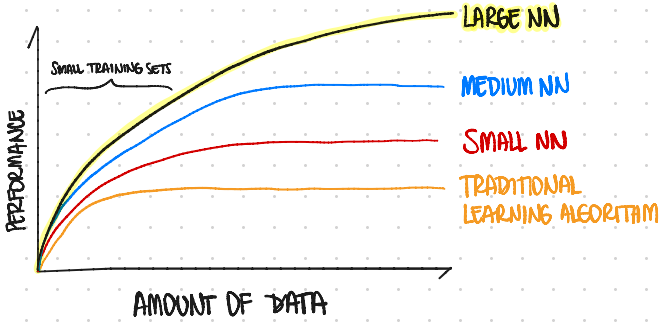
NEURAL NETWORKS AND DEEP LEARNING

COURSE I

WHAT IS A NEURAL NETWORK?

* Machine learning
* A computer learns to perform a task by analizing training samples.
* Consists of thousands or even millions of simple processing nodes that are interconnected.



why is deep learning taking off? →

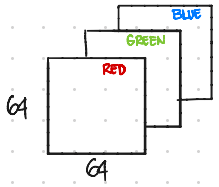
SCALE DRIVES DEEP LEARNING PROGRESS

INPUT (x)	OUTPUT (Y)	APPLICATION	
home features	price	real estate	STANDARD NN
ad, user info	click on ad? (0/1)	online advertising	STANDARD NN
image	object (1,...,1000)	photo tagging	→ CNN
audio	text transcript	speech recognition	→ RNN
image, radar info	position of other cars	autonomous driving	→ CUSTOM / HYBRID

LOGISTIC REGRESSION AS A NEURAL NETWORK

* logistic regression is an algorithm for binary classification

1 (cat)
0 (no cat) ← image classification



$X = \begin{bmatrix} \\ \\ \\ \end{bmatrix} = 64 \times 64 \times 3 = 12288$

m → # TRAINING SAMPLES

#1. given x, we want to find $\hat{y}$ → PREDICTION (OUTPUT)

$$\hat{y} = P(y=1 \mid x)$$

#2. parameters → $w \in \mathbb{R}^{n_x}$
$b \in \mathbb{R}^{n_x}$ (inter spectrum)

#3. output → result of a sigmoid function

$$\hat{y} = \sigma(\underbrace{w^T x + b}_{z})$$

NOTATION!

#1 (x, y), $x \in \mathbb{R}^{n_x}$ $\vee$ $y \in \mathbb{R}$

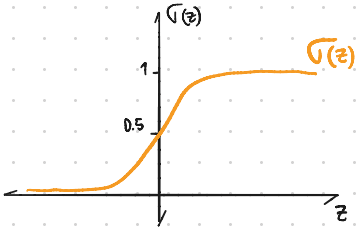
#2 m → training samples

$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

#3 $X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix}$ n_x ↓

m →

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



* z LARGE:
$\sigma(z) \approx \frac{1}{1+0} = 1$

* z LARGE NEGATIVE:
$\sigma(z) \approx \frac{1}{1+\infty} = 0$

* to train the parameters (w) and (b) we need to define a cost function.

$$\mathcal{L}(\hat{y}, y) = -\left[y \log \hat{y} + (1-y) \log (1-\hat{y})\right]$$

HOW GOOD IS OUR OUTPUT $\hat{y}$?

LOSS FUNCTION (ONE DATA SAMPLE)

$\neq$

(APPLIED TO ALL PARAMETERS)

COST FUNCTION

$y=1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y} \Rightarrow \log \hat{y}$ SMALL, $\hat{y}$ LARGE

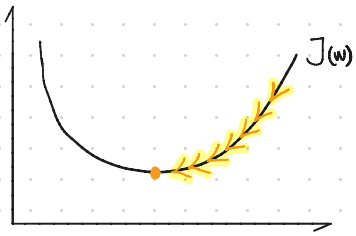
$y=0$: $\mathcal{L}(\hat{y}, y) = -\log (1-\hat{y}) \Rightarrow \log (1-\hat{y})$ LARGE, $\hat{y}$ SMALL

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1-\hat{y}^{(i)}) \log (1-\hat{y}^{(i)})\right]$$

how can we find (w) and (b) that minimizes $J(w,b)$?

GRADIENT DESCENT!

(DIRECTION OF STEEPEST DESCENT)



* you initialize (w) and (b) at some random point, and takes steps towards the deepest descent until you get close to the minimum.

this mostly works with logistic regression only

REPEAT {

$$w := w - \alpha \frac{dJ(w)}{dw}$$

}

$J(w,b)$ {

$$w = w - \alpha \frac{dJ(w,b)}{dw}$$

$$b = b - \alpha \frac{dJ(w,b)}{db}$$

* computation of neural networks:

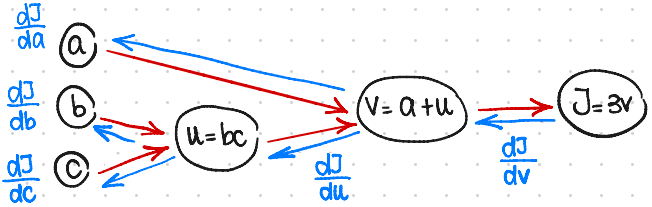
(#1) forward propagation → compute output of the neural network

(#2) backward propagation → compute gradients

* example:

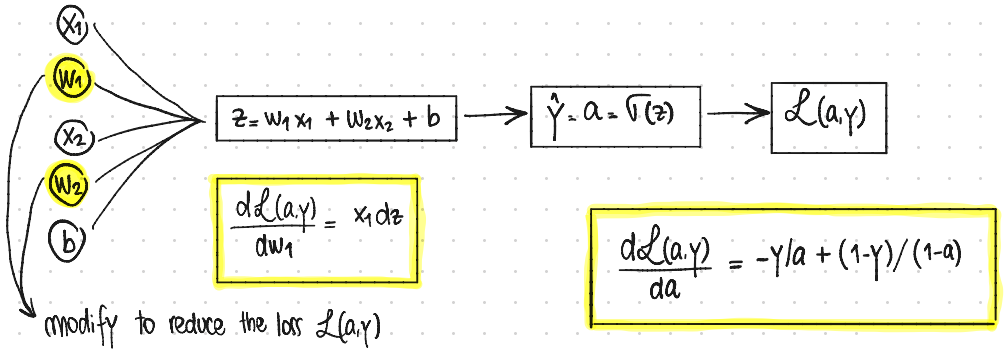
$J(a,b,c) = 3(a+bc)$

$u = bc$, $v = a+b$, $J = 3v$



FORWARD PROPAGATION
BACKWARD PROPAGATION

* in logistic regression...

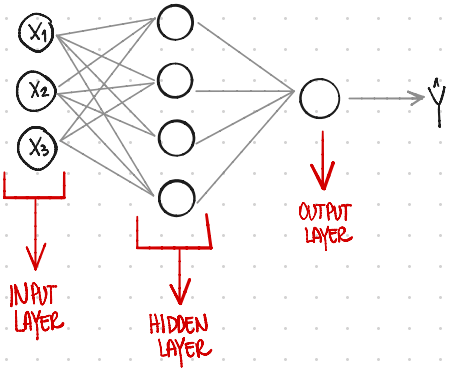


* gradient descent on (m) examples:

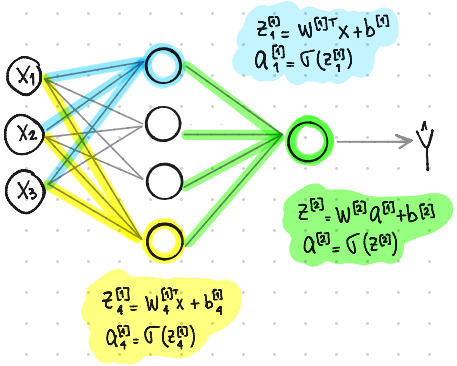
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y)$$

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

SHALLOW NEURAL NETWORKS



* how to compute a NN's output?



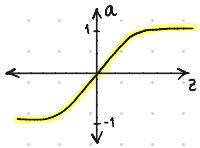
* vectorizing across multiple examples:

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

$A^{[1]} =$ [ ] HIDDEN UNITS
TRAIN SAMPLES

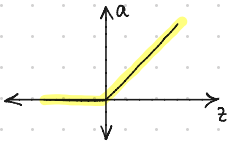
* activation functions:

(#1). $a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$



* always works better than $\sigma(z)$ for hidden units because the mean of the activation functions that come out from the hidden layers are close to 0.

(#2). ReLU $a = \max(0, z)$



* the derivative is 1 so long as z is positive, and it's 0 when z is negative.

(#3). leaky ReLU $a = \max(0.001z, z)$

(#4). linear activation function $g(z) = z$

* it might be useful for the output layer of a linear regression problem.

DOWNSIDE OF $\sigma(z)$ AND $\tanh(z)$:

* if z is either very large or very small, the gradient (derivative) becomes very small. It may end up being close to 0
  → this can slow down gradient descent.

* derivatives of activation functions:

(#1). Sigmoid activation function:

$$\frac{dg(z)}{dz} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) = g(z)(1-g(z))$$

$$a(1-a) = g'(z)$$

(#2). tanh activation function:

$$\frac{dg(z)}{dz} = 1 - [\tanh(z)]^2$$

$$1 - a^2 = g'(z)$$

(#3). ReLU and leaky ReLU:

$$g'(z) \begin{cases} 0 \text{ if } z<0 \\ 1 \text{ if } z \geq 0 \end{cases}$$ ReLU

$$g'(z) \begin{cases} 0.01 \text{ if } z<0 \\ 1 \text{ if } z \geq 0 \end{cases}$$ leaky ReLU

GRADIENT DESCENT FOR NEURAL NETWORKS

* parameters → $w^{[1]}$, dim $n^{[1]}, n^{[0]}$
  → $b^{[1]}$, dim $n^{[1]}, 1$
  → $w^{[2]}$, dim $n^{[2]}, n^{[1]}$
  → $b^{[2]}$, dim $n^{[2]}, 1$

* cost function → $J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}, y)$

* forward propagation:

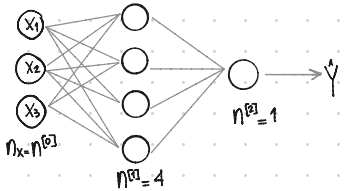
$Z^{[1]} = W^{[1]} X + b^{[1]}$
$A^{[1]} = g^{[1]}(Z^{[1]})$
$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$
$A^{[2]} = g^{[2]}(Z^{[2]})$

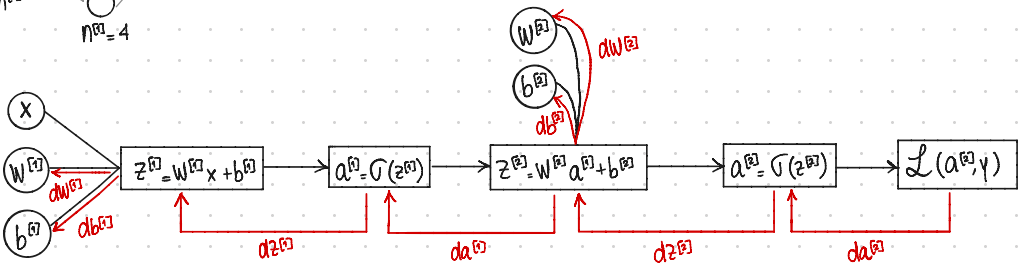
* back propagation:

$dZ^{[2]} = A^{[2]} - Y$
$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$
$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis=1, keepdims=True)$
$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g'^{[1]}(Z^{[1]})$
$dW^{[1]} = \frac{1}{m} dZ^{[1]} X.T$
$db^{[1]} = \frac{1}{m} np.sum(dZ^{[2]}, axis=1, keepdims=True)$

BACKPROPAGATION INTUITION

* for a given NN of form:





$dZ^{[1]} = W^{[2]T} dZ^{[2]} g'^{[1]}(Z^{[1]})$
$dW^{[1]} = dz^{[1]} X^T$
$db^{[1]} = dz^{[1]}$

dimensions:

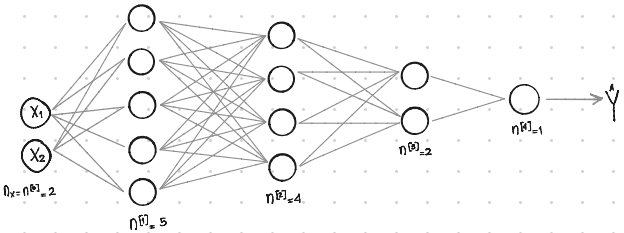
$dZ^{[1]} = W^{[2]T} \; dZ^{[2]} \; * \; g^{[1]'}(Z^{[1]})$
$(n^{[1]}, 1) = (n^{[2]}, n^{[1]}) \; (n^{[2]}, 1) \; * \; (n^{[1]}, 1)$

$dZ^{[2]} = a^{[2]} - y$
$dW^{[2]} = dz^{[2]} a^{[1]T}$
$db^{[2]} = dz^{[2]}$

RANDOM INITIALIZATION

* weights initialized with zeros ⟶ all hidden units compute the same operation or the same function.
* initialize weights randomly

DEEP NEURAL NETWORKS



$$Z^{[1]} = W^{[1]} \times + b^{[1]}$$
$$(n^{[1]}, 1) = (n^{[1]}, n^{[0]})\,(n^{[0]}, 1) + (n^{[1]}, 1)$$
$$(5,1) \quad (5,2) \quad (2,1) \quad (3,1)$$

$$W^{[l]} = (n^{[l]}, n^{[l-1]})$$
$$b^{[l]} = (n^{[l]}, 1)$$

* vectorization:

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$
$$(n^{[1]}, m) = (n^{[1]}, n^{[0]})\,(n^{[0]}, m) + (n^{[1]}, 1)$$

* building blocks of deep neural networks: