An Overview on Deep Neural Networks

By Hamed Araab

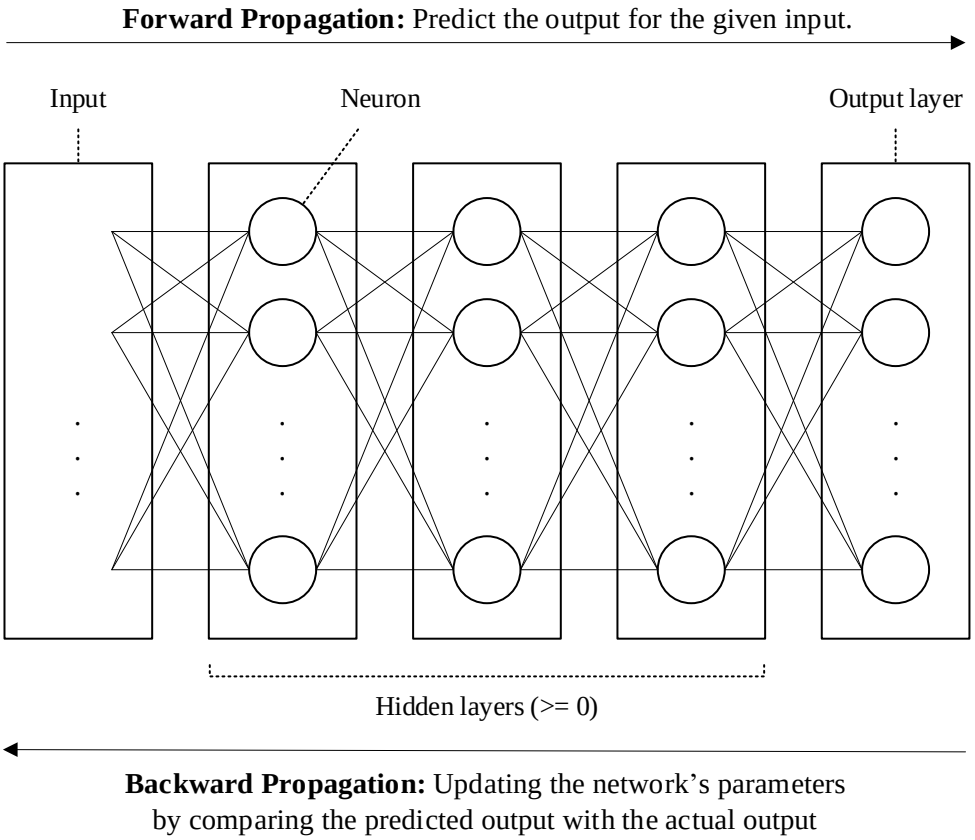
Supervisor: Dr. Marzieh Zarinbal

Winter 2024

Table of Contents

1 Introduction

Deep Neural Networks (DNNs) are a type of artificial neural networks that stand out for having multiple layers. This multi-layer structure allows them to model complex relationships and create hierarchical representations in datasets. In each layer of a DNN, there are nodes or neurons connected to each other. These connections have weights, and there's also a bias term. The bias term makes the model more flexible, helping it capture subtle patterns in the data.



2 Common Tasks in Deep Learning

2.1 Classification

2.1.1 Binary

Binary classification involves the process of categorizing items into one of two distinct classes or categories. The primary objective is to assign each item to either Class 1 or Class 0. To illustrate, consider the task of classifying items as either a "cat" (Class 1) or "not cat" (Class 0):

 → [0.98]  → [0.03]  → [0.1]

2.1.2 Multi-Class

Multi-class classification involves the assignment of items into more than two classes or categories. The primary objective is to categorize items across multiple classes rather than just two. To illustrate, consider a scenario where we aim to classify items into distinct categories such as "cat," "dog," and "neither":

2.1.3 Multi-Label

Multi-label classification is a classification task where each input is capable of belonging to multiple classes simultaneously. In this scenario, the goal is to identify and assign multiple labels to each input. For instance, we may seek to find out whether each item contains a "cat," "dog," or "elephant":

2.2 Regression

Regression is a type of machine learning task where the goal is to predict a continuous numerical value rather than a class label. It involves learning a mapping function from input features to a continuous output variable. For example, we can predict house prices based on features such as square footage, number of bedrooms, and location.

3 Comments on Some Notations

You can see the explanations for some notations that you are going to encounter later on:

- $\mathbb{Z}_{>0}$ denotes positive integers.
- $\mathbb{R}_{\geq 0}$ denotes non-negative real numbers.
- Superscript $[l]$ denotes the l^{th} layer.
- $\odot$ denotes element-wise product of two matrices.
- $\oslash$ denotes element-wise division of two matrices.
- : denotes tensor double contraction.

4 Components of a Simple DNN

4.1 Data

4.1.1 Sizes

- $n \in \mathbb{Z}_{>0}$ The **number** of the items in the dataset
- $m_x \in \mathbb{Z}_{>0}$ The **size** of the **input** for each item
- $m_y \in \mathbb{Z}_{>0}$ The **size** of the **output** for each item

4.1.2 Objects

- $X \in \mathbb{R}^{n \times m_x}$ The input **matrix**
- $Y \in \mathbb{R}^{n \times m_y}$ The actual output **matrix**

4.2 Trainable Parameters

- $W^{[l]} \in \mathbb{R}^{m_h^{[l-1]} \times m_h^{[l]}}$ The weight **matrix** of the l^{th} layer
- $b^{[l]} \in \mathbb{R}^{1 \times m_h^{[l]}}$ The bias **vector** of the l^{th} layer

4.3 Hyperparameters

- $L \in \mathbb{Z}_{>0}$ The **number** of the layers in the network (hidden + output)
- $m_h^{[l]} \in \mathbb{Z}_{>0}$ The **number** of the units (neurons) in the l^{th} layer
  It is also possible to denote $m_x = m_h^{[0]}$ and $m_y = m_h^{[L]}$.
- $\sigma^{[l]}\left(Z^{[l]} \in \mathbb{R}^{n \times m_h^{[l]}}\right) \to \mathbb{R}^{n \times m_h^{[l]}}$ The activation **function** of the l^{th} layer
- $J\left(Y \in \mathbb{R}^{n \times m_y}, \hat{Y} \in \mathbb{R}^{n \times m_y}\right) \to \mathbb{R}_{\geq 0}$ The **loss (error) function** of the model for all training examples
- $\alpha \in (0,1]$ The **learning rate** of the model

4.3.1 Examples

4.3.1.1 Activation Functions

- Rectified Linear Unit (ReLU) $\sigma^{[l]}\left(Z^{[l]}\right)_{i,j} = \max\left\{0, Z_{i,j}^{[l]}\right\}$
  Usually applied to **the output layer** in **regression**
  One of the common functions in **the hidden layers of modern models**

- Sigmoid $\sigma^{[l]}\left(Z^{[l]}\right)_{i,j} = \frac{1}{1+\exp\left(-Z_{i,j}^{[l]}\right)}$

  Usually applied to **the output layer** in **binary or multi-label classification**

- Softmax $\sigma^{[l]}\left(Z^{[l]}\right)_{i,j} = \frac{\exp\left(Z_{i,j}^{[l]}\right)}{\sum_{k=1}^{m_h^{[l]}} \exp\left(Z_{i,k}^{[l]}\right)}$

  Usually applied to **the output layer** in **multi-class classification**

4.3.1.2 Loss Functions

- Squared of Squared Errors (SSE) $J\left(Y, \hat{Y}\right) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m_y}\left(\hat{Y}_{i,j} - Y_{i,j}\right)^2$

  Commonly used in **regression**

  The sum is divided by two for easier derivative calculation.

- Binary Cross Entropy (BCE) $J\left(Y, \hat{Y}\right) = -\sum_{i=1}^{n}\sum_{j=1}^{m_y}\left(Y_{i,j}\log\left(\hat{Y}_{i,j}\right) + \left(1 - Y_{i,j}\right)\log\left(1 - \hat{Y}_{i,j}\right)\right)$

  Commonly used in **binary or multi-label classification**

- Categorical Cross Entropy (CCE) $J\left(Y, \hat{Y}\right) = -\sum_{i=1}^{n}\sum_{j=1}^{m_y}\left(Y_{i,j}\log\left(\hat{Y}_{i,j}\right)\right)$

  Commonly used in **multi-class classification**

4.3.1.3 Summary

Task	Output Layer Activation Function	Loss Function
Regression	ReLU	SSE
Binary of Multi-Label Classification	Sigmoid	BCE
Multi-Class Classification	Softmax	CCE

5 Forward Propagation

Forward propagation is a fundamental step in how deep neural networks work. It includes sending input data through the network's layers to produce an output. In each layer, the input is multiplied by weights, and the bias term is added. Activation functions bring in non-linearities to capture complex patterns effectively. The output from one layer becomes the input for the next layer, continuing through the network until the final layer makes the model's prediction.

5.1 Mathematical Representation

5.1.1 Objects

- $Z^{[l]} \in \mathbb{R}^{n \times m_h^{[l]}}$ The pre-activation **matrix** of the l^{th} layer
- $A^{[l]} \in \mathbb{R}^{n \times m_h^{[l]}}$ The activation **matrix** of the l^{th} layer
- $\hat{Y} \in \mathbb{R}^{n \times m_y}$ The predicted output **matrix**

5.1.2 Equations

- $A^{[0]} = X$
- $Z^{[l]} = A^{[l-1]}W^{[l]} + b^{[l]}$ $\forall l = 1,2, \dots, L$

  b is added to each row of $A^{[l-1]}W^{[l]}$. This is called broadcasting.
- $A^{[l]} = \sigma^{[l]}\left(Z^{[l]}\right)$ $\forall l = 1,2, \dots, L$
- $\hat{Y} = A^{[L]}$

6 Backward Propagation (Batch Gradient Descent)

Backward propagation, or backpropagation, is a crucial step in training deep neural networks. It helps the network learn from mistakes. In backpropagation, we calculate the error in the model's output by comparing what it predicted with the actual values we wanted. We then move backward through the network, one layer at a time. For each layer, we adjust the weights and biases in the opposite direction of the error gradient. This is done through an iterative process, guided by optimization algorithms like gradient descent. It helps the neural network improve its parameters, reducing prediction errors. Backward propagation is essential for the network to get better at making accurate predictions on new, unseen data. This makes deep learning models successful in various applications.

6.1 Mathematical Representation

6.1.1 Objects

- $C \in \mathbb{R}_{\geq 0}$ The **cost** of the predictions
- $\frac{\partial C}{\partial \hat{Y}} \in \mathbb{R}^{n \times m_y}$
- $\frac{\partial A^{[l]}}{\partial Z^{[l]}} \in \mathbb{R}^{n \times m_h^{[l]} \times n \times m_h^{[l]}}$
- $\frac{\partial Z^{[l]}}{\partial W^{[l]}} \in \mathbb{R}^{m_h^{[l-1]} \times n}$
- $\frac{\partial Z^{[l]}}{\partial b^{[l]}} \in \mathbb{R}^{1 \times n}$

- $\frac{\partial Z^{[l]}}{\partial A^{[l-1]}} \in \mathbb{R}^{m_h^{[l]} \times m_h^{[l-1]}}$
- $\frac{\partial C}{\partial A^{[l]}} \in \mathbb{R}^{n \times m_h^{[l]}}$
- $\frac{\partial C}{\partial W^{[l]}} \in \mathbb{R}^{m_h^{[l-1]} \times m_h^{[l]}}$
- $\frac{\partial C}{\partial b^{[l]}} \in \mathbb{R}^{1 \times m_h^{[l]}}$

6.1.2 Equations

6.1.2.1 Calculating the Cost of the Predictions

- $C = \frac{1}{n} J(Y, \hat{Y})$

6.1.2.2 Calculating the Partial Derivatives

- $\frac{\partial C}{\partial \hat{Y}} = \frac{1}{n} \frac{\partial J}{\partial \hat{Y}}$
- $\frac{\partial C}{\partial A^{[L]}} = \frac{\partial C}{\partial \hat{Y}}$
- $\frac{\partial A^{[l]}}{\partial Z^{[l]}} = \sigma'^{[l]}(Z^{[l]}) \qquad \forall l = L, L-1, \dots, 1$
- $\frac{\partial Z^{[l]}}{\partial W^{[l]}} = A^{[l-1],T} \qquad \forall l = L, L-1, \dots, 1$
- $\frac{\partial Z^{[l]}}{\partial b^{[l]}} = 1_{1 \times n} \qquad \forall l = L, L-1, \dots, 1$
- $\frac{\partial Z^{[l]}}{\partial A^{[l-1]}} = W^{[l],T} \qquad \forall l = L, L-1, \dots, 2$

6.1.2.3 Applying the Chain Rule

- $\frac{\partial C}{\partial A^{[L]}} = \frac{1}{n} \frac{\partial J}{\partial \hat{Y}}$
- $\frac{\partial C}{\partial W^{[l]}} = \frac{\partial Z^{[l]}}{\partial W^{[l]}} \left(\frac{\partial C}{\partial A^{[l]}} : \frac{\partial A^{[l]}}{\partial Z^{[l]}} \right) = A^{[l-1],T} \left(\frac{\partial C}{\partial A^{[l]}} : \sigma'^{[l]}(Z^{[l]}) \right) \qquad \forall l = L, L-1, \dots, 1$
- $\frac{\partial C}{\partial b^{[l]}} = \frac{\partial Z^{[l]}}{\partial b^{[l]}} \left(\frac{\partial C}{\partial A^{[l]}} : \frac{\partial A^{[l]}}{\partial Z^{[l]}} \right) = 1_{1 \times n} \left(\frac{\partial C}{\partial A^{[l]}} : \sigma'^{[l]}(Z^{[l]}) \right) \qquad \forall l = L, L-1, \dots, 1$
- $\frac{\partial C}{\partial A^{[l-1]}} = \left(\frac{\partial C}{\partial A^{[l]}} : \frac{\partial A^{[l]}}{\partial Z^{[l]}} \right) \frac{\partial Z^{[l]}}{\partial A^{[l-1]}} = \left(\frac{\partial C}{\partial A^{[l]}} : \sigma'^{[l]}(Z^{[l]}) \right) W^{[l],T} \qquad \forall l = L, L-1, \dots, 2$

6.1.2.4 Updating the Trainable Parameters

- $\Delta W^{[l]} = -\alpha \frac{\partial C}{\partial W^{[l]}} \qquad \forall l = L, L-1, \dots, 1$
- $\Delta b^{[l]} = -\alpha \frac{\partial C}{\partial b^{[l]}} \qquad \forall l = L, L-1, \dots, 1$

6.2 Activation Functions' Derivatives

$$\sigma'^{[l]}\left(Z^{[l]}\right)_{i,j,u,v} = \frac{\partial}{\partial Z^{[l]}_{u,v}} \sigma^{[l]}\left(Z^{[l]}\right)_{i,j}$$

- ReLU $$\sigma'^{[l]}\left(Z^{[l]}\right)_{i,j,u,v} = \begin{cases} 1 & i = u \text{ and } j = v \text{ and } Z^{[l]}_{u,v} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Sigmoid $$\sigma'^{[l]}\left(Z^{[l]}\right)_{i,j,u,v} = \begin{cases} A^{[l]}_{i,j}\left(1 - A^{[l]}_{i,j}\right) & i = u \text{ and } j = v \\ 0 & \text{otherwise} \end{cases}$$

- Softmax $$\sigma'^{[l]}\left(Z^{[l]}\right)_{i,j,u,v} = \begin{cases} A^{[l]}_{i,j}\left(1 - A^{[l]}_{i,j}\right) & i = u \text{ and } j = v \\ -A^{[l]}_{i,j}A^{[l]}_{i,v} & i = u \text{ and } j \neq v \\ 0 & i \neq u \end{cases}$$

6.3 Loss Functions' Derivatives

- SSE $$\frac{\partial J}{\partial \hat{Y}_{i,j}} = \hat{Y}_{i,j} - Y_{i,j} \quad \Rightarrow \quad \frac{\partial J}{\partial \hat{Y}} = \hat{Y} - Y$$
- BCE $$\frac{\partial J}{\partial \hat{Y}_{i,j}} = \frac{\hat{Y}_{i,j} - Y_{i,j}}{\hat{Y}_{i,j}\left(1 - \hat{Y}_{i,j}\right)} \quad \Rightarrow \quad \frac{\partial J}{\partial \hat{Y}} = \left(\hat{Y} - Y\right) \oslash \left(\hat{Y} \odot \left(1 - \hat{Y}\right)\right)$$
- CCE $$\frac{\partial J}{\partial \hat{Y}_{i,j}} = -\frac{Y_{i,j}}{\hat{Y}_{i,j}} \quad \Rightarrow \quad \frac{\partial J}{\partial \hat{Y}} = -Y \oslash \hat{Y}$$

7 Different Types of Gradient Descent

7.1 Batch

Batch gradient descent involves computing the gradient of the entire dataset's cost function with respect to the model parameters in each iteration. This method provides a stable and accurate estimate of the gradient but can be computationally expensive, especially for large datasets.

7.2 Stochastic

Stochastic gradient descent takes a different approach by updating the model parameters after evaluating the gradient for each individual data point in the training set. While this can be computationally more efficient, it introduces higher variability in the updates and may result in noisy convergence. Despite the increased variance, stochastic gradient descent is particularly useful for large datasets and online learning scenarios.

7.3 Mini-Batch

Mini-batch gradient descent strikes a balance between the previous two approaches. It involves dividing the dataset into smaller batches and computing the gradient based on each mini-batch. This method combines the computational efficiency of stochastic gradient descent with the stability of batch gradient descent. The mini-batch size is a hyperparameter that can be tuned to find the right balance between efficiency and accuracy, making mini-batch gradient descent a popular choice in many machine learning applications.

8 Appendix

8.1 Some Mathematical Notations and Their NumPy Equivalents

- AB $\equiv$ `A @ B`
- $A : B$ $\equiv$ `np.tensordot(A, B)`
- $A \odot B$ $\equiv$ `A * B`
- $A \oslash B$ $\equiv$ `A / B`