Assignment 5

BIOENG 145/245

Due Monday 3/6 11:59PM

PLEASE DO NOT MODIFY THE FUNCTION NAMES AND THE FILE NAMES OR THE AUTOGRADER WILL BREAK!!

1 Neural Network Components

In order to use gradient descent to optimize the weights of a neural network, we need to be able to find the gradient of the loss with respect to the weights through a process called back-propagation (to be discussed in detail later). This will require us to do a 'backward pass' through all the components of our neural network, and, for each component, to compute the derivative of the output with respect to the input, until we can compute the derivative of the loss with respect to the weights.

In this question, we will compute the forward pass and backward pass for a simple linear node, as well as the ReLU and softmax activation functions.

- `linear_forward`, `linear_backward`: Calculate the output of a single linear node (no activation function). Then, calculate the derivatives of the output of the node with respect to the weights and the bias.
- `relu_forward`, `relu_backward`: Calculate the output of the ReLU activation function. Then, calculate the derivative of the output with respect to the input.
- `softmax_forward`, `softmax_backward`: Calculate the output of a softmax activation function. Then, calculate the derivative of the output with respect to the input. Softmax activation is a general version of the sigmoid activation function: while the sigmoid activation function generates a single value between 1 and 0 that can be used for binary classification, the softmax function generates a vector of size K, where K is the number of classes. Each value in this vector is between 1 and 0, and all the values in the vector sum to 1, so it can be used to represent the probability that the datapoint should be assigned to each class. The softmax function has the following formula for the ith position in the output vector:

$$\text{softmax}(\vec{z})_i = \frac{e_i^z}{\sum_{j=1}^{K} e_j^z}$$

  where $\vec{z}$ is the input vector of size K.

For the coding portion, ***PLEASE DO NOT MODIFY THE FUNCTION NAMES AND THE FILE NAMES OR THE AUTOGRADER WILL BREAK!!*** The autograder will run your program and test the code, and whatever score you see after the autograder finishes running is the score you receive for the coding portion.

2 PyTorch Tutorial

In this module, please open `q2.ipynb` on Google Colab (or your local machine if you already have PyTorch installed, but we recommend Google Colab). In this homework, we will teach you the fundamentals of PyTorch. The notebook will cover PyTorch's own NumPy class–`torch.tensor` and neural network modules. In the end, you will need to train your own model that reaches a 90% on the hidden dataset.

Your TODO's in this homework is to mainly implement your own training loop and a neural network with the architect of your own choice. Instructions are all included in the notebook. Below is some additional background information on optimizers and the softmax loss function, which will help understand some parts of the tutorial.

2.1 Optimizers

In gradient descent's most original form, we have:

$$\theta = \theta - \eta * \nabla_\theta L$$

In this equation, θ is the weight matrix to be optimized whereas L is the loss function and η is our learning rate. However, researchers have found many different modifications of this update rule in order to achieve better performance. In our homework (and a lot of other machine learning practices), we will be using Adam. This optimizer is so powerful it has been set as the default optimization method for deep learning applications, according to Stanford CS 231n course instructor Andrej Karpathy. That said, Adam is a combination of Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation Algorithm (RMSProp).

- AdaGrad Intuition: an algorithm that tends to assign higher learning rates η to infrequent features, which ensures that the parameter updates rely less on frequency and more on relevance.
- RMSProp Intuition: learning rates are adapted based on the average of recent magnitudes of the gradients for the weight.

For Adam, it takes the best of both worlds and updates its weight matrix using mechanisms from both AdaGrad and RMSProp. Optimization is its own art, and we will not be able to (nor would you need to) understand how all of it works in this class. The main takeaway is that there are many ways people have modified the gradient descent update rule, and some work better than others with certain tasks. All in all, the most common, default optimizer is Adam.

2.2 Softmax & Loss Function

You will notice that our loss function (`loss_fn` in the code) is set to `CrossEntropyLoss()`. In the homework 4 linear regression problem, we used mean squared error for our linear regression. Intuitively, this makes sense - we want to see how far our prediction is from the ground truth values. That was a regression problem, meaning that we are predicting a value in a continuous space.

Now, our task in this homework is to perform classification. We are trying to output a probability distribution for various classes. Intuitively, we want to have the model tell us something like... ”the probability of this data point being a shirt is 6%, 4% being a pair of pants, and 90% being a dress”, so we will output the class dress from the model.

Now, how do we assess whether our distribution is correct or no? The answer is cross entropy loss:

$$L = -\sum_{i=1}^{N}\sum_{c=1}^{K} y_{i,c}\log(\hat{y}_{i,c})$$

Here, $y_{i,c}$ is 1 if the data sample i belongs to class c and 0 otherwise, and $\hat{y}_{i,c}$ is the predicted probability that the model assigns to the correct class c. Intuitively, we can then consider that we are adding up the negative-log of the probabilities produced by the model. For example, if a sample point is in fact a dress, and the probability produced by the model is 0.01, then the model is penalized with a loss of $-\log(0.01) = 2$ (we do not care about the probabilities the model predicted for the other classes - only the ground-truth class). On the other hand, if the model is smart enough to assign the data point a 0.99 probability as a dress, the loss would be $-\log(0.99) = 0.0044$. As can be seen, the loss is high if the model has a low probability for the right class and the loss is low if the model has a high probability for the right class.

3 Submission

You will submit `q1.py`, `q2.ipynb`, `predictions.npy`, and `my_model.pt`.