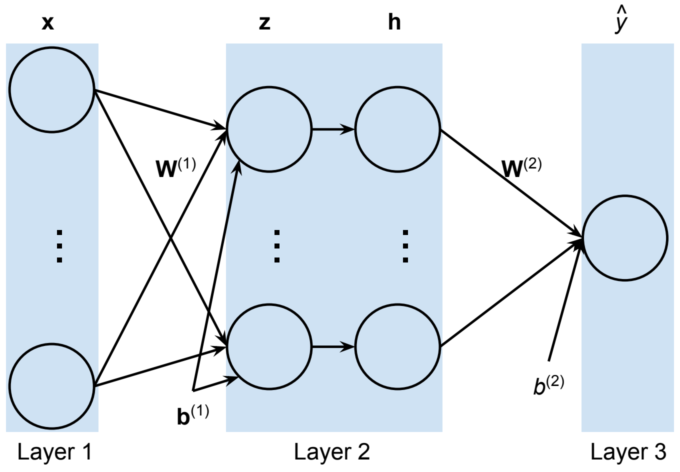
Homework 5 – Machine Learning (CS4342, Whitehill, Spring 2022)

You may do this homework either by yourself or in a team of 4 people.

1. **3-layer neural network for facial age estimation [70 points]**: In this problem you will implement and train a 3-layer neural network to estimate the age of a face image. Similarly to Homework 2, the input to the network will be a 48×48-pixel image (converted into a 2304-dimensional vector); the output will be a scalar. Specifically, the network you create should implement a function $g : \mathbb{R}^{2304} \to \mathbb{R}^1$, where:

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

Computing each of the intermediate outputs $\mathbf{z}$, $\mathbf{h}$, 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: Since this is a regression problem, it makes sense to use the (half-)MSE loss:

$$f_{\text{MSE}}(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, b^{(2)}) = \frac{1}{2n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

where n is the number of examples. You may also choose to regularize it by adding terms to penalize the sum of squared elements in either the $\mathbf{W}^{(1)}$ or $\mathbf{W}^{(2)}$ weight matrices, but this is up to you.

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 and biases randomly, except that $b^{(2)}$ can benefit slightly from being initialized to the average value of the training labels (i.e., average age value of the training set). (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.)

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

where column-vector $\mathbf{g}$ is defined so that its transpose (i.e., a row vector) is defined as follows:

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

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: 5,10,20,30)
- Learning rate (suggestions: 1e-4, 1e-5, 1e-6). Note that it is useful to **anneal** the learning rate over epochs, e.g., use 1e-4 for the first 20 epochs, then 1e-5 for the next 100 epochs, and then 1e-6 for 100 more epochs.
- Minibatch size (suggestions: 32, 64, 128, 256)
- Number of epochs (suggestions: 50, 100, 250, 500).
- Regularization strength (suggestions: 0, 1e-3).

Your tasks:

(a) Implement stochastic gradient descent for the network shown above to minimize the (half-)MSE with respect to $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}$, and $b^{(2)}$. You should use the same datasets as in Homework 2. Before trying to compute the gradients for a mini-batch of size > 1, first make sure you can do so for a mini-batch of size 1 (see below). [**40 points**]

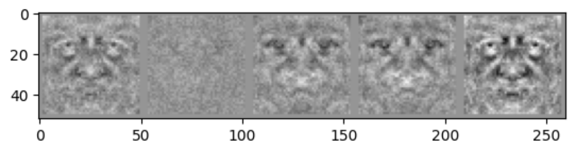
(b) Verify that your implemented loss and gradient functions are correct (the discrepancy should be less than 1e-5) by comparing to the `correctGrad*.npy` files that are provided; for this comparison, set `NUM_HIDDEN=20`. **Note**: this check is only for a mini-batch of size 1. It is possible that your gradient works correctly for a mini-batch size of 1 but fails for sizes > 1. [**10 points**]

(c) Through a combination of architecture exploration, data augmentation, and hyperparameter tuning, train your network so that the accuracy on the "testing" dataset is as small as possible. To get full credit (10 out of 10 points), it should be < 82 half-MSE. If you get ≥ 82 but < 85 half-MSE, then you will get 8 points. If you get ≥ 85 but < 100 half-MSE, then you will get 4 points. Otherwise, you will get 0 points for this task.

Include a screenshot in your submitted PDF showing the training and testing loss values (show the last 20 iterations) for the best architecture/hyperparameters that you found.

Note: since you will likely will be repeatedly trying different neural networks and training conditions and adaptively applying them to the same test set, **it is very likely that you will be overfitting to the test data; hence, your final accuracy is not a reliable estimate of how well your machine would do on novel data. [10 points]**

(d) Create a figure (see `show_weight_vectors()` in the starter code) showing the different weight vectors you obtained for the first layer of your neural network (i.e., the rows of $\mathbf{W}^{(1)}$ rendered as 48×48 images). Submit this figure in your PDF. An example of 5 weight vectors is shown below. [**10 points**]



In addition to your Python code (`homework5_WPIUSERNAMES.py`, create a PDF file (`homework5_WPIUSERNAMES.pdf` containing the screenshots described above. **Please submit both the PDF and Python files in a single Zip file.** If you are working as part of a team, then make sure you register as one of the pre-allocated teams on Canvas; only one person should then submit the Zip file (and you will both receive credit).