

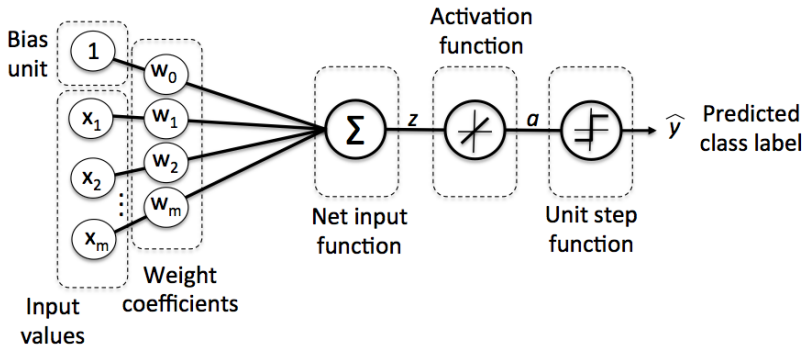
# Chapter 12

## Artificial Neural Networks

October 29, 2019

- Big in ML
- Set of algorithms to train neural networks
- Python libraries available
- Outline
  - Forward propagation in ANNs
  - Backpropagation to learn the parameters
  - Debugging ANNs
  - Alternative architectures (CNN, RNN)

# Single neuron review



- Perceptron

- Update all weights, then recompute  $\hat{y}$
- Weight update done after seeing each sample

$$\Delta w_j = \eta \left( y^{(i)} - \hat{y}^{(i)} \right) x_j^{(i)}$$

- Adaline

- Weight update done after entire training set has been seen
- In every epoch, update all weights as follows:

$$\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}, \quad \text{where } \Delta \mathbf{w} = -\eta \nabla J(\mathbf{w})$$

- I.e. compute the gradient based on all samples in the training set (this is known as batch gradient descent)
- SGD updates after seeing  $n$  samples
- Mini-batch: middle ground between SGD and batch GD

# Weight update details

Partial derivative for each weight  $w_j$  in the weight vector  $\mathbf{w}$ :

$$\frac{\partial}{\partial w_j} J(\mathbf{w}) = \sum_i (y^{(i)} - a^{(i)}) x_j^{(i)}$$

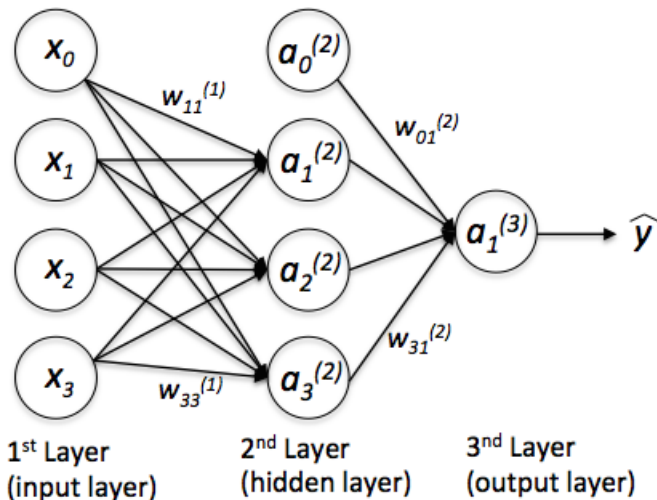
Here  $y^{(i)}$  is the target class label of a particular sample  $x^{(i)}$ , and  $a^{(i)}$  is the *activation* of the neuron, which is a linear function in the case of Adaline: Remember that we defined the *activation function*  $\phi(\cdot)$  as follows:

$$\phi(z) = z = a$$

Here, the net input  $z$  is a linear combination of the weights that are connecting the input to the output layer:

$$z = \sum_j w_j x_j = \mathbf{w}^T \mathbf{x}$$

# Multi-layer feedforward neural network



- We denote the  $i$ th activation unit in the  $l$ th layer as  $a_i^{(l)}$
- The activation units  $a_0^{(1)}$  and  $a_0^{(2)}$  are the *bias units*, respectively, which we set equal to 1
- The activation of the units in the input layer:

$$\mathbf{a}^{(i)} = \begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ \vdots \\ a_m^{(1)} \end{bmatrix} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix}$$

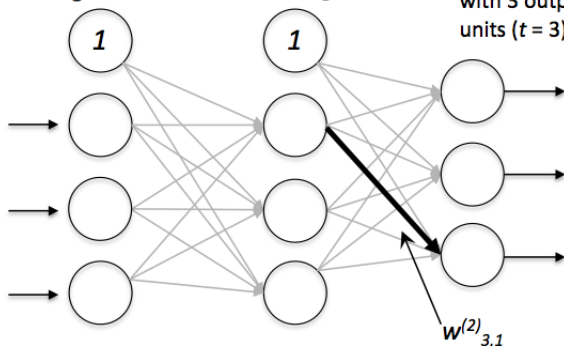
- The connection between the  $k$ th unit in layer  $l$  to the  $j$ th unit in layer  $l + 1$  written as  $w_{j,k}^{(l)}$

# Notation summary

layer  $l = 1$  with 3  
Input units ( $m = 3$ )  
not counting bias

layer  $l = 2$  with 3  
hidden units ( $h = 3$ )  
not counting bias

Layer  $l = 3$   
with 3 output  
units ( $t = 3$ )



Number of layers:  $L = 3$

connects 1<sup>st</sup> non-bias  
neuron in layer 2 to the 3<sup>rd</sup>  
unit layer 3



# MLP learning procedure

- 1 Starting at the input layer, forward propagate  $\mathbf{x}^{(i)}$
- 2 Calculate the error that we will want to minimize
- 3 Find its derivative with respect to each weight
- 4 Update the weights

# Forward propagation

- 1 Assume, input has  $m$  dimensions
- 2 Compute the net input  $a_1^{(2)}$  for unit 1 in the hidden layer:

$$z_1^{(2)} = a_0^{(1)} w_{1,0}^{(1)} + a_1^{(1)} w_{1,1}^{(1)} + \cdots + a_m^{(1)} w_{l,m}^{(1)}$$

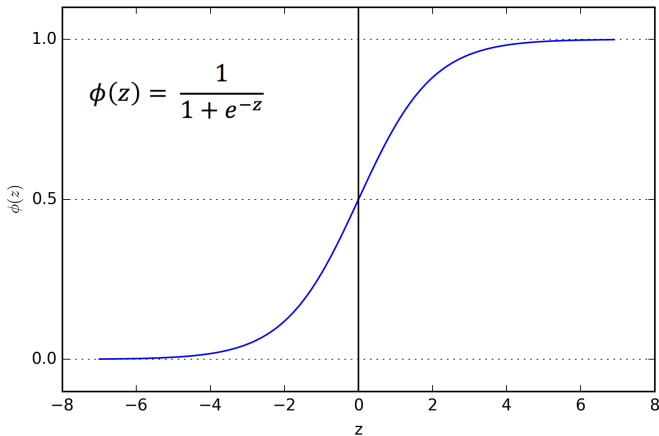
- 3 Compute the activation for unit 1 in the hidden layer:

$$a_1^{(2)} = \phi(z_1^{(2)})$$

- 4 Here  $\phi(\cdot)$  is the activation function
- 5 Logistic sigmoid is often used:

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

# Sigmoid function



# Vectorized notation

- Write activation in a matrix form
- Readability + more efficient code
- Net inputs for the hidden layer:

$$\mathbf{z}^{(2)} = \mathbf{W}^{(1)}\mathbf{a}^{(1)}$$

- Dimensions (ignoring bias units for simplicity)

$$[h \times 1] = [h \times m][m \times 1]$$

- Activations for the hidden layer:

$$\mathbf{a}^{(2)} = \phi(\mathbf{z}^{(2)})$$

# Matrix notation

- Generalize computation to all  $n$  samples in the training set

$$\mathbf{Z}^{(2)} = \mathbf{W}^{(1)} [\mathbf{A}^{(1)}]^T$$

- Matrix dimensions

$$[h \times n] = [h \times m][n \times m]^T$$

- Activation matrix

$$\mathbf{A}^{(2)} = \phi(\mathbf{Z}^{(2)})$$

- Now activation of the output layer

$$\mathbf{Z}^{(3)} = \mathbf{W}^{(2)} \mathbf{A}^{(2)}$$

- Matrix dimensions

$$[t \times n] = [t \times h][h \times n]$$

- Output of the network

$$\mathbf{A}^{(3)} = \phi(\mathbf{Z}^{(3)}), \mathbf{A}^{(3)} \in \mathbb{R}^{t \times n}.$$

# Cost function

The logistic Cost function is the same we used for logistic regression:

$$J(\mathbf{w}) = - \sum_{i=1}^n y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)})$$

Here,  $a^{(i)}$  is the sigmoid activation of the  $i$ th unit  $a^{(i)} = \phi(z^{(i)})$ .  
Regularization:

$$L2 = \lambda \|\mathbf{w}\|_2^2 = \lambda \sum_{j=1}^m w_j^2$$

$$J(\mathbf{w}) = - \left[ \sum_{i=1}^n y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)}) \right] + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

# Cost function for all units in output layer

The activation of the third layer and the target class could be:

$$\mathbf{a}^{(3)} = \begin{bmatrix} 0.1 \\ 0.9 \\ \vdots \\ 0.3 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

So, we need to generalize the logistic cost function to all activation units  $j$  in our network. The cost function (without the regularization term) becomes:

$$J(\mathbf{w}) = - \sum_{i=1}^n \sum_{j=1}^t y_j^{(i)} \log(a_j^{(i)}) + (1 - y_j^{(i)}) \log(1 - a_j^{(i)})$$

Superscript  $i$  is the index of a particular sample in training set

# Cost function for the entire network

Sum all the weights in the entire network in the regularization term:

$$J(\mathbf{w}) = - \left[ \sum_{i=1}^n \sum_{j=1}^m y_j^{(i)} \log \left( \phi \left( z_j^{(i)} \right) \right) + \left( 1 - y_j^{(i)} \right) \log \left( 1 - \phi \left( z_j^{(i)} \right) \right) \right] + \\ + \frac{\lambda}{2} \sum_{l=1}^{L-1} \sum_{i=1}^{u_l} \sum_{j=1}^{u_{l+1}} \left( w_{j,i}^{(l)} \right)^2$$

The following expression represents the L2-penalty term:

$$\frac{\lambda}{2} \sum_{l=1}^{L-1} \sum_{i=1}^{u_l} \sum_{j=1}^{u_{l+1}} \left( w_{j,i}^{(l)} \right)^2$$



# Minimizing the cost function

We want to minimize the cost function  $J(\mathbf{w})$ , so we calculate the partial derivative with respect to each weight for every layer in the network:

$$\frac{\partial J(\mathbf{W})}{\partial w_{j,i}^{(l)}}$$

▸ Deep Learning with Python GitHub

▸ IMDB classification example