# Homework 5 Questions

## Instructions

- 2 socially-responsible computing questions, which will be expanded on in discussion sections.
- 6 technical questions (Q7 and Q8 with code components).
- *This will take longer than usual! Q7 is the most substantial technical written part so far, and Q8 is a tricky setup—start early!*.
- Write code where appropriate; feel free to include images or equations.
- Please make this document anonymous.
- This assignment is **fixed length**, and the pages have been assigned for you in Gradescope. As a result, **please do NOT add any new pages**. We will provide ample room for you to answer the questions. If you *really* wish for more space, please add a page *at the end of the document*.
- **We do NOT expect you to fill up each page with your answer.** Some answers will only be a few sentences long, and that is okay.
- Question 7 has a coding component that must be submitted to Gradescope under a separate assignment (Homework 5 Written (Numpy)).

**Q1:** 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.

(a) What is different about the construction of the filters in each of these approaches?

(b) Please declare and explain the advantages and disadvantages of these two approaches.

**A1:** Your answer here.

**Q2:** Many CNNs have a fully connected multi-layer perceptron (MLP) after the convolutional layers as a general purpose 'decision-making' subnetwork. What effects might a *locally-connected* MLP have on computer vision applications, and why?

Please give your answer in terms of the learned convolution feature maps, their connections, and the perceptrons in the MLP.

**A2:** Your answer here.

**Q3:** Given a neural network classifier and the stochastic gradient descent training approach, discuss how the *learning rate*, *batch size*, and *number of epochs* hyperparameters might affect the training process and outcome.

**A3:** Your answer here.

**Q4:** What effects does adding a max pooling layer have for a single convolutional layer, where the output with max pooling is some size larger than 1 x 1 x d?

*Notes:* 'Global' here means whole image; 'local' means only in some image region.

*LaTeX:* To fill in boxes, replace '\square' with '\blacksquare' for your answer.

**A4:** Multiple choice. Choose all that apply.

| | |
|---|---|
| Increases computational cost of training | ☐ |
| Decreases computational cost of training | ☐ |
| Increases computational cost of testing | ☐ |
| Decreases computational cost of testing | ☐ |
| Increases overfitting | ☐ |
| Decreases overfitting | ☐ |
| Increases underfitting | ☐ |
| Decreases underfitting | ☐ |
| Increases the nonlinearity of the decision function | ☐ |
| Decreases the nonlinearity of the decision function | ☐ |
| Provides local rotational invariance | ☐ |
| Provides global rotational invariance | ☐ |
| Provides local scale invariance | ☐ |
| Provides global scale invariance | ☐ |
| Provides local translational invariance | ☐ |
| Provides global translational invariance | ☐ |

**Something to think about (ungraded):** Given some input to a convolutional layer with stride $2\times2$ and kernel size $3\times3$, and ignoring the boundary, what is the minimum number of convolutional filters required to preserve all input information in the output feature map?

Multiple choice. *LaTeX:* Use command '\bullet' (•) to fill in the dots.

| | |
|---|---|
| 0.5 | ☐ |
| 1 | ☐ |
| 2 | ☐ |
| 4 | ☐ |
| It's impossible | ☐ |

*Here:* Feel free to leave optional short written justification if desired.

**Q5.1:** CNN interpretability remains an open question. In medical imaging, hand designing features from biological experts might lead to a generalizable and explainable method for physicians, regulators, or patients. However, deep learning methods can show higher test accuracy, even if their results may be less explainable.

For your final project, you decide to create a CNN for medical imaging diagnosis. After extensive testing, you find that your CNN can detect a rare disease with 90% accuracy. Prior technology can detect this disease with only 50% accuracy. You have no medical background, training, or professional qualification. You cannot explain how your CNN accurately detects the disease so accurately, just that it does.

Should your model be used in a clinical setting? Describe three factors that would make you more or less confident in its use. If your model is deployed and an incorrect result causes patient harm, who should be held responsible? [6–7 sentences]

**A5.1:** Your answer here.

**Q5.2:** Please complete this multiple choice question about the previous scenario.

What answer did you pick, and why?

**A5.2:** Your answer here.

**Q6:** There have been many attempts to interpret CNN predictions:

- We saw this website in class, which visualizes how a CNN predicts numerals when trained on the MNIST dataset.
- This homework asks you to use LIME to attempt to explain your model predictions.
- Another option is saliency maps. Please read this short article describing how saliency maps work, and see more examples here from a saliency map software package.

To what extent would each of these methods improve your understanding of your medical imaging CNN's predictions, and why? More generally, when is it important to be able to interpret the reasons a machine made a decision, and when is it not? [6–7 sentences]

**A6:** Your answer here.

**Q7 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 image is 28×28 pixels, and so the network input is a 784-dimensional vector $\mathbf{x} = (x_1, x_2, \ldots, x_i, \ldots, x_{784})$. The output of the neural network is 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 trained, 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 (remember that: error = 1 - accuracy) between the labeled ground truth distribution and our predicted distribution.

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 two distributions. Other losses are also 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 to the training process, we call this an *epoch*. Typically 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 via a learning rate $\lambda$:

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

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

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. Recall that our network has one layer of neurons followed by a softmax function. To compute the change in the cross-entropy loss 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 $\frac{\delta l_j}{w_{ij}}$.

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:

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

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

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.

---

**Q7:** 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 also provide two models: a neural network, and then a neural network whose logits are used as input to an SVM classifier. 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.

*Tasks:* Please follow the steps to implement the forward model evaluation and backward gradient update steps. Then, 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—see A7.1 below.

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, complete the following questions.

**A7.1:** How well did each model perform on each dataset? Please use this table to structure your response.

- 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% s

**Q7.2:** What do the training loss and accuracy numbers tell us about a) the capacity of the network, b) the complexity of the two problems, and c) the value of the two different classification approaches?

**A7.2:** Answer here.

**Q8:** Follow the GCP guide to set up Google Cloud Platform access. *Note:* This is tricky, and it will take you some time to become familiar with the system! Once finished, complete one of these two Tensorflow MNIST tutorials on GCP:

- TensorFlow 2 quickstart for beginners
- TensorFlow 2 quickstart for experts

The work for the quickstart can be completed in a new Python file inside or outside of your homework code, as you won't have to turn in any quickstart code.

**A8:** Please insert a screenshot of the quickstart code output, running on GCP:

## Feedback? (Optional)

Please help us make the course better. If you have any feedback for this assignment, we'd love to hear it!