SYDE 671 Advanced Image Processing

David Abou Chacra University of Waterloo Fall 2021

Assignment 4 - Questions Part

Instructions

- 5 questions
- This is a long assignment, so make sure you start early!
- Write code where appropriate; feel free to include images or equations.
- Your final submission will be one Jupyter Notebook (and you can also include any scripts you wrote for Q5)

Questions

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.

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.

Q3: Given a neural network being trained using the stochastic gradient descent approach, discuss how the *learning rate*, *batch size*, and *training time* hyperparameters might affect the training process and outcome.

Q4: (multiple choice question) 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' in the following context means whole image; 'local' means only in some image region.

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	☐

Q5 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 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: The interested is referred to Chapter 1 of Prof. Charniak's deep learning notes, which derives these gradient update rules. You are also referred 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.

Q5: You will implement these steps in code using numpy. We provide a code stencil `main.py` which loads one of two datasets: MINST and the scene recognition dataset from Assignment 3. You are also provided with 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 you must implement are in function `train_nn()`.
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 and accuracy as the number of training epochs increases.
What do these numbers tell us about the capacity of the network, the complexity of the two problems, the value of training, and the value of the two different classification approaches?
Please also include your `train_nn()` code here (without the helper comments).

A5:

- 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%

```
        def train_nn(self):
        indices = list(range(self.train_images.shape[0]))
        delta_W = np.zeros((self.input_size, self.num_classes))
        delta_b = np.zeros((1, self.num_classes))

        for epoch in range(hp.num_epochs):
        loss_sum = 0
        random.shuffle(indices)

        for index in range(len(indices)):
        i = indices[index]
        img = self.train_images[i]
        gt_label = self.train_labels[i]

        ################
        # FORWARD PASS:
        # Step 1:

        # Step 2:

        # Step 3:

        #loss_sum = loss_sum + your_loss

        ################
        # BACKWARD PASS (BACK PROPAGATION):
        # Step 4:

        # Step 5:

        print( "Epoch "+str(epoch)+": Total loss: "+str(loss_sum) )
```

Credit

This assignment is adapted from CSCI1430 course