

Homework 5 Written Questions

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**] (good 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.

(a) **[4 points]**

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

The difference between the construction of convolutional filters in classical machine learning techniques and those used in convolutional neural networks is that the filter kernels in CNN's are learned. The weights in the kernels in classical techniques are designed by hand with a human-designed purpose in mind: to mimic gaussians or detect edges, for example. In addition, in a CNN convolution typically refers to correlation (where the kernel is not rotated).

(b) [4 points]

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

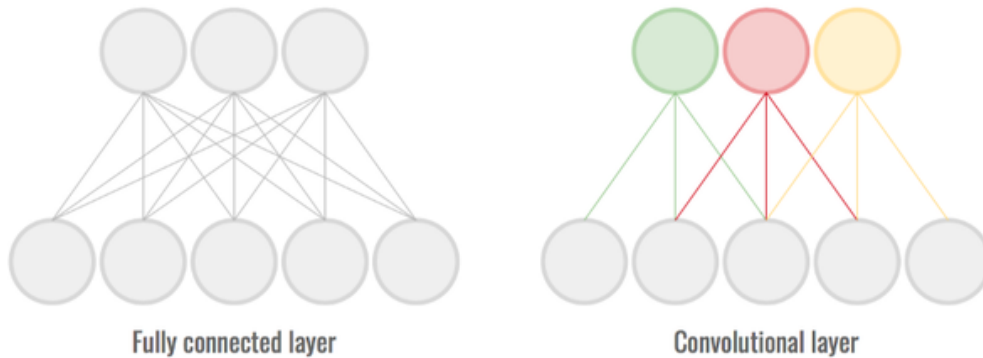
An advantage to human-designed kernels is that they are well-understood by human researchers. Researchers know what makes certain kernels effective for specific tasks, which gives a system that uses them a degree of interpretability by default.

Another advantage to human-designed kernels is that they can be added to a model quickly without additional training. If we would like to learn the weights for a kernel, that takes additional time, while human-designed kernels can be used out of the box.

An advantage of learned kernels is that they can be tweaked to work well for a specific task. An effective machine learning method can pick values for kernel that takes advantage of patterns in the data that humans don't recognize.

Another disadvantage of human-designed kernels is that there just aren't that many of them. Humans can design kernels for a few specific tasks, but cannot feasibly design the number needed for deeply nesting kernels like in a CNN.

Q2: [6 points] Many CNNs 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 operates on local windows of the previous layer activations.



Suppose we imagine a ‘locally-connected MLP’, where every output perceptron is only connected to a small subset of nearby inputs.

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]**

With a fixed number of nodes in the network, a dense multi-layer perceptron will take more time to train. There are simply more weights to be optimized. This can mean that a final model produced this way would be poorly optimized if there is a limited budget of training time. In addition, making a network more dense has diminishing benefits and may not have that large of an improvement in performance. A dense network could take longer to evaluate and require more memory in order to perform the additional calculations. A third difference is that dense multi-layer perceptron may have better performance on training since it can fit more of the details of the model.

Q3: [6 points]

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

(a) [2 points] Learning rate [2–3 sentences]

A larger learning rate can make training less predictable as it takes larger steps in the feature space. This can lead to it oscillating back and forth over a bowl-region in the feature space. A smaller learning rate can have the downside of being more likely to get caught in local minima, which can decrease the performance of the final model.

(b) [2 points] Batch size [2–3 sentences]

Having a larger batch-size can make training slower. However, a small batch size has the downside that a small batch may be less representative of the true data, and so a model produced with a small batch size may perform worse. In addition, training with a small batch size can be unstable.

(c) **[2 points]** Number of epochs **[2–3 sentences]**

With more epochs, the model is almost certain to better fit to the training data. However, the downside of an increased number of epochs is that the model may learn to overfit on patterns in the data that reflect noise in the data, which can lead to worse test performance. Some models may become additionally unstable after a large number of epochs of training.

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.

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

(a) [1 point]

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

Increases	<input type="checkbox"/>
Stays the same	<input checked="" type="checkbox"/>
Decreases	<input type="checkbox"/>

(b) [1 point]

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

Increases	<input type="checkbox"/>
Stays the same	<input type="checkbox"/>
Decreases	<input checked="" type="checkbox"/>

(c) [1 point]

What happens to the complexity of the decision function?

TODO: Select the right option

Increases	<input type="checkbox"/>
Stays the same	<input type="checkbox"/>
Decreases	<input checked="" type="checkbox"/>

(d) [1 point]

What happens to the potential for overfitting?

Increases	<input type="checkbox"/>
Stays the same	<input type="checkbox"/>
Decreases	<input checked="" type="checkbox"/>

(e) [1 point]

What happens to the potential for underfitting?

Increases	<input checked="" type="checkbox"/>
Stays the same	<input type="checkbox"/>
Decreases	<input type="checkbox"/>

- (f) [3 point] 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 detect objects at any position in the image.

Which of the following does max pooling provide?

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

Provides local rotational invariance	<input checked="" type="checkbox"/>
Provides global rotational invariance	<input type="checkbox"/>
Provides local scale invariance	<input checked="" type="checkbox"/>
Provides global scale invariance	<input type="checkbox"/>
Provides local translational invariance	<input checked="" type="checkbox"/>
Provides global translational invariance	<input type="checkbox"/>

- (g) [2 point]

In a CNN, is the convolutional property of translation equivariance maintained through max pooling? Why or why not?

Another way to think about it: Suppose I was classifying birds. If a bird translates within the image frame but remains 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. [4–6 sentences]

Max pooling does preserve translation equivariance. The pooling process samples from a neighborhood of pixels in order to determine the activation in the new layer. Since the max pool takes the maximum of the activations in the previous layer, shifts that take place on the order of the window size will not be observed. This is a special property of the max function, since it is naturally anti-aliasing. The network will be equivariant to shifts larger than the window size since they will lead to activations across other windows.

Q5 (a): [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-inflammatory 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.

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]**

One factor that would make me more confident in its use would be if my model's test accuracy and training accuracy were both comparably high. This would make me more likely to think that the model was picking up on true patterns within the data instead of fitting on noise or irrelevant details.

Another factor that would make me more confident in its use is if Rhode Island Hospital did not have access to good prior detection technology and their human doctors performed poorly at detecting these images. If Rhode Island Hospital had no other better options, then a deploying a poorly understood machine learning model would be undesirable but would might be warranted.

A factor that would make be less confident in its use is if the International Skin Imaging Collaboration dataset was not representative of the kinds of images that Rhode Island Hospital would capture clinically. If my model was being deployed outside of its training environment, then its behavior could be unpredictable and create harm due to false positive or false negative diagnoses.

Q5 (b): [6 points] 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 [Sections 10.2.6–10.2.9](#) 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).

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

The LIME technique would be effective in understanding the predictions of the model because it communicates which features in the image were integral in making the decision. For example, it can provide a view of the features in the image that were important in making the decision, since these superpixels being turned on or off show the change in the confidence of the decision.

The saliency map could also help with understanding the prediction because it gives a view into which features of the image had the biggest impact on the final outcome. If the saliency map showed a high activation around features of the image, like rulers, it may cast some doubt into the decision.

Both the LIME maps and the saliency maps could be shown to a patient, who could understand that they represented the features in the lesion that the model thought were most significant for the diagnosis. This could be comparable to a human doctor describing the features of the lesion that they thought were alarming.

Q6: Two important notes:

1. *The next question is extra credit, worth up to 10 pts. If you are capstoning the class, this can also 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.*

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 database image is 28×28 pixels, and comes with a label as supervision denoting the image’s class $\mathbf{y} = (y_1, y_2, \dots, y_i, \dots, y_{10})$. As our image is 28×28 , the network input is a linearization of each image into a 784-dimensional vector $\mathbf{x} = (x_1, x_2, \dots, x_i, \dots, x_{784})$. The output of the neural network is a probability distribution $\mathbf{p} = (p_1, \dots, p_j, \dots, p_{10})$ 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 $\mathbf{w} = (w_1, \dots, w_i, \dots, w_{784})$, 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 $\mathbf{l} = (l_1, l_2, \dots, l_j, \dots, l_{10})$ as:

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

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:

$$p_j = \frac{e^{l_j}}{\sum_j e^{l_j}} \quad (2)$$

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*:

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

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 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 :

$$w_{ij} = w_{ij} - \lambda \frac{\partial L}{\partial w_{ij}}, \quad (4)$$

$$b_j = b_j - \lambda \frac{\partial L}{\partial b_j}. \quad (5)$$

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:

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

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

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. 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)

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