FIT3181/5215 Deep Learning

Tutorial 02: Feed-forward Neural Networks

Lecturer: Dr Trung Le, A/Prof Zongyuan Ge, Dr Arghya Pal

Head Tutors: Dr Ruda Nie and Leila Mahmoodi

Email: trunglm@monash.edu

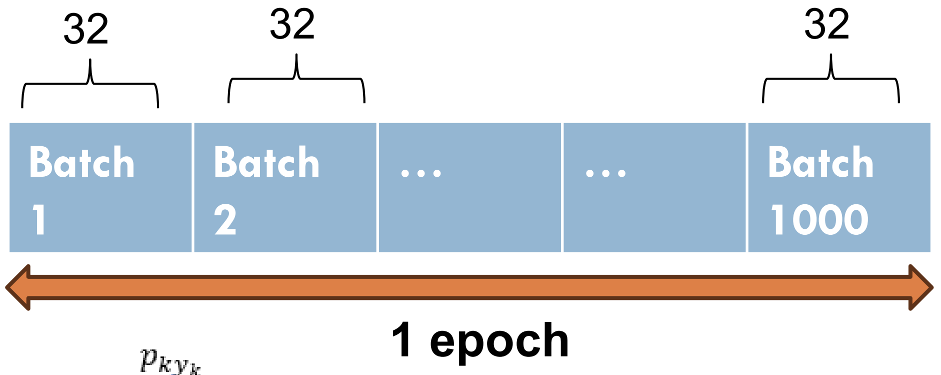
Department of Data Science and AI
Faculty of Information Technology, Monash University, Australia

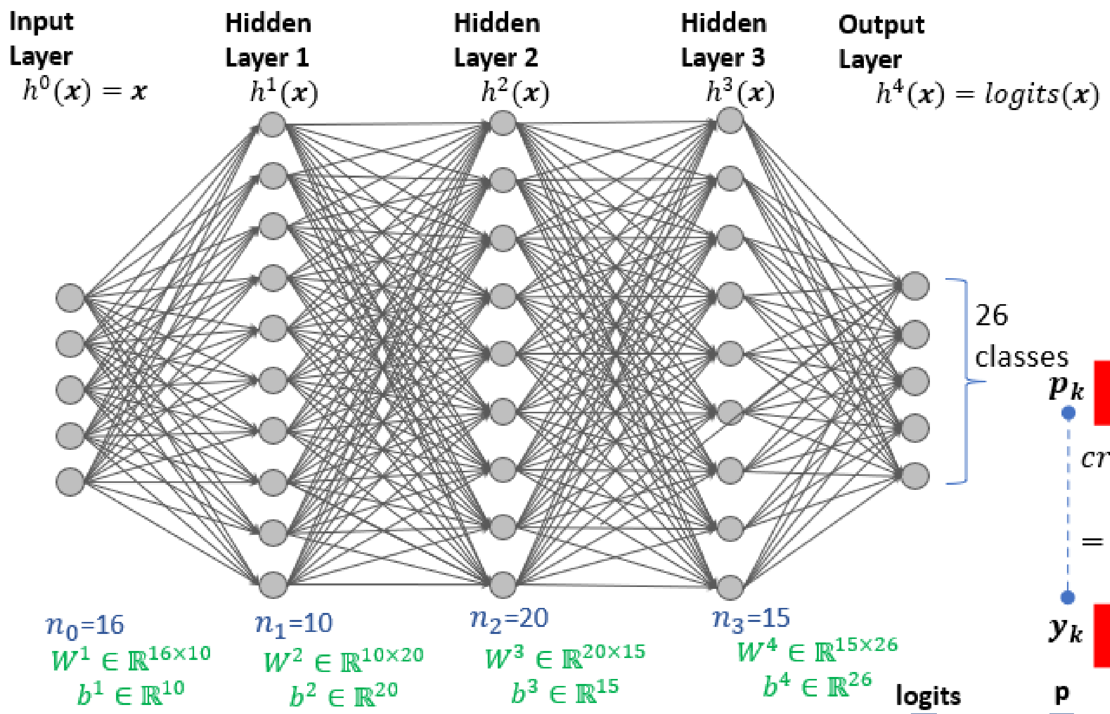
Introduction

- The **purpose** of this tutorial is to practise:
  1. Understand the forward propagation
  2. How to implement feed-forward NNs in PyTorch

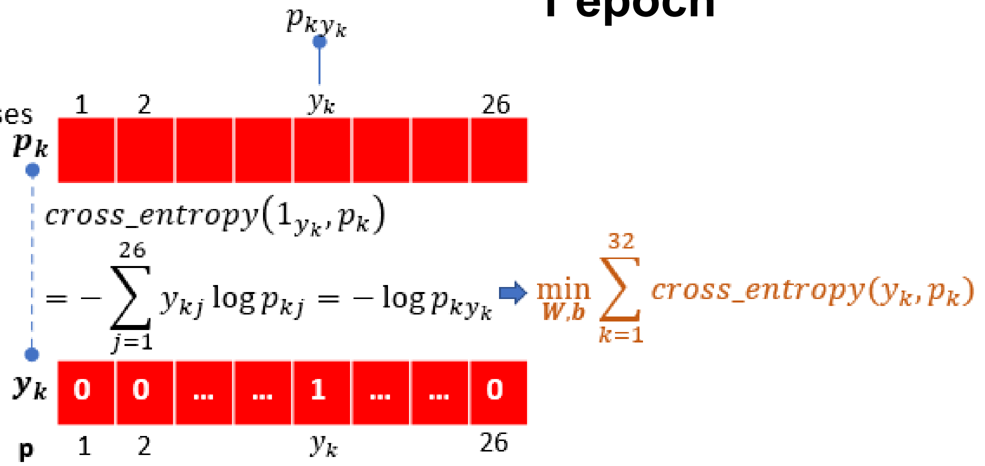
Mini-batch Forward and Loss

Shuffling training set and splitting to many equal batches



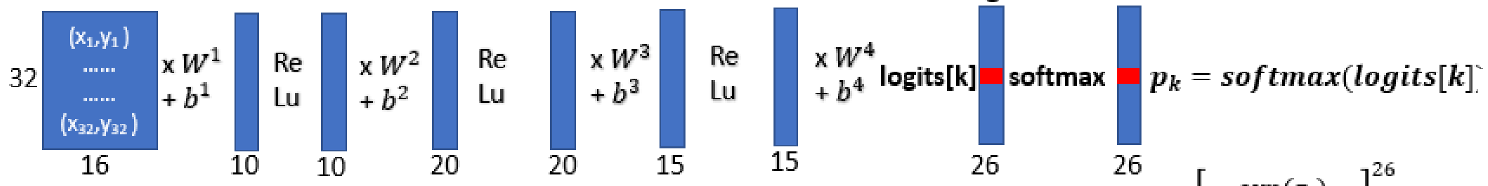


Input Layer
$h^0(x) = x$
Hidden Layer 1
$h^1(x)$
Hidden Layer 2
$h^2(x)$
Hidden Layer 3
$h^3(x)$
Output Layer
$h^4(x) = logits(x)$
26 classes
n_0=16
$W^1 \in \mathbb{R}^{16\times10}$
$b^1 \in \mathbb{R}^{10}$
n_1=10
$W^2 \in \mathbb{R}^{10\times20}$
$b^2 \in \mathbb{R}^{20}$
n_2=20
$W^3 \in \mathbb{R}^{20\times15}$
$b^3 \in \mathbb{R}^{15}$
n_3=15
$W^4 \in \mathbb{R}^{15\times26}$
$b^4 \in \mathbb{R}^{26}$



$cross\_entropy(1_{y_k}, p_k)$

$$= -\sum_{j=1}^{26} y_{kj}\log p_{kj} = -\log p_{ky_k} \Rightarrow \min_{W,b} \sum_{k=1}^{32} cross\_entropy(y_k, p_k)$$



$p_k = softmax(logits[k])$

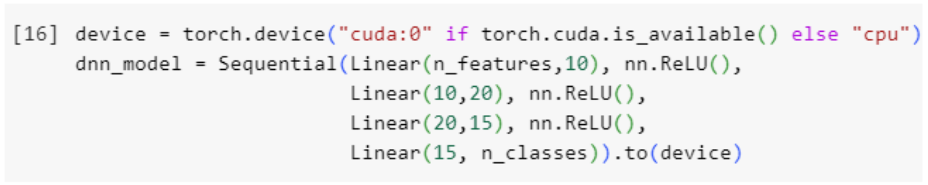
$$softmax(\mathbf{z}) = \left[\frac{\exp(z_i)}{\sum_{j=1}^{26}\exp(z_j)}\right]_{i=1}^{26}$$

- Input X_b with mini-batch size of 32
- $h_1 = ReLu(X_b \times W^1 + b^1) \in \mathbb{R}^{32\times10}$
- $h_2 = ReLu(h_1 \times W^2 + b^2 \in \mathbb{R}^{32\times20}$.
- $h_3 = ReLu(h_2 \times W^3 + b^3 \in \mathbb{R}^{32\times15}$.
- $logits = h_3 \times W^4 + b^4 \in \mathbb{R}^{32\times26}$
- $p = softmax(logits) \in \mathbb{R}^{32\times26}$

- Optimizer uses batch loss to update model parameters
  - $\theta = [W^l, b^l]_{l=1}^{L}$

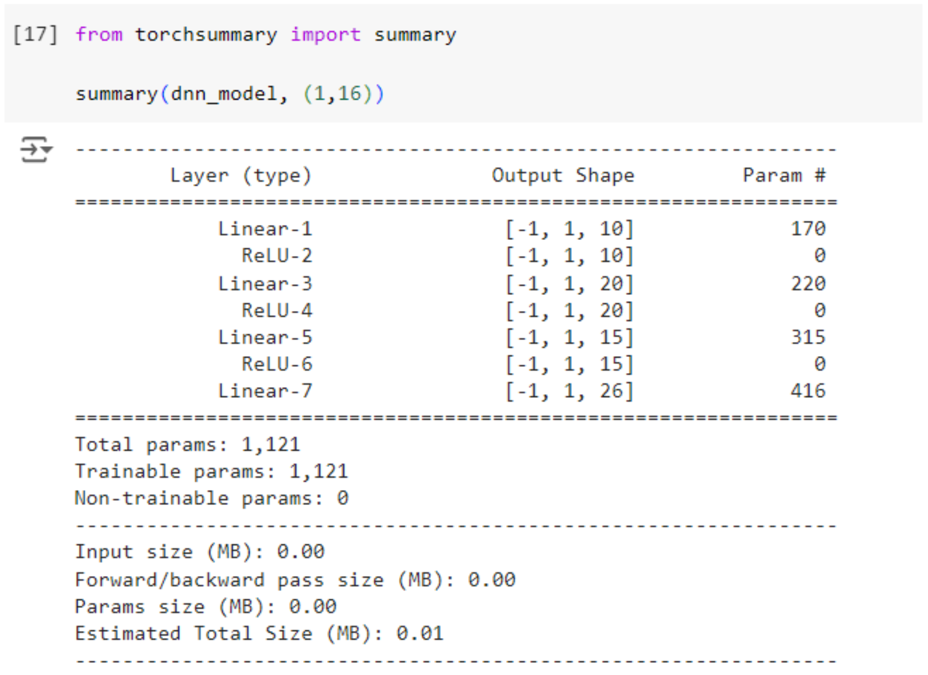
Implementation Feed-forward NNs with PyTorch

Declare a DL model using torch.nn.Sequential



```
[16] device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
     dnn_model = Sequential(Linear(n_features,10), nn.ReLU(),
                            Linear(10,20), nn.ReLU(),
                            Linear(20,15), nn.ReLU(),
                            Linear(15, n_classes)).to(device)
```

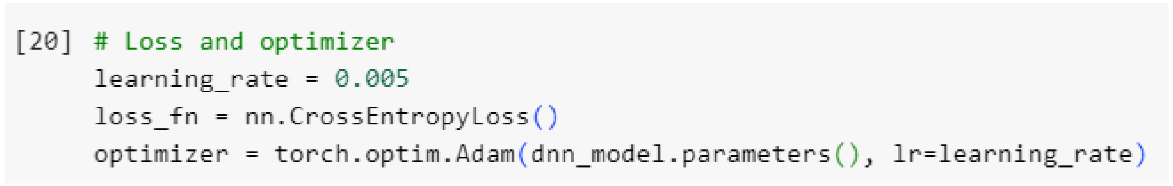
We print out the model summary.



```
[17] from torchsummary import summary

     summary(dnn_model, (1,16))
```

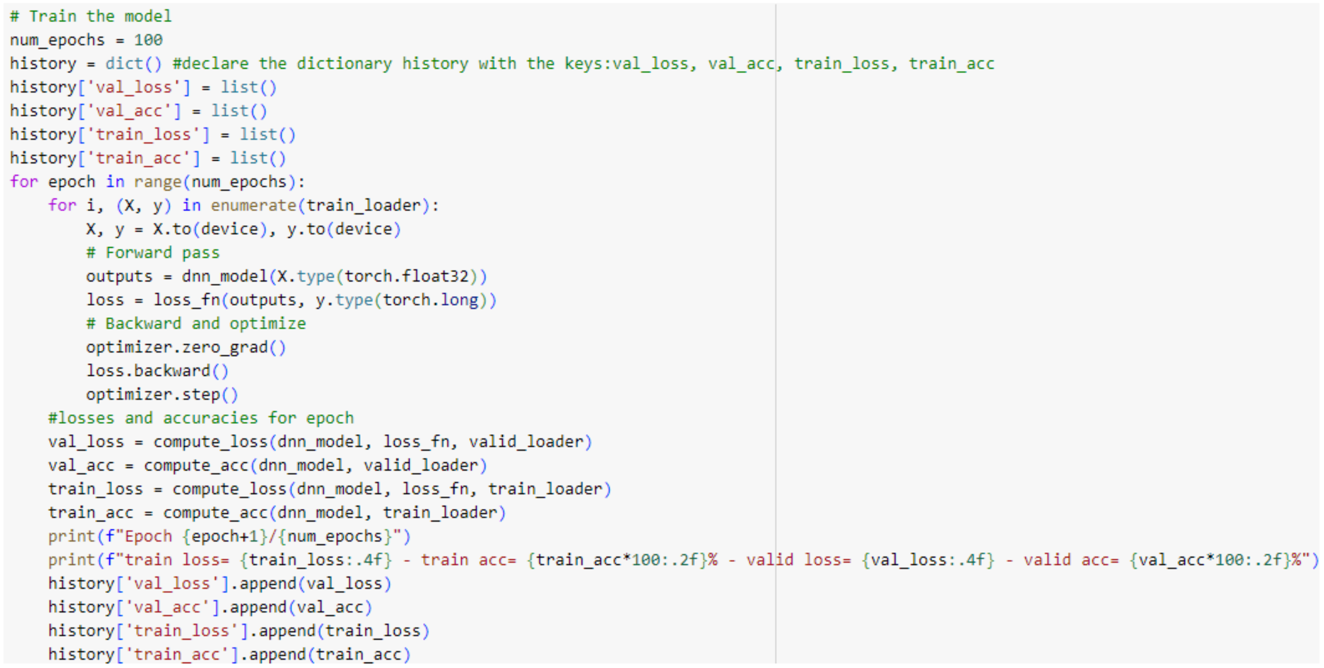
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 10]             170
              ReLU-2                [-1, 1, 10]               0
            Linear-3                [-1, 1, 20]             220
              ReLU-4                [-1, 1, 20]               0
            Linear-5                [-1, 1, 15]             315
              ReLU-6                [-1, 1, 15]               0
            Linear-7                [-1, 1, 26]             416
================================================================
Total params: 1,121
Trainable params: 1,121
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.01
----------------------------------------------------------------
```

Loss and optimizer



```
[20] # Loss and optimizer
     learning_rate = 0.005
     loss_fn = nn.CrossEntropyLoss()
     optimizer = torch.optim.Adam(dnn_model.parameters(), lr=learning_rate)
```

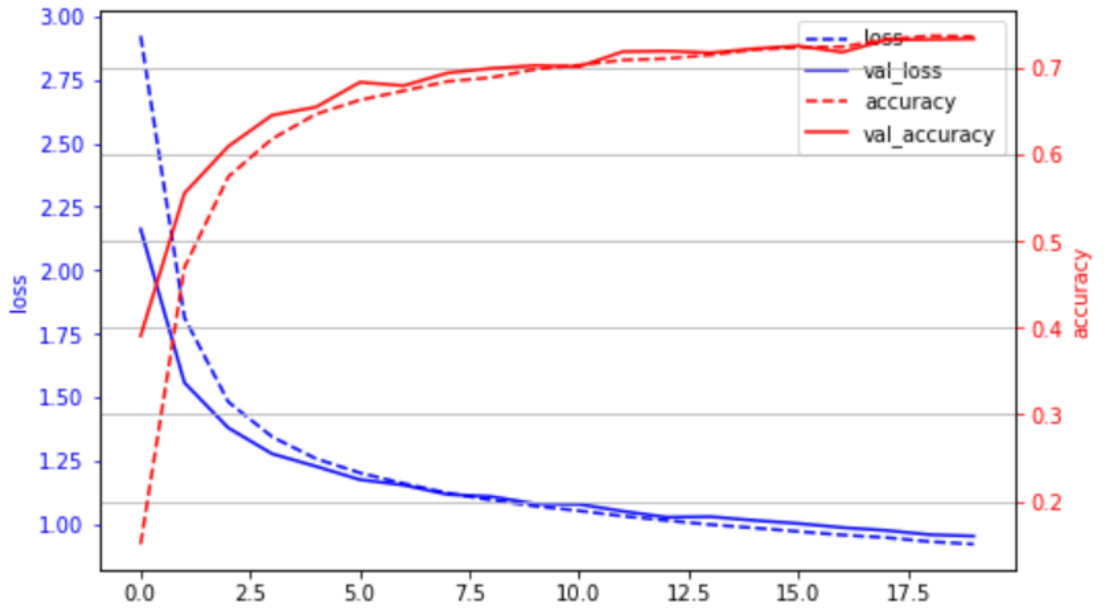
Train model



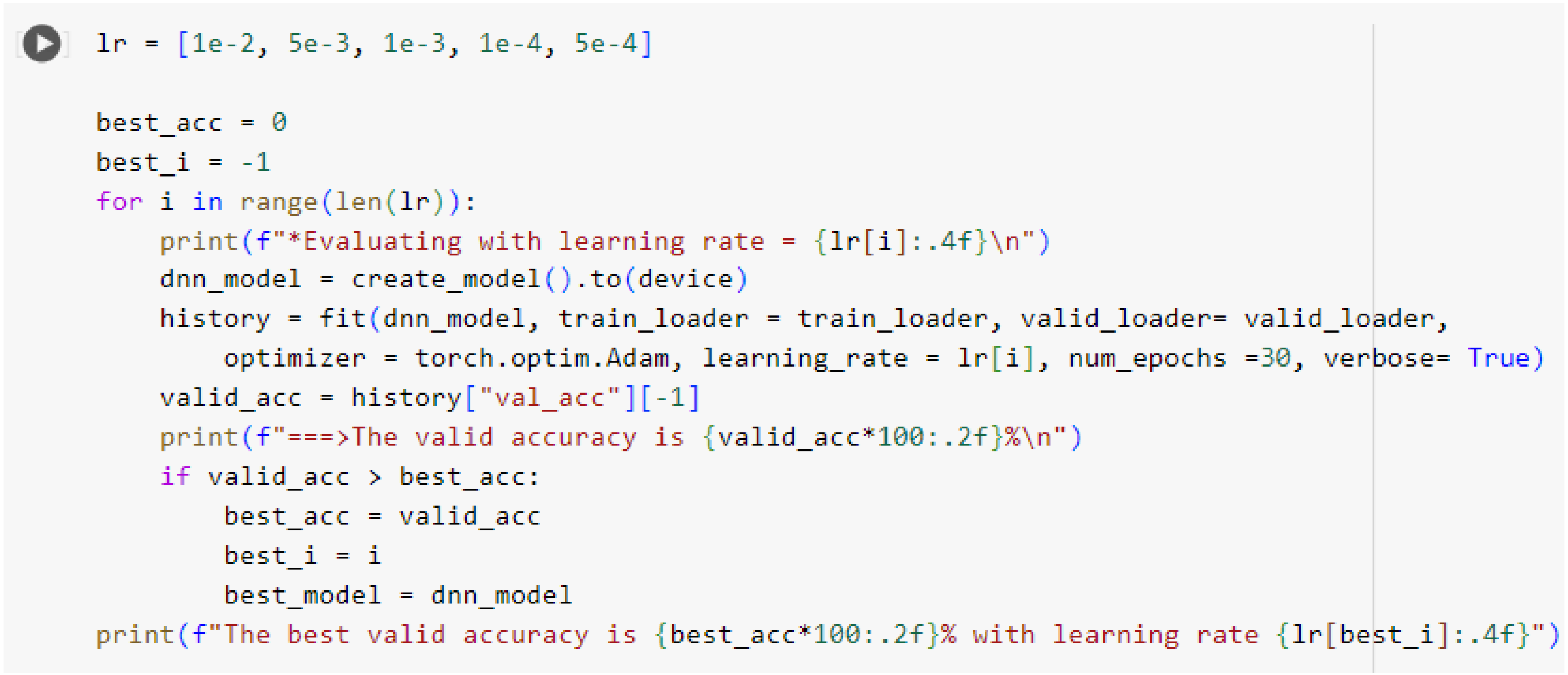
```
# Train the model
num_epochs = 100
history = dict() #declare the dictionary history with the keys:val_loss, val_acc, train_loss, train_acc
history['val_loss'] = list()
history['val_acc'] = list()
history['train_loss'] = list()
history['train_acc'] = list()
for epoch in range(num_epochs):
    for i, (X, y) in enumerate(train_loader):
        X, y = X.to(device), y.to(device)
        # Forward pass
        outputs = dnn_model(X.type(torch.float32))
        loss = loss_fn(outputs, y.type(torch.long))
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    #losses and accuracies for epoch
    val_loss = compute_loss(dnn_model, loss_fn, valid_loader)
    val_acc = compute_acc(dnn_model, valid_loader)
    train_loss = compute_loss(dnn_model, loss_fn, train_loader)
    train_acc = compute_acc(dnn_model, train_loader)
    print(f"Epoch {epoch+1}/{num_epochs}")
    print(f"train loss= {train_loss:.4f} - train acc= {train_acc*100:.2f}% - valid loss= {val_loss:.4f} - valid acc= {val_acc*100:.2f}%")
    history['val_loss'].append(val_loss)
    history['val_acc'].append(val_acc)
    history['train_loss'].append(train_loss)
    history['train_acc'].append(train_acc)
```

Diagnosing the training

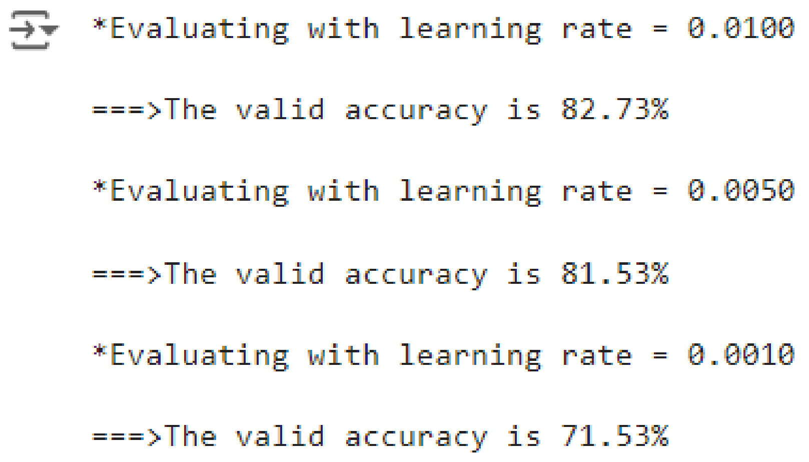
```
import pandas as pd
import matplotlib.pyplot as plt

his = history.history
fig = plt.figure(figsize=(8, 5))
ax = fig.add_subplot(111)
ln1 = ax.plot(his['loss'], 'b--',label='loss')
ln2 = ax.plot(his['val_loss'], 'b-',label='val_loss')
ax.set_ylabel('loss', color='blue')
ax.tick_params(axis='y', colors="blue")

ax2 = ax.twinx()
ln3 = ax2.plot(his['accuracy'], 'r--',label='accuracy')
ln4 = ax2.plot(his['val_accuracy'], 'r-',label='val_accuracy')
ax2.set_ylabel('accuracy', color='red')
ax2.tick_params(axis='y', colors="red")

lns = ln1 + ln2 + ln3 + ln4
labels = [l.get_label() for l in lns]
ax.legend(lns, labels)
plt.grid(True)
plt.show()
```



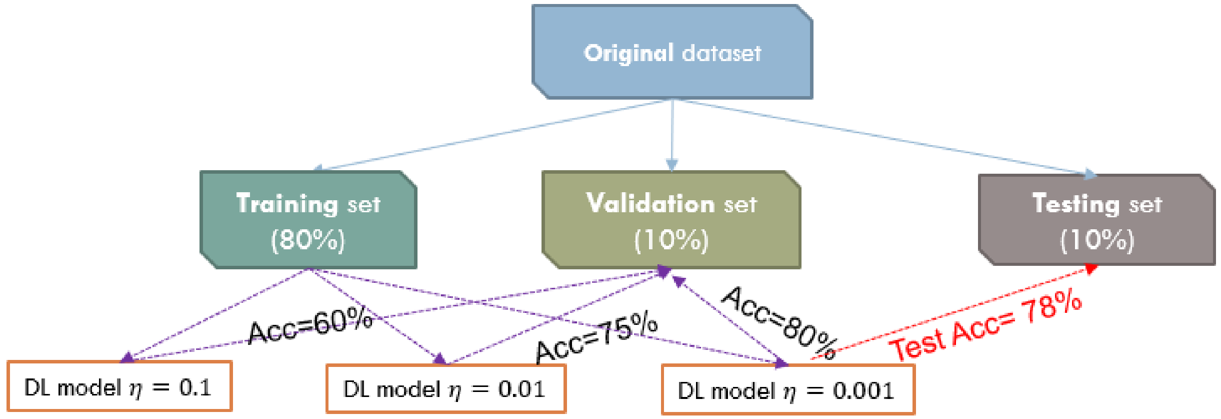
Fine-tune the Learning Rate using Valid Set



```python
lr = [1e-2, 5e-3, 1e-3, 1e-4, 5e-4]

best_acc = 0
best_i = -1
for i in range(len(lr)):
    print(f"*Evaluating with learning rate = {lr[i]:.4f}\n")
    dnn_model = create_model().to(device)
    history = fit(dnn_model, train_loader = train_loader, valid_loader= valid_loader,
        optimizer = torch.optim.Adam, learning_rate = lr[i], num_epochs =30, verbose= True)
    valid_acc = history["val_acc"][-1]
    print(f"===>The valid accuracy is {valid_acc*100:.2f}%\n")
    if valid_acc > best_acc:
        best_acc = valid_acc
        best_i = i
        best_model = dnn_model
print(f"The best valid accuracy is {best_acc*100:.2f}% with learning rate {lr[best_i]:.4f}")
```



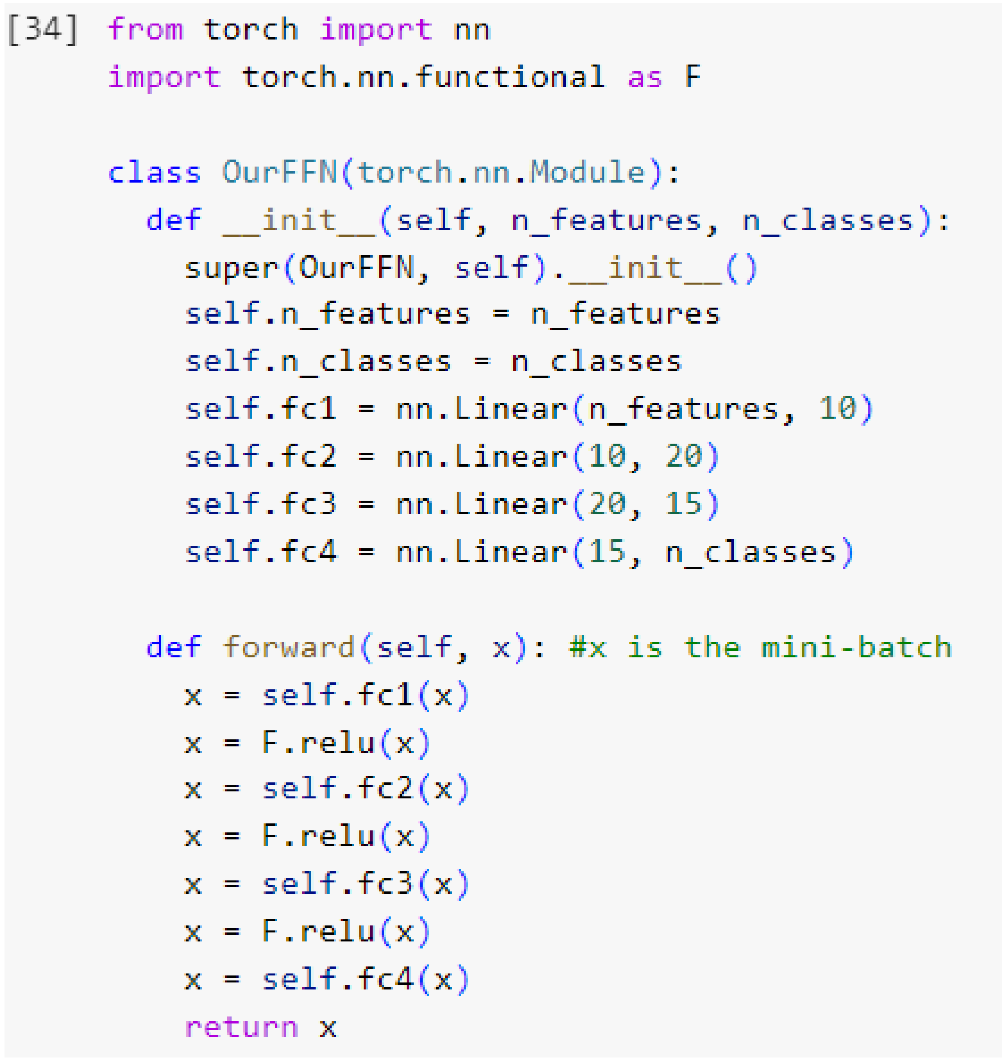
```
*Evaluating with learning rate = 0.0100

===>The valid accuracy is 82.73%

*Evaluating with learning rate = 0.0050

===>The valid accuracy is 81.53%

*Evaluating with learning rate = 0.0010

===>The valid accuracy is 71.53%
```



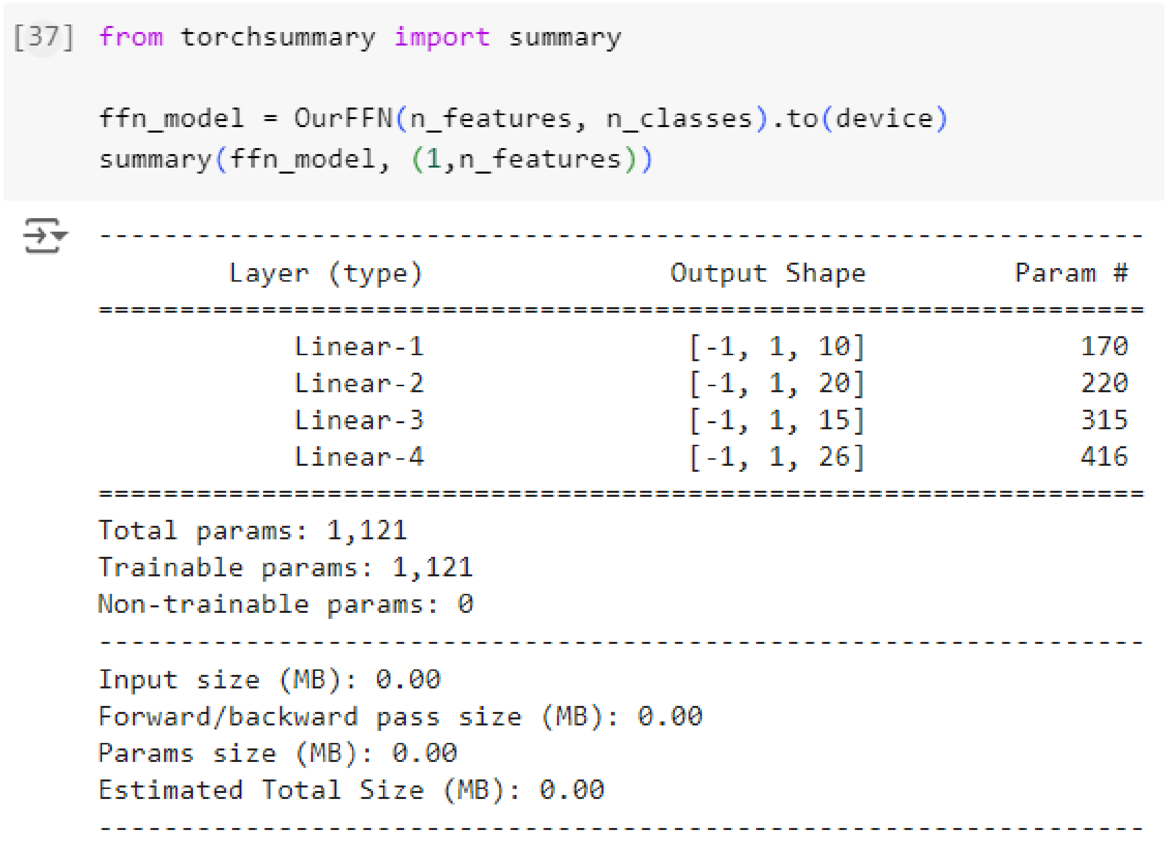
DL Pipeline

Another Approach to Implement DL models with PyTorch

Declare a class inherited from torch.nn.Module



```python
[34] from torch import nn
     import torch.nn.functional as F

     class OurFFN(torch.nn.Module):
       def __init__(self, n_features, n_classes):
         super(OurFFN, self).__init__()
         self.n_features = n_features
         self.n_classes = n_classes
         self.fc1 = nn.Linear(n_features, 10)
         self.fc2 = nn.Linear(10, 20)
         self.fc3 = nn.Linear(20, 15)
         self.fc4 = nn.Linear(15, n_classes)

       def forward(self, x): #x is the mini-batch
         x = self.fc1(x)
         x = F.relu(x)
         x = self.fc2(x)
         x = F.relu(x)
         x = self.fc3(x)
         x = F.relu(x)
         x = self.fc4(x)
         return x
```



```python
[37] from torchsummary import summary

     ffn_model = OurFFN(n_features, n_classes).to(device)
     summary(ffn_model, (1,n_features))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 10]             170
            Linear-2                [-1, 1, 20]             220
            Linear-3                [-1, 1, 15]             315
            Linear-4                [-1, 1, 26]             416
================================================================
Total params: 1,121
Trainable params: 1,121
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
```

Thanks for your attention!

Enjoy the tutorial 2!