ANNs with Pytorch API

Algorithms in Bioinformatics

Who am I?

PhD student with Morten

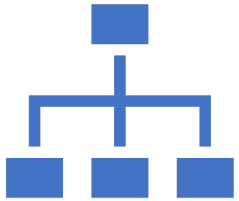
Graduated in March 2019

Warning! I am no expert

What’s up for today?

Play with Pytorch

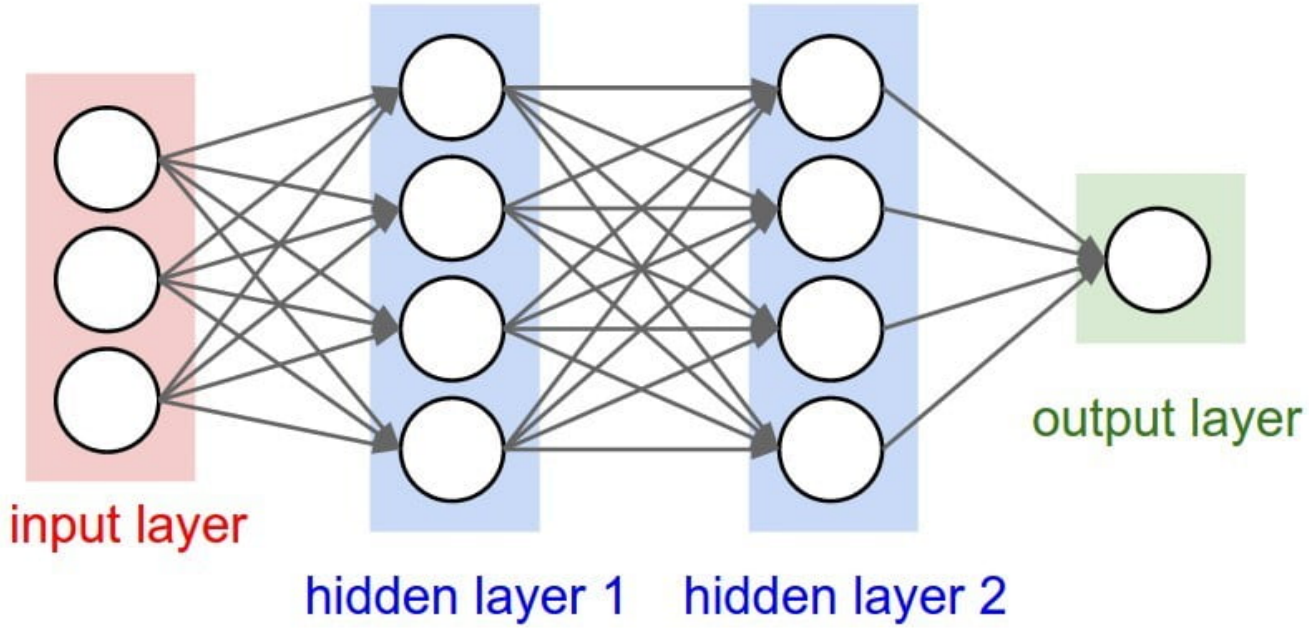
Why is Pytorch smart?



Other frameworks?

How to build an Artificial Neural Network?

- Invariant parameters
  - Input dimension
  - Output dimension
- Semi-variable
  - Loss-function
- Variable
  - # hidden layers
  - # hidden neurons
  - Activation functions
  - GD optimizations (learning rate, momentum, batch)
- Add-ons
  - Normalization
  - Regularization
  - Dropout



How do I build the best network?

You most likely won’t

Hyper-parameter search

Methods

- Manual Search
- Grid Search

```
from sklearn.model_selection import GridSearchCV
```

- Random Grid Search

```
from sklearn.model_selection import RandomizedSearchCV
```

Evaluation

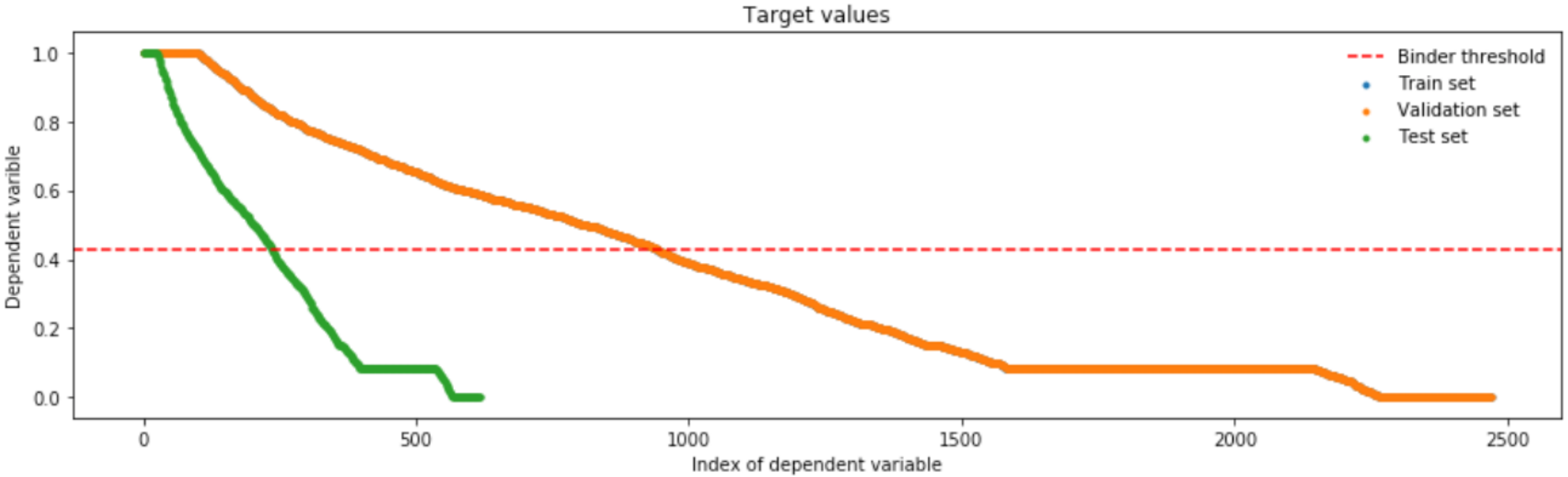
- Evolutionary Algorithms
- Bayesian Hyper-parameter Evaluation

  https://github.com/HIPS/Spearmint

Getting started

- Load Data
- Build Model
- Select Hyper-parameters
- Compile Model
- Train Model
- Evaluate Model

Load data


Target values
Binder threshold
Train set
Validation set
Test set
Dependent varible
Index of dependent variable
1.0
0.8
0.6
0.4
0.2
0.0
0
500
1000
1500
2000
2500

Build model

```
class Net(nn.Module):

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

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

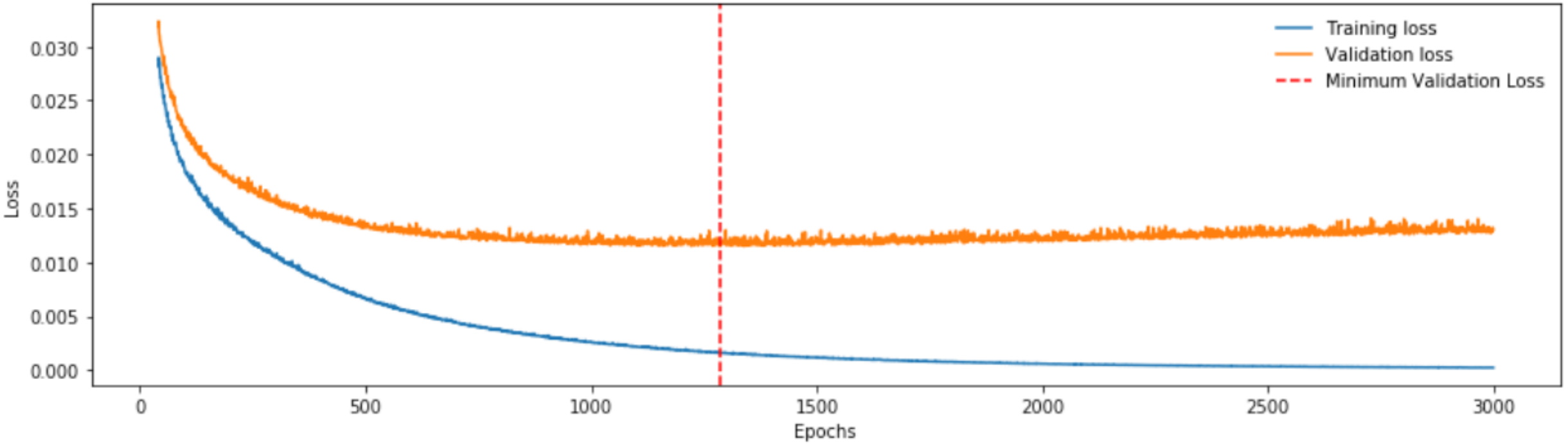
Compile model

```
net = Net(n_features, N_HIDDEN_NEURONS)

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

criterion = nn.MSELoss()
```

Train Model

```
def train():
    train_loss, valid_loss = [], []

    early_stopping =
        EarlyStopping(patience=PATIENCE)

    for epoch in range(EPOCHS):
        net.train()
        pred = net(x_train)
        loss = criterion(pred, y_train)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        train_loss.append(loss.data)
        batch_loss = 0
        net.eval()
        for x_valid, y_valid in valid_loader:
            pred = net(x_valid)
            loss = criterion(pred, y_valid)
            batch_loss += loss.data
        valid_loss.append(batch_loss /
                          len(valid_loader))

        if invoce(early_stopping,
                  valid_loss[-1], net,
                  implement=False):
            load_checkpoints()
            break

    return net, train_loss, valid_loss
```

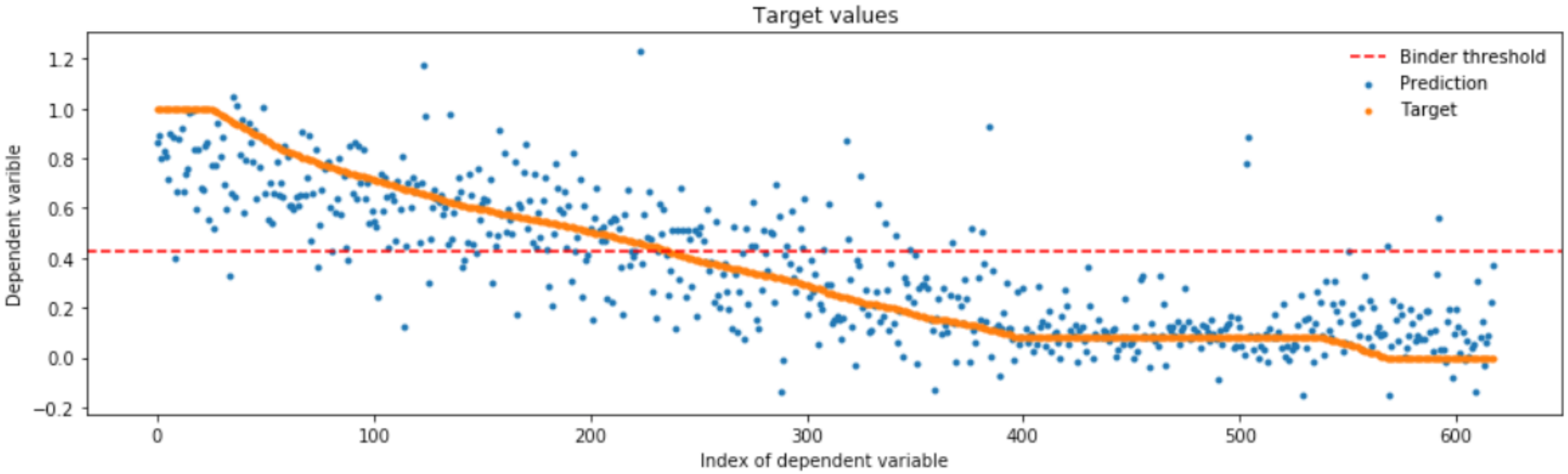
Load data


Training loss
Validation loss
Minimum Validation Loss
Loss
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
500
1000
1500
2000
2500
3000
Epochs

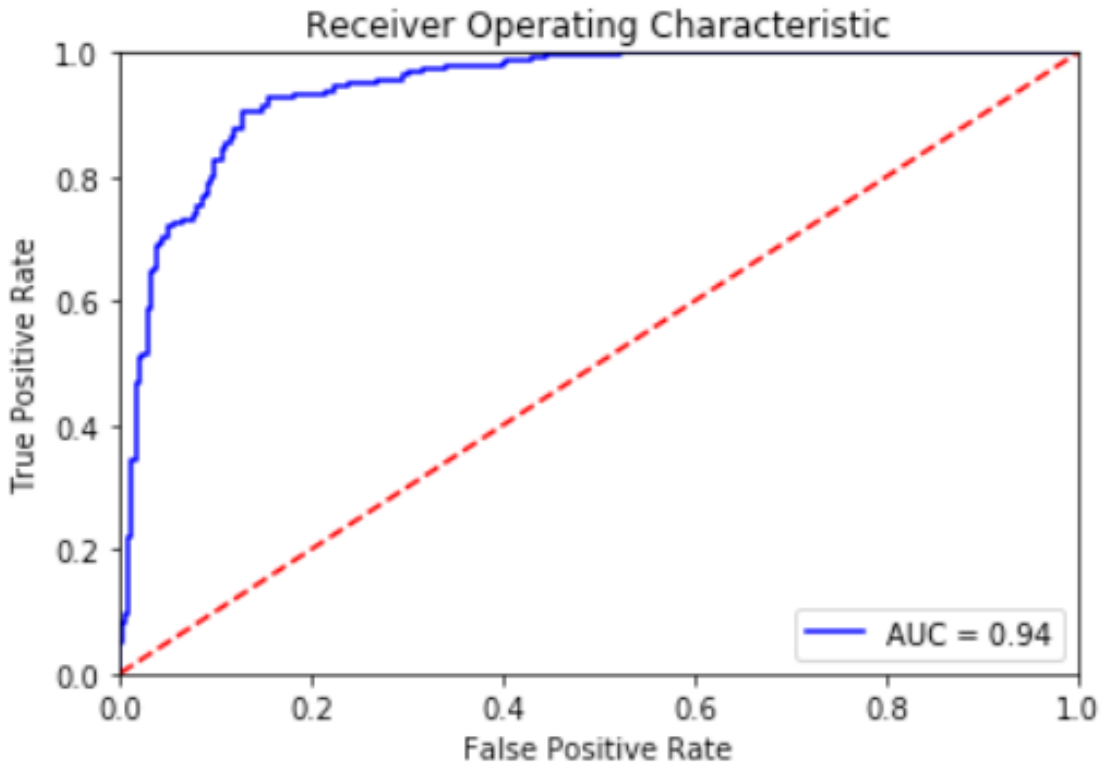
Evaluate model

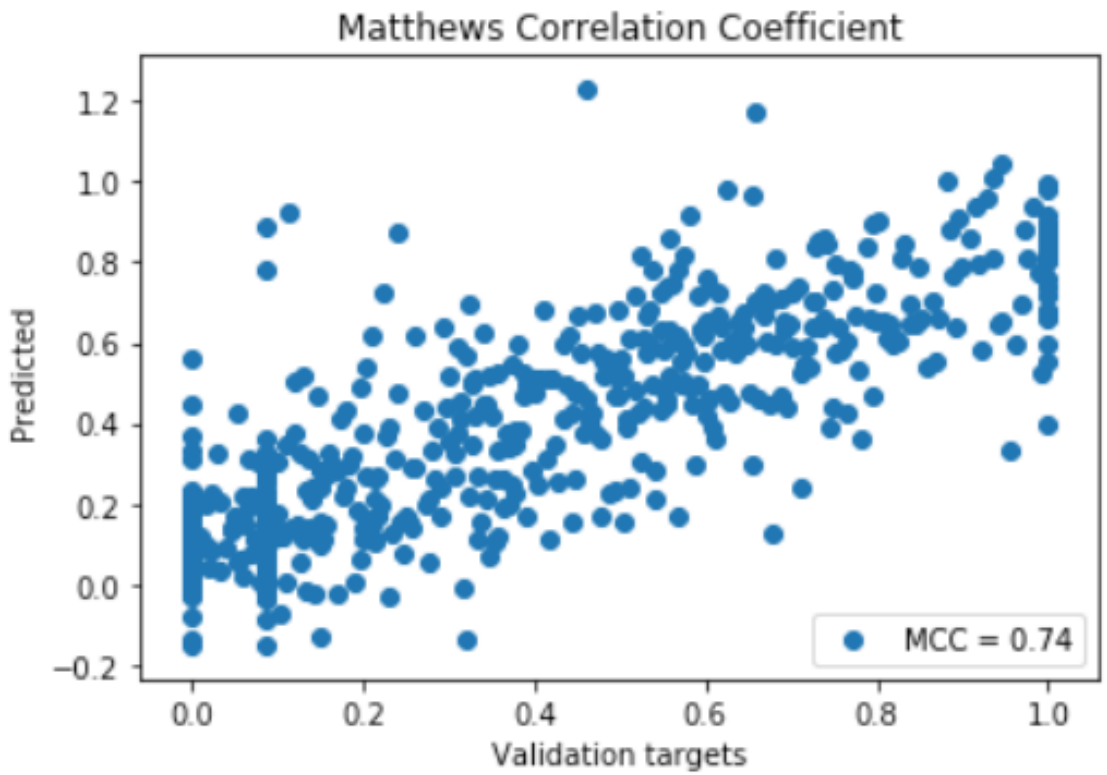
```
net.eval()
pred = net(x_test)
loss = criterion(pred, y_test)
```

Evaluate model



Evaluate model





Avoid overfitting

- L2 Regularization

```
optim.SGD(net.parameters(), lr=LEARNING_RATE, weight_decay=0)
```

- L1 Regularization

```
Manual implementation using nn.L1Loss
```

- Dropout

```
Def __init__(): self.drop_layer = nn.Dropout(p=p)
Def Forward(): x = self.drop_layer(x)
```

- Batch normalization

```
Def __init__(): self.batch_norm = nn.BatchNorm1d(n_neurons)
Def Forward(): x = self.batch_norm(x)
```

Give it a go!