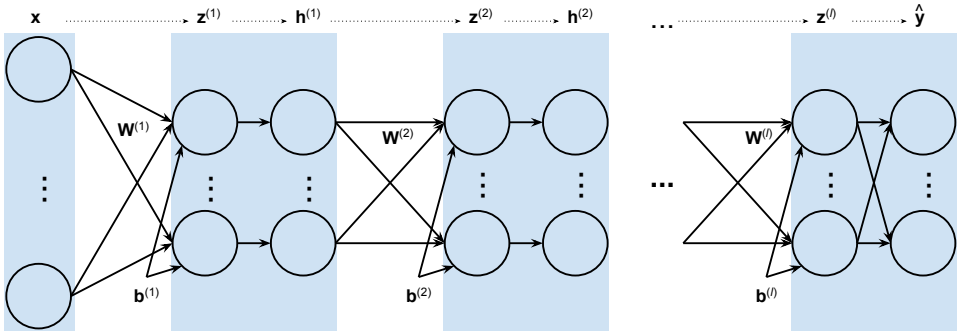
Homework 4 – Deep Neural Networks (CSDS/541, Whitehill, Spring 2023)

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

1. **Feed-forward neural network** [50 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:

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

   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 and architectural design decisions that will impact the network's performance:

   - 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)
   - Frequency & rate of learning rate decay
   - Variance & type of random noise added to training examples for data augmentation
   - ...

These can all have a big impact on the test accuracy. In contrast to previous assignments, there is no specific requirement for how to optimize them. However, in practice it will be necessary to do so in order to get good results.

Numerical gradient check: To make sure that you are implementing your gradient expressions correctly, you should use the `check_grad` (and possibly its sister function, `approx_fprime`). These methods take a function f (i.e., a Python method that computes a function; in practice, this will be the regularized cross-entropy function you code) as well as a set of points on which to compute f's derivative (some particular values for the weights and biases). The `approx_fprime` will return the numerical estimate of the gradient of f, evaluated at the points you provided. The `check_grad` also takes another parameter, ∇f, which is what you *claim* is the Python function that returns the gradient of the function f you passed in. `check_grad` computes the discrepancy (averaged over a set of points that you specified) between the numerical and analytical derivatives. Both of these methods require that *all* the parameters of the function (in practice: the weights and biases of your neural network) are "packed" into a single *vector* (even though the parameters actually constitute both matrices and vectors). For this reason, the starter code we provide includes a method called `unpack` that take a vector of numbers and extracts $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$, ..., as well as $\mathbf{b}^{(1)}$, $\mathbf{b}^{(2)}$, ... Note that the training data and training labels are *not* parameters of the function f whose gradient you are computing/estimating, even though they are obviously needed by the cross-entropy function to do its job. For this reason, we "wrap" the call to `fCE` with a Python `lambda` expression in the starter code.

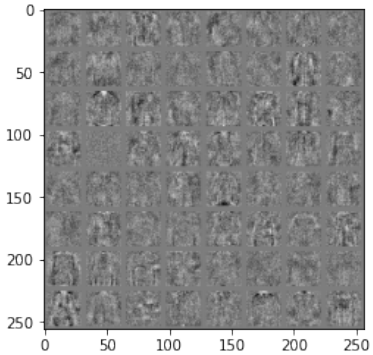
Your tasks:

(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.

(b) Verify that your gradient function is correct using the `check_grad` method. In particular, include in your PDF the real-valued output of the call to `check_grad` for a neural network with 3 hidden layers, each with 64 neurons (the discrepancy between the numerical and analytical derivative for this case should be less than 1e-4).

(c) Include a screenshot in your submitted PDF file showing multiple iterations of SGD (just to show that you actually ran your code successfully). For each iteration, report both the test accuracy and test unregularized cross-entropy. For full credit, the accuracy (percentage correctly classified test images) should be at least 88%.

(d) Visualize the first layer of weights $\mathbf{W}^{(1)}$ that are learned after training your best neural network. In particular, reshape each row of the weights matrix into a 28×28 matrix, and then create a "grid" of such images. Include this figure in your PDF. Here is an example (for $\mathbf{W}^{(0)} \in \mathbb{R}^{64 \times 784}$).



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.