Recognition IV
Deep Networks

Computer Vision I
CSE252A
Lecture 18

Announcements

• HW4 due Friday 12/7.
• Final Exam: Friday 12/14 at 7pm-10pm
• Final Exam review session (Q&A): TBD
Final Exam

- Closed book
- One cheat sheet
  - Single piece of paper any size, handwritten, no photocopying, no physical cut & paste, both sides OK.
- What to study
  - Basically material presented in class, and supporting material from readings
  - If it was in text, but NEVER mentioned in class, it is very unlikely to be on the exam
- Question style:
  - Short answer
  - Some longer problems to be worked out.
- When in doubt write something. There will be partial credit.

Perceptron

Note: For \( x=(x_1, \ldots, x_n) \), \( x_i \) can be binary or a real number.

\[
\begin{align*}
O(x) &= \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ 0 & \text{otherwise.} \end{cases} \\
o(x_1, \ldots, x_n) &= \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ 0 & \text{otherwise.} \end{cases}
\end{align*}
\]
Two Layer Network

- Fully connected network
- Nodes are nonlinear function of weighted sum inputs:
  \[ f(x; w) = S(w^T x + w_0) \]

The nodes of multilayered network

\[ f(x; w) = S(w^T x + w_0) \]

Nonlinearities: \( S(x) \)
- Threshold (perceptron)
- Sigmoid
- Rectified Linear Unit (ReLU)
Feedforward Networks

- Let \( y = f^*(x) \) be some function we are trying to approximate.
- This function could be assignment of an input to a category as in a classifier.
- This function could be one or more real numbers (regression).
- Let a feedforward network approximate this mapping \( y = f(x; w) \) by learning parameters \( w \).

Feedforward Networks

- These networks are composed of functions represented as "layers".

\[
f(x) = l^3(l^2(l^1(x; w_1); w_2); w_3)
\]

with weights \( w_i \) associated with layer \( i \).
- \( f(x) \) can be a scalar function or a vector function.
Classification Networks

- To classify the input $x$ into one of $c$ classes, we have $c$ outputs.

- Output $i$ can be viewed as $p(\omega_i | x)$. That is the posterior probability of the class, given the input. Recognition decision is $\arg\max p(\omega_i | x)$.

- If the network were certain about the class, one output would be 1 and the rest would be zero.

- More generally, $\sum_i p(\omega_i | x) = 1$, the $c$ outputs must sum to 1.

- This can be implemented with a softmax layer:

\[
O_i = \frac{e^{z_i}}{\sum_{j=1}^c e^{z_j}}
\]

High level view of evaluation and training

- Training data: $\{(x_i, y_i) : 1 \leq i \leq n\}$

- Training: Find $w$ that minimizes the total loss.

- Total Loss: $\sum_{i=1}^n L(f(x_i, w), y_i)$

- Training: Find $w$ that minimizes the total loss.
The loss function

- The loss function is really important. It determines how we compare what the network produces to our labels.

- Common ones:
  - Regression problems:
    - Distance: \( L(y, \hat{y}) = \|y - \hat{y}\|^p \), usually \( p = 1 \) or \( 2 \)
  - Classification
    - Softmax: \( O_i = \frac{e^{z_i}}{\sum_{j=1}^{c} e^{z_j}} \)
    - Cross entropy (See homework)

Optimization for Deep Nets

- Deep learning optimization is usually expressed as a loss summed over all the training samples.

- Our goal is not so much find the weights that globally minimize the loss but rather to find parameters that produce a network with the desired behavior.

- Note that there are LOTS of solutions to which our optimization could converge to—with very different values for the weights—but each producing a model with very similar behavior on our sample data.
Optimization for Deep Nets

- Although there is a large literature on global optimization, gradient descent-based methods are used in practice.

- Our optimizations for deep learning are typically done in very high dimensional spaces, where the number of weights can run into the millions.

- And for these optimizations, when starting the training from scratch (i.e., some random initialization of the weights), we will need lots of labeled training data.

Back propagation

- Basically another name for gradient descent

- Because of the nature of the network $l_3(l_2(l_1(x;w_1);w_2);w_3)$, gradients with respect to $w_i$ are determined by the chain rule.

- Can be thought of as “propagating” from loss function to input.

- In homework, you'll use adaptive step size method called ADAM.
Training and Validation Sets

Labeled Data

Training Data  Validation Data

NEVER TRAIN ON YOUR VALIDATION SET!
Training and Validation Sets

Labeled Data → Training Data → Validation Data

NEVER TRAIN ON YOUR VALIDATION SET!
Training and Validation Sets

NEVER TRAIN ON YOUR VALIDATION SET!

Regularization

The goal of *regularization* is to prevent *overfitting* the training data with the hope that this improves *generalization*, *i.e.*, the ability to correctly handle data that the network has not trained on.
Regularization for MLP

\[ L = -z_i + \log \sum_j e^{z_j} + \frac{\lambda}{2} (\|W\|^2 + \|W_o\|^2) \]

- For every training batch through the network, dropout 0.5 of hidden units and 0.2 of input units. You can choose the probabilities as you like...
- Train as you normally would using SGD, but each time you impose a random dropout that essentially trains for that batch on a random sub-network.
- When you are done training, you use for your model the complete network with all its learned weights, except multiply the weight by the probability of including its parent unit.
- This is called the weight scaling inference rule. [Hinton et al., 2012]

Dropout [Srivastava et al., 2014]
Early Stopping

• Typical deep neural networks have millions and millions of weights!

• With so many parameters at their disposal how to we prevent them from overfitting?

• Clearly we can use some of the other regularization techniques that have been mentioned…

• …but given enough training time, our network will eventually start to overfit the data.
Deterministic vs. Stochastic Methods

- If we performed our gradient descent optimization using all the training samples to compute each step in our parameter updates, then our optimization would be deterministic.

- Confusingly, deterministic gradient descent algorithms are sometimes referred to as batch algorithms.

- In contrast, when we use a subset of randomly selected training samples to compute each update, we call this stochastic gradient descent and refer to the subset of samples as a mini-batch.

- And even more confusingly, we often call this mini-batch the “batch” and refer to its size as the “batch size.”

Stochastic Gradient Descent

The SGD algorithm could not be any simpler:

1. Choose a learning rate schedule $\eta_t$.
2. Choose stopping criterion.
3. Choose batch size $m$.
4. Randomly select mini-batch $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$
5. Forward and backpropagation
6. Update $\theta_{t+1} = \theta_t - \eta_t g$ \quad $g = \frac{1}{m} \sum_{i=1}^{m} \nabla \theta L(x^{(i)}, y^i)$
7. Repeat 4, 5, 6 until the stopping criterion is satisfied.
Finally, we get to images…

What if we just vectorized images and stuffed these into a MLP?

MLP  ?
Too many weights and connections!

- This fully connected hidden layer might have 75 million weights!
- And this is just for a thumbnail image and a two layer net.

Convolutional Neural Networks
Fully Connected (FC) Layer

Every input unit is connected to every output unit.

Consider a hidden unit: it connects to all units from the previous layer.
Convolutional Layer: Local Connections

Here the connections are spatially local and governed by the kernel size.
Convolutional Layer: Local Connections

Here the connections are spatially local and governed by the kernel size.

Convolutional Layer: Shared Weights

The weights are shared. They are the same for each position. So, this is like a convolution kernel.
Convolution in 2D

Convolution with 2D Kernel
Convolution with 2D Kernel

\[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1
\end{bmatrix}
\ast
\begin{pmatrix}
\ast
\end{pmatrix}
\Rightarrow
\text{Image}
\]
We can skip input pixels by choosing a \textit{stride} > 1.
We can skip input pixels by choosing a \textit{stride} > 1.

The output dim = (input dim - kernel size) / stride + 1.
Convolutional Layer: Padding + Stride

Output dimension = (input dim - kernel size + 2 * padding) / stride + 1

ReLU used with ConvNets

- Just like with our fully connected layers, for our convolutional layers we will follow the linear operation (convolution) with with a non-linear squashing function.
- Again the fcn to use for now is ReLU.
- But we are not done…there’s one more thing!
Pooling

- We can spatially pool the output from the ReLU to reduce the size of subsequent layers in our network.
- This reduces both computation and the number of parameters that need to be fit and helps prevent overfitting.
- The pooling operation is often the max value in the region, but it could be average, or median, etc.
- The pooling has a stride associated with it that determines the downsampling of the input.

Pooling Layer

The pooling layer pools values in regions of the conv layer.
Oops. Just one more thing…Recall

Convolutional Layer: Shared Weights

The weights for the kernel are shared. They are the same for each position.
Each kernel finds just one type of feature.

If a kernel shares weights then it can only extract one type of feature.

Multiple input channels

- Convolution kernel is 3-D: Goes across image dimensions & across channels
- Size = width x height x # input channels
Many kernels yielding many features!

- Convolution kernel is 4-D: For each output channel, kernel goes across input dimensions and channels
- Size = width x height x # input channels x # output channels

An example deep convolution network

- Input: 28x28 grayscale image
- Output: 10 classes. One output per class.
A Convolutional Net

- Let’s assume we have 28x28 grayscale images as input to our conv net. So we will input 28x28x1 samples into the net.
- Let’s fix our kernel size at 5x5 and, to make this simple, pad our images with zeros and use a stride = 1.
- Let’s use max pooling on the output, with a 2x2 pooling region and a stride of 2.
- Let’s extract 32 features after the first layer.
- So the output from this layer will be 14x14x32.
A Convolutional Net

- Now let’s make a second layer, also convolutional.
- Let’s fix our kernel size at 5x5, pad our images with zeros and use a stride = 1.
- Let’s use max pooling on the output again, with a 2x2 pooling region and a stride of 2.
- Let’s extract 64 features after the second layer.
- So the output from this layer will be 7x7x64.
A Convolutional Net

- Our third layer will be a fully connected layer mapping our convolutional features to a 1024 dimensional feature space.

- This layer is just like any of the hidden layers you’ve seen before. It is a linear transformation followed by ReLU.

- So the output from this layer will be 1x1x1024.
A Convolutional Net

- Finally, will map this feature space to a 10 class output space and use a softmax with a MLE/cross entropy loss function.
- And...we're done!

Parameters = \((5x5x1x32+32) + (5x5x32x64+64) + (7x7x64x1024+1024) + (1024x10+10)\)
Some Famous Deep Nets and Data sets

LeNet 5

AlexNet [Krizhevsky et al., 2012]

VGG16 [Simonyan and Zisserman, 2014]
GoogLeNet [Szegedy et al., 2014]

Another view of GoogLeNet’s architecture.

ResNet [He et al., 2016]
PASCAL VOC

20 categories in 11,530 images with 27,450 ROIs and 6,929 segmentations

[Everingham et al. 2005—2012]
Object Detection: PASCAL VOC mean Average Precision (mAP)

![ImageNet](https://example.com/image.png)

20,000+ categories x ~1000 instances = 14,000,000+ images

[Deng et al. 2009]
Classification: ImageNet Challenge top-5 error

Figure source: Kaiming He