Neural Networks and Deep Learning Notes [Course1]

September 7, 2021

1 What is a neural network?

A neural network is nothing but a function that learns a mapping from a set of inputs to an output. For example, for a given set of house properties, like size, number of bedrooms, etc, the function will try to learn to predict the optimum price for the house.

2 Supervised Learning

The supervised learning refers to the process of learning function parameters from a set of inputs X and outputs y. Below, there are some examples of supervised applications from real life experiences.

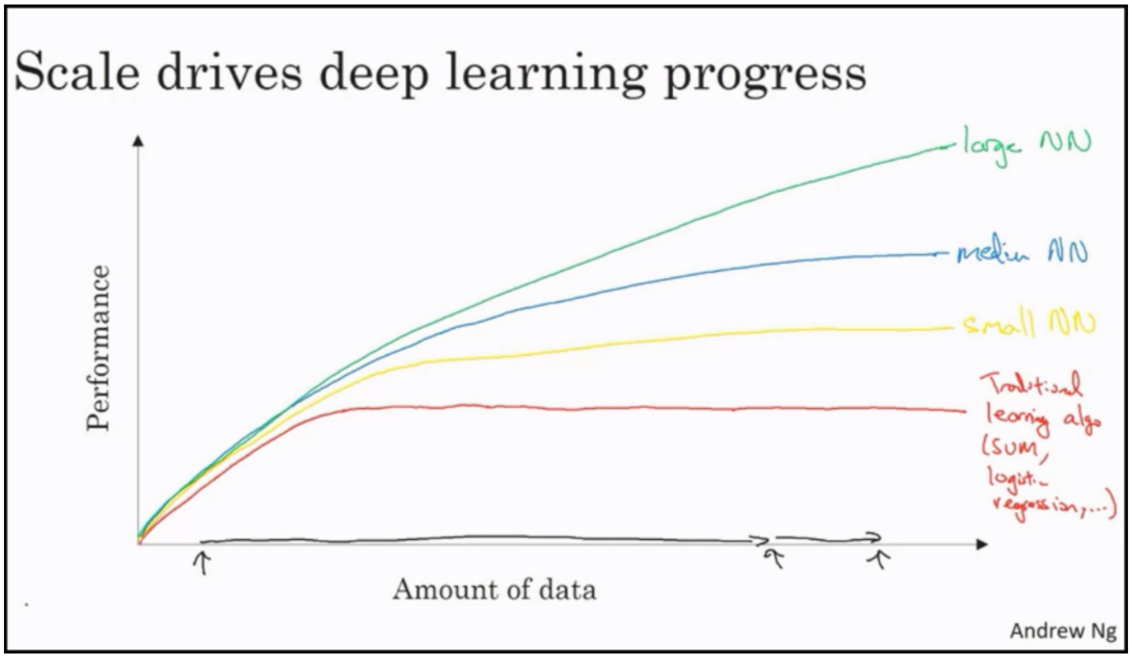
Inputs (x)	Output (y)	Application
Home Features	Price	Real Estate
ad, user info	click on ad? (0/1)	online advertising
Image	object (1,..,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	choice	machine translation
Image, Radar info	Position of other cars	Autonomous driving

3 Structured data vs unstructured data

Sturcutred data are the type of data where data items are organized on a sort of tabular format where many columns represents the inputs, i.e. features, and one column represent the output for each input. Unstructured data, in contrast, is any data that does not match the condition above like the audio and textual data. Usually, unstructured data is transformed to a format understandable by neural networks like Bag of Words transformation for textual data.

4 Why Deep Learning starts to take off just recently?

The current remarkable rase of deep learning happens because of the amount of the available data and the compute machinery that was not attainable before. Consider the graph below from the course lectures:



Couple of notes can be noticed here:

- Previously, traditional models, like svm and logestic regression, tend to work just well due to the limited amount of data available. However, they can not get the best advantage with a large amount of data.
- As long as the data gets bigger, deeper models can better exploit it and learn better mappings.
- Deeper models may be quite similar do shallower or traditional models in terms of performance given a small amount of data. However, with large amount of data, they advanced these shallow models.
- With a minimum amount of data, the hand engineering features could be the key to success even with shallower models. In fact, the order of the models within this range is not well defined.

From the above discussion, one can say that the modern world of deep learning is mainly derived by the scale, scale of data and computation. This is also not to ignore the big advances in the proposed algorithms that makes the learning of the network quite faster and more efficient.

One more note to add is that the process of training deep learning model is iterative. The network parameters are scaled over maturing experiments.

---

5 Binary Classification Problem

This problem is, for a given image, indicate whether the image has a cat or not. It is the practical problem that will be used through out the course to practice building the neural network.

The image will be represented as a three separate matrices. Each matrix represent one channel of red, green, and blue colors. These matrices will be refered as the set of features, X. Y, on the other hand, is a vector for each image indicating whether the image contains a cat or not.

In order to transform image matrices to an input vector, they need to be unrolled to a vector of length 64*64*3=12288 as each image is an 64 pexil image.

The problem is formulated as

$$F(X) \longrightarrow y$$

where $F(X)$ is the function to be learned by the neural network, X is the matrix of feature vectors, and Y is the vector of outputs for each feature vector.

6 Notations

- M represent an example ($x^{(i)}$,$y^{(i)}$).
- M_{train} represent training examples. Similarly, M_{test}, and M_{dev} represents testing and development examples respecitvely. Let m represents the number of training examples in M.
- X represents the matrix of x^i vectors as follows:

$$\begin{bmatrix} . & . & . & . \\ . & . & . & . \\ x^{(1)} & x^{(2)} & .. & x^{(m)} \\ . & . & . & . \\ . & . & . & . \end{bmatrix}$$

In that since, X is of size $n_x \times m$ where n_x is the length of any x^i vector, 12288 in this problem, and m is the size of the examples set M. More formally: $X \in \mathbb{R}^{(n_x,m)}$

- Y is the set of outputs. It is, as X, is stacked vertically as follows:

$$\begin{bmatrix} y^{(1)} & y^{(2)} & .. & y^{(m)} \end{bmatrix}$$

In this regard, Y will be defined more formally as: $Y \in \mathbb{R}^{(1,m)}$

7 Logestic Regression

given an input feature x, a vector of an image, the requirement is to predict a propability score

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

that the image contains a cat.

The parameters of the model are the weights, w and the bias b.

The formulat can be derived from the linear regression conjugated with sigmod function

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

where

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

7.0.1 Some notes about $\sigma(z)$ function

- If the value z is very large, then the value of the σ becomes very close to 1.
- If the value of z is very small, like a very large negative number, then the value of σ function becomes very small.

7.1 Logestic Regression Training

7.1.1 The loss function derivation

The linear regression loss (error) function

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

is not applicable here as it is not convex. i.e. the gradient descent algorithm is not guarnteed to find a global minimum for the function. The function is not convex due to the use of the sigmoid $\sigma(z)$ function.

The following Loss function is used in logestic regression

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

Some intuition why this loss function works:

- If $y = 1$, then, in order to minimize the function $\hat{y}$ should be as large as possible. The second term is zero.
- If $\hat{y} = 1$, then, $(1 - \hat{y})$ should be as large as possible, hense $\hat{y}$ should be as small as possible.

Hense, if the true value y is 1, then $\hat{y}$ is pushed to be as large as possible i.e. closer to 1. Also, if $y = 0$, then $\hat{y}$ is pushed to be as small as possible, i.e. closer to zero.

7.1.2 The Cost function derivation

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

where L is the loss function derived in the previous subsection. Hence, the final formula would be:

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

8 Gradient Descent

The key part of the gradient descent algorithm is iteratively updates the weights and biases until some threshold or predefined number of iterations. The formula is

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

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

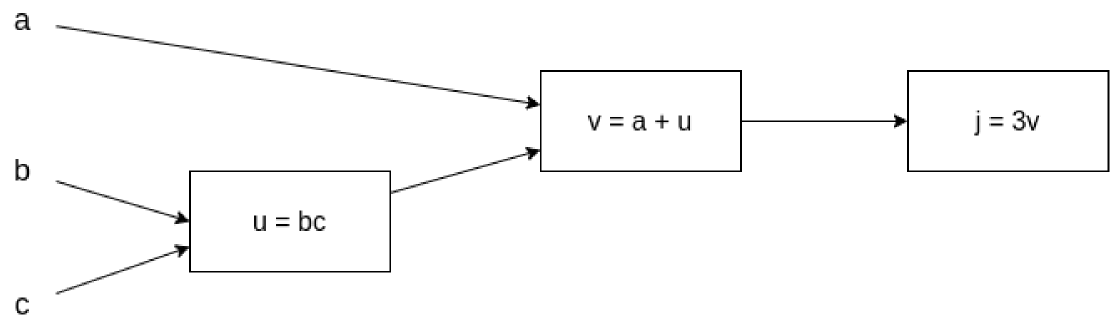
where: - $\frac{\partial J(w,b)}{\partial w}$ is the partial derivative of the cost function with respect to weights parameter, w. Similarly, $\frac{\partial J(w,b)}{\partial b}$ is the bias partial derivative. - α is a parameter used to control the update rate. It is usually called, the learning rate.

9 The Computation Graph

The computation graph organizes the computation of the gradient descent and its derivatives on both passes, forward pass and backward pass. Consider the following function

$$f(x) = 3(a + bc)$$

We can organize it as the following computation graph:

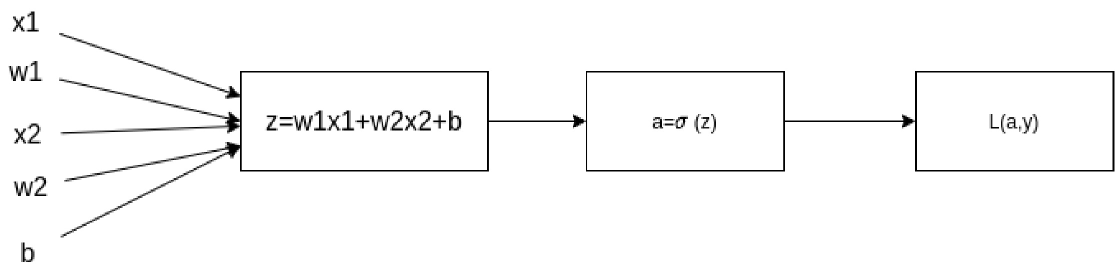


the variable u and v are used to facilitate the computation.

the derivative of j with respect to v can be calculated as $\frac{dj}{dv}$ which is 3. The derivative $\frac{dj}{du}$ can be calculated using the chain rule as $\frac{dj}{dv} \times \frac{dv}{du}$ which is $3 \times 1 = 3$ and so on until reaching a,b and c variables.

10 Logestic Regression Computation Graph

Consider the following graph for logestic regression with two features x1,x2 example.



Note that $\hat{y}$ is replaced with a in the computation graph just for presentation conveniences. Now, in order to compute the derivative of the loss function l with respect to a we have:

$$\frac{\partial l(a,y)}{\partial a} = -(y \log a + (1-y)(\log 1 - a))$$

$$-\frac{y}{a} + \frac{1-y}{1-a}$$

Similarly, to compute the derivative of the loss function with respect to z, the chain rule will be used as follows:

$$\frac{\partial l(a,y)}{\partial z} = \frac{\partial l(a,y)}{\partial a} * \frac{\partial a}{\partial z}$$

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

finally, to compute the derivatives for the parameters w,b as follows:

$$\frac{\partial l}{\partial w_1} = x_1 \times (a-y)$$

$$\frac{\partial l}{\partial w_2} = x_2 \times (a-y)$$

$$\frac{\partial l}{\partial b} = (a-y)$$

where $a - y$ is the value of $\frac{\partial l}{\partial z}$

Finally, this one example derivation of the gradient descent algorithm can be generalized for w_1 to a dataset of m examples as follows:

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

The rest of the varialbes, w_2 and b have similar derivation over m examples.

11 Vectorization and Broadcasting

Vectorization refers to the process of applying matrix operations with the help of parallelism in the processor architecture. Usually, it is achieved through special packages like `numpy` for Python. Examples of such operations could be dot product or matrix addition.

Broadcasting refers primarly to the process of aplying an operation of a scalar to a vector. For example adding, subtracting or multiplying a scalar by a vector exploiting parallelism features of the processor. These scalers can also be stacked to be added/subtracted etc with matrices.

The general principle of broadcasting is as follows.

- For a matrix of dimension (m, n), added/subtracted to $(1, n)$ vector, the second vector will be copied m times to match the dimension of (m, n) and will be added to the matrix.

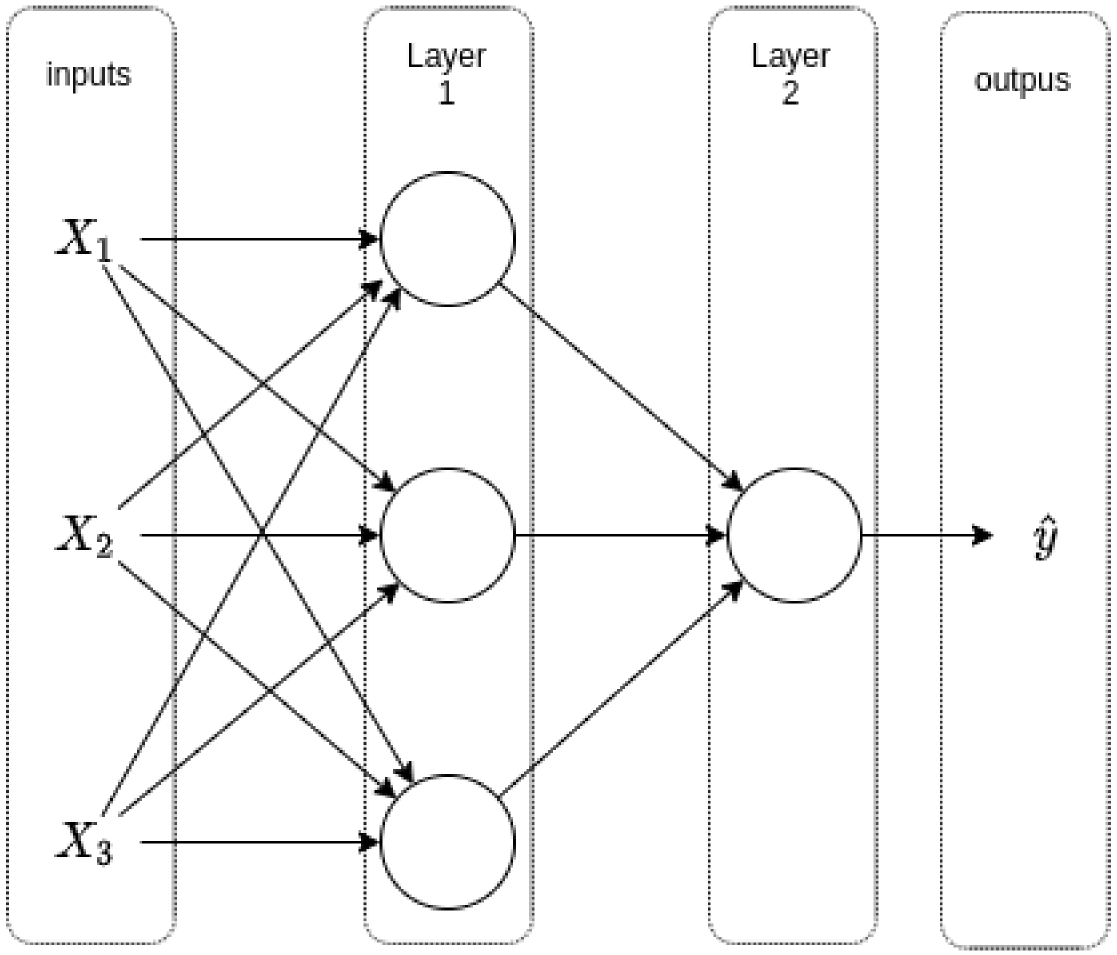
- For a matrix of dimension (m, n), added/subtracted to $(m, 1)$ vector, the second vector will be copied n times to match the dimension of (m, n) and will be added to the matrix.

The numpy documentation [https://numpy.org/doc/stable/user/basics.broadcasting.html] can be checked for further information on this operation.

---

12 Neural Networks Representation

Neural Networks are generalization of the previous logestic regression model with multiple layers. Each layer will apply multiple calculations of $\sigma(z)$ where $z = w^T x + b$. Consider the following image:



It can be noticed that this network can represent a 2-layer neural network. The first layer, layer one, will compute $w^{[1]}.T \times x + b$, and apply it to the sigmoid σ function. The second layer will compute again $w^{[2]}.T \times x + b$ and apply σ function to the output to give the final output $\hat{y}$. It can be seen that each circle does two main computations, the first computation is $z^{[i]}$ and the second computation is $a^{[i]} = \sigma(z)$ where i represents the layer index.

In other words, we can say that the previous network represent a 3 logestic regression units stacked in the first layer, and the output of these units is fed into another one logestic regression unit in the second layer which is responsible of producing the output $\hat{y}$.

The computation of the above network can be summarized in the following computation graph:

$$\boxed{\boxed{z^{[1]} = w^{[1]T} \times x + b^{[1]}} \longrightarrow \boxed{a^{[1]} = \sigma(z^{[1]})}} \longrightarrow \boxed{\boxed{z^{[2]} = w^{[2]T} \times a^{[1]} + b^{[2]}} \longrightarrow \boxed{a^{[2]} = \sigma(z^{[2]})}} \longrightarrow \boldsymbol{\ell}(a^{[2]}, y)$$

- In the above computation graph, outer boxes represents layers while inner boxes represents computations within layers.
- **An important note** on the notation is that the superscript square brackets “[]” will refer to the layer index. This is not to confuse it the parantheses “()” where they refere to the example index in the given training/dev/test set.
- The input layer, x can be also refered to as $ a^{[0]} $.
- vectorization and broadcasting techniques can be applied to represent and paralalize the previous network.

13 Activation Functions

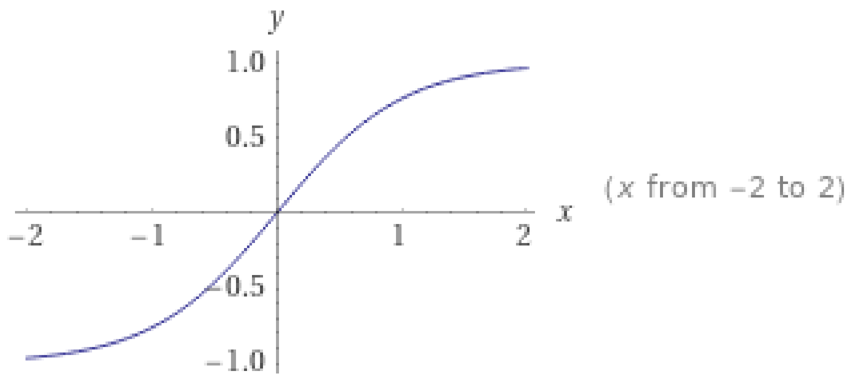
Throughout the previous material, the sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ function was used to transform the output of the linear regression $wx + b$ into some nonlinar format. However, this is not the only function to use to add some nonlinearity to the linear regression formula. Some other alternative functions are listed.

13.1 Tanh Function

Tanh function is limited between 1 and -1. It is, technically, a shifted and rescaled version of the sigmoid function. Tanh almost always, for hidden units, works better than the sigmoid function. This is because the mean of its values is closer to zero i.e. it kind of adds a centering effect to data in hand. The formula for the tanh function is:

$$\frac{e^z - e^{-z}}{e^z + e^{-z}}$$

The following graph, from walfram alpha, depicts the behaviour of the tanh function.



13.1.1 Downsides of tanh and sigmoid functions

The most prominent problem with tanh and sigmoid function is that as the input to the function,z, is on its exremes, either small or large, its derivative becomes very small, hence is the gradient. This slows down the learning of the network.

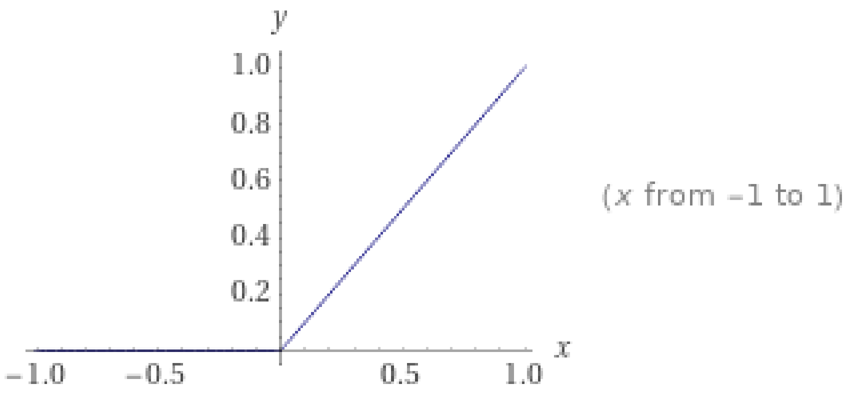
13.1.2 The Rectevied Linear Unit ReLU function

ReLU activation function comes mainly to address the limitations of the sigmoid and tanh functions.

The ReLU function is defined as follows

$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Hence, ReLU values when the input is on its extremes is not convergin to small values. It becomes the standard practice for the recent neural network implementations. The Graph of the ReLU , from walfram alpha, is illustrated below.



ReLU limitations

- The function is not differantiable around 0. However, this limitation, in practice, can be mitigated by considering that the function has a 0 or 1 derivative around 0.
- The function derivative is always zero for negative inputs. This limitation is amended by another version of the function called leaky ReLU where the function has a tiny slope for the negative input values, say

  $$f(x) = \begin{cases} x & x \geq 0 \\ 0.01x & x < 0 \end{cases}$$

  Its graph is shown below from paperswithcode.com[https://paperswithcode.com/method/leaky-relu, last-accessed: 31-08-2021]:

13.2 The need for activation functions

Suppose a neural network does not have any kind of activation function, let us say: an identity activation function. Therefore, it can be noticed that the results of the whole neural network can be achieved through one linear function. Consider the following computation happening accross layers:

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

let

$$w^{‘} = (w^{[2]}w^{[1]})$$

and

$$b^{‘} = (w^{[2]}b^{[1]} + b^{[2]})$$

Hence

$$a^{[2]} = w^{‘}x + b^{‘}$$

The above computation was applied on two layers network, but can be, similarly, generalized to k layers network.

It turns out that having a linear activation function for the hidden layers, the neural network is no different than a standard logestic regression model without any hidden layer. The above computation gives some insights on this conclusion.

Hence, a non-linear activation function is required to transform the regression function $wx + b$ into some nonlinear form.

13.3 Activation Functions Derivatives

The below table shows the derivatives of the aforementioned activation functions. These derivatives will be used in the backpropagation algorithm.

Name	Formula	Derivative
Sigmoid	$\sigma(z) = \frac{1}{1+e^{-z}}$	$\frac{1}{1+e^{-z}}(1 - \frac{1}{1+e^{-z}}) = \sigma(z)(1 - \sigma(z)$
Tanh	$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$1 - (\frac{e^z - e^{-z}}{e^z + e^{-z}})^2 = 1 - (\tanh(z))^2$

Name	Formula	Derivative
ReLU	$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$	$f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$

14 Gradient Descent for Neural Networks

Below, we describe the forward and the backward passes of the neural network using gradient descent optimization algorithm.

Note that, in the below derivation, $g^{[i]}(Z)$ refers to the actviation function used in the layer i.

For a network with n layers, the **forward pass** is:

for layer i from 0 to $n-1$*, do:*

- $Z^{[i+1]} = w^{[i+1]}A^{[i]} + b^{[i+1]}$
- $A^{[i+1]} = g^{[i+1]}(Z^{[i+1]})$

Finally, for the last layer n, we will use the sigmoid as an activation function.:

$A^{[n]} = g^{[n]}(Z^{[n]}) = \sigma(Z^{[i+1]})$

For the backward pass, the so-called backpropagation algorithm, its steps are described below:

for i *in layer 1 to* n*, update parameters* w *and* b *as follows:*

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

15 Random Initialization

Neural network wieghts cannot be initialized to zeros. **Why?**, because all hidden units will be computing the same function. Hence, no matter how deep the network is, all of the units in each layers even with multiple iterations compute an identical, i.e. symmetric, function. This can be proven with mathematical induction.

To illustrate how this happens, let us take the layer i. Each unit will have a zero wieght. The result of the calculation, $wx + b$ of each unit is identical. Consequenly, the results of the activation function of each unit as well.

Whta is the best practice, then? random initialization.

One more note to keep in mind is that the initializing of the wieghts should be relatively small values around zero. If they are quite large values, then, depending on the learning function, the learning process via gradient descent might be slow. This mainly happens with *tanh* and sigmoid function as their graph on the large values tend to flatten.

16 Deep L-Layer Neural Network

In this section, the shallow neural network model discussed in the previous sections, will be generalized to a deep network with L layers. Technically, logestic regression is a very shallow neural network. However, a network with quite large L layers is considered to be a deep network. As L get bigger, the network is said to be deeper.

16.1 Notations

Below, is a review of the previous notations as well as the new notations associated with notations from the deep L-layers networks.

- L refers to the number of layers.
- $n^{[l]}$ refers to the number of units in the layer l.
- $a^{[l]}$ denotes the activation applied in the layer l. $a^{[0]}$ denotes the inputs.
- $w^{[l]}$ represents the weights of layer l.
- $b^{[l]}$ represents the biases of layer l.

16.1.1 The general formula for the forward pass

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

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

17 L-layers network dimensions

For an L-layers network, the dimension of each of its variables and parameters are as follows.

variable	dimension	dimension on m training examples	Notes
$w^{[l]}$	$(n^{[l]}, (n^{[l-1]})$	$(n^{[l]}, (n^{[l-1]})$	
$\partial w^{[l]}$	$(n^{[l]}, (n^{[l-1]})$	$(n^{[l]}, (n^{[l-1]})$	
$b^{[l]}$	$(n^{[l]}, 1)$	$(n^{[l]}, 1)$	In Python, the vector will be added to the results of $(W.T \times X)$ via broadcasting

variable	dimension	dimension on m training examples	Notes
$\partial b^{[l]}$	$(n^{[l]}, 1)$	$(n^{[l]}, 1)$	
$z^{[l]}$	$(n^{[l]}, 1)$	$(n^{[l]}, m)$	
$\partial z^{[l]}$	$(n^{[l]}, 1)$	$(n^{[l]}, m)$	
$a^{[l]}$	$(n^{[l]}, 1)$	$(n^{[l]}, m)$	a^0 refers to the inputs
$\partial a^{[l]}$	$(n^{[l]}, 1)$	$(n^{[l]}, m)$	a^0 refers to the inputs

18 Why deep representation?

earlier layers of the network usually detects general simple features like edges in an image, for instance. Deeper layers compute more complex functions like face eyes, noses itc.

An intuition why deep representation works better than shallow representation comes from the circuit theory. In circuit theory, building multi level network of gates to compute basic circuit functions, say the XOR function, turns out to be easily calculated with only $\mathcal{O}(\log n)$ layers. However, it needs an exponentially large number of units if such calculation is restricted to be on only one layer! This is because an 2^{n-1} gates are needed to exhaust all the possible configurations of the n inputs.

19 Parameters vs Hyperparameters

Parameters of the neural network model are w and b. However, there are many other numbers that need to be set. For instance:

- Learning rate α.
- Number of iterations.
- Number of hidden layers L.
- Number of hidden units $n^{[l]}$ in the layer l.

- Choice of activation function.

These are some hyperparameters up to the current materials. However, there are other hyperparameters that will come on the way like the momentum term, minibatch size, regularization parameters, etc.

The current practice of applied deep learning is a very empirical process. The practitioner will start first with the idea, write the code, conduct the experiment, reiterate on the code and the hyperparameters, experiment again and so on until reaching to a the sought results.