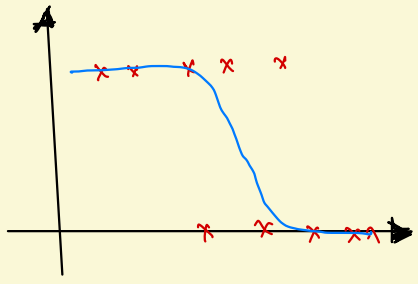
Topics covered

Source:
NPTEL: Mitesh Khapra
Deep Learning.
coursera Andrew NG
CodingCrave - Youtube
towards data science - blogs

1. Neural Network - Introduction & Analogies
2. Representations & Notations involved
3. Feed Forward & Back propagation derivation (optional) and Algorithm.
4. Activation functions
5. Example 1: Implementing XOR gate with a Neural Network.
6. Example 2: MNIST dataset

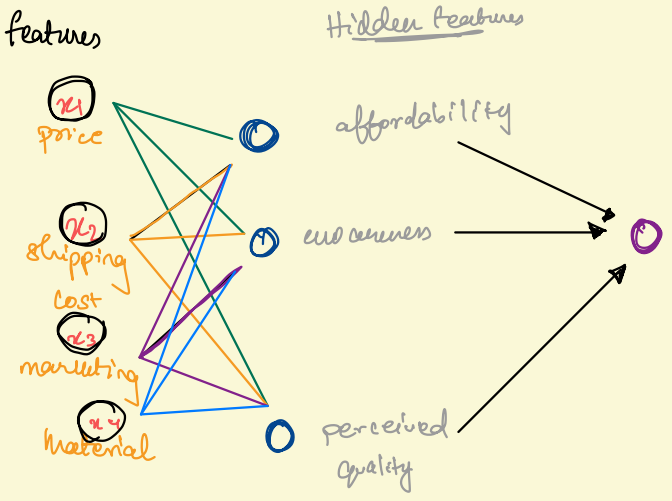
- **Neural Networks:** Algorithms that involves a network of neurons & trains on data to find the correct set of parameters to predict the correct o/p.

- **Demand Prediction:**



x = price — input

$a = f(x) = \frac{1}{1 + e^{-(wx+b)}}$ — output



L1:
- Inputs 4 numbers
- O/ps 3 numbers

L2:
- I/ps 3 no.s
- o/p 1

Speciality of NN:

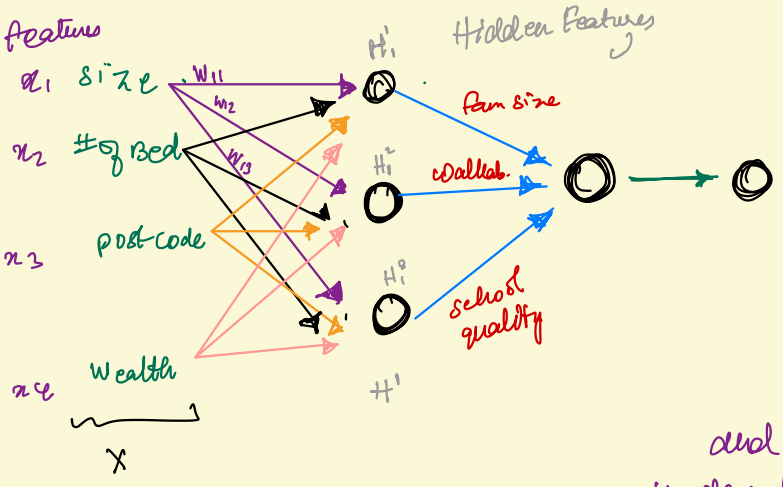
One of nice prop. of NN is when we train NN from data, we don't have to explicitly decide what other features such as affordability, awareness & so on. The NN figures out all by itself based on training data

A neural network as it trains automatically extracts the hidden features.

Source: Coursera

★ Housing Price Prediction

Price based on size, # of bedrooms, postal code.



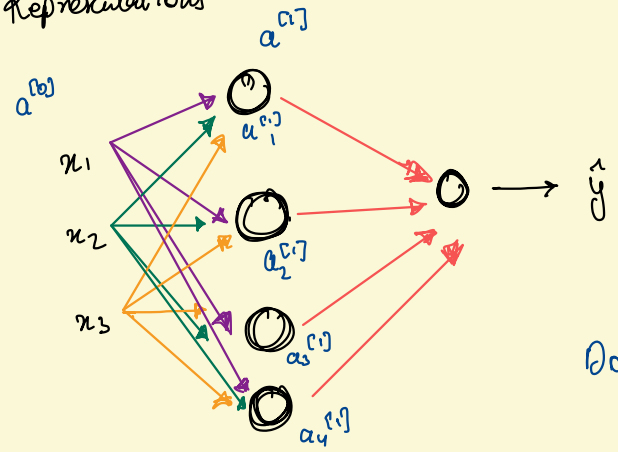
Instead of user saying that the feature family size will depend on feature x_1 & x_2, we provide all 4 features to family size for the neural n/w to decide on which feature it will depend upon.

And how does it decide which features will it depend on? (By understanding & observing from training set (It decides what weights w_{11}, w_{12}, w_{13} are!)

I/p layer connected with every hidden layer.

We can have another hidden layer (who knows what else parameters it might depend on like direction of the house, # of balcony etc etc)

★ Neural N/w Representations

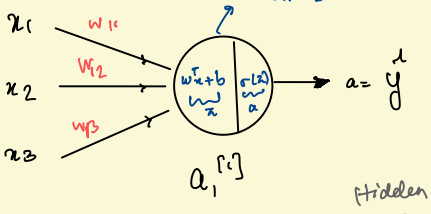


a = activations

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix}$$

2 Layer NN

Don't count the i/p layer



Layer number

$z_1^{[1]} = w_1^{[1]} x + b_1^{[1]}$

$a_1^{[1]} = \sigma(z_1^{[1]})$

$w_1^{[1]} = [w_{11}\ w_{12}\ w_{13}]$

Hidden Node: $z = w^T x + b$, $a = \sigma(z)$

Parallelly,

$z_2^{[1]} = w_2^{[1]} x + b_2^{[1]}$

$a_2^{[1]} = \sigma(z_2^{[1]})$

$z_3^{[1]} = w_3^{[1]} x + b_3^{[1]}$

$a_3^{[1]} = \sigma(z_3^{[1]})$

↳ **Vectorization :**

$$z_1^{[1]} = w_1^{[1]} x + b_1^{[1]}$$
$$z_2^{[1]} = w_2^{[1]} x + b_2^{[1]}$$
$$z_3^{[1]} = w_3^{[1]} x + b_3^{[1]}$$
$$z_4^{[1]} = w_4^{[1]} x + b_4^{[1]}$$

For four nodes in the hidden layer.

$$z^{[1]}$$

$$\begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} & w_{13}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} & w_{23}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} & w_{33}^{[1]} \\ w_{41}^{[1]} & w_{42}^{[1]} & w_{43}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]} x + b_1^{[1]} \\ w_2^{[1]} x + b_2^{[1]} \\ w_3^{[1]} x + b_3^{[1]} \\ w_4^{[1]} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

$$a_1^{[1]} = \sigma(z_1^{[1]}) \qquad a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

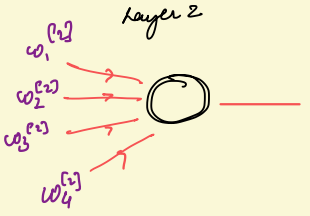
$$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$$
$$(4,1) \quad (4,3) \quad (3,1) \quad (4,1)$$

($a^{[0]} = x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$)

↓ 4 nodes in hidden layer each with 3 inputs

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$(1,1) \quad (1,4) \quad (4,1) \quad (1,1)$$

↓ 1 node in the present layer with 4 inputs

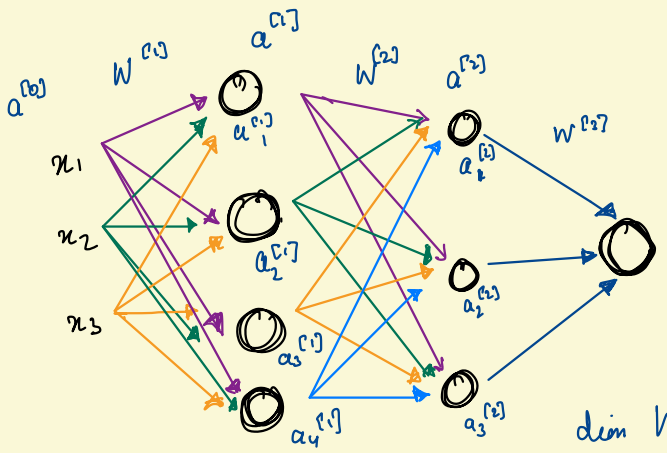


Each $w_1^{[2]}$, $w_2^{[2]}$, $w_3^{[2]}$, $w_4^{[2]}$ is a scalar.

$$W^{[2]} = [w_1^{[2]} \; w_2^{[2]} \; w_3^{[2]} \; w_4^{[2]}]$$

$\therefore$ $a_j^{[l]}$ = j^{th} o/p of l^{th} layer

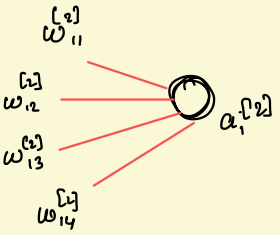
$$a_j^{[l]} = \sigma(z_j^{[l]}) = \sigma\left[\vec{w}_j^{[l]} \cdot \vec{a}^{[l-1]} + b_j^{[l]}\right]$$



$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

Aim: To find $a^{[2]}$

$\dim W^{[2]} = [\#\text{ of neurons}, \#\text{ of inputs}) = (3, 4)$



$$a_1^{[2]} = \sigma(z_1^{[2]})$$

$$z_1^{[2]} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} & w_{13}^{[2]} & w_{14}^{[2]} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} + b_1^{[2]}$$

$$w_1^{[2]} = \begin{bmatrix} w_{11}^{[2]} \\ w_{12}^{[2]} \\ w_{13}^{[2]} \\ w_{14}^{[2]} \end{bmatrix}$$

$$\therefore z_1^{[2]} = w_1^{[2]T} a^{[1]} + b_1^{[2]}$$

$$z_2^{[2]} = w_2^{[2]T} a^{[1]} + b_2^{[2]} \qquad z_2^{[2]} = w_2^{[2]T} a^{[1]} + b_3^{[2]}$$

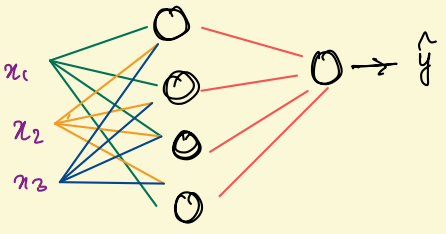
$$z^{[2]} = \begin{bmatrix} z_1^{[2]} \\ z_2^{[2]} \\ z_3^{[2]} \end{bmatrix} = \begin{bmatrix} - w_1^{[2]} - \\ - w_2^{[2]} - \\ - w_3^{[2]} - \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} + \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \\ b_3^{[2]} \end{bmatrix} = \underbrace{\begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} & w_{13}^{[2]} & w_{14}^{[2]} \\ w_{21}^{[2]} & w_{22}^{[2]} & w_{23}^{[2]} & w_{24}^{[2]} \\ w_{31}^{[2]} & w_{32}^{[2]} & w_{33}^{[2]} & w_{34}^{[2]} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}}_{w^{[2]} a^{[1]} + b^{[2]}} + \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \\ b_3^{[2]} \end{bmatrix}$$

$$z^{[2]} = \underset{(3,4)}{w^{[2]}} a^{[1]} + b^{[2]}$$

4 ips, 3 layers = inputs x # of layer

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

* VECTORIZATION



$$x^{(1)} = \begin{bmatrix} x^{11} \\ x^{12} \\ x^{13} \end{bmatrix}$$

Training examples

$x \longrightarrow a^{[2]} = \hat{y}$

$x^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$

$x^{(2)} \longrightarrow a^{[2](2)} = \hat{y}^{(2)}$

for $i = 1$ to m: (m = # of samples)

$$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

OR

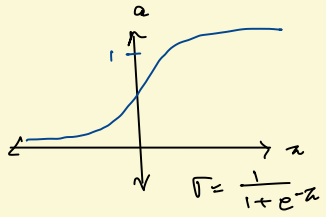
$$Z^{[1]} = W^{[1]} X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

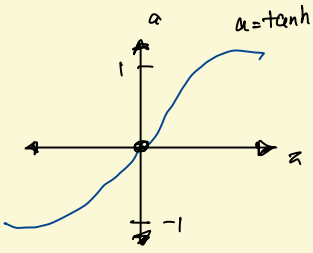
$X = A^{[0]}$ $\qquad A^{[0]} = \begin{bmatrix} | & | & | & & | \\ a^{[0](1)} & a^{[0](2)} & a^{[0](3)} \cdots & a^{[0](m)} \\ | & | & | & & | \end{bmatrix}$ ← Samples

$A^{[1]} = \begin{bmatrix} | & | & | & | \\ a^{[1](1)} & a^{[1](2)} & a^{[1](3)} & a^{[1](4)} \\ | & | & | & | \end{bmatrix}$

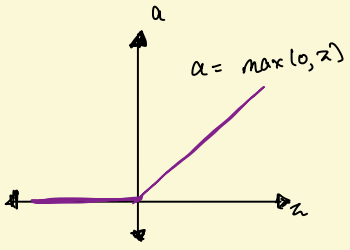
$X = \begin{bmatrix} | & | & | & | & & | \\ x^{(1)} & x^{(2)} & x^{(3)} & x^{(4)} \cdots & x^{(m)} \\ | & | & | & | & & | \end{bmatrix}$

* Activation Functions

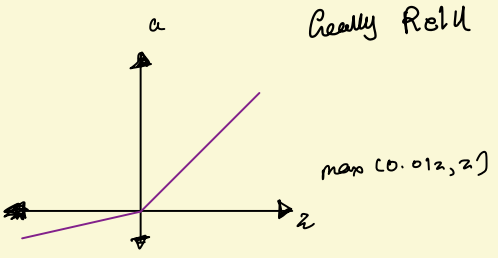




Slope is almost zero at +1 & -1 and increases sharply @ around zero.



"Rectified linear unit"



Leaky ReLU

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

$g(z) = z$ "linear activation function"

Aim: To compute weights and biases

Neural Network Classification:

Binary Classification

i) o/p y is 0 or 1

eg

Check if digit if 5 or 6

o/p: o/p = 0 it's 5

o/p = 1 it's 6

multiclass Classification.

i) o/p $y \in \mathbb{R}^K$

eg

Classify digits from 0 to 9

o/p = $\begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{matrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

Digit is 1

In summary NN:

- Data : $\{x_i, y_i\}_{i=1}^{N}$
- Model: $\hat{y}_i = f(x_i) = O\big(W^{[3]}\sigma\big(W^{[2]}\sigma\big(W^{[1]}x + b^{[1]}\big) + b^{[2]}\big) + b^{[3]}\big)$

  O = some output func. acting on $W^{[3]}A^{[2]} + b^{[3]}$
- Parameters : $\theta = W^{[1]}, W^{[2]}, \ldots W^{[n]}, b^{[1]}, b^{[2]} \ldots, b^{[n]}$

  n = # of hidden layers + 1
- Algorithm: Gradient descent with Back propagation.

Aim: To find an Algorithm for learning parameters of the above model.

★ Now that we have managed to find an o/p with the given initial weights and biases, How do I improve upon the weights?

With the given weights we have predicted the o/p. comparing it with the original o/ps, we need to find weights & biases s.t the error is minimum

★ Backpropagation: Algorithm used to improve the weights & biases using Gradient descent to minimize the errors.

Gradient descent:

$t = 0$

max_iter = 1000

repeat

{

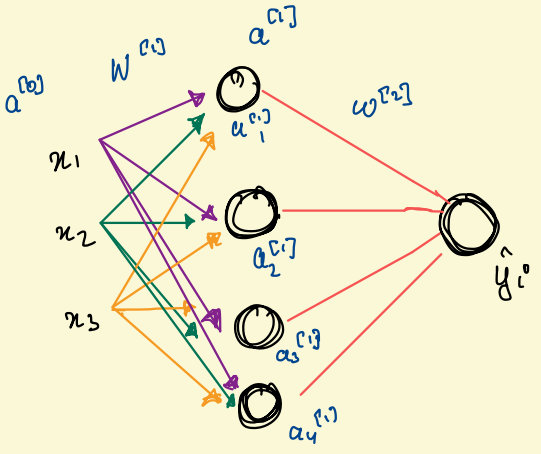
calculate $\frac{\partial J}{\partial w_i}, \frac{\partial J}{\partial b}$

$w_i = w_i - \eta \frac{\partial J}{\partial w_i} \quad \forall i$

$b_i = b_i - \eta \frac{\partial J}{\partial b}$

}

Strategy: Model → loss function → Find Parameters using Grad Descent



$$W^{[1]} = \begin{bmatrix} | & | & | \\ w_1^{[1]} & w_2^{[1]} & w_3^{[1]} \\ | & | & | \end{bmatrix}$$

$$\begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33} \\ w_{14} & w_{24} & w_{34} \end{bmatrix}$$

We have seen we choose loss functions based on the problem at hand.

★ BACKPROPAGATION using GD:

Cross / Cost func :

Cost $C = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}_i, y_i)$ where $\hat{y}_i = \sigma(W^{[l]} A^{[l-1]} + b^{[l]})$

L = Loss function

$w = w - \alpha \frac{\partial C}{\partial w}$

$b = b - \alpha \frac{\partial C}{\partial b}$

Minimize C
w, b

Aim : To find $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ in each layer. to update the weights in each layer.

BackPropagation :

Layer 2

$z = W^{[2]} A^{[1]} + b$ $\quad A = \sigma(z)$ $\quad \frac{\partial z}{\partial W^{[2]}} = A^{[1]}$

$C = \frac{1}{m} \sum_{i=1}^{m} L(y_i, A_i^{[2]})$

$\frac{\partial C}{\partial A^{[2]}}$ is given.

$\frac{\partial C}{\partial z^{[2]}} = \frac{\partial C}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial z^{[2]}}$

- $\frac{\partial C}{\partial W^{[2]}} = \frac{\partial C}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial w^{[2]}} = \frac{\partial C}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial z^{[2]}} A^{[1]^T} = \frac{\partial C}{\partial z^{[2]}} A^{[1]^T}$
- $\frac{\partial C}{\partial b^{[2]}} = \frac{\partial C}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial z^{[2]}} \frac{\partial z}{\partial b^{[2]}} = \frac{\partial C}{\partial z^{[2]}}$

$\therefore \quad W^{[2]} = W^{[2]} - \alpha \frac{\partial C}{\partial W^{[2]}} = W^{[2]} - \alpha \frac{\partial C}{\partial z^{[2]}} A^{[1]^T}$ # Updation of $W^{[2]}$

$b^{[2]} = b^{[2]} - \alpha \frac{\partial C}{\partial b^{[2]}} = b^{[2]} - \alpha \frac{\partial C}{\partial z^{[2]}}$

Layer 2

$$Z^{[1]} = W^{[1]}X + b^{[1]} \qquad Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

$$\frac{\partial C}{\partial A^{[1]}} = \frac{\partial C}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial Z^{[2]}} \frac{\partial Z^{[2]}}{\partial A^{[1]}} = (W^{[2]})^T \frac{\partial C}{\partial Z^{[2]}} \qquad \frac{\partial C}{\partial Z^{[1]}} = \frac{\partial C}{\partial A^{[1]}} \cdot \sigma'(Z^{[1]})$$

$$\frac{\partial C}{\partial W^{[1]}} = \frac{\partial C}{\partial A^{[1]}} \frac{\partial A^{[1]}}{\partial Z^{[1]}} \frac{\partial Z^{[1]}}{\partial W^{[1]}} = \frac{\partial C}{\partial Z^{[1]}} A^{[0]}$$

$$\frac{\partial C}{\partial b^{[1]}} = \frac{\partial C}{\partial Z^{[1]}} \frac{\partial Z^{[1]}}{\partial b^{[1]}} = \frac{\partial C}{\partial Z^{[1]}}$$

$$W^{[1]} = W^{[1]} - \alpha \frac{\partial C}{\partial W^{[1]}} = W^{[1]} - \alpha \frac{\partial C}{\partial Z^{[1]}} A^{[0]T}$$

$$b^{[1]} = b^{[1]} - \alpha \frac{\partial C}{\partial b^{[1]}} = W^{[1]} - \alpha \frac{\partial C}{\partial Z^{[1]}}$$

✱ Summarising, we have,

For each epoch, the backpropagation involves calc. of weights & biases of each layer. i.e from o/p to i/p.

From forward prop. we have value of $A^{[L]}$ and C is a func. depending on $A^{[L]}$.

Algorithm:

layer = Outermost layer L, $\frac{\partial C}{\partial A^{[L]}}$ is given.

repeat $L \geq 1$

{

$$\frac{\partial C}{\partial Z^{[L]}} = \sigma'(Z^{[L]}) \frac{\partial C}{\partial A^{[L]}}$$

$$\frac{\partial C}{\partial W^{[L]}} = \frac{\partial C}{\partial Z^{[L]}} (A^{[L-1]})^T$$

$$\frac{\partial C}{\partial b^{[L]}} = \frac{\partial C}{\partial Z^{[L]}}$$

$$W^{[L]} = W^{[L]} - \alpha \frac{\partial C}{\partial W^{[L]}}$$

$$b^{[L]} = b^{[L]} - \alpha \frac{\partial C}{\partial b^{[L]}}$$

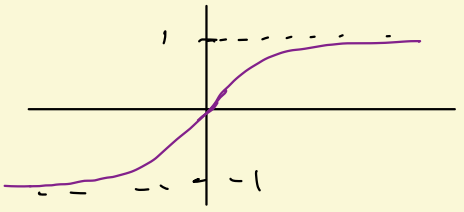
} Updation step.

$$\frac{\partial C}{\partial A^{[L-1]}} = (W^{[L]})^T \frac{\partial C}{\partial Z^{[L]}}$$

$L = L - 1$ }

Activation functions:

① Tanhx : $\frac{e^x - e^{-x}}{e^x + e^{-x}}$

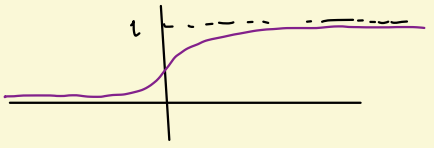


derivative:

$$\frac{(e^x+e^{-x})(e^x+e^{-x}) - (e^x-e^{-x})(e^x-e^{-x})}{(e^x+e^{-x})^2} = 1 - \frac{(e^x-e^{-x})^2}{(e^x+e^{-x})^2}$$ (seperating denominators)

$$= 1 - (\text{Tanh}\, x)^2$$

② Sigmoid : $\frac{1}{1+e^{-x}}$



derivative :

$$\frac{-(-e^{-x})}{(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}-1}{(1+e^{-x})^2} = \frac{1}{(1+e^{-x})} - \frac{1}{(1+e^{-x})^2}$$

$$= \text{Sig}(1 - \text{Sig})$$

Back propagation:

1) $\frac{\partial C}{\partial A^l}$ is given

Calculate $\frac{\partial C}{\partial Z^l} = \sigma'(z^l) \frac{\partial C}{\partial A^l}$ ↳ element wise mult'n.

2) $\frac{\partial C}{\partial W^l} = \frac{\partial C}{\partial Z^l} \cdot (A^{l-1})^T$

$\frac{\partial C}{\partial b^l} = \frac{\partial C}{\partial z^l}$

3) $W^l = W^l - \alpha \frac{\partial C}{\partial W^l}$

$b^l = b^l - \alpha \frac{\partial C}{\partial b^l}$

4) Update

$\frac{\partial C}{\partial A^{l-1}} = (W^l)^T \cdot \frac{\partial C}{\partial Z^l}$

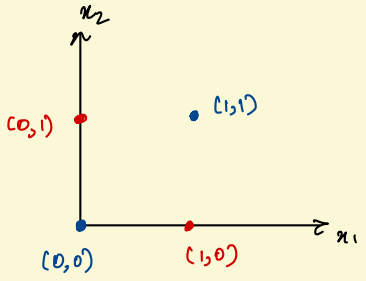
$A^{[l]} = \sigma(z^{[l]})$.

Hence, $\frac{\partial A^{[l]}}{\partial z^{[l]}} = \sigma'(z^{[l]})$

★ Neural Network Implementation:

The XOR Gate:

with a single perceptron, it is not possible find a linear hyperplane.



clearly the points are not linearly seperable.

why?

This is an example of an XOR gate.

x_1	x_2	XOR
0	0	0
1	0	1
0	1	1
1	1	0

$$w_0 + \sum_{i=1}^{2} w_i x_i^{\circ} < 0$$

$$w_0 + \sum_{i=1}^{2} w_i x_i^{\circ} \geq 0$$

$$w_0 + \sum_{i=1}^{2} w_i x_i^{\circ} \geq 0$$

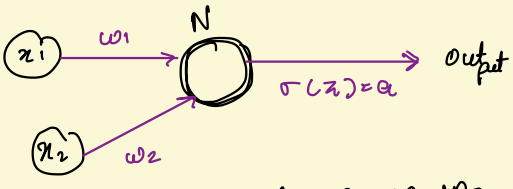
$$w_0 + \sum_{i=1}^{2} w_i x_i^{\circ} < 0$$

① $w_0 + w_1(0) + w_2(0) < 0 \Rightarrow w_0 < 0$

② $w_0 + w_1(0) + w_2(1) \geq 0 \Rightarrow w_2 \geq -w_0$

③ $w_0 + w_1(1) + w_2(1) \geq 0 \Rightarrow w_1 \geq -w_0$

④ $w_0 + w_1(1) + w_2(1) \geq 0 \Rightarrow w_1 + w_2 < -w_0$



No such w_0, w_1, w_2 s.t output of NN = output of XOR Gate

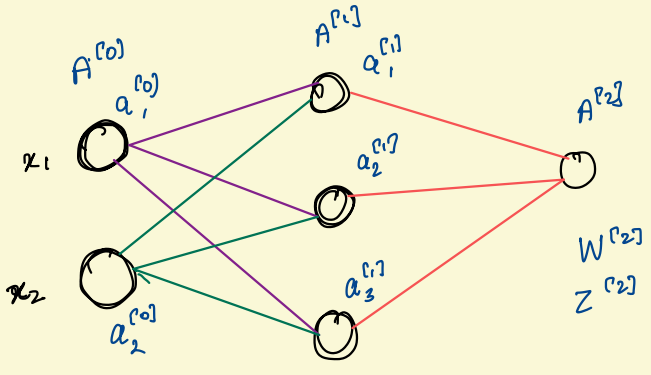
2^{nd} & 3^{rd} cond^n contradict the 4^{th} condition.

∴ we can't have a sol^n to this set of inequalities.

but it is possible to find a solution with a network of perceptrons.

① Input : 2 features x_1, x_2 and 4 samples

In this example, layer 1 has 3 neuro layer 2 is olp layer with one neuro



$W^{[1]}$
$z^{[1]}$

$\dim W^{[l]} = (\#\text{ of neurons in layer } l, \#\text{ of inputs})$

$\dim b^{[l]} = (\#\text{ of neurons in layer } l, 1)$

$$\begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} \end{bmatrix} \begin{bmatrix} a_1^{[0]} \\ a_2^{[0]} \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix}$$

$W^{[1]}$

$$\begin{bmatrix} a_1^{[0]} \\ a_2^{[0]} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

* Given training set with correct output, In each iteration or epoch, we try to find a set of weight & biases so that the error (diff b/w actual value & the one predicted by the set of weights) is minimum.

* In each epoch, we train the network with all training samples. one by one (ie, input 0, 0 adjust the weights, the 1,0 adjust the weights, etc).

So in one epoch, we will have to adjust the weights the # of sample times. This is inefficient.

Hence, one epoch, we will try to input all the samples as a Batch.

* Input:

Feature x_1: [0 0 1 1]

Feature x_2: [0 1 0 1]

training data set is same as testing data set.

x_1	x_2	XOR
0	0	0
1	0	1
0	1	1
1	1	0

X_train = [[0 0 1 1] [0 1 0 1]]

y_train = [[0 1 1 0]]

* Defining each layer through a class:

Each layer has 2 variables ① # of neurons in that layer

② # of inputs.

③ weights : dim : (# of neurons, # of i/ps)

④ biases : dim (# of neurons, 1)

- Iterating through Epochs.

In each epoch following takes place:

① Feed forward.

Instead of $z^{[1]} = W^{[1]}X + b^{[1]}$, $X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ → Single Input.

we now have a batch of inputs.

$$X = \begin{bmatrix} [x_1] \\ [x_2] \end{bmatrix} = \begin{bmatrix} [x_1^1, x_1^2 \cdots, x_1^m], \\ [x_2^1, x_2^2 \cdots, x_2^m] \end{bmatrix} = A^{[0]} = \begin{bmatrix} [0 \; 0 \; \cdots] \\ [0, 0, \cdots] \end{bmatrix}$$

← 4 samples → ; ↑ 2 features ↓

Code flow

(i) $z^{[1]} = W^{[1]} \; X + b^{[1]}$

($z^{[1]}$: (# of neurons in layer 1, # of samples); $W^{[1]}$: (# of neurons, # of i/p); X: (# of features, # of samples); $b^{[1]}$: (# of neurons, 1))

$$z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{bmatrix} = \begin{bmatrix} [z_{11}^{[1]}, z_{12}^{[1]}, \cdots & z_{1m}^{[1]}] \\ [z_{21}^{[1]}, z_{22}^{[1]} \cdots & z_{2m}^{[1]}] \\ [z_{31}^{[1]}, z_{32}^{[1]} \cdots & z_{3m}^{[1]}] \end{bmatrix}$$

$$A^{[1]} = \sigma(z^{[1]}) = \begin{bmatrix} [\sigma(z_{11}^{[1]}) \cdots &] \\ \vdots & \\ [\sigma(z_{31}^{[1]}) \cdots &] \end{bmatrix}$$

($A^{[1]}$: (# of neurons in layer 1, # of samples))

Note:
← This addition is possible with np.array. Each row of $b^{[1]}$ gets added to all columns of same row of $W^{[1]}$

Layer 1 has 3 neurons

$W^{[1]}$ = dim (3, 2)

$A^{[1]}$ = dim (3, 4)

# of samples = m = 4

(2) $z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$

(1, m) (1, 3) (3, m) (1, 1)

$W^{[2]}$ = dim (1, 3)

$$z^{[2]} = [z_1^{[2]}] = [\, [z_{11}^{[2]}, z_{12}^{[2]}, \cdots z_{1m}^{[2]}] \,]$$

$$A^{[2]} = \sigma(z^{[2]}) = [\, [\sigma(z_{11}^{[2]}), \sigma(z_{12}^{[2]}), \cdots \sigma(z_{1m}^{[2]})] \,]$$

dim $A^{[2]}$ = (1, m)

— FORWARD PROP —

(2) Calculating the cost

The cost function is

$$C(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(a_i) + (1-y^{(i)}) \log(1-a_i) \right]$$

→ summing over m samples → loss for all samples $(x_1^{(i)}, x_2^{(i)})$, $1 \le i \le m$

$a_i = A^{[i]}$ = last layer = predicted output.

- In our case, $A^{[i]} = A^{[2]} = [\, [\, \sigma(z_{11}^{[2]}), \sigma(z_{12}^{[2]}), \ldots, \sigma(z_{1m}^{[2]}) \,] \,]$
  - $\sigma(z_{11}^{[2]})$: predicted o/p of 1st sample
  - $\sigma(z_{12}^{[2]})$: pred. o/p of 2nd sample
  - $\sigma(z_{1m}^{[2]})$: pred. o/p of mth sample

- Actual o/p: $y^{[i]} = [\, [y_1, y_2, \ldots, y_m] \,]$

$$1 - y^{[i]} = [\, [(1-y_1), (1-y_2), \ldots, (1-y_m)] \,]$$

$$\log A^{[2]} = [\, [\log(\sigma(z_{11}^{[2]})), \ldots, \log(\sigma(z_{1m}^{[2]}))] \,]$$

$$\log(1-A^{[2]}) = [\, [\log(1-\sigma(z_{11}^{[2]})), \ldots, \log(1-\sigma(z_{1m}^{[2]}))] \,]$$

$*$ = Element-wise log

$$-\left[y^{[i]} * \log A^{[2]} + (1-y) * \log(1-A^{[2]}) \right] =$$

(array, array; array, array)

$$-[\, [y_1 * \log(\sigma(z_{11}^{[2]})), \ldots, y_m * \log(\sigma(z_{1m}^{[2]}))] \,] + [\, [(1-y_1) * \log(1-\sigma(z_{11}^{[2]})), \ldots, (1-y_m) * \log(1-\sigma(z_{1m}^{[2]}))] \,]$$

$$= -[\, [y_1 * \log(\sigma(z_{11}^{[2]})) + (1-y_1) * \log(1-\sigma(z_{11}^{[2]})), \ldots, y_m * \log(\sigma(z_{1m}^{[2]})) + (1-y_m) * \log(1-\sigma(z_{1m}^{[2]})) \,]$$

Cost = np.sum(above array)

(3) Now, have to update the weights through back prop.

$\dim A^{[2]} = (\#$ of neurons in layer 2, m samples$)$

$\dim Z^{[2]} = (\#$ of neurons in layer 2, m samples$)$

(i) calc. $\frac{\partial C}{\partial A^{[2]}}$

$$C(\vec{w}, b) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^i \log\left(a_i^{[2]}\right) + (1-y^i)\log\left(1-a_i^{[2]}\right)\right)$$

$$A^{[2]} = \left[\left[a_1^{[2]}, a_2^{[2]}, \ldots, a_m^{[2]}\right]\right]$$

$$\frac{\partial C}{\partial A^{[2]}} = \left[\left[\frac{\partial C}{\partial a_1^{[2]}}, \frac{\partial C}{\partial a_2^{[2]}}, \ldots, \frac{\partial C}{\partial a_m^{[2]}}\right]\right]$$

$$\frac{\partial C}{\partial a_i^{[2]}} = -\frac{1}{m}\left(\frac{y^i}{a_i^{[2]}} - \frac{(1-y^i)}{1-a_i^{[2]}}\right)$$

$$\frac{\partial C}{\partial a_i^{[2]}} = -\frac{1}{m}\left[\frac{y^i - a_i^{[2]}y^i - a_i^{[2]} + a_i^{[2]}y^i}{a_i^{[2]}(1-a_i^{[2]})}\right]$$

$$\frac{\partial C}{\partial a_i^{[2]}} = \frac{1}{m}\left[\frac{a_i^{[2]} - y^i}{a_i^{[2]}(1-a_i^{[2]})}\right]$$

$$\frac{\partial C}{\partial A^{[2]}} = \left[\left[\frac{1}{m}\left(\frac{a_1^{[2]} - y^1}{a_1^{[2]}(1-a_1^{[2]})}\right), \frac{(a_2^{[2]} - y^2)}{m\, a_2^{[2]}(1-a_2^{[2]})}, \ldots, \frac{1}{m}\left(\frac{a_m^{[2]} - y^m}{a_m^{[2]}(1-a_m^{[2]})}\right)\right]\right]$$

How can this be computed?

For Vectorizing: $(A^{[2]} - y\_train)$ → array dim = (1, # of samples) gives ↑

$A^{[2]} * (1-A^{[2]})$ → array dim = (1, # of samples)

Algorithm:

layer = Outermost layer L, $\frac{\partial C}{\partial A^{[L]}}$ is given.

repeat $L \geq 1$

{

$$\frac{\partial C}{\partial Z^{[L]}} = \sigma'(z^{[L]})\frac{\partial C}{\partial A^{[L]}}$$

$$\frac{\partial C}{\partial W^{[L]}} = \frac{\partial C}{\partial Z^{[L]}}(A^{[L-1]})^T\left(\frac{1}{m}\right)$$

$$\frac{\partial C}{\partial b^{[L]}} = \frac{\partial C}{\partial Z^{[L]}}$$

$$W^L = W^L - \alpha\frac{\partial C}{\partial W^L}, \quad b^L = b^L - \alpha\frac{\partial C}{\partial b^L}$$ } updation step

$$\frac{\partial C}{\partial A^{[L-1]}} = (W^{[L]})^T\frac{\partial C}{\partial Z^{[L]}}$$

$L = L-1$ }

(ii) Backprop for layer 2

$$\frac{\partial C}{\partial Z^{[2]}} = \sigma'(Z^{[2]}) * \frac{\partial C}{\partial A^{[2]}}$$

(1, # of samples) ← (1, # of samples) * (1, # of samples)

(1, 4) ← (1, # of samples) * (1, # of samples)

$$\frac{\partial C}{\partial W^{[2]}} = \frac{1}{m} * \frac{\partial C}{\partial Z^{[2]}} \cdot (A^{[1]})^T$$

(1, 3) ← (1, 4) · (4, 3)

$$\frac{\partial C}{\partial b^{[2]}} = \frac{1}{m} * \left(\frac{\partial C}{\partial Z^{[2]}}\right) \text{ summing across columns} \longrightarrow \text{Why?}$$

(1, 1) ← (1, 1)

- $Z^{[2]} = (1, 4)$

$\sigma'(Z^{[2]}) = sig(Z^{[2]})(1 - sig(Z^{[2]}))$ ← element-wise

$\left[\left[\sigma'(z_1^{[2]}), \sigma'(z_2^{[2]}), \ldots, \sigma'(z_m^{[2]})\right]\right]$

$sig(z_1^{[2]})(1 - sig(z_1^{[2]})) \rightarrow (1, \#\text{ of samples})$

- $A^{[1]} = (\#\text{ of neurons in Layer 1}, \#\text{ of samples})$

(3, 4)

Update the weights

$$W^{[2]} = W^{[2]} - \alpha \frac{\partial C}{\partial W^{[2]}}$$

$$b^{[2]} = b^{[2]} - \alpha \frac{\partial C}{\partial b^{[2]}}$$

$$\frac{\partial C}{\partial A^{[1]}} = (W^{[2]})^T \cdot \left(\frac{\partial C}{\partial Z^{[2]}}\right)$$

(3,4) ← (3,1) (1,4)

return $\frac{\partial C}{\partial A^{[1]}}$

Intuition:

for every sample we do,

$b^{[2]} = b^{[2]} - \alpha \left[\frac{\partial C}{\partial b^{[2]}}\right]$ that

∴ for each sample sample above happens · if we don't vectorize

Vectorizing, it happens when we add all columns together

why (3,4)?

$$A^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix} = \begin{bmatrix} [a_{11}^{[1]}, a_{12}^{[1]}, \dots, a_{1m}^{[1]}] \\ [a_{21}^{[1]}, a_{22}^{[1]}, \dots, a_{2m}^{[1]}] \\ [a_{31}^{[1]}, a_{32}^{[1]}, \dots a_{2m}^{[1]}] \end{bmatrix}$$

We found need $W^{[2]}$, $b^{[2]}$

$$\frac{\partial C}{\partial A^{[1]}} = \begin{bmatrix} [\frac{\partial C}{\partial a_{11}^{[1]}}, \frac{\partial C}{a_{12}^{[1]}}, \dots \frac{\partial C}{\partial a_{1m}^{[1]}}] \\ [\frac{\partial C}{\partial a_{21}^{[1]}}, \dots, \frac{\partial C}{\partial a_{2m}^{[1]}}] \\ [\frac{\partial C}{\partial a_{n1}^{[1]}}, \dots, \frac{\partial C}{\partial a_{nm}^{[1]}}] \end{bmatrix}$$

(iii) Backpropagation for Layer 1.

$\frac{\partial C}{\partial A^{[1]}}$ is calculated

$\frac{\partial C}{\partial A^{[1]}} = (3,4)$

$$\frac{\partial C}{\partial Z^{[1]}} = \sigma'(Z^{[1]}) \cdot \frac{\partial C}{\partial A^{[1]}}$$

(3, # of samples) ← (3, # of samples) (3, # of samples)

(3, 4)

$A^{[0]} = (2, \#\text{ of samples})$

(2, 4)

$$\frac{\partial C}{\partial W^{[1]}} = \frac{1}{m} * \frac{\partial C}{\partial Z^{[1]}} \cdot (A^{[0]})^T$$

(3,2) ← (3,4) (4,2)

$W^{[1]} = (3,2)$

$$\frac{\partial C}{\partial b^{[1]}} = \frac{1}{m} * \left(\frac{\partial C}{\partial Z^{[1]}}\right) \text{ summing across columns}$$

(3,1) ← (3,4)

Update the weights

$$W^{[i]} = W^{[i]} - \alpha \frac{\partial C}{\partial W^{[i]}}$$

$$b^{[i]} = b^{[i]} - \alpha \frac{\partial C}{\partial b^{[i]}}$$

$$\frac{\partial C}{\partial A^{[0]}} = (W^{[i]})^T \cdot \left(\frac{\partial C}{\partial z^{[i]}}\right)$$

$\frac{\partial C}{\partial A^{[0]}}$: (2,4) = $(W^{[i]})^T$: (2,3) · $\left(\frac{\partial C}{\partial z^{[i]}}\right)$: (3,4)

return $\frac{\partial C}{\partial A^{[0]}}$

Hence after the number of epochs we have the minimized weights & biases -

* MNIST dataset Example

PROBLEM: Build a neural network to classify digits from 0 to 9

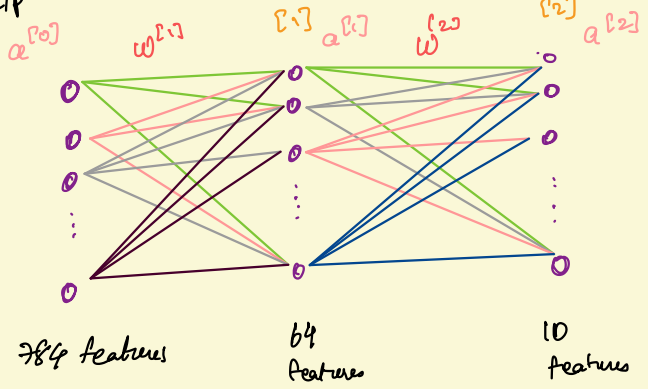
Data set: MNIST dataset

Each image is of $28 \times 28 = 784$ pixels $\therefore$ # of features = 784

m = # of samples.

$x\_train = (m, \#\text{ of features})$

I/p



If $a^{[2]} = \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ \vdots \\ 9 \end{matrix} \begin{bmatrix} 0.1 \\ 0.8 \\ 0.1 \\ 0.1 \\ \vdots \\ 0.1 \end{bmatrix}$ then there is 80% chance that the given sample is 1.

2 Hidden layers

layer 1 : (# of neurons, # of i/ps) = (64, 784)

layer 2 : (# of neurons, # of i/ps) = (10, 64)

Activation functions used:

layer I : $a^{[1]} = ReLU(z^{[1]})$

layer II : $a^{[2]} = Softmax(z^{[2]})$

We need to run gradient descent in every epoch to improve the weights & Bias of each neuron in every layer.

$$W^{[1]} = \begin{matrix} \\ neu_1 \\ neu_2 \\ \vdots \\ neu_{64} \end{matrix} \begin{bmatrix} feat_1 & feat_2 & \cdots & feat_{784} \\ w_{11} & w_{12} & \cdots & w_{1784} \\ w_{21} & w_{22} & \cdots & w_{2784} \\ & & & \vdots \\ w_{641} & w_{642} & \cdots & w_{64,784} \end{bmatrix} \begin{matrix} w_1^{[1]} \\ w_1^{[2]} \\ \vdots \\ w^{[64]} \end{matrix}$$

$$A^{[0]} = \begin{matrix} \\ fe_1 \\ fe_2 \\ \vdots \\ fe_{784} \end{matrix} \begin{bmatrix} s1 & samp.2 & \cdots & samp.m \\ x_{11} & x_{21} & \cdots & x_{m1} \\ x_{12} & x_{22} & \cdots & x_{m2} \\ \vdots & \vdots & \vdots & \vdots \\ x_{1784} & x_{2784} & \cdots & x_{m784} \\ \downarrow & \downarrow & & \downarrow \\ Samp1 & Samp2 & & Sampm. \end{bmatrix}$$

$$B^{[1]} = \begin{matrix} neu_1 \\ neu_2 \\ \vdots \\ neu_{64} \end{matrix} \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$$

$A^{[0]} = x\_train.T$

$y\_train = [\,[\,7 \;\; 8 \;\; 0 \;\; 1 \;\; \cdots \;\; 9\,]\,]_{1 \times m}$

(7 ↓ o/p samp1; 8 ↓ o/p samp2; 9 ↓ o/p samp m.)

① Forward Propagation.

$$Z^{[1]} = W^{[1]} A^{[0]} + B^{[1]}$$

$$A^{[1]} = \text{ReLU}(Z^{[1]})$$

$$Z^{[2]} = W^{[2]} A^{[1]} + B^{[1]}$$

$$A^{[2]} = \text{softmax}(Z^{[2]})$$

$Z^{[1]}$ = (neu1, neu2, ⋮, neu64) × (samp1, samp2, ..., samp m)

$W^{[2]}$ = (neu1, neu2, ⋮, neu10) × (fe1, fe2, ..., fe 64)

$Z^{[2]}$ = (n1, neu2, ⋮, neu10) × (samp1, samp2, .., sampm)

↓ Prediction of sample 1 ... ↓ Pred of sample m.

② Calculating the loss

$$\text{loss} = -\sum_{i=1}^{n} y_i \log a_i \quad \rightarrow \text{for one sample}$$

(1 sample)

$$\text{Cost} = -\sum_{i=1}^{m} \sum_{j=1}^{n} y_j^{[i]} \log(a_j^{[i]})$$

If $y_1 = 5$

y = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) × (0 0 0 0 0 1 0 0 0 0)

a_1 = (0.1, 0.1, 0.1, 0.8, 0.1, ⋮, 0.1)

Single sample

Softmax activation function

$$\begin{bmatrix} 1.3 \\ 5.1 \\ 2.2 \\ 0.7 \\ 1.1 \end{bmatrix} \rightarrow \left[\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_i}} \right] \rightarrow \begin{bmatrix} 0.02 \\ 0.90 \\ 0.05 \\ 0.01 \\ 0.02 \end{bmatrix}$$

$$l = \text{loss}(a_1, \ldots, a_N, y) = \begin{cases} -\log a_1, & y = 1 \\ -\log a_2, & y = 2 \\ \vdots & \\ -\log a_N, & y = N \end{cases}$$

③ Backpropagation.

(a) Layer II

Find $\frac{\partial C}{\partial Z^{[2]}}$ $\quad$ $\text{Cost } C = -\sum_{i=1}^{m} \sum_{j=1}^{n} y_j^{[i]} \log(a_j^{[i]})$

$$\frac{\partial C}{\partial Z^{[2]}} = \left[\frac{\partial C}{\partial z_1^{[2]}}, \frac{\partial C}{\partial z_2^{[2]}}, \ldots, \frac{\partial C}{\partial z_{10}^{[2]}} \right]$$

Expanding for all the m samples,

Algorithm:

layer = Outermost layer l, $\frac{\partial C}{\partial A^{[l]}}$ is given

repeat $l \geq 1$

{

$$\frac{\partial C}{\partial Z^{[l]}} = \sigma'(z^{[l]}) \frac{\partial C}{\partial A^{[l]}}$$

$$\frac{\partial C}{\partial W^{[l]}} = \frac{\partial C}{\partial Z^{[l]}} (A^{[l-1]})^T \left[\frac{1}{m}\right]$$

$$\frac{\partial C}{\partial b^{[l]}} = \frac{\partial C}{\partial Z^{[l]}}$$

$$W^{[l]} = W^{[l]} - \alpha \frac{\partial C}{\partial W^{[l]}}$$

$$b^{[l]} = b^{[l]} - \alpha \frac{\partial C}{\partial b^{[l]}}$$

} updation step

$$\frac{\partial C}{\partial A^{[l-1]}} = (W^{[l]})^T \frac{\partial C}{\partial Z^{[l]}}$$

$l = l - 1$ }

$$\frac{\partial C}{\partial z^{[2]}} = \begin{bmatrix} \frac{\partial C}{\partial z_{11}^{[2]}} & \frac{\partial C}{\partial z_{21}^{[2]}} & \cdots & \frac{\partial C}{\partial z_{10,1}^{[2]}} \\ \frac{\partial C}{\partial z_{12}^{[2]}} & \frac{\partial C}{\partial z_{22}^{[2]}} & \cdots & \frac{\partial C}{\partial z_{10,2}^{[2]}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial C}{\partial z_{1m}^{[2]}} & \frac{\partial C}{\partial z_{2m}^{[2]}} & \cdots & \frac{\partial C}{\partial z_{10,m}^{[2]}} \end{bmatrix}^T$$

(↑ m samples)

★ For a single sample,

$$A^{[2]} = \begin{bmatrix} a_1^{[2]} \\ a_2^{[2]} \\ \vdots \\ a_{10}^{[2]} \end{bmatrix} \qquad a_i = \frac{e^{z_i^{[2]}}}{\sum_{i=1}^{10} e^{z_i^{[2]}}} \qquad z^{[2]} = \begin{bmatrix} z_1^{[2]} \\ z_2^{[2]} \\ \vdots \\ z_{10}^{[2]} \end{bmatrix}$$

$\sigma'(z^{[2]})$

$$\frac{\partial A^{[2]}}{\partial z^{[2]}} = \begin{bmatrix} \partial a_1^{[2]}/\partial z_1^{[2]} & \partial a_1^{[2]}/\partial z_2^{[2]} & \cdots & \partial a_1^{[2]}/\partial z_{10}^{[2]} \\ \vdots & \vdots & \ddots & \vdots \\ \partial a_{10}^{[2]}/\partial z_1^{[2]} & \partial a_{10}^{[2]}/\partial z_2^{[2]} & \cdots & \partial a_{10}^{[2]}/\partial z_{10}^{[2]} \end{bmatrix}$$

Recall $a_i = \dfrac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$, softmax.

Find $\dfrac{\partial a_i}{\partial z_j}$:

Case I: $i \neq j$

$$\frac{\partial}{\partial z_j}\left[\frac{e^{z_i}}{e^{z_j} + \sum_{k\neq j} e^{z_k}}\right] = \frac{\left(0 \cdot e^{z_j} + \left(\sum_{k\neq j} e^{z_k}\right)\frac{\partial}{\partial z_j}(e^{z_i}) - e^{z_i} e^{z_j}\right)}{\left(\sum_k e^{z_k}\right)^2} = \frac{-e^{z_i}}{\left(\sum_k e^{z_k}\right)} \cdot \frac{e^{z_j}}{\left(\sum_k e^{z_k}\right)}$$

$$= -a_i a_j$$

Case II: $i = i$

$$\frac{\partial a_i}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\frac{e^{z_i}}{e^{z_i} + \sum_{k\neq i} e^{z_k}}\right) = \frac{\left(\sum_k e^{z_k}\right) e^{z_i} - \left(e^{z_i}\right)\left(e^{z_i}\right)}{\left(\sum_k e^{z_k}\right)^2}$$

$$= \frac{e^{z_i}\left(\sum_{k\neq i} e^{z_k}\right)}{\left(\sum_k e^{z_k}\right)\left(\sum_k e^{z_k}\right)} = a_i(1 - a_i)$$

★ $$\frac{\partial L}{\partial z} = \left[\frac{\partial L}{\partial z_1^{[2]}}, \frac{\partial L}{\partial z_2^{[2]}}, \cdots \frac{\partial L}{\partial z_{10}^{[2]}}\right]$$

Calculate $\dfrac{\partial L}{\partial z_i}$ wrt each z_i

$$\frac{\partial L}{\partial z_i} = \frac{\partial}{\partial z_i}\left[-y_i \log(a_i) - \sum_{k \neq i} y_k \log(a_k)\right] = -\frac{y_i}{a_i}\left(\frac{\partial a_i}{\partial z_i}\right) - \sum_{k \neq i} \frac{y_k}{a_k}\frac{\partial a_k}{\partial z_i}$$

i ranging from 1 to n.

$$= -\frac{y_i}{a_i} a_i(1-a_i) + \sum_{k \neq i} \frac{y_k}{a_k} a_k \cdot a_i = -y_i + y_i a_i + a_i \sum_{k \neq i} y_k$$

$$= -y_i + a_i\left(y_i + \sum_{k \neq i} y_k\right) = -y_i + a_i\left(\sum_{k=1}^{n} y_k\right) = a_i - y_i.$$

$$\therefore \frac{\partial L}{\partial z_i} = a_i - y_i \quad \forall\ i = 1, 2, \ldots n.$$

$$\frac{\partial C}{\partial z^{[2]}} = \begin{matrix} \uparrow \\ m \\ \text{samples} \\ \downarrow \end{matrix} \begin{bmatrix} \frac{\partial C}{\partial z_{11}^{[2]}} & \frac{\partial C}{\partial z_{21}^{[2]}} & \cdots & \frac{\partial C}{\partial z_{10,1}^{[2]}} \\ \frac{\partial C}{\partial z_{12}^{[2]}} & \frac{\partial C}{\partial z_{22}^{[2]}} & \cdots & \frac{\partial C}{\partial z_{10,2}^{[2]}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial C}{\partial z_{1m}^{[2]}} & \frac{\partial C}{\partial z_{2m}^{[2]}} & \cdots & \frac{\partial C}{\partial z_{10,m}^{[2]}} \end{bmatrix} = \begin{bmatrix} \frac{\partial C}{\partial z_1} & \frac{\partial C}{\partial z_2} & \cdots & \frac{\partial C}{\partial z_{10}} \end{bmatrix}$$

$$A^{[2]} = \begin{matrix} \\ \text{feat 1} \\ \text{feat 2} \\ \vdots \\ \text{feat 10} \end{matrix} \begin{matrix} \text{samp 1} & \text{samp 2} & & \text{samp m} \end{matrix} \begin{bmatrix} a_{11}^{[2]} & a_{21}^{[2]} & \cdots & a_{m1}^{[2]} \\ a_{12}^{[2]} & a_{22}^{[2]} & & a_{m2}^{[2]} \\ \vdots & \vdots & & \vdots \\ a_{110}^{[2]} & a_{210}^{[2]} & & a_{m10}^{[2]} \end{bmatrix}$$

From the above, we get,

$$\frac{\partial L_m}{\partial z} = \begin{bmatrix} \frac{\partial L_m}{\partial z_1} & \frac{\partial L_m}{\partial z_2} & \cdots & \frac{\partial L_m}{\partial z_{10}} \end{bmatrix}$$

$$\frac{\partial L_m}{\partial z} = \begin{bmatrix} a_{m,1} - y_{m1} & a_{m2} - y_{m2} & \cdots & a_{m,10} - y_{m10} \end{bmatrix}$$

$$\frac{\partial C}{\partial z^{[2]}} = \begin{bmatrix} \frac{\partial L_1}{\partial z_1} & \frac{\partial L_1}{\partial z_2} & \cdots & \frac{\partial L_1}{\partial z_{10}} \\ \frac{\partial L_2}{\partial z_1} & \frac{\partial L_2}{\partial z_2} & \cdots & \frac{\partial L_2}{\partial z_{10}} \\ \vdots & & & \\ \frac{\partial L_m}{\partial z_1} & \frac{\partial L_m}{\partial z_2} & \cdots & \frac{\partial L_m}{\partial z_{10}} \end{bmatrix}^T = \begin{bmatrix} a_{11}^{[2]} - y_{11}^{[2]} & a_{12}^{[2]} - y_{12}^{[2]} & \cdots & a_{1,10}^{[2]} - y_{10}^{[2]} \\ a_{21}^{[2]} - y_{21}^{[2]} & a_{22}^{[2]} - y_{22}^{[2]} & \cdots & a_{2,10}^{[2]} - y_{2,10}^{[2]} \\ \vdots & \vdots & & \vdots \\ a_{m1}^{[2]} - y_{m1}^{[2]} & a_{m2}^{[2]} - y_{m2}^{[2]} & \cdots & a_{m,10}^{[2]} - y_{m,10}^{[2]} \end{bmatrix}^T$$

$$= A^{[2]} - Y$$

Size of $\frac{\partial C}{\partial z^{[2]}} = (\#\text{ of neurons in layer 2}, \#\text{ of samples})$

Hence we have, For layer 2

$$\frac{\partial C}{\partial Z^{[2]}} = A^{[2]} - Y$$

$(10, m)$

$$\frac{\partial C}{\partial W^{[2]}} = \left(\frac{1}{m}\right) \frac{\partial C}{\partial Z^{[2]}} \cdot (A^{[1]})^T$$

$(10,64)$ $\quad (10,m)\ (m,64)$

$$\frac{\partial C}{\partial b^{[2]}} = \left(\frac{1}{m}\right) \frac{\partial C}{\partial Z^{[2]}}$$

$(10,1)$ $\quad (10,1)$

$$W^{[2]} = W^{[2]} - \alpha \frac{\partial C}{\partial W^{[2]}} \quad \text{of size } (10,64)$$

$$b^{[2]} = b^{[2]} - \alpha \frac{\partial C}{\partial b^{[2]}} \quad \text{of size } (10,1)$$

} Updating weights of the second layer.

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

$(64,m)$ $\quad (64,10)$ $\quad (10,m)$

We have successfully updated the layer 2 weights.

$$\frac{\partial C}{\partial Z^{[l]}} = \sigma'(Z^{[l]}) \frac{\partial C}{\partial A^{[l]}}$$

$$\frac{\partial C}{\partial W^{[l]}} = \frac{\partial C}{\partial Z^{[l]}} (A^{[l-1]})^T \left(\frac{1}{m}\right)$$

$$\frac{\partial C}{\partial b^{[l]}} = \frac{\partial C}{\partial Z^{[l]}} \left(\frac{1}{m}\right) \rightarrow \text{vector got by sum along columns}$$

$$W^{[l]} = W^{[l]} - \alpha \frac{\partial C}{\partial W^{[l]}}$$

$$b^{[l]} = b^{[l]} - \alpha \frac{\partial C}{\partial b^{[l]}}$$

} updation step

$$\frac{\partial C}{\partial A^{[l-1]}} = (W^{[l]})^T \frac{\partial C}{\partial Z^{[l]}}$$

$l = l - 1$ }

(b) Back propagation for layer 1

Given: $\frac{\partial C}{\partial A^{[1]}}$ from the previous layer.

To calc: $\frac{\partial C}{\partial Z^{[1]}}$.

$$\frac{\partial C}{\partial Z^{[1]}} = \sigma'(Z^{[1]}) \frac{\partial C}{\partial A^{[1]}}$$

Recall σ = ReLU for 1st layer.

$$\text{ReLU}(z) = \begin{cases} z, & z \geq 0 \\ 0 & \text{else.} \end{cases}$$

$$\therefore \sigma'(z) = \begin{cases} 1, & z \geq 0 \\ 0 & \text{else.} \end{cases}$$

$\sigma'(Z^{[1]})$: size $(64, m)$

Caution: This is an element wise multip.

Hence, $\frac{\partial C}{\partial Z^{[1]}} = \sigma'(Z^{[1]}) \cdot \frac{\partial C}{\partial A^{[1]}}$

$(64, m)$ $\quad (64, m)$ $\quad (64, m)$

$$\therefore \frac{\partial C}{\partial Z^{[1]}} = g'(Z^{[1]}) \cdot \frac{\partial C}{\partial A^{[1]}}$$

$(64, m) \quad (64, m) \quad (64, m)$

$$\frac{\partial C}{\partial W^{[1]}} = \left(\frac{1}{m}\right) \frac{\partial C}{\partial Z^{[1]}} (A^{[0]})^T$$

$(64, 784) \quad (64, m) \quad (m, 784)$

$$\frac{\partial C}{\partial b^{[1]}} = \left(\frac{1}{m}\right) \frac{\partial C}{\partial Z^{[1]}} \rightarrow \text{Vector summed along the columns.}$$

$[64, 1)$

$$W^{[1]} = W^{[1]} - \alpha \frac{\partial C}{\partial W^{[1]}}$$
$$b^{[1]} = b^{[2]} - \alpha \frac{\partial C}{\partial b^{[1]}}$$

} Updating the weights of the layer 1

End of Back propagation.

In Summary, we have,

Our NN will have a simple two-layer architecture. Input layer $a^{[0]}$ will have 784 units corresponding to the 784 pixels in each 28×28 input image. A hidden layer $a^{[1]}$ will have 10 units with ReLU activation, and finally our output layer $a^{[2]}$ will have 10 units corresponding to the ten digit classes with softmax activation.

Forward propagation

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$
$$A^{[1]} = g_{\text{ReLU}}(Z^{[1]}))$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$
$$A^{[2]} = g_{\text{softmax}}(Z^{[2]})$$

Backward propagation

$$dZ^{[2]} = A^{[2]} - Y$$
$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$
$$dB^{[2]} = \frac{1}{m} \Sigma dZ^{[2]}$$
$$dZ^{[1]} = W^{[2]T} dZ^{[2]} . * g^{[1]\prime}(z^{[1]})$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} A^{[0]T}$$
$$dB^{[1]} = \frac{1}{m} \Sigma dZ^{[1]}$$

Parameter updates

$$W^{[2]} := W^{[2]} - \alpha dW^{[2]}$$
$$b^{[2]} := b^{[2]} - \alpha db^{[2]}$$
$$W^{[1]} := W^{[1]} - \alpha dW^{[1]}$$
$$b^{[1]} := b^{[1]} - \alpha db^{[1]}$$

Forward prop

- $A^{[0]} = X$: 784 x m
- $Z^{[1]} \sim A^{[1]}$: 10 x m
- $W^{[1]}$: 10 × 784 (as $W^{[1]} A^{[0]} \sim Z^{[1]}$)
- $B^{[1]}$: 10 × 1
- $Z^{[2]} \sim A^{[2]}$: 10 x m
- $W^{[1]}$: 10 × 10 (as $W^{[2]} A^{[1]} \sim Z^{[2]}$)
- $B^{[2]}$: 10 × 1

Backprop

- $dZ^{[2]}$: 10 x m ($A^{[2]}$)
- $dW^{[2]}$: 10 × 10
- $dB^{[2]}$: 10 × 1
- $dZ^{[1]}$: 10 x m ($A^{[1]}$)
- $dW^{[1]}$: 10 × 10
- $dB^{[1]}$: 10 × 1

On our case, the first layer has 64 neurons so modify accordingly

Source: Many many Youtube videos for derivation. (Coding have mainly) For the code, primarily Samson Zhang

Implementation from Scratch.

final Note : If $Y\_train = [[4 \quad 2 \quad 5 \quad \dots \quad 0 \quad 6]]_{1 \times m}$

then Y matrix will look like:

$Y =$

	1	2	3	...	m-1	m
0	0	0	0	...	1	0
1	0	0	0		0	0
2	0	1	0		0	0
3	0	0	0	...	0	0
4	1	0	0			0
5	0	0	1		:	1
6	0	0	0			0
7	0	:	:			0
8	0	:				0
9	0	0	0		0	0

How can we make our python code run faster?

Vectorizing
Broadcasting

Broadcasting

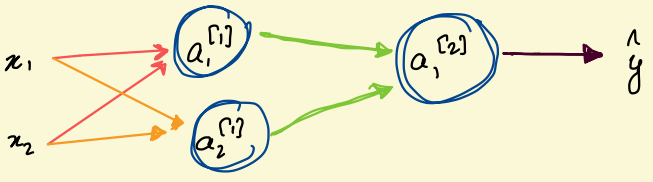
	Apples	Beef	Eggs	Potatoes
Carb	56	0	4.4	68
Protein	1.2	104	52	8
Fat	1.8	135	99	0.9

$A =$

Calculate %age of calories from Carb, protein and Fats. Can you do this without the explicit for loop?

```
cal = np.sum(A, axis = 0)
percentage = 100 * A / cal.reshape(1, 4)
```

* Random Initialization



Initializing b to zero is okay but initializing w to zero is not okay.

If we initialise $w^{[1]} = [0\ 0]$, $w^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$, $b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$$a_1^{[1]} = a_2^{[1]}$$

While Back propagation, $dz^{[1]} = dz^{[2]}$ When we initialize to zero, the hidden units are completely identical. In other words, completely symmetric

So even after n iterations, all the neurons compute the same function as in,

$$dw = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$$

```
w[1] = np.random.rand((2,2)) * 0.01
b[1] = np.zeros((2,1))
```

Small value

If w is large z is big so it may cause $\sigma(z) \approx 1$ or to be saturated. Slowing down the learning rate hence gradient descent

Logistic regression derivatives

$J = 0,\ dw_1 = 0,\ dw_2 = 0,\ db = 0$

→ for i = 1 to m:

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

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$ | $n_x = 2$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

$J = J/m,\ dw_1 = dw_1/m,\ dw_2 = dw_2/m,\ db = db/m$

Logistic regression derivatives

$J = 0,$ ~~$dw_1 = 0,\ dw_2 = 0,$~~ $db = 0$ $\quad dw = np.zeros((n_x, 1))$

→ for i = 1 to m:

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

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

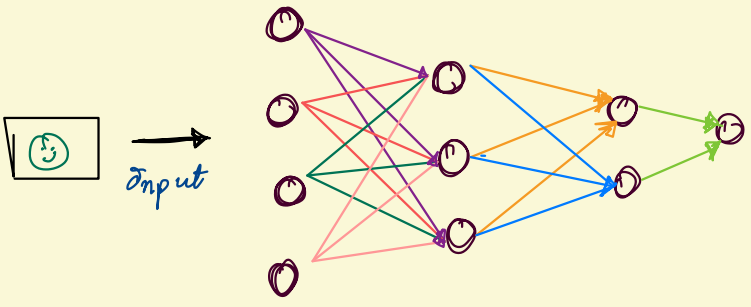
for j = 1 ... n_x: $dw_j \mathrel{+}= x_j^{(i)} dz^{(i)}$ | $n_x = 2$ $\quad dw \mathrel{+}= x^{(i)} dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

$J = J/m,$ ~~$dw_1 = dw_1/m,\ dw_2 = dw_2/m,$~~ $db = db/m$

$dw /= m.$

➢ Intuition about deep representations:



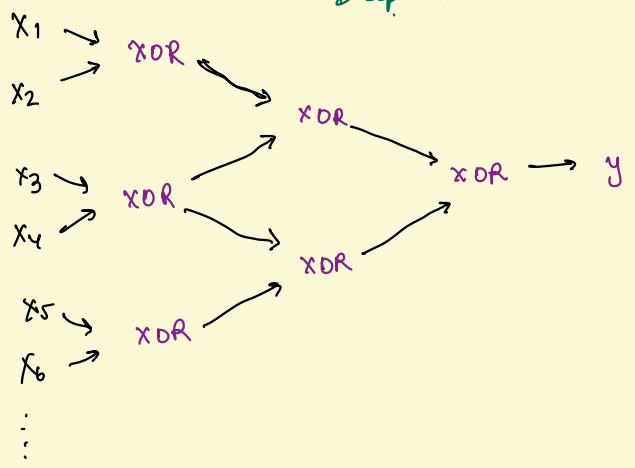
First layer starts off detecting simple things like edges, as the layers increase, the deep layers detect more complex things like eyes, faces etc.

★ Circuit theory and deep learning

There are functions you can compute with a small L-layer deep neural network than shallower networks that require exponentially more hidden units to compute

$y = x_1 \text{ XOR } x_2 \text{ XOR } x_3 \text{ XOR } x_4 \ \ldots \ \text{XOR } x_n$

Deep Network



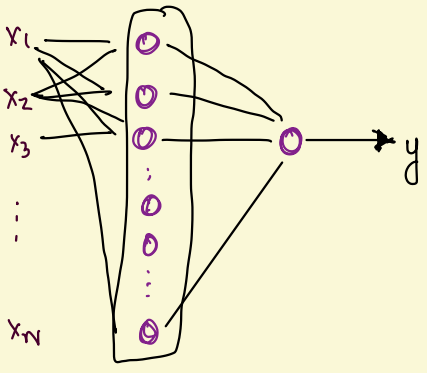
A Binary tree with n leaves has a height of a

With 4 leaves, $4 = 2^{(2)}$ — $2+1$

with 6, $6 = 2^{(3)}$ $3+2+1 = 6$

with 8,

Shallow Network



* Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]} b^{[3]}, \ldots$

Hyperparameters: learning rate α

↓

Parameters that ultimately control values of W & B

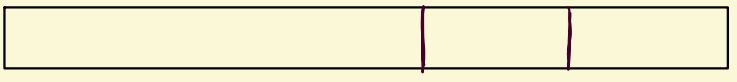
# of iterations n

# of hidden layers L

# of hidden units $n^{[1]}, n^{[2]}, \ldots$

Choice of activation functions

* Training set, Dev Set, Test Set



↓

to see which of many different models perform best on dev set

role of it is we test different algorithms on it & see which works better

60% train | 20% Dev | 20% test → Okay for a small dataset

98% train | 1% Dev | 1% test → Okay for Large datasets like 1 million

* Mismatched train/test distribution:

You are building an app that lets users upload a lot of pictures & the goal is to find pictures of cats in order to show the users.

Training set
Cat pics from webpages
high resolution, clear

Dev/test set
Cat pics from users your app.
low resolution, blurry

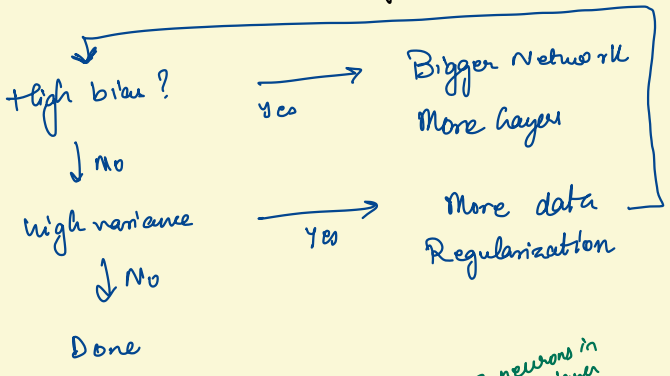
Both the sets come from different distributions.

Make sure dev/test set come from same distribution

Train set Error	1%	15%
Dev set Error	11%	16%
	high variance	high bias

Bayes Error ~ 0% (Human error)

* Basic recipe for Machine learning:



* Frobenius Norm:

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} (w_{ij}^{[l]})^2$$

$n^{[l]}$ = # of neurons in current layer; $n^{[l-1]}$ = # of neurons in previous layer

* Logistic regression

- L^2 regularization.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_{L^2}^2$$

- L^1 regularization

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_{L^1}$$

* Neural Network

$$J(w^{[1]}, b^{[1]}, \ldots, w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\|w^{[l]}\|^2$$

↓ Frob norm

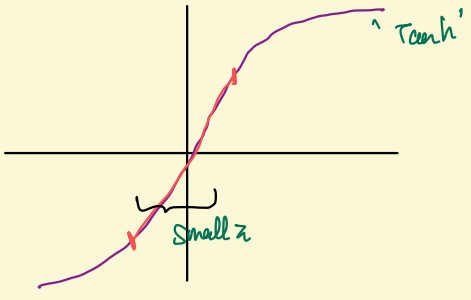
$$\|w^{[l]}\|^2 = \sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}} (w_{ij}^{[l]})^2$$

($n^{[l]}$: # of neurons in current layer; $n^{[l-1]}$: # of neurons in previous layer)

Now, $dw^{[l]}$ becomes, $dw^{[l]} = (\text{from backprop}) + \frac{\lambda}{m} w^{[l]}$

$$w^{[l]} = w^{[l]} - \alpha\, dw^{[l]}$$

* Why Regularization reduces overfitting?



If $\lambda \uparrow$, $w^{[l]} \downarrow$

$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$

$\Rightarrow z^{[l]}$ is small

the $g(z)$ is linear approx.

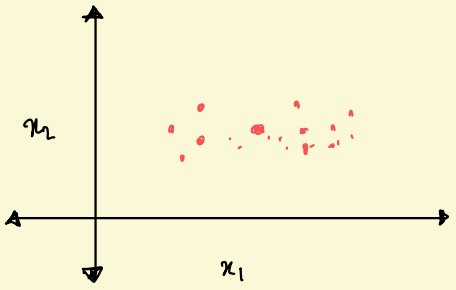
$\therefore$ every layer acts linear

* DROPOUT REGULARIZATION:

Go through each layer go through each node & toss a coin and have 0.5 chance of keeping each node & 0.5 chance of removing each node.

At each iteration remove different set of nodes and observe the performance of the remaining neural network. On each iteration, working with a smaller neural network

✱ Normalizing training sets:

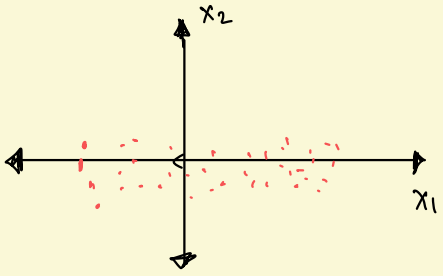


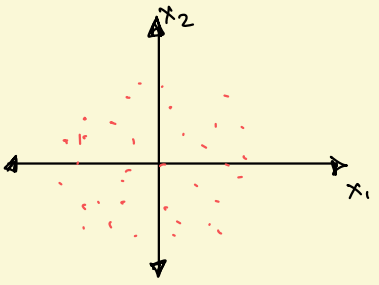
Subtract mean: $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{[i]}$

$x^{[i]} := x^{[i]} - \mu$

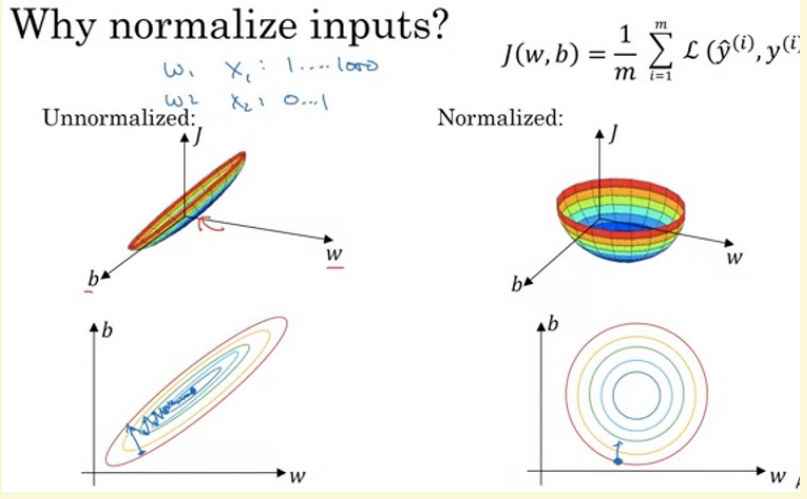
Variance: $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (x^{[i]})^2$

$x^{[i]} = x^{[i]} / \sigma$

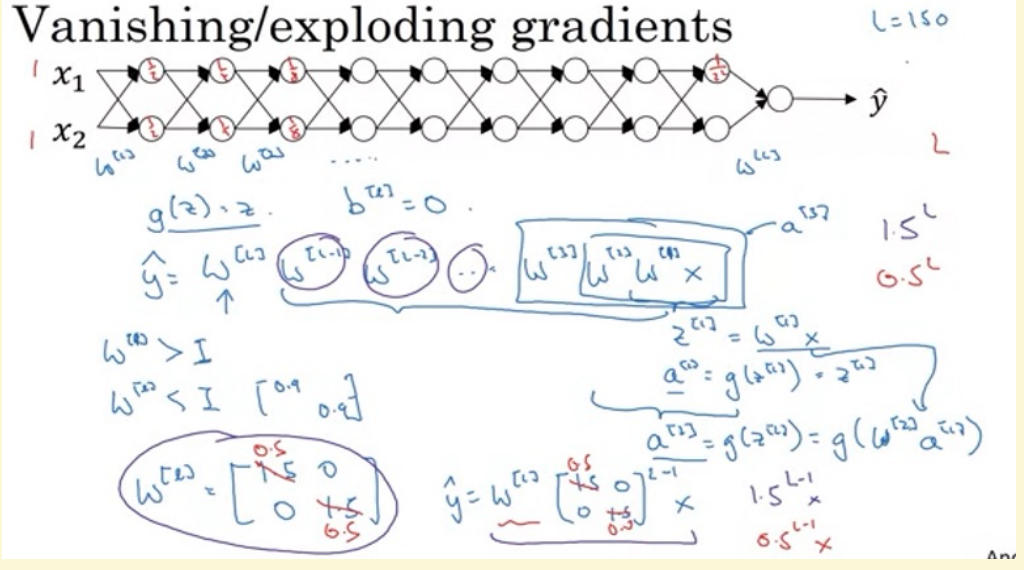




Use same σ & μ to normalise the test & train set.



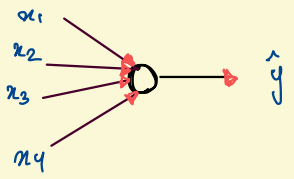
✱ Vanishing / Exploding Gradients: when training neural network, when it's very deep slopes can sometimes get either very big or very very small



If the $W^{[l]} > I$ then the activation o/p & hence the gradient can explode & if the $W^{[l]} < I$ then activation o/p & hence grad. can vanish.

Careful choice of initializing weights might help → solution

* Choice of Weight Initialization for the Neural Network.



$z = w_1 x_1 + w_2 x_2 + \cdots + w_N x_N + b$

larger n, smaller w_i

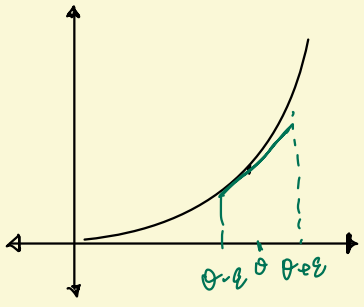
$Var(w) := \frac{1}{n}$ n = # of inputs

In case of ReLU act. func: $Var(w) = \frac{2}{n}$

$w^{[l]} = np.random.randn(shape) * np.sqrt\left(\frac{2}{n^{[l-1]}}\right)$

In case of tanh, one can use $Var(w) = \frac{1}{n^{[l-1]}}$ ← Xavier initialization.

* Numerical Approximation of Gradients.



$$\frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon} \approx g(\theta)$$

$$\frac{(1.01)^3 - (0.99)^3}{2(0.01)} \approx 3.0001 \approx 3$$

$$g(\theta) = 3\theta^2 = 3$$

Approx. error: 0.0001

Gradient Checking: For each i, $d\theta_{approx}[i] = \frac{J(\theta_1, \theta_2, .., \theta_i+\varepsilon, ..) - J(\theta_1, \theta_2, ..., \theta_i-\varepsilon, ...)}{2\varepsilon}$

$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i}$

Check $\frac{\| d\theta_{approx} - d\theta \|_2}{\| d\theta_{approx} \|_2 + \| d\theta \|_2}$ $\varepsilon = 10^{-7}$

* Optimization Algorithms:

Mini Batch Gradient Descent

Vectorization allows you to efficiently compute on m examples.

what if m was large? m = 5 million = 5,000,000

Split the training data into mini Batches into 1000 eg each.

$X = [\, x^{(1)} \; x^{(2)} \; \dots \; x^{(1000)} \mid x^{(1001)} \; \dots \; x^{(2000)} \mid \dots \mid x^{(m)} \,]$

$[n, m]$

$X^{\{1\}}$ $Y^{\{1\}}$ — $X^{\{2\}}$ $Y^{\{2\}}$ — $X^{\{5000\}}$ $Y^{\{5000\}}$

Batch: process entire training set @ the same time.

MiniBatch: process a mini batch @ one time.

Mini Batch

for $t = 1, \dots, 5000$

Forward Prop on $X^{\{t\}}$

$$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$\vdots$$
$$A^{[L]} = g^{[L]}(Z^{[L]})$$

Compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{l} \| W^{[l]} \|_F^2$

Back prop to wrt $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$)

$W^{[l]} := W^{[l]} - dW^{[l]}$, $b^{[l]} := b^{[l]} - db^{[l]}$

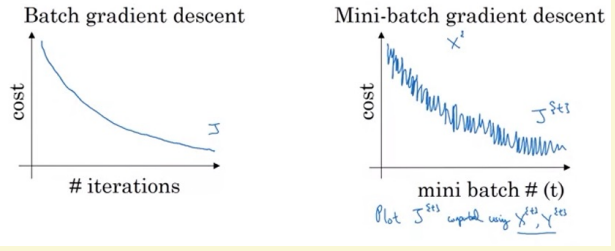
} Mini Batch gradient Descent runs much faster than the Batch gradient descent when size of training set is large.

Choosing Batches:

If small train set use Batch gradient descent ($m \leq 2000$)

Typical mini Batch sizes: 64 (2^6) 128 (2^7) 256 (2^8) 512 (2^9)

Training with mini batch gradient descent

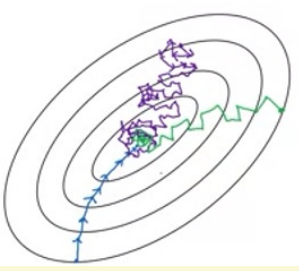


Choosing your mini-batch size

→ If mini-batch size = m : Batch gradient descent. $(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$.

→ If mini-batch size = 1 : Stochastic gradient descent. Every example is it own mini-batch.

In practice: Somewhere in between 1 and m



Stochastic gradient descent
↓
Lose speedup from vectorization

In-between (mini-batch size not too big/small)
↓
Fastest learning.
- Vectorization.
- Make progress without processing entire training set.

Batch gradient descent (mini-batch size = m)
↓
Too long per iteration

Andrew N

* Exponentially weighted averages.