**Slide 5-4**

* In practice we rarely train the convolutional Neural networks from scratch
* Instead we do finetuning
* We almost always do this pre-training and finetuning process

**Slide 5-5**

* Let say we have a CNN that we need to train
* We can train it on ImageNet data for instance and then fine tune it on the smaller dataset
* We can transfer our trained neural network to the small dataset to finetune

**Slide 5-6**

* This fine tuning works like:
  + We start with an image and train the convolutional neural network using Image Net
  + We train all the way to a classifier
  + We than take the pre-trained network and chop off the classifier (top layer)
  + Depending on how much data you have you only train the last layer of the network or you do backprop if you have a larger dataset.

**Slide 5-7**

* Pre-training step is done for you by other ppl
  + Ppl train neural networks for weeks sometimes and then upload the weights onto the internet.
  + Caffe Model zoo. Lots of pretrained ConvNets

https://github.com/BVLC/caffe/wiki/Model-Zoo

**Slide 5-10**

* Reminder
  + We are working in the framework of Mini-Batch Stochastic Gradient Descent to train neural networks

**Slide 5-11**

* When we repeat this process it comes down to an optimization problem where in the weight space we are converging into areas of the weight space where we have low loss.
* This means we are correctly classifying our training set

**Slide 5-12**

* We saw that these NN’s can get very large
* Neural Turing Machine
* Huge computational graphs where we need to do back prop through them

**Slide 5-13**

* Intuitions of Back prop and its really just a recurive application of chain rule from the back of the circuit to the front

**Slide 5-17**

* Neural networks without brain perspective

**Slide 5-18**

* Neural Networks with brain perspective

**Slide 5-20**

* We are now going to talk about how to train Neural Networks effectively
* First a brief history

**Slide 5-25**

* BREAKTHOUGHS IN 2010 & 2012 IN DEEP LEARNING
* First big results for Deep Learning
* 2010 Specifically, in speech recognition area
  + They swapped out GMM/HMM and inserted a Neural Network
  + Their results greatly improved
  + Context-Dependent Pre-Trained Deep Neural Networks for Large Vocabulary Speech Recognition
* 2012 Computer Vision saw large gains
  + Imagenet Classification with deep convolutional neural networks
* Better ways of initializing, GPU availability, much more data available all contributed to the explosion in progress

Slide 5-26: Overview

1. One time setup
2. Training Dynamics
3. Evaluation

**Slide 5-28: ACTIVATION FUNCTIONS**

* Function “f” at top of Neuron

**Slide 5-29**

* Can have many different forms
* We will go through pros and cons and what we want out of an activation function

**Slide 5-30: sigmoid**

* Historically used the most
* Sigmoid non linearity

**Slide 5-31**

* Three problems exist

1. Saturated neurons (Neurons that output very close to 0 or very close to 1) “kill” the gradients during back propagation

**Slide 5-32**

* Sigmoid gate receives some value x and sigma(x) comes out
* In backprop we have dL/dsigma and we would like to backprop it through the sigmoid gate using chain rule so that we have dL/dx.
* Through the sigmoid gate we know that dsigma/dx is multiplied by the gradient
* What happens if we back prop through the circuit if
  + X = -10 or X = 10: gradients would be near or at zero because its slope is zero. Therefore if the neuron is saturated, the gradient is “killed”. Gradient flow stops through sigmoid neuron
  + X = 0: Active region of a sigmoid. Gradients flow through the sigmoid neuron
* If we have a large network of sigmoid neurons and many of those are in saturation; then gradients can’t backpropagation through the network

**Slide 5-33**

* Problem

1. Sigmoid outputs are not zero-centered
   1. When we preprocess our data, we want to make sure it is zero centered

**Slide 5-34**

* Consider a neuron that computes this function
* What can we say about the gradients of w during backpropagation if all x’s are all positive numbers?
  + All gradients of w are either all positive or all negative
  + We end up constrained in the kind of update we can make
  + We end up with a zig zag path
  + We get slower convergence when we train with data that is not zero centered

**Slide 5-36**

* Problem

1. Exp() is a bit expensive to compute

**Slide 5-37**

* Tanh is an attempt to fix some of these problems: zero centering
  + LeCun recommended ppl use Tanh instead of sigmoid. 1991
* Zero centered where outputs are between -1 and 1
* However other problems remain

**Slide 5-38**

* In 2012 paper by Krizhevsky pointed out Rectified Linear Unit
* Using a max non-linearity function makes convergence happen faster by a factor of 6
* **THIS IS THE DEFAULT RECOMMENDATION**

**Slide 5-39**

* One slight annoyance
  + What happens when ReLU outputs a 0?
  + What happens in back prop when a ReLU neuron does not become active?
    - Gradient is “killed” during back prop

**Slide 5-40**

* X = -10: local gradient is zero. So any neuron that does not activate will not back propagate downwards. Its weights will not be updated and it does not contribute to the network
* X = 10: gradient = 1. If during forward pass the output of the ReLU is positive then during backprop the gradient will just be passed through
* X = 0: gradient is undefined

**Slide 5-41**

* Sometimes we can initialize the ReLU neurons badly.
* Suppose this is a data cloud of inputs to the ReLU neurons.
  + We can end up with a dead neuron
  + If a ReLU neuron only activates in a region outside of our data cloud then this dead ReLU will never become activated and it will never update
  + This can happen in one of two ways
    - At initialization we sample weights that will not allow ReLU to turn on
    - During training if learning rate is high

**Slide 5-42**

* Normally these neurons are initialized to zero
* However, initializing the ReLU neurons with a lightly positive bias will make it more likely that these neurons will active

**Slide 5-43**

Ppl trying to fix ReLUs

* Leaky ReLU
  + Mass et al., 2013
  + He et al., 2015
  + f(x) = max(0.01x,x)
  + For a ReLU the gradients die in the negative region
  + A Leaky ReLU adds a positive slope to this region
  + This works slightly better than ReLU (Not Completely Established)

**Slide 5-44**

* Parametric Rectifier (PReLU)
  + F(x) = max(alpha\*x,x)
  + 0.01 replaced with alpha
  + This alpha can be learned. We can backpropagate into it
  + These neurons can choose what slope to have in their negative region.

**Slide 5-45**

* Exponential Linear Units (ELU)
* Tries to retain all the benefits of ReLU but it tries to get rid of downside of being nonzero centered
* So new there is still controversy

**Slide 5-46**

* Maxout “Neuron”
  + Goodfellow et al., 2013
* Very different form of a neuron
* It changes how a neuron computes
* Two weights and it computes the max of two functions
* Does not have some of the downsides of a RELU
  + It does not die
  + Still piecewise linear
  + Still efficient
* However, every neuron has two weights and we have now doubled the number of neurons

**Slide 5-47**

* USE RELUs!!!
* DON’T USE SIGMOID

**Slide 5-49**

Data Preprocessing

* Very common to zero center data
* Not as common to normalized data wrt to image data since all image data is pixel information
* Normalizing data very important with data sets of different scales.

**Slide 5-50**

* We don’t end up using these with images even though they are common in machine learning

**Slide 5-51**

* What is common with images is mean centering
  + Mean centering
    - For every single pixel we compute its mean value over the training set and we subtract it out
  + Per channel mean
    - Red, Blue, and Green channel mean centering

**Slide 5-53: Weight Initialization**

* Q: what happens when W = 0 init is used?
* A: All neurons output the same thing and in backprop they will behave the same way
* There is no symmetry breaking

**Slide 5-54**

* Instead ppl use small random numbers
* Ppl sample from a unit Gaussian with zero mean and 1e-2 standard deviation

**Slide 5-55**

* This works OK with small networks
* This breaks when we have very deep networks

**Slide 5-56**

* Sampling data set of 1000 data points that are 500 dimensional
* Creating hidden layers and nonlinearities (10 layers of 500 units)
* Using Tanh
* Taking unit Gaussian data and forward propagating it through the network
* Initialization strategy is to take this random Gaussian data and scale it by 0.01
* We want to look at the statistics of the neuron activations throughout the network with this initialization (mean, and std dev, and histograms)

**Slide 5-57**

* As we get to the 10th hidden layer we seen that the std dev goes down to zero
* What ends up happening for this initialization of this 10 layer network is all the tanh neurons end up outputting zero or very small numbers

**Slide 5-58**

* Q: think about the backward pass. What do the gradients look like?
* A: We would expect the gradients to be very small
* Almost no gradient would be accumulated
* As we go through the layers of the network we will muliptly by w continuously which is a very small number

**Slide 5-59**

* If we tried 1.0 instead of 0.01 we get the complete opposite effect
* Almost all neurons completely saturated, either -1 and 1.
* Therefore gradients will be zero as well since nothing is back propagating
* **Therefore initialization is very tricky to set**

**Slide 5-60**

* There was a proposal for the “Xavier Initialization” from Glorot et al., 2010
* They proposed an initialization strategy.
  + Divide the random weights with the square root of the number of inputs for every single neuron
* If we have lots of inputs we have lower weights
* If we have only a few inputs we have larger weights.
* For a single linear neuron (no activation functions) If we are getting Unit Gaussian data as input and we would like this linear neuron to have a variance of 1 then we should initialize the weights like this.
* This works in the case of a tanh

**Slide 5-61**

* Javier Initialization does not talk about nonlinearities
* This does not really work for ReLU neurons

**Slide 5-62**

* Extra factor of 2 must be added to make the Javier Initialization work

**Slide 5-63**

* Comparison with factor of 2 and without factor of 2
* He et al., 2015
* **It is recommended to use this initialization with ReLU neurons**

**Slide 5-64**

* **Proper initialization is an active area of research**

**Slide 5-65: Batch Normalization**

* We want roughly unit Gaussian activations at every part of our network. Therefore we just do that.
* We insert Batch Normalization layers into the network in a Mini Batch case
* These layers take input x and makes sure that in every single feature dimension across the batch, we have unit Gaussian activations

**Slide 5-66**

* N things in mini batch
* D features or D activations of neurons
* Matrix X of activations
* Batch normalization evaluates the empirical mean and variance along every single feature and then divides by it
* Whatever X was batch normalization makes sure every single column is a unit Gaussian
* This is completely differentiable

**Slide 5-67**

* We insert the Batch Normalization layers after fully connected layers in this case or after convolutional layers in CNNs
* They make sure everything is roughly unit Gaussian at every single step of the network
* One small problem with this is we may not want the inputs to the tanh layers to be unit Gaussian

**Slide 5-68**

* Fix to Batch Normalization
* Not only do we normalize but we then scale by gamma and shift beta
* The network can choose to adjust these parameters
* We will not have the trouble of the network dying or exploding in the beginning of optimization
* This will train right away and back propagation will take over and fine tune over time
* Network has the capacity to undo batch normalization. It can learn to be an identify function through back propagation

**Slide 5-69**

* Several Nice Properties to Batch Norm
  + Improves gradient flow through the netweork
  + Allows for higher learning rates
  + Reduces strong dependence on initialization
  + Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

**Slide 5-70**

* **Recommended to use Batch Normalization**

**Slide 5-73**

* CIFAR-10 Data and 2 layer neural network with 50 hidden neurons
* How does one train a neural network in practice?

**Slide 5-74**

* What do we check to makes sure things are working?
* 2 layer neural network
* Weights and biases initialized using Naïve Gaussian
* Function used to train network and output loss and gradient (implementation not shown)
* We try and disable regularization
  + We try to make sure our loss comes out correct
  + We are expecting a loss of 2.3 from previous slides using a softmax classifier

**Slide 5-75**

* We then turn regularization back on and check to see that the loss went up

**Slide 5-76**

* We then try to do a good sanity check
* Take small piece of data and try to overfit the training data
* We try to get a loss of very near zero
* If we can’t overfit with a very small piece of data then something is clearly wrong.

**Slide 5-77**

* We make sure that our cost can go down to zero and we are getting an accuracy of 100%
* If you are not able to overfit then the implementation is wrong
* We should not be scaling up the implementation until we can pass the full sanity check

**Slide 5-79**

* So we have passed the sanity check and we are scaling up to a bigger data set
* We are trying to find the learning rate that works
* So we start with a very small learning rate

**Slide 5-80**

* The Loss is barely decreasing therefore we know that the learning rate is too small

**Slide 5-82**

* Lets try a very very high learning rate
* We get weird errors

**Slide 5-84**

* We try to narrow into a rough region which allows our cost to go down

**Slide 5-86: Hyper Optimization**

* We try to find the best hyper parameters for our network
* Course to fine strategy
* At first we have a rough idea by playing with it of where the best learning rate should be
* We then a course search over learning rates
* We then narrow in on regions that work best

**Slide 5-89**

* This can be worrying since the learning rate is at the very boundary what I am evaluating
* Therefore this could mean there are better results outside the boundary being evaluated.

**Slide 5-90**

* Sometimes ppl will search for parameters in a grid search instead of randomly
* This is not suggested
* ALWAYS USE RANDOM

**Slide 5-91**

* There are some Hyperparameters you want to play with. Some of the most common ones are:
  + Learning rate
  + Update type
  + Regularization

**Slide 5-93**

* Loss functions can take various different forms and you need to get good at what they mean

**Slide 5-94**

* Sometimes we can get learning rates that are just weird

**Slide 5-95**

* In this case the prime suspect would be initializations

**Slide 5-99**

* During training, another thing to look at is accuracies
* Sometimes we prefer to look at accuracies over loss functions since accuracies are interpretable
* We know what accuracies mean
* In this case we are seeing that our training data accuracy is getting much better but our validation accuracy is not improving
* One possibility for this is we are overfitting and we might want to try to regularize more strongly

**Slide 5-100**

* We also might want to try tracking the scale of our parameters and the scale of our updates to those parameters
* Suppose our weights are on the order of unit Gaussian then intuitively the updates that we are updating our weights in back propagation we don’t want the updates to be larger than the weights themselves or tiny
* Look at the update and look at its norm (sum of squares) and compare it to the rough scale of our parameters
* This should be around 1e-3
* If this is too high then we might want to increase our learning rate

**Slide 5-101: OVERALL RECOMMENDATIONS**

* **Use RELUs**
* **Subtract means**
* **Use Xavier init**
* **Use Batch Normalization**
* **Hyper Optimization: sample hyper params and do in log space where appropriate.**