Designing a fully connected neural network for housing data

The validation loss results for the first model (with only one hidden) was much higher than the model that we tested last time (the simple linear model). This is partly due to the added complexity of the fully connected model, which seems like it was designed to capture non-linear relationships in the housing dataset, where none really exist. As we have seen before, the housing dataset is relatively linear, and our simple gradient descent algorithm from the beginning of the semester has given us the most accurate model. This added complexity (**449** parameters compared to **13**) just increases our training time and actually decreases the accuracy. Shown below is the validation loss for an average training run of the 449 parameter model over 1000 epochs. The drastic difference between the training loss and the validation loss suggests that the model is suffering from overfitting. The model has essentially memorized the training data and isn't good at generalizing to other datasets.

```
Epoch 0, Loss: 0.009218944236636162, Validation Loss: 0.48150575160980225
Epoch 200, Loss: 0.003173336386680603, Validation Loss: 0.39438721537590027
Epoch 400, Loss: 0.0033827542793005705, Validation Loss: 0.37428173422813416
Epoch 600, Loss: 0.0019013026030734181, Validation Loss: 0.38602733612060547
Epoch 800, Loss: 0.0019346495391801, Validation Loss: 0.37936943769454956
Epoch 1000, Loss: 0.0020974143408238888, Validation Loss: 0.3704174757003784
```

For reference, these are the losses over 250 epochs for our linear model (defined in the previous homework).

```
Epoch 0, Loss: 0.10799746215343475, Validation Loss: 0.625154435634613
Epoch 50, Loss: 0.015391075052320957, Validation Loss: 0.3070577383041382
Epoch 100, Loss: 0.012124455533921719, Validation Loss: 0.2979496419429779
Epoch 150, Loss: 0.018195969983935356, Validation Loss: 0.2907794713973999
Epoch 200, Loss: 0.011080420576035976, Validation Loss: 0.29371151328086853
Epoch 250, Loss: 0.008752954192459583, Validation Loss: 0.2952178418636322
```

Maybe a more complicated model will help?

Shown below are the training and validation losses over 1000 epochs for our redesigned model, which now has **3585** parameters.

```
Epoch 0, Loss: 0.01825198158621788, Validation Loss: 0.31872621178627014
Epoch 100, Loss: 0.0016366925556212664, Validation Loss: 0.3743986189365387
Epoch 200, Loss: 0.0010305375326424837, Validation Loss: 0.33411169052124023
Epoch 300, Loss: 0.0003328001475892961, Validation Loss: 0.35629940032958984
Epoch 400, Loss: 0.00040134237497113645, Validation Loss: 0.3693268299102783
Epoch 500, Loss: 0.0007007385138422251, Validation Loss: 0.3330237865447998
Epoch 600, Loss: 0.000313433789415285, Validation Loss: 0.35649192333221436
Epoch 700, Loss: 0.0009206163231283426, Validation Loss: 0.3341801166534424
Epoch 800, Loss: 0.00013210176257416606, Validation Loss: 0.37515789270401
Epoch 900, Loss: 0.00018687245028559119, Validation Loss: 0.3516468107700348
Epoch 1000, Loss: 0.00017046900757122785, Validation Loss: 0.36044222116470337
```

With this added change, we can see clear signs of overfitting. The training loss is extremely low, while the validation loss still remains significantly higher than our original linear model. Adding to the model complexity for such a simple dataset did not improve our training time or accuracy.

The CIFAR-10 dataset

A fully connected neural network is a very poor choice for image recognition and classification. Because the input layer needs to be flattened in a FCN (due to the fact that you need to map a neuron to each input unit), you lose a lot of spatial information (distance from pixels and patterns) that are easily visible in a regular image. This approach also increases the amount of parameters necessary to train. A 32x32x3 image (3 channels for R,G,B) is already **3072** parameters. I would not expect any model we train in this section to have a high accuracy, at least compared to a convolutional neural network.

```
                    Epoch [1/10], Loss: 3.8973
                    Epoch [2/10], Loss: 2.9538
                    Epoch [3/10], Loss: 3.8309
                    Epoch [4/10], Loss: 2.8786
                    Epoch [5/10], Loss: 3.2471
                    Epoch [6/10], Loss: 3.3721
                    Epoch [7/10], Loss: 3.0107
                    Epoch [8/10], Loss: 4.1322
                    Epoch [9/10], Loss: 3.9115
                   Epoch [10/10], Loss: 4.4712
  took 41.66 seconds to train, and final model accuracy is 37.62%
```

Predictably, the first model (with only one hidden layer) does very poorly, ending with a final accuracy of **37.16%**.

One way to improve the performance of a FCN is to add more hidden layers. In this case, I chose to add layers in a pyramid form (512 neurons -> 256 -> 128 -> …) so that the features could trickle down from layer to layer. This, however had an unintended effect of halving our final model accuracy.

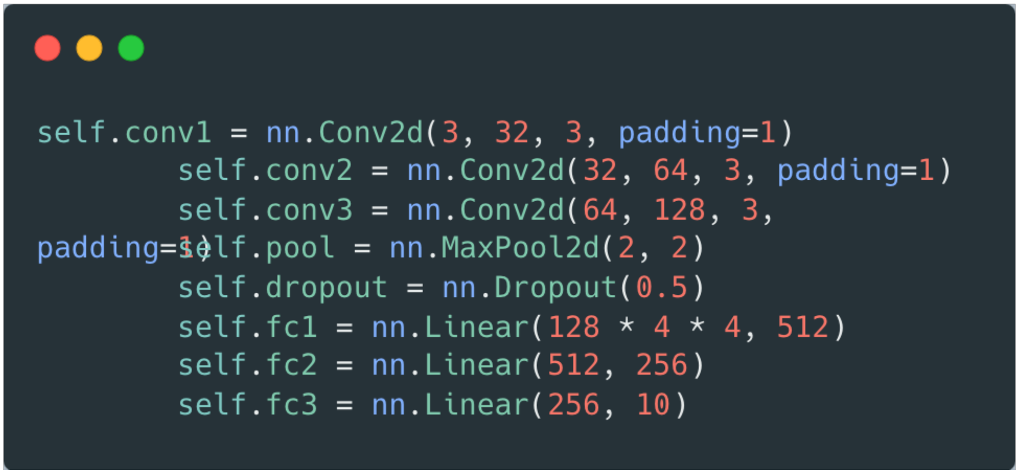
```
                    Epoch [1/20], Loss: 1.7459
                    Epoch [2/20], Loss: 1.7046
```

```
Epoch [3/20], Loss: 1.6538
Epoch [4/20], Loss: 1.6954
Epoch [5/20], Loss: 1.8152
Epoch [6/20], Loss: 1.7225
Epoch [7/20], Loss: 1.8525
Epoch [8/20], Loss: 1.8976
Epoch [9/20], Loss: 1.9167
Epoch [10/20], Loss: 1.8968
Epoch [11/20], Loss: 1.7799
Epoch [12/20], Loss: 1.9171
Epoch [13/20], Loss: 1.8737
Epoch [14/20], Loss: 1.9964
Epoch [15/20], Loss: 1.9948
Epoch [16/20], Loss: 2.0537
Epoch [17/20], Loss: 1.8573
Epoch [18/20], Loss: 2.0651
Epoch [19/20], Loss: 2.1287
Epoch [20/20], Loss: 2.1261
took 129.41 seconds to train, and final model accuracy is 18.99%
```

This model, with even more parameters, seems to do even worse than the original model. This could be due to overfitting, suboptimal hyper-parameter selection, or a range of other reasons. This combination of low training loss and high validation (test data) loss is a very strong indicator of our model overfitting.

Using a CNN

As a bonus, I decided to implement a CNN to try and increase the accuracy. The architecture I settled on is shown below.



```
self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, 3,
padding=1)self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.5)
        self.fc1 = nn.Linear(128 * 4 * 4, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 10)
```

Below are the results of training this model for 20 epochs.

```
Epoch [1/20], Loss: 1.5823
Epoch [2/20], Loss: 1.2962
Epoch [3/20], Loss: 1.1798
Epoch [4/20], Loss: 0.9053
Epoch [5/20], Loss: 0.9060
Epoch [6/20], Loss: 0.6863
Epoch [7/20], Loss: 0.7418
Epoch [8/20], Loss: 0.6138
Epoch [9/20], Loss: 0.7538
Epoch [10/20], Loss: 0.6263
Epoch [11/20], Loss: 0.7923
Epoch [12/20], Loss: 0.4616
Epoch [13/20], Loss: 0.5316
Epoch [14/20], Loss: 0.5991
Epoch [15/20], Loss: 0.6361
Epoch [16/20], Loss: 0.6423
Epoch [17/20], Loss: 0.3394
Epoch [18/20], Loss: 0.6074
Epoch [19/20], Loss: 0.4154
Epoch [20/20], Loss: 0.5483
```

took **164.86** seconds to train, and final model accuracy is **80.13%**

GitHub link: https://github.com/Anu78/intro-to-ml-hw