Week 1

Friday, August 25, 2017 8:47 AM

Specialization Outline

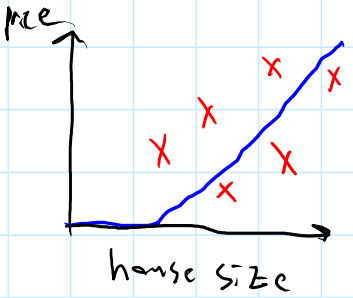
Courses

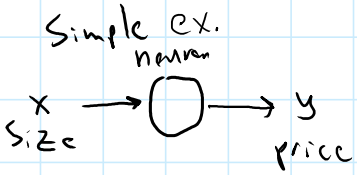
1) Neural Networks & Deep Learning (This one)
2) Improving Deep NNs: Hyperparameter Tuning, Regularization, Optimization
3) Structuring an ML project
4) Convolutional NNs
5) Natural Language Processing: Building Sequence Models

Intro to Deep Learning

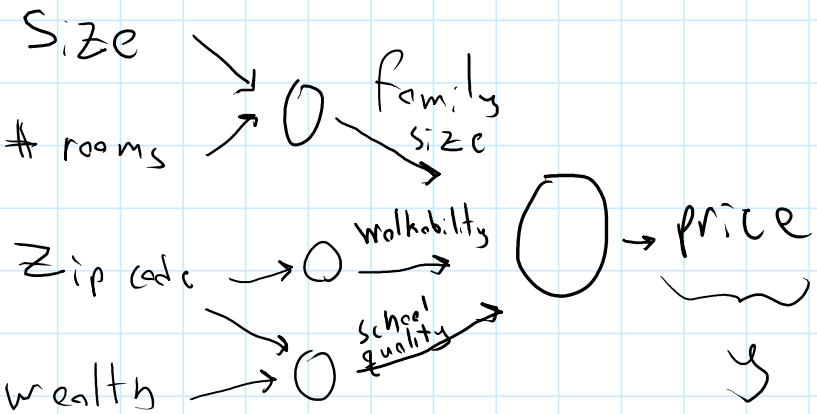
Friday, August 25, 2017 9:02 AM

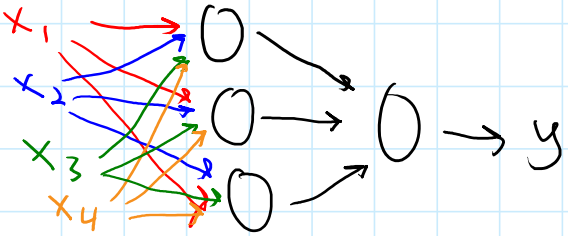
What is a Neural Network?





Rectified Linear Unit



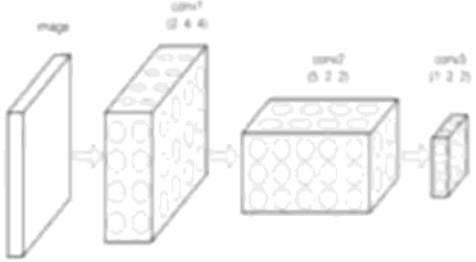


Supervised Learning w/ Neural Networks

Examples

Input (x)	Output (y)	Application	
Home features	Price	Real estate	} Std. NN
Ad, user info	Click (0/1)	Ad	

Home features	Price	Real estate } Std. NN
Ad, user info	Click? (0,1)	Ads } Std. NN
Image	Object (1, ..., 1000)	Photo tagging } CNN
Audio	Text	Speech recognition } RNN
English	Chinese	Machine Translation } RNN
Image, Radar	Position of other cars	Autonomous Driving } Custom / hybrid



Standard NN **Convolutional NN** **Recurrent NN**

Structured Data

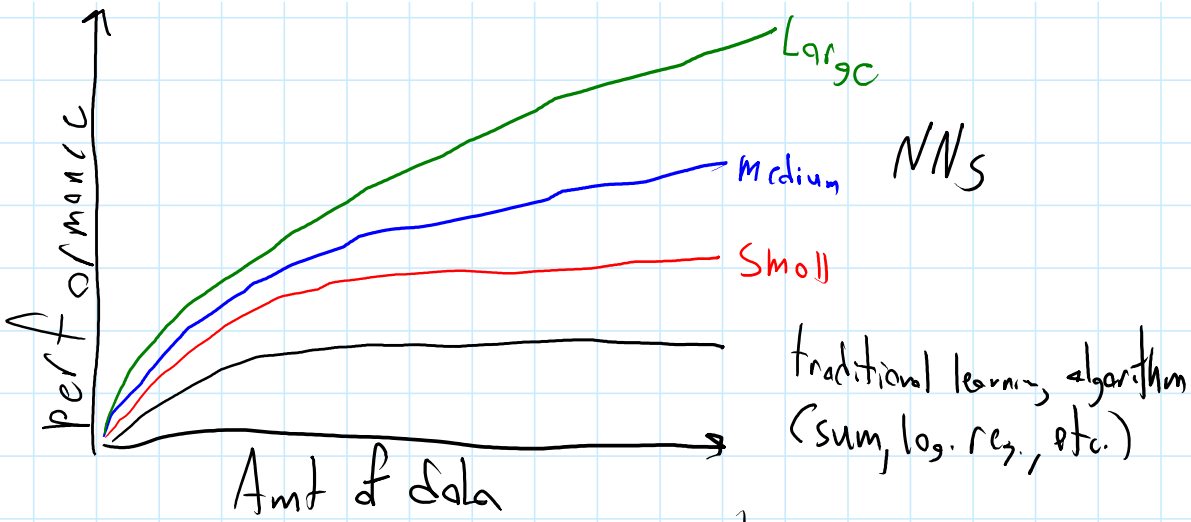
Tables

Unstructured Data

Audio Images

Text

Why have NNs become so popular



Employing ReLU instead of Sigmoids in processing has sped up training

Week 2

Monday, August 28, 2017 9:41 AM

Basics of NN programming

Vectorization
Forward & back propagation

Logistic Regression as a Neural Network

Monday, August 28, 2017 9:42 AM

Binary Classification

Image ⟶ cat? (0 or 1)

an $n \times m$ picture has 3 (R, G, B) $n \times m$ matrices

unroll into 1 feature vector x $\begin{bmatrix} x_R(:) \\ \vdots \\ x_G(:) \\ x_B(:) \end{bmatrix}$ $x \in \mathbb{R}^{1 \times (n \cdot m \cdot 3)}$

Notation

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

m training examples: $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ python notation

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad n_x \text{ rows}, \; m \text{ columns} \qquad X \in \mathbb{R}^{n_x \times m} \qquad X.shape = (n_x, m)$$

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix} \qquad Y \in \mathbb{R}^{1 \times m} \qquad Y.shape = (1, m)$$

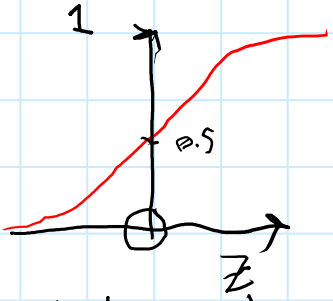
Logistic Regression

Given x want $\hat{y} = P(y = 1 \mid x)$ ($\hat{y}$: prediction) $x \in \mathbb{R}^{n_x}$

Given x, want $\hat{y} = P(y=1|x)$ ← prediction $\quad x \in \mathbb{R}^{n_x}$

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

Output: $\hat{y} = \sigma(w^T x + b)$ where $\sigma(z) = \frac{1}{1+e^{-z}}$ ← sigmoid function



Alt Notation: (used in Andrew Ng's ML course, but NOT here)

$x_0 = 1$, $x \in \mathbb{R}^{n_x+1}$, $\hat{y} = \sigma(\theta^T x)$, $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{n_x} \end{bmatrix}$ θ_0 = b, $\theta_1 \ldots \theta_{n_x}$ = w

---

Logistic Regression Cost Function

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

Loss (error function): $\mathcal{L}(\hat{y}, y)$

Sq. error $[\frac{1}{2}(\hat{y}-y)^2]$ struggles to find global minima w/ gradient descent when used w/ logistic regression

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

Intuition: $y=1$: $\mathcal{L}(\hat{y}, y) = -\log(\hat{y})$ ← want $\log \hat{y}$ large, want $\hat{y}$ large

$y=0$: $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$ ← want $\log(1-\hat{y})$ large, want $\hat{y}$ small

Cost function (loss for whole training set):

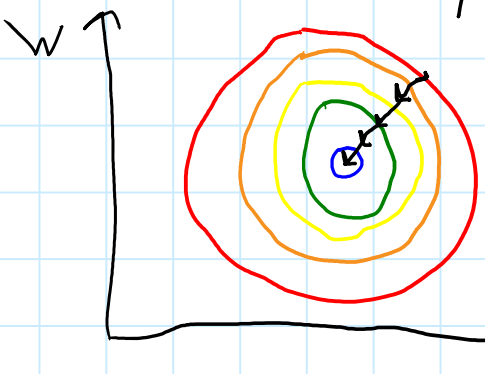
Cost function (loss for whole training set):

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

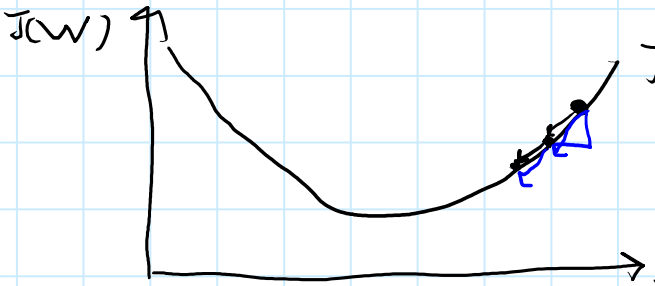
$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log (1-\hat{y}^{(i)})\right]$$

Gradient Descent (learning w & b parameters)

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



initialize @ 0 or random value,
at each step, move in direction of steepest descent



Repeat {

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

learning rate

"dw" in code

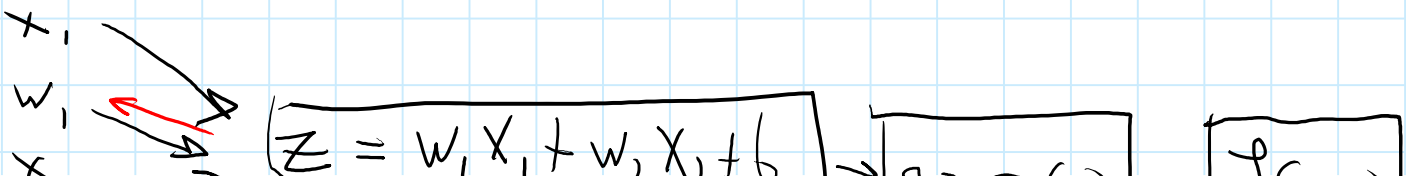
}

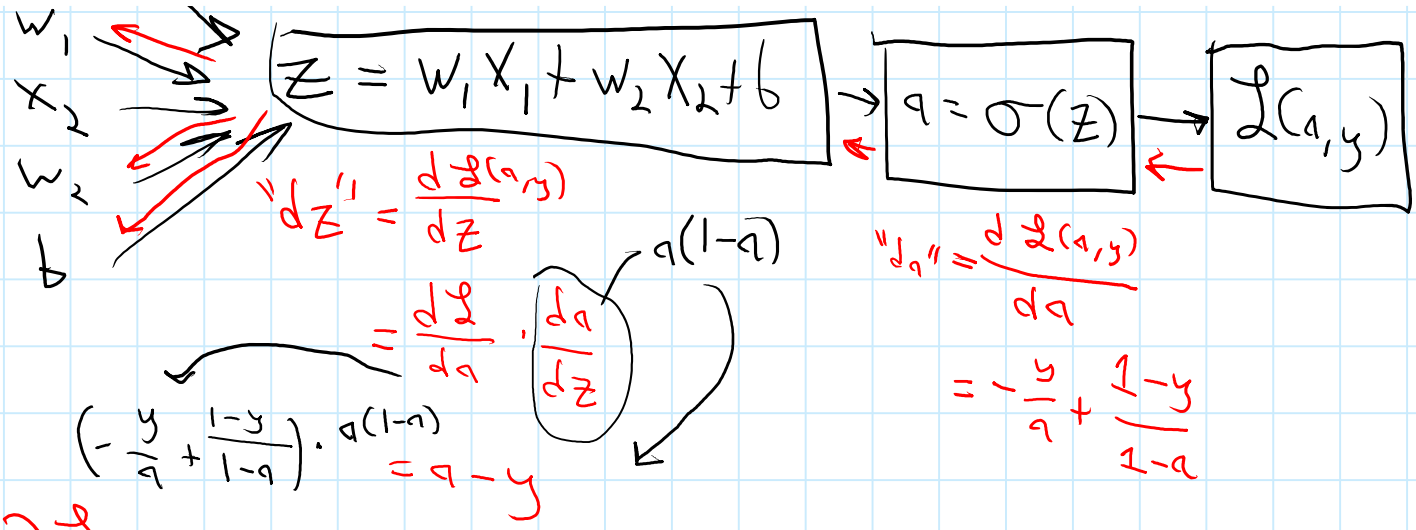
$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$

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

---

Logistic Regression & Gradient Descent





$\frac{\partial \mathcal{L}}{\partial w_n}$ = "dw_n" $= x_n dz$, $db = dz$

Gradient Descent over a Training Set

$$\frac{\partial}{\partial w_1} J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial w_1} \mathcal{L}(a^{(i)}, y^{(i)})$$

$dw_1^{(i)} - (x^{(i)}, y^{(i)})$

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

For i in range$(1, m+1)$:

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

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

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

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

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$ ↑ $n=2$ ↓ Can do as for loop, but inefficient, use vectorization

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

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

$J /= m$

$dw_1 /= m$

$dw_2 /= m$

$db /= m$

Python and Vectorization

Tuesday, August 29, 2017 9:02 AM

Vectorization

$z = w^T x + b$ $\quad w \in \mathbb{R}^{n_x}$

$w = \begin{bmatrix} \\ \\ \end{bmatrix}$ $x = \begin{bmatrix} \\ \\ \end{bmatrix}$ $\quad x \in \mathbb{R}^{n_x}$

```
# Non-vectorized
z = 0
for i in range(n-x):
    z += w[i] * x[i]
z += b
```

```
# Vectorized
z = np.dot(w,x) + b
#        wTx
```

iterating over 10^7 values

~ 450 ms ~ 1.5 ms

vectorization ~ 300 x faster !! Use over for loops whenever poss.

Exponentiation

$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \rightarrow u = \begin{bmatrix} e^{v_1} \\ \vdots \\ e^{v_n} \end{bmatrix}$

```
# non-vectorized
u = np.zeros((n,1))
for i in range(n):
    u[i] = math.exp(v[i])
```

```
#vectorized
u = np.exp(v)
```

Revisiting the Logistic Regression for loop

$dw = np.zeros((n-x,1))$

$J = 0, \; dw1 = 0, \; dw2 = 0, \; db = 0$

For i in range(1, m+1):

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

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

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

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

~~$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$~~

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

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

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

$J /= m$, ~~$dw1 /= m$, $dw2 /= m$,~~ $db /= m$

$dw /= m$

Vectorizing Logistic Regression

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

$\leftarrow m \rightarrow$

$w^T X + [b \cdots b] = [z^{(1)} \; z^{(2)} \cdots z^{(m)}]$

$[w^T x^{(1)} + b \quad w^T x^{(2)} + b \cdots w^T x^{(m)} + b] \Rightarrow$ **`Z = np.dot(w.T, X) + b`**

real number "broadcast" into $1 \times m$ matrix

$A = \sigma(Z)$

Vectorizing Logistic Regression's Gradient Descent

$dZ = [dz^{(1)} \cdots dz^{(m)}]$, $A = [a^{(1)} \cdots a^{(m)}]$, $Y = [y^{(1)} \cdots y^{(m)}]$

$dZ = A - Y$

$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)} \rightarrow \left(\frac{1}{m}\right)$ np.sum(dZ)

$dw = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} dz^{(i)} \rightarrow \left(\frac{1}{m}\right) X\, dZ^T \rightarrow \left(\frac{1}{m}\right)$ np.dot(X, dZ.T)

Implementation of code w/o for loops in process.

```
for iter in range(iterations):
    Z = np.dot(w.T, X) + b
    A = σ(Z)
    dZ = A - Y
    dw = (1/m) np.dot(X, dZ.T)
    db = (1/m) np.sum(dZ)

    w -= α dw
    b -= α db
```

Broadcasting Example

Calorie dist in 100g of diff foods

	Apples	Beef	Eggs	Potatoes
Carb	56.0	0.0	4.4	68.0
Protein	1.2	104.0	52.0	8.0
Fat	1.8	135.0	99.0	0.9

$= A$

Get % of calories of each nutrient class for each food w/

no for loops

```
cal = A.sum(axis=0)   # sum of calories per food
print(cal)   [59.  239.  155.4  76.9]

percentage = 100 * A / cal.reshape(1,4)
print(percentage)
```

(3,4) (1,4)

redundant here, but not costly

$$\begin{bmatrix} 94 & 0 & 2 & 88 \\ 2 & 43 & 33 & 10 \\ 3 & 56 & 63 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + 100 = \begin{bmatrix} 101 \\ 102 \\ 103 \\ 104 \end{bmatrix}$$

broadcasting

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \end{bmatrix}$$

(m,n)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \underset{(1,n)\rightsquigarrow(m,n)}{\begin{bmatrix} 100 & 200 & 300 \\ 100 & 200 & 300 \end{bmatrix}} = \begin{bmatrix} 101 & 202 & 303 \\ 104 & 205 & 306 \end{bmatrix}$$

(m,n)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \underset{(m,1)\rightsquigarrow(m,n)}{\begin{bmatrix} 100 & 100 & 100 \\ 200 & 200 & 200 \end{bmatrix}} = \begin{bmatrix} 101 & 102 & 103 \\ 204 & 205 & 206 \end{bmatrix}$$

General Principle

(m,n) $\begin{matrix} + \\ - \\ \times \\ \div \end{matrix}$ $(1,n) \rightsquigarrow (m,n)$

(m,n) $\begin{matrix} + \\ - \\ \times \\ \div \end{matrix}$ $(m,1) \rightsquigarrow (m,n)$

(m,n) $\begin{matrix} + \\ - \\ \times \\ \div \end{matrix}$ $\mathbb{R} \rightsquigarrow \mathbb{R}$ * np.ones(m,n)

(m, n) $\bar{x}$ $\mathbb{R} \rightsquigarrow \mathbb{R}$ np.ones(m,n)

Note: when implementing arrays, rank 1 (a.shape$=(n,)$) arrays behave in strange ways, use the reshape command to fit calculations

e.g. a = random.randn(5) → array$([a_1, a_2, a_3, a_4, a_5])$

a.reshape$(5,1) = \begin{bmatrix} a_1 \\ \vdots \\ a_5 \end{bmatrix}$ a.reshape$(1,5) = [a_1 \cdots a_5]$

Homework Notes

Thursday, August 31, 2017 9:59 AM

4.3 - Forward & Back prop

$$A = \sigma(w^T X + b) \rightarrow \text{Sigmoid}(\text{np.dot}(w.T, X) + b)$$

$$J = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)})$$

$$\rightarrow -(\text{np.sum}(Y * \text{np.log}(A) + (1 - Y) * \text{np.log}(1 - A))/m$$

$$\frac{\partial J}{\partial w} = \frac{1}{m} X (A - Y)^T \rightarrow \text{np.dot}(X, (A - Y).T)/m$$

$$\partial J / \partial b = \frac{1}{m} \sum_{i=1}^{m} (a^{(i)} - y^{(i)}) \rightarrow \text{np.sum}(A - Y)/m$$

4.3-2 - Optimize

$$\theta := \theta - \alpha d\theta$$

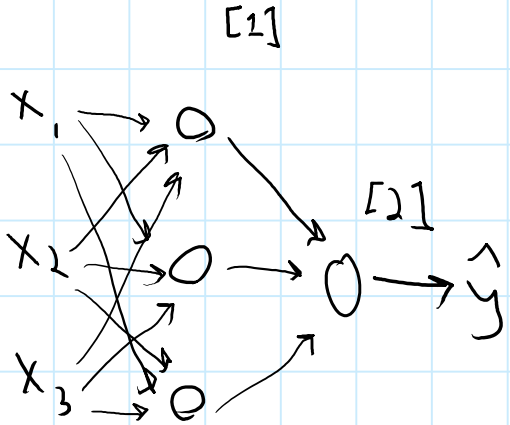
4.3-3 - Predict

Week 3

Wednesday, September 13, 2017 2:53 PM

Shallow NNs

What NNs are

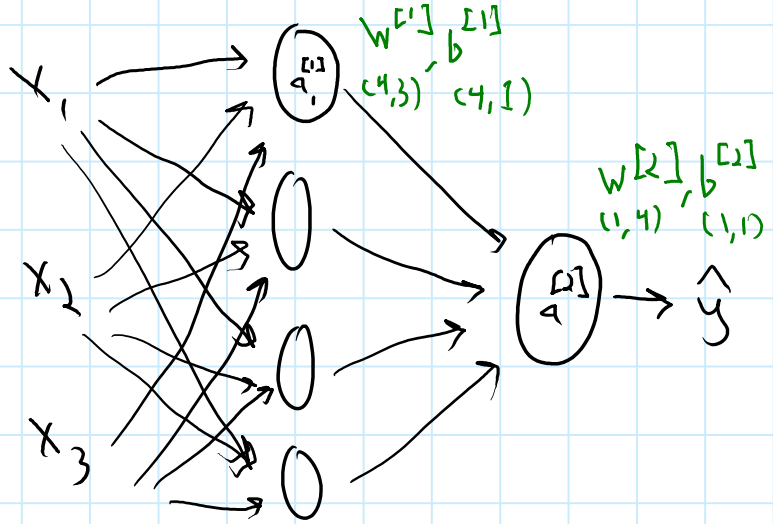


$$W^{[1]} x + b^{[1]} = z^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$W^{[2]} a^{[1]} + b^{[2]} = z^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

Shallow NNs

Wednesday, September 13, 2017 2:54 PM

Neural Network Representation



$$a^{[1]} = \begin{bmatrix} a^{[1]}_1 \\ a^{[1]}_2 \\ a^{[1]}_3 \\ a^{[1]}_4 \end{bmatrix}$$

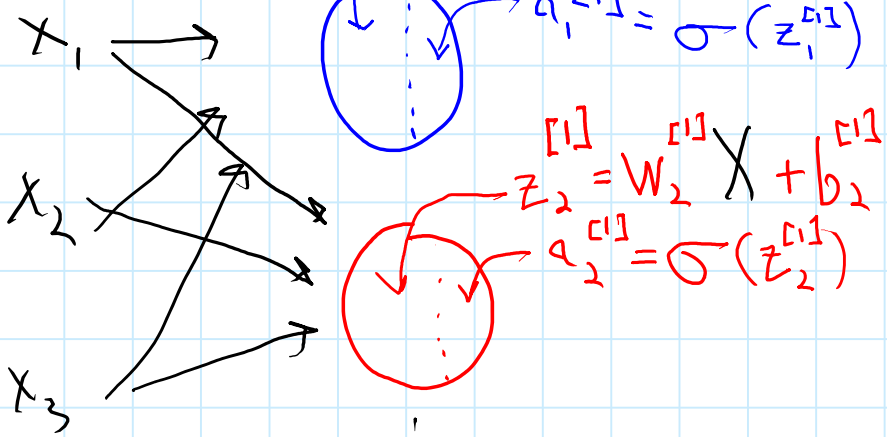
input layer

Hidden layer

output layer

Computing an NN output

$a^{[2]}_i$: 2^{nd} layer, i^{th} node in layer



etc for all 4 nodes

etc for all 4 nodes

Vectorizing

Inefficient! for i in range(0, nodes):

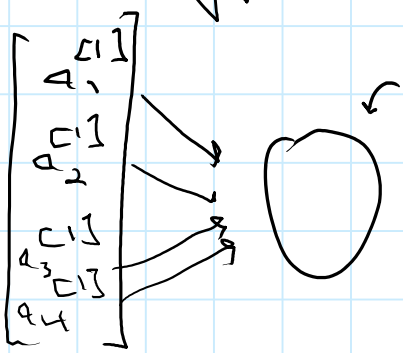
$$z_i^{[1]} = w_i^{[1]T} x + b_i^{[1]}$$
$$a_i^{[1]} = \sigma(z_i^{[1]})$$

Vectorize:

$$\begin{bmatrix} - w_1^{[1]T} - \\ - w_2^{[1]T} - \\ \vdots \\ - w_4^{[1]T} - \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ \vdots \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = z^{[1]}$$

$(4,3)$ $(3,1)$ $(4,1)$

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



$W^{[2]}$ $(1,4)$ $b^{[2]}$ $(1,1)$

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

Vectorizing Over Multiple Examples

Given m training examples

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

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

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

inefficient:

for $i = 1$ to m,

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

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

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

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

Better:

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

$$Z^{[1]} = W^{[1]} X + b^{[1]} = \begin{bmatrix} | & & | \\ z^{[1](1)} & \cdots & z^{[1](m)} \\ | & & | \end{bmatrix}$$

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

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

$$A^{[2]} = \sigma(Z^{[2]}) = \begin{bmatrix} | & & | \\ a^{[2](1)} & \cdots & a^{[2](m)} \\ | & & | \end{bmatrix}$$

Justification for Vectorization

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

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

$z^{[1](3)} = W^{[1]} x^{(3)} + b^{[1]}$

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

$$W^{[1]} X = Z^{[1]} = \begin{bmatrix} | & | & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} \\ | & | & | \end{bmatrix}$$

Activation Functions

Friday, September 15, 2017 10:05 AM

Sigmoid function not always best option

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

$\tanh\left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)$ "centers" mean closer to 0

ReLU ($a = \max(0, z)$)

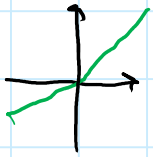
If using diff activation functions for diff layers, label w/ layer

e.g. $a^{[1]} = g^{[1]}(z^{[1]})$, $a^{[2]} = g^{[2]}(z^{[2]})$

General guide to use:

Sigmoid: output is binary (output layer)

ReLU: pretty much everything else, or Leaky ReLU



Why Non-linear Activation functions?

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

$= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'}$

This is no more expressive than logistic regression

Note: if $y \in \mathbb{R}$ (like housing prices) output layer may be ok to be linear

Derivatives of Activation Functions

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

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

ReLU: $g'(z) = \begin{cases} 0 & z < 0 \\ 1 & z > 0 \\ \text{undef} & z = 0 \end{cases}$

Leaky ReLU: $\begin{cases} 0.01 & z < 0 \\ 1 & z > 0 \\ \text{undef} & z = 0 \end{cases}$

Homework Notes

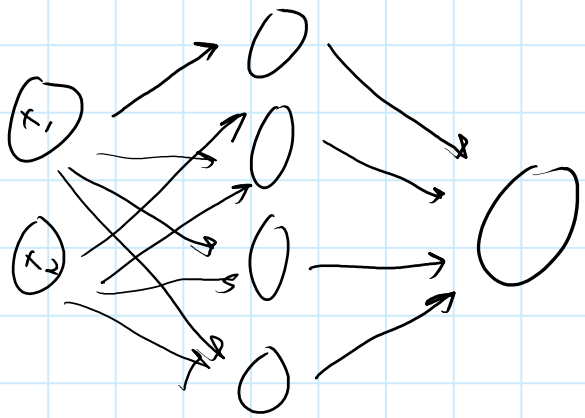
Thursday, September 14, 2017 10:19 AM

Goals: Develop intuition of back-prop

Recognize more hidden layers = more complex ID's,

Build all helper functions

"flower dataset"



$$y^{(i)} \begin{cases} 1 & a^{[2](i)} > 0.5 \\ 0 & \text{else} \end{cases}$$

$$J = -\frac{1}{m}\sum_{i=0}^{m}\left(y^{(i)}\log\left(a^{[2](i)}\right) + \left(1-y^{(i)}\right)\log\left(1-a^{[2](i)}\right)\right)$$

layer_sizes(X, Y):

n_x = size of input layer (parameters)

n_h = # of nodes in hidden layer

n_y = size of output layer

initialize_parameters(n_x, n_h, n_y):

W1 = randoms(n_h, n_x) ← filters, inputs

b1 = zeros(n_h, 1)

W2 = randoms(n_y, n_h)

W2 = random(n_y, n_h)
b2 = zeros(n_y, 1)

forward_prop(X, params): params = W1, b1, W2, b2 as dict entries

# unpack params
params = params["param"]

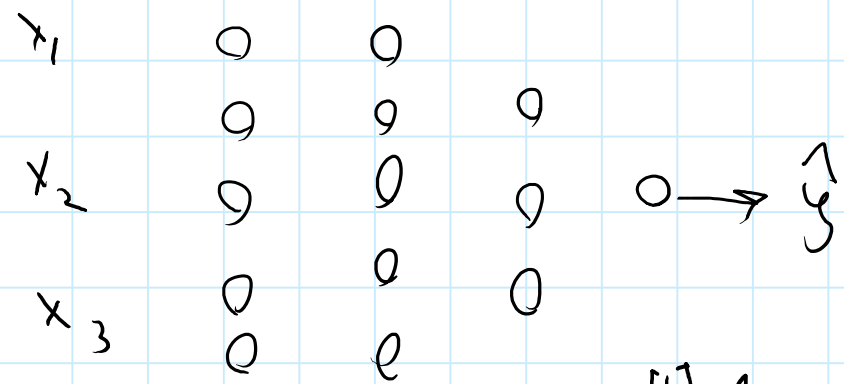
# forward
Z1 = np.dot(W1, X) + b1
A1 = sigmoid(Z1)
Z2 = np.dot(W2, A1) + b2
A2 = sigmoid(Z2)

Week 4

Thursday, September 21, 2017 9:19 AM

Deep Neural Networks (L-layers, L >> 2)

Notation



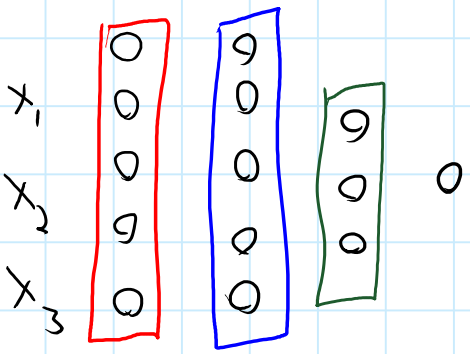
$n^{[0]} = n_x$ $n^{[1]} = 5 = n^{[2]}$ $n^{[3]} = 3$ $n^{[L]} = 1$

$a^{[l]} = g^{[l]}(z^{[l]})$ $W^{[l]}, b^{[l]}$ = weights for $z^{[l]}$

Deep NNs

Thursday, September 21, 2017 9:26 AM

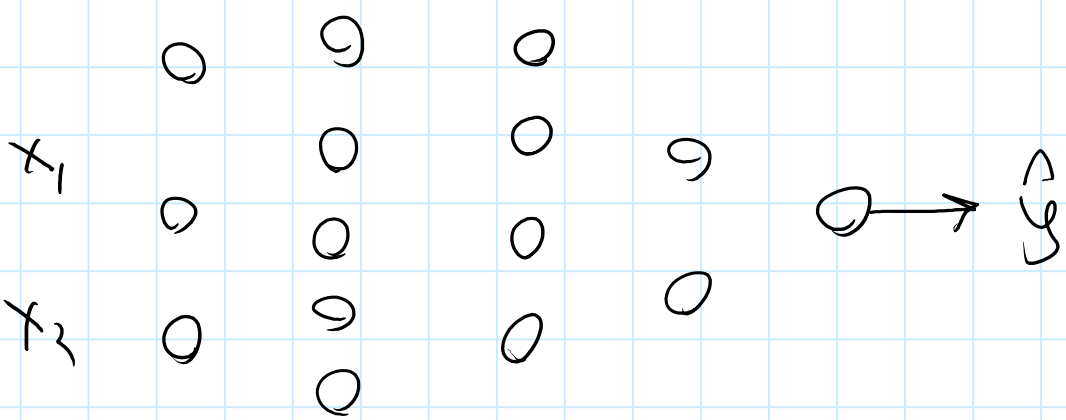
Forward Prop in a Deep NN



$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g^{[l]}(z^{[l]})$$

$$z^{[1]} = W^{[1]} x + b^{[1]}, \quad a^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}, \quad a^{[2]} = g^{[2]}(z^{[2]})$$
$$z^{[3]} = W^{[3]} a^{[2]} + b^{[3]}, \quad a^{[3]} = g^{[3]}(z^{[3]})$$

Getting Matrix Dimensions Right



General: $W^{[l]} = (n^{[l]}, n^{[l-1]}) = dW^{[l]}$

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

$b^{[x]} = (n^{[y]}, 1) = db$

$Z^{[x]}, A^{[x]} = (n^{[x]}, m) = dZ^{[x]}, dA^{[x]}$

Why Deep Networks?

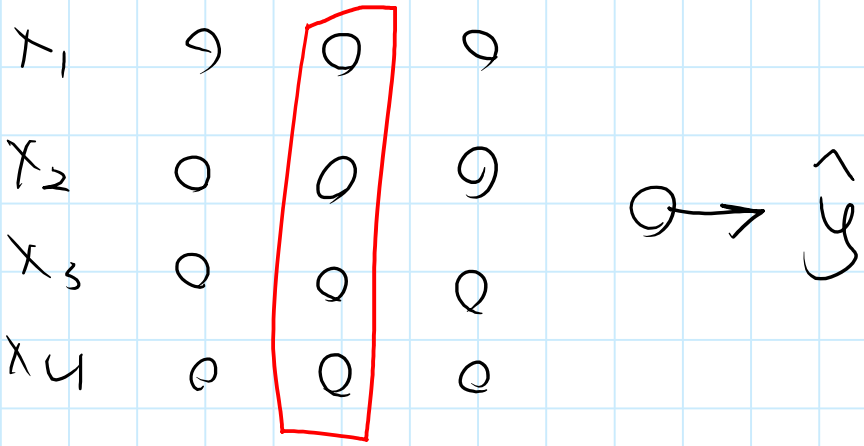
Layers split up tasks among neurons, (image and early layers may detect low level things (edges, waveform)

More layers can negate need for more hidden units in a given layer

Building Blocks

Thursday, September 21, 2017 10:14 AM

Forward & Backward Functions

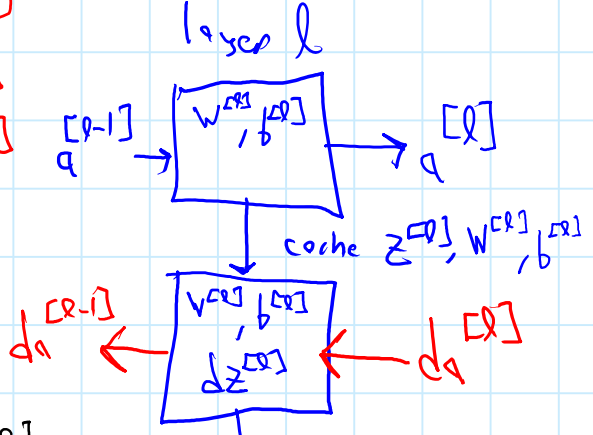


Forward: $z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$

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

"cache" $z^{[l]}$ for later use

Backward: Input $da^{[l]}$, output $da^{[l-1]}$, $dW^{[l]}$, $db^{[l]}$



$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$

$b^{[l]} := b^{[l]} - \alpha db^{[l]}$

Forward & Backward Prop

Forward: input $a^{[l-1]}$, output $a^{[l]}$, cache($z^{[l]}$

forward: input $a^{[l-1]}$, output $a^{[l]}$, cache($z^{[l]}$)

Backward: input $da^{[l]}$, output $da^{[l-1]}, dW^{[l]}, db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]}) = W^{[l+1]T} dz^{[l+1]} * g^{[l]\prime}(z^{[l]})$$

$$dW^{[l]} = dz^{[l]} \cdot a^{[l-1]}$$

$$db^{[l]} = dz^{[l]}$$

$$da^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$$

Vectorized:

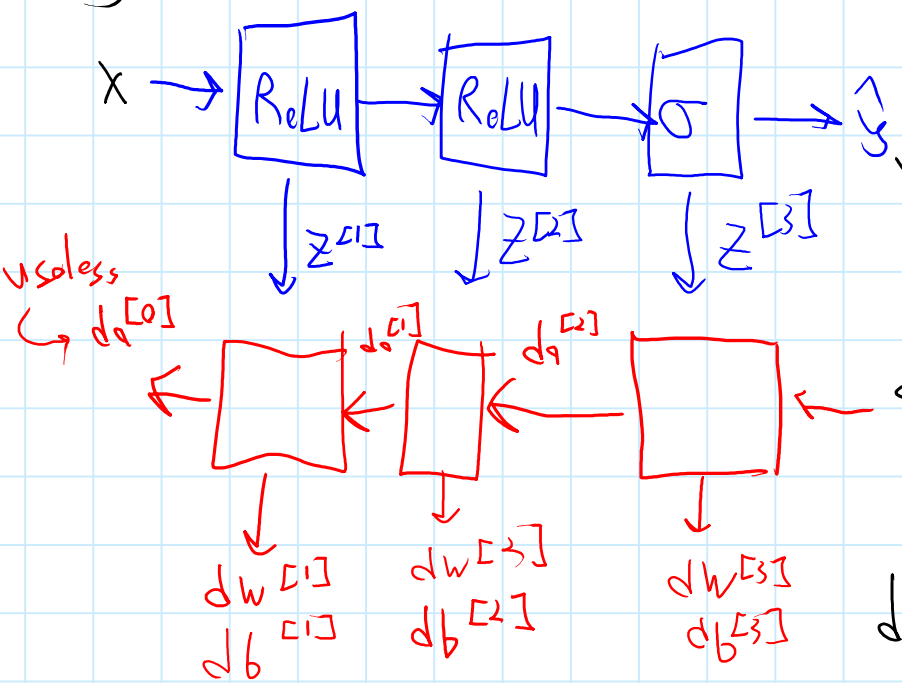
$$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$$

$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$

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

$$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$

Summary



Parameters vs Hyper parameters

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

Hyper parameters: learning rate (α)
# of iterations
# of hidden layers (L)
hidden units ($n^{[1]}$, $n^{[2]}$, ...)
Choice of activation function

Later: Momentum, mini batch size, regularizations, ...