COMPSCI 682 (Spring 2025): Study guide

1 Overview

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

1. The items on this review sheet.
2. All of the reading assignments on the course website that are marked as NOTES on the syllabus, and the notes under the Notes tab. You are not responsible for the material in other documents unless specifically stated below.
3. Material from all of the homework problem sets.

2 Specific material

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? Why can't you simply evaluate the performance of your model on the test set over and over?
- Know the formal definition of supervised learning. This involves having a training set of examples (x_i, y_i) drawn from a specific distribution $P(x, y)$, where x_i are the data vectors and y_i are the class labels. Then you are given a test set of x_i values, drawn from the same distribution $P(x, y)$, and asked to guess the class label.
- What is overfitting, and why is it bad? How can you tell if your model is overfitted? What about underfitting?
- What are the common methods to prevent the model from overfitting and/or underfitting?
- When can overfitting be useful? (Answer: when you are trying to make sure that your model training is working, you can try overfitting on a small amount of data. Typically, one should be able to get very high accuracy on the training set when the training set size is small relative to the number of parameters in the model.)
- What is cross-validation? When would you consider using cross-validation, and what are the pros and cons compared to using a single validation set?
- For k-nearest neighbors (k-NN) classifier, what procedure would you use to decide on a good value for k?
- Compare a fully connected layer and a convolutional layer. Mention at least one advantage and one disadvantage of each.
- 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 in the first few hidden layers will be local in the sense that they will only be functions of small areas of the input. That is, the receptive fields of the units in the early layers will be relatively small. While, in principle, a standard neural network can learn local features, learning can be faster and more efficient by forcing features to be local. CNNs dramatically reduce 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 copies of the same feature spread over the image, which can be implemented with convolution.
  - This idea has several advantages: A feature can be learned by combining data from many different parts of the image through parameter sharing.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 are stride, padding, and receptive field?
- A large receptive field is important for many computer vision tasks such as segmentation. Name at least two methods how one can increase the receptive eld of a node in a neural network. (Answers: Add pooling layers before this layer. Add other standard convolution layers before this layer. Increase the stride of previous layers.
- A network's input image is of size $h \times w$. After the first layer, the feature map is of size $h_2 \times w_2$. Which operations can lead to this decreased feature map? (Mention at least two.)
- Given a 7×7 input image, a filter of size 3×3 is applied with a stride of 1 and padding of 1 pixel. What is the size of the output?
- What are the differences between gradient descent, stochastic gradient descent (SGD), and mini-batch SGD?
  - **Gradient descent**: A method where one computes the gradient with respect to the exact loss function, which is usually defined over the entire dataset.
  - **Stochastic gradient descent (SGD)**: A method in which there is some randomness in the computation of the gradient. This randomness often arises from using a random subset of the training data to approximate the full gradient.
  - **Mini-batch gradient descent**: A version of stochastic gradient descent in which one uses subsets of the full training set to approximate the full gradient.
- Know how to compute the numerical gradient of a function and understand why the two-sided method is preferred.
- When computing the numerical gradient of a function, what could go wrong if the chosen epsilon value is too large or too small?
  - **If epsilon is too small**, there could be a precision error (for example, in the division), making the gradient estimate very inaccurate.
  - **If epsilon is too big**, the second evaluation of the function may be so far from the current point that the approximation of local linearity breaks down, resulting in a poor approximation of the gradient at the original point.
- What are the common data pre-processing options on input data to neural networks? Know the motivation behind each of these operations.

- What is the purpose of having pooling layers in neural networks? Is max pooling a linear operation?
- Write down the equation of two commonly used activation functions and their derivatives with respect to their input arguments.
- Write down the negative log loss with its derivative. Explain the intuition behind the negative log loss. What is the penalty if we assign zero probability to the correct answer?
- What is the 0-1 loss (also called classification loss)? What's wrong with trying to train a neural network using the 0-1 loss or classi cation loss? Answer: The problem with the 0-1 loss is that its derivative is zero almost everywhere. When the derivatives are zero, then the parameters cannot be changes in a way that is useful, so learning cannot occur. The problem is NOT that it is nondifferentiable. Many functions that are not differentiable everywhere are still good loss functions.
- Know how to derive the SVM and Softmax gradients.
- What are vanishing gradients, and what are possible reasons for those?
- Different types of non-linearities (ReLU (recti ed linear unit), leaky ReLU, tanh, logistic), 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 the mini-batch mean and standard deviation, subtract off the mean estimate, and divide by the 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 (under-fitting); training loss is much much lower than validation loss (over-fitting); training loss wont 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 layer's 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 show that the derivative of the logistic is $f(x)(1 - f(x))$. Why is this a useful thing to do? Answer: Because we already computed $f(x)$ during the forward pass, so it is very cheap to compute the backward pass.
- Be able to do the simple examples of backpropagation from class with pencil and paper.
- How to handle branches in backpropagation. 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 backpropagation. For example, if the Jacobian is $N \times N$, 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.
- 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.
- 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.
- 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.
- Know how to implement inverted dropout, and why it is usually preferred.
- Be able to describe the sets of parameters of a simple recurrent neural network. These include a set of parameters (W_{xh}) that map the input to a term in the hidden representation, a set of parameters (W_{hh}) that map the previous hidden state to another term in the new hidden state, and (W_{hy}) which maps the current hidden state to a set of output scores.
- What are the purposes of classification, localization, and object detection? What are the differences between them?
- How can we train a neural network for the task of object detection? What do we need to prepare (images and ground-truth)? What are the losses (classification and regression)?
- Describe differences between RCNN, Fast-RCNN, Faster-RCNN. Which of the models are the fastest, and why?
- Describe an evaluation metric for object detection? Answer: First, define what it means to correctly detect an object. This is typically done using the intersection-over-union ratio, where the intersection between the guess and the ground truth boxes is divided by the area of the union of these two boxes. If this ratio is above a threshold, then it is deemed to be a correct detection. Otherwise, a guessed detection is a false positive. By drawing a precision-recall curve by varying the threshold one can calculated the area under the precision-recall curve as a final metric. The maximum value for this metric is 1.