Scientific Computing Lecture Series
Introduction to Deep Learning

Mustafa Kütük*

*Scientific Computing, Institute of Applied Mathematics

Lecture III

Deep Learning: An Introduction for Applied Mathematicians

Lecture III–Outline

Introduction

- Deep learning is a powerful function that mimics the human brain in terms of its working style for decision making with data processing and pattern creation.

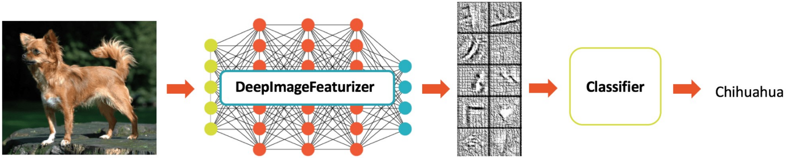
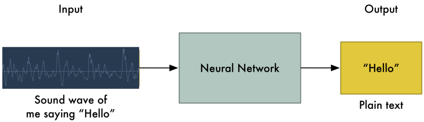


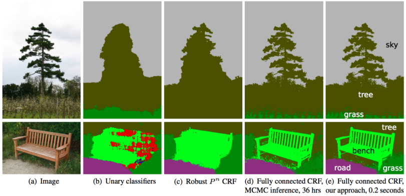
Figure 1: Classification with deep learning[1]

[1] https://databricks.com/blog/2017/06/06/databricks-vision-simplify-large-scale-deep-learning.html

Deep learning is widely used areas such that

- Image/Text Classification,
- Speech Recognition[2],
- Image Segmentation[3].





[2] https://medium.com/@ageitgey/machine-learning-is-fun-part-6-how-to-do-speech-recognition-with-deep-learning-28293c162f7a

[3] http://blog.qure.ai/notes/semantic-segmentation-deep-learning-review

Introduction & Goals

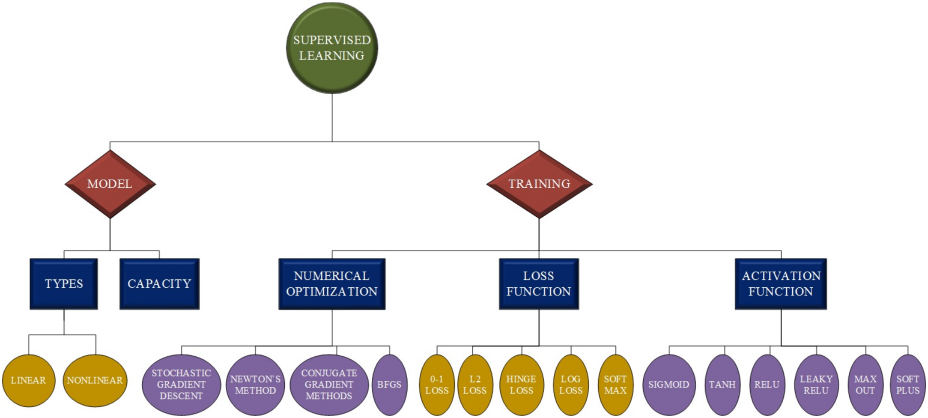
There are also some areas of mathematics that uses deep learning:

- Approximation Theory,
- Numerical Optimization,
- Linear Algebra.

Our aims are

- to give brief introduction for deep learning,
- to define some terms related to this area,
- to apply deep learning for a small example in Matlab.

Scheme of the Supervised Learning



Example Problem

- A map, which shows the oil drilling sites, is given below. The circles (class A) denote the successful outcome while crosses (class B) are the unsuccessful outcome.

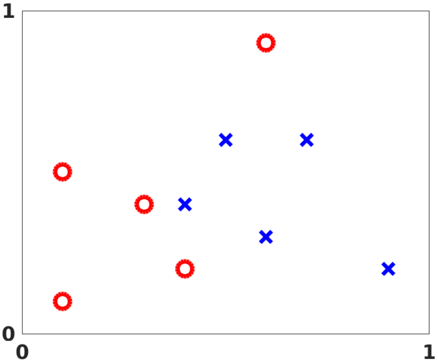


Figure 2: Oil Drilling Sites

A Simple Model:The Perceptron

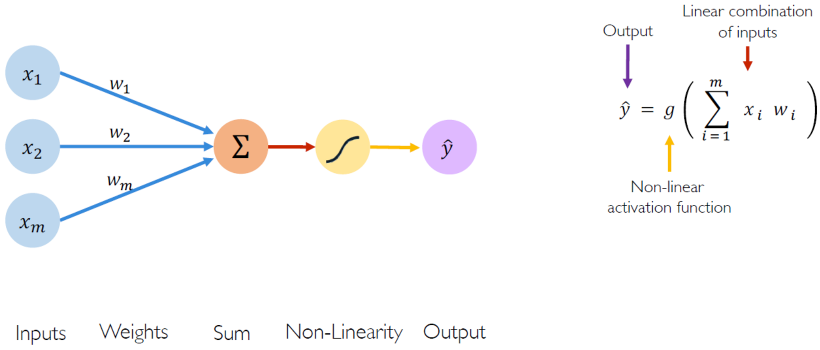
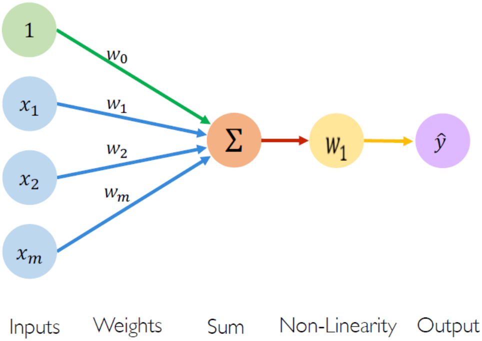
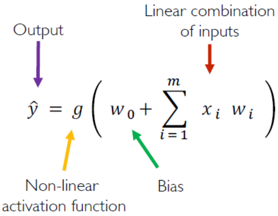


Figure 3: MIT's 6.S191:Introduction to Deep Learning Course[4]

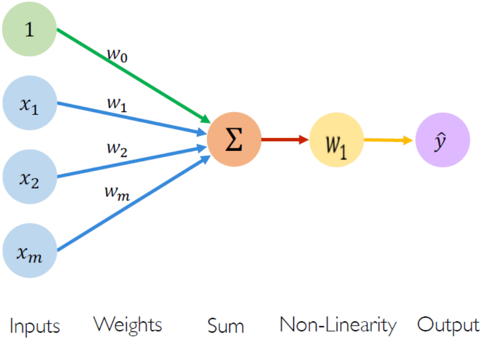
[4] http://introtodeeplearning.com/2019/materials/2019_6S191_L1.pdf

A Simple Model: The Perceptron





A Simple Model:The Perceptron

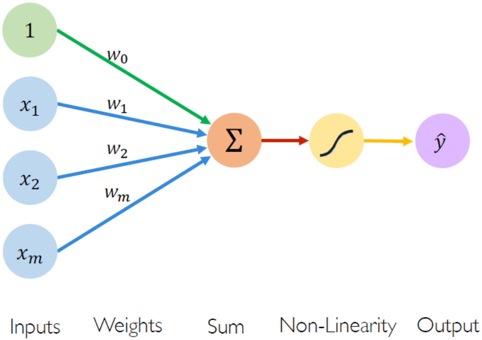


$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \; w_i \right)$$

$$\hat{y} = g\,(\, w_0 + \boldsymbol{X}^T \boldsymbol{W} \,)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

A Simple Model: The Perceptron

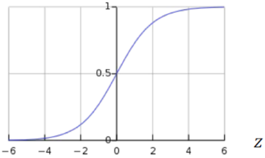


Activation Functions

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

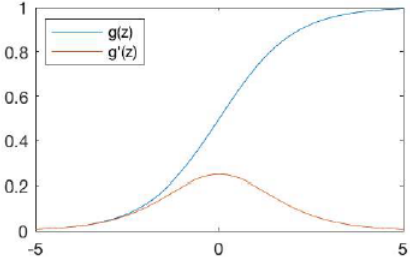
- Example: sigmoid function

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



Activation Functions:Sigmoid Function

Sigmoid Function

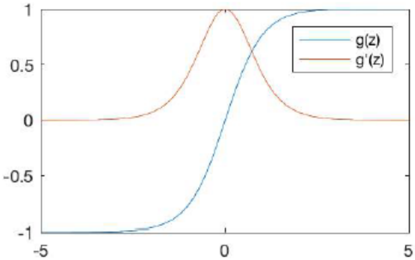


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

Activation Functions:Hyperbolic Tangent Function

Hyperbolic Tangent

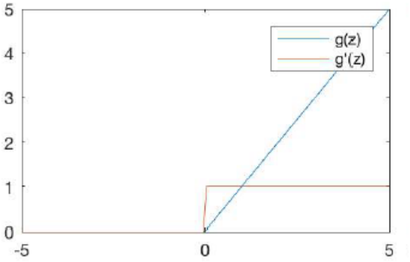


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Activation Functions:ReLU Function

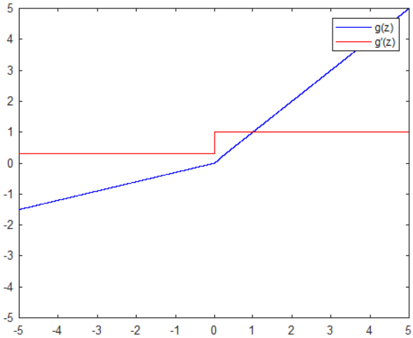
Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

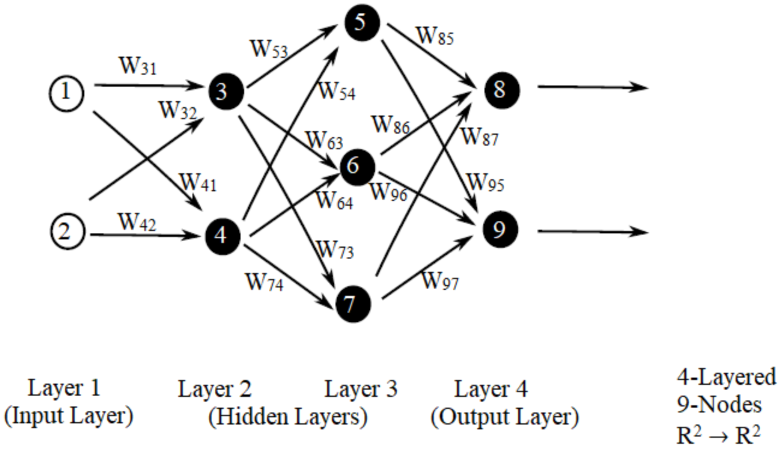
Activation Functions:Leaky ReLU Function



$$g(z) = \begin{cases} az, & z < 0 \\ z, & otherwise \end{cases}$$

$$g'(z) = \begin{cases} a, & z < 0 \\ 1, & otherwise \end{cases}$$

Multilayer Perceptron(MLP)



Multilayer Perceptron(MLP)

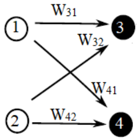
- The activation function can be written as

$$\sigma(\mathbf{WX} + b)$$

- Input of these model can be shown as

$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Multilayer Perceptron(MLP)



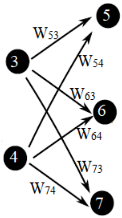
- The weight matrix and bias vector of the 2nd layer can be shown as

$$W^{[2]} = \begin{bmatrix} W_{31} & W_{32} \\ W_{41} & W_{42} \end{bmatrix} \quad b^{[2]} = \begin{bmatrix} b_3 \\ b_4 \end{bmatrix}$$

- The output of the 2nd layer can be obtained as

$$\begin{bmatrix} x_3 \\ x_4 \end{bmatrix} = \sigma\left(\begin{bmatrix} W_{31}x_1 + W_{32}x_2 + b_3 \\ W_{41}x_1 + W_{42}x_2 + b_4 \end{bmatrix} \right)$$

Multilayer Perceptron(MLP)



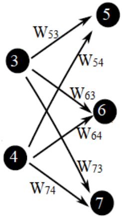
- The activation function of the 3rd layer can be written as

$$\sigma(W^{[3]}\sigma(W^{[2]}X + b^{[2]}) + b^{[3]})$$

- The weight matrix and bias vector of the second layer can be shown as

$$W^{[3]} = \begin{bmatrix} W_{53} & W_{54} \\ W_{63} & W_{64} \\ W_{73} & W_{74} \end{bmatrix} \quad b^{[3]} = \begin{bmatrix} b_5 \\ b_6 \\ b_7 \end{bmatrix}$$

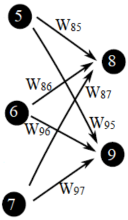
Multilayer Perceptron(MLP)



- The output of the 3rd layer can be obtained as

$$\begin{bmatrix} x_5 \\ x_6 \\ x_7 \end{bmatrix} = \sigma\left(\begin{bmatrix} W_{53}x_3 + W_{54}x_4 + b_5 \\ W_{63}x_3 + W_{64}x_4 + b_6 \\ W_{73}x_3 + W_{74}x_4 + b_7 \end{bmatrix} \right)$$

Multilayer Perceptron(MLP)



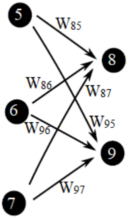
- The activation function of the 4th layer can be written as

$$\sigma(W^{[4]}\sigma(W^{[3]}\sigma(W^{[2]}X + b^{[2]}) + b^{[3]}) + b^{[4]})$$

- The weight matrix and bias vector of the third layer can be shown as

$$W^{[4]} = \begin{bmatrix} W_{85} & W_{86} & W_{87} \\ W_{95} & W_{96} & W_{97} \end{bmatrix} \quad b^{[4]} = \begin{bmatrix} b_8 \\ b_9 \end{bmatrix}$$

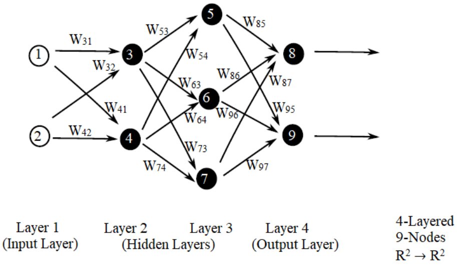
Multilayer Perceptron(MLP)



- The output of the 4th layer can be obtained as

$$\begin{bmatrix} x_8 \\ x_9 \end{bmatrix} = \sigma\left(\begin{bmatrix} W_{85}x_5 + W_{86}x_6 + W_{87}x_7 + b_8 \\ W_{95}x_5 + W_{96}x_6 + W_{97}x_7 + b_9 \end{bmatrix} \right)$$

Multilayer Perceptron(MLP)



- As a result, the overall model can be summarized as

$$\textbf{Input:} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \textbf{Output:} F(X) = \begin{bmatrix} x_8 \\ x_9 \end{bmatrix} \Longrightarrow F : \mathbb{R}^2 \rightarrow \mathbb{R}^2,$$

and this model includes totally 23 unknown parameters (16 weight parameters, 7 bias parameters).

Multilayer Perceptron(MLP)

- Aim is to produce a classifier by optimizing over all unknown parameters.
- We will require $F(x)$ to be close to $[1,0]^T$ for data points in class A and close to $[0,1]^T$ for data points in class B. Then, the classifier is:

$$class\ A,\ if\ F_1(x) > F_2(x)$$

$$class\ B,\ if\ F_1(x) < F_2(x)$$

- This requirement on F is specified through a cost function.

$$y(x^i) = \begin{cases} \begin{bmatrix} 1 & 0 \end{bmatrix}^T, & if\ x^i\ is\ in\ class\ A \\ \begin{bmatrix} 0 & 1 \end{bmatrix}^T, & if\ x^i\ is\ in\ class\ B \end{cases}$$

Cost Function

- Then the cost function can be shown as

$$Cost(W^{[2]}, W^{[3]}, W^{[4]}, b^{[2]}, b^{[3]}, b^{[4]}) = \frac{1}{10}\sum_{i=1}^{10}\frac{1}{2}||y(x^i) - F(x^i)||_2{}^2$$

  where $y(x^i)$ is the ground truth (labeled data) and $F(x^i)$ is the model output.

- This is a quadratic cost function (aka L_2-loss function).
- Choosing the weights and biases in a way that minimizes the cost function is referred to as **training** the network.

Steepest Descent Method

- The unknown parameters can be stored as a single vector that we call **p**.
- For our example, $\mathbf{p} \in \mathbb{R}^{23}$.
- Generally, $\mathbf{p} \in \mathbb{R}^s$ and $Cost : \mathbb{R}^s \to \mathbb{R}$.
- The classical method is steepest descent or gradient descent.

$$Cost(p + \Delta p) \approx Cost(p) + \sum_{r=1}^{s} \frac{\partial Cost(p)}{\partial p_\mathrm{r}} \Delta p_\mathrm{r} \Longrightarrow \textit{From Taylor Series Exp.}$$

$$(\nabla Cost(p))_\mathrm{r} = \frac{\partial Cost(p)}{\partial p_\mathrm{r}} \Longrightarrow Cost(p + \Delta p) \approx Cost(p) + \nabla Cost(p)^T \Delta p$$

- We have to choose Δp such that $\nabla Cost(p)^T \Delta p < 0$.
- Therefore, we should choose Δp to lie in the direction $-\nabla Cost(p)$. We can obtain

$$p_\mathrm{k+1} = p_\mathrm{k} - \eta \nabla Cost(p)$$

where η is the stepsize (aka **learning rate**).

Steepest Descent Method

- The cost function for individual terms is

$$C(x^i) = \frac{1}{2}||y(x^i) - a^{[L]}(x^i)||_2{}^2$$

$$\nabla Cost(p) = \frac{1}{N}\sum_{i=1}^{N}\nabla C(x^i)(p)$$

- When there are a large number of parameters and a large number of training points, computing the gradient vector $\nabla Cost(p)$ at every iteration of the steepest descent method can be expensive.

Stochastic Gradient Descent(SGD)

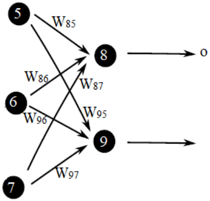
- An alternative way is to replace the mean of the individual gradients over all training points by the gradient at a single, randomly chosen, training point.
  1. Choose an integer i uniformly at random from {1,2,3,...,N}
  2. Update $p \to p - \eta \nabla C(x^i)(p)$
- As the iteration proceeds, the method sees more training points. So the cost decreases after a while.

Backpropagation

- An application of the chain rule.
- To compute the gradient of the error in the output layer, one has to compute the gradient iteratively layer by layer from the output layer to the input layer.

Backpropagation

- Let's consider the example given below



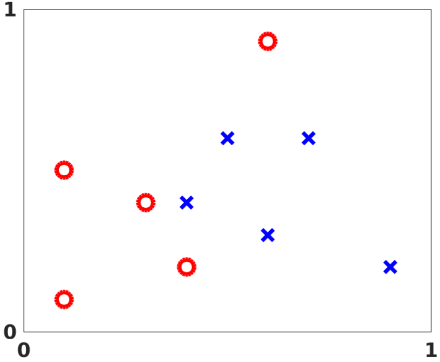
$$o = g(net_o) = g\left(\sum_{i=5}^{7} W_{oi}x_i\right)$$

$$\frac{\partial E}{\partial W_{oi}} = \frac{\partial E}{\partial o}\frac{\partial o}{\partial net_o}\frac{\partial net_o}{\partial W_{oi}}$$

where $\frac{\partial net_o}{\partial W_{oi}} = x_i$ and $\frac{\partial o}{\partial net_o} = g'$ (derivative of the activation function).

Oil Drilling Sites Problem

- Let's turn back to our problem and try to solve it in Matlab by using 4-layered MLP which is shown before.



References & Useful Links

- C.F. Higham, and D.J. Higham, "Deep Learning: An Introduction for Applied Mathematicians", SIAM Review, 2019.
- MIT 6.S191:Introduction to Deep Learning website, `http://introtodeeplearning.com`
- `http://playground.tensorflow.org`
- `https://www.wikiwand.com/en/Backpropagation`

Deep Learning Related Courses

- CENG562 - Machine Learning
- CENG783 - Deep Learning
- CENG564 - Pattern Recognition
- MMI727 - Deep Learning: Methods and Applications
- EE583 - Pattern Recognition
- IAM557 - Statistical Learning and Simulation

Optimization Related Courses

- MATH402 - Introduction to Optimization
- IAM566 - Numerical Optimization
- EE553 - Optimization

For More Information

- http://iam.metu.edu.tr/scientific-computing
- https://iam.metu.edu.tr/scientific-computing-lecture-series
- https://www.facebook.com/SCiamMETU/

...thank you for your attention !