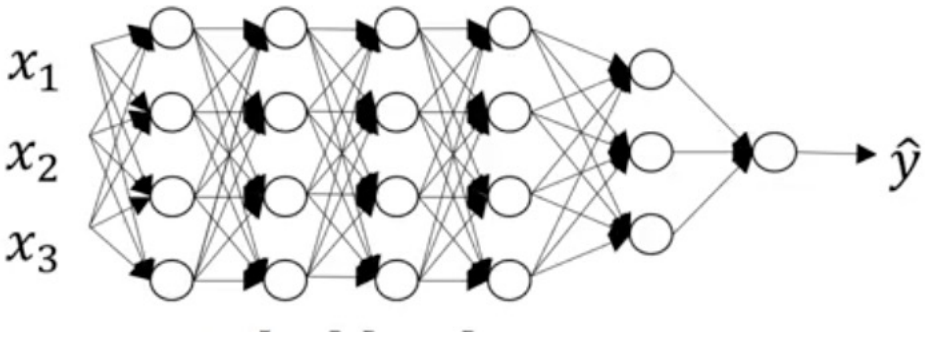
Weekly Report of deep learning

Haoyu Wang
School of Software Engineering
Shanghai Jiao Tong University
gogowhy@sjtu.edu.cn

Abstract

In the past few weeks, I finished the basic knowledge of deep learning,here are someof its details .

1 deep neural netwwork



In different layers, the network finish different things, from the outline to its details through each parts of the subject.For example,when entering a human face, the first layer recognize its outline and edge, the second figure out its part and the final layer tells its face.

Its forward prog is as follows:

Input $a^{[l-1]}$

Output $a^{[l]}$, cache $(z^{[l]})$

Its back prog is as follows:

Input $da^{[l]}$

Output $da^{[l-1]}, dW^{[l]}, db^{[l]}$

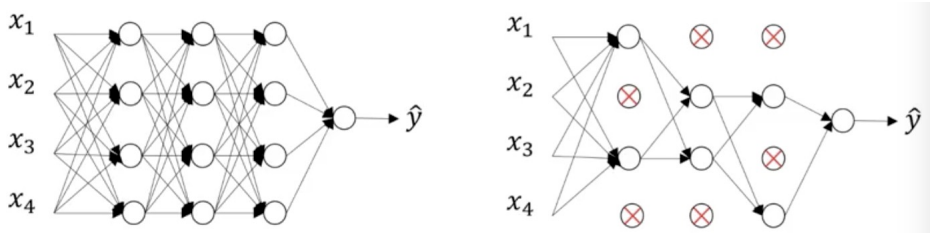
In a deep neural network,there are some parameters as follows:

Parameter:W[1],b[1],W[2],b[2].....
hyperparameter: learning rate α,number of iterations, number of hidden layers,number of hidden units, choice of activization function.....

2 deep neural netwwork

2.1 Regularization

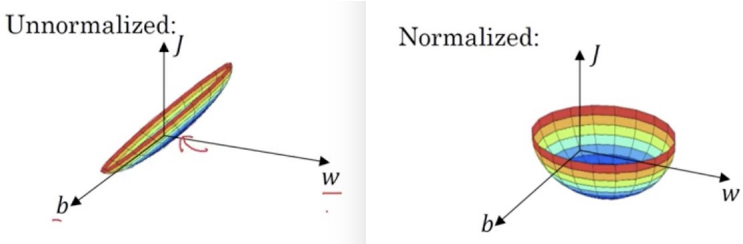
Regularization reduces overfitting(high variance),because in such function ,there can be only very few values for the function to be from minus1 to 1,which can totally reduce the chances of overfitting.
Here are some details of dropout regularization:
With dropout, what we're going to do is go through each of the layers of the network and set some probability of eliminating a node in neural network. we're going to, for each node, toss a coin and have a 0.5 chance of keeping each node and 0.5 chance of removing each node. So, after the coin tosses, maybe we'll decide to eliminate those nodes, then what you do is actually remove all the outgoing things from that no as well.



The net work is changed as above, which can definitely reduce the degree of reliability on each unit of the network. There are few ways to implement dropout:
Inverted dropout: illustrate with layer l=3,
$d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep.probkeep.prob = 0.8(from0to1)$
So, what it does is it generates a random matrix. And this works as well if you have factorized. So d3 will be a matrix. Therefore, each example have a each hidden unit there's a 0.8 chance that the corresponding d3 will be one, and a 20 percent chance there will be zero. So, this random numbers being less than 0.8 it has a 0.8 chance of being one or be true, and 0.2 chance of being false, of being zero
$a3 = np.multiply(a3, d3)$
$a3/ = keep.prob$
After doing so ,the the nn does not rely on any one feature, so we have to spread out weights.

2.2 Normalizing inputs

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



What's more,sometimes the gradients of a very deep NN explodes or vanishes if all of them are above or under 1 even a bit.So we should take good care of weight initialization and gradient checking to prevent this phenomenon.
$var(Wi) = \frac{1}{n}$
$W[l] = np.random.randn(shape) * np.sqrt(\frac{1}{n[l-1]})$
Then we should do something with Numerical approximation of gradients to check the gradient, then we should take all of the W and b and reshape them into a very big vectorθ, and then take dW

and db into a a big vector dθ.
for each i:
$d\theta(approx)[i] = \frac{J(\theta1,\theta2,.....\theta i+\epsilon,....)-J(\theta1,\theta2,.....\theta i-\epsilon,....)}{2\epsilon}$ we check if dθ == $d\theta(approx)$

2.3 optimization

2.3.1 mini-batch gradient descent

vectorization allows us to efficiently compute on all m examples, that allows to process thrwhole training set without an explicit formula. We can then divide the training set into totally different mini-sets. For exmaple, we divide an X(5,000,000)into x1,x2 ,.....xt:
for each xt:
foeward prog on xt
Zt=w[1]Xt+b[1]
a[1]=g(Z[1])
.....a[l]=g(z[l])
Then we do the cost function J=$\frac{1}{1000}\sum_{i=1}^{N} xiyi + \frac{\lambda}{1000}$
back prog and update the weights

2.3.2 exponentially weighted averages

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

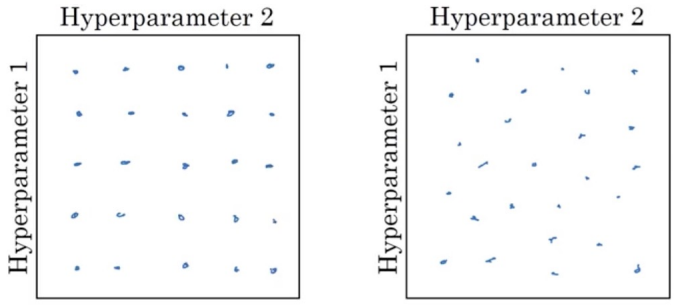
In the way above ,we can do the expectation with the effect of the former examples as well as taking very few memory, which can be used in BIAS correction.

2.3.3 supplement

There are some other ways of optimization that I've know, such as RMSprop, Adam optimization,Learning rate decay and so on.

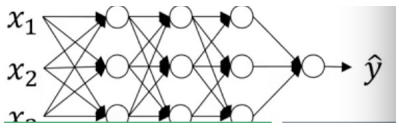
2.4 hyperparmater tuning

we shouldn't sample the points in a grid(left),instead we should pick it at random(right):



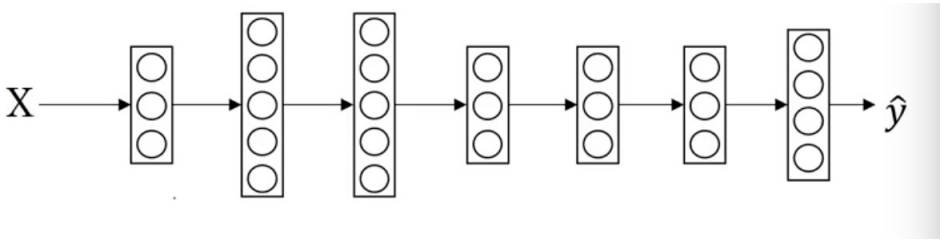
Caution: the hyperparameter should be chosen at a correct scale.

2.5 Batch normalization



in such a deepnetwork,we can normalize a through normalizing z:
$\mu = \frac{1}{m}\sum_{i=1}^{N} z(i)$
$\sigma^2 = \frac{1}{m}\sum_{i=1}^{N}(z(i)-\mu)^2$
$z(i)norm = \frac{z(i)-\mu}{\sqrt{(\sigma^2+\epsilon)}}$
$z(i) = \gamma * z(i)norm + \beta$
Batch Normalization works because this is doing a similar thing, but further values in your hidden units and not just for your input there. hat batch norm does, is it reduces the amount that the distribution of these hidden unit values shifts around.nd indeed they will change when the neural network updates the parameters in the earlier layers. But what batch norm ensures is that no matter how it changes, the mean and variance can always be the same.

2.6 Multi-class classification



we use the SOFTMAX REGRESSION to finish that:
$t = e^(z[L])$
$a[L] = \frac{e^(z[L])}{\sum_{i=1}^{N} ti}$
$a(i)[L] = \frac{ti}{\sum_{i=1}^{N} ti}$
for an example:

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} \quad g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

So we notice that in the z vector, the biggest element was 5, and the biggest probability ends up being this first probability
To calculate the loss function of SOFTMAX regression:

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, y(train) = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} ThenL(y, y(train)) = -\sum_{i=1}^{4} yi \log y(train)i$$

The Back prog is :
dZ[l]=y(train)-y ,which is going by our usual definition of what is dz, this is the partial derivative of the class function with respect to z[L].