

# Homework 6

```
In [1]: import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
```


```
In [3]: np.random.seed(0)
in1_numpy = np.random.randint(256, size=(3,3))
in1 = torch.tensor(in1_numpy)

training_data_numpy = np.random.randint(256, size=(99, 3, 3))
training_data = torch.tensor(training_data_numpy, dtype=torch.float)

in1_flattened = torch.reshape(in1, (1, 9))
training_data_flattened = torch.reshape(training_data, (-1, 9))
training_data_flattened = torch.cat((in1_flattened, training_data_flattened), dim=0)

picture_labels_numpy = np.random.randint(4, size=(100))
picture_labels = torch.tensor(picture_labels_numpy, dtype=torch.long)
```

Now that you have seen how a fully connected neural network works in last week's homework we are going to convert everything into PyTorch. When building actual neural networks in PyTorch we are going to use a class to represent the network. We will train an instance of this class for our model.

 Task 1: Create a class that will represent your neural network structure. You may add any layers you want to this, but keep in mind that between linear layers you are going to want a ReLu. Create the `__init__()` and `forward()` methods for the class.

The list of layers PyTorch has available can be easily found on Google. The syntax for using them in your constructor is `self.layerX = torch.nn.LayerType(Parameters)`.

Hint: Fully Connected Layers are referred to as linear layers in PyTorch.

Recall: What were the dimensions of the weight matrices in last week's assignment?

```
In [23]: #####
## YOUR CODE STARTS HERE
#####

class Animal_Classifier(nn.Module):
    def __init__(self):
        super(Animal_Classifier, self).__init__()
        # put your linear layers here

        self.layer1 = nn.Linear(9, 128)
        self.layer2 = nn.Linear(128, 64)
        self.output_layer = nn.Linear(64, 4)

    def forward(self, x):
        # Define the forward pass
        x = F.relu(self.layer1(x)) # Apply ReLU activation function after first linear
        x = F.relu(self.layer2(x)) # Apply ReLU activation function after second linear
        x = self.output_layer(x)   # No activation function here, will be applied in the
        return x

model = Animal_Classifier()
```

```
#####  
## YOUR CODE ENDS HERE  
#####
```



Task 2: Use the following training loop to train your model. Don't worry about splitting this model into training and testing data, these are just random values so they should not actually train to anything relevant.

In [24]:

```
num_epochs = 10  
learning_rate = 0.001  
momentum = 1  
num_epochs_to_print = 1  
  
criterion = nn.CrossEntropyLoss()  
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=momentum)  
  
for i in range(num_epochs):  
  
    optimizer.zero_grad()  
  
    outputs = model(training_data_flattened)  
    loss = criterion(outputs, picture_labels)  
    loss.backward()  
    optimizer.step()  
  
    if i % num_epochs_to_print == 0:  
        print(loss.item())
```

```
12.060713768005371  
18.172605514526367  
36.544612884521484  
51.59947204589844  
14.004297256469727  
3.266611099243164  
1.7757105827331543  
1.5319589376449585  
1.6709080934524536  
1.614730954170227
```

## Write Your Report Here

Introduction: We developed a neural network using PyTorch to classify 3x3 pixel images into four categories. The process involved preparing the data, designing the network architecture, and implementing it in PyTorch.

Data Preparation: Generated random 3x3 pixel images and labels using NumPy. Converted images and labels to PyTorch tensors, flattened the images for neural network input, and concatenated all data into a single tensor.

Neural Network Design: Created a `Animal_Classifier` class with an input layer for 9-pixel data, two hidden layers with ReLU activations (128 and 64 neurons), and an output layer producing four scores for classification.

Implementation: Defined the network structure in the `init` method and data flow in the forward method. Instantiated the `Animal_Classifier` model, ready for training with the prepared data.

Conclusion: Implemented a basic neural network in PyTorch for classifying small pixel images, demonstrating the use of linear layers, ReLU activations, and data preparation in PyTorch.

