**Slide 3-4: Recall from last time…**

- We can’t explicitly hard code these so we have to train them from data.

**Slide 3-5: Recall from last time…**

- Parametric Approach

- We are writing a function f from image directly to raw 10 scores if we have 10 classes.

- We can interpret these classifiers as matching templates or being in a high dimensional space.

**Slide 3-6: Recall from last time…**

- Suppose we have a training data set of these three images.

- Suppose we have these 10 classes from CFAR-10

- Our function f is assigning scores for every one of these images

- With some particular setting of weights (assigned randomly here) we get some scores out.

Cat image: Not well classified since some scores higher and some scores lower.

Car image: Very well classified. Class for car scored higher than the rest

Frog image: Not very well classified. Frog class scored very low.

- We get this notion that different weights yield better score than others

- We are therefore trying to find the weights that give the best scores for all the ground truth lablels.

- We must quantify this notion that some weights are better than others with a "Loss Function"

**Slide 3-7**

- Suppose, for simplicity, we only have three classes

**Slide 3-8: Multiclass SVM loss:**

- Binary Support Vector Machine.

- "s" is a vector of class scores

- What SVM Loss is saying is

+ its summing across all the incorrect examples, across all the incorrect classes.

+ its comparing the score that the correct class recieved and the score that the incorrect class recieved

+ j: incorrect label

+ yi: correct label

+ 1 is our safety margin

**Slide 3-13**

sj - syi would equal 0 and we would therefore just be inflating the loss by a constant of 1

**Slide 3-14**

This would have no effect on the final optimization of W's. Here we would merely be scaling all the losses by a constant amount 1/N

**Slide 3-15**

We would end up with a different loss.

**Slide 3-16**

Smallest loss = 0

Largest loss = Infinite

**Slide 3-17**

If the scores are ~= 0 at initialization we would expect the first loss to result in "2" for this example.

Very helpful for implementation

**Slide 3-19**

Loss function in its complete form

**Slide 3-21: There is a bug with the loss**

- Can we get a W that would be different but would achieve a 0 loss?

- We can scale this by some constant alpha which is larger than 1.

- We would merely be making the score differences larger and larger linearly.

- This is not a very desirable property because we would end up with an entire subspace of W which would be optimal but they are in fact completely the same.

**Slide 3-22**

-Scaling would be larger than one since we have the added 1 margin

- Scaling would not apply to the bias part but just to the weights

**Slide 3-23: Weight Regularization**

We would like to have a preference of some W's over others.

- Regularization measures the "niceness" of our W.

- Regularization is a way of trading off our training loss and our generalization loss on a test set.

- Regularization is a set of techniques where we are adding objectives to the loss.

- Adding regularization techniques makes the test set performance better even if it makes the training error worse( we don’t correctly classify all examples)

- Most common form of regularization is L2-reg or weight decay

Suppose W is 2D matrix

We basically take all the elements and square them

**Slide 3-24**

* X: input vector
* Suppose we have an example where we are in 4d space
* Two candidate weight matrices w1 and w2
* Dot product of transpose(2)\*X would be the same
* Regularization cost would favor w2. W2 is better because:
  + It takes into account the most things of our x vector.
  + Regularization wants to spread out the W’s so that they take into account all the input features.
  + Takes into account more of our x-input vector
  + This works very well in practice.
  + Regularization is a good idea

**Slide 3-25: Softmax Classifier (Multinominal Logistic Regression)**

* Different functional form for how loss is specified on these scores
* There is an interpretation that the softmax classifier puts on top of these scores.

**Slide 3-26**

* We interpret these scores as unnormalized log probabilities that are assigned to these classes
* The unnormalized log probability of Y given the image.

**Slide 3-27**

* In other words, we are assuming that if the scores are unnormalized log probabilities then the way to get probabilities of the different classes is that we take the scores, exponentiate them to get the unnormalized probabilities and we then normalized them to get the normalized probabilities.

**Slide 3-29**

* We want to maximize the log likelihood of the true class and we went to minimize the negative log likelihood of the correct class.
* We want the log likelihood of the correct class to be high where the log likelihood is the softmax function of the scores.

**Slide 3-35**

What is the min/max possible loss?

Smallest loss = 0 (if correct class has probability 1, then –log(1) = 0)

Largest Loss = Infinite (if correct class has probability ~0, then –log(~0) = infinity)

Same bounds of SVM

**Slide 3-36**

-log(1/Number of classes)

This is very helpful to know what the lower bounds are. We know we would never expect to get negative numbers so if we do get them then we know something is wrong.

**Slide 3-38**

* Assume we have three examples and three classes and scores for these three examples.
* For the third data point for instance [10,-100,-100], if we were to grab 10 and vary it by a small amount what happens to the loss for both SVM and Softmax?
* SVM will not care since margins have been met already.
* SVM doesn’t express a preference over the examples. SVM has an added robustness
* Softmax will express preference and we would get a better or worse loss.

**Slide 3-42**

* We have this “f” function and we have two loss formulations
* Full Loss is achieved as mean loss plus regularization.
* Diagram
  + Data x and y (images and labels)
  + Regularization loss is a function of the weights not the data
  + We don’t have control over the data but we do have control over the W.
  + As we change the W, we change the loss
  + If we have the W we can compute the loss
  + Loss is linked to how well we are classifying each of our examples
  + Low loss means we are classifying very well on the training data.

**Slide 3-46**

* Random search is equivalent to being blind folded and trying to find a valley.
* It is not a very good idea.
* Therefore we want to get the gradient or slope in every direction from where we are so that we can go downhill

**Slide 3-47**

* We can numerically evaluate this expression at some W.

**Slide 3-48**

* Suppose we have some current W and we have some loss.
* We want to get an idea about the slope.

**Slide 3-49**

* Derivative equation corresponds to me taking a small step in some direction and check if I went up or down.

**Slide 3-50**

* We can use the derivative formula with finite difference approximation ( small h) to derive the gradient or slope.
* - 2.5 means we are sloping downwards
* We can do this in the other dimensions.

**Slide 3-55**

* Problem with this is that I a CNN we have hundreds of millions of parameters. We can’t numerically check the loss before we take a single step

**Slide 3-58**

* The numerical evaluation is pointless since we have a calculus gradient expression for this

**Slide 3-62**

* In practice we always use the analytic gradient, do the calculus, to figure out what the gradient should be.
* But we do the implementation with a numerical gradient check.

**Slide 3-63**

* Knowing the gradient we can then perform a parameter update.
* Step size/Learning rate is the most critical parameter.
* Learning rate and regularization parameter cause the most headache and we cross validate over these

**Slide 3-65**

* In practice we have the option where we don’t evaluate the loss over the entire training data set but we use small batches of it.
* Mini-batch Gradient Descent
* If we only sample very few samples from our training set then the estimate of the gradient over the entire training set is noisy
* But this allows us to step more.
* Mini-batch works much better and it’s impractical to use Full batch gradient descent.
* Mini-batch sizes are dictated by the memory

**Slide 3-66**

* Loss goes down over time
* Full batch gradient descent would not have as much noise, it would be just a line
* Mini-batch GD has noise since some batches are better than others

**Slide 3-67**

* Effects of learning rate on cost function
* Very high learning rates cause you to thrash around in the W space so either you never converge or you explode
* Very low learning rate you are barely doing any updates at all so it takes a long time to converge
* High Learning rate you can get stuck in a bad position of loss. You could zip over the local minima since the learning rate is high.
* Many ppl start off with a high learning rate and they decay it over time.

**Slide 3-70**

* At this point in the class we know how to do Linear Classifiers
* However we don’t want to apply a Linear Classifier to pixel data
* Historical perspective of image recognition:

Ppl would therefore compute all the feature types of an image and we would get statistical summaries of what the image looked like. All these would be concatenated into large vectors which would then be piped to Linear Classifiers

**Slide 3-71**

* An example of a feature type is a Color (Hue) Histogram
* We go over all the pixels in the image and we bin them into bins for each color depending on the hue of the color.

**Slide 3-72**

* Another example is HOG/SIFT features
* Go over a small neighborhood in the image and look at edges of different orientations in the image.
* We end up with a summary of what kinds of edges there are in the image and where

**Slide 3-75**

* What it therefore looked like in computer vision before 2012 was we would take an image and perform a Feature Extraction step
  + This step would decide what the most interesting features of the image are.
  + We would end up with a feature vector
  + We would then apply a Linear Classifier to this feature vector.
* Since that time it was found that what works much better is:
  + You start with the raw image and are not designing parts in isolation
  + We come up with an architecture that can simulate a lot of these different features so to speak
  + Since this whole thing is a single function we can train all of it all the way down to the pixels. This would allow us to train our feature extractors effectively
  + We try to eliminate hand engineered components