Deep Learning

AlejandroMllo

This document serves as a very brief survey of the topics covered in each chapter of the book Deep Learning [1].

Disclaimer: This document is completely extracted from [1], the author does not attribute any ownership over the material.

1 Introduction

- Deep Learning (DL) learns complicated concepts by building them out of simpler ones. DL is the study of models composed of either learned functions or learned concepts.
- Data Representation is key when finding patterns.
- Many AI tasks can be solved by designing the right set of features to extract.
- **Representation Learning:** discovers the mapping from representation to output and the representation itself.
  - Makes use of an *Autoencoder* which is a combination of:
    - Encoder: Converts the input data into a different representation.
    - Decoder: Converts the new representation back into the original format.
- Factors: sources of influence in the model.
- **Depth**, in a DL model, enables the computer to learn a multistep computer program (not all the information in a layer's activations necessarily encodes factors that explain the input). There are two ways to measure it:
  - Based on the number of sequential instructions, or
  - Based on the depth of the graph describing how concepts are related to each other (used by deep probabilistic models).
- DL is not an attempt to simulate the brain. It borrows ideas from different fields (one of which is neuroscience).

2 Linear Algebra

- Scalar: a single number. Denoted by lowercase variable names and italics.
- Vectors: ordered array of numbers. Denoted by lowercase and bold typeface. $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$
- Matrices: ordered 2-D $m \times n$ array of numbers. Denoted by uppercase and bold typeface.

$$\mathbf{A} = \begin{bmatrix} A_{1,1} & \dots & A_{1,n} \\ \vdots & \dots & \vdots \\ A_{1,m} & \dots & A_{m,n} \end{bmatrix}$$

- Tensors: array of numbers arranged on a regular grid with a variable number of axes.
- Transpose: mirror image of a matrix/vector. $(A^\top)_{i,j} = A_{j,i}$.
- It is possible to add $m \times n$ matrices together. $\mathbf{C} = \mathbf{A} + \mathbf{B}$, where $C_{i,j} = A_{i,j} + B_{i,j}$.
- To add a scalar to a matrix or multiply a matrix by a scalar, perform that operation on each element of the matrix.

- Matrix Multiplication: $\mathbf{C} = \mathbf{AB}$ is defined as:

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

  where $\mathbf{A}$ is of shape $m \times n$, $\mathbf{B}$ is of shape $n \times p$ and $\mathbf{C}$ is of shape $m \times p$.
  Matrix product has some useful properties not covered in detail in the book.
- System of linear equations: $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{x}$ is a vector of unknown variables to be solved.
- Identity Matrix ($\boldsymbol{I}_n$): $n \times n$ matrix whose entries along the main diagonal are 1, while all the other entries are zero.
- Matrix Inverse ($\mathbf{A}^{-1}$): $\mathbf{A}^{-1}\mathbf{A} = \boldsymbol{I}_n$. For a matrix to have an inverse, it must be $n \times n$ and all its columns be linearly independent, it means that no column is a linear combination of another column.
- Norm (L^p): function that measures the size of vectors.

$$\|\mathbf{x}\|_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}}$$

  where $p \in \mathbb{R}$, $p \geq 1$.
  - The squared L^2 norm is commonly used and can be calculated as $\mathbf{x}^\top \mathbf{x}$.
  - L^∞ norm (max norm): absolute value of the element with the largest magnitude in the vector.
- Frobenius Norm: used to measure the size of a matrix (analogous to the L^2 norm of a vector).

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i,j} A_{i,j}^2}$$

- Symmetric Matrix: $\mathbf{A} = \mathbf{A}^\top$.
- Unit Vector: $\|\mathbf{x}\|_2 = 1$ (unit norm).
- Vectors $\mathbf{x}$ and $\mathbf{y}$ are orthogonal if $\mathbf{x}^\top \mathbf{y} = 0$. If both this vectors have unit norm, then they are called orthonormal.
- Orthogonal Matrix: square matrix whose rows are mutually orthonormal and whose columns are mutually orthonormal.
- An eigenvector of a square matrix $\mathbf{A}$ is a nonzero vector $\mathbf{v}$ such that multiplication by $\mathbf{A}$ alters only the scale of $\mathbf{v}$: $\mathbf{Av} = \lambda\mathbf{v}$. The scalar λ is known as the eigenvalue corresponding to this eigenvector. The eigendecomposition of $\mathbf{A}$ is given by: $\mathbf{A} = \mathbf{V}\text{diag}(\lambda)\mathbf{V}^{-1}$, where $\mathbf{V}$ is the matrix whose columns are each eigenvector of $\mathbf{A}$ and $\text{diag}(\lambda)$ is the diagonal matrix of eigenvalues such that the eigenvalue at $\lambda_{i,i}$ is the one associated with the eigenvector (column) i of $\mathbf{V}$. Eigendecomposition is not defined for every matrix.
- Singular Value Decomposition (SVD): $\mathbf{A} = \mathbf{U}\mathbf{D}\boldsymbol{V}^\top$, where $\mathbf{A}$ is a $m \times n$ matrix. $\mathbf{U}$ ($m \times m$) is composed by the eigenvectors of $\boldsymbol{A}\boldsymbol{A}^\top$; $\mathbf{V}$ ($n \times n$) is composed by the eigenvectors of $\boldsymbol{A}^\top\boldsymbol{A}$; and $\mathbf{D}$ ($m \times n$) is a diagonal matrix composed by the eigenvalues of $\boldsymbol{A}\boldsymbol{A}^\top$ or $\boldsymbol{A}^\top\boldsymbol{A}$.
- Moore-Penrose pseudoinverse: The pseudoinverse of $\mathbf{A}$ is defined as a matrix $\boldsymbol{A}^+ = \boldsymbol{V}\boldsymbol{D}^+\boldsymbol{U}^\top$, where $\mathbf{U}$, $\mathbf{D}$ and $\mathbf{V}$ are the SVD of $\mathbf{A}$, and the pseudoinverse $\boldsymbol{D}^+$ of a diagonal matrix $\mathbf{D}$ is obtained by taking the reciprocal of its nonzero elements then making the transpose of the resulting elements.
- The Trace operator gives the sum of the diagonal entries of a matrix: $\text{Tr}(\mathbf{A}) = \sum_i \boldsymbol{A}_{i,i}$. It has some useful identities to manipulate expressions.
- Determinant ($\det(\mathbf{A})$): function that maps matrices to real scalars. It is equal to the product of all the eigenvalues of a matrix.
- Principal Components Analysis (PCA): ML algorithm that can be derived using only knowledge of basic Linear Algebra. The reader can find a description of the implementation at section 2.12.

3 Probability and Information Theory

- Possible sources of uncertainty:
  1. Inherent stochasticity in the system being modeled.
  2. Incomplete observability.
  3. Incomplete modeling.

- Probability can be seen as the extension of logic to deal with uncertainty.
- Random Variable: variable x (discrete or continuous) that can take on different values randomly.
- Probability Distribution (PD): description of how likely a random variable or a set of random variables is to take on each of its possible states.
- Probability Mass Function (PMF): PD over discrete variables. A PMF, P, maps from a state of a random variable to the probability of that random variable taking on that state. A PMF acting on many variables at the same time is known as a joint probability distribution: $P(\mathrm{x} = x, \mathrm{y} = y)$.
- Probability Density Function (PDF): describes PD of continuous random variables. A PDF, $p(x)$, gives the probability of landing inside an infinitesimal region, not at a specific state.
- Marginal PD: it is the PD of a subset of a set of variables of which we know the PD.
- Conditional Probability: the probability of some event $\mathrm{x} = x$, given that some other event $\mathrm{y} = y$ has happened.

$$P(\mathtt{y} = y \mid \mathtt{x} = x) = \frac{P(\mathtt{y} = y\ ,\ \mathtt{x} = x)}{P(\mathtt{x} = x)},\ \textit{where}\ P(\mathtt{x} = x) > 0.$$

- Chain Rule of Conditional Probabilities: any joint PD over many random variables may be decomposed into conditional distributions over only one variable:

$$P(\mathtt{x}^{(1)}, \ldots, \mathtt{x}^{(n)}) = P(\mathtt{x}^{(1)}) \prod_{i=2}^{n} P(\mathtt{x}^{(i)} \mid \mathtt{x}^{(1)}, \ldots, \mathtt{x}^{(i-1)}).$$

- Two random variables x and y are independent ($\mathrm{x} \perp \mathrm{y}$) if:

$$\forall x \in \mathtt{x}, y \in \mathtt{y}, p(\mathtt{x} = x, \mathtt{y} = y) = p(\mathtt{x} = x)p(\mathtt{y} = y)$$

- Two random variables x and y are conditionally independent given a random variable z ($\mathrm{x} \perp \mathrm{y} | z$) if:

$$\forall x \in \mathtt{x}, y \in \mathtt{y}, z \in \mathtt{z}, p(\mathtt{x} = x, \mathtt{y} = y \mid \mathtt{z} = z) = p(\mathtt{x} = x \mid \mathtt{z} = z)p(\mathtt{y} = y \mid \mathtt{z} = z)$$

- The expectation ($\mathbb{E}_{x \sim P}[f(x)]$), or expected value, of some function $f(x)$ w.r.t. a PD $P(\mathrm{x})$ is the average, or mean value, that f takes on when x is drawn from P.

$$\mathbb{E}_{x \sim P}[f(x)] = \sum_{x} P(x)f(x),\ \textit{for discrete variables.}$$

$$\mathbb{E}_{x \sim p}[f(x)] = \int p(x)f(x)dx,\ \textit{for continuous variables.}$$

- The variance gives a measure of how much the values of a function of a random variable x vary as we sample different values of x from its PD. The square root of the variance is the standard deviation.

$$\mathtt{Var}(f(X)) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2]$$

- The covariance gives some sense of how much two values are linearly related to each other, as well as the scale of these variables. Two variables have zero covariance if they are not linearly dependent.

$$\mathtt{Cov}(f(X), g(y)) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])(g(y) - \mathbb{E}[g(y)])]$$

- The correlation normalizes the contribution of each variable in order to measure only how much the variables are related.
- The covariance matrix of a random vector $\mathbf{x} \in \mathbb{R}^n$ is an $n \times n$ matrix, such that $\mathrm{Cov}(\mathrm{x})_{i,j} = \mathrm{Cov}(\mathrm{x}_i, \mathrm{x}_j)$. The diagonal elements of the covariance give the variance: $\mathrm{Cov}(\mathrm{x}_i, \mathrm{x}_i) = \mathrm{Var}(\mathrm{x}_i)$.
- The Bernoulli Distribution is a distribution over a single binary random variable. It is controlled by a single parameter $\phi \in [0, 1]$, which gives the probability of the random variable being equal to 1. $P(\mathtt{x} = x) = \phi^x(1 - \phi)^{1-x}$, $\mathbb{E}_{\mathtt{x}}[\mathtt{x}] = \phi$, $\mathtt{Var}_{\mathtt{x}}(\mathtt{x}) = \phi(1 - \phi)$.
- The multinoulli, or categorical, distribution is a distribution over a single discrete variable with k different states, where k is finite. It is parametrized by a vector $\boldsymbol{p} \in [0, 1]^{k-1}$, where p_i gives the probability of the i-th state. The final, k-th state's probability is given by 1 - $1^\top \boldsymbol{p}$.
- Gaussian, normal, distribution: distribution over the real numbers. The parameters $\mu \in \mathbb{R}$ and $\sigma \in (0, \infty)$ control the normal distribution.

$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right).$$

To evaluate the PDF efficiently, parametrize the distribution with $\beta \in (0, \infty)$, to control the precision, or inverse variance:

$$\mathcal{N}(x; \mu, \beta^{-1}) = \sqrt{\frac{\beta}{2\pi}} \exp\left(-\frac{1}{2}\beta(x-\mu)^2\right).$$

Multivariate normal distribution: the normal distribution generalized to $\mathbb{R}^n$. It may be parametrized with a positive definite symmetric matrix $\boldsymbol{\Sigma}$:

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sqrt{\frac{1}{(2\pi)^n \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right).$$

where $\boldsymbol{\mu}$ is a vector with the mean of the distribution, and $\boldsymbol{\Sigma}$ gives the covariance matrix.
Use a precision matrix $\boldsymbol{\beta}$, to evaluate several times for different values of the parameters:

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\beta}^{-1}) = \sqrt{\frac{\det(\boldsymbol{\beta})}{(2\pi)^n}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\beta}(\boldsymbol{x}-\boldsymbol{\mu})\right).$$

- Exponential distribution: PD with a sharp point at $x = 0$: $p(x; \lambda) = \lambda \mathbf{1}_{x \geq 0} \exp(-\lambda x)$. It uses $\mathbf{1}_{x \geq 0}$ to assign probability zero to all negative values of x.
- Laplace distribution: PD that places a sharp peak of probability mass at an arbitrary point μ: $\text{Laplace}(x; \mu, \gamma) = \frac{1}{2\gamma} \exp\left(-\frac{|x-\mu|}{\gamma}\right)$.
- Dirac delta function: makes possible to specify that all the mass in a PD clusters around a point.
- Mixture distribution: PD made up of several component distributions.
- Latent variable: random variable that is not possible to observe directly.
- Logistic sigmoid:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

- Softplus function: (smoothed version of $x^+ = \max(0, x)$)

$$\zeta(x) = \log(1 + \exp x)$$

- Bayes' Rule:

$$P(\mathtt{x}|\mathtt{y}) = \frac{P(\mathtt{x})P(\mathtt{y}|\mathtt{x})}{P(\mathtt{y})}$$

  It is possible to compute $P(\mathrm{y})$ as: $P(\mathtt{y}) = \sum_x P(\mathtt{y}|\mathtt{x})P(\mathtt{x})$
- Information theory: quantifies how much information is present in a signal.
  - Basic intuition: learning that an unlikely event has occurred is more informative than learning that a likely event has occurred. Likely events have low (or zero) information content. Less likely events have higher information content. Independents events have additive information content.
  - Self-information of an event $\mathrm{x} = x$: $I(x) = -\log P(x)$ $[nats]$. One nat is the amount of information gained by observing an event of probability $\frac{1}{e}$. (The book uses log as the natural logarithm).
  - Shannon entropy: quantifies the amount of uncertainty in an entire PD. $H(\mathtt{x}) = \mathbb{E}_{x \sim P}[I(x)]$.
  - To measure how different two distributions over the same random variable are, use the Kullback-Leibler (KL) divergence: $D_{KL}(P||Q) = \mathbb{E}_{x \sim P}[\log P(x) - \log Q(x)]$.
  - Cross-entropy: $H(P, Q) = -\mathbb{E}_{x \sim P} \log Q(x)$.
- A Structured probabilistic model, or graphical model, represents the factorization of a PD with a graph (represents the PD as factors). Each node in the graph is a random variable, and an edge connecting two random variables means that the PD is able to represent direct interaction between those two random variables.
  - Directed models represent factorization into conditional PD.

$$p(\mathtt{x}) = \prod_i p(\mathtt{x}_i | PaG(x_i)),$$

    where $\mathrm{PaG}(\mathrm{x}_i)$ represents the parents of x_i.
  - Undirected models represent factorizations into a set of functions. Any set of nodes connected to each other is called a clique. Each clique $C(i)$ is associated with a factor $\phi^{(i)}(C^{(i)})$.

$$p(\mathtt{x}) = \frac{1}{Z} \prod_i \phi^{(i)}(C^{(i)}),$$

    where Z is a normalizing constant defined to be the sum or integral over all states of the product of the ϕ functions.

4 Numerical Computation

- Algorithms that solve mathematical problems by methods that update estimates of the solution via an iterative process.
- Underflow: numbers near zero that are rounded to zero.
- Overflow: numbers with large magnitude are approximated as ∞ or -∞.
- Conditioning: how rapidly a function changes w.r.t. small changes in its inputs.
- Optimization: minimize or maximize an objective function $f(\boldsymbol{x})$ by altering $\boldsymbol{x}$. Sometimes, the value that minimizes or maximizes a function, is denoted with a superscript $*$: $\boldsymbol{x}^* = arg\,min\ f(\boldsymbol{x})$.
- Gradient Descent: reduce $f(x)$ by moving x in small steps with the opposite sign of the derivative. $f(x - \epsilon \texttt{sign}(f'(x))$.
- Critical point: point with zero slope.
- In the context of Deep Learning it is tried to find a value of f that is very low but not necessarily minimal in any formal sense.
- The partial derivative $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$ measures how f changes as only the variable x_i increases at point $\boldsymbol{x}$. The gradient generalizes the notion of derivative to the case where the derivative is w.r.t. a vector: the gradient of f is the vector containing all the partial derivatives, denoted $\nabla_x f(\boldsymbol{x})$.
- The directional derivative in direction $\boldsymbol{u}$ (a unit vector) is the slope of the function f in direction u. The minimization occurs when the gradient points directly uphill, and the negative gradient points directly downhill.
- It is possible to decrease f by moving in the direction of the negative gradient. This is the method of steepest descent, or gradient descent: $\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_x f(\boldsymbol{x})$, where ϵ is the learning rate, a positive scalar determining the size of the step. This method converges when every element of the gradient is zero (or very close to zero). To jump directly to the critical point, solve: $\nabla_x f(\boldsymbol{x}) = 0$.
  Hill climbing is the generalization of this method for discrete spaces.
- If there is the function $\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}^n$, then the Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{n \times m}$ of $\boldsymbol{f}$ is defined such that $J_{i,j} = \frac{\partial}{\partial x_j} f(\boldsymbol{x})_i$.
- The Hessian matrix $\boldsymbol{H}(f)(\boldsymbol{x})$ is defined such that:

$$\boldsymbol{H}(f)(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_i x_j} f(\boldsymbol{x}).$$

  Equivalently, the Hessian is the Jacobian of the gradient. Also, it is symmetric (if the function is continuous at such points). The second derivative in a specific direction represented by a unit vector $\boldsymbol{d}$ is given by $\boldsymbol{d}^\top \boldsymbol{H} \boldsymbol{d}$; when $\boldsymbol{d}$ is an eigenvector of $\boldsymbol{H}$, the second derivative in that direction is the corresponding eigenvalue.
  To the extent that the function to be minimized can be approximated well by a quadratic function, the eigenvalues of the Hessian thus determine the scale of the learning rate.
  Using the eigendecomposition of the Hessian matrix, it is possible to generalize the second derivative test to multiple dimensions.
- Newton's Method: uses a second-order Taylor series expansion to approximate $f(\boldsymbol{x})$ near some point $\boldsymbol{x}^{(0)}$.
- In Deep Learning, sometimes the functions used are restricted to those that are either Lipschitz continuous or have Lipschitz continuous derivatives. A Lipschitz continuous function is a function f whose rate of change is bounded by a Lipschitz constant $\mathcal{L} : \forall \boldsymbol{x}, \forall \boldsymbol{y}, |f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq \mathcal{L} ||\boldsymbol{x} - \boldsymbol{y}||_2$.
- Constrained Optimization: used to find the maximal or minimal value of $f(\boldsymbol{x})$ for values of $\boldsymbol{x}$ in some set $\mathbb{S}$.

5 Machine Learning Basics

- ML Algorithm: algorithm that is able to learn from data.
- Learning is the means of attaining the ability to perform a task.
- Some common ML tasks: classification, classification with missing inputs, regression, transcription, machine translation, structured output, anomaly detection, synthesis and sampling, imputation of missing values, denoising, density estimation or probability mass function estimation.
- Performance measure: quantitative measure (specific to the task) of the abilities of a ML algorithm.
- Unsupervised learning algorithms experience a dataset containing many features, then learn useful properties of the structure of this dataset.

- Supervised learning algorithms experience a dataset containing features, but each example is also associated with a label or target.
- Reinforcement learning algorithms interact with an environment, so there is a feedback loop between the learning system and its experiences.
- Design matrix: matrix containing a different example in each row; each column corresponds to a different feature.
  The vector $\boldsymbol{y}$, provides the label y_i to example i.
- Parameters (weights): values that control the behavior of the system.
- Linear Regression example:
  - Task: to predict y from $\boldsymbol{x}$ by outputting $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$.
  - Performance measure: mean square error of the model on the test set.

$$\texttt{MSE}_{\texttt{test}} = \frac{1}{m} \sum_i (\hat{\boldsymbol{y}}^{(\texttt{test})} - \boldsymbol{y}^{(\texttt{test})})_i^2$$

  - The objective is to design an algorithm that will improve the weights $\boldsymbol{w}$ in a way that reduces $\texttt{MSE}_{\texttt{test}}$ when the algorithm is allowed to gain experience by observing a training set $(\boldsymbol{X}^{(\texttt{train})}, \boldsymbol{y}^{(\texttt{train})})$. The way of doing this, it to minimize the MSE on the training set (solve for where its gradient is zero).
  - The idea is to minimize the training error, but measure the performance based on the test error.
  - The intercept term b of an affine function is often called the bias parameter.
- Generalization: the ability to perform well on previously unobserved inputs.
- Generalization/Test error: the expected value of the error on a new input.
- ML assumes that the datasets are independent and identically distributed.
- The factors determining how well a ML algorithm performs are its ability to make the training error small and make the gap between training and test error small.
- Underfitting: the model is not able to obtain a sufficiently low error value on the training set.
- Overfitting: the gap between the training error and test error is too large.
- Capacity: the model's ability to fit a wide variety of functions. One way to change it, is to change the number of input features it has and simultaneously add new parameters associated with those features.
- Hypothesis space: the set of functions that the learning algorithm is allowed to select as being the solution.
- Determine the capacity of a deep learning algorithm is difficult because the effective capacity is limited by the capabilities of the optimization algorithm.
- The no free lunch theorem for ML states that, averaged over all possible data-generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.
- The behavior of an algorithm is also affected by the specific identity of the functions in its hypothesis space.
- It is possible to regularize a model that learns a function $f(\boldsymbol{x}; \theta)$ by adding a penalty called a regularizer to the cost function. Regularization is any modification made to a learning algorithm that is intended to reduce its generalization error but not its training error.
- Expressing preferences for one function over another is a more general way of controlling a model's capacity than including or excluding members from the hypothesis space.
- Hyperparameters: settings used to control the algorithm's behavior.
- Point estimation: the attempt to provide the single 'best' prediction of some quantity of interest.
- Function estimation: approximatting f with a model or estimate $\hat{f}$.
- The bias of an estimator is defined as $\text{bias}(\hat{\boldsymbol{\theta}_m}) = \mathbb{E}(\hat{\boldsymbol{\theta}_m}) - \boldsymbol{\theta}$. It measures the expected deviation from the true value of the function or parameter.
- The variance of an estimator is simply the variance: $\text{Var}(\hat{\theta})$. It measures the deviation from the expected estimator value that any particular sampling of the data is likely to cause.
  The standard error, denoted $\text{SE}(\hat{\theta})$, is the square root of the variance.
- The generalization error is often estimated computing the sample mean of the error on the test set.
- Desirable estimators are those with small MSE and these are the estimators that manage to keep their bias and variance somewhat in check.

- Consistency: as the number of data points m in the dataset increases, the point estimates converge to the true value of the corresponding parameters. It ensures that the bias induced by the estimator diminishes as the number of data examples grows.
- Maximum Likelihood Estimation: minimizes the dissimilarity between the empirical distribution $\hat{p}_{data}$, defined by the training set and the model distribution, with the degree of dissimilarity between the two measured by the KL divergence. The KL divergence is given by

$$D_{KL}(\hat{p}_{data}||p_{model}) = \mathbb{E}_{x\sim\hat{p}_{data}}[\log \hat{p}_{data}(\boldsymbol{x}) - \log p_{model}(\boldsymbol{x})]$$

  When training the model, it is only needed to minimize $-\mathbb{E}_{x\sim\hat{p}_{data}}[\log \hat{p}_{data}(\boldsymbol{x})]$.
- Bayesian statistics: considers all possible values of $\boldsymbol{\theta}$ when making a prediction. It uses probability to reflect degrees of certainty in states of knowledge.
  - The knowledge of $\boldsymbol{\theta}$ is represented using the prior probability distribution, $p(\boldsymbol{\theta})$.
  - To recover the effect of data on what is believed about $\boldsymbol{\theta}$:

$$p(\boldsymbol{\theta}|x^{(1)}, \cdots, x^{(m)}) = \frac{p(x^{(1)}, \cdots, x^{(m)}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(x^{(1)}, \cdots, x^{(m)})}$$

- Support Vector Machine (SVM): supervised learning algorithm that outputs a class identity.
- k-nearest neighbors: family of supervised learning techniques used for classification or regression.
- Principal Component Analysis learns a representation that has lower dimensionality than the original input. It also learns a representation whose elements have no linear correlation with each other. This is a first step toward the criterion of learning representations whose elements are statistically independent. To achieve full independence, a representation learning algorithm must also remove the nonlinear relationships between variables.
- Stochastic Gradient Descent (SGD): nearly all of deep learning is powered by this algorithm.
  - The insight of SGD is that the gradient is an expectation. It can be approximately estimated using a small set of samples.
  - Each step of the algorithm samples a minibatch of examples $\mathbb{B} = \{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m')}\}$ drawn uniformly from the training set.
  - The estimate of the gradient is formed as:

$$\boldsymbol{g} = \frac{1}{m'}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

    using examples from the minibatch $\mathbb{B}$.
  - The algorithm then follows the estimated gradient downhill:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g}$$

    where ϵ is the learning rate.
- The recipe, of most, deep learning algorithm can be described as: combine a specification of a dataset, a cost function, an optimization procedure and a model.
- The Curse of Dimensionality: many ML algorithms become exceedingly difficult when the number of dimensions in the data is high. The number of possible distinct configuration of a set of variables increases exponentially as the number of variables increases.
- Smoothness prior or local constancy prior: this prior states that the learned function should not change very much within a small region.
- The core idea in deep learning is that is assumes that the data was generated by the composition of factors, or features, potentially at multiple levels in a hierarchy.
- Manifold: connected region. Mathematically, it is a set of points associated with a neighborhood around each point. From any given point, it locally appears to be a Euclidean space. - In ML it tends to be used to designate a connected set of points that can be approximated well by considering only a small number of degrees of freedom, or dimensions, embedded in a higher-dimensional space.
- Manifold learning algorithms assume that most of $\mathbb{R}^n$ consists of invalid inputs, and that interesting inputs occur only along a collection of manifolds containing a small subset of points, with interesting variations in the output of the learned function occurring only along directions that lie on the manifold, or with interesting variations happening only when moving from one manifold to another.

6 Deep Feedforward Networks

- They define a mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ and learn the value of the parameters $\boldsymbol{\theta}$ that result in the best function approximation.
- There are no feedback connections in which outputs of the model are fed back into itself.
- The model is associated with a directed acyclic graph describing how the functions are composed together.
- Output layer: final layer (function) of the network.
- The training data provides noisy, approximate examples of $f^*(\boldsymbol{x})$ evaluated at different training points. Each example $\boldsymbol{x}$ is accompanied by a label $y \approx f^*(\boldsymbol{x})$, that specify what the output layer must do. The learning algorithm must decide how to use the hidden layers to best implement an approximation of f^* (the training data does not show a desired output for each of these layers).
- This networks require to initialize all weights to small random values.
- To apply gradient-based learning, a cost function and output representation must be chosen.
- The total cost of the network will often combine one of the primary cost functions with a regularization term.
- Most modern NN are trained using maximum likelihood.This means that the cost function is simply the negative log-likelihood:
$$J(\boldsymbol{\theta}) = -\mathbb{E}_{x,y\sim\hat{y}_{data}} \log p_{model}(\boldsymbol{y}|\boldsymbol{x})$$
The expansion of the above equation typically yields some terms that do not depend on the model parameters and may be discarded.
- The equivalence between maximum likelihood estimation and minimization of MSE holds regardless of the $f(\boldsymbol{x}; \theta)$ used to predict the mean of the Gaussian.
- The gradient of the cost function must be large and predictable enough to serve as a good guide for the learning algorithm.
- It is possible to view the cost function as a functional. A functional maps from functions to real numbers. It can have its minimum at some specified function.
- The choice of cost function is tightly coupled with the choice of output unit.
- Linear units for Gaussian distributions. The layer produces the mean of a conditional Gaussian distribution: $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$.
- Sigmoid units for Bernoulli distributions. Useful for predicting the value of a binary variable y.
- Logit: variable (usually denoted by z) defining a probability distribution based on exponentiation and normalization over binary variables.
- Softmax units for Multinoulli distributions. Useful to represent a PD over a discrete variable with n possible values.
- The objective functions that do not use a log to undo the exp of the softmax fail to learn when the argument to the exp becomes very negative.
- In general, NN represent a function $f(\boldsymbol{x}|\boldsymbol{y})$. The outputs of $f(\boldsymbol{x}|\boldsymbol{y}) = \boldsymbol{\omega}$ provide the parameters for a distribution over y. The loss function can then be interpreted as $-\log p(\mathrm{y}, \boldsymbol{\omega}(\boldsymbol{x}))$.
- Gaussian mixture: lets predict real values from a conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$ that can have several different peaks in $\boldsymbol{y}$ space for the same value of $\boldsymbol{x}$.
- The function used in the context of NN usually have defined left derivatives and defined right derivatives.
- Most hidden units can be described as accepting a vector of inputs $\boldsymbol{x}$, computing an affine transformation $\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}$, and then applying an element-wise nonlinear function $g(\boldsymbol{z})$.
- ReLU are an excellent default choice of hidden unit. They use the activation function $g(\boldsymbol{z}) = \max\{0, z\}$. They cannot learn via gradient-based methods on examples for which their activation is zero.
  Three generalizations of ReLU are based on using a nonzero slope α_i when $z_i < 0 : h_i = g(\boldsymbol{z}, \boldsymbol{\alpha})_i = \max(0, z_i) + \alpha_i \min(0, z_i)$.
  - Absolute value error rectification fixes $\alpha_i = -1$ to obtain $g(z) = |z|$.
  - Leaky ReLU fixes α_i to a small value like 0.01.
  - Parametric ReLU (PReLU) treats α_i as a learnable parameter.

- Maxout units divide $\boldsymbol{z}$ into groups of k values. Each maxout unit then outputs the maximum element of one of these groups: $g(\boldsymbol{z})_i = \max_{j \in \mathbb{G}^{(i)}} z_j$, where $\mathbb{G}^{(i)}$ is the set of indices into the inputs for group i, $\{(i-1)k+1, \cdots, ik\}$. They can resist a phenomenon called catastrophic forgetting, in which NN forget how to perform tasks that they were trained on in the past.
- Logistic Sigmoid activation function: $g(z) = \sigma(z)$. Sigmoidal units saturate across most of their domain. Their use as output units is compatible with the use of gradient-based learning when an appropriate cost function can undo the saturation of the sigmoid in the output layer.
- Hyperbolic Tangent activation function: $g(z) = \tanh(z) = 2\sigma(2z) - 1$. It resembles the identity function more closely, in the sense that $\tanh(0) = 0$ while $\sigma(0) = \frac{1}{2}$.
- It is acceptable for **some** layers of the NN to be purely linear.
- Radial basis function (RBF) unit: $h_i = \exp\left(-\frac{1}{\sigma_i^2}||\boldsymbol{W}_{:,i} - \boldsymbol{x})||^2\right)$. It becomes more active as $\boldsymbol{x}$ approaches a template $\boldsymbol{W}_{:,i}$. Because it saturates to 0 for most $\boldsymbol{x}$, it can be difficult to optimize.
- Softplus units: $g(a) = \zeta(a) = \log(1 + \mathrm{e}^a)$. Its use is generally discouraged.
- Hard tanh unit: $g(a) = \max(-1, \min(1, a))$.
- Architecture of the network: how many units it should have and how these units should be connected to each other.
- Most NN are organized into groups of units called layers. Most NN architectures arrange these layers in a chain structure, with each layer being a function of the layer that preceded it. In these architectures the main considerations are choosing the depth of the network and the width of each layer. Deeper networks generalize better (most of the time).
- Universal Approximation theorem: a feedforward NN with a linear output layer and at least one hidden layer with any "squashing" activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired nonzero amount of error, provided that the network is given enough hidden units. The derivatives of the feedforward network can also approximate the derivatives of the function arbitrarily well.
- Many architectures build a main layers chain but then add extra architectural features to it, such as skip connections going from layer i to layer $i+2$ or higher. These skip connections make it easier for the gradient to flow from output layers to layers nearer the input.
- During training, forward propagation can continue onward until it produces a scalar cost $J(\boldsymbol{\theta})$. The back-propagation algorithm allows the information from the cost to then flow backward through the network in order to compute the gradient.
- Back-propagation is the method for computing the gradient, while another algorithm, such as stochastic gradient descent, is used to perform learning using this gradient. In learning algorithm, the gradient that is most often required is the gradient of the cost function w.r.t. the parameters: $\boldsymbol{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$.
- The chain rule of calculus is used to compute the derivatives of functions formed by composing other functions whose derivatives are known. Backprop computes the chain rule, with a specific order of operations that is highly efficient.
  Suppose that $\boldsymbol{x} \in \mathbb{R}^m, \boldsymbol{y} \in \mathbb{R}^n, g$ maps from $\mathbb{R}^m$ to $\mathbb{R}^n$, and f maps from $\mathbb{R}^n$ to $\mathbb{R}$. If $\boldsymbol{y} = g(\boldsymbol{x})$ and $z = f(\boldsymbol{y})$, then
  $$\frac{\partial z}{\partial x_i} = \sum_i \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$
  In vector notation, this may be equivalently written as
  $$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^\top \nabla_{\boldsymbol{y}} z,$$
  where $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ is the $n \times m$ Jacobian matrix of g.
  So, the gradient of a variable $\boldsymbol{x}$ can be obtained by multiplying a Jacobian matrix $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ by a gradient $\nabla_{\boldsymbol{y}} z$. The backprop algorithm consists of performing such a Jacobian-gradient product for each operation in the graph.
- Algorithms 6.1, 6.2, 6.3, 6.4, 6.5 in the book further enhance understanding of backprop.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.