PyTorch
Deep learning Framework

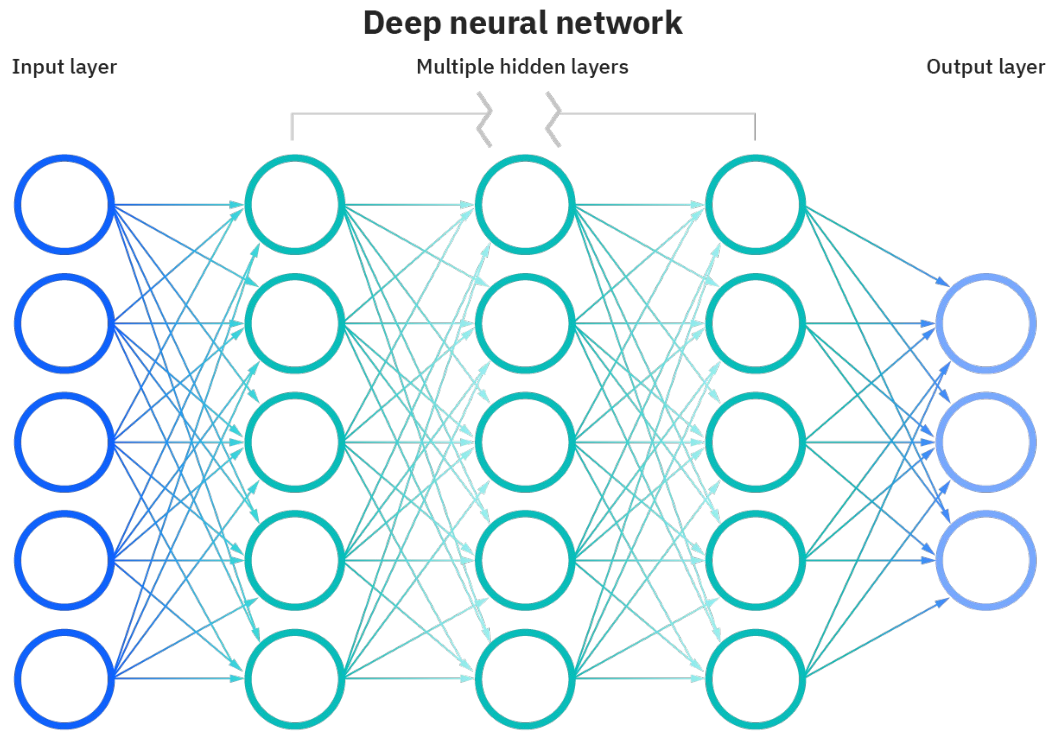
Yuchen Li

10 June 2022

22125 Algorithms in Bioinformatics

Before starting

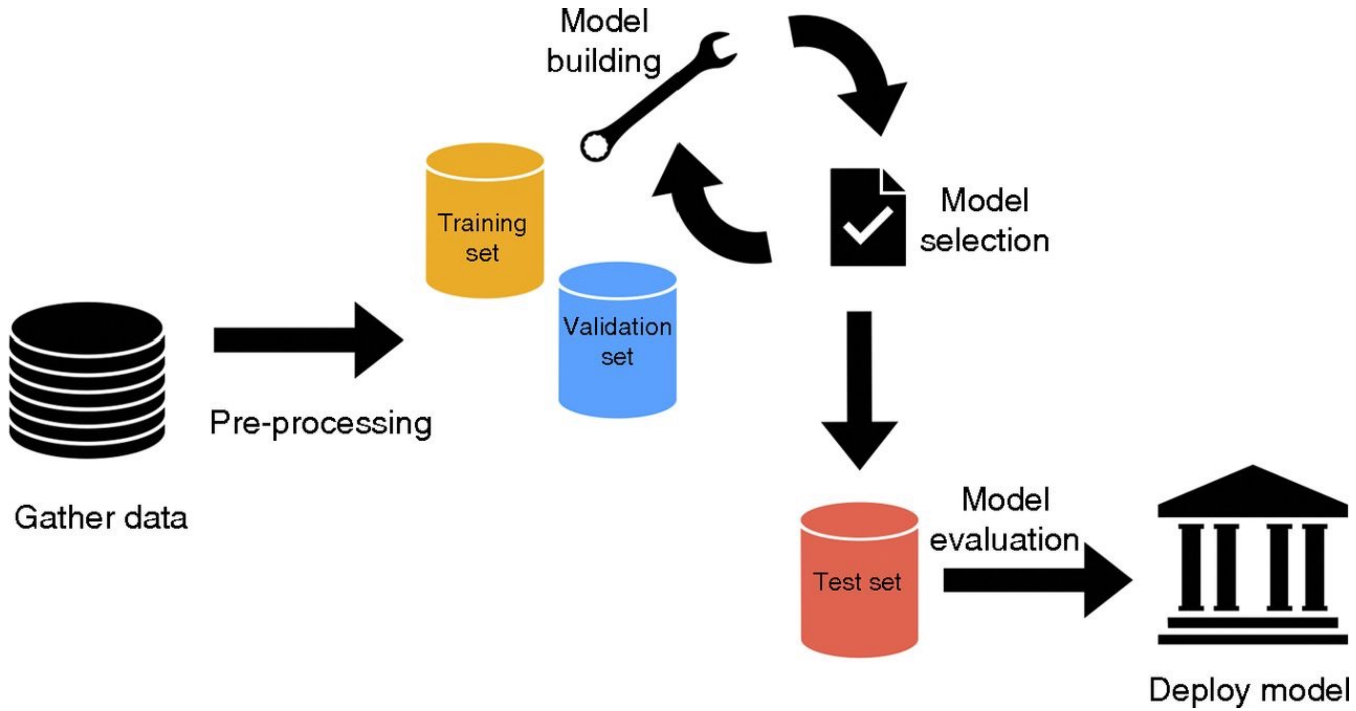
- Input dimensions
- Output dimensions
- Loss functions
- Network structures
  - FFNN/CNN
  - Activation functions



https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks

Build neural network

- Load data
- Build model
- Training
- Evaluation



https://kidney360.asnjournals.org/content/2/5/878

Load data

```python
from torch.utils.data import Dataset, DataLoader

class MyDataset(Dataset):
    def __init__(self):
        # define your data paths
        # define other arguments
        self.data = 'XXX'

    def __getitem__(self, index):
        # preprocessing
        return self.data[index]

    def __len__(self):
        # return the size of dataset
        return len(self.data)

    # define other functions
```

```python
dataset = MyDataset()
dataloader = DataLoader(dataset, batch_size)
```

Build Model

```
class Net(nn.Module):

    def __init__(self, n_features, n_l1):
        super(Net, self).__init__()
        # define hidden layers
        self.fc1 = nn.Linear(n_features, n_l1)
        self.fc2 = nn.Linear(n_l1, 1)

        # define activation functions
        self.relu = nn.ReLU()
        self.tanh = nn.Tanh()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        # connect hidden layers
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

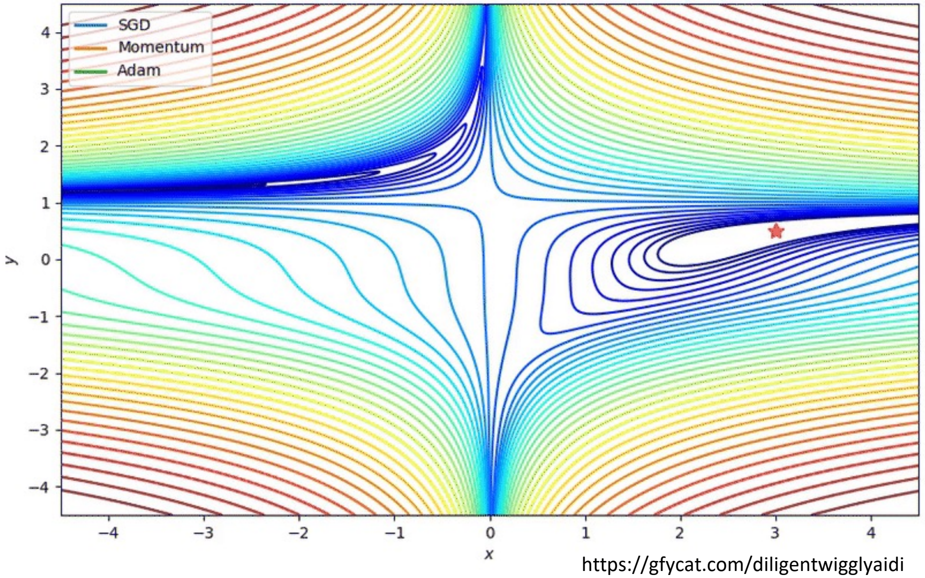
```
net = Net(n_features, N_HIDDEN_NEURONS)
```

Build Model

```
class CNNpep(nn.Module):

    def __init__(self, n_filters, k, n_l1):
        super(CNNpep, self).__init__()
        self.conv_layer = nn.Conv1d(in_channels=21,
                                    out_channels=n_filters,
                                    kernel_size=k,
                                    stride=1,
                                    padding=0)

        self.fc1 = nn.Linear(n_filters, n_l1)
        self.fc2 = nn.Linear(n_l1, 1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        # Permutation of the dimensions for the cnn
        x = x.permute(0, 2, 1)
        x = self.relu(self.conv_layer(x))
        x, _ = torch.max(x, axis=2)
        x = self.relu(self.fc1(x))
        out = self.sigmoid(self.fc2(x))

        return out
```

Optimizer & loss function

```
#define optimizer
optimizer = optim.SGD(net.parameters(), lr=LEARNING_RATE)

# define loss function
criterion = nn.MSELoss()
```



https://gfycat.com/diligentwigglyaidi

`nn.MSELoss`	Creates a criterion that measures the mean squared error (squared L2 norm) between each element in the input x and target y.
`nn.CrossEntropyLoss`	This criterion computes the cross entropy loss between input and target.

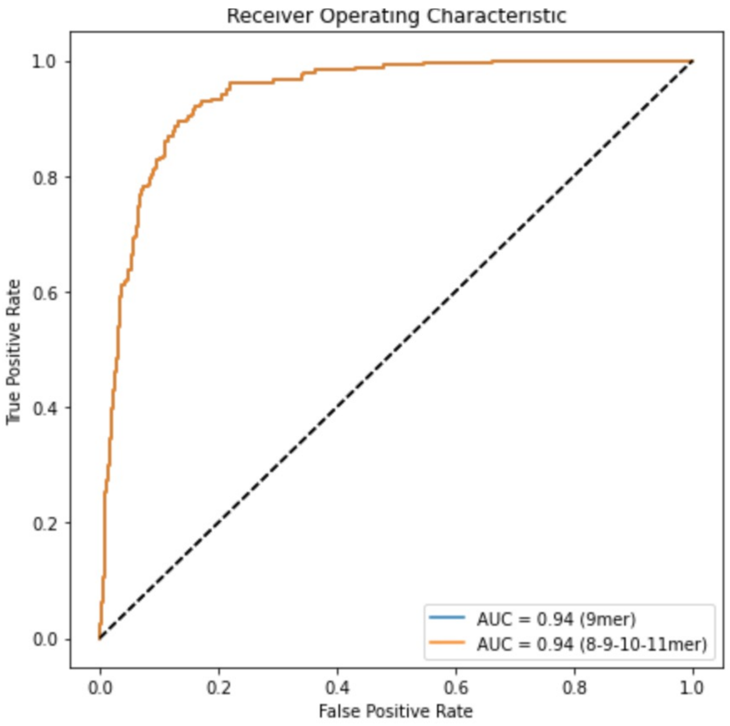
https://pytorch.org/docs/stable/nn.html

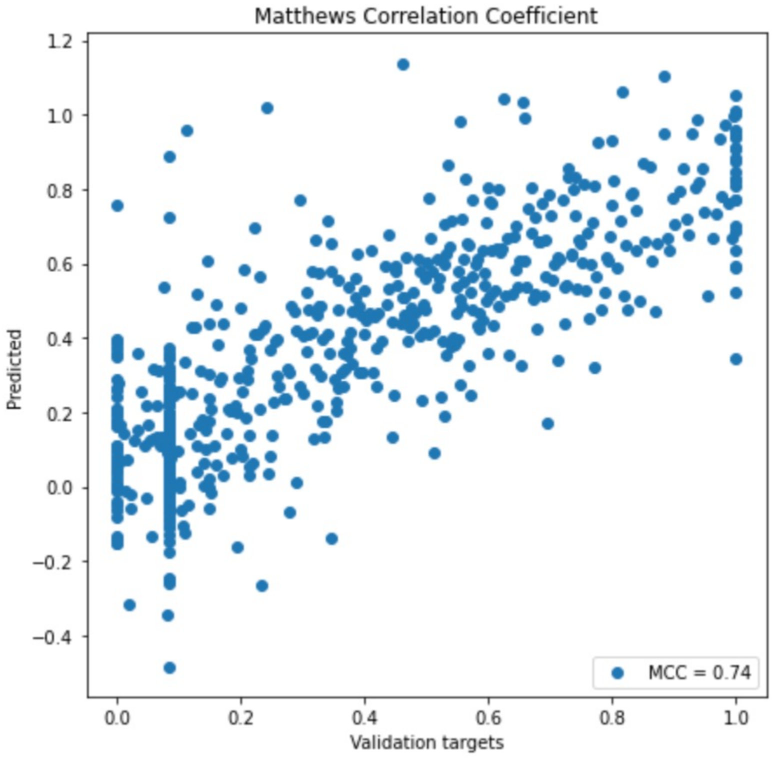
Training

```
train_loss, valid_loss = [], []
early_stopping = EarlyStopping(patience=PATIENCE)

for epoch in range(EPOCHS):
    # model training
    net.train()
    # predict training data and calculate the loss function
    pred = net(x_train)
    loss = criterion(pred, y_train)

    # backward propagation
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    train_loss.append(loss.data)

    if epoch % (EPOCHS//10) == 0:
        print('Train Epoch: {}\tLoss: {:.6f}'.format(epoch, loss.data))

    # model evaluation
    net.eval()
    pred = net(x_valid)
    loss = criterion(pred, y_valid)
    valid_loss.append(loss.data)

    if invoke(early_stopping, valid_loss[-1], net, implement=True):
        net.load_state_dict(torch.load('checkpoint.pt'))
        break
```

Evaluation

```
net.eval()
pred = net(x_test)
loss = criterion(pred, y_test)
```





Overfitting

- Weight decay

```
optimizer = optim.SGD(net.parameters(), lr=LEARNING_RATE, weight_decay=1e-4)
```

- Dropout

```
        # define dropout
        self.drop_layer = nn.Dropout(0.2)


    def forward(self, x):
        # connect hidden layers
        x = self.relu(self.fc1(x))
        x = self.fc2(self.drop_layer(x))
        return x
```

Overfitting

- Early stopping

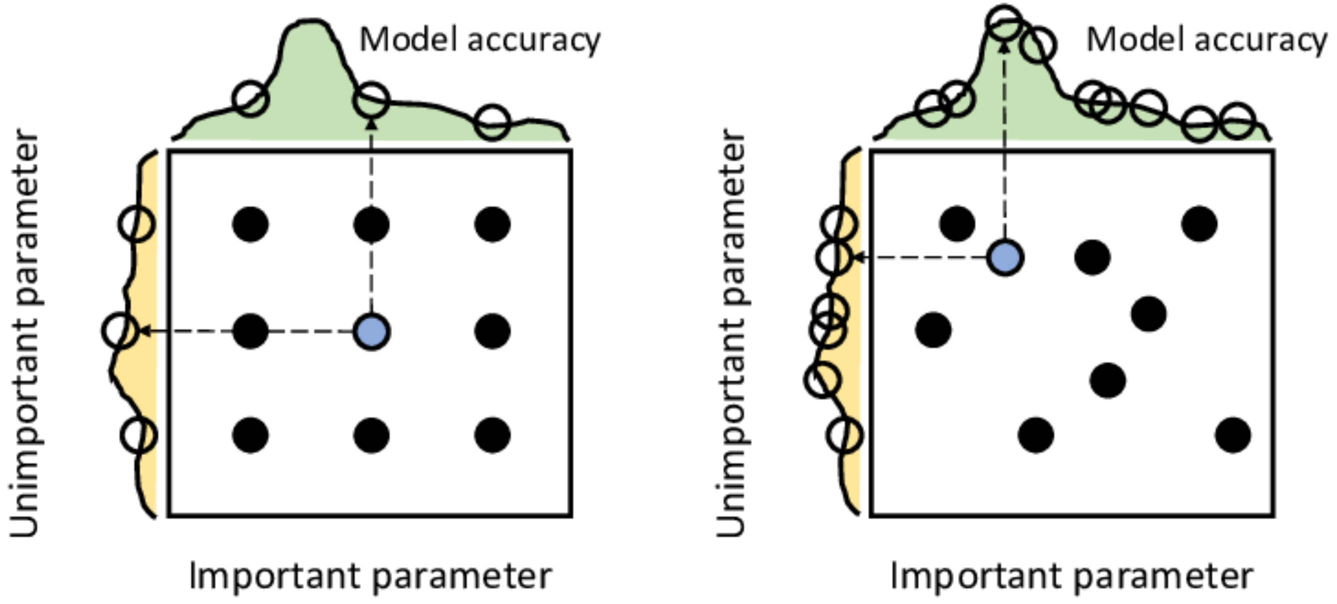
```
early_stopping = EarlyStopping(patience=PATIENCE)

    ......

    if invoke(early_stopping, valid_loss[-1], net, implement=True):
        net.load_state_dict(torch.load('checkpoint.pt'))
        break
```

- Batch normalization

```
        # define batch normalizaiton
        self.batch_norm = nn.BatchNorm1d(n_l1)


    def forward(self, x):
        # connect hidden layers
        x = self.relu(self.fc1(x))
        x = self.fc2(self.batch_norm(x))
        return x
```

Hyperparameters

- Number of hidden units
- Number of hidden layers
- Number of channels
- Batch size
- Learning rate
- Weight decay
- Optimizer
- ……



(a) (b)

10.1109/TIE.2020.2996142