Tuning & Debugging Neural Networks

CMSC 389A: Lecture 6

Sujith Vishwajith

University of Maryland

Agenda

1. Practical 1 Review
2. Loss Functions
3. Visualizing Training
4. Dropout
5. Optimizers
6. Hyperparameter Selection

Practical 1 Review

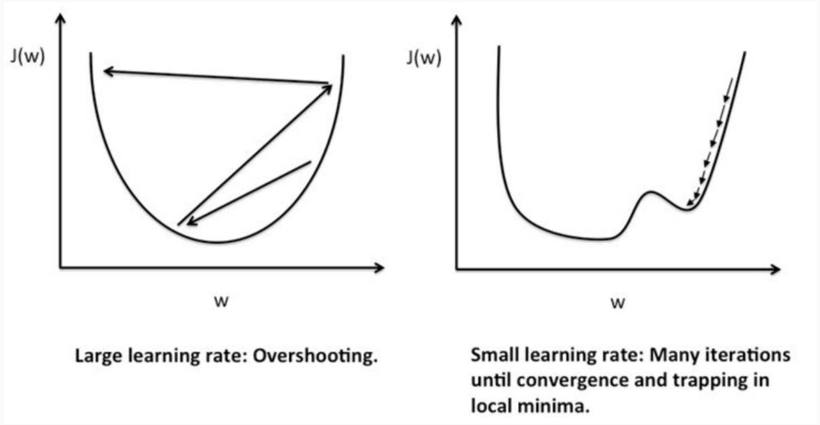
Overview of coding part.

Analysis - Question 1

Learning rate is how fast weights are updated/changed based on new examples.

Higher learning rate means more rapid changes but can case model to jump back and forth over optimal weights.

Low learning rate takes too long to converge and can get caught in a local minima.


J(w)
w
Large learning rate: Overshooting.
J(w)
w
Small learning rate: Many iterations until convergence and trapping in local minima.

Analysis - Question 2

Gave everyone full credit because question wasn’t explained well.

One pass over training data refers to an **epoch**.

Our model was trained for 100 epochs so 100 passes over training data.

Analysis - Question 3/4

We can look at the weights to determine which features are good or bad.

Features with a high magnitude (absolute value) weights are good predictors.

Best Features: *Plasma glucose concentration a 2 hours, BMI*

Features with weights close to 0 are bad predictors.

Worst Features: *Triceps skin fold thickness, Age*

Intuitively the larger the value in terms of absolute value, the more it contributes.

Note: Ideally features should be normalized to get accurate results.

Analysis - Question 5

Dropped because question didn’t make sense.

Loss Functions

Loss Functions

Also known as a cost-function.

Intuitively, the loss function measures distance between our predictions and the truth.

We want to minimize this function over time indicating that our model is getting better.

You can also output a negative cost value and the goal can be to maximize it over time. Think about it!

Should be indicative of how well the model is doing at the task at hand.

Mean Squared Loss

Extremely simple but useful

Distance between prediction and label

Computed by averaging $(\text{prediction-label})^2$

Cross-Entropy Loss

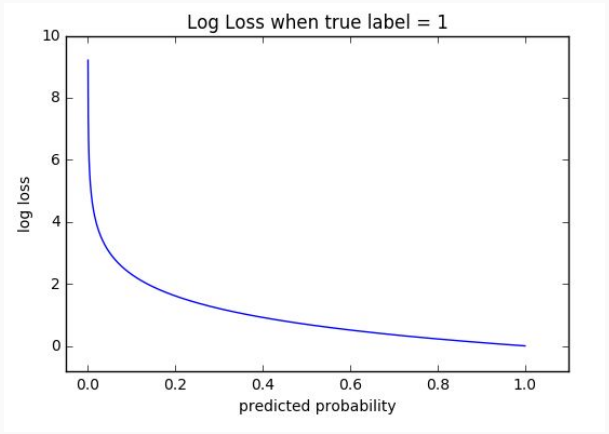
Measures performance of model which outputs probability between 0 and 1.

Loss increases as true label diverges from predicted probability.

For binary classification: −(y * log(p) + (1−y) * log(1−p))

For multi-class: Calculate a separate loss for each class label per observation and sum the result.

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$



L1 and L2 Loss Functions

L1 Loss function minimizes the **absolute differences** between the estimated values and the existing target values.

$$S = \sum_{i=0}^{n} (y_i - h(x_i))^2$$

L2 loss function minimizes the **squared differences** between the estimated and existing target values.

$$S = \sum_{i=0}^{n} |y_i - h(x_i)|$$

L1 loss is more robust and not sensitive to outliers while L2 is heavily affected by outliers.

Visualizing Training

Validation Set

Similar to how we keep aside some of our data for testing, we can keep aside some for validation. For example keeping aside 10% of the data.

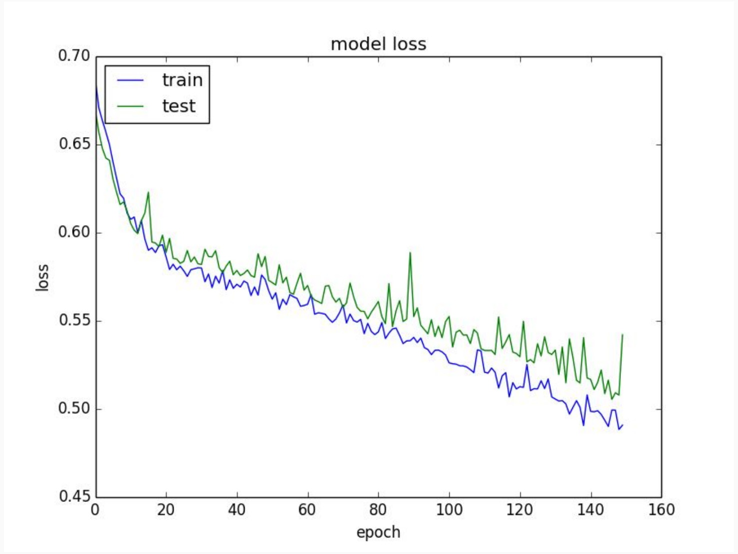
Validation data isn’t trained on but used during training time to get a live update on how our model is doing on unseen data.

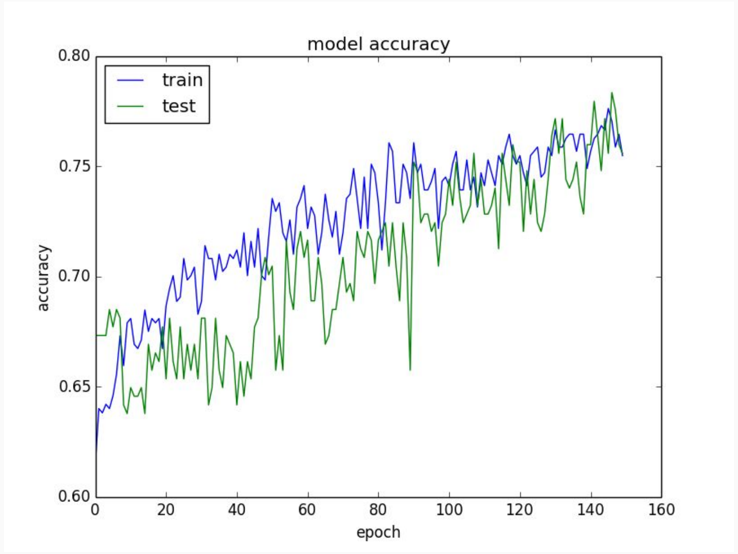
Most of the time we don’t have test data so we use validation sets to make sure our model is generalizing.

Visualizing Training

We can plot our metrics over time based on both the validation and training data.

Loss should have a downward trend while accuracy should be upwards.

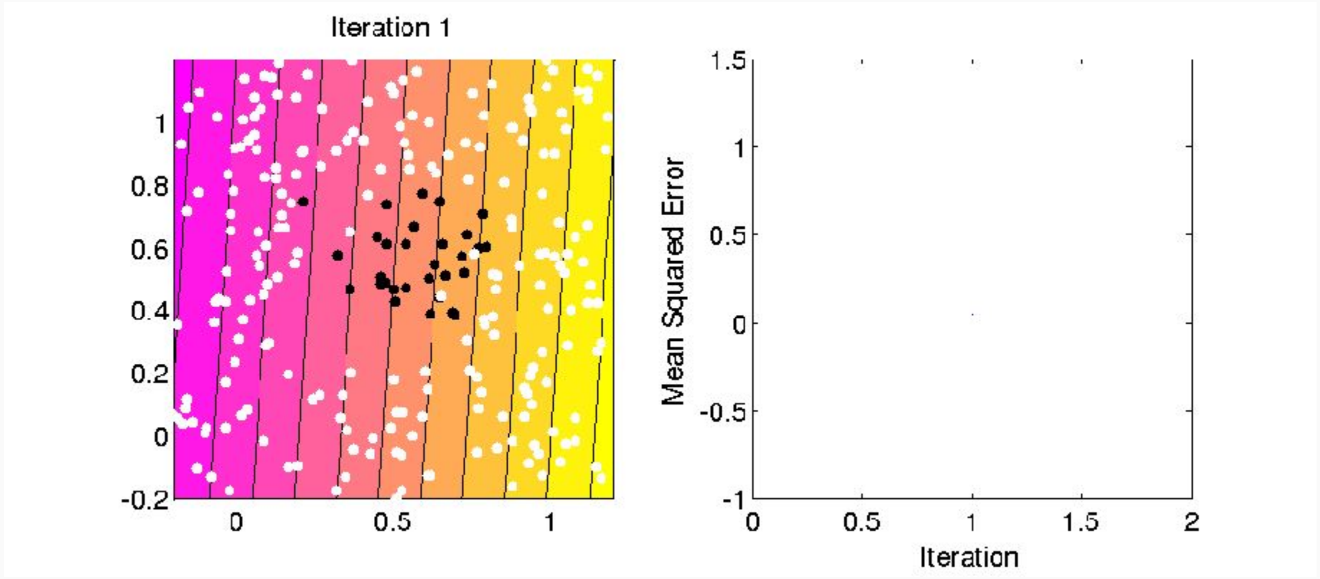




Visualizing Decision Boundaries

Helps you see the data and how our model tries to seperate it.

Not very feasible for more than 3 dimensions.



Dropout

Dropout Overview

Goal: Prevent overfitting the data as it avoids neuron units developing co-dependencies between each other.

Randomly choosing units to ignore during training for a forward or backwards pass.

Probability that the unit is dropped is *p* and known as the dropout rate.

In Keras we can simply add a dropout layer after each Dense layer and specify the dropout rate.

```
model.add(Dropout(0.2))
```

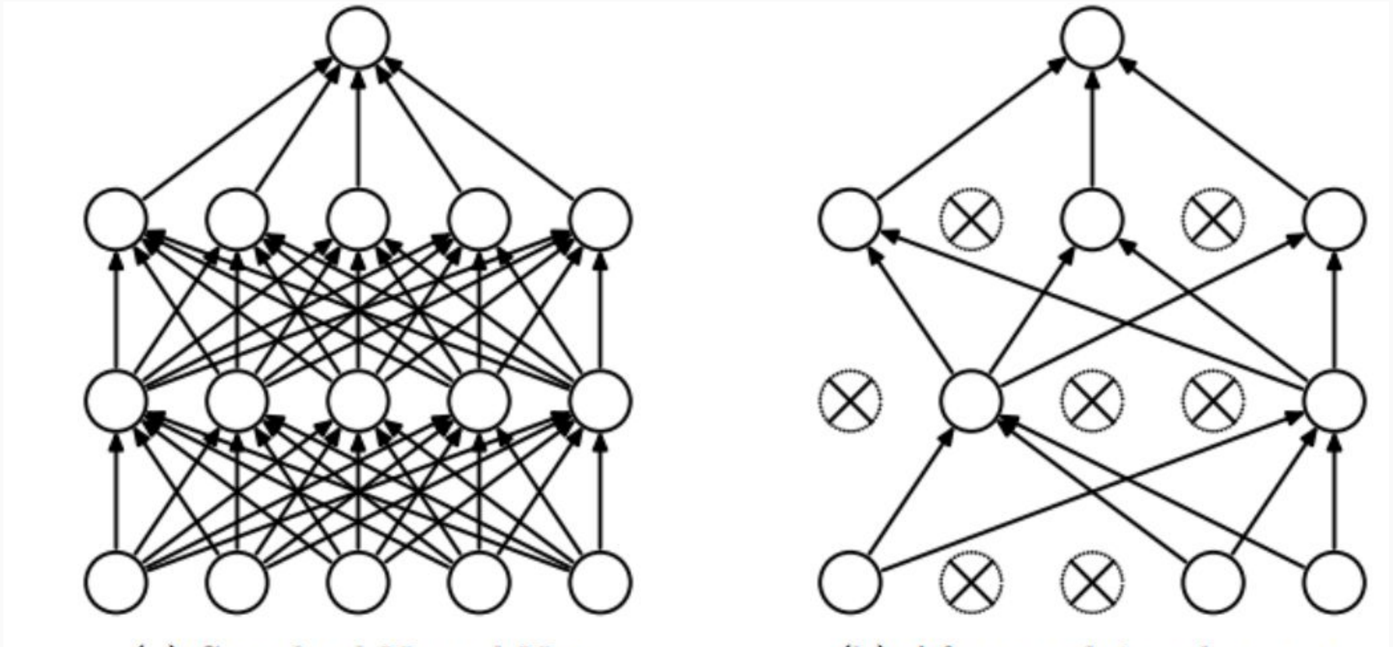
Results

Forces a network to learn more robust features.

May cause poorer performance on training dataset but that’s okay! We care about test data performance more.

Requires more iterations to converge (more epochs) but smaller training time since parameters are dropped.

Visualization of Dropout


(a) Standard Neural Net (b) After applying dropout.

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

Optimizers

Optimizers Overview

Optimizers are algorithms used to minimize (or maximize) loss functions.

The most common one and easiest to understand is stochastic gradient descent.

More complex optimization algorithms use the second and third derivatives to update.

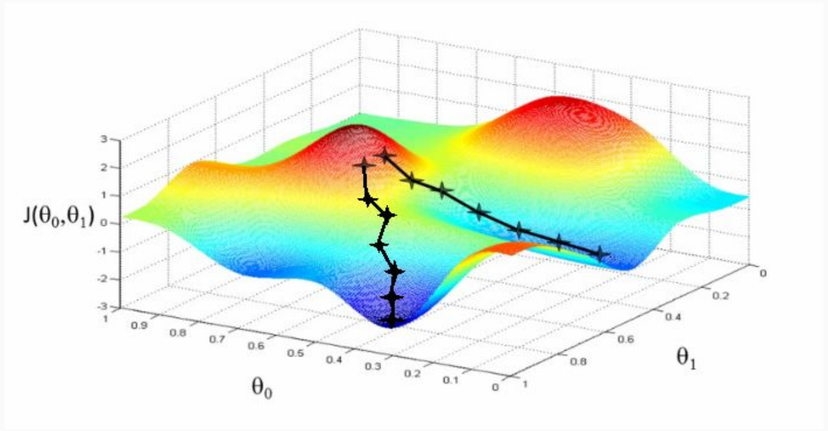
Intuition: Finding a path to move downwards in a hill to the lowest point.

Gradient Descent

First order optimization algorithm (meaning only uses first derivatives).

Computes gradients of the loss function and moves in the opposite direction to minimize the loss as much as possible.

Incredibly powerful but also very basic.

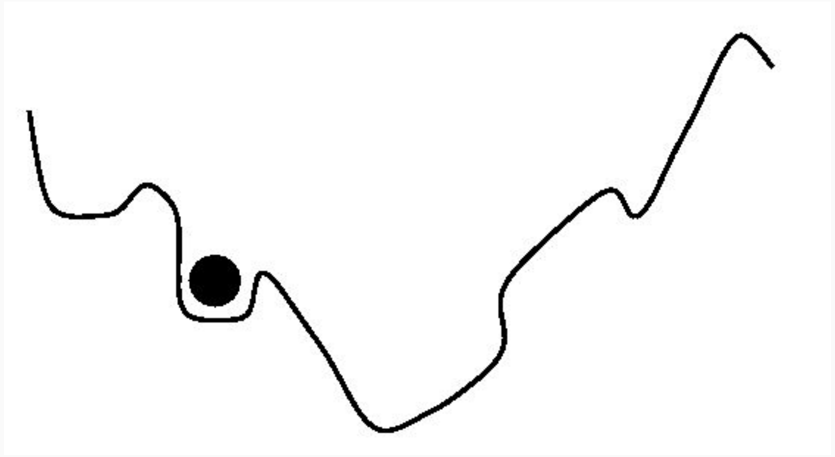


Momentum

SGD often has a tendency of getting stuck in local minima.

We can use the concept of momentum to help it get over a local minima.

Think of a ball rolling down a hill. Does it stop immediately once it hits the bottom or does it keep rolling up back and forth till it stabilizes?

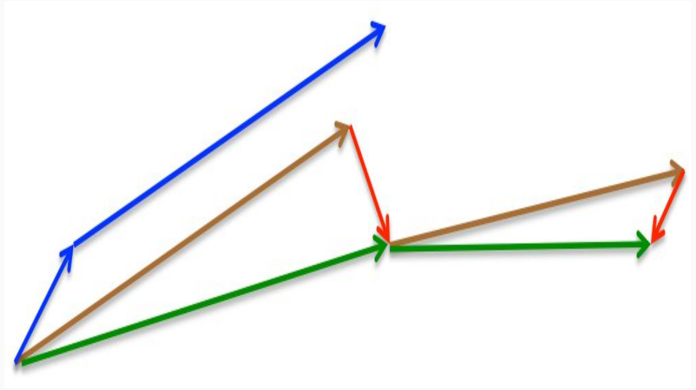


Nesterov Momentum

In regular momentum the ball blindly follows the slope of the hill.

Goal is to have a smarter ball that understands where it is going so that it knows to slow down before the hill slopes up again.

Measures the gradient and then corrects it to adjust for how it’s rolling down (looks at what the future parameters will be).



Adagrad

Adapts our updates to individual parameters to perform larger or smaller update steps depending on the frequency of the parameters.

Uses a different learning rate for every parameter.

However has a mathematical flaw causing the learning rate to eventually become too small.

Adam

Adaptive Moment Estimation (Adam)

Computes adaptive learning rates for each parameter.

Keeps an exponentially decaying average of past gradients.

Other Optimizers

Many other optimizers exist and are constantly being invented:

- Nadam
- RMSprop
- AMSgrad
- AdaMax
- Adadelta
- etc...

Hyperparameter Selection

Hyperparameters in a Model

Hyperparameters we can choose include:

- Learning rate
- Optimizer
- Loss Function
- Number of Epochs
- Number of Neurons per Layer
- Number of Layers
- And more….

How do we know which hyperparameters will work and which are the best?

We don’t!

Hyperparameter Selection

Often best to start small and build up.

One way to decide what hyperparameter to use is by guessing and checking which works better than the other. E.g. trying out different learning rates, increasing number of layers…

Another way is to do a hyperparameter sweep which is an automated approach that experiments with a grid of parameters options and selects the best one.

Downside: **Extremely slow** as it tries out all the different combinations.

Announcements

Announcements

Practical 2 is due today by **11:59 p.m**.

For those having trouble installing the packages you can use Google Collab.

Questions?