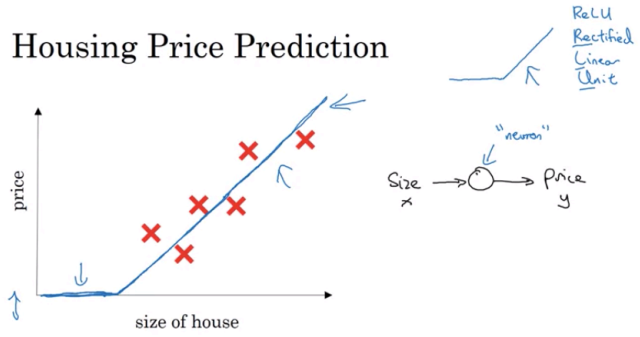
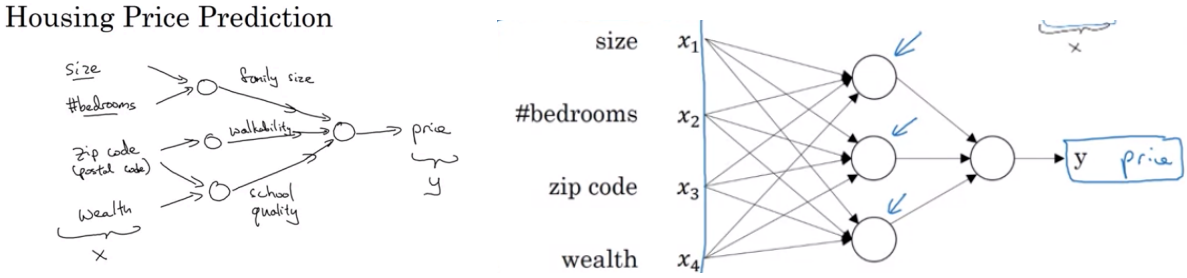
1. Neural Networks Introduction

Consider a simple function of predicting the house prices based on the size of the house. This is a simple function that finds a linear fit of the data and also a simple neural network that takes a size of the house and outputs the price.



A simple network fits a function to points shown in cross marks. The resulting function is called RELU function (Rectified Linear Unit) which will be described later. However, the methods to fit the function to set of predictors is what NN does. The term **deep learning** refers to training deep neural networks which will be discussed later.

If we have multiple features such as, size, #bedrooms, Zip code etc. then we may have lot more cells (or neurons).



A Neural Network is a set of nodes that fits a function to predict output.

1.1. Notations

m	Size of length of training examples	n	Length of predictors $X = \|X\|$
X	Notation for input predictors		Super script (i) represents the ith example

2. Binary classification

Binary classification is given an input, classify the data into one or other class. As an example: if we are given a set of images, classify them as pictures of cat or non-cats.

2.1. Logistic regression

Logistic regression is used for classification problems. The term **regression** here seems to indicate otherwise. Model predicts a number between 0 and 1 and a threshold is used to compare the result to predict one or other depending upon if it is greater or lesser than the threshold. A threshold T (usually 0.5) which indicates if it the predicted number is less than threshold T then we predict one otherwise other class.

Here we are restricting for two classes - however, later we show how it can be extended to show multiple classes.

In Linear regression we had the hypothesis function

$$h = \theta^T X \tag{1}$$

We would like to interpret h as a probability function. As is, it is not a function that can do binary classification.
Therefore if we consider the **logit** AKA **log of odds** where P is the probability. And X is the input:

$$P = g(x)$$
$$\frac{P}{1-P} = z = \theta^T X$$

g is a function that translates h *to* P the probability give X

Given X we want to compute the output $\hat{y}$ = P(y = 1 | X) ==> Probability that y = 1 give X

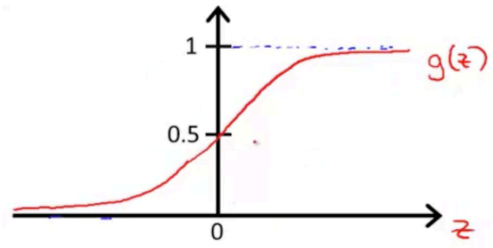
Solving for P, we get:

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

Therefore, in Logistic Regression, the form of the hypothesis takes the form:

$$h(x) = g(\theta^T x)$$
$$g(\theta^T x) = \frac{1}{1+e^{-z}}$$
$$z = \theta^T x$$

g is some function that transforms linear equation to probability



The function g is called Logistic function or Sigmoid function. Logistic or Sigmoid can be used interchangeably.

It is shown on the left. When $z >> 0$ the function reaches 1, when $z << 0$, it reaches zero, when at 0, $g(x)$ is 0.5. when z is +ve the term $e^{-z} < 1$, therefore, $\frac{1}{1+e^{-z}}$ is $\frac{1}{<2} > .5$; and when z is -ve the denominator is > 2 and thus g will be < 0.5

$$g = \begin{cases} \frac{1}{1+e^0} = 0.5, & \text{if } z = 0 \\ > .5, & z < 1 \\ < 0.5, & z > 1 \end{cases}$$

In Linear regressions we had the cost function

$$Cost(\text{AKA Loss function}\mathscr{L}) = J(\theta) = \frac{1}{2m}\sum_{1}^{m}(\hat{y} - y)^2$$ sum of squares of residues

We need to modify this because $(\hat{y} - y)^2$ is non-convex function for y *in* $\{0, 1\}$. If you plot it it will look wavy with no one max or minima. Therefore, we want a convex function so that we can gradient decent converge to global minimum.

In Logistic Regression, given a X examples :

$(x^{(1)}, y^1), (x^{(2)}, y^2), \ldots (x^{(m)}, y^m)$, we want to compute $\hat{y} \approx y^{(i)}$

Where $\hat{y} = \frac{1}{1+e^{-z}}$ at least in the training example want it to be as close match as possible.

We define the cost function for single training example:

$$Cost(\hat{y}, y) = \begin{cases} -\log \hat{y}, & \text{if } y = 1 \\ -\log(1-\hat{y}) & \text{if } y = 0 \end{cases}$$

If y is 1, then want $\hat{y}$ to be close to 1 anything less is incurred as cost

If y is 0, then want $\hat{y}$ to be close to 0, otherwise cost is the difference.

Since log of number in [0, 1] is negative, we take the negative log. The above equation can be combined:

$[-y_i \log \hat{y} - (1 - y_i)\log(1 - \hat{y})]$ when y_i is 0 first term goes away and similarly other term

The cost function for the entire training set is:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^i \log(\hat{y}^i) + (1 - y^i)\log(1 - \hat{y}^i)) \qquad (2)$$

Now all that remains to be done is to find θ to minimize $J(\theta)$ that is convex so that the slope is zero (first derivate) is zero.

We now have:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^i \log(\hat{y}^i) + (1 - y^i)\log(1 - \hat{y}^i)) \qquad \text{Same as (2) above}$$

NOTE - We want to find θ that minimize the cost function. In order to find the minimum,

2.1. We take the first order differentiation of the equation to be minimized
2.2. Find the roots from step (1) - Since Cost function is convex, there should be only one root.

That is what we will do next,

where $h_\theta(x)$is defined as follows:

$$\hat{y} = h_\theta(x) = g(\theta^T x)$$
$$g(z) = \frac{1}{1 + e^{-z}} \qquad (4)$$
$$\frac{\partial}{\partial \theta_j} J(\theta) = \sum_{i=1}^{m} (\hat{y} - y^i) x_j^{(i)}$$

Detailed Derivation: Now, we show detailed derivation of equation (4).

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^i \log(\hat{y}^i) + (1-y^i)\log(1-\hat{y}^i))$$
$$\theta x^i = \theta_0 + \theta_1 x_1^i + \ldots + \theta_p x_p^i .$$

Lets just consider and notice,

$$\log \hat{y}^i = \log \frac{1}{1+e^{-\theta x^i}} = -\log(1+e^{-\theta x^i})$$

$$\log(1-h_\theta(x^i)) = \log(1-\frac{1}{1+e^{-\theta x^i}}) = \log(e^{-\theta x^i}) - \log(1+e^{-\theta x^i}) = -\theta x^i - \log(1+e^{-\theta x^i}),$$

[this used: $1 = \frac{(1+e^{-\theta x^i})}{(1+e^{-\theta x^i})}$, the 1's in numerator cancel, we used: $\log(x/y) = \log(x) - \log(y)$]

Since our original cost function is the form of:

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

Plugging in the two simplified expressions above, we obtain

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

which can be simplified to:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i\theta x^i - \theta x^i - \log(1+e^{-\theta x^i})\right] = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i\theta x^i - \log(1+e^{\theta x^i})\right], \quad (*)$$

where the second equality follows from

$$-\theta x^i - \log(1+e^{-\theta x^i}) = -\left[\log e^{\theta x^i} + \log(1+e^{-\theta x^i})\right] = -\log(1+e^{\theta x^i}) .$$

[we used $\log(x) + \log(y) = log(xy)$]

All you need now is to compute the partial derivatives of (*)w.r.t. θ_j.
As

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{\partial}{\partial\theta_j}\Big(-\frac{1}{m}\sum_{i=1}^{m}\Big[y_i\theta x^i - \log(1+e^{\theta x^i})\Big]\Big)$$

$$\frac{\partial}{\partial\theta_j}y_i\theta x^i = y_i x_j^i,$$

$$\frac{\partial}{\partial\theta_j}\log(1+e^{\theta x^i}) = \frac{x_j^i e^{\theta x^i}}{1+e^{\theta x^i}} = x_j^i h_\theta(x^i)$$

Thus,

$$\frac{\partial}{\partial\theta_j}J(\theta) = \sum_{i=1}^{m}(h_\theta(x^i) - y^i)x_j^i \qquad \blacksquare$$

Once we know the derivative of the *Cost* function. Next step is to use it to find the parameters, θ that minimizes the cost function.

2.3. Gradient Decent (GD)

To recap, we have

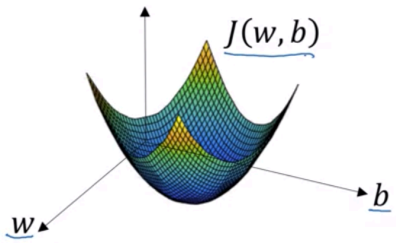
$$\hat{y} = \sigma(\theta^T X + b) \qquad \text{we did not speak about b before}$$

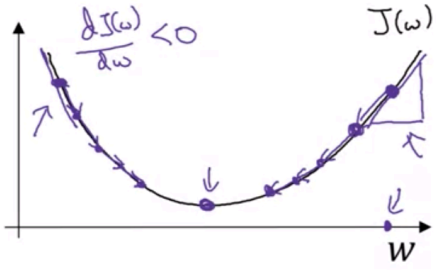
$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad 2.2.1$$

$$J(\theta, b) = -\frac{1}{m}\sum_{i=1}^{m} y^i \log(\hat{y}^i) + (1-y^i)\log(1-\hat{y}^i)) \qquad 2.2.2$$

$$\frac{\partial}{\partial\theta_j}J(\theta) = \sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})x_j^i \qquad 2.2.3$$

We can use the above equations in *Gradient Descent* algorithm to find the optimal (θ, *b)* to minimize the cost function *J(θ)*. The rationale is as follows. Any convex function would have a graph as shown below , if we plot :





Procedure

1. Initialize (θ, *b)* to any random value
2. Compute the derivative as given in 2.2.3
3. Update θ as $\theta = \theta - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta)$ where alpha is learning rate determines how big of a step we take each time.
4. Continue until the error reaches acceptable limit.

Why does this work. Let's take an example as shown in the right side of the figure above. If the point is on the right side, the slope (or the derivative is +ve) therefore we subtract from θ. On the other hand, if it is on the left of the minimum the slope is -ve and we move θ towards right. Therefore GD will move θ in the right direction to find minimum point.

2.4. Rationale for Logistic Regression loss function.

$\hat{y} = \sigma(\theta^T X + b)$

We want to interpret:

$$\hat{y} = P(Y = 1 \mid X) : \begin{cases} \text{if } \; y = 1 \; \text{ then } \; P(y = 1 \mid X) \; = \; \hat{y} \\ \text{if } \; y = 0 \; \text{ then } \; P(y = 0 \mid X) \; = \; 1 - \hat{y} \end{cases}$$

We can write both in one equation:

$$P(Y \mid X) = \hat{y}^{y}(1 - \hat{y})^{(1-y)} \qquad 2.3.1$$

when y is 0, $P(Y/X) = (1 - \hat{y})$; similarly other case

We want to maximize this across all examples: $P = \prod_{i=1}^{m} P(y \mid X)$

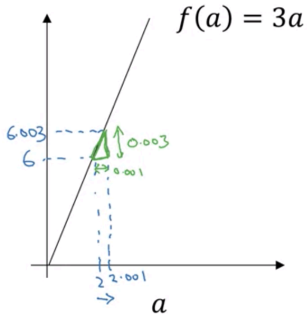
Maximizing the probability is equivalent to maximizing the log of the probability. Therefore,

$$log(P_{\text{all training examples}}) = log\left(\prod_{i=1}^{m} P(y^i \mid X^i)\right) = \sum_{i=1}^{m} log(P(y^i \mid X^i)) \qquad 2.3.2$$

From 2.3.1, $log(P(y \mid X) = y^i \log(\hat{y}^i) + (1 - y^i)\log(1 - \hat{y}^i))$

We want to maximize P and minimize the loss that is -ve of P, therefore, we want to choose parameters that minimizes the cost of the function that can be does using maximum likelihood estimation methods. Therefore the final cost function from 2.3.2 averaged over all training examples is:

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



■

3. Derivatives

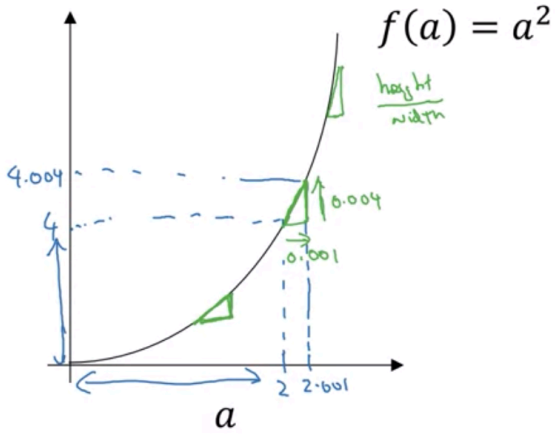
3.1. Intuitive definition of derivative

Suppose if we have a function:
f(a) = 3a when plotted, it looks like in the figure as shown.
Derivative is just a slope of the e=function evaluated at any point *x*.

For example at *x = 2*, slope is height divided by width. Suppose if we take small width 0.001

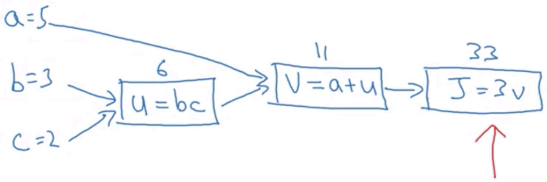
(6.0001 - 6.0003) / (2 - 2.0001) = 3.
For this function, slope is 3 wherever you take *x*.

In another example, where slope is different at different points.
It is shown in the example in the picture. At point *a = 2,* slope is 4, when *a* = 5, the slope is 10.
I n



$a = 2$ $\quad f(a) = 4$
$a = 2.001$ $\quad f(a) \approx 4.004$ (4.004001)
slope (derivation) of $f(a)$ at $a = 2$ is 4.
$\frac{d}{da} f(a) = 4$ when $a = 2$.
$a = 5$ $\quad f(a) = 25$
$a = 5.001$ $\quad f(a) \approx 25.010$
$\frac{d}{da} f(a) = 10$ when $a = 5$

general for this slope or the derivative is *2a.*



As you can see, we took a small nudging at x of 0.001, but we want this to be infinitely small. In general:

$$\lim_{\Delta a \to 0} \frac{f(a) - f(a + \Delta a)}{a + \Delta a - a} = \lim_{\Delta a \to 0} \frac{f(a) - f(a + \Delta a)}{\Delta a}$$

3.2. Computation graphs

In order to compute the derivatives of the cost function, it gets complicated; therefore computation graphs helps to break down the complex functions to make it easier to compute derivates. It also helps to create automated derivation code. Let see an example.

Suppose we have a function *J(a,b,c) = 3 (a + b c)*, we want to find the *J'* - derivative of *J w.r.t. a,b,c*.

Lets write this as:

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

We can write this as graph as shown here.
To find derivative of *J* w.r.t. *a, b, c,* first find the derivative of *J w.r.t v,* which is *3*.
Next, we find *J'* w.r.t *u,* we can do this as follows:

$$\frac{dJ}{du} = \frac{dJ}{dv} \cdot \frac{dv}{du} \qquad \text{we already know } \frac{dJ}{dv} = 3; \text{ and } \frac{dv}{du} = 1 \Rightarrow \frac{dJ}{du} = 3$$

Similarly, $\frac{dJ}{da} = \frac{dJ}{dv} \cdot \frac{dv}{da} = 3 \cdot 1 = 3 = 3$

Continuing this way,

$$\frac{dJ}{db} = \frac{dJ}{du} \cdot \frac{du}{db} = 3 \cdot c$$
$$\frac{dJ}{dc} = \frac{dJ}{du} \cdot \frac{du}{dc} = 3 \cdot b$$

3.3. Logistic regression - Back propagation

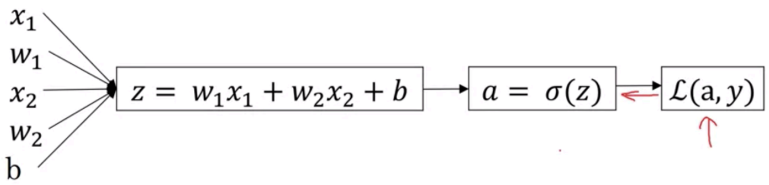
Lets consider a Logistic regression with only two predictors

$$a = \sigma(WX + b)$$
$$z = WX + b = w_1 . x_1 + w_2 x_2 + b$$
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Loss function $\mathscr{L}(a, y) = -(y \log(a) + (1 - y)\log(1 - a))$

These are exactly same as before except we replace y-hat with a that stands for activation function. (also makes typing so much easy)

Computation graph is



To back propagate we need to know how to adjust w, w2 and b to minimize the loss. This means, we need to find derivative of Loss function w.r.t to w1, w2, b and use it adjust the weights to get to the minimum.

From the computation graph,

$$\frac{d\mathscr{L}}{dw_1} = \frac{d\mathscr{L}}{da}\frac{da}{dz}\frac{dz}{dw_1} \tag{I}$$

$$\frac{d\mathscr{L}}{da} = \frac{-y}{a} + \frac{1-y}{1-a} \qquad \text{because: } log(1-a) = \frac{-1}{1-a}$$

$$= \frac{-(1-a)y + a(1-y)}{a(1-a)} = \frac{a-y}{a(1-a)} \tag{A}$$

next Consider:

$$\frac{da}{dz} = \frac{(1+e^{-z})\cdot 0 + e^{-z}}{(1+e^{-z})^2} \qquad \text{reminder:} a = \frac{1}{(1+e^{-z})}$$

$$= \frac{e^{-z}}{(1+e^{-z})^2} \tag{B}$$

Notice:

$$a = \frac{1}{1+e^{-z}} \therefore (1-a) = \frac{1+e^{-a}-1}{1+e^{-z}} = \frac{e^{-z}}{1+e^{-z}} \tag{C}$$

.

Substituting (C) in (B):

.

$$\frac{da}{dz} = a(1-a) \tag{D}$$

.

Finally:

.

$$\frac{dz}{dw_1} = x_1 \tag{E}$$

.

Sbstituing (A), (D), (E) in (I):

.

$$\frac{d\mathscr{L}}{dw_1} = \frac{d\mathscr{L}}{da}\frac{da}{dz}\frac{dz}{dw_1}$$

$$= \frac{(a-y)}{a(1-a)} a(1-a)\cdot x_1$$

$$\frac{d\mathscr{L}}{dw_1} = (a-y)x_1$$

Similarly we get:

$$\frac{d\mathscr{L}}{dw_2} = (a-y)x_2$$

$$\frac{d\mathscr{L}}{db} = (a-y)$$

In general the derivative of loss function will be multiplying corresponding *(a-y)* with the predicator.

With this we can implement back-propagation as follows:
For one training example compute the following to adjust weights:

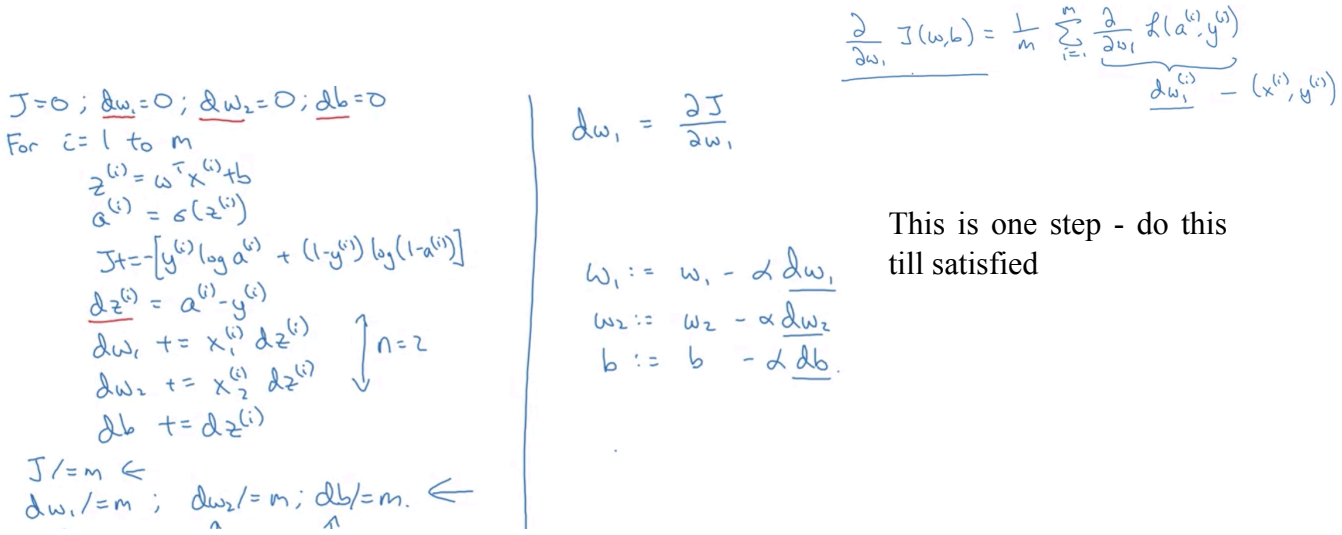
```
dz    =     (a - y)
dw1   =     dz * w1
dw2   =     dz * w2
db    =     dz

# Update
w1    = w1 - alpha * dw1
w2    = w2 - alpha * dw2
b     = b - dz
```

Repeat until error is minimized; Not a great algorithm. A complete example for m training example is shown below.

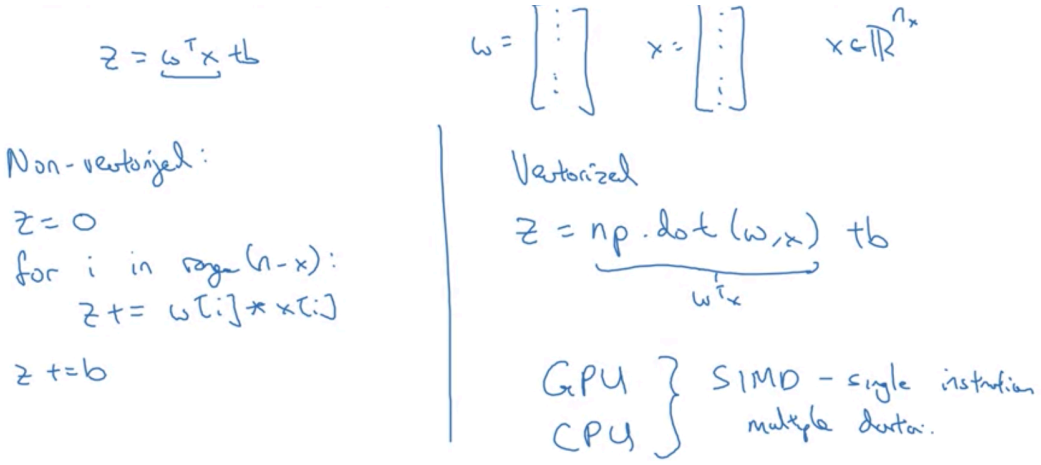
3.4. Gradient Descent with loss function to adjust weights.

As show before the loss function for one training example. It so happens that for *m* training examples, we can use it compute the total cost function.



3.5. Vectorization.

A mechanism to get rid of explicit for-loops.

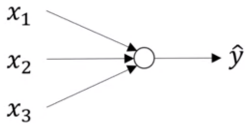


3.6. Notes.

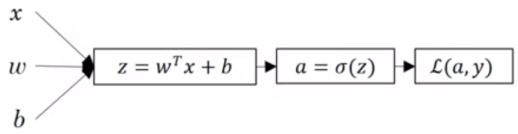
In python, arrays shape look like (5,) - they are called rank 1 arrays.

A = np.random(5) creates a vector a.shape = (5,) => avoid using the rank 1 shaped vectors.

4. Neural Networks

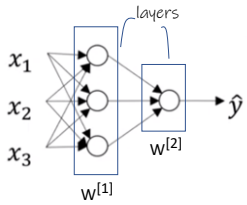


A logistic regression (also a neural network with one node) looks like the figure.

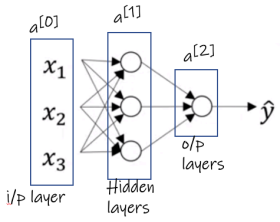


The inputs are passed in and through back propagation, W is adjusted and refined to find the optimal values for W and b to minimize the cost function.

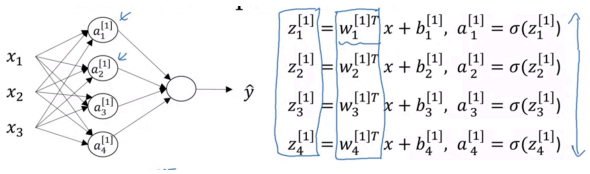
A Neural Network may consists of more than one such node stacked, also possibly arranged in layers to create a complex network. Each layer will compute z and a and finally last layer will compute the loss function $\mathscr{L}$.



A various types of Neural Network. A two layer network is shown. Two because, most times, input layer is not counted as a layer.



4.1. Computations of weights in NN.



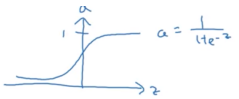
Similar to logistic regression in forward propagation, for each cell we compute the activations Z1, Z2 etc. Think each cell as one logistic regression cell. The back propagation happens for each cells very similar to one cell logistic regression and it happens to every cell to back propagate.

4.2. Activation Functions.

Sigmoid, tanh, RELU, Leaky RELU

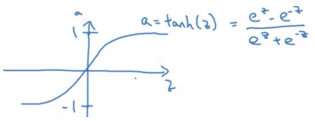
4.2.1. Sigmoid functions

Sigmoid function that is shown that we have been using so far



4.2.2. tanh functions

tanh function

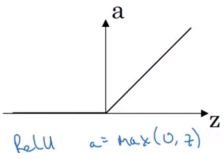


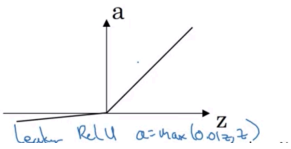
$g'(z) = \frac{d}{dz} g(z)$ = slope of $g(z)$ at z

$= 1 - (\tanh(z))^2$ ←

4.2.3. RELU functions

Or Leaky RELU function, shown below





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

~~undef if $z = 0$~~

$z = 0.0000\ldots0$

4.3. Take aways.

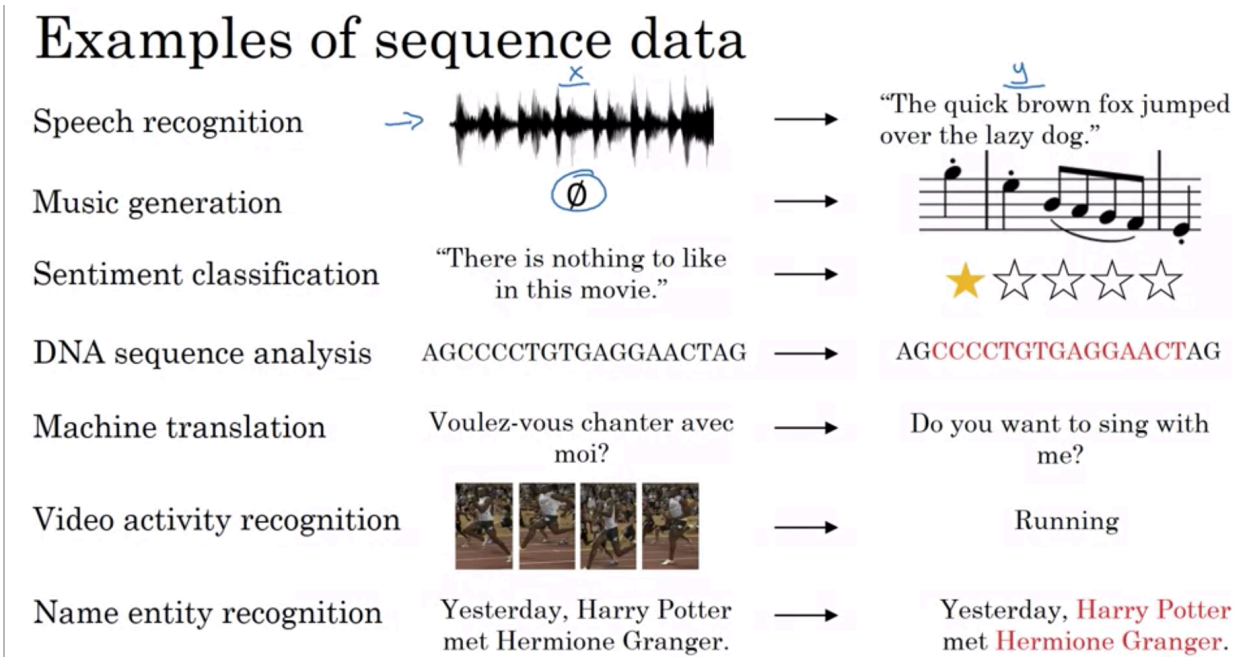
4.3.1. Never use sigmoid in hidden layers except output layers
4.3.2. If you are not sure what to use as activation for hidden layer, choose RELU
4.3.3. Do not use Linear Activation function in hidden layer - it practically makes layers useless.
4.3.4. Only place where linear activation function may be used is in output layer

5. Implementation

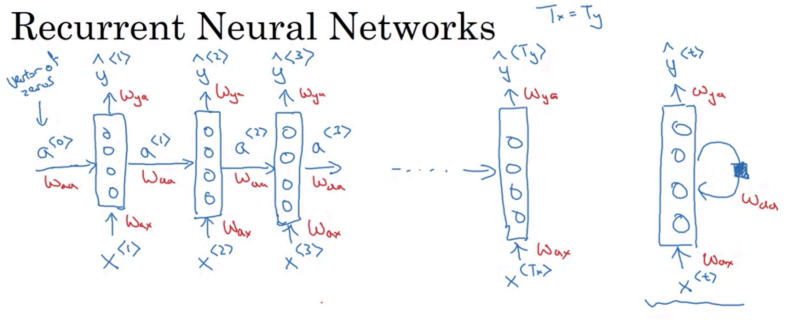
Keras implementation are in 01_NN-Keras.ipynb

6. Sequence Models

6.1. Examples



6.1. Notations

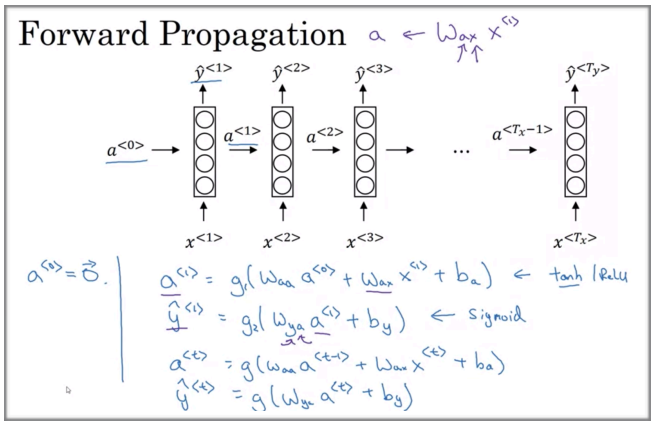


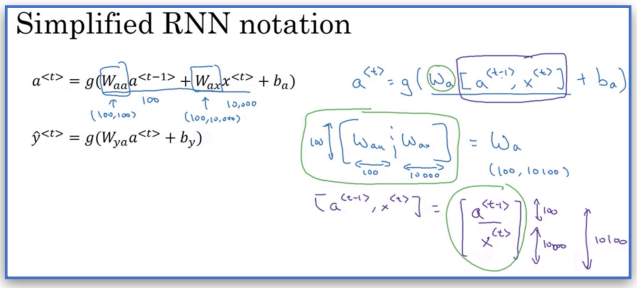
$x^{(i)<t>}$: input at time t T_x : Length of input sequence

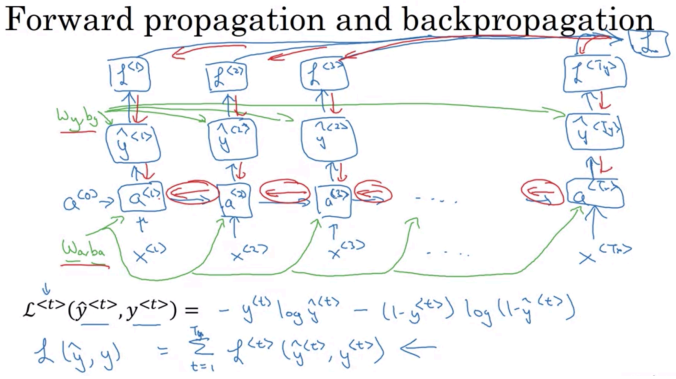
$y^{(i)<t>}$: input at time t T_y : Length of input sequence

Also note that we have bi-directional RNN

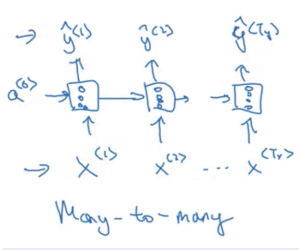
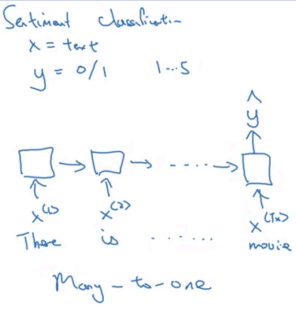
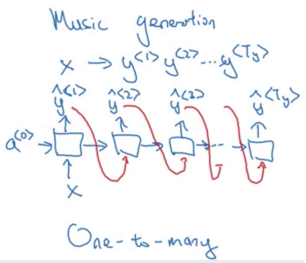
6.1. Calculations







6.1. Types of RNN

Many to Many	Many to 1	1 to many	1 to 1
			

Tx, Ty can be different Ex: translation	Sentiment classification	Music Generation	Becomes generic simple NN

6.1. Language modeling

Given a set of sentence, which sentence is more likely

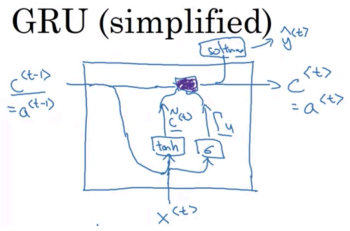
The apple and pair salad ex: P = 3.2
The apple and pear salad P = 5.7 <== this is more likely

6.1. Vanishing/exploding gradient

In a much deeper NN, then the gradient is difficult to propagate up. The errors in later step to affect the weights in earlier in the layer. Therefore outputs are influenced strongly influenced by local nodes.

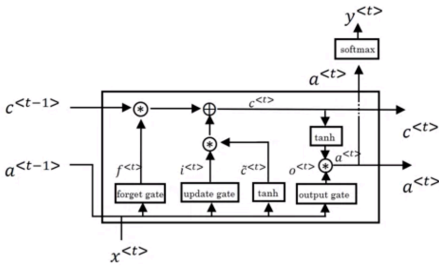
Vanishing gradients can cause the weights to almost becomes zero. Exploding gradients is where weights increase astronomically. Exploding gradients are easy to spot and easy to manage by gradient clipping. Vanishing does more things to do to handle.

6.1. Gated Recurrent Unit (GRU)



$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$
$$\Gamma_u = \sigma(W_u[\, c^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_r = \sigma(W_r[\, c^{<t-1>}, x^{<t>}] + b_r)$$
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$
$$a^{<t>} = c^{<t>}$$

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$
$$\Gamma_u = \sigma(W_u[\, a^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_f = \sigma(W_f[\, a^{<t-1>}, x^{<t>}] + b_f)$$
$$\Gamma_o = \sigma(W_o[\, a^{<t-1>}, x^{<t>}] + b_o)$$
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$
$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$



6.1. Long Short Term memory (LSTM)