CS/DS 541: Deep Learning

Homework 3

Tom Arnold & Nate Hindman

Due: 5:59pm ET Monday September 29

This problem can be done in teams of up 2 students.

Problem 1: Deriving He's initialization [10 points + 5 bonus pts]

In Lecture 7, we learned about a widely used technique for setting the initial values of the weight parameters in a neural network: the **He initialization**. Specifically, it samples each weight value from $\mathcal{N}(0, 2/n_l)$—i.e., a 0-mean Gaussian distribution with variance $2/n_l$ where n_l is the number of columns of the weight matrix $W^{[l]}$ layer l (or, equivalently the dimension of the inputs fed to layer l).

Here you are asked to derive some of the steps we skipped in class. We will start from:

$$Var[z^{[l]}] = Var[w^{[l]}x^{[l]}] = Var[\sum_{j=1}^{n_l} w_j^{[l]}x_j^{[l]}]$$

$$Var[\sum_{j=1}^{n_l} w_j^{[l]}x_j^{[l]}] = n_l Var[w_l x_l],$$

1. (2 points) Explain what are the assumptions that allow us to write:

$$Var[\sum_{j=1}^{n_l} w_j^{[l]}x_j^{[l]}] = n_l Var[w_l x_l],$$

where w_l represents any of the $w_j^{[l]}$ weights and x_l represents any of the $x_j^{[l]}$ inputs.

2. (4 points) Show that

$$Var[w_l x_l] = Var[w_l]\mathbb{E}[(x_l)^2]$$

using the following equality regarding the variance of the product of two mutually independent random variables A and B:

$$Var[AB] = Var[A]Var[B] + Var[A]\mathbb{E}[B]^2 + Var[B]\mathbb{E}[A]^2 \quad (0.0.1)$$

You will need to: (i) review and use the assumptions made about the distribution of w_l and (ii) use the relationship between variance and second moment for a random variable B:

$$Var[B] = \mathbb{E}[B^2] - \mathbb{E}[B]^2. \quad (0.0.2)$$

3. Last, you need to relate the variance of the input x_l with the variance of the pre-activation value z_{l-1} when the activation function is **ReLU**, i.e. when $x_l = ReLU(z_{l-1}) = ReLU(w_{l-1}x_{l-1} + b_{l-1})$. Specifically, you need to prove the relationship:

$$Var[z_{l-1}] = 2 \times \mathbb{E}[x_l^2].$$

To do so, you will need to use these assumptions:

- w_{l-1} has a **symmetric distribution around 0**, i.e., the density $f_{w_{l-1}}(w) = f_{w_{l-1}}(-w)$.
- $b_{l-1} = 0$

As a guide, try to answer these questions in order:

(a) (2 points) What is the expected value of z_{l-1}? Does it depend on the value of x_{l-1}?

(b) (1 point) Think of x_{l-1} as a constant. What is the density $f_{z_{l-1}}(z)$!

(c) (1 point) Is the distribution of z_{l-1} symmetric around 0?

(d) (BONUS: 3 points) The variance of a continuous random variable X is given by $\int_{-\infty}^{\infty}(x - \mathbb{E}[X])^2 f_X(x)dx$. Write an expression for $Var[z_{l-1}]$. Remember what you found in item (a) about $\mathbb{E}[z_{l-1}]$ and try to use the symmetry of the distribution to write $Var[z_{l-1}]$ only in terms of the positive values of z_{l-1}.

(e) (BONUS: 2 points) Last, starting from the definition of the second moment $\mathbb{E}[x_l^2] = \int_{-\infty}^{\infty} x_l^2 f_{x_l}(x)d_x$, find ways to replace x_l by z_{l-1} conditioning on whether z_{l-1} is negative or positive. You should be able to find the expression you derived in item (d) for $Var[z_{l-1}]$.

Problem 2: Training NNs with pytorch [15 points + 2 bonus pts]

In this problem you will use pytorch to train a multi-layer neural network to classify images of fashion items (10 different classes) from the **Fashion MNIST** dataset. Similarly to Homework 2, the input to the network will be a 28×28 pixel image; the output will be a real number.

Specifically, the starting network you will create should implement a function $f : \mathbb{R}^{784} \rightarrow \mathbb{R}^{10}$, where:

$$
\begin{aligned}
z^{(1)} &= W^{(1)}x + b^{(1)} \\
h^{(1)} &= ReLU(z^{(1)}) \\
z^{(2)} &= W^{(2)}h^{(1)} + b^{(2)} \\
&\vdots \\
z^{(l)} &= W^{(l)}h^{(l-1)} + b^{(l)} \\
\hat{y} &= softmax(z^{(l)})
\end{aligned}
$$

The network specified above is shown in the figure below: X $\rightarrow W^{(1)} \rightarrow b^{(1)} \rightarrow z^{(1)} \rightarrow h^{(1)} \rightarrow W^{(2)} \rightarrow \mathfrak{D}^{(2)} \rightarrow z^{(2)}\ h^{(2)} \rightarrow \ldots \rightarrow z^{(l)} \rightarrow W^{(l)}/\ b^{(l)} \rightarrow \hat{y}$

As usual, the (unregularized) cross-entropy cost function should be:

$$f_{CE}(W^{(1)}, b^{(1)}, ..., W^{(l)}, b^{(l)}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{10} y_k^{(i)} log \hat{y}_k^{(i)}$$

where n is the number of examples.

The Fashion MNIST dataset can be obtained from the following web links: https://s3.amazonaws.com/jrwprojects/fashion_mnist_train_images.npy https://s3.amazonaws.com/jrwprojects/fashion_mnist_train_labels.npy https://s3.amazonaws.com/jrwprojects/fashion_mnist_test_images.npy https://s3.amazonaws.com/jrwprojects/fashion_mnist_test_labels.npy

1. (6 points) Implement the same network using PyTorch or Tensorflow.
2. (5 points) Implement a method called **hyperparam_tuning**. Optimize the hyperparameters by training on the training set and selecting the parameter settings that optimize performance on the validation set. You should systematically (i.e., in code) try at least **10** (in total, not for each hyperparameter) different hyperparameter settings; **Hyperparameter tuning:** In this problem, there are several different hyperparameters that will impact the network's performance:

    - (Required) **Number of hidden layers** (suggestions: $\{3, 4, 5\}$)
    - **Number of units in each hidden layer** (suggestions: $\{30, 40, 50\}$)
    - **Learning rate** (suggestions: $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$)
    - **Minibatch size** (suggestions: $\{16, 32, 64, 128, 256\}$)
    - **Number of epochs**

- L_2 **Regularization strength** applied to the weight matrices (but not bias terms) In order not to "cheat" and thus overestimate the performance of the network it is crucial to optimize the hyperparameters **only on the validation set**; do **not** use the test set. (The training set would be ok, but typically leads to worse performance.) Hence, just like in Homework 2, you should fork off part of the training set into a validation portion.

3. (4 points) After you have optimized your hyperparameters, then re-train your network again and show a screenshot displaying the training loss evolving over the last **20 iterations** of SGD. (You can pick the last 20 mini-batches or last 20 epochs; it's up to you). In addition, make sure your screenshot also displays the **test accuracy** of the final trained classifier. The accuracy (percentage correctly classified test images) should be **at least 87%**.

4. (2 bonus points) Now you have to experiment with a variant of the previous network that includes either **BatchNorm, Dropout layers** or **different activation functions** (e.g., LeakyReLU, ELU or PReLU). Find an architecture that outperforms the one you found in item 3 and report the test accuracy.

Problem 3: Visualizing the loss landscape and optimization trajectories (15 points)

Visualize the SGD trajectory of your network when trained on Fashion MNIST:

1. Plot in 3-D the cross-entropy loss f_{CE} as a function of the neural network parameters (weights and bias terms). Rather than showing the loss as a function of any particular parameters (e.g., the third component of $b^{(2)}$), use the **x and y-axes** to span the two directions along which your parameters vary the most during SGD training i.e., you will need to use **principal component analysis (PCA)**. (In this assignment, you are free to use `sklearn.decomposition.PCA`.) The **z-axis** should represent the (unregularized) f_{CE} on training data. For each point (x, y) on a grid, compute f_{CE} and then interpolate between the points. (The interpolation and rendering are handled for you completely by the `plot_surface` function; see the starter code).

2. Superimpose a scatter plot of the different points (in the neural network's parameter space) that were reached during SGD (just sample a few times per epoch). To accelerate the rendering, train the network and plot the surface using just a small subset of the training data (e.g., 2500 examples) to estimate f_{CE}. See the starter code, in particular the `plotPath` function, for an example of 3-D plotting. Submit your graph in the PDF file and the code in the Python file.

Here is what I get when I superimpose two scatter plots corresponding to two different initializations and SGD runs (note that you only need to include a scatter plot for one run); as you can see, the two SGD trajectories descended into distinct local minima (though with similar cost):

Problem 4: Autoencoder (20 points)

This assignment explores one powerful concept in deep learning: **unsupervised representation learning with autoencoders**. You will build a neural network that learns to compress and then reconstruct images, forcing it to learn a meaningful representation of the data in the process.

Task 1: Data Preparation (2 point)

1. Load the **Fashion MNIST** dataset using `torchvision.datasets.FashionMNIST`.
2. Use `torchvision.transforms.ToTensor` to convert the images to PyTorch tensors and normalize the pixel values to be in the range $[0.0, 1.0]$.
3. Create a `DataLoader` for both the training and test sets. For this unsupervised task, we only need the image data, not the labels.

Task 2: Autoencoder Architecture (6 points)

Construct a **dense autoencoder model** by creating a class that inherits from `torch.nn.Module`. The model should consist of an **encoder** and a **decoder** with the following symmetric architecture:

- **An Encoder** that takes a flattened 784-dimensional input and maps it to a compressed representation.
  - Input layer (implicitly defined by the input shape).
  - Dense layer (`nn.Linear`) with **128 units** followed by a **ReLU** activation (`nn.ReLU`).
  - Dense layer with **64 units** followed by a **ReLU** activation. This layer's output is the **latent representation**, also known as the "**bottleneck**".
- **A Decoder** that takes the 64-dimensional latent representation and reconstructs the 784-dimensional image.
  - Dense layer with **128 units** followed by a **ReLU** activation.

– Dense layer with **784 units** followed by a **Sigmoid** activation (`nn.Sigmoid`). The sigmoid activation ensures the output values are in the range $[0.0, 1.0]$, matching the normalized input data.

Task 3: Model Training (6 points)

1. Instantiate your model, a loss function (`nn.MSELoss` is a good choice), and an optimizer (`torch.optim.Adam`).
2. Write a training loop that iterates for **20 epochs**. In each epoch, iterate through the training `DataLoader`.
3. For each batch of images, you must:
   - Flatten the 28×28 images into 784-dimensional vectors.
   - Perform a forward pass to get the reconstructed images.
   - Calculate the loss between the original and reconstructed images.
   - Zero the gradients, perform a backward pass, and update the weights.

Task 4: Visualizing Reconstructions (6 points)

1. Set your model to **evaluation mode**.
2. Get a batch of images from the test set and pass them through your trained autoencoder to get the reconstructions.
3. Create a plot showing **two different original images** from the test set and their corresponding reconstructions directly below them.
4. **Deliverable:** A single figure containing 4 images (2 original test images, 2 reconstructed images), clearly labeled.

Submission

Submit one PDF file that includes your notes for the theoretical problems (scanned or typed) and screenshots of your code for the programming problems. All material in the submitted PDF must be presented in a clear and readable format.

If you are working as part of a group, then indicate the members on canvas: https://canvas.wpi.edu/courses/76771/14853. Once you do that, be aware that any submission from a team member will overwrite an existing one.

Teamwork

You may complete this homework assignment either individually or in teams up to 2 people.