Homework 5 Written Questions

CSCI 1430

Template Instructions

This document is a template with specific answer regions and a fixed number of pages. Given large class sizes and limited TA time, the template helps the course staff to grade efficiently and still focus on the content of your submissions. Please help us in this task:

- Make this document anonymous.
- Questions are in the orange boxes. Provide answers in the green boxes.
- Use the footer to check for correct page alignment.
- **Do NOT remove the answer box.**
- **Do NOT change the size of the answer box.**
- **Extra pages are not permitted unless otherwise specified.**
- **Template edits or page misalignment will lead to a 10 point deduction.**

Gradescope Submission

- Compile this document to a PDF and submit it to Gradescope.
- Pages will be automatically assigned to the right questions on Gradescope.

This Homework

- 5 questions **[8 + 6 + 6 + 10 + 12 = 42 points]**.
- 1 extra credit question **[10 points]** (e.g., for capstone).
- Include code, images, and equations where appropriate.

Q1 — [8 points]
Many traditional computer vision algorithms use convolutional filters to extract feature representations, e.g., in SIFT, to which we then often apply machine learning classification techniques. Convolutional neural networks also use filters within a machine learning algorithm.

Q1.1 — [4 points]

What is different about the construction of the filters in each of these approaches? **[3–4 sentences]**

Only Q1.1 should be on this page

Q1.2 — [4 points]

Please declare and explain at least two advantages and disadvantages of each of these two approaches (four total). **[6–8 sentences]**

Only Q1.2 should be on this page

Q2 — [6 points]
CNNs for classification might have a multi-layer perceptron (MLP) after the convolutional layers as a general purpose decision-making subnetwork. The MLP is constructed from 'dense' or 'fully-connected' layers. In these, every output perceptron is connected to every input. This contrasts with convolutional layers, where each kernel is a local window onto the previous layer activations that slides across it.

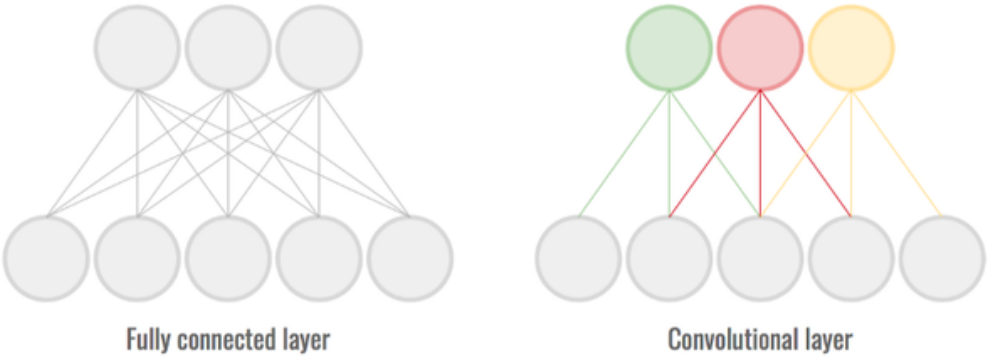

Figure 1: Right diagram: Each color shows a different slid position of the same kernel.

Suppose we imagine a 'locally-connected MLP', where every output perceptron is only connected to a small subset of nearby inputs but does not slide across them.

Describe three differences that might occur when using a *locally-connected* MLP versus a *fully-connected* MLP, and explain why. This might be in terms of the network's performance, its behavior on different data, the learned convolution kernel or perceptron weights, the optimization process, etc. **[6–9 sentences]**

Only Q2 should be on this page

Q3 — [6 points]
Given a neural network classifier and the stochastic gradient descent optimizer, discuss how the following hyperparameters might affect both the training process and outcome.

Q3.1 — [2 points]

Learning rate. **[2–3 sentences]**

Q3.2 — [2 points]

Batch size. **[2–3 sentences]**

Q3.3 — [2 points]

Number of epochs. **[2–3 sentences]**

Q4 — [10 points]
What effects are caused by adding a spatial max pooling layer (stride > 1) between two convolutional layers, where the output with max pooling is some size larger than $1 \times 1 \times d$?

There may be multiple correct answers per question.

Q4.1 — [1 point]

What happens to the computational cost of training? (Ignore the cost of the max pooling layer itself.)

- ☐ Increases
- ☐ Stays the same
- ☐ Decreases

Q4.2 — [1 point]

What happens to the computational cost of testing? (Ignore the cost of the max pooling layer itself.)

- ☐ Increases
- ☐ Stays the same
- ☐ Decreases

Q4.3 — [1 point]

What happens to the complexity of the decision boundary? *Note:* here we do not mean computational complexity.

- ☐ Increases
- ☐ Stays the same
- ☐ Decreases

Q4.4 — [1 point]

What happens to the potential for overfitting?

- ☐ Increases
- ☐ Stays the same
- ☐ Decreases

Q4.5 — [1 point]

What happens to the potential for underfitting?

- ☐ Increases
- ☐ Stays the same
- ☐ Decreases

In computer vision, we often care about producing invariant (or equivariant) representations. For instance, convolution is translation equivariant because a translation of the input induces a translation of the output. CNNs attempt to exploit this property to consider objects at any position in the image.

Q4.6 — [3 points]

Which of the following does max pooling provide?

Here, 'global' = the whole image, and 'local' = only within the pooling region.

- ☐ Provides local rotational invariance
- ☐ Provides global rotational invariance
- ☐ Provides local scale invariance
- ☐ Provides global scale invariance
- ☐ Provides local translational invariance
- ☐ Provides global translational invariance

Q4.7 — [2 points]

In a CNN, is the convolutional property of translation equivariance maintained through max pooling? Why or why not? **[4–6 sentences]**

Another way to think about it: Suppose I classify birds. If a bird translates within the image but is otherwise identical, will I still receive the same confidence of bird class?

Note: It might help to relate different pooling functions to downsampling, and to consider what we know about downsampling, image frequency, and aliasing.

Q5 — [6 points]
Stakeholders of automated decision-making systems often require explanations for the system's decisions. In some situations, there exists a right to explanation; the US Government's Blueprint for an AI Bill of Rights includes such a right. But, the benefits of a technology may exist even if an explanation for its mechanism is unknown; the mechanism for the painkiller and anti-inflamatory drug *aspirin*, in wide use since the early 20th century, was only correctly identified in the 1970s.

For example, in medical imaging, hand designed features from biological experts might lead to explainable methods for patients, physicians, and regulators. However, deep learning methods can show higher test accuracy, even if interpreting their 'black box' nature to explain their decisions remains an open question.

For your final project, you create a CNN to diagnose skin cancer from skin lesion images in the International Skin Imaging Collaboration (ISIC) dataset. In testing, you find that your CNN can detect this disease with 90% accuracy. Prior technology can detect this disease with only 60% accuracy. You have no medical qualification. You cannot explain how your CNN accurately detects the disease so accurately, just that it does.

Q5.1:

Suppose you were asked by Rhode Island Hospital to consider using this model in a clinical setting. Describe three factors or assessments that would make you more or less confident in its use and explain why. **[6–9 sentences]**

Only Q5.1 should be on this page

There are many attempts to explain CNN predictions; we refer the interested reader to Christoph Molar's Interpretable ML Book. Let's consider three:

1. In class, we saw Adam Harley's neural network visualizer, which visualizes how a CNN predicts numerals when trained on the MNIST dataset.
2. In the code, we will use Local Interpretable Model-agnostic Explanations (LIME) to attempt to explain model predictions. LIME assesses which image superpixels are important to a classification, independent of any inner workings of the model.
3. Saliency maps attempt to show which image pixels are sensitive to the classification via the dependence on the inner working of the model via its gradients. Please read Section 28 Examples for examples and analysis.

In the ISIC dataset, it was found that "half of the images of benign lesions contain elliptical coloured patches (colour calibration charts), whereas the malignant lesion images contain none" (Nauta et al., see Figure 1 for patch examples) and that imaging rulers and standard surgical ink markers were unevenly distributed in the data. These can lead to *spurious correlations* in training where unimportant image features are associated to classification outcomes. Winkler et al. found that the presence of ink markings led to a 40% increase in false positive rate (see Figure 1 for ink marking examples).

Q5.2 — [6 points]

To what extent would each of these interpretability methods improve your understanding of your medical imaging CNN's predictions, and why? To what extend would each of these interpretability methods help explain the CNNs decision to a patient? **[6–9 sentences]**

Q6 — [10 points]
Two important notes:

1. *The next question is extra credit, worth up to 10 pts.*
   If you are capstoning the class, this can count as your required extra credit attempt for HW5; none further in the code is required.
2. The next question provides details for how we construct simple neural networks from scratch with sigmoid and softmax activations. Then, it asks you to map this to Python by filling in gaps in functions given in stencil code. *It is longer and more involved, in tutorial style, and requires careful reading.* If you want insight from implementing a simple neural network, then this is the question to answer.

Background: Let us consider using a neural network (non-convolutional) to perform classification on the MNIST dataset of handwritten digits, with 10 classes covering the digits 0–9. Each database image is 28×28 pixels, and comes with a label as supervision denoting the image's class $\mathbf{y} = (y_1, y_2, \ldots, y_i, \ldots, y_{10})$. As our image is 28×28, the network input is a linearization of each image into a 784-dimensional vector $\mathbf{x} = (x_1, x_2, \ldots, x_i, \ldots, x_{784})$. The output of the neural network is a probability distribution $\mathbf{p} = (p_1, \ldots, p_j, \ldots, p_{10})$ over the 10 classes. Suppose our network has one fully-connected layer with 10 neurons—one for each class. Each neuron has a weight for each input $\mathbf{w} = (w_1, \ldots, w_i, \ldots, w_{784})$, plus a bias b. As we only have one layer with no multi-layer composition, there's no need to use an activation function.

When we pass in a vector $\mathbf{x}$ to this layer, we will compute a 10-dimensional output vector $\mathbf{l} = (l_1, l_2, ..., l_j, ..., l_{10})$ as:

$$l_j = \mathbf{w}_j \cdot \mathbf{x} + b_j = \sum_{i=1}^{784} w_{ij} x_i + b_j \tag{1}$$

These distances from the hyperplane are sometimes called 'logits' (hence l) when they are the output of the last layer of a network. In our case, we only have *one* layer, so our single layer is the last layer.

Often we want to talk about the confidence of a classification. So, to turn our logits into a probability distribution $\mathbf{p}$ for our ten classes, we apply the *softmax* function:

$$p_j = \frac{e^{l_j}}{\sum_j e^{l_j}} \tag{2}$$

Each p_j will be positive, and $\sum_j p_j = 1$, and so softmax is guaranteed to output a probability distribution. Picking the most probable class provides our network's prediction.

If our weights and biases were optimized, Eq. 2 would classify a new test example.

We have two probability distributions: the true distribution of answers from our training labels $\mathbf{y}$, and the predicted distribution produced by our current classifier $\mathbf{p}$. To train our network, our goal is to define a loss to reduce the distance between these distributions.

Let $y_j = 1$ if class j is the true label for x, and $y_j = 0$ if j is not the true label. Then, we define the *cross-entropy loss*:

$$L(w, b, x) = -\sum_{j=1}^{10} y_j \ln(p_j), \tag{3}$$

which, after substitution of Eqs. 2 and 1, lets us compute an error between the labeled ground truth distribution and our predicted distribution (remember: error = 1 - accuracy).

So...why does this loss L work? Using the cross-entropy loss exploits concepts from information theory—Aurélien Géron has produced a video with a succinct explanation of this loss. Briefly, the loss minimizes the difference in the amounts of information needed to represent the labeled ground truth distribution and our predicted distribution—when the amount of information needed is similar, both distributions are similar, and our neural network predicts the correct labels for the input images. Other losses are applicable, with their own interpretations, but these details are beyond the scope of this course.

Onto the training algorithm. The loss is computed once for every different training example. When every training example has been presented, we call this an *epoch*. We will train for many epochs until our loss over all training examples is minimized.

Neural networks are usually optimized using gradient descent. For each training example in each epoch, we compute gradients via backpropagation (an application of the chain rule in differentiation) to update the classifier parameters w_{ij}, b_j via a learning rate λ:

$$w_{ij} = w_{ij} - \lambda \frac{\partial L}{\partial w_{ij}}, \tag{4}$$

$$b_j = b_j - \lambda \frac{\partial L}{\partial b_j}. \tag{5}$$

To update the weights and bias with gradient descent via Eqs. 4 and 5, we must deduce $\frac{\partial L}{\partial w_{ij}}$ and $\frac{\partial L}{\partial b_j}$ as expressions in terms of x_i and p_j.

Intuition: Let's just consider the weights for now. Recall that our network has one layer of neurons followed by a softmax function. To compute the change in the cross-entropy loss L with respect to neuron weights $\frac{\partial L}{\partial w_{ij}}$, we will need to compute and chain together three different terms:

1. The change in the loss with respect to the softmax output $\frac{\delta L}{\delta p_j}$,
2. The change in the softmax output with respect to the neuron output $\frac{\delta p_j}{\delta l_j}$, and
3. The change in the neuron output with respect to the neuron weights $\frac{\delta l_j}{\delta w_{ij}}$.

We must derive each individually, and then formulate the final term via the chain rule. The biases follow in a similar fashion.

The derivation is beyond the scope of this class, and so we provide them here:

$$\frac{\delta L}{\delta w_{ij}} = \frac{\delta L}{\delta p_a}\frac{\delta p_a}{\delta l_j}\frac{\delta l_j}{\delta w_{ij}} = \begin{cases} x_i(p_j - 1), a = j \\ x_i p_j, a \neq j \end{cases} \tag{6}$$

$$\frac{\delta L}{\delta b_j} = \frac{\delta L}{\delta p_a}\frac{\delta p_a}{\delta l_j}\frac{\delta l_j}{\delta b_j} = \begin{cases} (p_j - 1), a = j \\ p_j, a \neq j \end{cases} \tag{7}$$

Here, a is the predicted class label and j is the true class label. An alternative form you might see shows $\frac{\delta L}{\delta w_{ij}} = x_i(p_j - y_j)$ and $\frac{\delta L}{\delta b_j} = p_j - y_j$ where $y_j = 1$ if class j is the true label for x, and $y_j = 0$ if j is not the true label.

So...after all of that, our gradient update rules are surprisingly simple!

Further details: For interested students, we refer students to Chapter 1 of Prof. Charniak's deep learning notes, which derives these gradient update rules. We also refer to Prof. Sadowski's notes on backpropagation: Section 1 derives terms first for cross-entropy loss with logistic (sigmoid) activation, and then Section 2 derives terms for cross-entropy loss with softmax. The Wikipedia article on the backpropagation algorithm likewise represents a derivation walkthrough of the general case with many hidden layers each with sigmoid activation functions, and a final layer with a softmax function.

We will implement these steps in code using numpy. We provide a code stencil `main.py` which loads one of two datasets: MNIST and the scene recognition dataset from Homework 4. We will consider two models: a neural network, and then a neural network whose logits are used as input to an SVM classifier—this is a hybrid approach closer to HW4 but where the feature transform is learned end-to-end. Please look at the comments in `main.py` for the arguments to pass in to the program for each condition.

The neural network model is defined in `model.py`, and the parts we must implement are marked with TODO comments. Each requires only a few lines of code at most:

1. Initialize the weights and biases in `__init__()`
2. Complete the overall training flow in `train_nn()`
3. Define the forward pass in `forward_pass()`
4. Define the gradient descent update in `gradient_descent()`
5. Define the cross-entropy loss in `loss()`
6. Define the back-propagation rule in `back_propagation()`

Tasks: Implement the neural network and its optimization. **[10 points]**

Please upload your completed `model.py` to Gradescope under the assignment titled "Homework 5 Written (Numpy)". The autograder will check that you correctly implemented parts of your model. Then, report your accuracy here.

Run your model on all four conditions and report training loss (calculated using *training* set) and accuracy (calculated using *testing* set) as the number of training epochs increases. Why does the SVM perform differently?

- NN on MNIST: xx% (highest accuracy)
  - Epoch 0 loss: xx Accuracy: xx%
  - Epoch 9 loss: xx Accuracy: xx%
- NN+SVM on MNIST: xx% (highest accuracy)
  - Epoch 0 loss: xx Accuracy: xx%
  - Epoch 9 loss: xx Accuracy: xx%
- NN on SceneRec: xx% (highest accuracy)
  - Epoch 0 loss: xx Accuracy: xx%
  - Epoch 9 loss: xx Accuracy: xx%
- NN+SVM on SceneRec: xx% (highest accuracy)
  - Epoch 0 loss: xx Accuracy: xx%
  - Epoch 9 loss: xx Accuracy: xx%

Why does the SVM perform differently?
TODO: Answer here

Feedback? (Optional)

We appreciate your feedback on how to improve the course. You can provide anonymous feedback through this form, that can be accessed using your Brown account (your identity will not be collected). If you have urgent non-anonymous comments/questions, please email the instructor.