Hints on implementing your neural network

- Implement layers and activations as objects if you are comfortable with OOP

```
class FullyConnectedLayer:
    def __init__(self, num_input, num_output, lr=1e-3, scale=2):
        # your code
        pass
    def forward(self, input_data):
        # your code
        return None
    def backward(self, gradient_data):
        # your code
        return None
```

Hints on implementing your neural network

- Isolate layer definition, network definition, and training codes.

```
network_1 = [
        FullyConnectedLayer(input_num, 100, lr=lr),
        ReLULayer(),
        FullyConnectedLayer(100, 40, lr=lr),
        ReLULayer(),
        FullyConnectedLayer(40, output_num, lr=lr),
        SoftmaxOutput_CrossEntropyLossLayer()
    ]
```

```
def network_forward(network, input_data, label_data=None, phase='train'):
    for layer in network:
        if type(layer) is not SoftmaxOutput_CrossEntropyLossLayer:
            input_data = layer.forward(input_data)
        else:
            layer.eval(input_data, label_data, phase)
    return network
```

```
def network_backward(network):
    for layer in reversed(network):
        if type(layer) is SoftmaxOutput_CrossEntropyLossLayer:
            gradient = layer.backward()
        else:
            gradient = layer.backward(gradient)
    return network
```

Useful links

- https://chrischoy.github.io/research/nn-gradient/
- https://deepnotes.io/softmax-crossentropy