682 Fall 23, Midterm Review

November 6, 2023

1 Overview

For the midterm, you should cover all of the following:

- The items on this review sheet.
- All of the reading assignments on the course website that are marked as "NOTES" on the syllabus. You are not responsible for the material in other documents unless specifically stated below.
- All of the material in Homework 1 and 2. Notice that many topics from Homework 3 are also covered, but they will be explicitly listed below if you are responsible for them for the midterm.

2 Specific material

The exam covers topics till "Lecture 15 - Adversarial examples and style transfer." In particular, here are the topics you should understand.

- The difference between fine-tuning, pre-training, and "training from scratch".
- Why do you need to have a hold-out validation set?
- Know the formal definition of supervised learning. This involves having a training set of examples (x_i, y_i) drawn from a specific distribution, where x_i's are the data vectors and the y_i's are the class labels. Then you are given a test set of x_i's and asked to estimate the class label.
- The definition of parametric models and non-parametric models. Give an example for each that is covered in the lectures. The name "non-parametric" model is a poor name, since non-parametric models are usually models in which the number of parameters grows with the number of data points, as in K-nearest neighbors. The key idea is that a non-parametric model is a model that does not have a fixed number of parameters. A parametric model is a model with a fixed number of parameters.
- What are the advantages and disadvantages of learned representations compared with hand-crafted representations (SIFT, HOG, etc.)?
- Differences between regular neural networks and convolutional neural networks. Main justifications for CNNs over regular neural networks. These include
  - the idea that many useful features will be local in the sense that they will only be functions of small areas of the input. While in principle, a standard neural net can learn local features, learning can be faster and more efficient by forcing features to be local. It also dramatically reduces the number of parameters to be learned, which leads to effective training with smaller training sets.
  - the idea that if a feature is useful at one position in an image, it is likely to be useful at other positions in the image. This leads to the idea of having many, many copies of the same feature spread over the image, which can be implemented with convolution. This idea has several advantages. First of all, a feature can be learned by combining data from many different parts of

the image through parameter sharing. Second, we can learn a feature for a particular part of the image even if we never saw the feature in that location during training. That means we can get away with much less training data.

- What is the difference between gradient descent, stochastic gradient descent (SGD), and mini-batch SGD.
- Know the update rules of SGD, SGD+momentum
- Different types of non-linearities (ReLU (rectified linear unit), leaky ReLU, tanh, logistic, maxout neuron), and how to make the choice in practice. Understand the advantages and disadvantages of each type of non-linearity. Also, make sure you understand why we need non-linearities in the first place. For example, a network built out of only linear layers can only compute linear functions. This is clear if you write the whole function of the network down. A whole sequence of matrix multiplies is equivalent to another matrix multiple, which means the whole network is linear. Note that a linear network can only compute linear classification boundaries. Thus, you could never separate classes if you can't draw a linear boundary between them.
- What is a "dead ReLU"? How can you tell if your model is suffering from the problem, and how can you deal with it? How can a dead ReLU come back to life?
- Loss functions (negative log loss, multi-class SVM loss, l2 loss, etc.), and how to make the choice in practice.
- Make up your own loss. Can you name an appropriate application and say something good or bad about your loss for it?
- Know what data loss, regularization loss, and total loss are.
- The motivation behind batch normalization, and why it solves the problem. Know the procedure for training time: estimate mini-batch mean and standard deviation, subtract off mean estimate and divide by standard deviation estimate.
- How can a convolutional layer be implemented as a fully connected layer? How can a fully connected layer be implemented as a convolutional layer?
- What is the purpose of regularization? What's the difference between L2 and L1 regularization? Can you come up with another type of regularization?
- Understand how to address the following situations during training: training loss is equal to validation loss (underfitting); training loss is much much lower than validation loss (overfitting); training loss won't go down (bad initialization or learning rate too low); training loss goes up (learning rate too high).
- What the problem of disappearing gradients is, and how to deal with it.
- Can two neural networks that have different arrangements of weights produce the exact same function? Explain how to take a neural network and rearrange the weights to make an equivalent neural network whose weight matrices are different. *Here is how to do this. Consider the first hidden layer of a standard neural network, like the kind in assignment 1 used to do CIFAR classification. Recall that the weights connected to each hidden unit can be shown as an "image", to visualize what the weights are doing. Let's call each of these sets of hidden weights a "filter". Furthermore, call the first filter in our first layer A and the second filter B. Now imagine another neural network in which we have swapped the position (in the weight matrix W) of filters A and B. That is, we have made filter A the second filter and filter B the first filter. Now, this new network is computing the same set of functions at the first layer as the original network; it is just that they are in a different order. To make sure that the two networks are the same, we would then have to swap the next layers weights corresponding to the two filters that were swapped. By shuffling the filters in any neural network, and making an equivalent change to the higher level weights that use the outputs of these filters, we can produce a large number of networks that compute exactly the same function of the input.*

- Be able to do show that the derivative of the logistic is $f(x)(1 - f(x))$. Why is this a useful thing to do? Because we already computed $f(x)$ during the forward pass, so it is very cheap to compute the backward pass.
- Backpropagation. Be able to do the simple examples from class with pencil and paper.
- How to handle branches in backprop. Note that you should be able to handle branches in both directions, where one value is copied many times to produce inputs to many other functions, or where many values are put through a single function to produce a single output.
- Understand how to justify efficient implementations of backprop. For example, if the Jacobian is N×N, how can I get away with only storing N of its values sometimes? A good example is the derivative of the ReLU function. Since each output of the ReLU is only a function of a single input, all of the "cross derivatives" (element i of input with respect to element j of output) are all 0. Thus, there is no need to store them.
- Understand the how numerical gradients are computed, including one sided differences $(f(x + h) - f(x))/h$, two-sided differences $(f(x-h)-f(x+h))/2h$, and higher-sided differences (see lecture slides).
- Understand the difference between numerical gradients, symbolic gradients, and backpropagation.
- Why could the accuracy on the training set go up significantly when there is a very small relative change in the loss function of a network, especially right after initialization? Answer: when training begins, most of the output values may have nearly equal probability. Thus, a very small increase of the probability of the correct class may render the probability of the correct class higher than all of the others. Thus, the accuracy would go up despite small changes in the probabilities.
- Suppose I initialize a neural network with small random Gaussian weights. If I have 100 classes, what do I expect the data loss to be after the first forward pass (before the weights have been adapted at all), and why?
- Definition of over-fitting. Why it is a problem, how to tell if you have the issue of over-fitting, and how to deal with it.
- Name one problem with grid search for exploring hyperparameter settings. Name one advantage relative to random sampling of hyperparameter settings.
- How can over-fitting be used for debugging in practice? Answer: you can test whether your network can find a way to make the training error zero for a small data set and 0 regularization. If your network can't do this, there may be a problem with the backprop.
- Be able to take the derivative of a scalar/vector/matrix w.r.t. a scalar/vector/matrix. What are the shapes of the resulting arrays?
- The difference between model parameters and hyper-parameters.
- Strategies for searching optimal hyper-parameters: random search, grid search, coarse to fine search.
- What is a possible cause if training loss explodes?
- What is dropout, and how is it related to the concept of model ensembles? Answer: dropout can be thought of as training a separate network for each separate forward pass. At test time, this can be thought of as taking the average of a very large number of different networks, which is a massive ensemble. Also, you should understand why you have to multiply by "p", the probability of dropout, at test time, to correct for the fact that no nodes are dropped out.
- What is the problem with randomly initializing all layers with the same Gaussian, even if those layers are different sizes? How do Xavier initialization and improved Xavier initialization (He et al. 2015) solve the problem? You should know how to implement them.

- Understand the basic principles behind convnets and how they are implemented. This includes the convolution and pooling operations, the impact they have on learning, and the differences between conv and fully connected layers.
- Have a high-level understanding of various convnet architectures such as LeNet, AlexNet, and ResNet. We don't expect you to remember the exact number of layers and their types, but you should understand the basic progression between the models.
- Understand how convnets can be used for spatial localization tasks such as object detection, instance segmentation, and semantic segmentation. In particular:
  - Understand the differences between various spatial localization tasks. What is the expected output from the models, and what sorts of labels do you need to train these models in a fully supervised manner?
  - For object detection, understand how detection can be cast as a repeated region classification task. Understand the differences between R-CNN, Fast R-CNN, and Faster R-CNN.
  - For instance segmentation, understand the basic task and show how the detection architectures can be augmented to produce instance segmentations using a separate branch that produces region masks.
  - For semantic segmentation, understand how classification architectures can be modified to produce multiple pixel labels (e.g., using a fully convolutional architecture with upsampling layers). Give examples of various image labeling tasks such as labeling categories of each pixel, converting a grayscale image to color, or producing depth and normal maps from color images.
- Techniques to visualize and understand convnets. These include:
  - Visualizing the weights directly (which is only meaningful for the first layer).
  - Visualizing patches that maximally activate neurons.
  - Visualizing the representation space using dimensionality reduction techniques such as t-SNE or PCA.
  - Visualizing how the predictions of the model change by occluding different parts of the image.
  - Visualizing using the backward pass (deconvolution approaches).
  - Visualizing by optimizing over the input image (inversion approaches).
- Understand what image properties are preserved across layers of a convnet. For example, most of the spatial and color information of the input image is retained in the first layer, but higher-layer representations preserve the most important elements of the image for the classification task while the spatial and color information becomes less precise. This can be seen in the nature of the "inverted images" from different layer activations in a convnet.
- Understand how neural style transfer works. Show that the Gram matrices from different layers can capture multi-scale texture representations and how adding a "style loss" with the "context loss" can enable us to mimic the style and content of different images.
- Understand how one can "fool" convnets by posing it as an optimization problem to maximize any class score.