Appendix 2

Our process

After a brief introduction in the lecture about deep neural networks and how they work, we started working on the code to try and improve it. The existing code was a chatbot and our job was to improve the accuracy. As mentioned the lecture only gave a short brief of the task and that made the process a little confusing. Luckily our group had a few members with some skills in python so we managed to achieve some results.

The Outcome

- torch.sigmoid(x) gives a better result than F.sigmoid(x) at the second layer of 3. 0.057 -> 0.0548. It's better, but still barely noticeable.
- The amount of neurons gives a much better result. By just having two layers and increasing the amount of neurons (to i.e 8192), the margin of error will be reduced to 0.03, even with just 300 steps.
- More layers increases the time of how long the training takes rather drastically. Especially with many neurons in each layer. It gives better results with for instance three layers at 0.0548 and two layers at 0.072.

Three layers with 256 neurons per layer and 3000 steps:

```
0.08595419
0.08544537
0.08494359
0.08444873
0.08396063
ready
Chatbot:What's to discuss?
Human:
```

Four layers with 256 neurons per layer and 3000 steps:

```
0.09574244
0.095125824
0.09451793
ready
Chatbot:Combination.  I don't know -- I thought he'd be different.  More of a gentleman...
Human:
```

Five layers with 256 neurons per layer and 3000 steps:

```
0.08650594
0.08599483
0.08549085
0.08499383
ready
Chatbot:Daddy, people expect me to be there!
Human:
```

Six layers with 256 neurons per layer and 3000 steps:

```
0.0878881
0.08735979
0.08683884
0.08632513
0.08581859
0.085319005
0.08482631
ready
Chatbot:Well, I surely know what a quadrant is!  But I've never seen it used at night before.
Human:
```

The code for variable 1 (we can assume there is some errors):

```
class Net1(nn.Module):
    def __init__(self):
        super(Net1, self).__init__()
        self.fc1 = nn.Linear(6,256)
        self.fc2 = nn.Linear(256,256)
        self.fc3 = nn.Linear(256, 256)
        self.fc4 = nn.Linear(256, 256)
        self.fc5 = nn.Linear(256, num_classes)
        self.fc6 = nn.Linear(256, num_classes)

    def forward(self,x):
        x = self.fc1(x)
        x = F.sigmoid(x)
        x = self.fc2(x)
        x = torch.sigmoid(x)
        x = self.fc3(x)
        x = torch.sigmoid(x)
        x = self.fc4(x)
        x = torch.sigmoid(x)
        x = self.fc5(x)
        x = torch.sigmoid(x)
        x = self.fc6(x)
        x = torch.sigmoid(x)
        return x
```

We have an error on the screenshot, the "num_classes" in self.fc5 needs to be changed to 256 in this scenario. As we discover later, we should incrementally increase the numbers for each layer instead of having the same input and output in all of them.

4096 neurons at two layers and 3000 steps:

```
0.03155896
0.03154565
0.031532586
ready
Chatbot:Dorsey can plow whoever he wants. I'm just in this for the cash.
Human:
```

4096 neurons at two layers and 1000 steps:

```
0.03596696
0.035771225
0.035587426
0.03541455
ready
Chatbot:Who cares?
Human:
```

1024 neurons at two layers and 1000 steps:

```
0.065226495
0.06409313
0.06302009
0.062003057
ready
Chatbot:Thank you.
Human:
```

1024 neurons at three layers and 1000 steps (higher??):

```
0.07496922
0.07362272
0.07234032
ready
Chatbot:I hate peas.
Human:
```

1024 neurons at six layers and 1000 steps (takes 10 minutes to run):

```
0.07762786
0.07618591
0.07481326
0.073505536
0.07225882
ready
Chatbot:And were you never ambitious, Excellency?  Or is ambition only a virtue among the nobles, a fault for the rest of us?
Human:
```

At this point we realized that our code was wrong and needed increasing/decreasing numbers in and out of each layer, as there's no point processing the same data over and over with the same in and outs, as they will come to the same conclusion.

A change done to the layers to see if that changes anything to the results:

```
class Net1(nn.Module):
    def __init__(self):
        super(Net1, self).__init__()
        self.fc1 = nn.Linear(6, 256)
        self.fc2 = nn.Linear(256,512)
        self.fc3 = nn.Linear(512, 256)
        self.fc4 = nn.Linear(256, 512)
        self.fc5 = nn.Linear(512, 1024)
        self.fc6 = nn.Linear(1024, num_classes)
```

```
0.075566925
0.07424398
0.0729828
ready
Chatbot:I know, just let me sleep
Human:
```

About the same. Just decreasing layers, from 4096 to 256:

```
0.16341121
0.16166106
0.15993927
ready
Chatbot:In the kitchen.
Human:
```

Much lower, just increasing layers from 256 to 4096:

```
0.038595255
0.038313773
0.038048908
ready
Chatbot:But it's Gigglepuss - I know you like them.  I saw you there.
Human:
```

By increasing the numbers for each layer, so the first layer has 6 inputs and 256 outputs, the second layer has 256 in, 512 outs etc, we got better results.

4 layers, layer 1: 6-12, 2: 12-24, 3: 24-48, 4: 48-num_classes (30):

```
0.23496123
0.23441188
0.23386371
0.23331706
ready
Chatbot:The water's going putrid in the barrels.
Human:
```

5 layers, same increments of increase, double outs of each layer, for a total of 96:

```
0.22003067
0.21903333
0.2180416
ready
Chatbot:Other than my upchuck reflex? Nothing.
Human:
```

6 layers, same increments, 192 out:

```
0.16925271
0.16789375
0.16655105
ready
Chatbot:What do you wanna watch?  We've got crap, crap, crap or crap
Human:
```

To compare 2 layers with the same amount of neurons with 6 layers:

```
0.16482759
0.16295268
0.16111124
ready
Chatbot:We lost cousins, friends.  We will wash this in blood.
Human:
```

Almost identical, which is confusing.

Reflections

We experienced this task as a little confusing because we had little previous knowledge around the topic. We had to spend some time going through trial and error to get the results. This process was quite time consuming and we had to spend quite a lot of time doing independent research around the different aspects of the given code. It is however very interesting to get some insight in machine learning and deep neural networks to see how they work. This is a very interesting topic and we enjoyed working with it to the degree that we understood what we were doing.

In addition most of the experimenting with numbers came under the #Variant1 part of the code, as Variant 2 never really triggered, hence the group didn't see the point in conducting the same experiments there.

We had a few interesting finds. Having more than two layers was pointless if the different layers had the same in and outs, or decreasing in and outs. With the same numbers the next layer would just process the exact same data and get the same outcome, for an increased cost of time. This led us to increase the numbers of each layer, which provided a better result, however the group struggled to see the point of several layers, when the last few experiments showed that 2 layers with the same amount of neurons as the 6th layered incremental increase, gave a better result. As an explanation this could be where Variant 2 of the code triggered, which the group did not really check. Another explanation would be that the group did something wrong when creating new layers, as more layers didn't give a better or "smarter" AI.