CS480/680: Introduction to Machine Learning

Homework 3

Due: 11:59 pm, November 12, 2019, submit on LEARN.

Include your name, student number and session!

Submit your writeup in pdf and all source code in a zip file (with proper documentation). Write a script for each programming exercise so that the TAs can easily run and verify your results. Make sure your code runs!

[Text in square brackets are hints that can be ignored.]

Exercise 1: Automatic Differentiation (20 pts)

Recall that in a computational graph \( G \), each node \( v \) executes some function \( f_v \) on its input nodes \( I_v \) and then sends the output, which we also denote as \( v \), i.e. \( v = f_v(I_v) \). Take two arbitrary nodes \( u \) and \( v \) in \( G \), we claim the following formula:

\[
\frac{\partial u}{\partial v} = \sum_{P \in \text{path}(v \to u)} \prod_{(v_i, v_{i+1}) \in P} \frac{\partial v_{i+1}}{\partial v_i},
\]

where \( \frac{\partial v_{i+1}}{\partial v_i} = \frac{\partial f_{v_{i+1}}}{\partial v_i} \) and by definition node \( v_i \) is an input to node \( v_{i+1} \).

1. (5 pts) Compute the following function (i.e., express the sink node \( t \) as a function of the source node \( s \)):

\[
\begin{align*}
&\text{\( s \)} & \text{\( s^2 \)} & \text{\( \log(1 + a) \)} & \text{\( b \)} & \text{\( \sqrt{c} \)} & \text{\( d + 1 \)} & \text{\( t \)} \\
&\text{\( u \)} & \text{\( v \)} & \text{\( w \)} & \text{\( x \)} & \text{\( y \)} & \text{\( z \)} & \text{\( t \)}
\end{align*}
\]

(The function implemented by each node is depicted on the incoming edge. For node \( d \), we also add the two incoming edges.) [Not required: change addition in node \( d \) to multiplication.]

2. (5 pts) Compute the derivative of \( t \) wrt \( s \) using (1). Does it match the “usual derivative” that you can derive using the explicit function computed in item 1?

3. (10 pts) Prove the formula (1). [Not required: how would you efficiently compute formula (1)?]

Exercise 2: Regularization (20 pts)

Notation: For the vector \( \mathbf{x}_i \), we use \( x_{ji} \) to denote its \( j \)-th element.

Overfitting to the training set is a big concern in machine learning. One simple remedy to avoid overfitting to any particular training data is through injecting noise: we randomly perturb each training data before feeding it into our machine learning algorithm. In this exercise you are going to prove that injecting noise to training data is essentially the same as adding some particular form of regularization. We use least-squares regression as an example, but the same idea extends to other models in machine learning almost effortlessly.

Recall that least-squares regression aims at solving:

\[
\min_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{n} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2,
\]

where \( \mathbf{x}_i \in \mathbb{R}^d \) and \( y_i \in \mathbb{R} \) are the training data. (For simplicity, we omit the bias term here.) Now, instead of using the given feature vector \( \mathbf{x}_i \), we perturb it first by some independent noise \( \mathbf{e}_i \) to get \( \tilde{\mathbf{x}}_i = f(\mathbf{x}_i, \mathbf{e}_i) \), with different choices of the perturbation function \( f \). Then, we solve the following expected least-squares regression problem:

\[
\min_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{n} \mathbb{E}[(y_i - \mathbf{w}^\top \tilde{\mathbf{x}}_i)^2],
\]
where the expectation removes the randomness in \( \tilde{x}_i \) (due to the noise \( \epsilon_i \)), and we treat \( x_i, y_i \) as fixed here. [To understand the expectation, think of \( n \) as so large that we have each data appearing repeatedly many times in our training set.]

1. (10 pts) Let \( \tilde{x}_i = f(x_i, \epsilon_i) = x_i + \epsilon_i \) where \( \epsilon_i \sim \mathcal{N}(0, \lambda) \) follows the standard Gaussian distribution. Simplify (3) as the usual least-squares regression (2), plus a familiar regularization function on \( w \).

2. (10 pts) Let \( \tilde{x}_i = f(x_i, \epsilon_i) = x_i \circ \epsilon_i \), where \( \circ \) denotes the element-wise product and \( p \epsilon_j \sim \text{Bernoulli}(p) \) independently for each \( j \). That is, with probability \( 1 - p \) we reset \( x_{ji} \) to 0 and with probability \( p \) we scale \( x_{ji} \) as \( x_{ji}/p \). Note that for different training data \( x_i, \epsilon_i \)'s are independent. Simplify (3) as the usual least-squares regression (2), plus a different regularization function on \( w \). [This way of injecting noise, when applied to the weight vector \( w \) in a neural network, is known as Dropout (DropConnect).]

### Exercise 3: Convolutional Neural Networks (CNN) Basics (20 pts)

Consider the following CNN architecture:

In all layers we do not add the bias term (for simplicity), and the nonlinear activations are omitted throughout. Note that the third dimension of each conv filter is omitted as it can be inferred from the previous layer, while pooling (with each pixel appearing exactly once in all pooling windows) is performed on each channel/feature map separately. Answer the following questions. [For (b) (c) (d), you will receive partial credit if you correctly derive some of the intermediate results.]

1. (5 pts) What can we do to achieve the same output size (excluding the dimension of channels/feature maps) as the input size for each convolutional layer? Explain using a filter with size \((2w + 1) \times (2h + 1)\) and stride 1 on an input image with size \(W \times H\).

2. (5 pts) Without padding what is the output size of each intermediate layer (except the first input layer and the last fully connected layer)?

3. (5 pts) What is the receptive field of a single (output) neuron on the last convolutional layer (ignoring any boundary issue)?

4. (5 pts) How many parameters do the above architecture have? For this exercise we assume the output image size of each convolutional layer is the same as its input image size (except the number of channels, of course).

### Exercise 4: CNN Implementation (40 pts)

**Note:** Please mention your Python version (and maybe the version of all other packages) in the code.

In this exercise you are going to run some experiments involving CNNs. You need to know Python and install the following libraries: Keras, Tensorflow, NumPy and all their dependencies. To perform any required image transformation, we suggest using Pillow. You can find detailed instructions and tutorials for each of these libraries on the respective websites. [To install, try pip install keras and pip install pillow. For Tensorflow, follow the installation steps on its webpage.]

For all experiments, running on CPU is sufficient. You do not need to run the code on GPUs. Before
start, we suggest you review what we learned about each layer in CNN, and read at least this [tutorial]

1. Train a VGG11 net on the [MNIST] dataset. VGG11 was an earlier version of VGG16 and can be found as model A in Table 1 of this [paper] whose Section 2.1 also gives you all the details about each layer. The goal is to get as close to 0 loss as possible. Note that our input dimension is different from the VGG paper. You need to resize each image in MNIST from its original size $28 \times 28$ to $32 \times 32$ [why?], and it might be necessary to change at least one other layer of VGG11. [This experiment will take up to 1 hour on a CPU, so please be cautious of your time. If this running time is not bearable, you may cut the training set by 1/10, so only have $\sim 600$ images per class instead of the regular $\sim 6000$.]

2. Once you’ve done the above, the next goal is to inspect the training process. Create the following plots:
   (a) (5 pts) test accuracy vs the number of epochs (say 3 ~ 5)
   (b) (5 pts) training accuracy vs the number of epochs
   (c) (5 pts) test loss vs the number of epochs
   (d) (5 pts) training loss vs the number of epochs

   [If running more than 1 epoch is computationally infeasible, simply run 1 epoch and try to record the accuracy / loss every few minibatches.]

3. Then, it is time to inspect the generalization properties of your final model. Rotate and blur the test set images using any python library of your choice. We recommend [pillow] to complete the following two plots:
   (e) (5 pts) test accuracy vs the degree of rotation. Try the following rotations: -40, -30, ... 30, 40, and plot the test accuracy at each rotation. What is the effect?
   (f) (5 pts) test accuracy vs Gaussian noise. Try adding standard Gaussian noise to each test image with variance 0.01, 0.1, 1 and plot the test accuracy vs the variance. What is the effect?

4. (10 pts) Lastly, let us verify the effect of regularization. Re-do part 1.3 with data augmentation. Create the same plots and explain what kind of data augmentation you use.