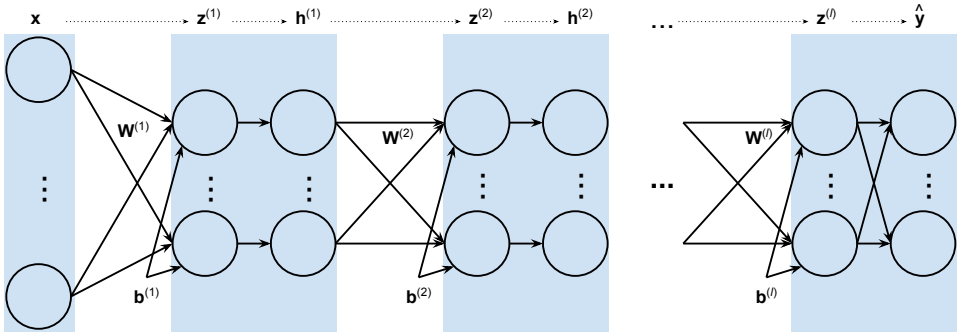
Homework 4 – Deep Neural Networks (CSDS/541, Whitehill, Spring 2021)

You may complete this homework assignment either individually or in teams up to 3 people.

1. **Feed-forward neural network** [55 points]: In this problem you will train a multi-layer neural network to classify images of fashion items (10 different classes) from the Fashion MNIST dataset. Similarly to Homework 3, the input to the network will be a 28×28-pixel image; the output will be a real number. Specifically, the network you create should implement a function $f : \mathbb{R}^{784} \rightarrow \mathbb{R}^{10}$, where:

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

The network specified above is shown in the figure below:



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

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

where n is the number of examples.

Hyperparameter tuning: In this problem, there are several different hyperparameters that will impact the network's performance:

- Number of hidden layers (suggestions: $\{3, 4, 5\}$) (**required**)
- 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 3, you should fork off part of the training set into a validation portion.

Your tasks:

(a) Implement stochastic gradient descent (SGD; see Section 5.9 and Algorithm 6.4 in the *Deep Learning* textbook) for the multi-layer neural network shown above. **Important**: your backprop algorithm must work for *any* number of hidden layers (not just 1 hidden layer).

(b) Verify that your gradient function is correct using the `check_grad` method.

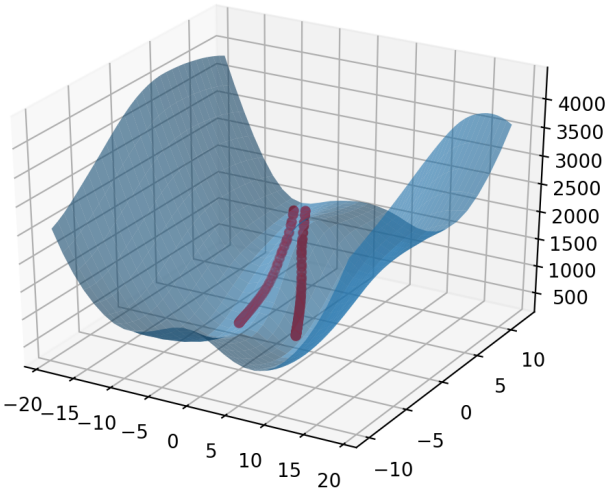
(c) Optimize the hyperparameters by training on the **training** set and selecting the parameter settings that optimize performance on the **validation** set. **You should *systematically* (i.e., in code) try at least 10 (in total, not for each hyperparameter) different hyperparameter settings**; accordingly, make sure there is a method called `findBestHyperparameters` . One of the hyperparameters that you vary systematically *must* be the number of layers; make sure to vary it across 3 different values (all of which must be at least 3).

(d) After you have optimized your hyperparameters, then run your trained network on the **test set** and report the accuracy (percent correctly classified images). **Include a screenshot** showing both these values during the last 20 epochs of SGD. **The accuracy (percentage correctly classified test images) should be at least** 87.5%**.**

2. **Mountains and valleys [20 points]**: Visualize the SGD trajectory of your network when trained on Fashion MNIST:

(a) 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 $\mathbf{b}^{(2)}$), 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.)

(b) 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):



In addition to your Python code (`homework4_WPIUSERNAME1.py` or `homework4_WPIUSERNAME1_WPIUSERNAME2.py` for teams), create a PDF file (`homework4_WPIUSERNAME1.pdf` or `homework4_WPIUSERNAME1_WPIUSERNAME2.pdf` for teams) containing the screenshots described above.