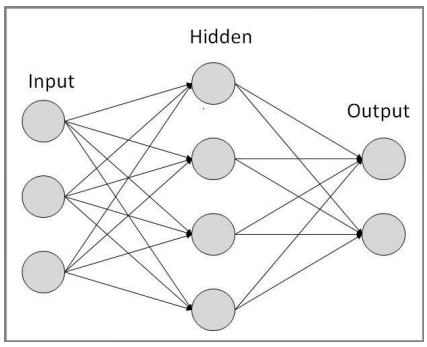
Part1. Neural Networks and Deep Learning

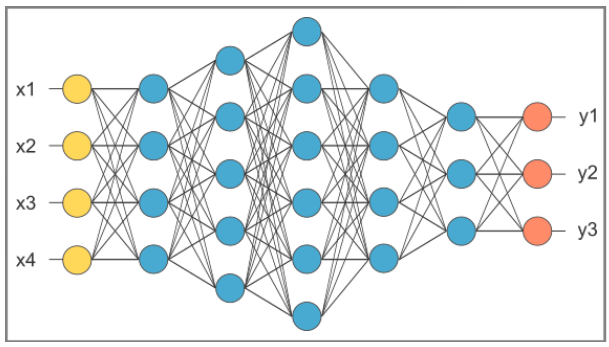
1. Introduction to Deep Learning
2. Neural Networks Basics
3. Shallow Neural Networks
4. Deep Neural Networks

1. Introduction to Deep Learning

What is a NN (Neural Network)?



Simple NN Graph



Deeper NN Graph

Different types of neural networks for supervised learning includes: CNN (convolutional neural networks; used in computer vision), RNN (recurrent neural networks; used in speech recognitino or NLP), standard NN (used in structured data) and hybrid/custom NN or a collection of NN types.

Why is deep learning taking off?

1. Data

For small data, NN can perform as linear regressino or SVM (Support Vector Machine). However, for big data, a small NN is better than SVM and a large NN is much better than small NN or medium NN. Since the amount of data available is increasing due to mobile phones or IoT (Internet of Things), NN is getting more popularity nowadays.

2. Computation

Using GPUs, powerful CPUs, distributed computing and ASICs, heavy computation of NN is now readily executed.

3. Algorithm

Crreative algorithms which changed the way NN operates has appeared. For example, using ReLUs function is much better than using Sigmoid function in training NN because it deals with the vanishing gradient problem.

2. Neural Networks Basics

Notations

- M: the number of training vectors
- N_x: the size of input vector, N_y: the size of output vector
- $X(k)$: the k-th input vector, $Y(k)$: the k-th output vector
  - $X = [X(1), X(2), \ldots, X(M)]$
  - $Y = [Y(1), Y(2), \ldots, Y(M)]$
- cf. Using NumPy, you can make matrices and corresponding matrices in a faster and more reliable time.

Binary Classification using Logistic Regression

- $Y = w^T X + b$ $(y = a\,x + b)$
  - w is a matrix of shape (N_x,1) and b is a real number
- In order to make Y between 0 and 1 to represent probability, $Y = \text{sigmoid}(w^T X + b)$
- In some notations, $Y = \text{sigmoid}(w^T X)$
  - $b = w_0$ of w and $X_0 = 1$, but we won't use this notation in the course

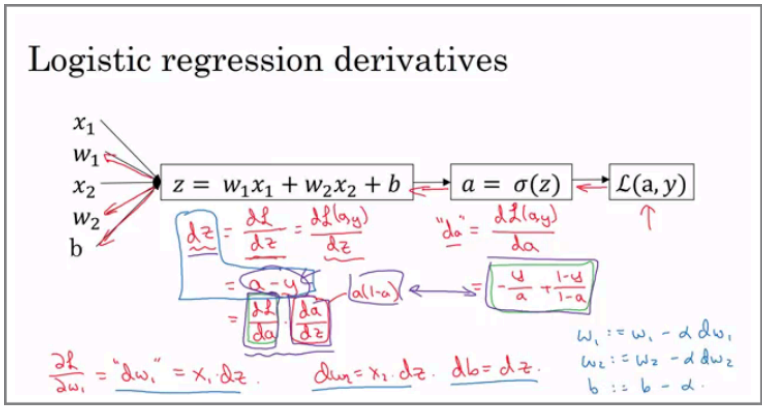
Logistic Regression Cost Function

- The loss function computes the error for a single training example; the cost functino averages the loss function of the entire training set
- $L(y', y) = \frac{1}{2}(y' - y)^2$
  - Not widely used since it leads to a function which is non-convex containing local optimum points
- $L(y', y) = -(y log y' + (1 - y) log(1 - y'))$
  - If $y = 0$, $L(y',0) = -log(1 - y')$ —> want $1 - y'$ to be the largest, y' to be the smallest
  - If $y = 1$, $L(y',1) = -log y'$ —> want y' to be the largest
- $J(w, b) = \frac{1}{m} \sum (L(y'[i], y[i]))$

Gradient Descent : How to predict w and b in order to minimize the cost function

- Initialize w and b to np.zeros and a random value in the convex function, respectively
- Try to improve the values to reach the minimum cost
- Similar to greedy algorithms, the derivative gives us the direction to improve our parameters.

- The gradient descent algorithm repeats: $w = w - \alpha d\frac{J(w,b)}{dw}$ and $b = b - \alpha d\frac{J(w,b)}{db}$ where α is the learning rate.



Gradient Descent in Logistic Regression

Exercise1. From right to left, calculate derivations.
Exercise2. Conclude the logisitic regression pseudo code.
(Vectorization is important on deep learning to reduce for-loops. We can make the whole loop in one step using vectorization!)

Vectorization

Deep learning shines when the dataset are big. However, for-loops will make you wait a lot for a result. That's why we need vectorizatino to get rid of some of our for loops. NumPy library functions, mostly are vectorized versions, are widely used for such purposes. So, whenever possible avoid for-loops!

Vectorizing Logisitic Regression

As an input, we have a (N_x, m) matrix X and a (N_y, m) matrix Y. We will then compute at instance $[z1,z2,...,zm] = W^T X + [b, b, \ldots, b]$.

- Reshape is cheap in calculations so put it everywhere you're not sure about the calculations.
- Broadcasting works when you do a matrix operation with matrices that doesn't match for the operationr, in this case NumPy automatically makes the shapes ready for the operation by broadcasting the values. In general priniciple of broadcasting, if you have a (m,n) matrix and you conduct a polynomial computation with a $(1,n)$ matrix, then it will duplicate the latter matrix m times into a (m,n) matrix. Then, it would apply the calculations element-wise.
- Gradient Descent converges faster after normalization of the input matrices.

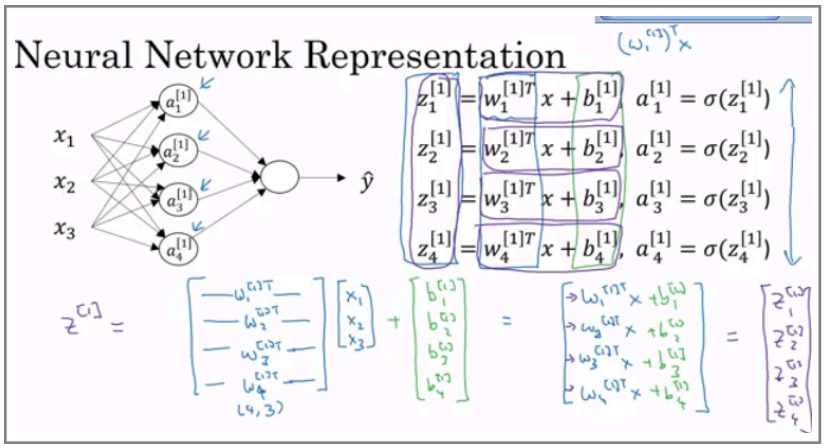
3. Shallow Neural Networks

How to build a Neural Network

1. Define the model structure, such as number of features or hidden layers, etc.
2. Initialize the model's parameters
3. Loop
    1. Calculate current loss (forward propagation)
    2. Calculate current gradient (backward propagation)
    3. Update parameters (gradient descent)

- Preprocessing the dataset is important.
- Tuning the learning rate (which is an example of a "hyperparameter") can make a big difference to the algorithm.

Shallow Neural Network Representation



Equations of Hidden Layers

Exercise3. What are the shapes of each variables?

Activation Functions

- **Sigmoid** can lead to a gradient descent problem where the updates are too slow because its range is [0,1].
- **Tanh** activation function's range is [-1,1]; it is a shifted version of sigmoid function.
    - It turns out that the tanh activation usually works better than sigmoid activation function for hidden units because the means of its output is closer to zero, and so it centers the data better for the next layer.
    - However, the disadvantage for both sigmoid and tanh function is that if the input is too small or too big, the slope will be near zero which will cause the gradient descent problem.

- One of the popular activation functions that solved the slow gradient descent is the **ReLU** function. ReLU(z) = $max(0,z)$; if z is negative, the slope would be 0 and if z is positive the slope remains linear.
- Leaky ReLU activatino function differs from ReLU in that if the input is negative, the slope will be so small, but not 0. Leaky ReLU(z) = $max(0.01z, z)$
- If your classification is between 0 and 1, use the output activation as sigmoid and the others as ReLU. However, it turns out that there are no explicit guidelines. You should try all activation functions!

Random Initialization

- In logistic regression, it wasn't important to initialize the weights randomly, while in NN we have to initialize them randomly.
- If we initialize all the weights with zeros in NN, it won't work. (Initializing bias with zero is OK)
  - All hidden units will be completely identical (symmetric); computing exactly the same fucntion.
  - On each gradient descent iteration, all the hidden units will always update the same.

- We need small values because in sigmoid or tanh function, for example, if the weight is too large, you are more likely to end up even at the very start of training with very large values of Z. Such incident would cause your tanh or sigmoid activation function to be saturated, thus slowing down learning steps. If you don't have any sigmoid or tanh activation functions throughout your neural network, this is less of an issue.

4. Deep Neural Networks

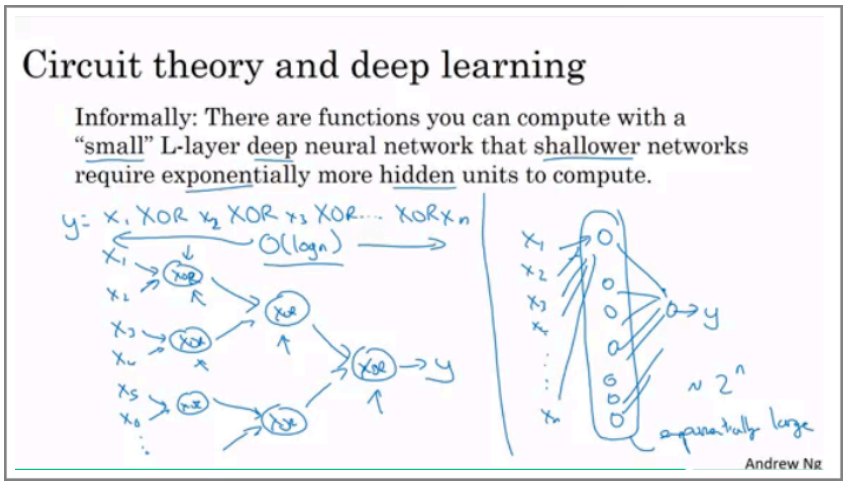
Why Deep Representations?

Deep NN makes relations with data from simpler to more complex. In each layer, it tries to make a relation with the previous layers, such as face recognition application or audio recognition application. Natural researchers think that deep neural netwroks "think" like human brains.

The analogy that "It's like human brains." has become really an oversimplified explanation. There is a very simplistic analogy between a single logistic uni and a single neuron in the brain, and no human today have understand how a human brain works.

The field of computer vision has taken a bit more inspiratino from the humn brains than other disciplines that also apply deep leraning.

When starting on an application, however, don't start directly to dozens of hidden layers. Try the simplest solution, then try the shallow neural network and so on.



Circuit Theory and Deep Learning

Parameters and Hyperparameters

- Main parameters of the NN are W and b
- Hyperparameters (parameters that control the algorithm) are like:
  - Learning rate, Number of iterations, Number of hidden layers L, Number of hidden units n, choice of activation functions etc.
- You have to try values yourself of hyperparemters.
- In the earlier days of DL and ML, learning rate was often called a parameter, but it really is (and now everybody calls it) a hyperparameter.