1 Chapter 01 : Foundations of Deep Learning

1.1 McCulloch-Pitts Neuron

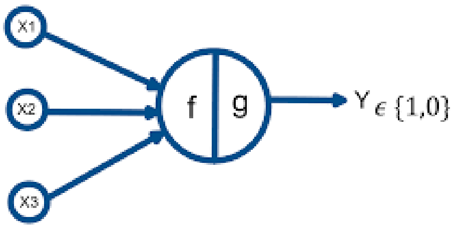


Figure 1: McCulloch-Pitts Neuron

Equations :

$$\begin{cases} y = g(f(x_1, x_2, ..., x_n)) \\ f(x_1, x_2, ..., x_n) = \sum\limits_{i=1}^{n} w_i * x_i \\ g(z) = \begin{cases} 1 \text{ if } z >= \theta \\ 0 \text{ otherwise} \end{cases} \end{cases}$$

Characteristics :

- weights and threshold are set manually (no learning).
- can not solve non-linear problems.
- takes only boolean inputs.

1.2 Perceptron

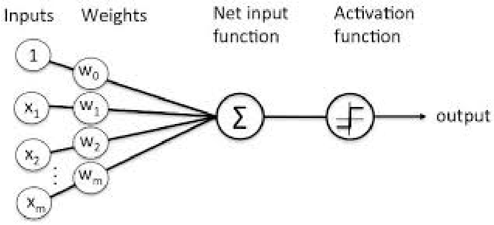


Figure 2: Perceptron

Equations :

$$\begin{cases} y = g(f(x_1, x_2, ..., x_n)) \\ f(x_1, x_2, ..., x_n) = W^t * x + b = \sum\limits_{i=1}^{n} w_i * x_i + b \\ g(z) = \begin{cases} 1 \text{ if } z >= 0 \\ 0 \text{ otherwise} \end{cases} \end{cases}$$

g is called the $Heaviside$ activation function.

Characteristics :

- weights and threshold are learned.
- can not solve non-linear problems.
- handles only classification problems.

Algorithm 1 Perceptron's learning algorithm

```
for $i \leftarrow 1$ to $L$ do
    for $j \leftarrow 1$ to $m$ do
        error $\leftarrow y_j - g(w^T * x_j + b)$
        $w \leftarrow w + r \times error \times x_j$
        $b \leftarrow b + r \times error$
    end for
end for
```

1.3 ADALINE

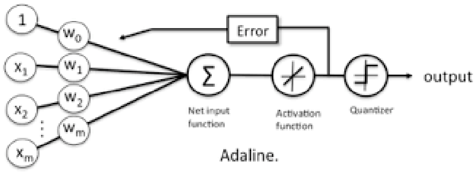


Figure 3: Adaline

the main difference between **Adaline** and **Perceptron** is in the learning process,adaline uses the continuous values (before applying Heaviside/Quantizer) to update the weights.

Equations :

$$\begin{cases} y = g(f(x_1, x_2, ..., x_n)) \\ f(x_1, x_2, ..., x_n) = W^t * x + b = \sum_{i=1}^{n} w_i * x_i + b \\ g(z) = \begin{cases} 1 \text{ if } z >= 0 \\ 0 \text{ otherwise} \end{cases} \end{cases}$$

Characteristics :

- weights and threshold are learned.
- can not solve non-linear problems.
- multi-layer perception is equivalent to a simple linear regression.

Algorithm 2 Adaline's learning algorithm

```
for $i \leftarrow 1$ to $L$ do
    for $j \leftarrow 1$ to $m$ do
        error $\leftarrow y_j - f(x_j)$
        $w \leftarrow w + \alpha \times error \times x_j$
        $b \leftarrow b + \alpha \times error$
    end for
end for
```

1.4 Multi-layer Perceptron

1.4.1 How can a multi-layer perceptron help solve more complex tasks ? : XOR Example

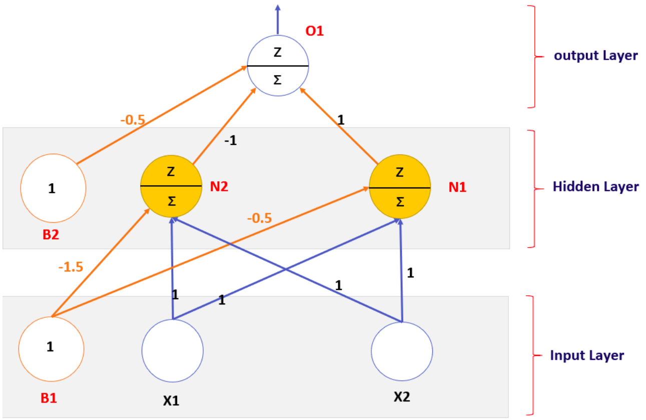


Figure 4: Multi-layer perceptron

x_1	x_2	n_1		n_2		o_1	
		z	y	z	y	z	y
0	0	-0.5	0	-1.5	0	-0.5	0
1	0	0.5	1	-0.5	0	0.5	1
0	1	0.5	1	-0.5	0	0.5	1
1	1	1.5	1	0.5	1	-0.5	0

1.4.2 Neutral Networks Characteristics

- **size :** number of the nodes in the model.
- **width :** number of nodes in a specific layer.
- **depth :** number of layers in the neutral network.
- **architecture :** the specific arrangement of the layers and nodes in the model.

1.4.3 How to design a Neutral Network architecture ?

- **Experimentation.**
- **Intuition :** comes from experience in a specific domain.
- **Go for depth :** deeper neutral networks can help solve complex problems but are more vulnerable to overfitting.
- **Borrow ides :** architectures that are proved to work well on similar problems is a good starting point.
- **Search :** Heuristics,Random search.

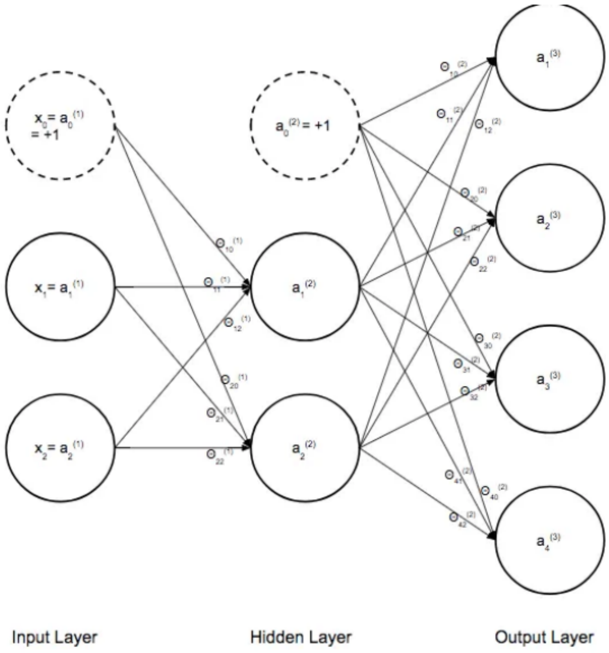
1.4.4 Neutral Networks : Forward pass

Notations :

- L : the number of layers.
- g_l : the activation function of the l^{th} layer.
- $a^{[l]}$: the output of the l^{th} layer.
- $z^{[l]}$: the output of the l^{th} layer before applying the activation function.
- $a_n^{[l]}$: the output of the n^{th} neuron of the l^{th} layer.
- θ_l : the weights of the l^{th} layer.

$$X = \begin{bmatrix} x_0 & x_1 & x_2 \\ 1 & 0.504 & -0.4161 \\ 1 & -0.99 & -0.6536 \\ 1 & 0.2837 & 0.9602 \end{bmatrix}$$

x_0 is always equal to 1 because it represents the bias.



Forward Equations

$$\begin{cases} a^{[1]} = X^t \\ a^{[l+1]} = g_l(\theta_{l+1} \times a_{[l]}) + \text{add a row of ones} \\ \hat{y} = (a^L)^t + \text{without the additional row of ones} \end{cases}$$

Loss

Cross entropy loss :

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{K} [-y_j^{(i)} * log(\hat{y}_j^{(i)})]$$

Binary cross entropy loss :

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} [-y^{(i)} * log(\hat{y}^{(i)}) - (1 - y^{(i)}) * log(1 - \hat{y}^{(i)})]$$

Backward Equations For an MLP with one hidden layer

$$\begin{cases} \frac{\partial J}{\partial \hat{\theta}_2} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial \hat{\theta}_2} \\ \frac{\partial J}{\partial z_2} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} = \hat{y} - y \text{ if the activation of the last layer is sigmoid or softmax} \\ \frac{\partial z_2}{\partial \hat{\theta}_2} = a_1 \text{ the output of hidden layer} \end{cases}$$

$$\begin{cases} \frac{\partial J}{\partial \theta_1} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial \theta_1} \\ \frac{\partial J}{\partial z_2} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} = \hat{y} - y \text{ if the activation of the last layer is sigmoid or softmax} \\ \frac{\partial z_2}{\partial a_1} = \theta_2 \\ \frac{\partial a_1}{\partial z_1} = \frac{\partial g_1}{\partial t}(z_1) \text{ depends on the hidden's layer activation function} \\ \frac{\partial z_1}{\partial \theta_1} = x \text{ the input data} \end{cases}$$

1.4.5 Activation functions

why use non-linear activation functions ?

Because a neutral network with a linear activation function is equivalent to a simple linear regression model,so it will perform poorly non-linear separable problems,linear activation is only used in the output layer in regression problems.

Common activation functions

name	expression	derivative	notes
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$\sigma(x) * (1 - \sigma(x))$	- used in hidden and output layer. - can cause vanishing gradient descent problem. - expensive to compute
ReLU	$(x) = max(0, x)$	$\begin{cases} 1 \text{ if } x >= 0 \\ 0 \text{ otherwise.} \end{cases}$	- used in hidden layers. - can cause dying relu problem. - doesn't cause dying vanishing gradient descent. - easy to compute
Tanh	$\tanh(x) = \frac{e^x + e^{-x}}{e^x - e^{-x}}$	$1 - tanh^2(x)$	
Softmax	$softmax(x) = \frac{e^x}{\sum i{=}1^n e_i^x}$		used in the output layer of multi-classifications problems.

Table 1: Common activation functions

2 Chapter 02 : Optimizing deep neutral networks

2.1 Batch Gradient Descent Vs Stochastic Gradient descent Vs Mini-Batch Gradient Descent

Batch GD	Stochastic GD	Mini-Batch GD
Processes all the dataset each iteration	processes one sample each iteration	process a portion (batch) of the dataset each iteration
$batch\_size = m$	$batch\_size = 1$	$batch\_size$ is a hyper-parameter.

	Advantages	Disadvantages
Batch GD	- guaranteed to converge in theory - unbiased estimate of the gradient	- slow for large datasets - memory issues for large datasets
Mini-Batch GD	- Faster than Batch GD - adds noise which can help improve generalization	- can cause oscillations - may require learning rate decay
Stochastic GD	- Same as Mini-Batch	- slow run time - adds a lot of noise

Classic SGD can cause oscillation and prevent using larger learning rates making convergence slower.

2.2 SGD with momentum

$$\begin{cases} W_{t+1} = W_t - \alpha * V_t \\ V_t = \beta * V_t - (1-\beta) * \Delta W_t \end{cases}$$

the default value for β is **0.9**.

2.3 RMSProp optimizer

$$\begin{cases} W_t = W_{t-1} - \alpha * \frac{\Delta w_t}{\sqrt{V_t + \epsilon}} \\ V_t = \beta * V_{t-1} + (1-\beta) * \Delta W_t^2 \end{cases}$$

the default values for β and ϵ are **0.999** and 10^{-8} respectively.

2.4 Adam optimizer

$$\begin{cases} W_t = W_{t-1} - \alpha * \frac{\Delta V_t}{\sqrt{S_t + \epsilon}} \\ S_t = \beta_2 * S_{t-1} + (1-\beta_2) * \Delta W_t^2 \\ V_t = \beta_1 * V_{t-1} + (1-\beta_1) * \Delta W_t \end{cases}$$

the default values for β_1,β_2 and ϵ are 0.9,0.999 and 10^{-8} respectively.

2.5 Learning rate decay

Time based decay :

$$\alpha_t = \alpha_0 \frac{1}{1 + decay\_rate * t}$$

Step based decay :

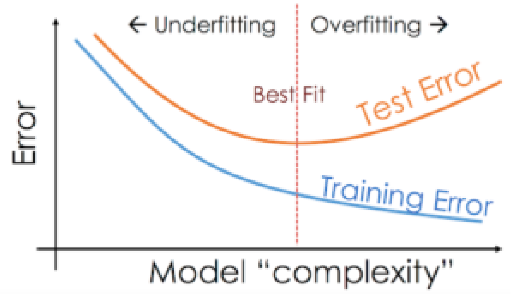
$$\alpha_t = \alpha_0 * drop\_rate^{t/epoch\_drop}$$

Exponential decay :

$$\alpha_t = \alpha_0 * e^{-decay\_rate * t}$$

2.6 Overfitting & Regularization Techniques

2.6.1 Overfitting Vs Model complexity



2.6.2 L1 & L2 normalization

L2 Norm (weight decay)

$$\begin{cases} J(\theta) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y},y) + \frac{\lambda}{2*m}\sum_{l=1}^{L}(\parallel W^l \parallel)^2 = E(\theta) + \frac{\lambda}{2*m}\sum_{l=1}^{L}(\parallel W^l \parallel)^2 \\ \parallel W^l \parallel^2 = \sum_{i,j} w_{i,j}^2 \\ \frac{\partial J}{\partial W} = \frac{\partial E}{\partial W} + \frac{\lambda}{m} * W \end{cases}$$

L1 Norm

$$\begin{cases} J(\theta) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y},y) + \frac{\lambda}{2*m}\sum_{l=1}^{L}(\parallel W^l \parallel) = E(\theta) + \frac{\lambda}{2*m}\sum_{l=1}^{L}(\parallel W^l \parallel) \\ \parallel W^l \parallel = \sum_{i,j} \mid w_{i,j} \mid \end{cases}$$

2.6.3 Dropout :

Dropout is a regularization technique that a regularization technique that randomly turning off some neurons each iteration,the number of the parameters to mask is controlled by the **dropout rate** parameter, the dropout is desactivated during inference.

2.6.4 Early stopping :

- Monitoring model performance : the choise of metric,validation set.
- Trigger to Stop : stop the training when the loss increases or became unstable for a certain number of epochs.
- The choice of the model : save the weights each time the loss decreases.

2.6.5 Batch Normalization :

Batch normalization is a regularization technique that speeds up training and handles internal covariant shift,batch normalization is an extra layer that normalizes the batch by subtracting its mean and dividing it by its standard deviation.

2.7 Gradient checking

Gradient checking is a technique used to verify whether the implantation of the backpropagation is true or not by calculating an approximation of the derivatives and comparing with the results of the backpropagation.

$$\frac{\partial J}{\partial \theta} \approx \frac{J(\theta + \epsilon) + J(\theta - \epsilon)}{2 * \epsilon}$$

2.8 HYPERPARAMETER TUNING

Hyperparameter tuning technique aims to find a better combination of hyperparameters that leads to a better performance and less overfitting,weather those parameters are related to the network's architecture or the optimizer.

- Manual search : babysitting.
- random search.
- grid search.
- Bayesian optimization.

GRID SEARCH VS RANDOM SEARCH

Random Search	Grid Search
Doesn't guarantee to find the best hyperparameters	guarantees to find the best hyperparameters
Pick random points to try from the configuration space	tries all the possible combinations.
Good in high spaces	Curse of dimentiality
Good results in less iterations	computationally expensive.

Table 2: Grid search Vs Random Search

Grid search & Random search share a common downside : "each guess is independent from the previous one",solution : **Bayesian optimization**.

3 Chapter 03 : Convolutional neural networks

3.1 Foundations of CNNs

Convolutional neural networks are a type of deep learning architectures designed at first for computer vision tasks but later proved to work on other fields like text and voice processing.
A convolutional neural network mainly consists of three type of layers :

- Convolutional layers : for feature extraction.
- Pooling layers : reduce the size of the feature map.
- Fully connected layers : decision making.

3.1.1 Convolutional layer

A convolutional layer takes as an input a **feature map** of size : $W \times H \times C$,C is called the number of channels,and applies M different filters of dimensions $F_w \times F_h$ to produce a new feature map of dimenion : $W' \times H' \times M$.

The convolutional layer processes the input feature map using a moving window of size $F_w \times F_h$,the window moves by S steps,S is called the stride.

Optionally,before inputing the feature map to the convolutional layer a *padding* can be added by adding additional layers of pixels around the border of an image it can serve tow main purposes : **dimension preservation (same padding)** and **preventing information loss around the borders**.

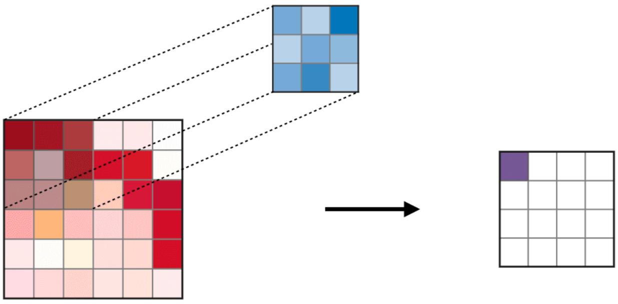


Figure 5: Convoluational layer in action

Calculating the size of the new feature map :

For simplification : $F_w = F_h = F$ and $W = H = N$ and $W' = H' = N'$.

$$N' = \frac{N - F + 2 * P}{S} + 1$$

Calculating the number of parameters :

$$F^2 \times C \times M + M$$

Example :

$$P = 1, S = 2, F = 3, C = C_{in} = 1, M = C_{out} = 1, N = 3$$

The size of the new feature map :

$$N' = \frac{3 - 3 + 2 * 1}{2} + 1 = 2$$

Numerical demonstration :

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 125 & 15 & 30 & 0 \\ 0 & 250 & 25 & 2 & 0 \\ 0 & 174 & 255 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$F = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$Y = X \odot F = \begin{bmatrix} -125 & -5 \\ -149 & -10 \end{bmatrix}$$

3.1.2 Pooling layer

Pooling layers help in reducing the dimensions of the feature maps while retaining the most important features.

- Max pooling : takes the maximum of the region.
- Average pooling : takes the average of the region.
- Global max pooling: takes the maximum value over the entire feature map.
- Global average pooling: takes the average value over the entire feature map.
- L2 pooling: takes the L2 norm of each region.
- Fractional max pooling: takes the maximum value over a randomly chosen subset of the region.

The dimensions of the resulted feature map is calculated the same way as the convolutional layer,because pooling layers also process the input feature map using a moving window.

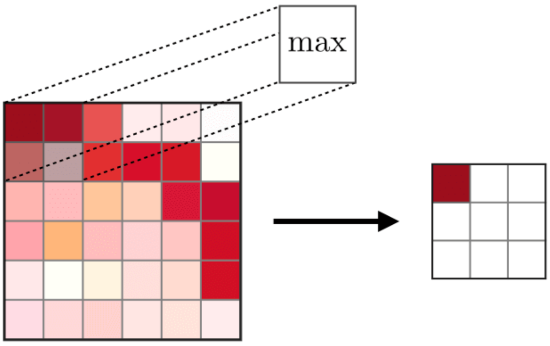


Figure 6: Max-Pooling layer in action

Example :

3.1.3 CNN vs ML

	CNN	ML
Data Requirement	Large amount of labeled data	small amount of data
Feature extraction	Automatic	Manual or using unsupervised algorithms
Training	requires computational resources	doesn't requires computational resources
Generalization	can generalize well on unseen data	may struggle on new data

Table 3: CNN vs ML