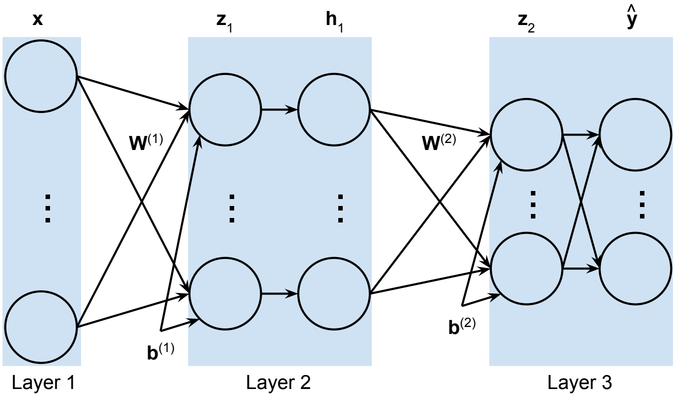
Homework 6 – Machine Learning (CS4342, Whitehill, Spring 2021)

1. **3-layer neural network [65 points]**: In this problem you will implement and train a 3-layer neural network to classify images of clothing items from the Fashion MNIST dataset. Similarly to Homework 3, the input to the network will be a 28 × 28-pixel image (converted into a 784-dimensional vector); the output will be a vector of 10 probabilities (one for each clothing type). Specifically, the network you create should implement a function $g : \mathbb{R}^{784} \rightarrow \mathbb{R}^{10}$, where:

$$\begin{aligned}
\mathbf{z}^{(1)} &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\
\mathbf{h}^{(1)} &= \mathrm{relu}(\mathbf{z}^{(1)})) \\
\mathbf{z}^{(2)} &= \mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)} \\
\hat{\mathbf{y}} = g(\mathbf{x}) &= \mathrm{softmax}(\mathbf{z}^{(2)})
\end{aligned}$$

Computing each of the intermediate outputs $\mathbf{z}^{(1)}$, $\mathbf{h}^{(1)}$, $\mathbf{z}^{(2)}$, and $\hat{\mathbf{y}}$ is known as *forward propagation* since it follows the direction of the edges in the directed graph shown below:



Loss function: For the Fashion MNIST dataset you should use the cross-entropy loss function:

$$f_{\mathrm{CE}}(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{10} \mathbf{y}_k^{(i)} \log \hat{\mathbf{y}}_k^{(i)}$$

where n is the number of examples.

Gradient descent: To train the neural network, you should use stochastic gradient descent (SGD). The hard part is computing the individual gradient terms. This can be done efficiently using *backward propagation* ("backprop"), which is called as such because it proceeds *opposite* the direction of the edges in the network graph above. You should start by initializing the weights randomly, and initializing the bias terms to small positive numbers. The reason is that – due to the ReLU activation function which has a gradient of 0 whenever its argument is less than 0 – we want to give enough bias to "encourage" the argument of ReLU to be positive. This is already performed for you in the starter code. Then, update the weights according to SGD using the gradient expressions shown below. (Note that these expressions are obtained by deriving and multiplying the Jacobian matrices as described in class, and then simplifying the result analytically.)

$$\begin{aligned}
\nabla_{\mathbf{W}^{(2)}} f_{\text{CE}} &= (\hat{\mathbf{y}} - \mathbf{y})\mathbf{h}^{(1)^\top} \\
\nabla_{\mathbf{b}^{(2)}} f_{\text{CE}} &= (\hat{\mathbf{y}} - \mathbf{y}) \\
\nabla_{\mathbf{W}^{(1)}} f_{\text{CE}} &= \mathbf{g}\mathbf{x}^\top \\
\nabla_{\mathbf{b}^{(1)}} f_{\text{CE}} &= \mathbf{g}
\end{aligned}$$

where column-vector $\mathbf{g}$ is defined so that

$$\mathbf{g}^\top = \left((\hat{\mathbf{y}} - \mathbf{y})^\top \mathbf{W}^{(2)}\right) \odot \text{relu}'(\mathbf{z}^{(1)^\top})$$

In the equation above, relu$'$ is the derivative of relu. Also, make sure that you follow the transposes exactly!

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

- Number of units in the 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
- Regularization strength

In order not to "cheat" – and thus overestimate the performance of the network – it is crucial to optimize the hyperparameters **only** on a **validation** set; do **not** use the test set. To create a validation set, simply randomly select and set aside 20% of the training examples and associated labels.

Your task: Use stochastic gradient descent to minimize the cross-entropy with respect to $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}$, and $\mathbf{b}^{(2)}$. Specifically:

(a) Implement stochastic gradient descent for the network shown above. [40 points]

(b) Implement the `pack` and `unpack` functions shown in the starter code. Use these to verify that your implemented cost and gradient functions are correct (the discrepancy should be less than 0.01) using a numerical derivative approximation – see the call to `check_grad` in the starter code. [10 points]

(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` (and please name it as such to help us during grading) [10 points]. **Include a screenshot** showing the progress and final output (selected hyperparameter values) of your hyperparameter optimization.

(d) After you have optimized your hyperparameters, then run your trained network on the **test set** and report (1) the cross-entropy and (2) the accuracy (percent correctly classified images). **Include a screenshot** showing both these values during the last 20 epochs of SGD. **The test accuracy (percentage correctly classified test images) should be at least** 87%**.** [5 points]

Datasets: You should use the following datasets:

- `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`

In addition to your Python code (`homework6_WPIUSERNAME.py`, create a PDF file (`homework6_WPIUSERNAME.pdf` containing the screenshots described above. **Please submit both the PDF and Python files in a single Zip file.**