμών στο MovieLens 100k dataset, **ua set** και συγκρίναμε τις προβλεπόμενες βαθμολογίες. Τα αποτελέσματα φαίνονται παρακάτω:

Metric	**Value**
root mean squared error (RMSE)	0.0803184360998
mean absolute error (MAE)	0.0209178842034
match percentage	91.2514516501 %

Method	**Mean Rating**
VAE in TensorFlow	1.60717332671
K-NN Missing Values algorithm	1.61095209334

Πινακας 5.4: Σύγκριση μεταξύ των 10-NN και VAE σε TensorFlow αλγορίθμων συμπλήρωσης ελλιπών τιμών.

Από τον πίνακα αυτόν και κρίνοντας από τα σφάλματα, συμπεραίνουμε ότι οι προβλέψεις των δύο αλγορίθμων για τις βαθμολογίες των χρηστών είναι πολύ κοντινές. Μάλιστα, όταν κάναμε στρογγυλοποίηση εξαλείφοντας τα δεκαδικά ψηφία των βαθμολογιών, καταλήξαμε ότι τα αποτελέσματα ταυτίζονται κατά περίπου 91%. Τέλος, η μέση βαθμολογία για τον VAE αλγόριθμο σε TensorFlow είναι περίπου 1.61, ενώ η μέση βαθμολογία για τον K-NN αλγόριθμο συμπλήρωσης ελλιπών τιμών είναι περίπου 1.6. Οι βαθμολογίες είναι στρογγυλοποιημένες ώστε να παίρνουν ακέραιες τιμές στο διάστημα $[1, 5]$ και η τιμή που δείχνει ότι ένας χρήστης δεν έχει βαθμολογήσει μια ταινία είναι 0.

Κεφάλαιο 6

Γραφική Διεπαφή (GUI) για Variational Autoencoder & Αλγορίθμους Συμπλήρωσης Ελλιπών Τιμών

Μαζί με το project αυτής της διατριβής έχει υλοποιηθεί μια εφαρμογή γραφικής διεπαφής (GUI), σε προγραμματιστική γλώσσα Python 3, κάνοντας χρήση της βιβλιοθήκης **Tkinter**.

Πρώτα πηγαίνετε στο directory **"vaes_gui"** της εργασίας και εγκαταστείστε τις απαιτούμενες Python βιβλιοθήκες, πληκτρολογώντας:

```
pip install -r depencencies.txt
```

ΚΩΔΙΚΑΣ 6.1: εντολή για εγκατάσταση Python εξαρτήσεων

Για να εκτελέσετε το GUI από το τερματικό, πληκτρολογείστε την εντολή:

```
python vaes_gui.py
```

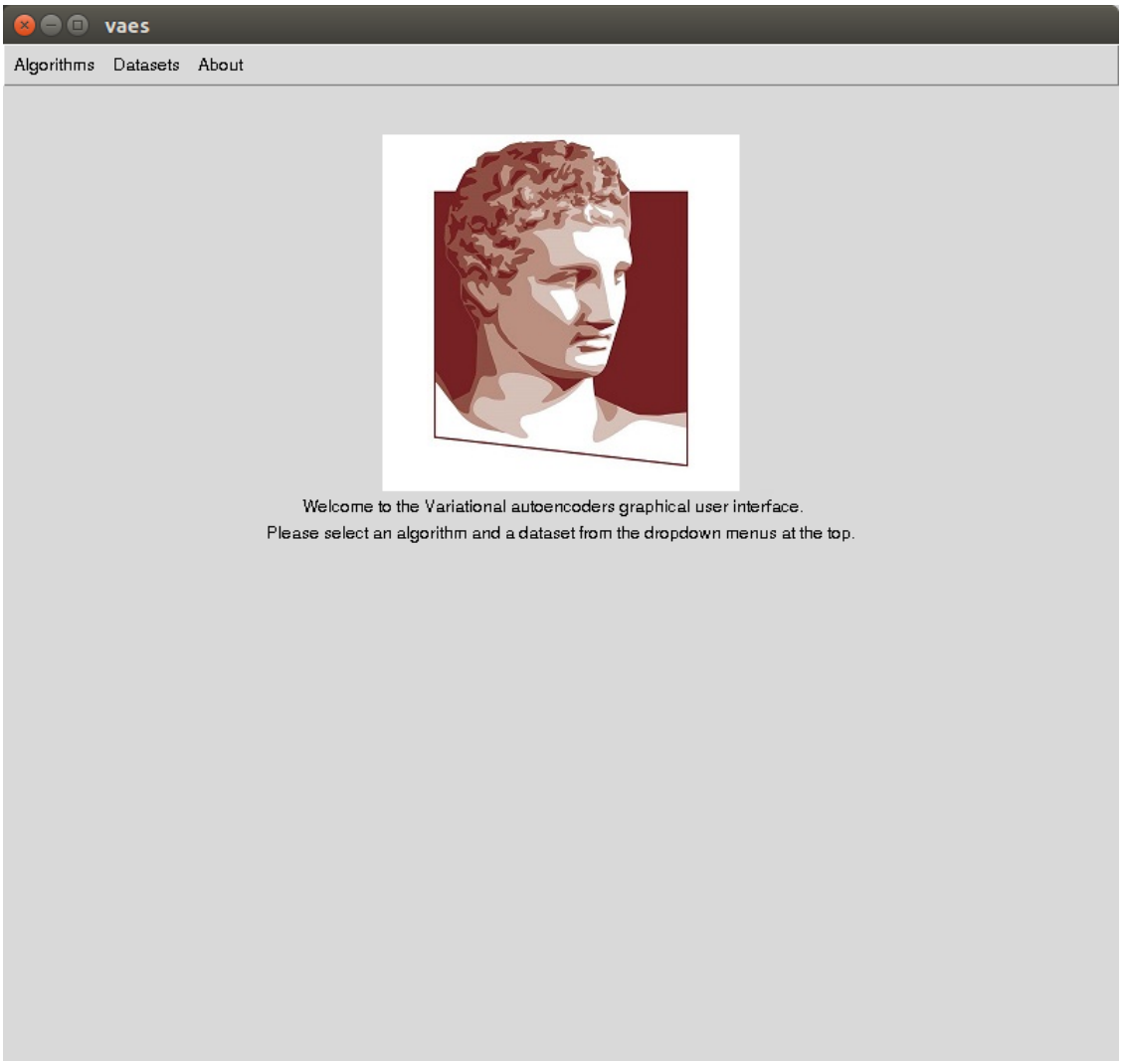
ΚΩΔΙΚΑΣ 6.2: εντολή για εκτέλεση του GUI

Για να δημιουργήσετε ένα εκτελέσιμο αρχείο για το GUI (".exe"), το οποίο θα μπορείτε να ανοίξετε ανά πάσα ώρα σε Windows περιβάλλον, πληκτρολογείστε:

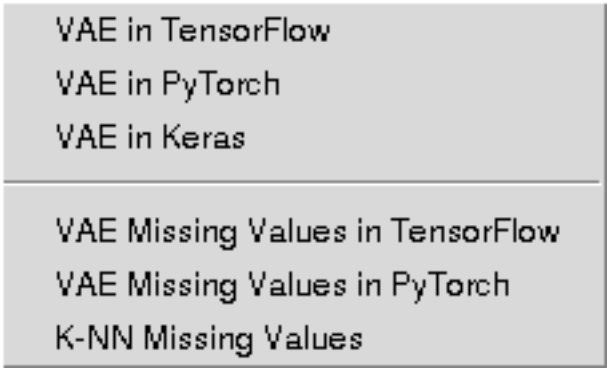
```
pip install pyinstaller
pyinstaller vaes_gui.spec
```

ΚΩΔΙΚΑΣ 6.3: εντολή για δημιουργία εκτελέσιμου αρχείου για το GUI

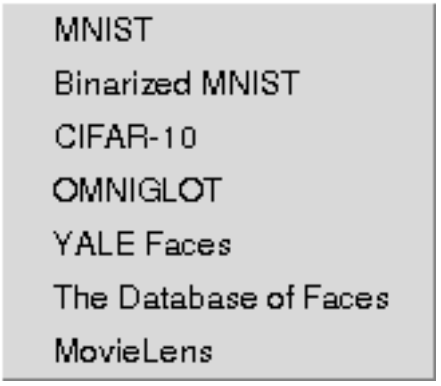
Στη συνέχεια, κατεβάστε όλα τα dataset από τα URL στο αρχείο **"datasets_urls.txt"** και μετακινήστε τα στον νέο φάκελο **"dist"**. Εντός του φακέλου, υπάρχει ένας άλλος φάκελος με όνομα **"vaes_gui"**, ο οποίος περιλαμβάνει το εκτελέσιμο αρχείο **"vaes_gui.exe"**.



Εικονα 6.1:
Σελίδα καλωσορίσματος του GUI.

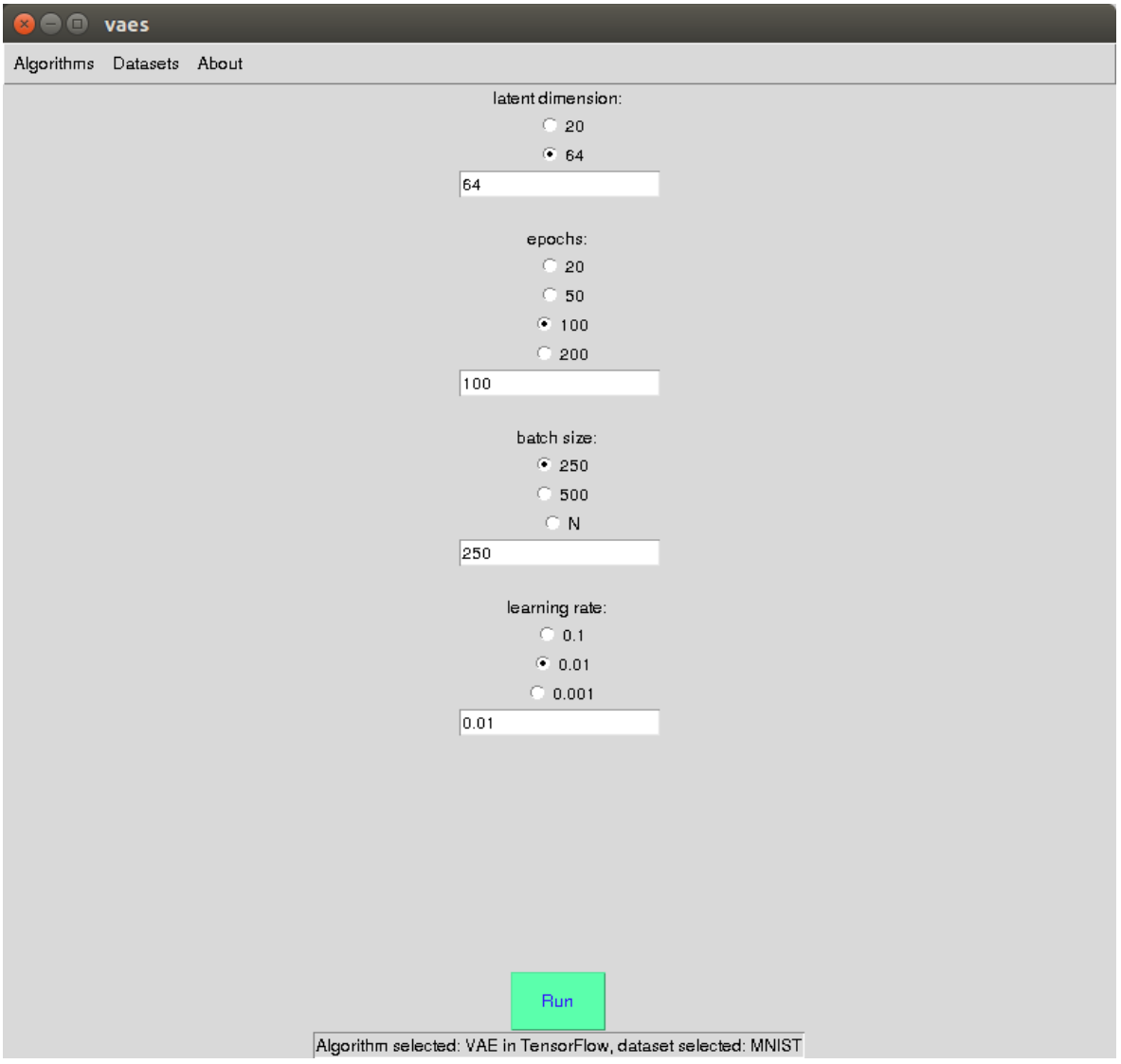


(a) GUI dropdown menu Αλγορίθμων.

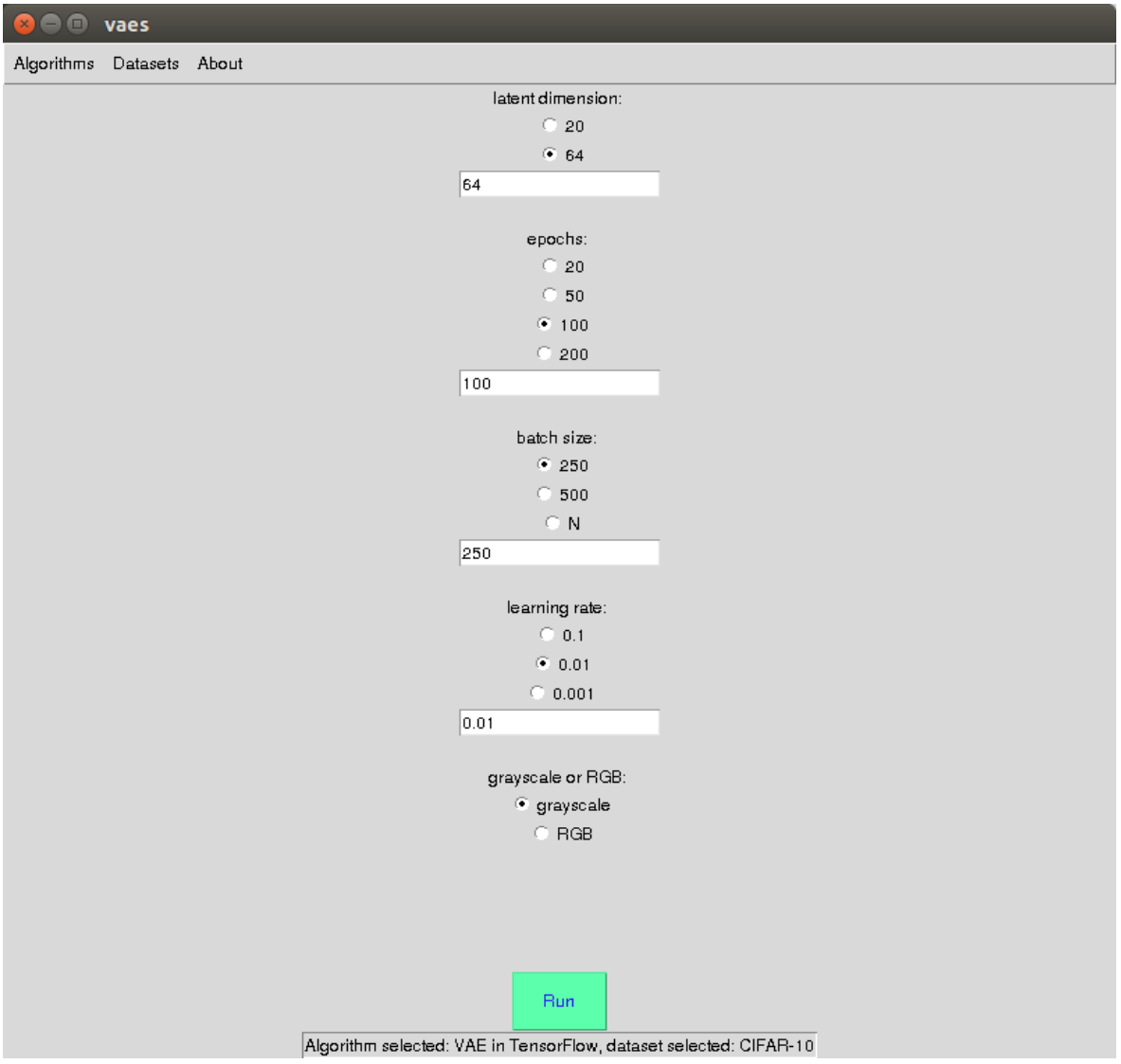


(b) GUI Datasets dropdown menu.

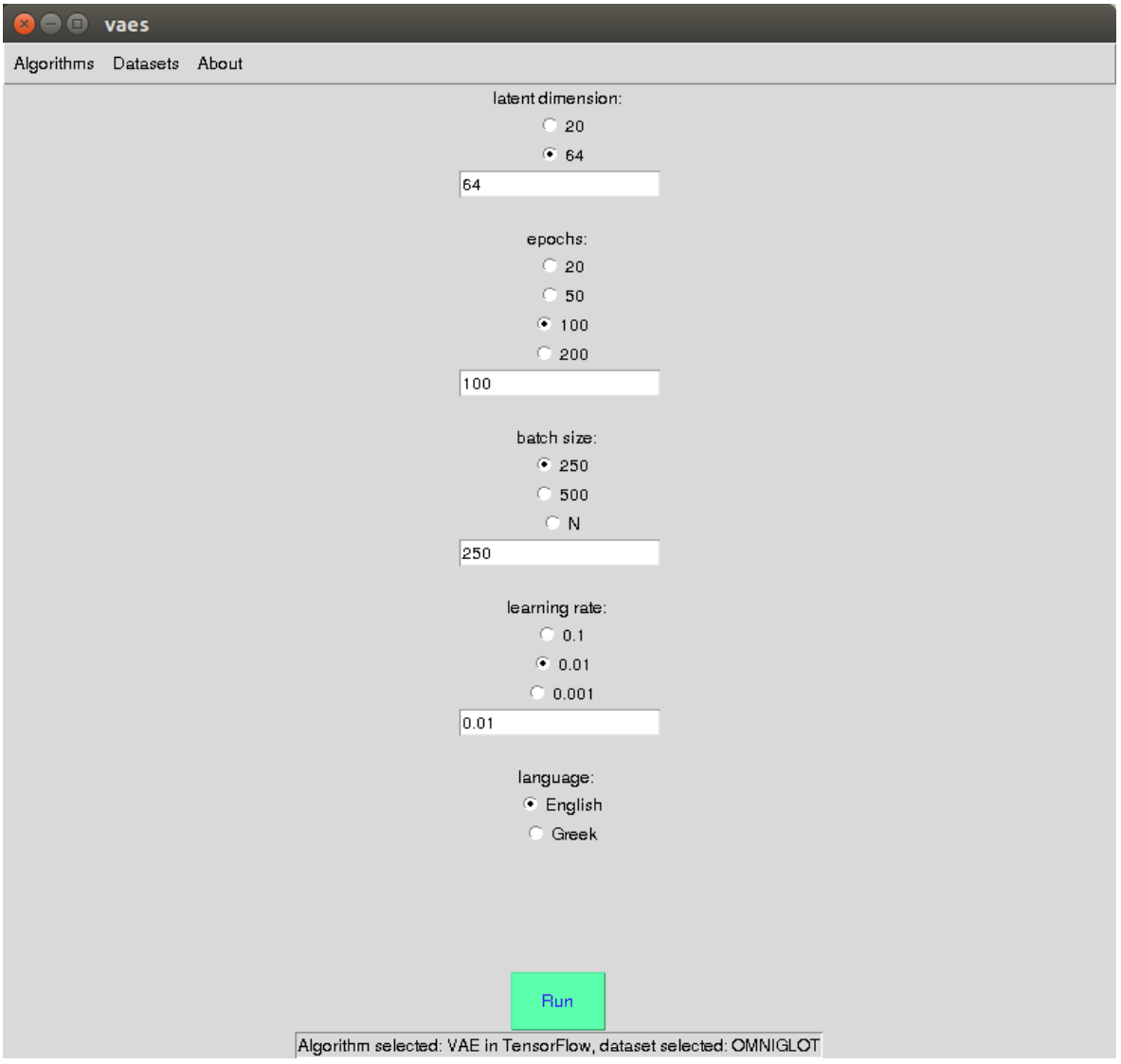
Εικονα 6.2: Dropdown menus.



Εικονα 6.3: GUI VAE σε TensorFlow, MNIST dataset.

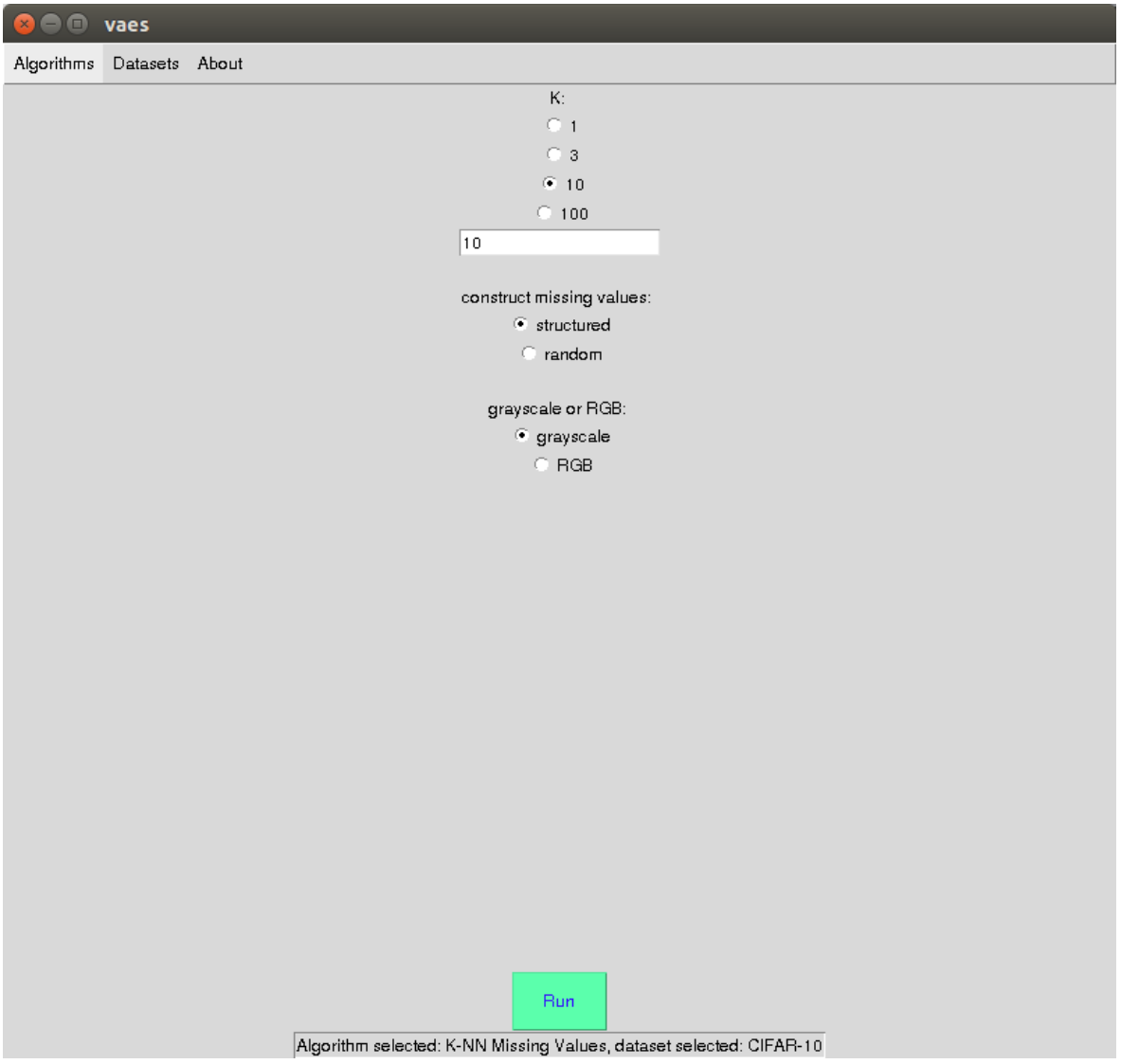


Εικονα 6.4: GUI VAE σε TensorFlow, CIFAR-10 dataset.

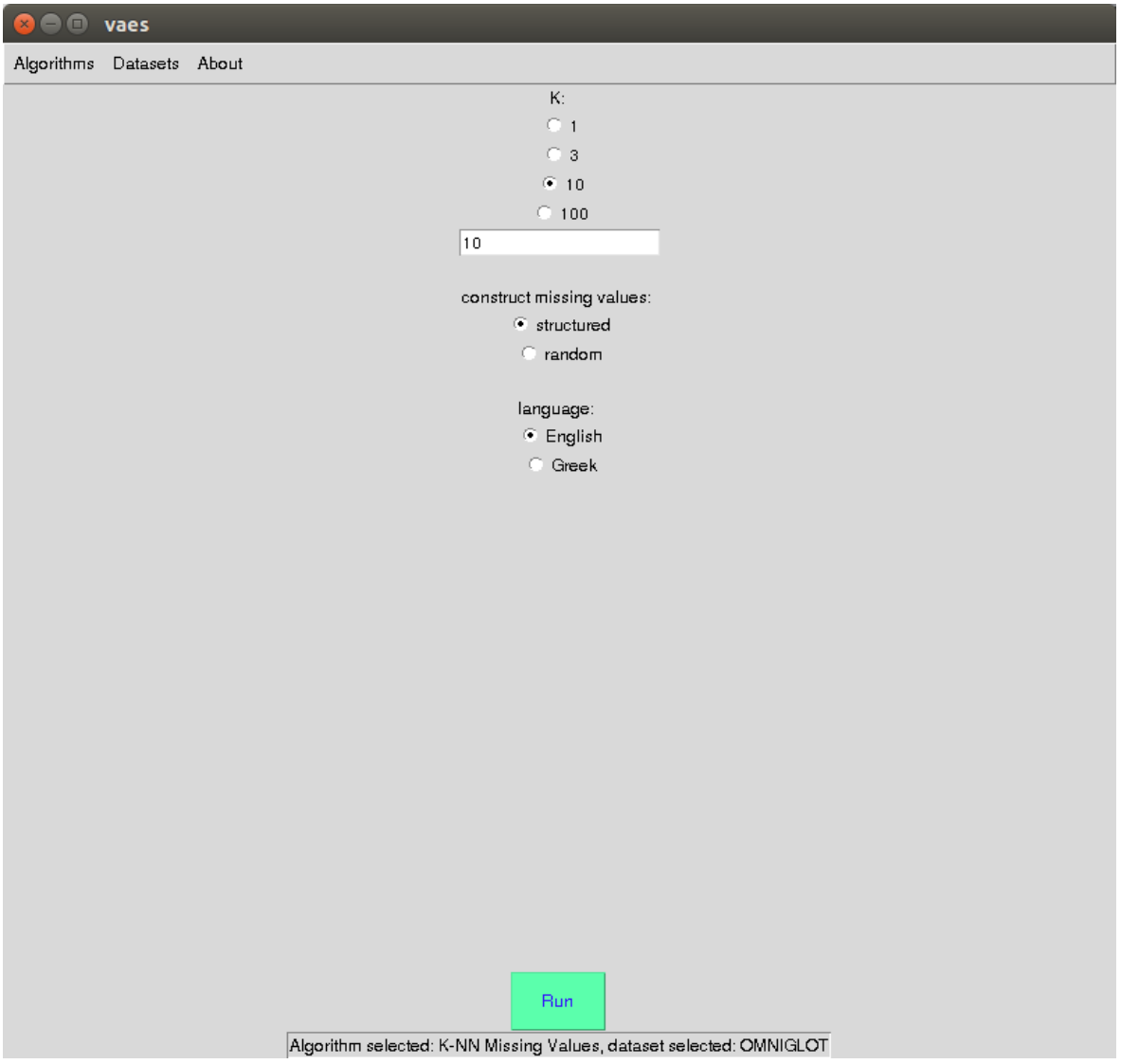


Εικονα 6.5: GUI VAE σε TensorFlow, OMNIGLOT dataset.

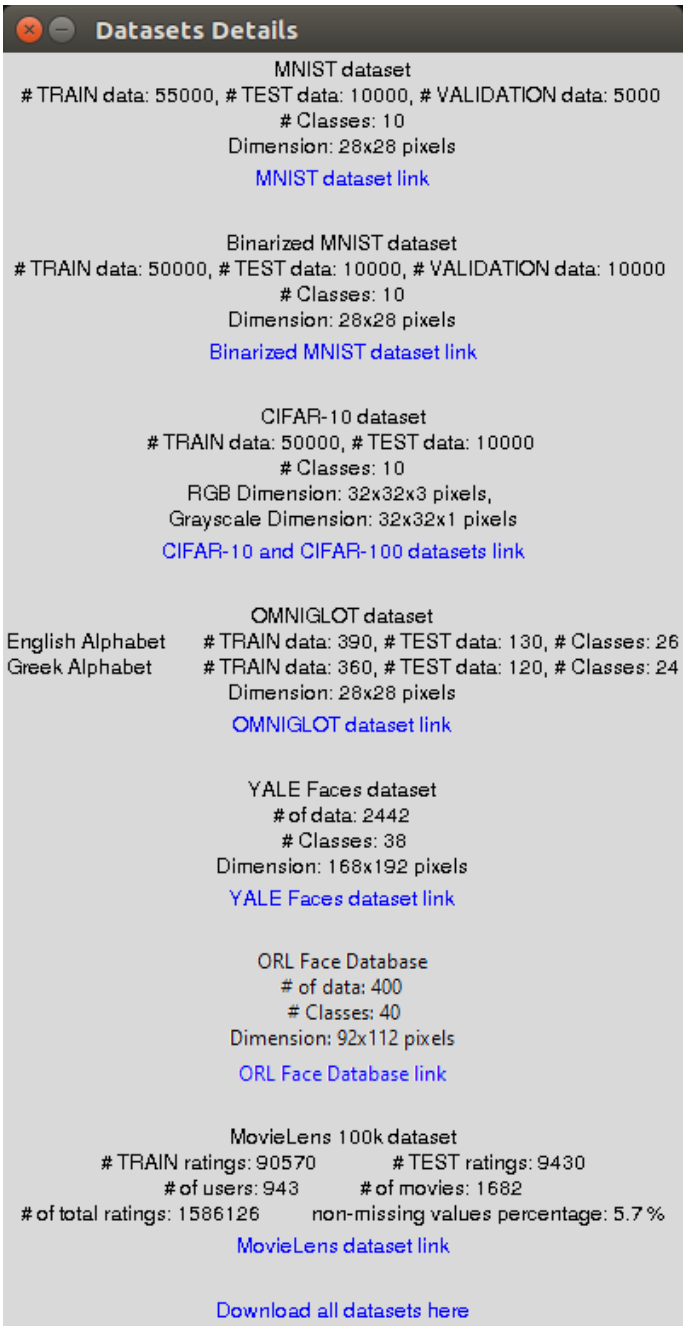
Εικονα 6.6: GUI K-NN αλγόριθμος Ελλιπών Τιμών, MNIST dataset.



Εικονα 6.7: GUI K-NN αλγόριθμος Ελλιπών Τιμών, CIFAR-10 dataset.



Εικονα 6.8: GUI K-NN αλγόριθμος Ελλιπών Τιμών, OMNIGLOT dataset.



Εικονα 6.9: GUI λεπτομέρειες των dataset.

Εικονα 6.10: GUI σχετικά.

Κεφάλαιο 7

Περαιτέρω Εφαρμογές των Variational Autoencoder

(Aaron Courville, Ian Goodfellow, Yoshua Bengio 2016. [4])

Οι autoencoder έχουν εφαρμοστεί επιτυχώς σε διεργασίες όπως: **1) ελάττωση διάστασης** και **2) ανάκτηση πληροφοριών**. Η ελάττωση διάστασης ήταν από μια από τις πρώτες εφαρμογές της αντιπροσωπευτικής μάθησης (representation learning) και της βαθιάς μάθησης (deep learning).

Αναπαραστάσεις σε χαμηλότερη διάσταση μπορούν να βελτιώσουν την απόδοση σε πολλές διεργασίες, όπως η κατηγοριοποίηση. Μοντέλα μικρότερου όγκου καταναλώνουν λιγότερη μνήμη και χρόνο εκτέλεσης. Τα στοιχεία που παρέχονται από τη χαρτογράφηση στο χώρο χαμηλότερης διάστασης βοηθούν στη γενίκευση (generalization).

Μια διαδικασία που ωφελεί παραπάνω από το συνηθισμένο στην ελάττωση διάστασης είναι η **ανάκτηση πληροφοριών**. Ανάκτηση πληροφοριών είναι η διαδικασία αναζήτησης καταχωρήσεων, από μια βάση δεδομένων που μοιάζουν σε μια καταχώρηση-ερώτημα (query entry). Αυτή η διαδικασία αντλεί τα ίδια πλεονεκτήματα με άλλες διεργασίες, από την ελάττωση διάστασης, και έχει επιπλέον το πλεονέκτημα ότι η αναζήτηση μπορεί να γίνει υπερβολικά επαρκής σε
συγκεκριμένα είδη χώρων χαμηλής διάστασης. Ειδικά, αν εκπαιδεύσουμε τον αλγόριθμο ελάττωσης διάστασης ώστε να παράγει έναν κωδικό που να είναι μικρής διάστασης και δυαδικός, τότε μπορούμε να αποθηκεύσουμε όλες τις καταχωρήσεις της βάσης δεδομένων σε ένα πίνακα κατακερματισμού (hash table), αντιστοιχίζοντας διανύσματα δυαδικού κωδικού σε καταχωρίσεις (entries). Αυτός ο πίνακας κατακερματισμού μας επιτρέπει να εκτελέσουμε ανάκτηση πληροφοριών επιστρέφοντας όλες τις καταχωρήσεις της βάσης δεδομένων που έχουν τον ίδιο δυαδικό κωδικό με το ερώτημα (query). Μπορούμε επίσης να ψάξουμε σε λίγο λιγότερο παρόμοιες καταχωρήσεις, πολύ αποτελεσματικά, αναστρέφοντας απλώς μεμονωμένα bits από την κωδικοποίηση του ερωτήματος. Αυτή η προσέγγιση στην ανάκτηση πληροφοριών μέσω της ελάττωσης διάστασης και της κωδικοποίησης σε δυαδικό σύστημα ονομάζεται **σημασιολογικός κατακερματισμός** (semantic hashing).

Παράρτημα Α

Θεωρητικό Υπόβαθρο

Α.1 Κανόνας του Bayes

Κανόνας του Bayes για υπό συνθήκη (conditional) πιθανότητες:

$$P(A|B) \cdot P(B) = P(B|A) \cdot P(A) \Rightarrow$$

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Επίσης, ισχύει η παρακάτω εξίσωση:

$$P(A|B) = \frac{P(A,B)}{P(B)}$$

Α.2 Συνάρτηση Softmax

Έστω Χ ένα διάνυσμα ή ένας πίνακας, το δεδομένο εισόδου. Η συνάρτηση softmax κατασκευάζει βάρη για κάθε στοιχείο της εισόδου Χ. Το στοιχείο με τη μεγαλύτερη τιμή θα πάρει το βάρος με τη μεγαλύτερη τιμή. Επίσης, το άθροισμα όλων των βαρών πρέπει να είναι ίσο με 1. Επομένως, η συνάρτηση softmax δίνεται από τον παρακάτω τύπο:

$$\mathbf{w_i} = \mathbf{softmax}(\mathbf{X}) = \frac{\mathbf{e^{X_i}}}{\sum_{\mathbf{j=1}}^{\mathbf{N}} \mathbf{e^{X_j}}},$$

$$\text{and } w_1 + w_2 + w_3 + ... + w_N = \sum_{i=1}^{N} w_i = \sum_{i=1}^{N} \frac{e^{X_i}}{\sum_{j=1}^{N} e^{X_j}} = \frac{\sum_{i=1}^{N} e^{X_i}}{\sum_{j=1}^{N} e^{X_j}} = 1$$

Α.3 Εντροπία Θεωρίας Πληροφοριών (Entropy)

Η εντροπία πληροφορίας ορίζεται ως η μέση ποσότητα πληροφορίας που παράγεται από μια στοχαστική πηγή δεδομένων.

Ο τύπος της εντροπίας πληροφορίας μιας κατανομής P είναι ο ακόλουθος:

$$H(P) = -\sum_{x} P(x) \cdot log_b P(x)$$

όπου b είναι η βάση του λογαρίθμου

Η εντροπία πληροφορία υπολογίζεται τυπικά σε bits (εναλλακτικά επονομαζόμενα **"Shannons"**) για *b=2* ή ορισμένες φορές σε "φυσικές μονάδες" (natural units) (**nats**) για *b=e* (σταθερά του Euler), ή σε δεκαδικά ψηφία (επονομαζόμενα "dits", "bans", ή "hartleys") για *b=10*. Η μονάδα μέτρησης εξαρτάται από τη βάση του λογαρίθμου που χρησιμοποιείται για να καθορίσει την εντροπία.

Α.4 Cross-Entropy

Έστω Q μία "αφύσικη" πιθανοτική κατανομή και έστω P η "πραγματική" κατανομή για κάποιο σύνολο συμβάντων. Το cross entropy μεταξύ δύο πιθανοτικών κατανομών P και Q υπολογίζει τον μέσο αριθμό από bit που χρειάζονται για να αναγνωριστεί συμβάν επιλεγμένο από το σύνολο.

Ο τύπος του cross entropy δύο διακριτών κατανομών P και Q είναι ο ακόλουθος:

$$H(P, Q) = -\sum_{x} P(x) \cdot log_b Q(x)$$

Παρομοίως, ο τύπος του cross entropy δύο συνεχών κατανομών P και Q είναι ο ακόλουθος:

$$H(P, Q) = -\int_{-\infty}^{+\infty} P(x) \cdot log_b Q(x)\, dx \Rightarrow$$

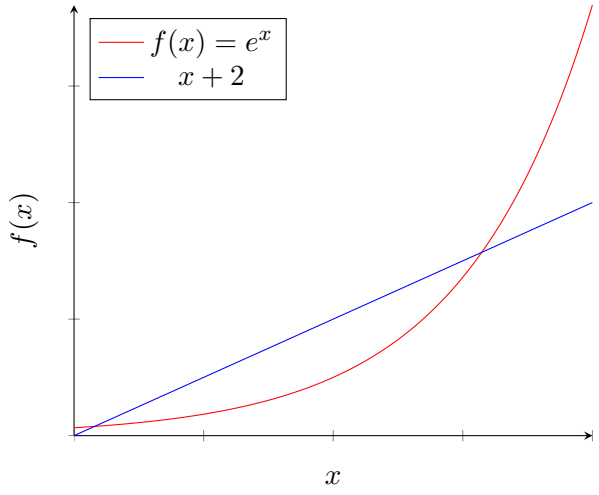
$$H(P, Q) = -E_P[-logQ(x)]$$

όπου b είναι η βάση του λογαρίθμου

Α.5 Ανισότητα του Jensen

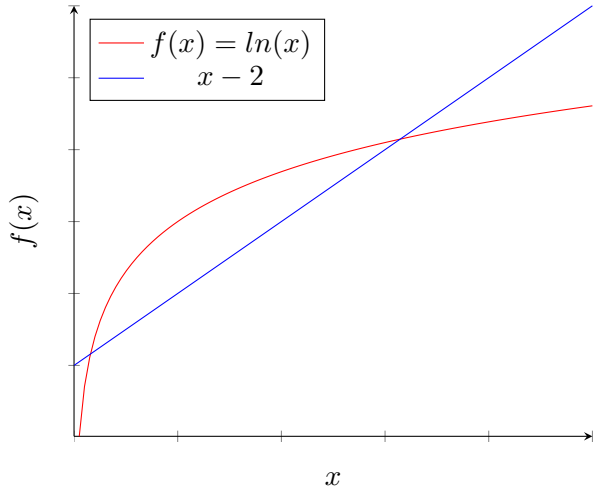
Η ανισότητα του Jensen, η οποία πήρε το όνομά της από το Δανό μαθηματικό Johan Jensen, συσχετίζει την τιμή μιας κυρτής (ή κοίλης) συνάρτησης ενός ολοκληρώματος με το ολοκλήρωμα μιας κυρτής συνάρτησης. Γενικεύει τη δήλωση ότι μια τέμνουσα γραμμή σε μια κυρτή συνάρτηση βρίσκεται πάνω από τη γραφική παράσταση της συνάρτησης.

$f(E[X]) \leq E[f(X)]$, όπου f είναι μια κυρτή συνάρτηση



Με την ίδια λογική, μια τέμνουσα γραμμή σε μια κοίλη συνάρτηση βρίσκεται κάτω από τη γραφική παράσταση της συνάρτησης.

$f(E[X]) \geq E[f(X)]$, όπου f είναι μια κοίλη συνάρτηση



Α.6 Kullback-Leibler (KL) Απόκλιση

Αν έχουμε δύο ξεχωριστές πιθανοτικές κατανομές $P(x)$ και $Q(x)$ για την ίδια T.M. x, μπορούμε να μετρήσουμε πόσο διαφορετικές είναι αυτές οι δύο κατανομές χρησιμοποιώντας την Kullback-Leibler (KL) απόκλιση (divergence):

$$D_{KL}[P\,||\,Q] = E_{X\sim P}[log\frac{P(X)}{Q(X)}] = E_{X\sim P}[logP(X) - logQ(X)]$$

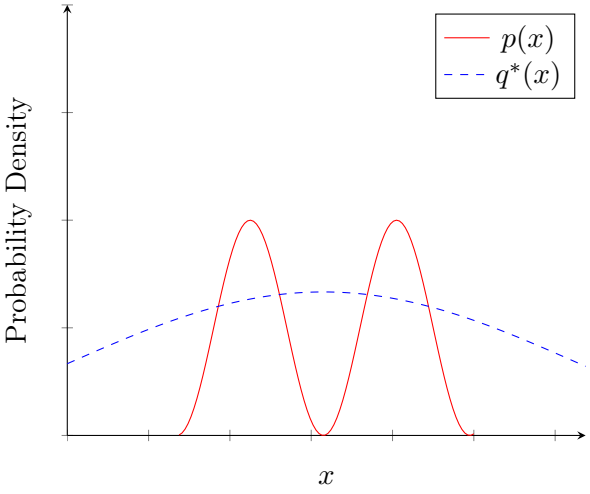
Στην περίπτωση των διακριτών τυχαίων μεταβλητών, αυτή η απόκλιση είναι η επιπλέον ποσότητα πληροφορίας (μετρημένη σε bit αν χρησιμοποιούμε λογάριθμο βάσης 2, αν και στη Μηχανική Μάθηση συνήθως χρησιμοποιούμε nat και το φυσικό λογάριθμο) που απαιτείται για να στείλουμε ένα μήνυμα που περιέχει σύμβολα σχεδιασμένα από μια πιθανοτική κατανομή P, όταν χρησιμοποιούμε μια κωδικοποίηση που έχει σχεδιαστεί για να ελαχιστοποιεί το μέγεθος των μηνυμάτων που
εξάγονται από την κατανομή Q. Η KL κατανομή έχει πολλές χρήσιμες ιδιότητες, κυρίως το ότι είναι μη αρνητική. Η KL απόκλιση είναι 0 αν και μόνο αν η P και η Q είναι η ίδια κατανομή, στην περίπτωση των διακριτών μεταβλητών, ή είναι ίσες "σχεδόν παντού", στην περίπτωση των συνεχών μεταβλητών. Επειδή η KL απόκλιση είναι μη αρνητική και μετράει τη διαφορά μεταξύ δύο κατανομών, συχνά έχει την έννοια ότι μετράει κάποιου είδους απόσταση μεταξύ αυτών των κατανομών. Ωστόσο, δεν είναι πραγματικό μέτρο απόστασης, επειδή δεν είναι συμμετρικό: $D_{KL}(P\,||\,Q) \neq D_{KL}(Q\,||\,P)$ για κάποιο P και Q. Αυτή η ασυμμετρία σημαίνει ότι υπάρχουν σημαντικές συνέπειες στον αν θα επιλέξουμε να χρησιμοποιήσουμε $D_{KL}(P\,||\,Q)$ ή $D_{KL}(Q\,||\,P)$. Μια ποσότητα που είναι στενά συνδεδεμένη με την KL απόκλιση είναι το cross-entropy, $H(P,Q) = H(P) + D_{KL}(P\,||\,Q)$, η οποία είναι παρόμοια με την KL απόκλιση, αλλά χωρίς τον όρο στα αριστερά:

$$H(P,Q) = -E_{X\sim P}\, log(Q(X))$$

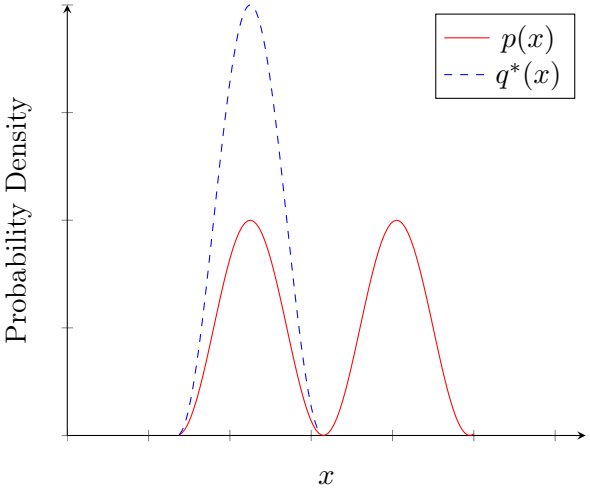
Η ελαχιστοποίηση του cross-entropy ως προς Q είναι ισοδύναμη με την ελαχιστοποίηση της KL απόκλισης, επειδή το Q δεν συμμετέχει στον παραλειπόμενο όρο. Υπολογίζοντας πολλές από αυτές τις ποσότητες, πολύ συχνά μπορεί να συναντήσουμε εκφράσεις της μορφής $0 \cdot log\, 0$. Κατά κανόνα, στα πλαίσια της Θεωρίας Πληροφοριών, χειριζόμαστε αυτές τις εκφράσεις ως $\lim_{x\to 0} x \cdot log\, x = 0$.

Η KL απόκλιση είναι ασύμμετρη:

$q^* = argmin_q D_{KL}(p \,||\, q)$



$q^* = argmin_q D_{KL}(q \,||\, p)$



A.7 Mean Absolute Error (MAE)

Μέσο Απόλυτο Σφάλμα

$$\frac{\sum_{i=1}^{N} |X_i - X_{recon\_i}|}{N}$$

A.8 Mean Squared Error (MSE)

Μέσο Τετραγωνικό Σφάλμα

$$\frac{\sum_{i=1}^{N} (X_i - X_{recon\_i})^2}{N}$$

A.9 Το Ξυράφι του Occam

Το ξυράφι του Occam είναι επιστημονική αρχή που δηλώνει ότι μεταξύ συγκρουόμενων υποθέσεων, αυτή με τους λιγότερες εικασίες πρέπει να υπερισχύει. Αποδίδεται στον Άγγλο φιλόσοφο Λογικής και φραγκισκανό μοναχό του 14ου αιώνα, Γουλιέλμο του Όκαμ. Αυτή η αρχή στα Λατινικά εκφράζεται ως εξής: **"Pluralitas non est ponenda sine neccesitate"**. Το γνωμικό αυτό πρωτοδιατυπώνεται από τους Πυθαγόρειους για περισσότερο από δυο χιλιετίες, όπως μας πληροφορεί ο φιλόσοφος Πρόκλος, του οποίου το Πανεπιστήμιο ήταν στην οδό Ηρώδου Αττικού, θαμμένο κάτω από το πεζοδρόμιο, μπροστά στο θέατρο του Διονύσου. Στα Αρχαία Ελληνικά διατυπώνεται ως εξής: **"ἐξ ἐλαχίστων καὶ ἁπλουστάτων ὑποθέσεων δεικνύναι τὰ ζητούμενα"**. Στην απλούστερη διατύπωσή του, το ξυράφι του Occam εκφράζεται ως εξής: **"Κανείς δεν θα πρέπει να προβαίνει σε περισσότερες εικασίες από όσες είναι απαραίτητες"**.

A.10 Root Mean Squared Error (RMSE)

Ρίζα Μέσου Τετραγωνικού Σφάλματος

$$\sqrt{\frac{\sum_{i=1}^{N} (X_i - X_{recon\_i})^2}{N}}$$

Α.11 Κανόνες Ενημέρωσης Βαρών των Νευρωνικών Δικτύων

Αυτοί είναι οι γενικευμένοι κανόνες ενημέρωσης βαρών που χρησιμοποιούνται στους αλγορίθμους προς τα πίσω διάδοσης (back-propagation).

- **Batch gradient descent**
  Κανόνες ενημέρωσης για τις παραμέτρους του κωδικοποιητή (encoder):

$$W = W - \eta \cdot \frac{\partial E}{\partial W}$$

- **Stochastic gradient descent**
  Κανόνες ενημέρωσης για τις παραμέτρους του κωδικοποιητή (encoder):

$$for\ each\ x_i : W = W - \eta \cdot \frac{\partial E_i}{\partial W}$$

- **Batch gradient ascent**
  Κανόνες ενημέρωσης για τις παραμέτρους του κωδικοποιητή (encoder):

$$W = W + \eta \cdot \frac{\partial E}{\partial W}$$

- **Stochastic gradient ascent**
  Κανόνες ενημέρωσης για τις παραμέτρους του κωδικοποιητή (encoder):

$$for\ each\ x_i : W = W + \eta \cdot \frac{\partial E_i}{\partial W}$$

όπου W είναι τα βάρη του Νευρωνικού Δικτύου και E είναι η συνάρτηση κόστους που θα μεγιστοποιηθεί (gradient ascent) ή ελαχιστοποιηθεί (gradient descent).

Παράρτημα Β

Πιθανοτικές Κατανομές

Β.1 Κατανομή Bernoulli

(Christopher M. Bishop 2006. [8]) Αυτή είναι η κατανομή για μια απλή δυαδική μεταβλητή $x \in 0, 1$ που αναπαριστά, για παράδειγμα, το αποτέλεσμα της ρίψης ενός νομίσματος. Κυριαρχείται από μια και μόνο συνεχή παράμετρο $\mu \in [0, 1]$ που αναπαριστά την πιθανότητα του $x = 1$.

$$Bern(x|\mu) = \mu^x \cdot (1 - \mu)^{1-x}$$

$$E[x] = \mu$$

$$VAR[x] = \mu \cdot (1 - \mu)$$

$$mode[x] = \begin{cases} 1, \ if \ \mu >= 0.5 \\ 0, \ \text{διαφορετικά} \end{cases}$$

$$H[x] = -\mu \cdot ln\mu - (1 - \mu) \cdot ln(1 - \mu)$$

Η Bernoulli είναι μια ειδική περίπτωση της διωνυμικής κατανομής στην περίπτωση μιας μεμονωμένης παρατήρησης. Η συζυγής εκ των προτέρων (conjugate prior) κατανομή της για μ είναι η κατανομή beta.

pmf	**Mean**	**Variance**
$\begin{cases} q = (1 - p), \ for \ k = 0 \\ p, \ for \ k = 1 \end{cases}$	p	$p \cdot (1 - p) = p \cdot q$

Πινακας Β.1: Κατανομή Bernoulli, συνάρτηση μάζας πιθανότητας (pmf), μέση τιμή και διασπορά.

Β.2 Διωνυμική Κατανομή

(Christopher M. Bishop 2006. [8]) Η διωνυμική κατανομή μας δίνει την πιθανότητα της παρατήρησης m περιστατικών της μεταβλητής $x = 1$ σε ένα σύνολο από N δείγματα από μια Bernoulli κατανομή, όπου η πιθανότητα της μεταβλητής $x = 1$ είναι $\mu \in [0, 1]$.

$$Bin(k|N, \mu) = \binom{N}{k} \cdot \mu^k \cdot (1-\mu)^{N-k}$$

$$E[k] = N \cdot \mu$$

$$VAR[k] = N \cdot \mu \cdot (1-\mu)$$

$$mode[k] = \lfloor (N+1) \cdot \mu \rfloor$$

όπου $\lfloor (N+1) \cdot \mu \rfloor$ συμβολίζει το μεγαλύτερο ολοκλήρωμα που είναι μικρότερο ή ίσο με $(N+1) \cdot \mu$, και η ποσότητα:

$$\binom{N}{k} = \frac{N!}{k!(N-k)!}$$

συμβολίζει τον αριθμό των τρόπων που μπορούμε να επιλέξουμε m αντικείμενα από ένα σύνολο N πανομοιότυπων αντικειμένων.

Εδώ το $m!$, το οποίο προφέρεται ως *'παραγοντικό m'*, συμβολίζει το γινόμενο $m \times (m-1) \times \ldots, \times 2 \times 1$. Η ιδιαίτερη περίπτωση όπου το $N = 1$ είναι γνωστή ως η κατανομή Bernoulli, και για μεγάλα N η διωνυμική κατανομή είναι προσεγγιστικά Gaussian. Η συζυγής εκ των προτέρων (conjugate prior) κατανομή της για μ είναι η beta κατανομή.

pmf	**Mean**	**Variance**
$\binom{n}{k} = \frac{n!}{k!(n-k)!}$	$n \cdot p$	$n \cdot p \cdot (1-p)$

Πινακας Β.2: Διωνυμική κατανομή, συνάρτηση μάζας πιθανότητας (pmf), μέση τιμή και διασπορά.

Β.3 Κατανομή Gauss (ή Κανονική Κατανομή)

(Christopher M. Bishop 2006. [8]) Η Gaussian είναι η πιο ευρέως χρησιμοποιημένη κατανομή για συνεχείς μεταβλητές. Είναι επίσης γνωστή ως κανονική κατανομή. Στην περίπτωση μιας μεμονωμένης μεταβλητής $x \in (-\infty, \infty)$ κυριαρχείται από δύο παραμέτρους, τη μέση τιμή $\mu \in (-\infty, \infty)$ και τη διασπορά $\sigma^2 > 0$.

$$N(x|\mu, \sigma^2) = \frac{1}{\sqrt{2 \cdot \pi\sigma^2}} \cdot e^{-\frac{1}{2 \cdot \sigma^2} \cdot (x-\mu)^2}$$

$$E[x] = \mu$$

$$VAR[x] = \sigma^2$$

$$mode[x] = \mu$$

$$H[x] = \frac{1}{2} \cdot ln\sigma^2 + \frac{1}{2} \cdot (1 + ln(2 \cdot \pi))$$

Το αντίστροφο της διασποράς $\tau = \frac{1}{\sigma^2}$ ονομάζεται ακρίβεια, και η τετραγωνική ρίζα της διασποράς Σ ονομάζεται τυπική απόκλιση. Η συζυγής εκ των προτέρων (conjugate prior) κατανομή της για μ είναι πάλι η κατανομή Gauss, ενώ η συζυγής εκ των προτέρων κατανομή της για τ είναι η κατανομή Gaussian-gamma. Για το διάνυσμα D διαστάσεων x, η κατανομή Gauss κυριαρχείται από το D-διάστατο διάνυσμα μέσων τιμών μ και ο $D \times D$ πίνακας συνδιακύμανσης Σ πρέπει να είναι συμμετρικός και θετικά ορισμένος.

$$N(x|\mu, \Sigma) = \frac{1}{(2 \cdot \pi)^{D/2}} \cdot \frac{1}{\sqrt{|\Sigma|}} \cdot \mathrm{e}^{-\frac{1}{2} \cdot (x-\mu)^T \cdot \Sigma^{-1} \cdot (x-\mu)}$$

$$E[x] = \mu$$

$$Cov[x] = \Sigma$$

$$mode[x] = \mu$$

$$H[x] = \frac{1}{2} \cdot ln|\Sigma| + \frac{D}{2} \cdot (1 + ln(2 \cdot \pi))$$

Ο αντίστροφος του πίνακα συνδιακύμανσης $\Lambda = \Sigma^{-1}$ είναι ο πίνακας ακρίβειας, ο οποίος είναι επίσης θετικός και θετικά ορισμένος. Σύμφωνα με το κεντρικό οριακό θεώρημα, οι μέσες τιμές τυχαίων μεταβλητών τείνουν σε κατανομή Gauss και το άθροισμα δύο Gaussian μεταβλητών είναι πάλι Gaussian. Η Gaussian είναι κατανομή που μεγιστοποιεί την εντροπία για μια δεδομένη διασπορά (ή συνδιακύμανση). Κάθε γραμμικός μετασχηματισμός μιας Gaussian τυχαίας μεταβλητής είναι πάλι Gaussian. Η περιθώρια (marginal) κατανομή μιας πολυποίκιλης (multivariate) Gaussian ως προς ένα υποσύνολο των μεταβλητών είναι και αυτή Gaussian, και παρομοίως η υπό συνθήκη (conditional)

κατανομή είναι επίσης Gaussian. Αν έχουμε περιθώρια κατανομή Gauss για x και υπό συνθήκη κατανομή Gauss για y δεδομένου του x υπό τη μορφή:

$$P(x) = N(x|\mu, \Lambda^{-1)}$$

$$P(y|x) = N(y|A \cdot x + b, L^{-1})$$

τότε η περιθώρια κατανομή του y, και η υπό συνθήκη κατανομή του x δεδομένου του y, ορίζονται ως εξής:

$$P(y) = N(y|A \cdot \mu + b, L^{-1} + A \cdot \Lambda^{-1} \cdot A^T)$$

$$P(x|y) = N(x|\Sigma A^T \cdot L(y - b) + \Lambda \cdot \mu, \Sigma)$$

όπου:

$$\Sigma = (\Lambda + A^T \cdot L \cdot A)^{-1}.$$

Αν έχουμε μια από κοινού κατανομή Gauss $N(x|\mu, \Sigma)$ με $\Lambda = \Sigma^{-1}$ και ορίζουμε τους παρακάτω διαχωρισμούς:

$$x = \begin{bmatrix} x_a \\ x_b \end{bmatrix}, \mu = \begin{bmatrix} \mu_a \\ \mu_b \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix}, \Lambda = \begin{bmatrix} \Lambda_{aa} & \Lambda_{ab} \\ \Lambda_{ba} & \Lambda_{bb} \end{bmatrix}$$

τότε η υπό συνθήκη κατανομή $P(x_a|x_b)$ δίνεται από τον τύπο:

$$P(x_a|x_b) = N(x|\mu_{a|b}, \Lambda_{aa}^{-1})$$

$$\mu_{a|b} = \mu_a - \Lambda_{aa}^{-1} \cdot \Lambda ab \cdot (x_b - \mu_b)$$

και η περιθώρια κατανομή $P(x_a)$ δίνεται από τον τύπο:

$$P(x_a) = N(x_a|\mu_a, \Sigma_a).$$

pmf	**Mean**	**Variance**
$\frac{1}{\sqrt{2\cdot\pi\sigma^2}} \cdot e^{-\frac{1}{2\cdot\sigma^2}\cdot(x-\mu)^2}$	μ	σ^2

Πινακας Β.3: Κατανομή Gauss, συνάρτηση μάζας πιθανότητας (pmf), μέση τιμή και διασπορά.

Β.4 Νόμος Μεγάλος Αριθμών (Ν.Μ.Α.)

(Σταύρος Τουμπής, Γιάννης Κοντογιάννης 2011. [9]) Το 1^{st} θεμελιώδες θεώρημα της θεωρίας πιθανοτήτων είναι ο Νόμος των Μεγάλων Αριθμών (Ν.Μ.Α.).

Ορισμός

Έστω $X_i, i = 1, 2, \ldots$ μια ακολουθία από ανεξάρτητες τυχαίες μεταβλητές (Τ.Μ.) με την ίδια κατανομή, μέση τιμή $E(X) = \mu$ και διασπορά $VAR(X) = \sigma^2$. Το 1^{st} θεμελιώδες αποτέλεσμα των πιθανοτήτων, ο Ν.Μ.Α., μας εξασφαλίζει ότι η πιθανότητα ο εμπειρικός μέσος όρος:

$$\bar{X}_N \approx \frac{1}{N} \cdot \sum_{i=1}^{N} X_i$$

να παρουσιάζει απόκλιση από το μ μεγαλύτερη από ϵ τείνει στο 0, δηλαδή:

$$\lim_{N\to\infty} P(|\bar{X}_N - \mu| > \epsilon) = 0$$

Επιπλέον, το ακριβές ποσοτικό άνω φράγμα για την πιθανότητα ο εμπειρικός μέσος όρος $\bar{X}_N$: να απέχει από τη μέση τιμή μ κατά τουλάχιστον ϵ έχει αποδειχθεί ότι είναι ίσο με:

$$P(|\bar{X}_N - \mu| \geq \epsilon) \leq \frac{\sigma^2}{N \cdot \epsilon^2}$$

όπου: $E(\bar{X}_N) = \mu$ και $Var(\bar{X}_N) = \frac{\sigma^2}{N}$, για κάθε N

Β.5 Κεντρικό Οριακό Θεώρημα (Κ.Ο.Θ.)

(Σταύρος Τουμπής, Γιάννης Κοντογιάννης 2011. [9]) Το 2^{nd} θεμελιώδες θεώρημα της θεωρίας πιθανοτήτων είναι το Κεντρικό Οριακό Θεώρημα (Κ.Ο.Θ.). Μέσω αυτού του θεωρήματος μπορούμε να εκτιμήσουμε με υψηλή ακρίβεια το όριο $\lim_{N\to\infty} P(|\bar{X}_N - \mu| > \epsilon)$. Συγκεκριμένα, το Κ.Ο.Θ. μας λέει ότι υπό συγκεκριμένες περιστάσεις, η κατανομή του εμπειρικού μέσου $\bar{X}_N$ μπορεί να υπολογιστεί με υψηλή ακρίβεια από την Gaussian (γνωστή και ως κανονική) κατανομή με παραμέτρους μ και $\frac{\sigma^2}{N}$, όπου σ^2 η διασπορά των τυχαίων μεταβλητών (Τ.Μ.) X_i.

Ορισμός

Έστω μια ακολουθία από ανεξάρτητες τυχαίες μεταβλητές (Τ.Μ.) $X_1, X_2, \ldots$ που έχουν όλες την ίδια κατανομή και κατά συνέπεια την ίδια μέση τιμή $\mu = E(X_i)$ και την ίδια διασπορά $\sigma^2 = VAR(X_i)$. Έστω το κανονικοποιημένο άθροισμα:

$$\bar{S}_N = \frac{(\frac{1}{N} \cdot \sum_{i=1}^{N} X_i - \mu)}{\sigma \cdot \sqrt{\frac{1}{N}}} = \frac{1}{\sigma \cdot \sqrt{N}} \cdot \sum_{i=1}^{N} (X_i - \mu) = \frac{(\sum_{i=1}^{N} X_i) - N \cdot \mu}{\sigma \cdot \sqrt{N}}$$

Έστω επίσης $a, b \in \mathbb{R}, a < b$. Ισχύουν τα ακόλουθα, καθώς $N \to \infty$:

- $P(a \leq \bar{S}_N \leq b) \to \Phi(b) - \Phi(a)$
- $P(\bar{S}_N \leq b) \to \Phi(b)$
- $P(\bar{S}_N \geq a) \to 1 - \Phi(a)$

όπου $\Phi(x)$ είναι η Gaussian (γνωστή και ως κανονική) κατανομή

Παράρτημα C

Προγραμματιστικές Υλοποιήσεις για Variational Autoencoder σε Python

C.1 Υλοποίηση VAE με χρήση βιβλιοθήκης TensorFlow

Ακολουθεί μια υλοποίηση VAE αλγορίθμου, με βιβλιοθήκη TensorFlow:

```
import numpy as np

X_train  # matrix with the original data
y_train  # labels of the matrix "X_train"
N # the number of training examples
input_dim  # the input dimensionality D
hidden_encoder_dim  # number of neurons in the encoder
hidden_decoder_dim  # number of neurons in the decoder
latent_dim  # Z_dim
epochs  # the number of total epochs for the training
batch_size  # the number of data trained in each iteration
learning_rate  # the rate of learning "eta" for the optimization algorithm
log_dir  # the location where the tensorboard graph will be saved

x, loss_summ, apply_updates, summary_op, saver, elbo, x_recon_samples =
    vae(batch_size, input_dim, hidden_encoder_dim, hidden_decoder_dim, latent_dim,
    lr)

X_recon = np.zeros((N, input_dim))

with tf.Session() as sess:
    summary_writer = tf.summary.FileWriter(log_dir, graph=sess.graph)

    for epoch in range(epochs):
        iterations = int(N / batch_size)

        for i in range(iterations):
            start_index = i * batch_size
            end_index = (i + 1) * batch_size

            batch_data = X_train[start_index:end_index, :]
```

```
31              batch_labels = y_train[start_index:end_index]
32
33              feed_dict = {x: batch_data}
34              loss_str, _, summary_str, cur_elbo, cur_samples =
35                  sess.run([loss_summ, apply_updates, summary_op,
36                  elbo, x_recon_samples], feed_dict=feed_dict)
37
38              X_recon[start_index:end_index, :] = cur_samples
39
40              summary_writer.add_summary(loss_str, epoch)
41              summary_writer.add_summary(summary_str, epoch)
```

ΚΩΔΙΚΑΣ C.1: Κύριος Βρόχος του VAE σε TensorFlow

Ακολουθεί η υλοποίηση της συνάρτησης "vae", της υλοποίησης με TensorFlow:

```
import tensorflow as tf

# HELPER FUNCTIONS #

def initialize_weight_variable(shape, name=''):
    initial = tf.truncated_normal(shape, stddev=0.001,
                name='truncated_normal_' + name)
    return tf.Variable(initial, name=name)


def initialize_bias_variable(shape, name=''):
    initial = tf.constant(0., shape=shape)
    return tf.Variable(initial, name=name)

#####

def vae(batch_size, input_dim, hidden_encoder_dim, hidden_decoder_dim,
                latent_dim, lr=0.01):
    # Input placeholder
    with tf.name_scope('input_data'):
        x = tf.placeholder('float', [batch_size, input_dim], name='X')

    # ====== Q(Z|X) = Q(Z) - Encoder NN ====== #

    # The encoder is a neural network with 2 hidden layers.
    with tf.name_scope('encoder'):
        with tf.name_scope('Phis'):
            # phi1: M1 x D
            phi1 = initialize_weight_variable([hidden_encoder_dim,
                        input_dim], name='phi1')
            # bias_phi1: 1 x M1
            bias_phi1 = initialize_bias_variable([hidden_encoder_dim],
                        name='bias_phi1')

            # phi_mu: Z_dim x M1
            phi_mu = initialize_weight_variable([latent_dim,
                        hidden_encoder_dim], name='phi_mu')
            # bias_phi_mu: 1 x Z_dim
            bias_phi_mu = initialize_bias_variable([latent_dim],
                                name='bias_phi_mu')

            # phi_logvar: Z_dim x M1
            phi_logvar = initialize_weight_variable([latent_dim,
                            hidden_encoder_dim], name='phi_logvar')
            # bias_phi_logvar: 1 X Z_dim
            bias_phi_logvar = initialize_bias_variable([latent_dim],
```

```
                                    name='bias_phi_logvar')

        with tf.name_scope('hidden_layer1'):
            # Hidden layer 1 activation function of the encoder
            # hidden_layer_encoder: N x M1
            # RELU
            hidden_layer_encoder = tf.nn.relu(tf.nn.bias_add(tf.matmul(
                                    x, tf.transpose(phi1)),
                                    bias_phi1))

        with tf.name_scope('hidden_layer2_mu'):
            # mu_encoder: N x Z_dim
            mu_encoder = tf.nn.bias_add(tf.matmul(hidden_layer_encoder,
                            tf.transpose(phi_mu)), bias_phi_mu)

        with tf.name_scope('hidden_layer2_logvar'):
            # the log sigma^2 of the encoder
            # logvar_encoder: N x Z_dim
            logvar_encoder = tf.nn.bias_add(tf.matmul(hidden_layer_encoder,
                                tf.transpose(phi_logvar)),
                                bias_phi_logvar)

        with tf.name_scope('sample_E'):
            # Sample epsilon
            # epsilon: N x Z_dim
            epsilon = tf.random_normal((batch_size, latent_dim),
                        mean=0.0, stddev=1.0, name='epsilon')

        with tf.name_scope('construct_Z'):
            # Sample epsilon from the Gaussian distribution. #
            # std_encoder: N x Z_dim
            std_encoder = tf.exp(logvar_encoder / 2)
            # z: N x Z_dim
            z = tf.add(mu_encoder, tf.multiply(std_encoder, epsilon),
                    name='Z')

    # ===== P(X|Z) - Decoder NN ===== #

    # The encoder is a neural network with 2 hidden layers.
    with tf.name_scope('decoder'):
        with tf.name_scope('Thetas'):
            # theta1: M2 x Z_dim
            theta1 = initialize_weight_variable([hidden_decoder_dim,
                        latent_dim], name='theta1')
            # bias_theta1: 1 x M2
            bias_theta1 = initialize_bias_variable([hidden_decoder_dim],
                        name='bias_theta1')

```

```
95              # theta2: D x M2
96              theta2 = initialize_weight_variable([input_dim,
97                          hidden_decoder_dim], name='theta2')
98              # bias_theta2: 1 x D
99              bias_theta2 = initialize_bias_variable([input_dim],
100                         name='bias_theta2')
101
102         with tf.name_scope('hidden_layer1'):
103             # Hidden layer 1 activation function of the decoder
104             # hidden_layer_decoder: N x M2
105             # RELU
106             hidden_layer_decoder = tf.nn.relu(tf.nn.bias_add(tf.matmul(z,
107                         tf.transpose(theta1)), bias_theta1))
108
109         with tf.name_scope('output_layer'):
110             # x_hat: N x D
111             x_hat = tf.nn.bias_add(tf.matmul(hidden_layer_decoder,
112                         tf.transpose(theta2)), bias_theta2)
113
114     with tf.name_scope('reconstructed_data'):
115         # X_recon_samples: N x D, reconstructed data
116         x_recon_samples = tf.nn.sigmoid(x_hat, name='X_recon_samples')
117
118     # ======= TRAINING ========== #
119
120     with tf.name_scope('ELBO'):
121         with tf.name_scope('reconstruction_cost'):
122             # log P(X)
123             # reconstruction_cost: N x 1
124             reconstruction_cost = tf.reduce_sum(tf.nn.
125                     sigmoid_cross_entropy_with_logits(logits=x_hat,
126                     labels=x), reduction_indices=1)
127
128         with tf.name_scope('KL_divergence'):
129             # KLD: N x 1
130             KLD = 0.5 * tf.reduce_sum(1 + logvar_encoder -
131                     tf.square(mu_encoder) - tf.exp(logvar_encoder),
132                     reduction_indices=1)
133
134         elbo = tf.reduce_mean(reconstruction_cost - KLD,
135                     name='lower_bound')
136
137         loss_summ = tf.summary.scalar('ELBO', elbo)
138
139         phis = [phi1, bias_phi1,
140                 phi_mu, bias_phi_mu,
141                 phi_logvar, bias_phi_logvar]
142         thetas = [theta1, bias_theta1,
```

```
143                     theta2, bias_theta2]
144         var_list = phis + thetas
145 
146         # Adam Optimizer (WORKS BEST!) #
147         grads_and_vars = tf.train.AdamOptimizer(learning_rate=lr). \
148                          compute_gradients(loss=elbo, var_list=var_list)
149         apply_updates = tf.train.AdamOptimizer(learning_rate=lr). \
150                         apply_gradients(grads_and_vars=grads_and_vars)
151 
152     # add op for merging summary
153     summary_op = tf.summary.merge_all()
154 
155     # add Saver ops
156     saver = tf.train.Saver()
157 
158     return x, loss_summ, apply_updates, summary_op, saver,
159             elbo, x_recon_samples
```

ΚΩΔΙΚΑΣ C.2: Συνάρτηση VAE σε TensorFlow

ΣΗΜΕΙΩΣΗ: Για το βελτιστοποιητή (optimizer) των παραγώγων (gradient), χρησιμοποιήσαμε τον Adam optimizer (tf.train.AdamOptimizer), επειδή επιφέρει τα καλύτερα αποτελέσματα. Όλοι οι optimizer που παρέχει η βιβλιοθήκη TensorFlow είναι οι εξής:

- tf.train.GradientDescentOptimizer
- tf.train.AdadeltaOptimizer
- tf.train.AdagradOptimizer
- tf.train.AdagradDAOptimizer
- tf.train.MomentumOptimizer
- **tf.train.AdamOptimizer**
- tf.train.FtrlOptimizer
- tf.train.ProximalGradientDescentOptimizer
- tf.train.ProximalAdagradOptimizer
- tf.train.RMSPropOptimizer

C.2 Υλοποίηση VAE με χρήση βιβλιοθήκης PyTorch

Ακολουθεί μια υλοποίηση VAE αλγορίθμου, με βιβλιοθήκη PyTorch:

```
import numpy as np

X_train  # matrix with the original data
y_train  # labels of the matrix "X_train"
N  # the number of training examples
input_dim  # the input dimensionality D
hidden_encoder_dim  # number of neurons in the encoder
hidden_decoder_dim  # number of neurons in the decoder
latent_dim  # Z_dim
epochs  # the number of total epochs for the training
batch_size  # the number of data trained in each iteration
learning_rate  # the rate of learning "eta" for the optimization algorithm

# initialize_weights:
# A function that initializes the weights of the encoder and
# the decoder neural networks.
params # the weights of the encoder and the decoder neural networks,
       # that are updated on each iteration
solver # the optimizer used for training (e.g SGD)
params, solver = initialize_weights(input_dim, hidden_encoder_dim, \
                           hidden_decoder_dim, latent_dim, lr)

for epoch in range(epochs):
    iterations = int(N / batch_size)

    for i in range(iterations):
        start_index = i * batch_size
        end_index = (i + 1) * batch_size

        batch_data = X_train[start_index:end_index, :]
        batch_labels = y_train[start_index:end_index]

        cur_samples, cur_elbo = train(batch_data, batch_size, \
                                      latent_dim, params, solver)

        X_train[start_index:end_index, :] = cur_samples
```

Κωδικας C.3: Κύριος Βρόχος του VAE σε PyTorch

Ακολουθεί η υλοποίηση της συνάρτησης "initialize_weights" της υλοποίησης με PyTorch:

```
import numpy as np
import torch
import torch.optim as optim
from torch.autograd import Variable


def xavier_init(size):
    in_dim = size[0]
    xavier_stddev = 1. / np.sqrt(in_dim / 2.)
    return Variable(torch.randn(*size) * xavier_stddev,
                    requires_grad=True)


def initialize_weights(X_dim, hidden_encoder_dim, hidden_decoder_dim,
                       Z_dim, lr=0.01):

    # ====== Encoder Parameters: Phis ====== #

    # M1 x D
    phi1 = xavier_init(size=[hidden_encoder_dim, X_dim])
    # 1 x M1
    bias_phi1 = Variable(torch.zeros(hidden_encoder_dim),
                         requires_grad=True)

    # Z_dim x M1
    phi_mu = xavier_init(size=[Z_dim, hidden_encoder_dim])
    # 1 x Z_dim
    bias_phi_mu = Variable(torch.zeros(Z_dim),
                           requires_grad=True)

    # Z_dim x M1
    phi_logvar = xavier_init(size=[Z_dim, hidden_encoder_dim])
    # 1 x Z_dim
    bias_phi_logvar = Variable(torch.zeros(Z_dim),
                               requires_grad=True)

    # ====== Decoder Parameters: Thetas ====== #

    # M2 x Z_dim
    theta1 = xavier_init(size=[hidden_decoder_dim, Z_dim])
    # 1 x M2
    bias_theta1 = Variable(torch.zeros(hidden_decoder_dim),
                           requires_grad=True)

    # D x M2
    theta2 = xavier_init(size=[X_dim, hidden_decoder_dim])
```

```
47    # 1 x D
48    bias_theta2 = Variable(torch.zeros(X_dim),
49                           requires_grad=True)
50
51    # ====== TRAINING ======
52
53    phis = [phi1, bias_phi1,
54            phi_mu, bias_phi_mu,
55            phi_logvar, bias_phi_logvar]
56    thetas = [theta1, bias_theta1,
57          theta2, bias_theta2]
58    params = phis + thetas
59
60    solver = optim.Adam(params, lr=lr)
61
62    return params, solver
```

Κωδικας C.4: Αρχικοποίηση των Βαρών του VAE σε PyTorch

ΣΗΜΕΙΩΣΗ: Για το βελτιστοποιητή (optimizer) των παραγώγων (gradient), χρησιμοποιήσαμε τον Adam optimizer (torch.optim.Adam), επειδή επιφέρει τα καλύτερα αποτελέσματα. Όλοι οι optimizer που παρέχει η βιβλιοθήκη PyTorch είναι οι εξής:

- torch.optim.Adadelta
- torch.optim.Adagrad
- **torch.optim.Adam**
- torch.optim.SparseAdam
- torch.optim.AdaMax
- torch.optim.ASGD
- torch.optim.LBFGS
- torch.optim.RMSprop
- torch.optim.Rprop
- torch.optim.SGD

Ακολουθεί η υλοποίηση της συνάρτησης "train" της υλοποίησης με PyTorch:

```
import torch
import torch.nn.functional as nn
from torch.autograd import Variable


# @: denotes matrix multiplication
def train(x, mb_size, Z_dim, params, solver):
    x = Variable(torch.from_numpy(x).float())

    # Phis
    phi1 = params[0]  # M1 x D
    bias_phi1 = params[1]  # 1 x M1
    phi_mu = params[2]  # Z_dim x M1
    bias_phi_mu = params[3]  # 1 x Z_dim
    phi_logvar = params[4]  # Z_dim x M1
    bias_phi_logvar = params[5]  # 1 x Z_dim

    # Thetas
    theta1 = params[6]  # M2 x Z_dim
    bias_theta1 = params[7]  # 1 x M2
    theta2 = params[8]  # D x M2
    bias_theta2 = params[9]  # 1 x D

    # ======= Q(Z|X) = Q(Z) - Encoder NN ======= #

    hidden_layer_encoder = nn.relu(x @ torch.transpose(phi1, 0, 1) +
                           bias_phi1.repeat(mb_size, 1))
    mu_encoder = hidden_layer_encoder @ torch.transpose(phi_mu, 0, 1) +
                 bias_phi_mu.repeat(hidden_layer_encoder.size(0), 1)
    logvar_encoder = hidden_layer_encoder @ torch.transpose(phi_logvar, 0, 1)
                + bias_phi_logvar.repeat(hidden_layer_encoder.size(0), 1)

    # Sample epsilon from the Gaussian distribution. #
    epsilon = Variable(torch.randn(mb_size, Z_dim))
    # std_encoder: N x Z_dim
    std_encoder = torch.exp(logvar_encoder / 2)
    # Sample the latent variables Z. #
    z = mu_encoder + std_encoder * epsilon

    # ======= P(X|Z) - Decoder NN ======= #

    hidden_layer_decoder = nn.relu(z @ torch.transpose(theta1, 0, 1) +
                           bias_theta1.repeat(z.size(0), 1))
    x_hat = hidden_layer_decoder @ torch.transpose(theta2, 0, 1) +
            bias_theta2.repeat(hidden_layer_decoder.size(0), 1)
    x_recon_samples = nn.sigmoid(x_hat)
```

```
47
48      # Loss #
49      recon_loss = nn.binary_cross_entropy(x_recon_samples,
50                   x, size_average=False) / mb_size
51      kl_loss = torch.mean(0.5 * torch.sum(1 + logvar_encoder
52                - mu_encoder ** 2 - torch.exp(logvar_encoder), 1))
53      elbo_loss = recon_loss - kl_loss
54
55      # Backward #
56      elbo_loss.backward()
57
58      # Update #
59      solver.step()
60
61      # Housekeeping #
62      for p in params:
63          p.grad.data.zero_()
64
65      # convert to numpy data type and return
66      x_recon_samples = x_recon_samples.data.numpy()
67      elbo_loss = elbo_loss.data.numpy()[0]
68
69      return x_recon_samples, elbo_loss
```

Κωδικας C.5: Συνάρτηση Εκπαίδευσης του VAE σε PyTorch

C.3 Υλοποίηση VAE με χρήση βιβλιοθήκης Keras

Ακολουθεί υλοποίηση VAE αλγορίθμου, με βιβλιοθήκη Keras (χρησιμοποιώντας το TensorFlow ως backend):

```
import numpy as np
from keras.callbacks import TensorBoard
from keras.layers import Input, Dense
from keras.models import Model

X_train  # matrix with the original train data
X_test  # matrix with the test data
X_valid  # matrix with the cross-validation data
input_dim  # the input dimensionality D
latent_dim  # Z_dim
epochs  # the number of total epochs for the training
batch_size # the number of data trained in each iteration
log_dir # the location where the tensorboard graph will be saved

# this is our input placeholder
input_img = Input(shape=(input_dim,))

# 'encoded' is the encoded representation of the input
encoded = Dense(latent_dim, activation='relu')(input_img)
# 'decoded' is the lossy reconstruction of the input
decoded = Dense(input_dim, activation='sigmoid')(encoded)

# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)

# ====== Q(Z|X) = Q(Z) - Encoder NN ====== #

# this model maps an input to its encoded representation
encoder = Model(input_img, encoded)

# ====== P(X|Z) - Decoder NN ====== #

# create a placeholder for an encoded latent_dim input
encoded_input = Input(shape=(latent_dim,))
# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# create the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))

# compile the model
# optimizer: Adadelta
# loss function: binary crossentropy loss
```

```
43 autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
44
45 # fit the model
46 autoencoder.fit(X_train, X_train,
47                 epochs=epochs,
48                 batch_size=batch_size,
49                 shuffle=True,
50                 validation_data=(X_valid, X_valid),
51                 callbacks=[TensorBoard(log_dir=log_dir,
52                 histogram_freq=0, write_graph=False)])
53
54 # encode and decode some digits
55 # note that we take them from the *test* set
56 encoded_imgs = encoder.predict(X_test)
57 decoded_imgs = decoder.predict(encoded_imgs)
```

Κωδικας C.6: Κύριος Βρόχος του VAE σε Keras

C.4 K-NN Αλγόριθμος Συμπλήρωσης Ελλιπών Τιμών σε Python

Ακολουθεί ο πλήρης κώδικας για τον K-NN αλγόριθμο συμπλήρωσης ελλιπών τιμών σε Python:

```
X_train_missing  # TRAIN data with missing values
X_test_missing  # TEST data with missing values
K = 1
missing_value = 0.5

X_test_predicted = kNNMatrixCompletion(X_train_missing,
                    X_test_missing, K, missing_value)

# Customized Euclidean distance function.
def sqdist(X_train, X_test_instance, missing_value):
    Ntrain = X_train.shape[0]
    D = X_test.shape[1]

    X_train_common = np.array(X_train)
    s = np.where(X_test_instance == missing_value)
    X_train_common[:, s] = missing_value

    X_test_common = np.zeros((Ntrain, D))
    mean_values = np.mean(X_train, axis=0)  # 1 x D array
    for k in range(Ntrain):
        X_test_common[k, :] = X_test_instance
        s = np.where(X_train[k, :] == missing_value)
        if len(s[0]) != 0:
            X_test_common[k, s] = mean_values[s]

    dist = np.matrix(np.sqrt(np.sum(np.square(X_train_common - X_test_common),
            axis=1))).T
    return dist

def kNNMatrixCompletion(X_train, X_test, K, missing_value,
                        use_softmax_weights=True, binarize=False):
    Ntest = X_test.shape[0]

    X_test_predicted = np.array(X_test)
    for i in range(Ntest):
        print('data%i' % i)
        X_test_i = np.squeeze(np.array(X_test[i, :]))

        distances = sqdist(X_train, X_test_i, missing_value)

        # sort the distances in ascending order
        # and store the indices in a variable
```

```
43          closest_data_indices = \
44              (np.argsort(distances, axis=0)).tolist()
45
46          # select the top k indices
47          closest_k_data_indices = closest_data_indices[:K]
48          closest_k_data = \
49              np.squeeze(X_train[closest_k_data_indices, :])
50
51          closest_k_data_distances = \
52                  np.squeeze(distances[closest_k_data_indices, :]).T
53
54          if use_softmax_weights:
55              # distribute weights, in the following manner:
56              # wk = e^(-k) / sum_{i=1}^{K} e^(-i)
57              # such that: w1 + w2 + ... wK = 1
58              a = 1
59              weights = softmax(-a * closest_k_data_distances)
60          else:
61              # ALTERNATIVE: all weights equal to 1 / K
62              weights = np.ones((K, 1)) / float(K)
63
64          closest_k_data = np.multiply(closest_k_data, weights)
65
66          # sum across all rows for each column, returns a column vector
67          predicted_pixels = np.sum(closest_k_data, axis=0).T
68
69          X_test_predicted[i, np.where(X_test_i == missing_value)] = \
70              predicted_pixels[np.where(X_test_i == missing_value)].T
71
72      if binarize:
73          X_test_predicted = np.round(X_test_predicted + 0.3)
74
75      return X_test_predicted
```

ΚΩΔΙΚΑΣ C.7: K-NN αλγόριθμος συμπλήρωσης ελλιπών τιμών σε Python

C.5 VAE Αλγόριθμος Συμπλήρωσης Ελλιπών Τιμών σε PyTorch

Ακολουθεί ο πλήρης κώδικας για το VAE αλγόριθμο συμπλήρωσης ελλιπών τιμών σε Python, με χρήση **PyTorch**.

```
import numpy as np

X_train  # matrix with the original data
y_train  # labels of the matrix "X_train"
X_train_missing  # matrix with missing values
X_train_masked  # matrix with 0s where the pixel are missing and
                # 1s where the pixel are not missing
X_filled = np.array(X_train_missing)  # the final matrix of predicted pixels
N  # the number of training examples
input_dim  # the input dimensionality D
hidden_encoder_dim  # number of neurons in the encoder
hidden_decoder_dim  # number of neurons in the decoder
latent_dim  # Z_dim
epochs  # the number of total epochs for the training
batch_size  # the number of data trained in each iteration
learning_rate  # the rate of learning "eta" for the optimization algorithm

# initialize_weights:
# initializes the weights of the encoder and the decoder.
# params: the weights of the encoder and the decoder
# solver: the optimizer used for training (e.g SGD)
params, solver = initialize_weights(input_dim, hidden_encoder_dim,
                    hidden_decoder_dim, latent_dim, lr)

for epoch in range(epochs):
    iterations = int(N / batch_size)

    for i in range(iterations):
        start_index = i * batch_size
        end_index = (i + 1) * batch_size

        batch_data = X_filled[start_index:end_index, :]
        batch_labels = y_train[start_index:end_index]
        masked_batch_data = X_train_masked[start_index:end_index, :]

        cur_samples, cur_elbo = train(batch_data, batch_size,
                                      latent_dim, params, solver)

        cur_samples = np.multiply(masked_batch_data, batch_data) +
                      np.multiply(1 - masked_batch_data, cur_samples)
        X_filled[start_index:end_index, :] = cur_samples
```

ΚΩΔΙΚΑΣ C.8: VAE αλγόριθμος συμπλήρωσης ελλιπών τιμών σε PyTorch

ΣΗΜΕΙΩΣΕΙΣ:

- Οι υλοποιήσεις των συναρτήσεων "initialize_weights" και "train" βρίσκονται στο Παράρτημα.
- Η υλοποίηση για το VAE αλγόριθμο συμπλήρωσης ελλιπών τιμών σε Python, με χρήση **TensorFlow**, δε συμπεριλαμβάνεται επειδή είναι παρόμοια με την υλοποίηση σε PyTorch.

Βιβλιογραφία

[1] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009. ISSN 1935-8237. doi: 10.1561/2200000006. http://dx.doi.org/10.1561/2200000006.

[2] Carl Doersch. Tutorial on variational autoencoders. *Variational Autoencoders*, June 2016. https://arxiv.org/abs/1606.05908.

[3] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *Variational Autoencoders*, December 2013. https://arxiv.org/abs/1312.6114.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. *Deep Learning*, 2016. http://www.deeplearningbook.org.

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[6] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Machine Learning*, pages 34–37, April 2009.

[7] A.S. Georghiades, P.N. Belhumeur, and D.J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Trans. Pattern Anal. Mach. Intelligence*, 23(6):643–660, 2001.

[8] Christopher M. Bishop. Pattern recognition and machine learning. *Machine Learning*, 2006.

[9] Γιάννης Κοντογιάννης και Σταύρος Τουμπής. Σημειώσεις μαθήματος "Πιθανοτήτων". *Οικονομικό Πανεπιστήμιο Αθηνών*, 2011.

Σύνδεσμοι

- Εικόνες Google https://images.google.com/
- Analytics Vidhya,
  An Introduction to Implementing Neural Networks using TensorFlow
  https://www.analyticsvidhya.com/blog/2016/10/an-introduction-to-implementing-neural-networks-using-tensorflow/
- Arxiv Insights YouTube κανάλι,
  Variational Autoencoders
  https://www.youtube.com/watch?v=9zKuYvjFFS8
- Augustinus Kristiadi's Blog, Many flavors of Autoencoder
  https://wiseodd.github.io/techblog/2016/12/03/autoencoders/
- Christopher Bourez's blog,
  Symbolic computing and deep learning tutorial with Tensorflow/Theano: learn basic commands of 2 libraries for the price of 1
  http://christopher5106.github.io/big/data/2016/03/06/symbolic-computing-and-deep-learning-tutorial-on-theano-and-google-tensorflow.html
- Edureka YouTube κανάλι,
  Introduction To TensorFlow | Deep Learning Using TensorFlow | TensorFlow Tutorial | Edureka
  https://www.youtube.com/watch?v=uh2Fh6df7Lg&t=1562s
- Fast Forward Labs Blog,

  1. Introducing Variational Autoencoders (in Prose and Code)
     http://blog.fastforwardlabs.com/2016/08/12/introducing-variational-autoencoders-in-prose-and.html

  2. Under the Hood of the Variational Autoencoder (in Prose and Code)
     http://blog.fastforwardlabs.com/2016/08/22/under-the-hood-of-the-variational-autoencoder-in.html

- Jaan Altosaar, Tutorial - What is a Variational Autoencoder?
  https://jaan.io/what-is-variational-autoencoder-vae-tutorial/
- kevin frans, Variational Autoencoders Explained
  http://kvfrans.com/variational-autoencoders-explained/

- Nando de Freitas YouTube κανάλι,
  Deep Learning Lecture 14: Karol Gregor on Variational Autoencoders and Image Generation
  https://www.youtube.com/watch?v=P78QYjWh5sM

- Siraj Raval YouTube κανάλι,
  How to Generate Images - Intro to Deep Learning #14
  https://www.youtube.com/watch?v=3-UDwk1U77s

- studio otoro, Generating Large Images from Latent Vectors
  http://blog.otoro.net/2016/04/01/generating-large-images-from-latent-vectors/

- The Keras Blog, Building Autoencoders in Keras
  https://blog.keras.io/building-autoencoders-in-keras.html

- videolectures.net,
  Building Machines that Imagine and Reason: Principles and Applications of Deep Generative Models
  http://videolectures.net/deeplearning2016_mohamed_generative_models/

- videolectures.net,
  Variational Autoencoder and Extensions
  http://videolectures.net/deeplearning2015_courville_autoencoder_extension/

- Xitong Yang's Blog, Understanding the Variational Lower Bound
  https://xyang35.github.io/2017/04/14/variational-lower-bound/

- wiseodd GitHub, Generative Models
  https://github.com/wiseodd/generative-models

- Βικιπαίδεια,

1. Autoencoder
   https://en.wikipedia.org/wiki/Autoencoder
2. Bayes' theorem
   https://en.wikipedia.org/wiki/Bayes'_theorem
3. Softmax function
   https://en.wikipedia.org/wiki/Softmax_function
4. Cross entropy
   https://en.wikipedia.org/wiki/Cross_entropy
5. Entropy (information theory)
   https://en.wikipedia.org/wiki/Entropy_(information_theory)
6. Jensen's inequality
   https://en.wikipedia.org/wiki/Jensen's_inequality
7. Mean absolute error
   https://en.wikipedia.org/wiki/Mean_absolute_error
8. Mean squared error
   https://en.wikipedia.org/wiki/Mean_squared_error
9. Kullback–Leibler divergence
   https://en.wikipedia.org/wiki/Kullback-Leibler_divergence
10. Ξυράφι του Όκαμ
    https://el.wikipedia.org/wiki/Ξυράφι_του_Όκαμ
11. Root-mean-square error
    https://en.wikipedia.org/wiki/Root-mean-square_deviation
12. TensorFlow
    https://en.wikipedia.org/wiki/TensorFlow
13. Variational Bayesian methods
    https://en.wikipedia.org/wiki/Variational_Bayesian_methods