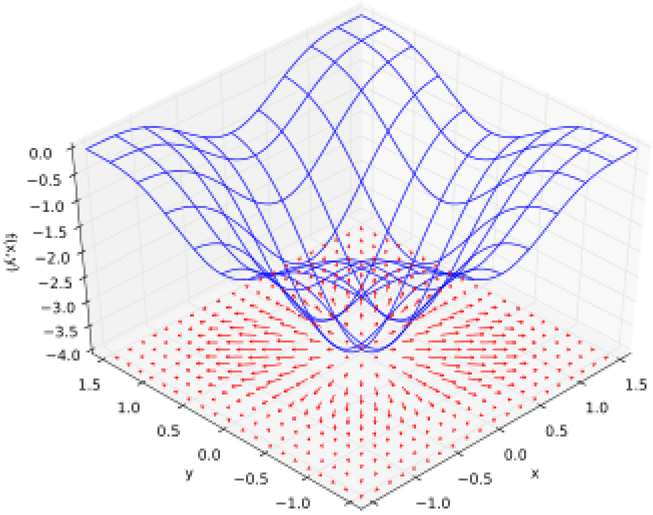
Camron Godbout Follow
Deep Learning and NLP enthusiast
Dec 6, 2016 · 5 min read

Deep Learning Cheat Sheet

Deep Learning can be overwhelming when new to the subject. Here are some cheats and tips to get you through it.

What're Those?!?!

In this article we will go over common concepts found in Deep Learning to help get started on this amazing subject.



Visualization of gradient. The red arrows are the gradient of the blue plotted function.

Gradient ∇ (Nabla)

The gradient is the partial derivative of a function that takes in multiple vectors and outputs a single value (i.e. our cost functions in Neural Networks). The gradient tells us which direction to go on the graph to increase our output if we increase our variable input. We use the gradient and go in the opposite direction since we want to decrease our loss.

Back Propagation

Also known as back prop, this is the process of back tracking errors through the weights of the network after forward propagating inputs through the network. This is used by applying the chain rule in calculus.

Sigmoid σ

A function used to activate weights in our network in the interval of [0, 1]. This function graphed out looks like an 'S' which is where this function gets is name, the s is sigma in greek. Also known as the logistic function

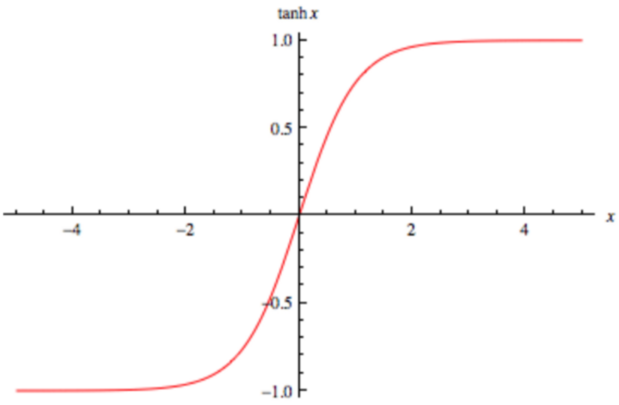
$$\sum_{i=1}^{N} \sigma(x - i + 0.5) \approx \log(1 + e^{x}),$$

where $x = \mathbf{v}\mathbf{w}^T + b.$

Formula for ReLU from Geoffrey Hinton

Rectified Linear Units or ReLU

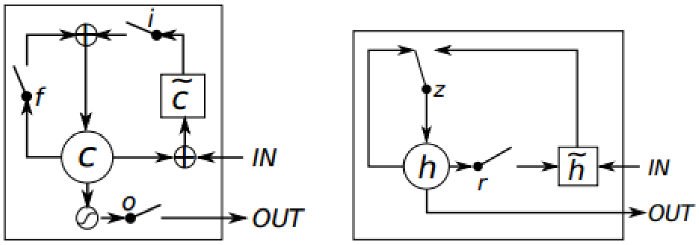
The sigmoid function has an interval of [0,1], while the ReLU has a range from [0, infinity]. This means that the sigmoid is better for logistic regression and the ReLU is better at representing positive numbers. The ReLU do not suffer from the vanishing gradient problem.



Tanh function

Tanh

Tanh is a function used to initialize the weights of your network of [-1, 1]. Assuming your data is normalized we will have stronger gradients: since data is centered around 0, the derivatives are higher. To see this, calculate the derivative of the tanh function and notice that input values are in the range [0,1].The range of the tanh function is [-1,1] and that of the sigmoid function is [0,1]. This also avoids bias in the gradients.



(a) Long Short-Term Memory

(b) Gated Recurrent Unit

Figure 1: Illustration of (a) LSTM and (b) gated recurrent units. (a) i, f and o are the input, forget and output gates, respectively. c and $\tilde{c}$ denote the memory cell and the new memory cell content. (b) r and z are the reset and update gates, and h and $\tilde{h}$ are the activation and the candidate activation.

LSTM/GRU

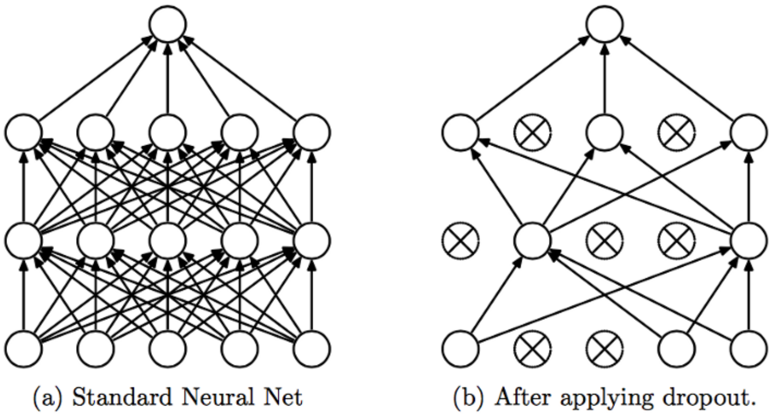
Typically found in Recurrent Neural Networks but are expanding to use in others these are little “memory units” that keep state between inputs for training and help solve the vanishing gradient problem where after around 7 time steps an RNN loses context of the input prior.

Softmax

Softmax is a function usually used at the end of a Neural Network for classification. This function does a multinomial logistic regression and is generally used for multi class classification. Usually paired with cross entropy as the loss function.

L1 and L2 Regularization

These regularization methods prevent overfitting by imposing a penalty on the coefficients. L1 can yield sparse models while L2 cannot. Regularization is used to specify model complexity. This is important because it allows your model to generalize better and not overfit to the training data.



Drop out

[1] “It prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently“(Hinton). This method randomly picks visible and hidden units to drop from the network. This is usually determined by picking a layer percentage dropout.

Batch Normalization

[1] When networks have many deep layers there becomes an issue of internal covariate shift. The shift is “the change in the distribution of network activations due to the change in network parameters during training.” (Szegedy). If we can reduce internal covariate shift we can train faster and better. Batch Normalization solves this problem by normalizing each batch into the network by both mean and variance.

Objective Functions

Also known as **loss function, cost function** or **opimization score function**. The goal of a network is to minimize the loss to maximize the accuracy of the network.

A more technical explanation [3]:

The loss/cost/optimization/objective function is the function that is computed on the prediction of your network. Your network will output a prediction, y_hat which we will compare to the desired output of y. This function then is used in back propagation to give us our gradient to allow our network to be optimized. Examples of these functions are f1/f score, categorical cross entropy, mean squared error, mean absolute error, hinge loss… etc.

F1/F Score

A measure of how accurate a model is by using precision and recall following a formula of:

```
F1 = 2 * (Precision * Recall) / (Precision + Recall)
```

Precise: of every prediction which ones are actually positive?

```
Precision = True Positives / (True Positives + False
Positives)
```

Recall: of all that actually have positive predicitions what fraction actually were positive?

```
Recall = True Positives / (True Positives + False
Negatives)
```

Cross Entropy

Used to calculate how far off your label prediction is. Some times denoted by CE.

$$loss = -\sum_{i}(y_i.log(y\_predicted_i) + (1 - y_i).log(1 - y\_predicted_i))$$

Cross entropy is a loss function is related to the entropy of thermodynamics concept of entropy. This is used in multi class classification to find the error in the predicition.

Learning Rate [2]

The learning rate is the magnitude at which you're adjusting your weights of the network during optimization after back propagation. The learning rate is a hyper parameter that will be different for a variety of problems. This should be cross validated on.

```
# ---- Vanilla Gradient Descent ----
# W is your matrix of weights
# dW is the gradient of W

# Adjust the weights of our layer by multiplying our
gradient times our learning_rate (Which is a scalar value)

W -= learning_rate * dW
```

> *This is a living document, if anything you think should be added let me know and I will add and credit you.*

[1] * Suggested by: InflationAaron

[2] * Suggested by: Juan De Dios Santos

[3] * Suggested by: Brandon Fryslie

To take a look at different types of models and sample code check out TensorFlow in a Nutshell—Part Three: All the Models.

As always check out my other articles at camron.xyz