Understanding model optimization

Why optimization is hard

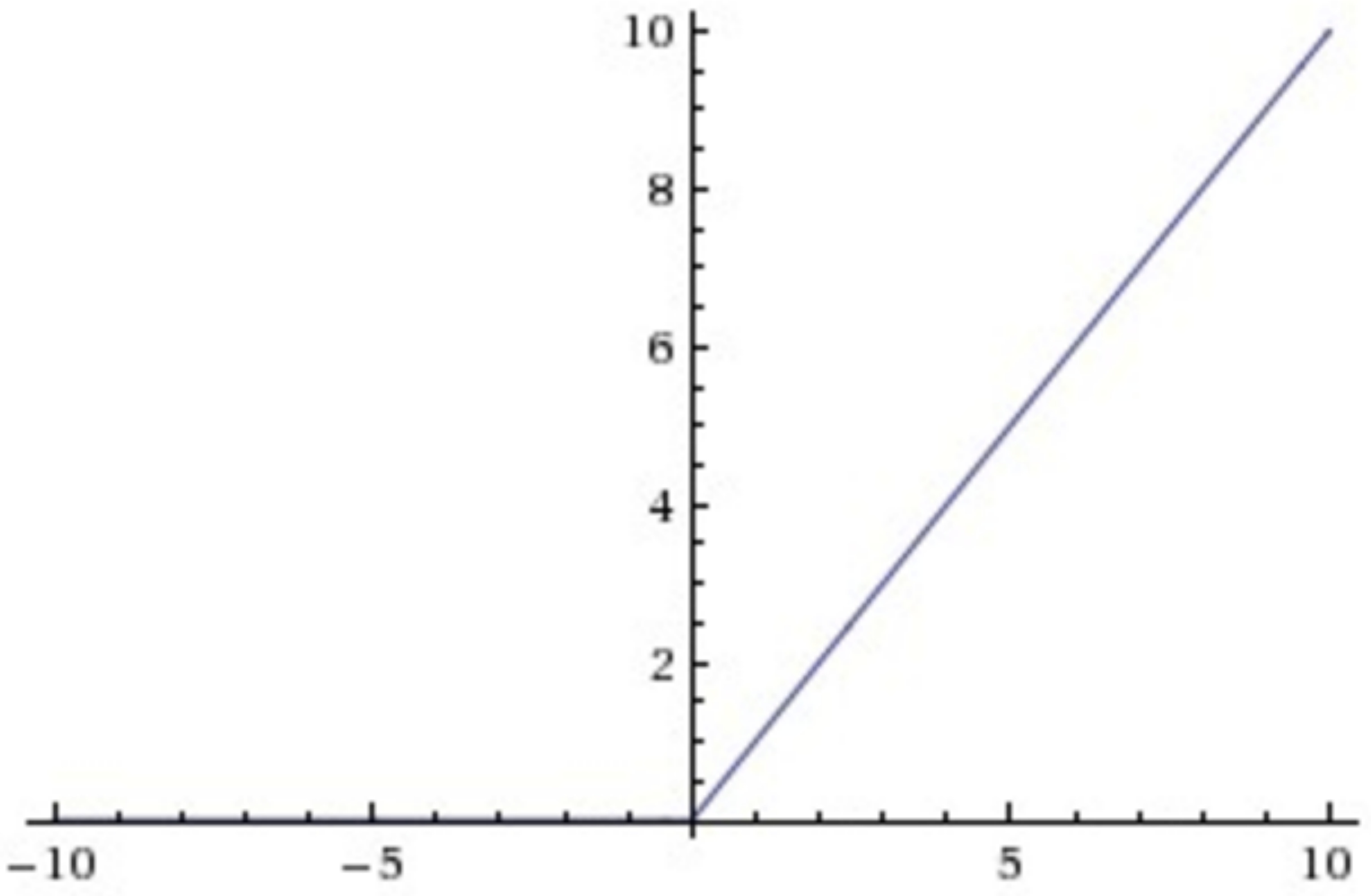
- Simultaneously optimizing 1000s of parameters with complex relationships
- Updates may not improve model meaningfully
- Updates too small (if learning rate is low) or too large (if learning rate is high)

Stochastic gradient descent

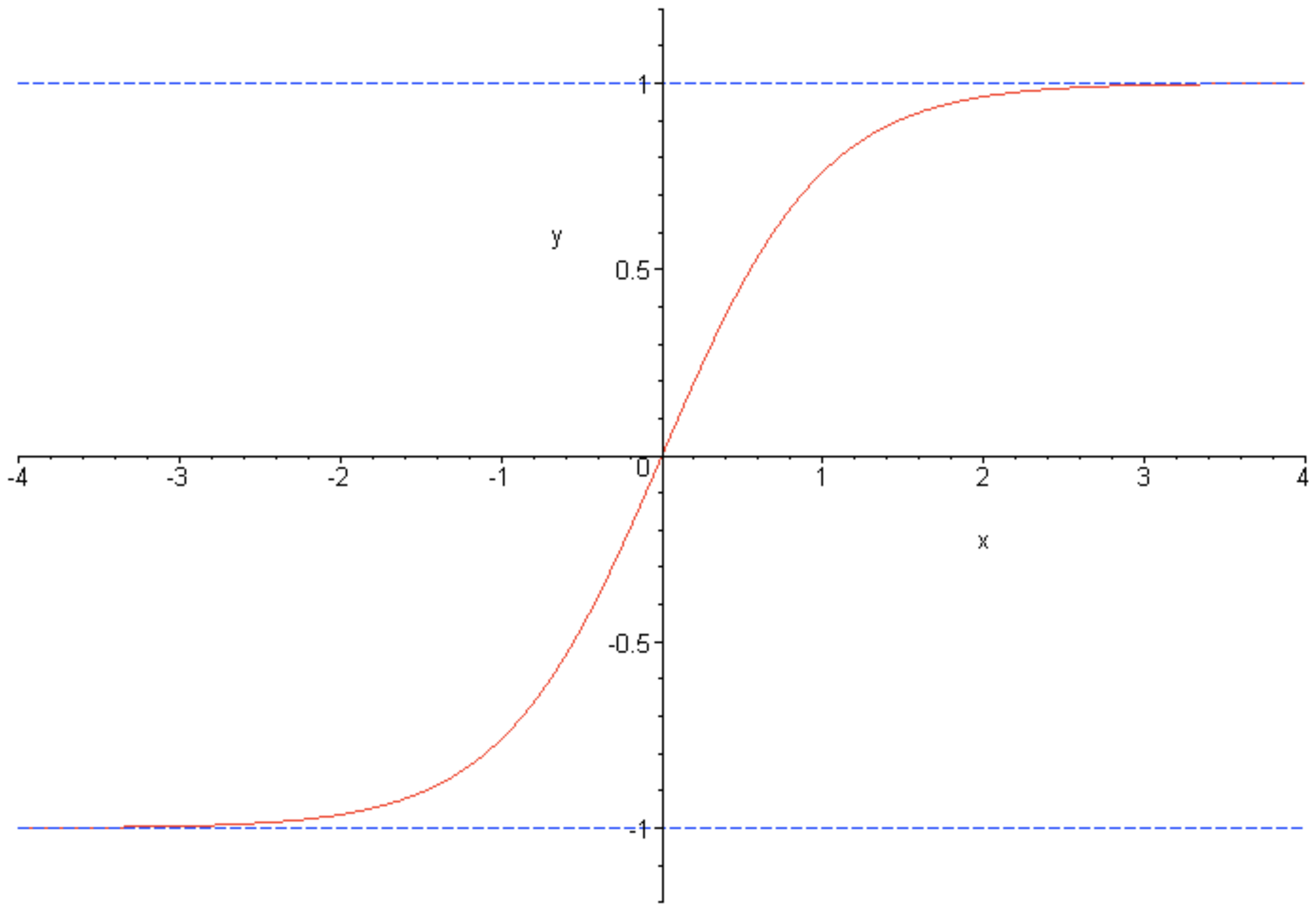
sgd.py

```
def get_new_model(input_shape = input_shape):
    model = Sequential()
    model.add(Dense(100, activation='relu', input_shape = input_shape))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(2, activation='softmax'))
    return(model)

lr_to_test = [.000001, 0.01, 1]

# loop over learning rates
for lr in lr_to_test:
   model = get_new_model()
   my_optimizer = SGD(lr=lr)
   model.compile(optimizer = my_optimizer, loss = 'categorical_crossentropy')
   model.fit(predictors, target)
```

The dying neuron problem



Vanishing gradients



tanh function

Vanishing gradients

- Occurs when many layers have very small slopes (e.g. due to being on flat part of tanh curve)
- In deep networks, updates to backprop were close to 0

Let's practice!

Model validation

Validation in deep learning

- Commonly use validation split rather than cross-validation
- Deep learning widely used on large datasets
- Single validation score is based on large amount of data, and is reliable
- Repeated training from cross-validation would take long time

Model validation

```
In [1]: model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics=['accuracy'])

In [2]: model.fit(predictors, target, validation_split=0.3)

Epoch 1/10
89648/89648 [==============================] - 3s - loss: 0.7552 - acc: 0.5775 - val_loss: 0.6969 -
val_acc: 0.5561
Epoch 2/10
89648/89648 [==============================] - 4s - loss: 0.6670 - acc: 0.6004 - val_loss: 0.6580 -
val_acc: 0.6102
...
Epoch 8/10
89648/89648 [==============================] - 5s - loss: 0.6578 - acc: 0.6125 - val_loss: 0.6594 -
val_acc: 0.6037
Epoch 9/10
89648/89648 [==============================] - 5s - loss: 0.6564 - acc: 0.6147 - val_loss: 0.6568 -
val_acc: 0.6110
Epoch 10/10
89648/89648 [==============================] - 5s - loss: 0.6555 - acc: 0.6158 - val_loss: 0.6557 -
val_acc: 0.6126
```

Early Stopping

```
In [3]: from keras.callbacks import EarlyStopping

In [4]: early_stopping_monitor = EarlyStopping(patience=2)

In [5]: model.fit(predictors, target, validation_split=0.3, nb_epoch=20,
   ...:           callbacks = [early_stopping_monitor])
```

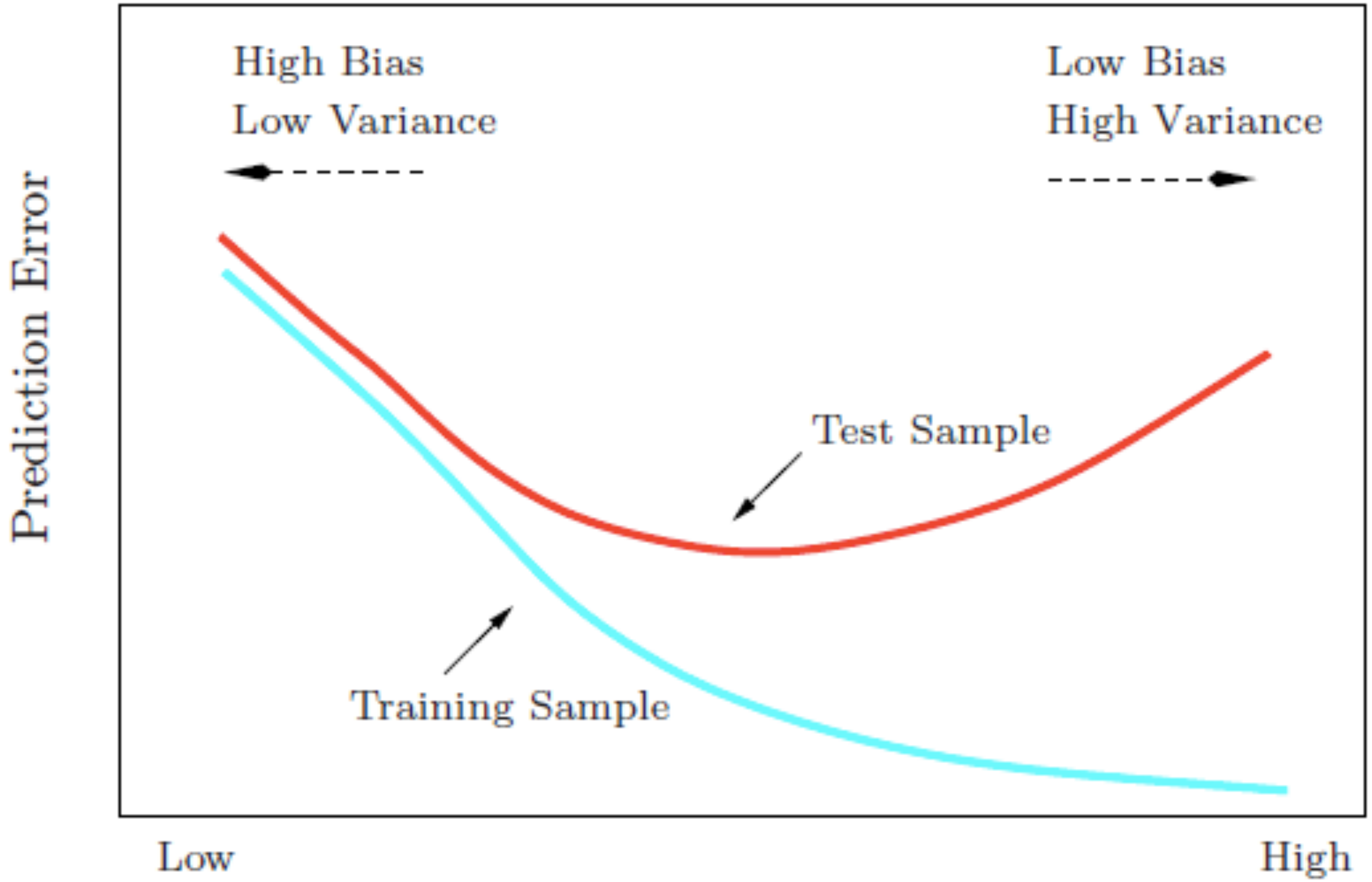
Output from early stopping

```
Train on 89648 samples, validate on 38421 samples
Epoch 1/20
89648/89648 [==============================] - 5s - loss: 0.6550 - acc: 0.6151 - val_loss: 0.6548 -
val_acc: 0.6151
Epoch 2/20
89648/89648 [==============================] - 6s - loss: 0.6541 - acc: 0.6165 - val_loss: 0.6537 -
val_acc: 0.6154
...
Epoch 8/20
89648/89648 [==============================] - 6s - loss: 0.6527 - acc: 0.6181 - val_loss: 0.6531 -
val_acc: 0.6160
Epoch 9/20
89648/89648 [==============================] - 7s - loss: 0.6524 - acc: 0.6176 - val_loss: 0.6513 -
val_acc: 0.6172
Epoch 10/20
89648/89648 [==============================] - 6s - loss: 0.6527 - acc: 0.6176 - val_loss: 0.6549 -
val_acc: 0.6134
Epoch 11/20
89648/89648 [==============================] - 6s - loss: 0.6522 - acc: 0.6178 - val_loss: 0.6517 -
val_acc: 0.6169
```

Let's practice!

Thinking about model capacity

Overfitting



Model Capacity

Workflow for optimizing model capacity

- Start with a small network
- Gradually increase capacity
- Keep increasing capacity until validation score is no longer improving

Sequential experiments

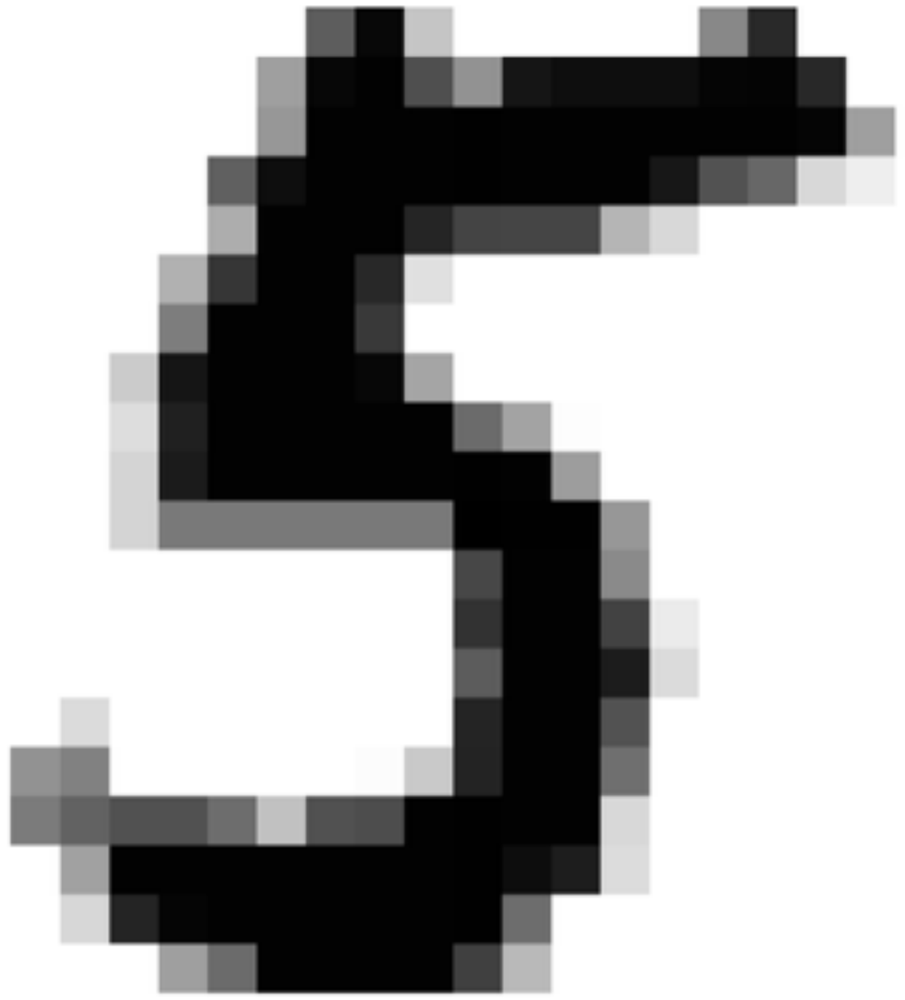
Hidden Layers	Nodes Per Layer	Mean Squared Error	Next Step
1	100	5.4	Increase Capacity
1	250	4.8	Increase Capacity
2	250	4.4	Increase Capacity
3	250	4.5	Decrease Capacity
3	200	4.3	Done

Let’s practice!

Stepping up to images

Recognizing handwritten digits

- MNIST dataset
- 28 x 28 grid flattened to 784 values for each image
- Value in each part of array denotes darkness of that pixel



Let's practice!

Final thoughts

Next steps

- Start with standard prediction problems on tables of numbers
- Images (with convolutional neural networks) are common next steps
- keras.io for excellent documentation
- Graphical processing unit (GPU) provides dramatic speedups in model training times
- Need a CUDA compatible GPU
- For training on using GPUs in the cloud look here: http://bit.ly/2mYQXQb

Congratulations!