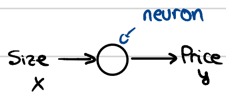
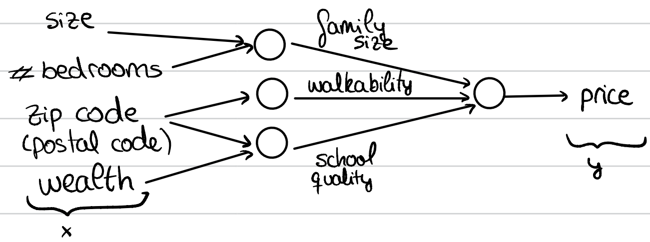
INTRODUCTION TO DEEP LEARNING

What is a Neural Network

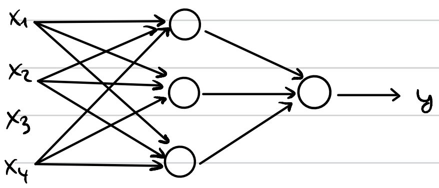
Deep learning → Training Neural Networks

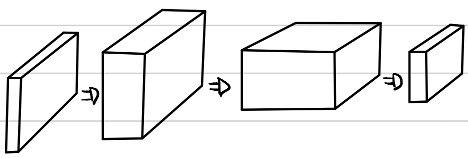


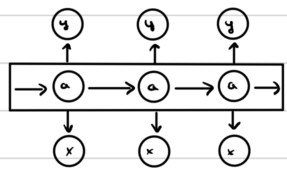


Supervised Learning with Neural Networks

Input (x)	Output (y)	Application	Method
Home features	Price	Real state	standard NN
Ad, user info	Click on ad? (0/1)	Online advertising	
Image	Object (1,...,1000)	Photo tagging	CNN ~ Convolution NN
Audio	Text transcript	Speech recognition	RNN ~ Recurrent NN
English	Chinese	Machine translation	
Image, Radar Info	Position	Autonomous driving	Custom/hybrid NN

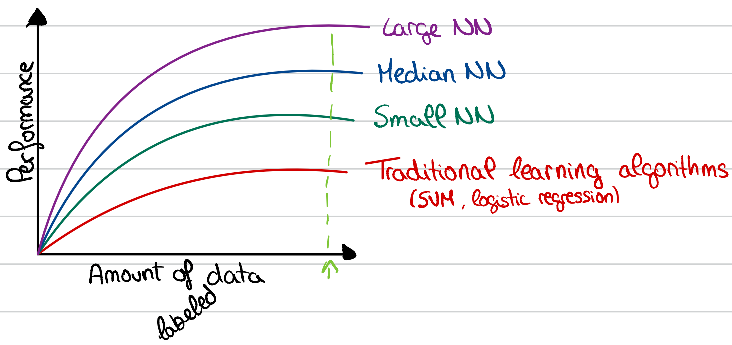






Standard NN | Convolutional NN | Recurrent NN

Why is Deep Learning taking off?



LOGISTIC REGRESSION AS A NEURAL NETWORK

Binary Classification

Output label (y) is 1 or 0

Notation

(x,y) $x \in \mathbb{R}^{n_x}$, $y \in \{0,1\}$

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

$m = m_{train}$ $\quad$ $m_{test} = \#$ test examples

$$X = \underbrace{\begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}}_{m} \updownarrow n_x \qquad X \in \mathbb{R}^{n_x \times m}$$

↳ Python → X.shape(n_x, m)

$$y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}] \qquad y \in \mathbb{R}^{1 \times m}$$

↳ Python → Y.shape $(1,m)$

Logistic Regression

Given x, want $\hat{y} = P(y=1|x)$ → $0 \le \hat{y} \le 1$

$x \in \mathbb{R}^{n_x}$ → Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

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

↳ sigmoid function $\quad \sigma(z) = \dfrac{1}{1+e^{-z}}$

If z large $\sigma(z) \approx \frac{1}{1+0} = 1$

If z large negative $\sigma(z) \approx \frac{1}{1+\infty} = 0$

Logistic Regression cost function

$$L(\hat{y},y) = -(y \log \hat{y} + (1-y)\log(1-\hat{y}))$$

If $y=1$: $L(\hat{y},y) = -\log \hat{y}$ → want $\hat{y}$ large

If $y=0$: $L(\hat{y},y) = -\log(1-\hat{y})$ → want $\hat{y}$ small

Cost function

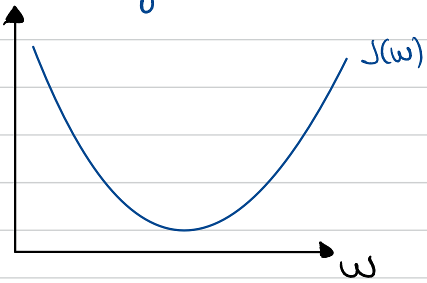
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})]$$

Loss function → Error for a single training example

Cost function → Average of the loss functions of the entire training set

Gradient Descent

Want to find w, b that minimize $J(w,b)$



learning rate

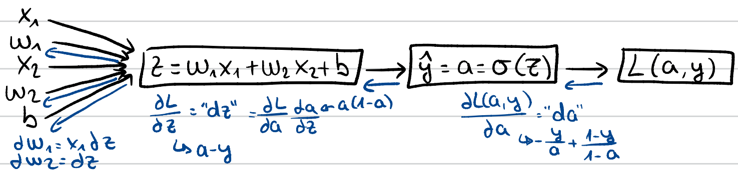
Repeat {

$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$ dw ~ slope

$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$ db

}

Logistic Regression Gradient Descent



Logistic regression ⇒ modify parameters (w and b) in order to minimize the loss

Gradient Descent on m Examples

Initialize ⇒ $J=0, dw_1=0, dw_2=0, db=0$

for $i=1$ to m {

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

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

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

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

$\left.\begin{array}{l} dw_1 += x_1^{(i)} dz^{(i)} \\ dw_2 += x_2^{(i)} dz^{(i)} \end{array}\right\} n=2$

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

}

$J /= m$

$dw_1 /= m$; $dw_2 /= m$; $db /= m$

Update:

$w_1 := w_1 - \alpha dw_1$

$w_2 := w_2 - \alpha dw_2$

$b := b - \alpha db$

PYTHON AND VECTORIZATION

Vectorization

$z = w^T x + b$ → Python → $z = np.dot(w, x)$

Vectorizing Logistic Regression

$z^{(1)} = w^T x^{(1)} + b$ $\quad z^{(2)} = w^T x^{(2)} + b$ $\quad z^{(3)} = w^T x^{(3)} + b$

$a^{(1)} = \sigma(z^{(1)})$ $\quad a^{(2)} = \sigma(z^{(2)})$ $\quad a^{(3)} = \sigma(z^{(3)})$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}_{n_x, m}$$

$Z = [z^{(1)}, z^{(2)}, \dots, z^{(m)}] = w^T X + [b, b, \dots, b] = [w^T x^{(1)} + b, w^T x^{(2)} + b, \dots, w^T x^{(3)} + b]$

for iter in range(1000):

↳ python → $Z = np.dot(w.T, X) + b$ "broadcasting"

$A = [a^{(1)}, a^{(2)}, \dots, a^{(m)}] = \sigma(Z)$

Vectorizing Logistic Regression's Gradient Output

$dz^{(1)} = a^{(1)} - y^{(1)}$ $\quad dz^{(2)} = a^{(2)} - y^{(2)}$

$dZ = [dz^{(1)}, dz^{(2)}, \dots, dz^{(m)}]_{1,m}$

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

$dZ = A - Y$

$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)}$ → Python → $\frac{1}{m}$ np.sum(dZ)

$dw = \frac{1}{m} X dZ^T = \frac{1}{m} \begin{bmatrix} | & & | \\ x^{(1)} & \dots & x^{(m)} \\ | & & | \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix} = \frac{1}{m} [x^{(1)} dz^{(1)} \dots x^{(m)} dz^{(m)}]$

$w := w - \alpha dw$

$b := b - \alpha db$

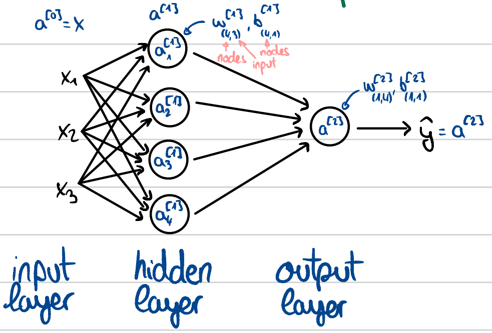
Broadcasting in Python

$ab + ac - b - c$

(m,n) $\begin{matrix} + \\ - \\ * \\ / \end{matrix}$ $\begin{matrix} (1,n) \sim (m,n) \\ (m,1) \sim (m,n) \\ \mathbb{R} \sim (m,n) \end{matrix}$

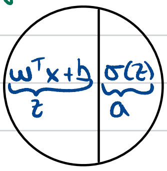
SHALLOW NEURAL NETWORKS

Neural Networks Representation



2 layer NN
└ layer 0 – input layer

Computing a Neural Network's Output



$a_i^{[l]}$ ← layer, ← node in layer

Vectorizing Across Multiple Examples

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

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

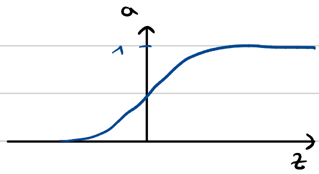
training examples

$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$ $\quad Z = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$ $\quad A^{[1]} = \begin{bmatrix} | & | & | \\ a^{[1](1)} & a^{[1](2)} & a^{[1](m)} \\ | & | & | \end{bmatrix}$ hidden units

Activation functions

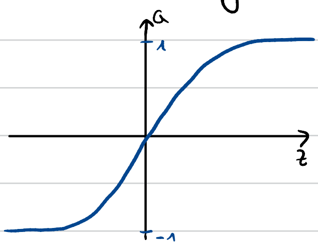
Can be different for different layers

Sigmoid



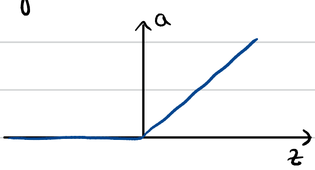
$a = \frac{1}{1+e^{-z}}$

Hyperbolic tangent



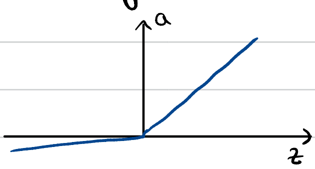
$a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Rectified Linear Unit (ReLU)



$a = \max(0, z)$

Leaky ReLU



$a = \max(0.01z, z)$

Derivatives of Activation Functions

Sigmoid

$$g(z)=\frac{1}{1+e^{-z}} \rightarrow g'(z)=\frac{\partial}{\partial z}g(z)=\frac{1}{1+e^{-z}}\left(1-\frac{1}{1+e^{-z}}\right)=g(z)(1-g(z))=a(1-a)$$

Hyperbolic tangent

$$g(z)=\tanh(z)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \rightarrow g'(z)=\frac{\partial}{\partial z}g(z)=1-(\tanh(z))^2=1-g(z)^2=1-a^2$$

ReLU

$$g(z)=\max(0,z) \rightarrow g'(z)=\begin{cases}0 & \text{if } z<0\\ 1 & \text{if } z\geq 0\end{cases}$$

Leaky ReLu

$$g(z)=\max(0.01z,z) \rightarrow g'(z)=\begin{cases}0.01 & \text{if } z<0\\ 1 & \text{if } z\geq 0\end{cases}$$

Gradient Descent for Neural Networks (2 layers)

Parameters:

$$\underset{(n^{[1]},n^{[0]})}{W^{[1]}},\ \underset{(n^{[1]},1)}{b^{[1]}},\ \underset{(n^{[2]},n^{[1]})}{W^{[2]}},\ \underset{(n^{[2]},1)}{b^{[2]}}$$

$$n_x=n^{[0]},\ n^{[1]},\ n^{[2]}=1$$

Cost function (binary classification)

$$J(W^{[1]},b^{[1]},W^{[2]},b^{[2]})=\frac{1}{m}\sum_{i=1}^{n}\mathcal{L}(\underbrace{\hat{y},y}_{a^{[2]}})$$

Gradient descent

Repeat {

Compute predicts $(\hat{y}^{(i)}, i,\dots,m)$

$dW^{[1]}=\frac{\partial J}{\partial W^{[1]}}$, $db^{[1]}=\frac{\partial J}{\partial b^{[1]}}$ ...

$W^{[1]}=W^{[1]}-\alpha\, dW^{[1]}$

$b^{[1]}=b^{[1]}-\alpha\, db^{[1]}$

$W^{[2]}=W^{[2]}-\alpha\, dW^{[2]}$

$b^{[2]}=b^{[2]}-\alpha\, db^{[2]}$

}

Forward propagation

$Z^{[1]}=W^{[1]}X+b^{[1]}$

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

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

$A^{[2]}=g^{[2]}(Z^{[2]})$

Backward propagation

$dZ^{[2]}=A^{[2]}-Y$ a me funciona con $y-A^{[2]}$

$dW^{[2]}=1/m\, dZ^{[2]}A^{[1]T}$

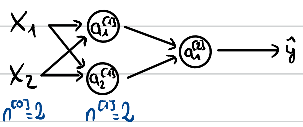
$db^{[2]}=1/m$ np.sum($dZ^{[2]}$, axis=1, keepdims=True)

$dZ^{[1]}=\underbrace{W^{[2]T}dZ^{[2]}}_{dA^{[1]}} * g^{[1]\prime}(Z^{[1]})$ element-wise product

$dW^{[1]}=1/m\, dZ^{[1]}X^{T}$

$db^{[1]}=1/m$ np.sum($dZ^{[1]}$, axis=1, keepdims=True)

Random Initialization



Zero initialization → Symmetry → After every iteration, the two hidden units are still computing exactly the same function.

$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ $b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

↳ $a_1^{[1]} = a_2^{[1]}$ ↳ $dz_1^{[1]} = dz_2^{[1]}$

Random initialization → Avoids the symmetry

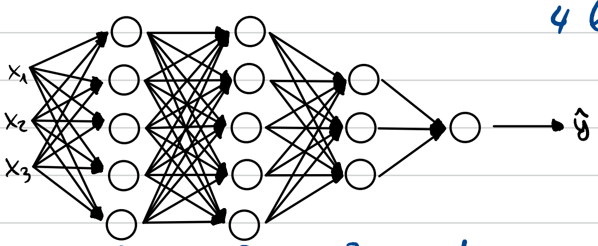
$W^{[1]} = $ np.random.randn((2,2)) *0.01
$b^{[1]} = $ np.zero((2,1))
$W^{[2]} = $ np.random.randn((1,2)) *0.01
$b^{[2]} = 0$

DEEP NEURAL NETWORK

Deep L-layer Neural Network

layer: 0 1 2 3 4

4 layer NN



elements: 5 5 3 1

$L = 4 \sim$ # layers
$n^{[l]} =$ # nodes/units in layer l
$a^{[l]} =$ activations in layer l
$a^{[l]} = g^{[l]}(z^{[l]})$
$W^{[l]} =$ weights for $z^{[l]}$
$b^{[l]}$

$x = a^{[0]}$
$y = a^{[L]}$

$n^{[1]} = 5$ $n^{[3]} = 3$ $n^{[0]} = n_x = 3$
$n^{[2]} = 5$ $n^{[4]} = n^{[L]} = 1$

shallow NN
deep NN

Forward propagation in a Deep Network

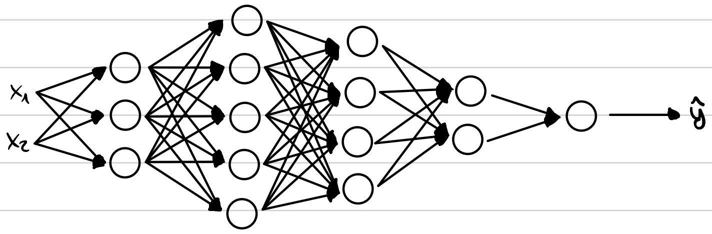
$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$
$A^{[l]} = g^{[l]}(Z^{[l]})$ ↓ for $l = 1:L$

All are column vectors stacked horizontally

Getting your Matrix Dimensions Right

Parameters $W^{[l]}$ and $b^{[l]}$

layer: 0 1 2 3 4 5



elements: 3 5 4 2 1

$\underset{(n^{[l]}, m)}{Z^{[l]}} = \underset{(n^{[l]}, n^{[l-1]})}{W^{[l]}} \; \underset{(n^{[l-1]}, m)}{a^{[l-1]}} + \underset{(n^{[l]}, 1)}{b^{[l]}}$ ← broadcasting

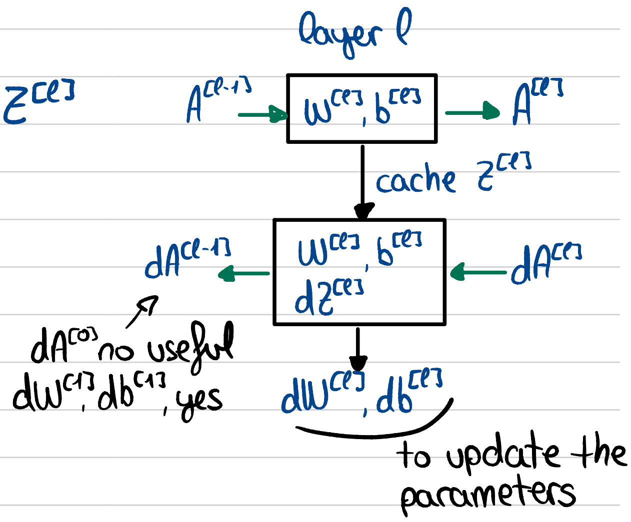
$dW^{[l]}$ $(n^{[l]}, n^{[l-1]})$
$db^{[l]}$ $(n^{[l]}, 1)$

Why Deep representations

Circuit theory and deep learning (informally) → There are functions you can compute with a "small" L-layer deep neural network that shallower networks require exponentially more hidden units to compute.

Building blocks of Deep Neural networks

Layer l: $W^{[l]}, b^{[l]}$
Forward: Input $A^{[l-1]}$, Output $A^{[l]}$
$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$ → cache the value of $Z^{[l]}$
$A^{[l]} = g^{[l]}(Z^{[l]})$
Backward: Input $dA^{[l]}$, Output $dA^{[l-1]}$
(also) $dW^{[l]}, db^{[l]}$



Parameters vs Hyperparameters

Parameters: $W^{[l]}$, $b^{[l]}$

Hyperparameters: α, # iterations, # hidden layers (L), # hidden units ($n^{[l]}$), choice of activation function (sigmoid, ReLU, tanh)

To choose the correct α: