Deep Learning

Stephan Robert

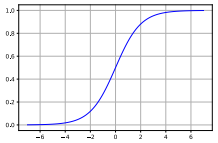
1

Content

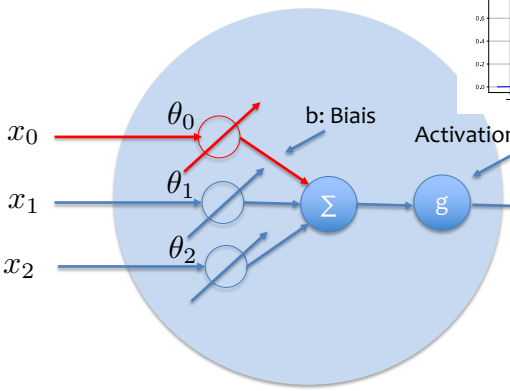
- Introduction to Neural Networks
- Backpropagation equation
- TensorFlow
- Hyperparameters tuning and regularization
- Convolutional Neural Networks (CNNs)
- Sequential Models: Recurrent Neural Networks
- LSTM
- Autoencoders
- GANs

2

A simple Neuron



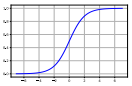
$$\frac{1}{1+e^{-\Theta^t \mathbf{x}}}$$

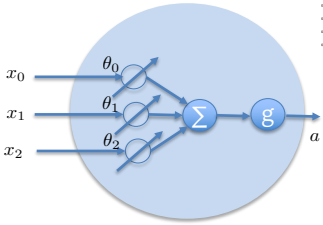


$$a = g(x_1 * \theta_1 + x_2 * \theta_2 + b)$$

3

Feedforward





$a = g(x_1 * \theta_1 + x_2 * \theta_2 + b)$

Numerical Example

$$x_0 = 1$$

$$\Theta^t \mathbf{x} = 4 + 0 * 3 + 0.5 * 5 = 6.5$$

$$a = g(6.5) = 0.998$$

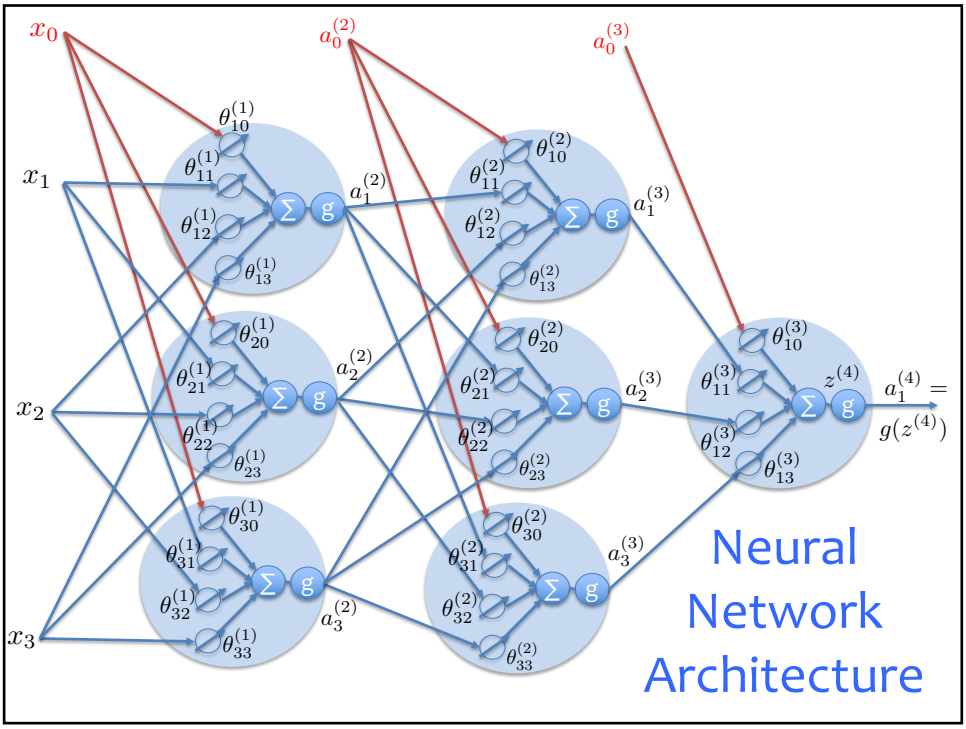
$$\mathbf{\Theta} = \begin{pmatrix} 4 \\ 0 \\ 0.5 \end{pmatrix} \quad \mathbf{x} = \begin{pmatrix} 1 \\ 3 \\ 5 \end{pmatrix}$$

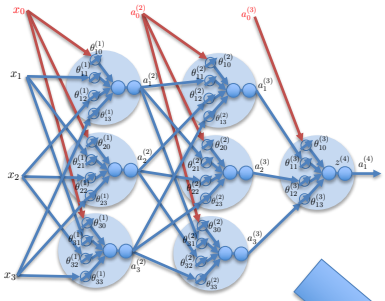
4

Python Code

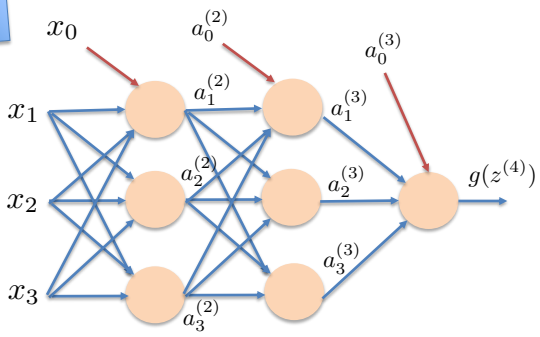
```
import numpy as np

def sigmoid(x):
  # Activation function: f(x) = 1 / (1 + e^(-x))
  return 1 / (1 + np.exp(-x))

class Neuron:
  def __init__(self, thetas, bias):
    self.thetas = thetas
    self.bias = bias

  def feedforward(self, inputs):
    total = np.dot(self. thetas, inputs) + self.bias
    return sigmoid(total)

thetas= np.array([0, 0.5])  # theta_1 = 0, theta_2 = 0.5
bias = 4                    # x0*theta_0=b = 4
n = Neuron(thetas, bias)

x = np.array([3, 5])        # x0=1, x1 = 3, x2 = 5
print(n.feedforward(x))     # 0.998 (V. Zhou, Princeton)
```

5

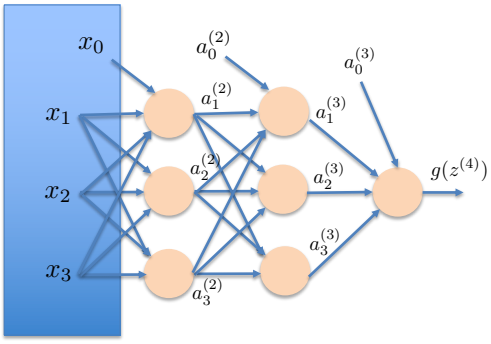




Simplified view of the Neural Network Architecture

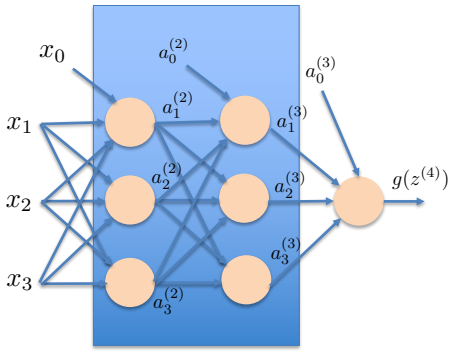


Terminology



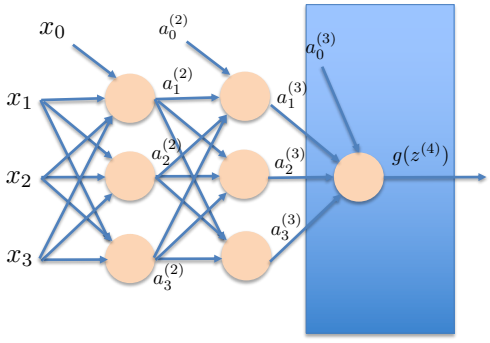
Input Layer
Layer 1

Terminology



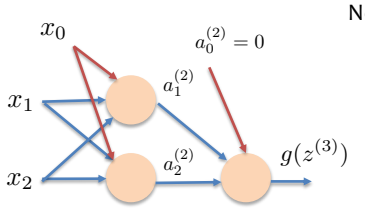
Hidden Layers
Layers 2 & 3

Terminology



Output Layer
Layer 4

A very simple Neural Network



Notice: every neuron has the same Θ

$$\Theta^t = (0 \;\; 0 \;\; 1) \qquad \mathbf{x} = \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix}$$

$$\Theta^t \mathbf{x} = 0 * 1 + 0 * 1 + 1 * 3 = 3$$

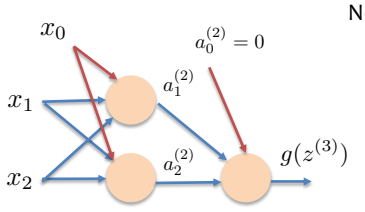
$$a_1^{(2)} = a_2^{(2)} = g(3) = 0.95$$

$$\Theta^t \mathbf{a}^{(2)} = 0 * 0 + 0 * 0.95 + 1 * 0.95 = 0.95$$

$$g\left(\Theta^t \mathbf{a}^{(2)}\right) = g\left(z^{(3)}\right) = g(0.95) = 0.72$$

11

A very simple Neural Network



Notice: every neuron has the same Θ

$$\Theta^t = (0 \;\; 0 \;\; 1) \qquad \mathbf{x} = \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix}$$

$$\Theta^t \mathbf{x} = 0 * 1 + 0 * 1 + 1 * 3 = 3$$

$$a_1^{(2)} = a_2^{(2)} = g(3) = 0.95$$

$$\Theta^t \mathbf{a}^{(2)} = 0 * 0 + 0 * 0.95 + 1 * 0.95 = 0.95$$

$$g\left(\Theta^t \mathbf{a}^{(2)}\right) = g\left(z^{(3)}\right) = g(0.95) = 0.72$$

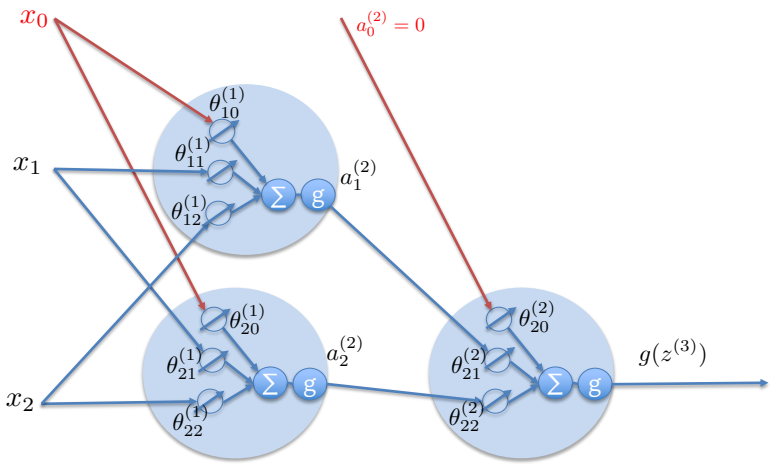
Notice:
we can add as **many layers** as we want,
and as **many neurons** per layer also…

12

Python code

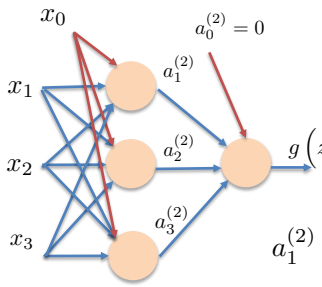
$\theta_0^{(1)} = \theta_0^{(2)} = \text{biais} = 0$

```
# ... code from previous slides here
class OurNeuralNetwork:
def __init__(self):
  thetas = np.array([0, 1])
  bias = 0

  # The Neuron class here is from the previous slides

  self.a1 = Neuron(thetas, bias)
  self.a2 = Neuron(thetas, bias)
  self.o1 = Neuron(thetas, bias)

 def feedforward(self, x):
  out_a1 = self.a1.feedforward(x)
  out_a2 = self.a2.feedforward(x)

  # The inputs for o1 are the outputs from a1 and a2
  out_o1 = self.o1.feedforward(np.array([out_a1, out_a2]))

  return out_o1

network = OurNeuralNetwork()
x = np.array([1, 3])
print(network.feedforward(x)) # 0.72 (V. Zhou, Princeton)
```



14

Output



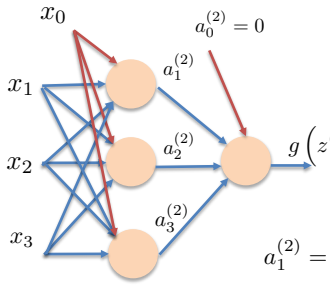
$$g\left(z^{(3)}\right) = a_1^{(3)} = h_{\Theta}(\mathbf{x})$$

$$a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right)$$
$$a_2^{(2)} = g\left(\theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3\right)$$
$$a_3^{(2)} = g\left(\theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3\right)$$

$$h_{\Theta}(\mathbf{x}) = a_1^{(3)} = g\left(\theta_{10}^{(2)}a_0^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)}\right)$$

15

Vectorization



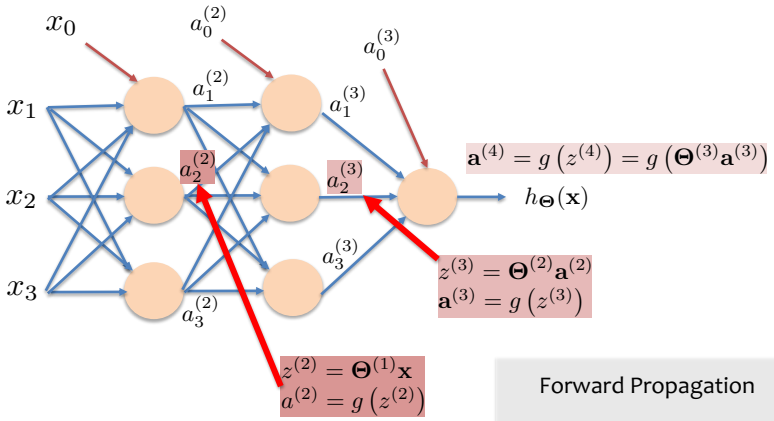
$$g\left(z^{(3)}\right) = a_1^{(3)} = h_{\Theta}(\mathbf{x})$$

$$a_1^{(2)} = g(\underbrace{\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3}_{z_1^{(2)}}) = g(z_1^{(2)})$$
$$a_2^{(2)} = g(\theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3) = g(z_2^{(2)})$$
$$a_3^{(2)} = g(\theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3) = g(z_3^{(2)})$$

$$h_{\Theta}(\mathbf{x}) = a_1^{(3)} = g\left(\theta_{10}^{(2)}a_0^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)}\right)$$

16

Vectorization (2)

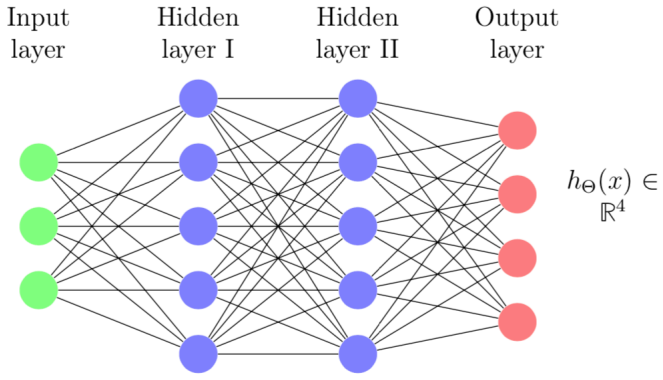


Forward Propagation

$$h_{\mathbf{\Theta}}(\mathbf{x}) = g\left(\sum_i \mathbf{\Theta}_i^{(j)}\mathbf{a}_i^{(j)}\right)$$

17

Multiclass Classification



18

Loss Function

Remember lesson 1:

$$\min_{\theta_0,\theta_1} J(\theta_0,\theta_1) = \min_{\theta_0,\theta_1} \frac{1}{2m}\sum_{i=1}^{m}(f(x_i)-y_i)^2$$

Function to minimize: J

What we can tune: $\boldsymbol{\Theta}$

What is given: $\mathbf{x}$

Idea: $\theta_{11}^{(1)} = \theta_{11}^{(1)} - \eta\frac{\partial J}{\partial \theta_{11}^{(1)}}$

Problem: how to compute $\frac{\partial J}{\partial \theta_{11}^{(1)}}$?

19

Loss Function

Trick: Chain rule

$$\frac{\partial J}{\partial \theta_{11}^{(1)}} = \left(\frac{\partial J}{\partial y_p}\right)\left(\frac{\partial y_p}{\partial a_1^{(2)}}\right)\left(\frac{\partial a_1^{(2)}}{\partial \theta_{11}^{(1)}}\right)$$

Notice: it is working backwards: known as the

Backpropagation

20

Training set

x_0	x_1	x_2	x_3	x_4	y
	Size	#bedrooms	#floors	Age	Price (mios)
1	125	3	2	20	0.8
1	220	5	2	15	1.2
1	400	7	3	5	4
1	250	4	2	10	2

21

Backpropagation

Training set $\left\{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \ldots, \left(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}\right)\right\} \equiv \{(\mathbf{x})_1, (\mathbf{y})_1), \ldots, ((\mathbf{x})_m, (\mathbf{y})_m)\}$

$\{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\}$

Cost function

$\mathbf{J}(\boldsymbol{\Theta})$

To compute

$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\boldsymbol{\Theta})$ Node i, input j, layer l

Let

$\delta_j^l =$ error of the node j of the layer l

Example, L=4

$\delta_j^4 = a_j^{(4)} - y_j = (h_{\boldsymbol{\Theta}}(\mathbf{x}))_j - y_j$ or (vectorization) $\delta^{(4)} = \mathbf{a}^{(4)} - \mathbf{y}$

22

Backpropagation

Example, L=4

$$\delta^{(4)} = \mathbf{a}^{(4)} - \mathbf{y}$$

$$\delta^{(3)} = \left(\boldsymbol{\Theta}^{(3)}\right)^t \delta^{(4)}.g'\left(z^{(3)}\right)$$

$$\delta^{(2)} = \left(\boldsymbol{\Theta}^{(2)}\right)^t \delta^{(3)}.g'\left(z^{(2)}\right)$$

No $\delta^{(1)}$

$$\delta_j^4 = a_j^{(4)} - y_j = (h_{\boldsymbol{\Theta}}(\mathbf{x}))_j - y_j$$

$$\frac{\partial}{\partial \theta_{ij}^l} J(\Theta) = a_j^{(l)} \delta_i^{l+1}$$

23

Algorithm

procedure

$\Delta_{ij}^{(l)} \leftarrow 0$ ▷ $\forall i, j, l$, use to compute $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\boldsymbol{\Theta})$

for $i = 1$ to m **do**

$\mathbf{a}^{(1)} = \mathbf{x}^{(i)}$ ▷ Input value

Perform forward propagation to compute $\mathbf{a}^{(l)}$ for $l = 2, ..., L$

Using $\mathbf{y}^{(i)}$, compute $\delta^{(L)} = \mathbf{a}^{(L)} - \mathbf{y}^{(i)}$ ▷ Compute the last error term

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ ▷ Vectorized version: $\boldsymbol{\Delta}^{(l)} = \boldsymbol{\Delta}^{(l)} + \delta^{(l)}(\mathbf{a}^{(l)})^t$

if $j \neq 0$ **then**

$D_{ij}^{(l)} \leftarrow \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\theta_{ij}^{(l)}$

else

$D_{ij}^{(l)} \leftarrow \frac{1}{m}\Delta_{ij}^{(l)}$ ▷ Biais term

return $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$.

24