Homework 5 Written Questions

Document Instructions

- 6 questions **[8 + 8 + 6 + 11 + 4 + 8 = 45 + 3 bonus points]**.
- Fill all your answers within the answer boxes, and **please do NOT remove the answer box outlines**.
- Questions are highlighted in the **orange boxes**, bonus questions are highlighted in **blue boxes**, answers should be recorded in the **green boxes**.
- Include code, images, and equations where appropriate.
- To identify all places where your responses are expected, search for 'TODO'.
- The answer box sizes have been set by the staff beforehand and will truncate your text if it goes beyond the limit. Please make sure your responses fit in the appropriate spaces. **Extra pages are not permitted unless otherwise specified.**
- Make sure your submission has the right number of pages to validate page alignment sanity (check the footer).
- Please make this document anonymous.

Gradescope Instructions

- When you are finished, compile this document to a PDF and submit it directly to Gradescope.
- The pages will be automatically assigned to the right questions on Gradescope *assuming you do not add any unnecessary pages*. **Inconsistently assigned pages will lead to a deduction of 2 points per misaligned page (capped at a maximum 6 point deduction).**

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.

(a) **[4 points]**

What is different about the construction of the filters in each of these approaches? **[6-7 sentences]**

Convolutional filters used in traditional computer vision algorithms have fixed weights. On ther other hand, filters used in convolutional neural networks are not fixed; they are learned. To take an example of gaussian filter of size 3 by 3, the weights of the filter. Sobel filter also has fixed weights. Filters of CNN, however, are initialized randomly. Usually sampled from gaussian distribution. Then the weights get learned by backpropagation.

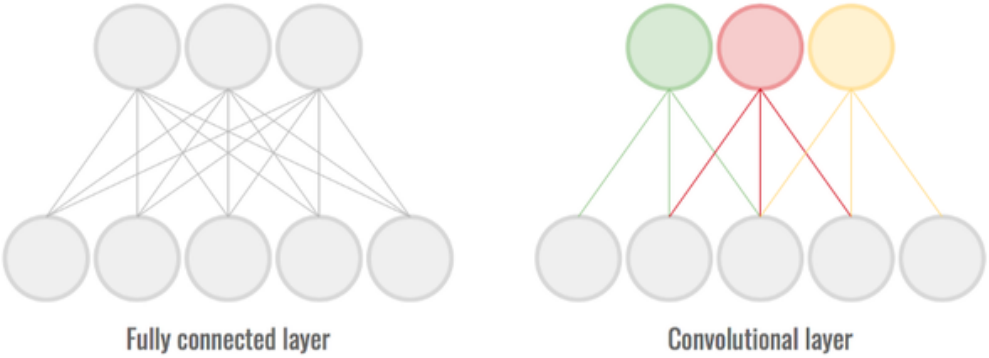
(b) **[4 points]**

Please declare and explain at least two advantages and disadvantages of each of these two approaches. **[6-7 sentences]**

Traditional approach is easy to interpret: we know why and how our model is working. Hence it's relatively easier to interpret what our model / filter is doing / focusing at. Also this approach doesn't require big memory since it doesn't learn weights. However it's hard to optimize. As mentioned in previous question, there's less freedom; filter weights are fixed, which ends up in low accuracy.
On the contrary, CNN approach learns weights as we don't fix weights in filters. Models can learn the weights that best fit to given data. But this approach requires a lot of training data. Also, CNN is hard to interpret. It's a blackbox model.Although we don't understand how it works, we can use CNN for complicated tasks.

Q2: **[8 points]** Many CNNs have a fully connected multi-layer perceptron (MLP) after the convolutional layers as a general purpose 'decision-making' subnetwork. Unlike in convolutional layers and locally connected MLPs, where not every input connects to every perceptron, a fully connected MLP guarantees that every input is conntected to every perceptron.

What differences in impact could be caused by using a *locally-connected* MLP versus a *fully-connected* MLP, and why?



(a) **[4 points]**

Explain these differences in terms of learned convolution feature maps, their connections, and the perceptrons in the MLP. **[5 - 6 sentences]**

Learned convolution feature maps: Fully connected layer is adequate for classification as it connects with all perceptrons from previous layer. Locally connected MLP would work better for spatial arrangement.

Connection: Convolutional layers extracts features from input images. Fully connected layer uses the extracted features to classify given images. If we used the same weight for locally connected MLP, that would be the same thing as convolutional neural network.

Perceptrons in the MLP: FC has much more number of parameters. It is likely to suffer overfitting. In the above image, for example, the number is 15. Locally connected MLP has requires less parameters. In the above image, it's 9.

(b) **[4 points]**

Explain these differences in the context of a few different real world examples of computer vision. What cases would a locally connected MLP work better than a fully connected MLP? What could be the stakes if the wrong type of MLP is chosen? **[5 - 6 sentences]**

Locally connected MLP might work better for face recognition. If we used a fully connected MLP, a model would think two different faces are the same because they are both faces. Or hard to find which parts (such as nose, mouse, etc) are located where. But locally connected MLP would work well only when input images are registered as each filter has different weights and covers only specific region of images. For example, kernel that covers regions where left eye is located would be trained to located eye. So if an input image is augmented such that the face is far way, it would be hard for a model to recognize a left eye.

Q3: [6 points]

Given a neural network classifier and the stochastic gradient descent training approach, discuss how the following hyperparameters might affect the training process and outcome.

(a) **[2 points]** Learning rate **[4-6 sentences]**

Training process will generally take longer if we used too low learning rate as it might take more time to optimize the model to have the lowest loss. If we used too large learning rate, a model would have diverge output; it won't converge to global minimum of loss functions.

(b) **[2 points]** Batch size **[4-6 sentences]**

If we used small batch size, loss will zig zag but eventually converge to minimum loss. This might take more "steps" than bigger batch size. But considering the time it takes for each epoch, this could be faster. Also, it's more efficient to update our weights for each batch rather than updating for entire training data, for example. Batch size wouldn't impact much on output; using any batch size for training would end up in similar accuracies.

(c) **[2 points]** Number of epochs **[4-6 sentences]**

If the number of epochs is too small, it's highly likely that the model won't converge to minimum. The loss would be too high. That means the outcome isn't reliable at all. As for training process, the higher the number of epochs, the more time it takes to train the model of course. Other than that, it would be the same what number of epochs we use because one epoch means we train all the batches once. We can set the number of epoch from one to infinity.

Q4: [11 + 1 bonus points] What effects are caused by adding a spatial max pooling layer (stride > 1) after a single convolutional layer, where the output with max pooling is some size larger than $1 \times 1 \times d$?

There may be multiple correct answers per question.

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

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

(a) **[1 point]**

What happens to the computational cost of training?

TODO: Select the right option

Increases ☐
Decreases ■

(b) **[1 point]**

What happens to the computational cost of testing?

TODO: Select the right option

Increases ☐
Decreases ■

(c) **[2 points]** Read this article about the environmental impacts that deep learning can have.

How does the implementation of this max pooling layer and its resulting implications for the model's computational costs fit into this discussion? **[3-4 sentences]**

If we used max pooling, there are much less weights that should be trained. This will reduce carbon footprints by large amount. The output of maxpooing is 1*1*d, which means I only pick one max pixel value from each filter of the previous output.

(d) **[1 point]**

What happens to overfitting?

TODO: Select the right option

Increases	☐
Decreases	■

(e) **[1 point]**

What happens to underfitting?

TODO: Select the right option

Increases	■
Decreases	☐

(f) **[2 points]**

Briefly discuss the tradeoff between overfitting and underfitting in the context of a CNN. Why wouldn't you want to go too far in either direction and what real-world stakeholders might it affect if you do? **[3 - 5 sentences]**

Overfitted model will predict outputs with high accuracy. This can be bad in real-time applications because companies will use this model to predict of unseen data. Overfitted model will perform poorly as it has weak generalization. Overfitted model could have chance to be used in market at least. Underfitted model can't be even used at all as it has poor prediction accuracy on training data set. But we could use it as a weak reference for estimating very rough output. In short, trade off between oerfitting and underfitting relates with bias and variance; overfitting: high variance, low bias, underfitting: low variance, high bias.

(g) **[1 point]**

What happens to the nonlinearity of the decision function?

TODO: Select the right option

Increases	☐
Decreases	■

(h) **[2 point]**

Which of the following occur?

TODO: Select all that apply

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

(i) **(Bonus) [1 point]**

Given some input to a convolutional layer with stride 2×2 and kernel size 3×3, 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.

Tip: Use command '\bullet' (•) to fill in the dots.

TODO: Choose one

0.5	☐
1	☐
2	☐
4	■
It's impossible	☐

Q5: **[4 points]** 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.

CNN interpretability remains an open question. For example, 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.

When is it important to be able to interpret the reasons a machine made a decision, and when is it not? **[5-7 sentences]**

Availability of interpreting the reasons is not important when humans can agree with the output. Theses kinds of tasks are mostly done by machine because it's time consuming. For example, if a task is as simple as classifying whether an image is cat or not, we don't need to understand why a machine made a specific decision because we know the answer. It's important to be able to interpret the reasons for decisions when we ask a machine to solve problem that we don't know. We don't know if an output is correct or wrong; we need to understand why a machine made this decision to be convinced of the correctness of the result.

Q6 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 λ:

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

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

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.

Q6: 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.

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.

(a) **[4 points]**

How well did each model perform on each dataset?

- NN on MNIST: 89% (highest accuracy)
  - Epoch 0 loss: 167709 Accuracy: 88%
  - Epoch 9 loss: 132571 Accuracy: 89%
- NN+SVM on MNIST: 90% (highest accuracy)
  - Epoch 0 loss: 169104 Accuracy: 87%
  - Epoch 9 loss: 131445 Accuracy: 88%
- NN on SceneRec: 15% (highest accuracy)
  - Epoch 0 loss: 30814 Accuracy: 12%
  - Epoch 9 loss: 23937 Accuracy: 12%
- NN+SVM on SceneRec: 21% (highest accuracy)
  - Epoch 0 loss: 31019 Accuracy: 16%
  - Epoch 9 loss: 23228 Accuracy: 19%

(b) **[4 points]**

What do the training loss and accuracy numbers tell us about a) the capability of the network, b) the complexity of the two problems, and c) the usefulness of the two different classification approaches?

Both networks work well on the MNIST dataset but not on the SceneRec dataset. The difference stems from characteristics of two datasets. Both networks easily learn simple dataset such as white number on black background. However, scene classification require more complicated process; we need to find each object in the scene that represents a class of scene. For example, chairs and desks are usually seen in indoor, not outdoor.
For MNIST dataset, there's no big difference between two networks. So SVM has no big use on thise dataset. On the other hand, SVM improves the accuracy quite a bit on SceneRec data set.

Discussion Attendance:

Extra Credit: [2 points]

Please mark this box only if you've attended the discussion session in person.

■ I attended the discussion session on DATE

Only discussion attendance should be on this page

+

Feedback? (Optional)

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