CSE 465
Lecture 10

Deep learning projects

Tuning Hyper Parameters

Parameters vs Hyperparameters

- What are Hyperparameters?
  - In statistics, a hyperparameter is a parameter from a prior distribution; it captures the prior belief before data is observed
  - These parameters need to be initialized before training a model
- **Hyperparameters** are the variables that we set and tune
- **Parameters** are the variables that the network updates using the optimization algorithm (gradient descent)
  - They are learned and updated by the network during training, and we do not adjust them
  - In neural networks, parameters are the weights and biases that are optimized automatically during the backpropagation process
- In contrast, hyperparameters are variables that are not learned by the network
  - They are set by the ML engineer before training the model and then tuned
- These are variables that define the network structure and determine how the network is trained
- Hyperparameter examples include learning rate, batch size, number of epochs, number of hidden layers, etc.

Hyperparameters

- We can categorize neural network hyperparameters into three main categories:
- Network architecture
  - Number of hidden layers (network depth)
  - Number of neurons in each layer (layer width)
  - Activation type
- Learning and optimization
  - Learning rate and decay schedule
  - Mini-batch size
  - Optimization algorithms
  - Number of training iterations or epochs (and early stopping criteria)
- Regularization techniques to avoid overfitting
  - L2 regularization
  - Dropout layers
  - Data augmentation

Deep Learning Projects: Hyperparameters

- Its impossible to get all hyperparameters right on a new application from the first time
- The idea is to go through the loop: Idea -- > Code --> Experiment

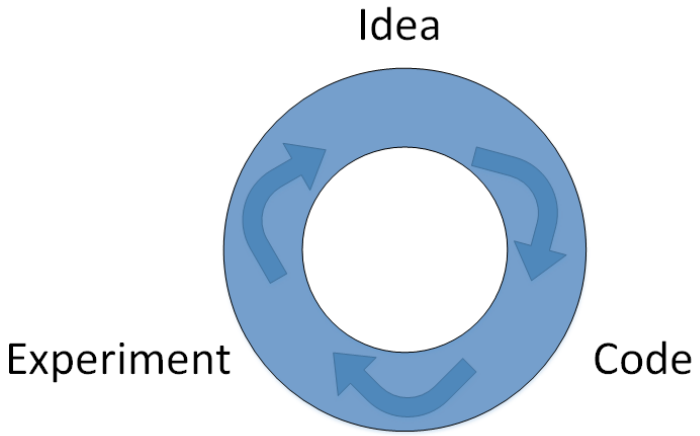
layers

hidden units

learning rates

activation functions

...



- We have to go through the loop many times to figure out the hyperparameters

Deep Learning project: Data Split

Data will be split into three parts

Training set (Has to be the largest set)

Hold-out cross validation set / Development or "dev" set

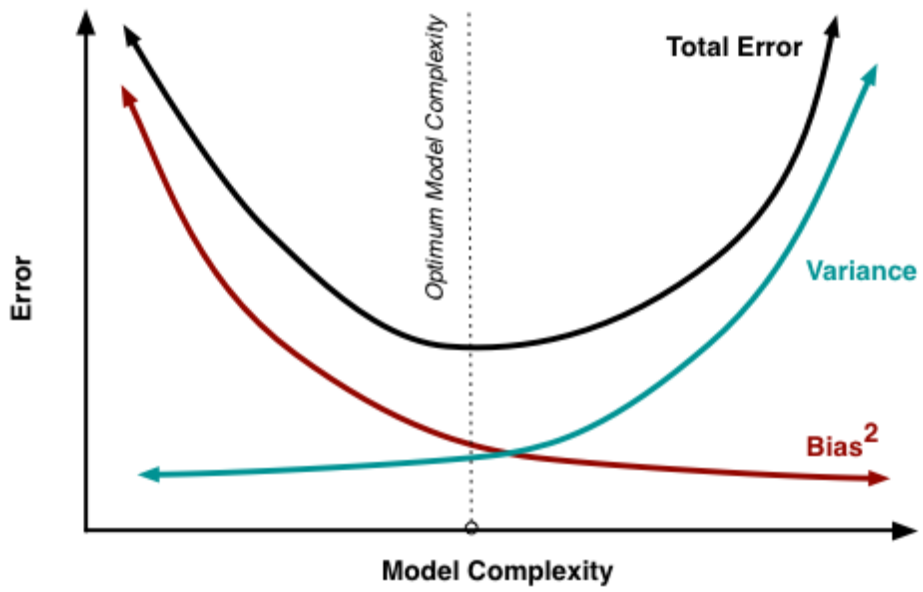
Testing set

We build models using the **training set** then try to optimize hyperparameters on **dev set** as much as possible. After the model is ready, we try and evaluate it on the **testing set**

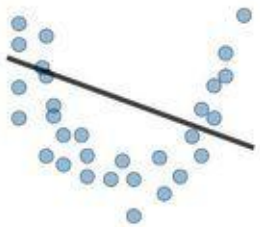
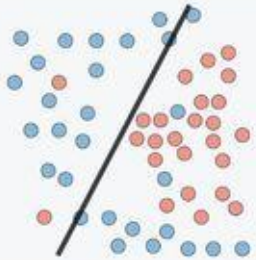
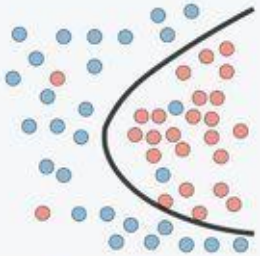
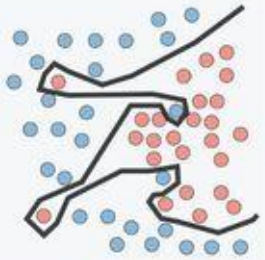
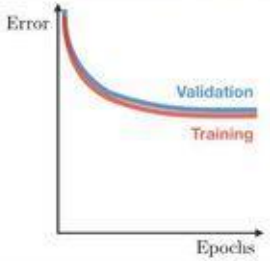
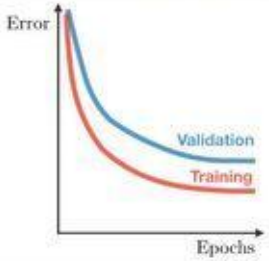
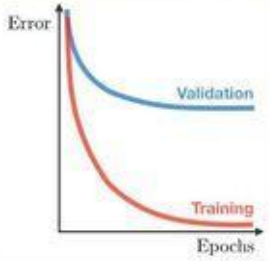
Deep Learning Project: More on Data Split

What is the ratio of splitting the models?	If size of the dataset is 100 to 1 Million: 60/20/20 If size of the dataset is 1 Million to Inf: 98/1/1 or 99.5/0.25/0.25
Current practices takes most of the data for the training	
Also make sure the dev and test set are coming from the same distribution	If cat training pictures is from the web and the dev/test pictures are from user's cell phone they will mismatch It is better to make sure that dev and test set are from the same distribution
The dev set rule is to try them on some of the good models we've created	
Its OK to only have a dev set without a testing set	
But a lot of people in this case call the dev set the test set	
A better terminology is to call it a dev set as its used in the development	

Deep Learning Projects: Bias/Variance



- What is Bias/Variance?
  - If a model is underfitting (logistic regression of nonlinear data) it has a "high bias"
  - If a model is overfitting, then it has a "high variance"
  - We need to find a sweet spot in between to balance the Bias/Variance

	Underfitting	Just right	Overfitting
Symptoms	• High training error • Training error close to test error • High bias	• Training error slightly lower than test error	• Very low training error • Training error much lower than test error • High variance
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	• Complexify model • Add more features • Train longer		• Perform regularization • Get more data

Bias/Variance how to find balance

High variance (overfitting) example

- Training error: 1%
- Dev error: 11%

High bias (underfitting) for example

- Training error: 15%
- Dev error: 14%

High Bias (underfitting) and High variance (overfitting) example

- Training error: 15%
- Dev error: 30%

A balanced example

- Training error: 0.5%
- Dev error: 1%

Recipe for a deep learning project

If your algorithm has a high bias

Try to make your network bigger (size of hidden units, number of layers)	Try a more complex model that is suitable for your data	Try to run it longer	**Use more advanced optimization algorithms**

If your algorithm has a high variance

Use more data	**Try regularization**	Try a less complex model that is suitable for your data

You should try the previous two points until you have a low bias and low variance

Projects & Metrics

How to work on the project?

- Deep Learning is a very empirical process
  - It relies on running experiments and observing model performance more than having one go-to formula for success that fits all problems
- We often have an initial idea for a solution, code it up, run the experiment to see how it did, and then use the outcome of this experiment to refine our ideas
- When building and tuning a neural network we try to resolve the following questions:
  - What is a good architecture to start with?
  - How many hidden layers should we stack?
  - How many hidden units or filters should go in each layer?
  - What is the learning rate?
  - Which activation function should we use?
  - Which yields better results, getting more data or tuning hyperparameters?

Defining performance metrics: Accuracy

- Performance metrics allow us to evaluate our system
- When we develop a model, we want to find out how well it is working
- The simplest way to measure the "goodness" of our model is by measuring its accuracy
- The accuracy metric measures how many times our model made the correct prediction
- So, if we test the model with 100 input samples, and it made the correct prediction 90 times, this means the model is 90% accurate

$$\text{accuracy} = \frac{\text{correct predictions}}{\text{total number of examples}}$$

Confusion matrix & related scores

- A confusion matrix is a table that is often used to **describe the performance of a classification model** (or "classifier") on a set of test data for which the true values are known
- What can we learn from this matrix?
  - There are two possible predicted classes: "yes" and "no“
  - If we were predicting the presence of a disease, for example, "yes" would mean they have the disease, and "no" would mean they don't have the disease
  - The classifier made a total of 165 predictions (e.g., 165 patients were being tested for the presence of that disease)
  - Out of those 165 cases, the classifier predicted "yes" 110 times, and "no" 55 times
  - In reality, 105 patients in the sample have the disease, and 60 patients do not

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

Confusion matrix terms

- **True Positives (TP):** These are cases in which we predicted yes (they have the disease), and they do have the disease
- **True Negatives (TN):** We predicted no, and they don't have the disease
- **False Positives (FP):** We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
- **False Negatives (FN):** We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Confusion matrix other terms

- **Accuracy:** Overall, how often is the classifier correct?
  - (TP+TN)/total = (100+50)/165 = 0.91
- **Misclassification Rate:** Overall, how often is it wrong?
  - (FP+FN)/total = (10+5)/165 = 0.09
  - equivalent to 1 minus Accuracy
  - also known as "Error Rate"
- **True Positive Rate:** When it's actually yes, how often does it predict yes?
  - TP/actual yes = 100/105 = 0.95
  - also known as "Sensitivity" or "Recall"
- **False Positive Rate:** When it's actually no, how often does it predict yes?
  - FP/actual no = 10/60 = 0.17
- **True Negative Rate:** When it's actually no, how often does it predict no?
  - TN/actual no = 50/60 = 0.83
  - equivalent to 1 minus False Positive Rate
  - also known as "Specificity"
- **Precision:** When it predicts yes, how often is it correct?
  - TP/predicted yes = 100/110 = 0.91
- **Prevalence:** How often does the yes condition actually occur in our sample?
  - actual yes/total = 105/165 = 0.64

Recall

- *Recall* is the rate of correctly identifying the sick patients among all sick patients
- In other words, on average how many times did the model *correctly* diagnose a sick patient as positive
- More related to the positives of the actual dataset
- Recall is calculated by the following equation

$$\text{Recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

Precision

- *Precision* is the opposite of recall
- It tells us how often the identification of sick patients are correct
- In other words, how many times did the model's positive diagnose of a patient as sick is correct
- More related to the positive identification of the model
- Precision is calculated by the following equation:

$$\text{Precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$$

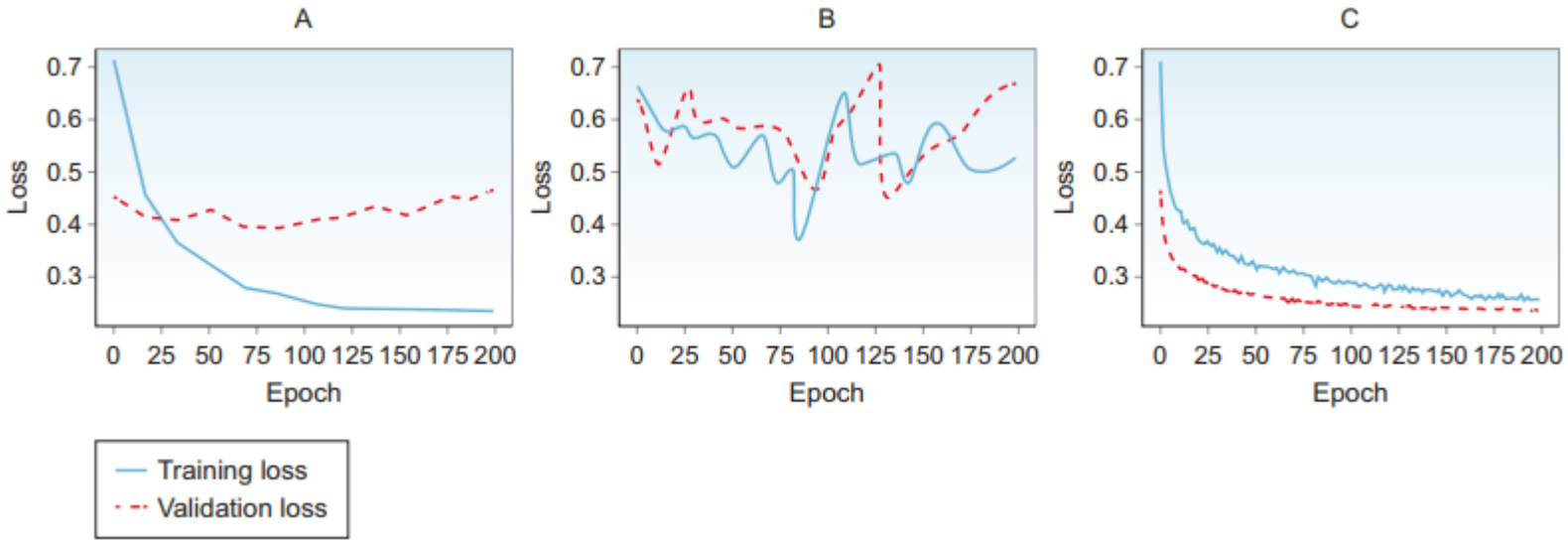
F-score

- In many cases, we want to summarize the performance of a classifier with a single metric that represents both recall and precision
- To do so, we can convert precision (p) and recall (r) into a single F-score metric
- In mathematics, this is called the *harmonic mean* of p and r:

$$\text{F-score} = \frac{2pr}{p+r}$$

Plotting the learning curve

- Looking at training output and comparing numbers is a cumbersome job
  - A better way is to check the plots of training and validation errors
- Plot A below shows that the network improves the loss value on the training data but fails to generalize on the validation data
  - Learning on the validation data progresses in the first couple of epochs and then flattens out and maybe decreases
    - This is a form of overfitting
  - Note that this graph shows that the network is actually learning on the training data, a good sign that training is happening
  - So we don't need to add more hidden units, nor do you need to build a more complex model
  - If anything, network is too complex for the data, because it is learning *so much* that it is actually memorizing the data and failing to generalize to new data
    - In this case the next step might be to collect more data or apply techniques to avoid overfitting



Plotting the learning curve

- Figure B shows that the network performs poorly on both training and validation data
  - In this case the network is not learning
- We don't need more data, because the network is too simple to learn from the data
- Next step is to build a more complex model
- Figure C shows that the network is doing a good job of learning the training data and generalizing to the validation data
  - This means there is a good chance that the network will have good performance out in the wild on test data.