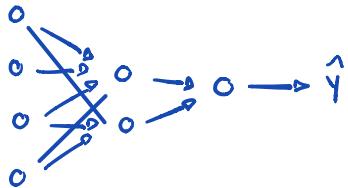


Charlie Mackie  
May 22nd 2020

## Neural Networks and Deep Learning

### Basic Neural Architecture



- ⊗ if all of the nodes are connected between two layers  $[L-1]$  and  $[L]$   
then these layers are 'deeply connected'

### Supervised Learning:

- unstructured Data [ Sequences (RNNs): audio  $\rightarrow$  transcript  
Images (CNNs / convnets): object detection ]  
structured [ Structured Data (Regression / ANNs): House \$ given — ]

### unsupervised learning:

given a dataset without any explicit instructions, find structure in data

Clustering: look for training data that are similar & group them

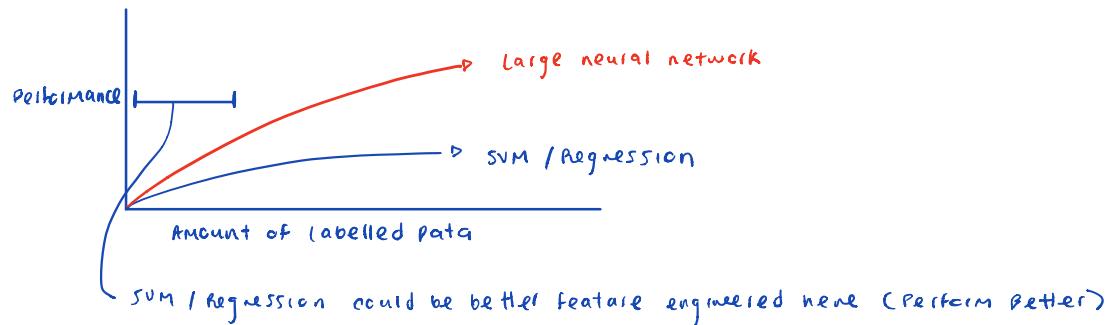
Anomaly detection: flag outliers in data (ex: credit card fraud)

Association: similarity metrics in data

### Reinforcement learning:

Iterative process of rewarding an agent (network) for good behaviour. Agent relies on past data and exploration of new tactics.

### General Network Performance:



- ⊗ with lots of data available, we use Neural Nets

### Current State/Progress of NN's:

- ① Data: always getting more/better data
- ② Computation: more powerful compute = more tuning (iterative process)
- ③ Algorithms: ~~sigmoid~~ → ReLU

### Logistic Regression as a Neural Network

Training Data:  $M = M_{\text{train}}$

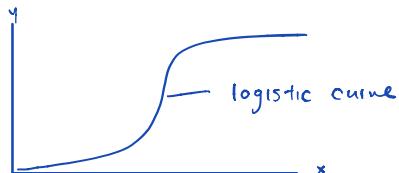
$$X = \begin{bmatrix} x^{(1)} & x^{(2)} & \dots & x^{(n)} \end{bmatrix}^T \quad (n_x, m)$$

$\downarrow \downarrow \downarrow \downarrow$

$m$

$$Y = [y^{(1)}, y^{(2)}, \dots, y^{(n)}] \quad (1, m)$$

### Logistic Regression (Binary Classification)



\* How can we fit this curve to data to classify it as either 1 or 0?

### Training

$$\Theta = [w, b] = \begin{bmatrix} \Theta_0 \\ \Theta_1 \\ \vdots \\ \Theta_n \end{bmatrix} \quad \begin{array}{l} \text{bias} \\ \text{weights} \end{array}$$

we want to multiply all  $x$  by a weight  $w$  and add a bias  $b$ .  
↳ use  $(w^T x)$

\* use  $\hat{y} = \sigma(w^T x + b)$  where sigmoid =  $\frac{1}{1 + e^{-x}}$

↑  
Sigmoid  
↑ weight  
↑ data  
↑ bias

• If  $x$  is small, sigmoid is  $\rightarrow 0$   
•  $x$  is large, sigmoid  $\rightarrow 1$

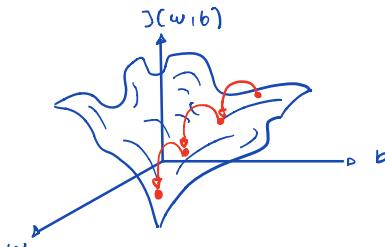
↳ This will determine if  $y$  is 1 or 0  
which we can check with the training labels

## Loss function

$$\left. \begin{aligned} L(\hat{y}, y) &= \frac{1}{2} (\hat{y} - y)^2 \quad (\text{Squared Error}) \\ \star J(\hat{y}, y) &= -[y \log \hat{y} + (1-y) \log(1-\hat{y})] \end{aligned} \right\} \text{loss function examples}$$

↓  
sum the loss function over all data

$$\text{Cost Function} = J(w, b) = \frac{1}{m} \sum_i J(\hat{y}^{(i)}, y^{(i)})$$



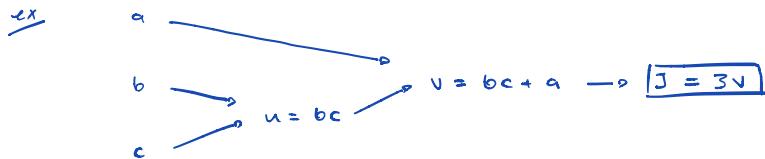
④ we want to iterate through our training data and 'minimize' our cost function

Repeat:  $w := w - \alpha \frac{\partial J(w, b)}{\partial w}$  learning rate ex: 0.01

④ update the weights until a 'global optimum' is reached

$$\frac{\partial J(w, b)}{\partial w} = \text{slope of } J(w, b) \text{ in the direction of } w$$

Computation Graph → how derivatives are calculated given select equations & values



④ applied to a our neural network

Our goal is to find  $\frac{\partial L}{\partial w_1}$  &  $\frac{\partial L}{\partial w_2}$  so we can optimize

Parameters	Intermediate values	Loss of Example
$x_1, w_1, x_2, w_2, b$	$z = w_1 x_1 + w_2 x_2 + b \rightarrow a = \sigma(z) \rightarrow L(a, y)$	
	$\frac{\partial L(a, y)}{\partial z} = \frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z}$	$\frac{\partial L(a, y)}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a}$
	$= a(1-a)$	
	$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w_1}$	

Gradient Descent ② with a basic  $\sigma \rightarrow \sigma \rightarrow \hat{y}$  structure

$$\text{minimizing functions: } J(w, b) = \frac{1}{m} \sum_i J(a^{(i)}, y)$$

$$a^{(i)} = \hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$$

for  $m$  in training examples:

$$z^{(i)} = w^T \cdot x^{(i)} + b$$

$$a^{(i)} = \text{sigmoid}(z^{(i)})$$

$$J_{+} = -[-y^{(i)} \log(a^{(i)}) + (1-y^{(i)}) \log(1-a^{(i)})] \quad (\text{loss function})$$

$$\delta z^{(i)} = a^{(i)} - y^{(i)}$$

$$\delta w^{(i)} += x^{(i)} \cdot \delta z$$

$$\delta b^{(i)} += \delta z^{(i)}$$

$$J_{+} / = m$$

$$\delta w^{(i)} / = m$$

Vectorization (for loops have time/space complexity ... use vectors)

ex  $z = np.\text{dot}(w.T, x) + b$

$$z = np.\text{dot}(w.T, x) + B$$

$$A = \sigma(z)$$

$$\delta z = A - Y$$

$$\delta w = \frac{1}{m} X \delta z^T$$

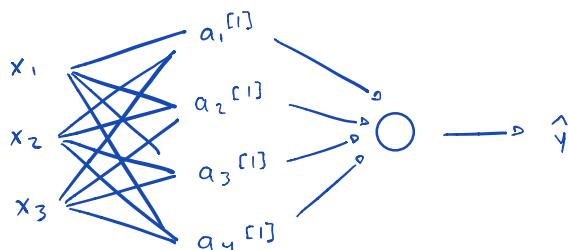
$$\delta b = np.\text{sum}(\delta z)$$

Summary of a LR Neural Network:

- Model structure / Data
- Initialize parameters ( $w, b$ )
- Loop over: current loss  
current Grad  
update parameters

### Shallow Neural Network

A Shallow Neural Network has multiple layers of nodes:



Compute forward output of Shallow Neural Network:  
 (vectorized implementation)

①

$$z^{[1]} = \underbrace{w^{[1]} a^{[0]} + b^{[1]}}_{\text{② weights for inputs} \rightarrow L_1}$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}_{(n,1)} \cdot \begin{bmatrix} w_1 & \rightarrow \\ w_2 & \rightarrow \\ w_3 & \rightarrow \\ w_4 & \rightarrow \end{bmatrix}_{(n,4)} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}_{(4,1)} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}_{(n,1)}$$

②

$$a^{[1]} = \sigma(z^{[1]})$$

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}_{(n,1)} = \frac{1}{1+e^{-x}} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}_{(n,1)}$$

③

$$z^{[2]} = \underbrace{w^{[2]} \cdot a^{[1]} + b^{[2]}}_{\text{④ weight matrix dims are used to achieve desired output size}}$$

$$\begin{bmatrix} z \end{bmatrix}_{(1,1)} = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 \end{bmatrix}_{(1,4)} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}_{(4,1)} + \begin{bmatrix} b \end{bmatrix}_{(1,1)}$$

④

$$a^{[2]} = \sigma(z^{[2]}) = \hat{y}$$

$$\begin{bmatrix} a \end{bmatrix}_{(1,1)} = \sigma \begin{bmatrix} z \end{bmatrix}_{(1,1)} = \hat{y}$$

\* we just vectorized across all weights / bias , also vectorize across the number of examples.

{

For  $i = 1$  to  $m$  (training Examples):

$$\begin{aligned} z^{[1](i)} &= w^{[1]} x^{(i)} + b^{[1]} \quad \text{⑤ Notice how these} \\ a^{[1](i)} &= \sigma(z^{[1](i)}) \quad \text{params } (w, b) \text{ are} \\ z^{[2](i)} &= w^{[2]} a^{[1](i)} + b^{[2]} \quad \text{consistent across Tx,} \\ a^{[2](i)} &= \sigma(z^{[2](i)}) \quad \text{changed with GD.} \end{aligned}$$

\* This is vectorized (no more 'For' loop)

$$z^{[1]} = w^{[1]} \cdot x + b^{[1]}$$

$$\downarrow \qquad \downarrow$$

$$\begin{array}{l} ([L+1], m) \\ \text{⑥ (hidden units, Tx)} \end{array} \qquad \begin{array}{l} (n_x, m) \\ \text{⑦ (inputs, Training examples)} \end{array}$$

NOW ...

$$z^{[1]} = w^T \cdot x + b, \quad a^{[1]} = \sigma(z^{[1]})$$

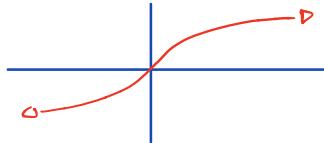
$\begin{bmatrix} 1 & x^{(1)} & x^{(2)} & \dots & x^{(m)} \end{bmatrix}$        $\begin{bmatrix} 1 & a^{[1](1)} & a^{[1](2)} & \dots & a^{[1](m)} \end{bmatrix}$

Activation Function - at this point, we can try to use some better activation functions

activation functions add complexity ... we need them to be non-linear for hidden layers to be effective

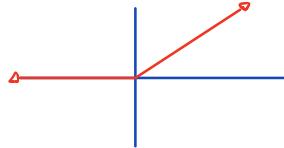
### ① tanh

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



### ② Relu

$$\text{Relu}(z) = \max(0, z)$$



Tends to learn faster because of the gradient of Relu.

In tanh or sigmoid, gradient get very small given large values

## Forward Propagation through our shallow NN

### Forward Prop:

$$\begin{aligned} z^{[1]} &= w^{[1]} X + b^{[1]} \\ a^{[1]} &= g^{[1]}(z^{[1]}) \\ z^{[2]} &= w^{[2]} a^{[1]} + b^{[2]} \\ a^{[2]} &= g^{[2]}(z^{[2]}) \end{aligned}$$

★  $g^{(x)}$  = activation function chosen for that layer

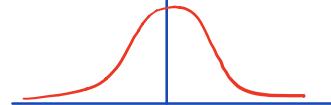
### Backward Prop:

$$\begin{aligned} dz^{[2]} &= A^{[2]} - y \\ dw^{[2]} &= \left(\frac{1}{m}\right) dz^{[2]} A^{[1]T} \\ db^{[2]} &= \left(\frac{1}{m}\right) \text{np.sum}(dz^{[2]}, \text{axis}=1, \text{keepdims}) \end{aligned}$$

$$\begin{aligned} dz^{[1]} &= (w^{[2]T} \cdot dz^{[2]} * g^{[1]'} \cdot z^{[1]}) \\ dw^{[1]} &= \left(\frac{1}{m}\right) dz^{[1]} \cdot X^T \end{aligned}$$

$$db^{[1]} = \left(\frac{1}{m}\right) \text{np.sum}(dz^{[1]})$$

### Random Initialization ( $w, b$ )

- ④ when we initialize with zeros the activation gradients are low which impedes learning speed
- ⑤  $w^{[1]} = np.random.randn(2, 2) * 0.01$   
 $b^{[1]} = np.zeros(2, 1)$       
  
↳ keeps derivatives high and learning fast

`np.random.randn`,  
returns numbers from a  
normal distribution.

- ⑥ high chance of numbers close to 0

**Deep Neural Network** → every hidden layer serves a purpose, deeper layers compute more complex features

Matrix Dimensions:       $w^{[L]} = (n^{[L]}, n^{[L-1]})$       # of hidden units layer L  
 $b^{[L]} = (n^{[L]}, 1)$

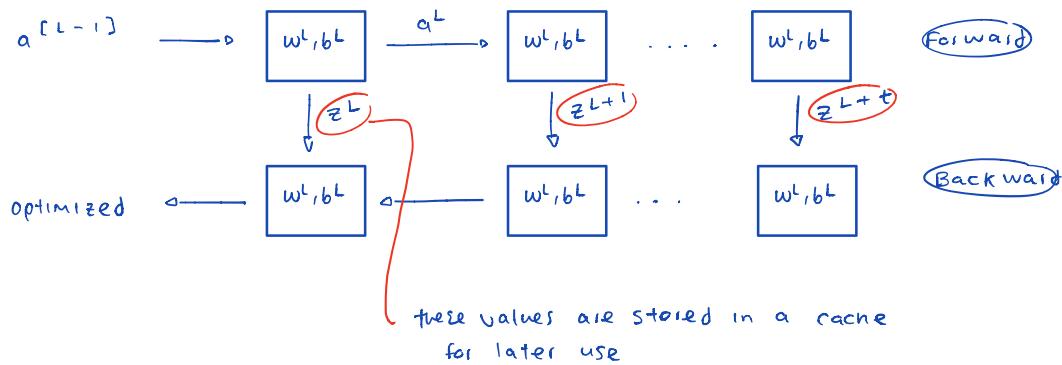
↳ vectorized across 'm' examples:

$$z^{[1]} = w^{[1]} X + b^{[1]}$$

$\downarrow \quad \downarrow \quad \downarrow$

$$(n^{[1]}, m) \quad (n^{[1]}, n^{[0]}) \quad (n^{[0]}, m)$$

### Process Visualization (for coding purposes)



\*  $[w, b]$  are parameters, LR  $\alpha$ , num iterations, hidden units/layers  
choice of activation are Hyperparameters