Lecture Notes on

Deep Learning

David Raj M

Last updated: August 19, 2024

Note: Please note that, this lecture notes is incomplete, may have printing mistakes, and its available online only for the reference. It is not a substitute for your textbook.

Contents

Chapter 1

Introduction

1.1 What is neural network?

1.2 McCulloch–Pitts unit and Thresholding logic

1.3 Perceptron Learning Algorithm

1.4 Computing the Feed Forward Pass

Problem 1.1. Consider the neural network shown in Figure 1.1:

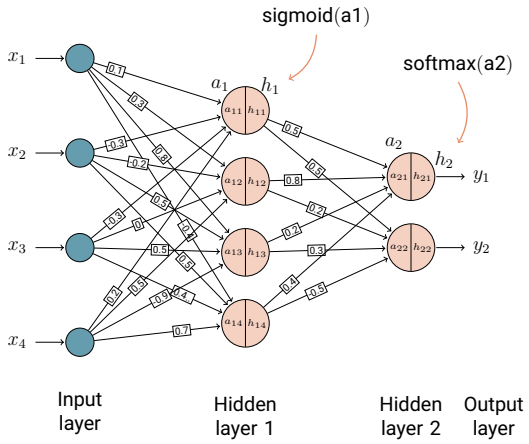


Figure 1.1: Network for Problem 1.1

(a) Draw the computational graph for this network.
(b) Write the network in composition of functions.
(c) What is the closed form of this network?
(d) Compute one forward pass for the input $[4, 5, 2, 9]$.

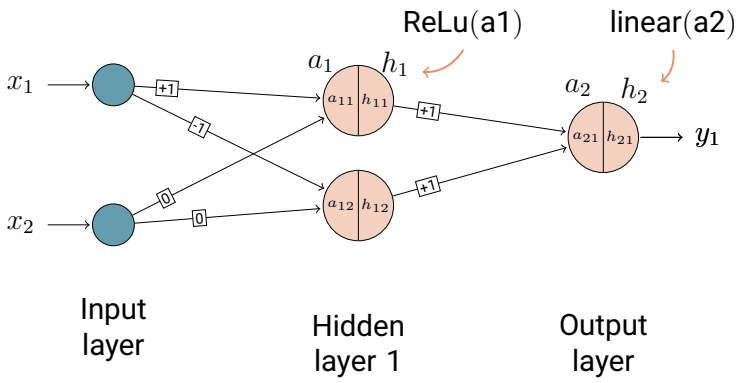


Figure 1.3: Example of a Non Linear Activation

1.5 Why do we need activation function?

Activation function is one of the important component of neural network architecture. It is responsible for introducing the non-linearity in the architecture.

Remark 1. If there is no activation function in any layer of the neural network (or equivalently using only identify activation function), the resulting neural network is just the linear regression only. (why?)

For example, consider the dataset, were the first row and the last row belongs to one class (yes) and middle one belongs to the other class (no).

x_1	x_2	y
-1	2	Yes
1	2	No
2	2	Yes

Table 1.1: Example dataset

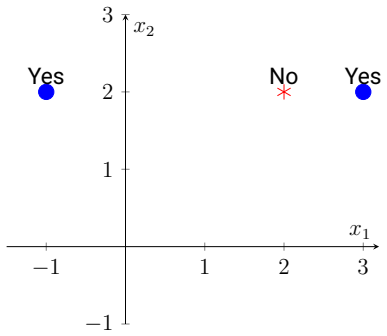


Figure 1.2: Scatter plot of the dataset Table 1.1

Note that, the dataset is not linearly separable (that is, we will not be able to draw a line and say that one side represent the "Yes" class and other side represent the "No" class).

However, if we consider one hidden layer (Ref Figure 1.3), with Relu activation function (ie., $\max\{0, x\}$) with the weight matrix $\begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}$, we get, linearly separable points which can be further classified using linear classification output layer.

Calculations:

$$X = \begin{bmatrix} -1 & 2 \\ 0 & 2 \\ 1 & 2 \end{bmatrix}, W = \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}, \text{ then,}$$

$$a_1 = XW = \begin{bmatrix} -1 & 2 \\ 0 & 2 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}$$
$$= \begin{bmatrix} -1 & 1 \\ 0 & 0 \\ 1 & -1 \end{bmatrix}$$

Now, by applying Relu on each entry, we get

$$ReLu(a_1) = \begin{bmatrix} \max\{0,-1\} & \max\{0,1\} \\ \max\{0,0\} & \max\{0,0\} \\ \max\{0,1\} & \max\{0,-1\} \end{bmatrix}$$
$$= \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}.$$

Note that, this set of points are now linearly separable (Refer Figure 1.4).

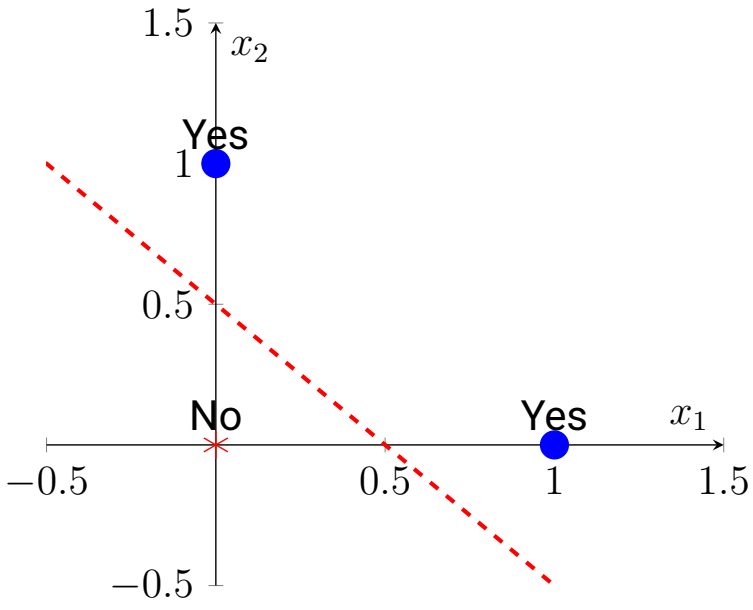


Figure 1.4: After Relu Activation

Remark 2. You could have already identified that we have set the weight safely to avoid using the x_2 in the previous neural network to make the data to linearly separable. That's okay. (why?) However, in real-world scenarios, it's common to initialize the weights

randomly and train the neural network to learn these weights. The reason for this is that in real data, the initial weights are typically unknown and need to be learned from the data. Random weight initialization allows the neural network to explore different weight configurations during training, leading to better generalization and improved performance on unseen data.

Now, that we have understood that we need some non-linear activation functions to estimate the output in a better manner. In the rest of the sections, we shall introduce some of the commonly used activation functions.

1. **Identify function:** $g(x) = x$.
2. **Sign function:** $g(x) = \begin{cases} -1 & \text{if } x < 0 \\ +1 & \text{if } x \geq 0 \end{cases}$.
3. **Sigmoid function:** $g(x) = \text{sigmoid}(x) = \dfrac{1}{1+e^{-x}}$
4. **tanh function:** $g(x) = \tanh(x)$.

   Remark 3. Note that,

   $$\begin{aligned} \tanh(x) &= \frac{e^{2x}-1}{e^{2x}+1} \\ &= 2\,\text{sigmoid}(2x) - 1 \end{aligned}$$

   So, we can use sigmoid activation function when we need only positive values (between 0 and 1) as output, whereas tanh activation function when we need output ranging from -1 to 1.

5. **ReLu function:** In recent years, piecewise linear activation functions are also used majorly in the literature, such as ReLu activation function.

   $$\text{ReLU}(x) = \max\{0, x\}, \tag{1.1}$$

   or equivalently,

   $$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{1.2}$$

6. **Hard tanh function:** This is also one of the piecewise linear activation function.

   $$\text{hardtanh}(x) = \max\{\min\{x, 1\}, -1\}. \tag{1.3}$$

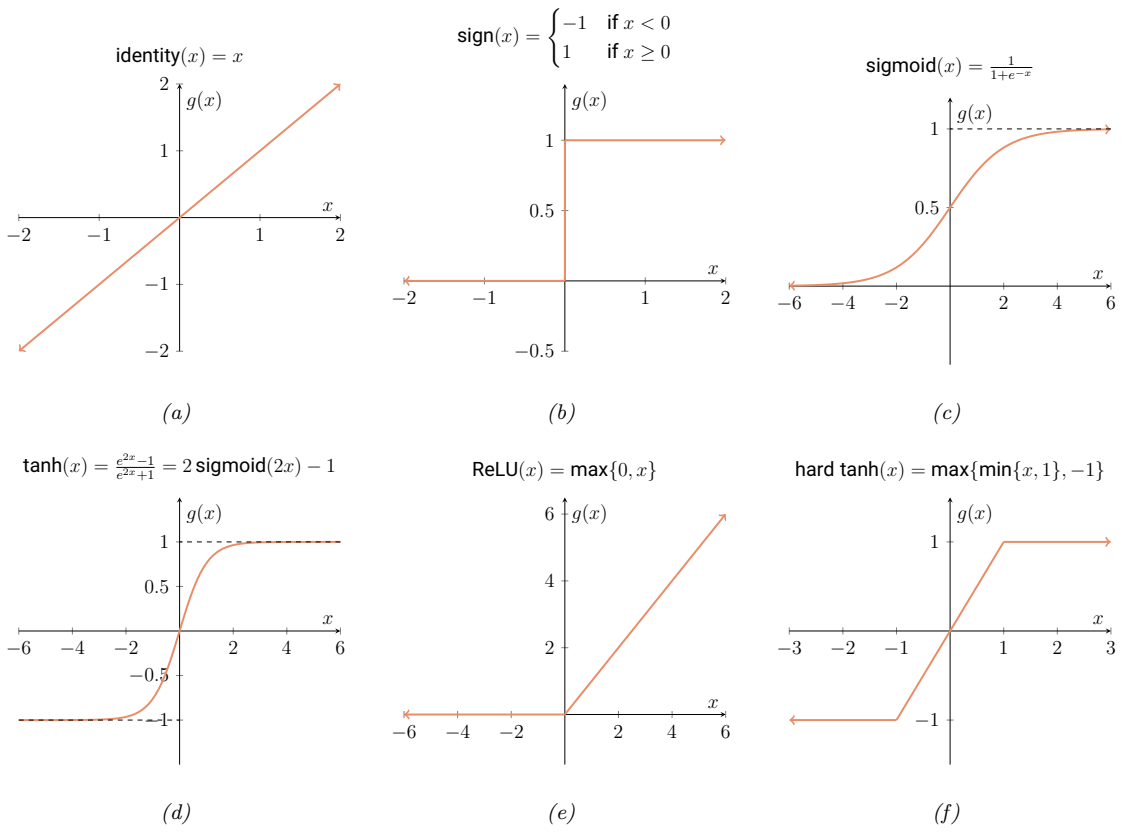


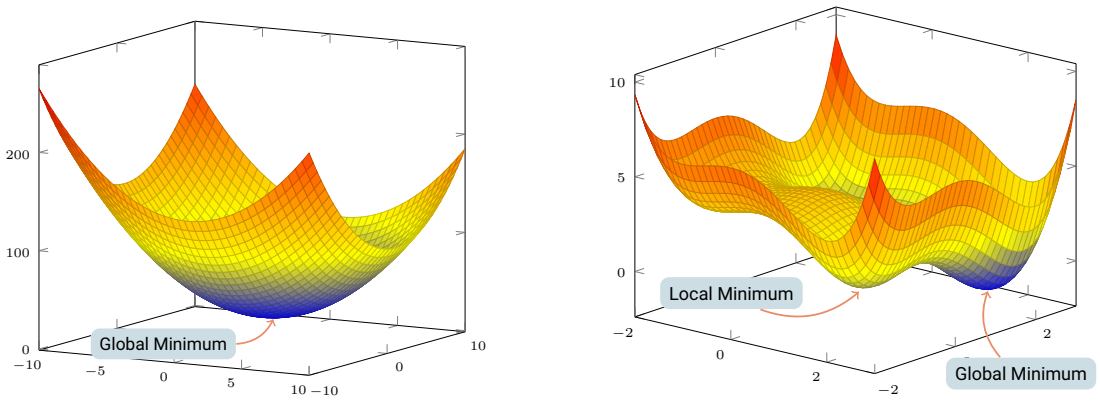
Figure 1.5: Commonly used activation functions

Chapter 2

Training algorithms for Feedforward networks

2.1 Loss function

Loss function should be convex to use gradient descent algorithm. (Why?)



(a) Example of a Convex function

(b) Example of a Non-convex function

Figure 2.1: Convex functions have only one minimum whereas non-convex functions may have more than one local minimum values

2.1.1 Common Loss functions

1. **Least square Loss function:** When the output layer contains the numerical outputs, we use squared error loss function.

$$L(y, \hat{y}) = \sum_{i=1}^{k} (y_i - \hat{y}_i)^2, \tag{2.1}$$

where k is the number of neurons in the output layer.

2. **Logistic Error function:** When we have binary output such as $\{-1, +1\}$, then the prediction $\hat{y}$ uses sigmoid activation function to compute the probability that the true output is 1 [Refer Figure 2.2].

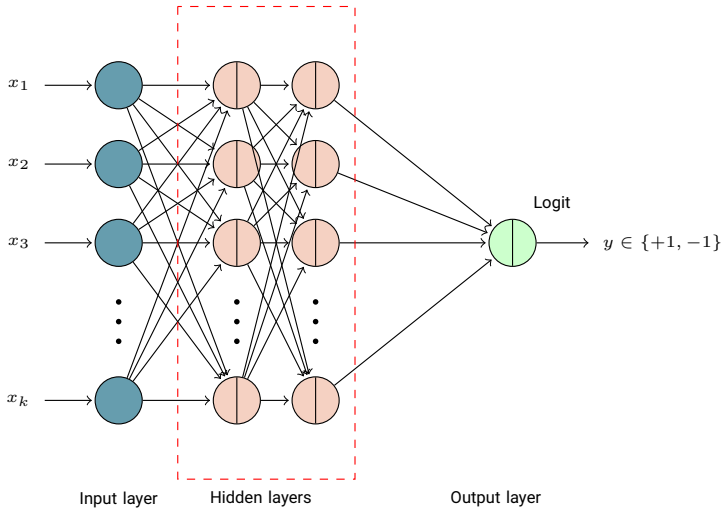


Figure 2.2: Binary Output

Then note that, $|y/2 - 1/2 + \hat{y}|$ gives the probability that the predicted is correct. Hence, the loss function is defined as

$$L(y, \hat{y}) = |y/2 - 1/2 + \hat{y}| \,. \tag{2.2}$$

3. **Cross Entropy Loss:** When we have the output layer which is supposed to be predicting k classes, then we apply softmax function to the final output layer to get the probabilities of each class. That is, $\hat{y}_1, \ldots, \hat{y}_k$ will be the probablities of that they belong to the class 1,. . . ,k, respectively [Refer Figure 2.3].

   In this case, the cross entropy loss function is utilized, which is defined as:

$$L(y, \hat{y}) = -\sum_{i=1}^{k} y_k \log \hat{y}_k \tag{2.3}$$

Problem 2.1. Consider the least squares loss function for the network illustrated in Problem 1.1. Assume that the true label for the given data is $[1, 0]$. Update the weights $w_{21}^{(2)}$ and $w_{13}^{(1)}$. Also, update $w_{23}^{(1)}$. Which of the previously computed partial derivatives can you use for these updates?

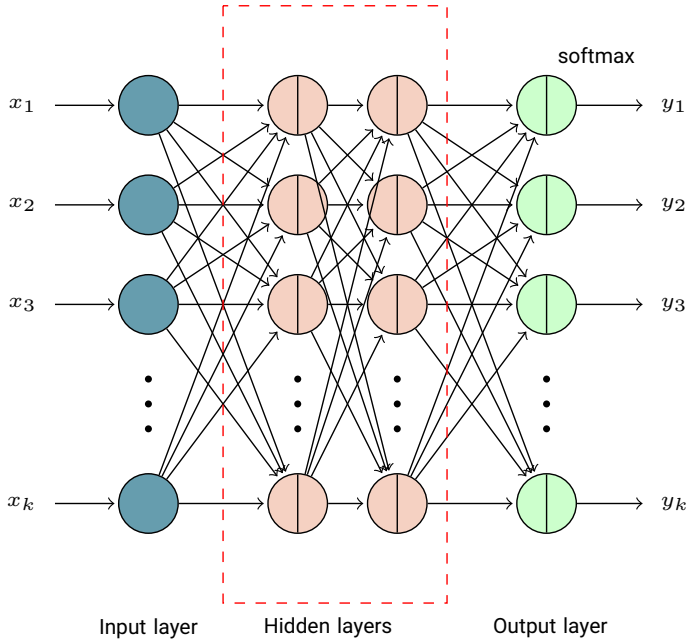


Figure 2.3: Multiple Categorical Output as Probabilites

2.2 Derivatives of Loss function

2.3 Saturation

2.4 Heuristics to avoid local minima

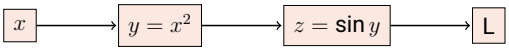
2.5 Empirical Risk Minimization

2.6 Regularization

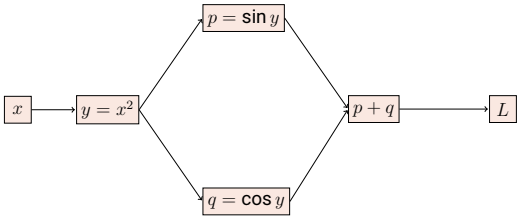
2.7 Auto encoders

2.8 Exercises

1. Write the expression for $\frac{\partial L}{\partial x}$ from the following computational graph.



2. Write the expression for $\frac{\partial L}{\partial x}$ from the following computational graph.



3. Consider a two input neuron that multiplies its two inputs x_1 and x_2 to obtain the output o. Let L be the loss function that is computed at o. Suppose that you know that $\frac{\partial L}{\partial o} = 5$, $x_1 = 2$ and $x_2 = 3$. Compute the values of $\frac{\partial L}{\partial x_1}$ and $\frac{\partial L}{\partial x_2}$

4. Let $f(x)$ be defined as follows:

$$f(x) = \sin x + \cos x.$$

Consider the function $f(f(f(f(x))))$.

   (a) Write this function in closed form. (note that computing derivative with respect to x is very ugly)

   (b) Evaluate the derivative of this function at $x = \pi/3$ radians by using a computational graph abstraction.

5. Consider a network with two inputs x_1 and x_2. It has two hidden layers, each of which contain two units. Assume that the weights in each layer are set so that top unit in each layer applies sigmoid activation to the sum of its inputs and the bottom unit in each layer applies tanh activation to the sum of its inputs. Finally, the single output node applies ReLU activation to the sum of its two inputs. Write the output of this neural network in closed form as a function of x_1 and x_2.

6. Compute the partial derivative of the closed form computed in the previous exercise with respect to x_1. Is it practical to compute derivatives for gradient descent in neural networks by using closed-form expressions (as in traditional machine learning)?

Chapter 3

Better Learning Algorithms

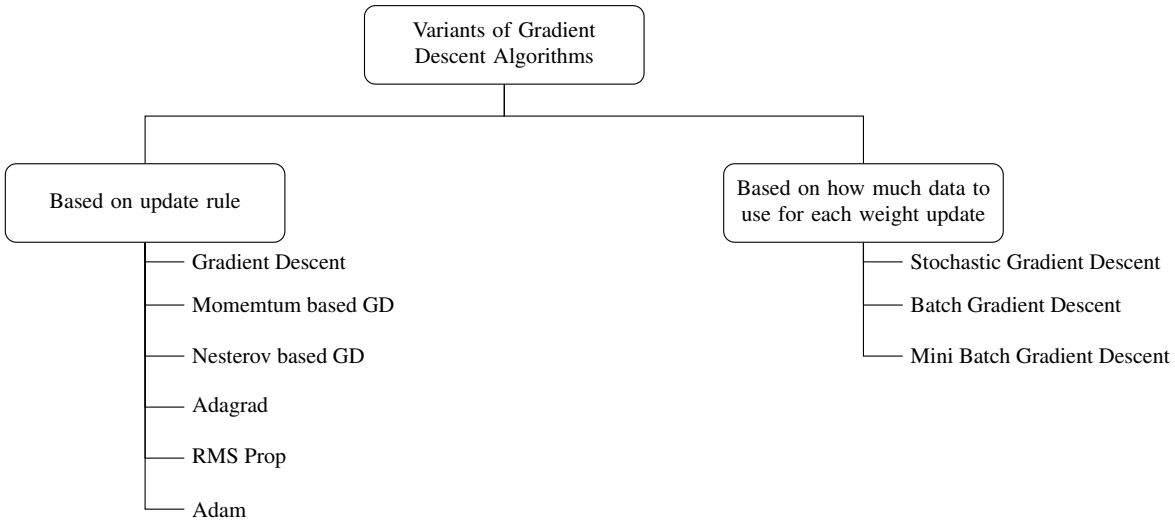


Figure 3.1: Variants of Gradient Descent Algorithms

3.1 Based on Update Rule

3.1.1 Gradient Descent

$$w_{t+1} = w_t + \eta \nabla w_t$$

3.1.2 Momentum based Gradient Descent

$$v_t = \nu * v_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - v_t$$

3.1.3 Nesterov based Gradient Descent

$$
\begin{aligned}
w_{\text{temp}} &= w_t + \nu * v_{t-1} \\
w_{t+1} &= w_{\text{temp}} - \eta \nabla w_{\text{temp}} \\
v_t &= \nu * v_{t-1} + \eta \nabla w_{\text{temp}}
\end{aligned}
$$

3.1.4 Adagrad

$$
\begin{aligned}
v_t &= v_{t-1} + (\nabla w_t)^2 \\
w_{t+1} &= w_t - \frac{\eta}{\sqrt{v_t} + \epsilon} \nabla w_t
\end{aligned}
$$

3.1.5 RMSProp

$$
\begin{aligned}
v_t &= \beta * v_{t-1} + (1 - \beta) (\nabla w_t)^2 \\
w_{t+1} &= w_t - \frac{\eta}{\sqrt{v_t} + \epsilon} \nabla w_t
\end{aligned}
$$

3.1.6 Adam

$$
\begin{aligned}
m_t &= \beta_1 * v_{t-1} + (1 - \beta_1) (\nabla w_t) \\
v_t &= \beta_2 * v_{t-1} + (1 - \beta_2) (\nabla w_t)^2 \\
w_{t+1} &= w_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t
\end{aligned}
$$

3.2 Based on Data to use

Depending on the data to use, there are three stratergies to update the parameters. They are batch gradient descent, stochastic gradient descent and mini-batch gradient descent. Some basic notions to build an algorithm or to understand the difference in these stratergies are given below

- 1 epoch = one pass over the entire data
- 1 step = one update of the parameters
- N = number of data points
- B = mini-batch size

Table 3.1: Algorithm with their number of update in the parameter in one epoch

Algorithm	# of steps in one epoch
Batch gradient descent	1
Stochastic gradient descent	N
Mini-Batch gradient descent	N/B

Chapter 4

Deep Neural Networks