APS360 Fundamentals of AI

Lisa Zhang

Lecture 3; May 13, 2019

Agenda

Last Class:

- We trained our first neural network!

This Class:

- Review new ideas / terminology
  - activation functions
  - neural network architecture
  - training / test sets
- More on neural network training
- (We won't get through all the slides today)

Agenda

Last Class:

- We trained our first neural network!

This Class:

- Review new ideas / terminology
  - activation functions
  - neural network architecture
  - training / test sets
- More on neural network training
- (We won't get through all the slides today)

Reminder: Lab 1 is due May 15, 9pm

Agenda

Last Class:

- We trained our first neural network!

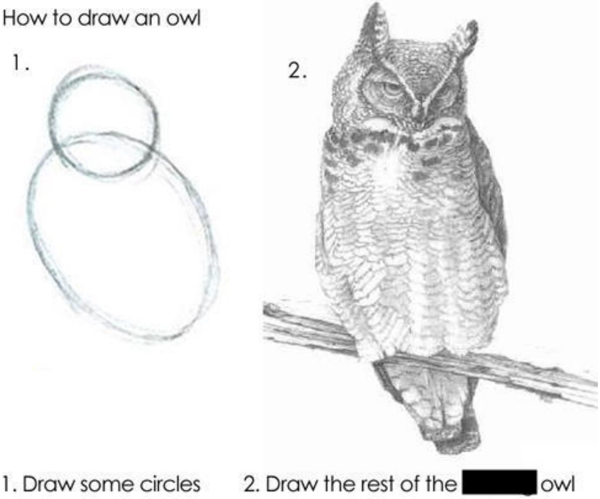
This Class:

- Review new ideas / terminology
  - activation functions
  - neural network architecture
  - training / test sets
- More on neural network training
- (We won't get through all the slides today)

Reminder: Lab 1 is due May 15, 9pm

I will be away the next two classes. **Jake** will deliver the lectures.

Last Class


How to draw an owl
1.
2.
1. Draw some circles
2. Draw the rest of the [redacted] owl

Code from Last Class

It is completely okay to not understand all the code.

We will be writing very similar code several times.

You should have a high-level understanding of how neural networks are trained, and how it is similar/different from “training” a biological neural network.

Supervised Training

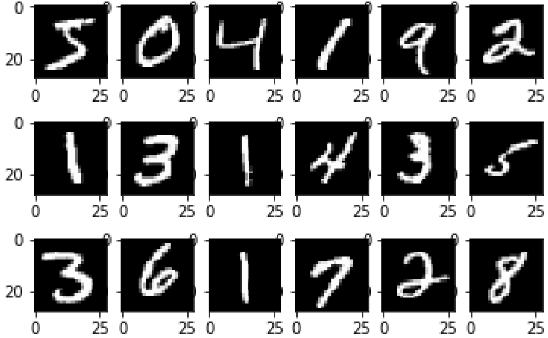
Here is how we will train our artificial neural network:

1. Make a prediction for some input data, whose output we already know.
2. Compare the predicted output to the *ground truth* (actual output).
3. Adjust the *weights/biases* to make the prediction close to the ground truth.
4. Repeat steps 1-3 for some number of iterations.

Problem

From last class. . .

- Input: An 28x28 pixel image
- Output: Whether the digit is a **small** digit (0, 1, or 2)
  - output=1 means that the digit is small
  - output=0 means that the digit is not small



Making Predictions

Code from last class:

```
inval = img_to_tensor(image)
outval = pigeon(inval)        # compute output activation
prob = torch.sigmoid(outval) # turn into a probability
```

How do we convert a (continuous) probability into a (discrete) prediction?

Cut-off at 0.5

We set a threshold at `prob = 0.5`.

For example, in this code:

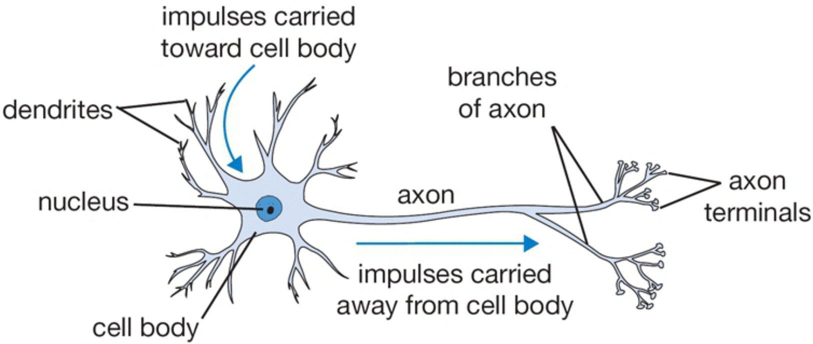
```
error = 0
for (image, label) in mnist_train[:1000]:
    prob = torch.sigmoid(pigeon(img_to_tensor(image)))
    if (prob < 0.5 and label < 3) or \
            (prob >= 0.5 and label >= 3):
        error += 1
```

First Hour of Today

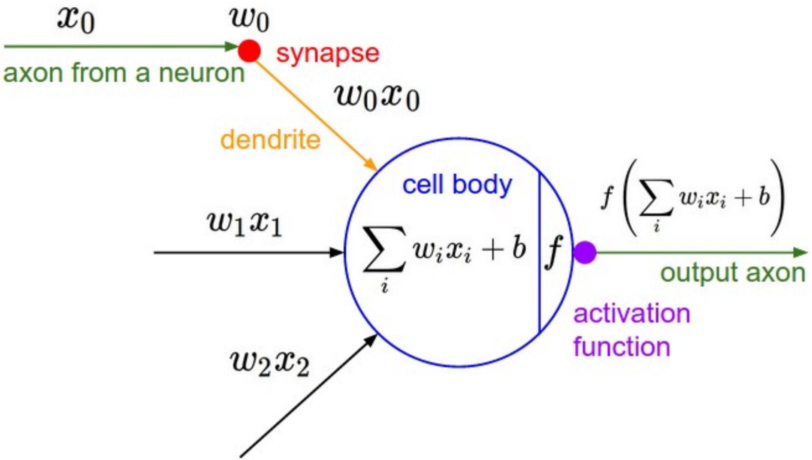
- Start by reviewing the new ideas and terminology
- We'll write more code starting in the second hour

Neural Network Terminology

Review: Biological Neuron



Review: Artificial Neuron

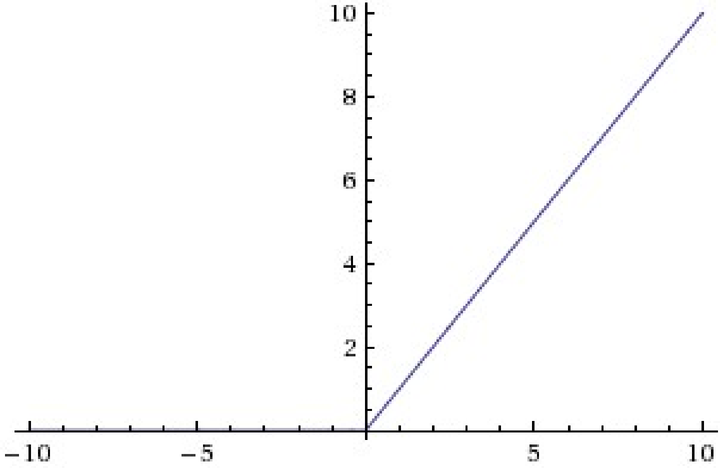


Activation Function

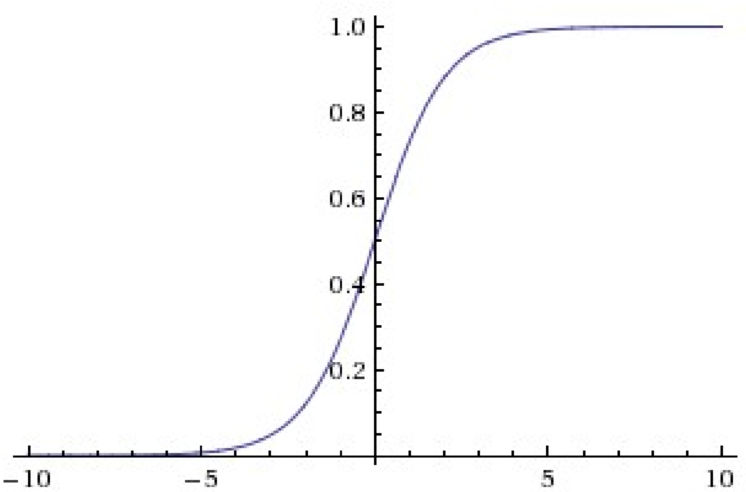
An **activation function** computes the activation of the neuron based on the total contributions from neurons in the layer below.

The activation function should be **nonlinear**. (Why?)

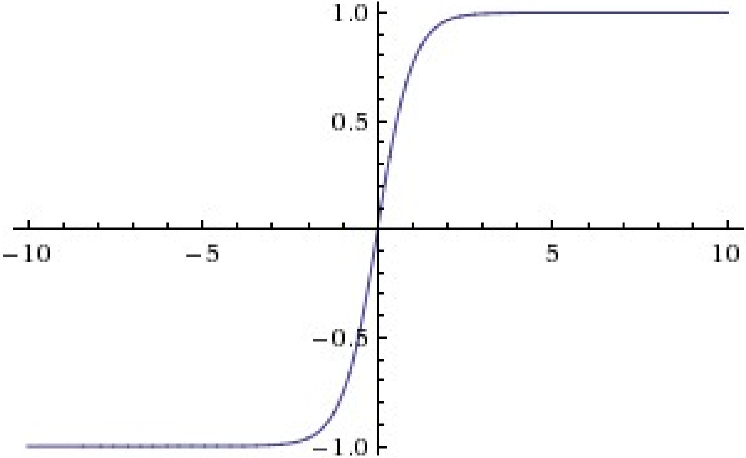
ReLU Activation



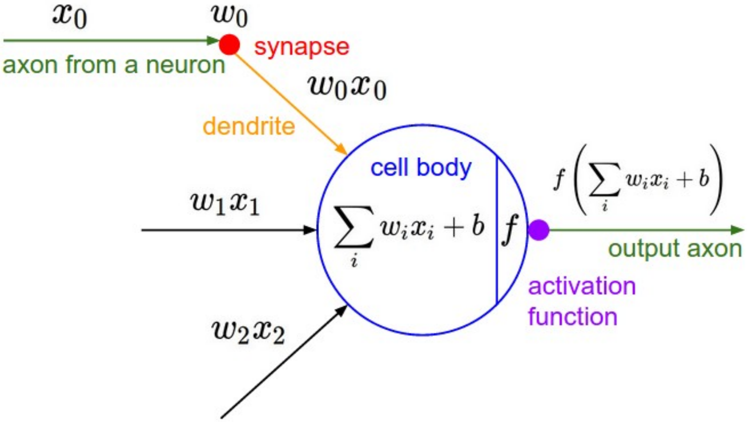
Sigmoid Activation



Tanh Activation



Parameters



The **parameters** of a network are the numbers that can be tuned to train the network. The parameters include the **weights** and **biases**.

We often use **weights** and **parameters** synonymously.

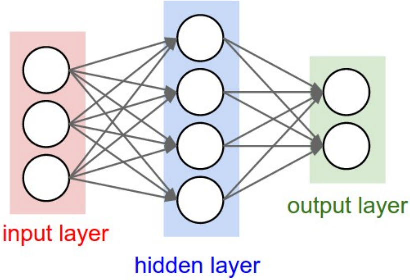
The **number of parameters** of a network is a measure of its size.

Neural Network Architecture

An **architecture** of a neural network describes the neurons and their connectivity in the network.

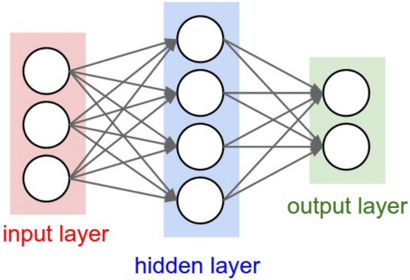
Feed-forward network

Information only flows from one layer to a later layer, from the input to the output.

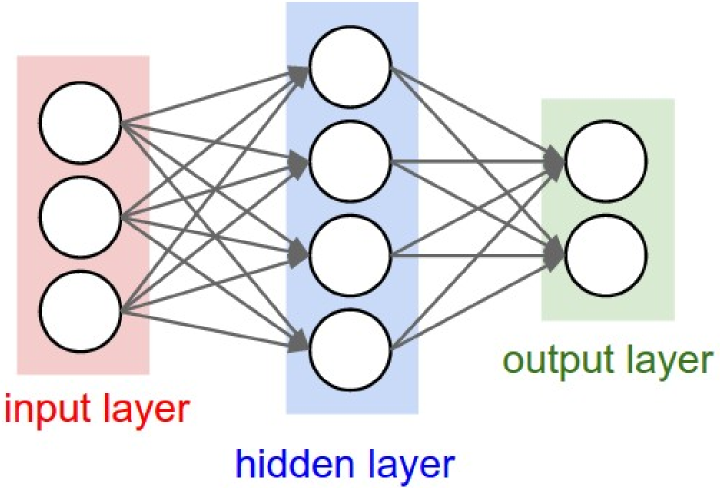


Fully-connected layer

Neurons between adjacent layers are fully pairwise connected.

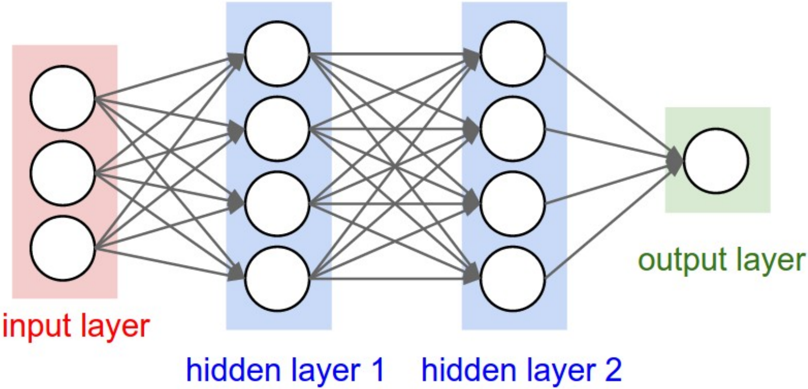


Number of Layers



This is a 2-layer neural network. We do not count the *input layer*, so the number of layers equal number of sets of weights and baises.

Number of Layers



This is a 3-layer neural network.

Training

We **train** a neural network to adjust its *weights*

Loss (Loss Function)

A **loss function** $L(actual, predicted)$ computes how "bad" a set of predictions was, compared to the ground truth.

- Large loss = the network's prediction differs from the ground truth
- Small loss = the network's prediction matches the ground truth

Optimizer

An optimizer determines, based on the value of the **loss function**, how each parameter should change.

The optimizer solves the **credit assignment problem**: how do we assign credit (blame) to the parameters when the network performs poorly?

Optimize Step

We take **one step** towards solving the optimization problem:

$$min_{weights} L(actual, predicted, weights)$$

How do we do this?

Optimize Step

We take **one step** towards solving the optimization problem:

$min_{weights} L(actual, predicted, weights)$

How do we do this?

Using an optimizer like **gradient descent**.

Optimizer: Gradient Descent

All neural network optimizers you see in this course will be based on **gradient descent**.

We use the derivative of the loss function at a training example, and take a step towards its negative gradient.

You don't need to know how optimizers work for this course.

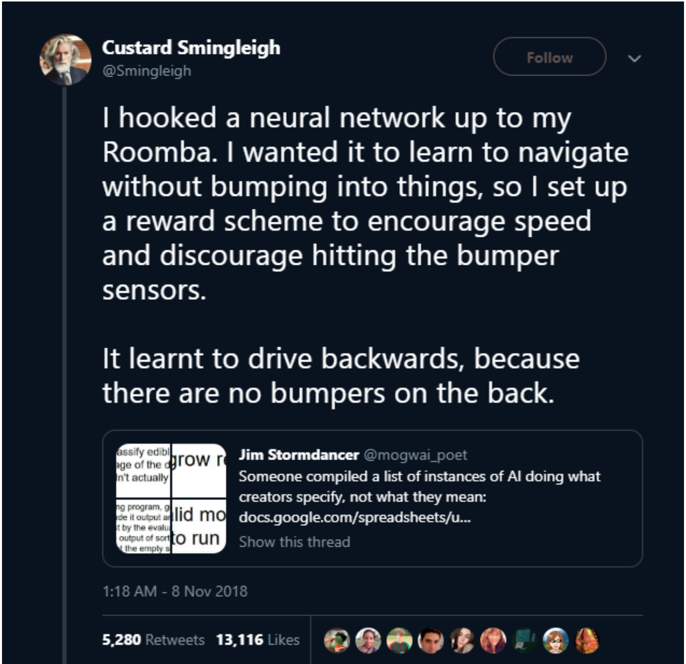
From learning to optimization

Defining a loss function turned a **learning problem** into an **optimization problem**.

- Recurrent theme in Machine Learning

Determining what to optimize is not trivial!

Caveats


Custard Smingleigh
@Smingleigh
Follow
I hooked a neural network up to my Roomba. I wanted it to learn to navigate without bumping into things, so I set up a reward scheme to encourage speed and discourage hitting the bumper sensors.
It learnt to drive backwards, because there are no bumpers on the back.
Jim Stormdancer @mogwai_poet
Someone compiled a list of instances of AI doing what creators specify, not what they mean:
docs.google.com/spreadsheets/u...
Show this thread
1:18 AM - 8 Nov 2018
5,280 Retweets 13,116 Likes

Train, and Test Set

- **Training Set**: Used to tune parameters
- **Test Set**: Used to measure network accuracy

Training and Test Splits

For standard data sets, there are standard train/test splits:

```
mnist_train = datasets.MNIST('data', train=True)
mnist_test = datasets.MNIST('data', train=False)
```

Why?

Neural Network Training

Last week's training code

- We drew motivations from "training" real pigeons
- However, artificial neural networks are unlike biological pigeon in important ways

Summary of last week's training code

1. use our network to make the predictions for **one image**
2. compute the loss for that **one image**
3. take a "step" to optimize the loss of the **one image**

Batching

1. use our network to make the predictions for n **images**
2. compute the *average* loss for those n **image**
3. take a “step” to optimize the *average* loss of those n **image**

Averaging Loss

- Average loss across multiple training inputs is less "noisy"
- Less likely to provide "bad information" because of a single "bad" input

(You can think of the *average loss* as an approximation of the loss across the entire training set.)

Training without batching

```
for (image, label) in mnist_train[:1000]:
    # actual ground truth: is the digit less than 3?
    actual = (label < 3).reshape([1,1]) \
                        .type(torch.FloatTensor)
    # prediction
    out = pigeon(img_to_tensor(image))
    # update the parameters based on the loss
    loss = criterion(out, actual) # compute loss
    loss.backward()         # compute param updates
    optimizer.step()        # make param updates
    optimizer.zero_grad()   # clean up
```

Training without batching (no comments)

```
for (image, label) in mnist_train[:1000]:

    actual = (label < 3).reshape([1,1]) \
                        .type(torch.FloatTensor)
    out = pigeon(img_to_tensor(image))
    loss = criterion(out, actual)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

Training with batching

```
train_loader = torch.utils.data.DataLoader(
                    mnist_train,
                    batch_size=64)
for n, (imgs, labels) in enumerate(train_loader):
    if n >= 10: break
    actual = (label < 3).reshape([1,1]) \
                        .type(torch.FloatTensor)
    out = pigeon(img_to_tensor(image))
    loss = criterion(out, actual)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

Training with batching

```
train_loader = torch.utils.data.DataLoader(
                    mnist_train,
                    batch_size=64)
for n, (imgs, labels) in enumerate(train_loader):
    if n >= 10: break
    actual = (label < 3).reshape([1,1]) \
                        .type(torch.FloatTensor)
    out = pigeon(img_to_tensor(image))
    loss = criterion(out, actual)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

The inside of the loop looks exactly the same!

Let's try it out!

Batch Size

The **batch size** is the number of training examples used per optimization "step".

Each optimization "step" is known as an **iteration**.

The parameters are updated once per iteration.

Q: What happens if the batch size is too small? Too large?

Ineffective Batch Size

- **Too small**:
  - We optimize a (possily very) different function L at each iteration
  - Noisy
- **Too large**:
  - Expensive
  - Average loss might not change very much as batch size grows

Epoch

An **epoch** is a measure of the number of times all training data are used once to update the parameters.

Example:

- There are 1000 images we use for training
- If `batch_size = 10` then 100 iterations $=$ 1 epoch

Optimizer Settings

- The optimizer settings can also affect the speed of neural network training.

```
optimizer = optim.SGD(pigeon.parameters()
                      lr=0.005,
                      momentum=0.9)
```

Learning Rate

The **learning rate** determines the size of the "step" that an optimizer takes during each *iteration*.

Larger step size = make a bigger change in the parameters in each iteration.

Q: What happens if the learning rate is small? Large?

Learning Rate Size

- **Too small**:
  - Parameters don't change very much in each iteration
  - Takes a long time to train the network
- **Too large**:
  - "Noisy"
  - Average loss might not change very much as batch size grows
  - Very large can be detrimental to neural network training

Appropriate Learning Rate

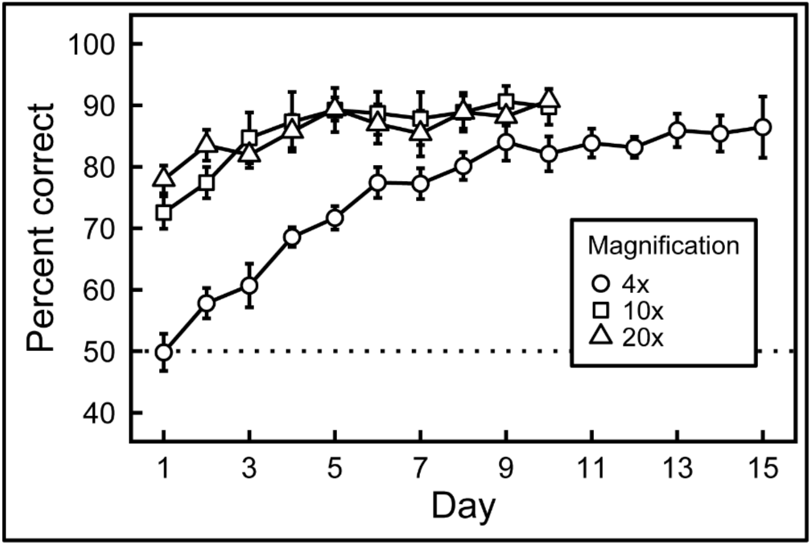
Depends on:

- The learning problem
- The optimizer
- The batch size
  - Smaller learning rate for larger batch size
  - Larger learning rate for smaller batch size
- The stage of training
  - *Reduce* learning rate as training progresses

Tracking Training

- How do we know when to stop training?
- Is training going well?
- Do we have a good batch size?
- Do we have a good learning rate?

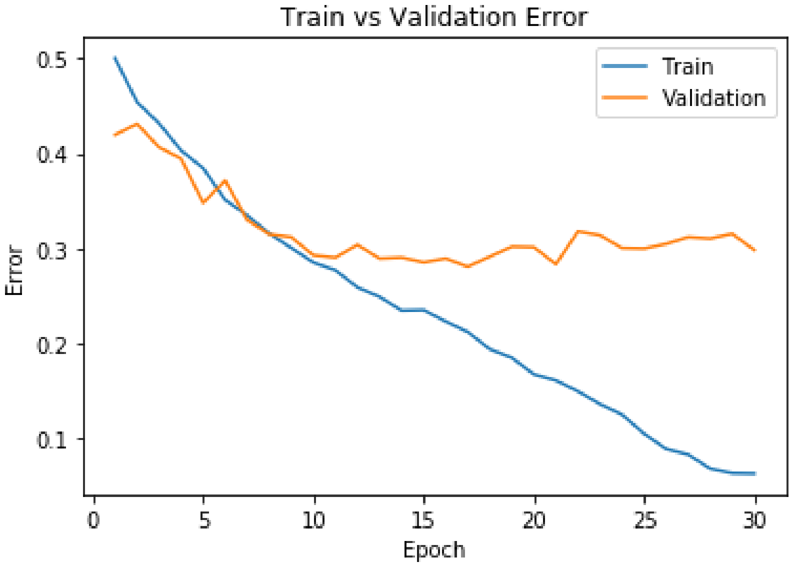
Training Curve for Biological Pigeon



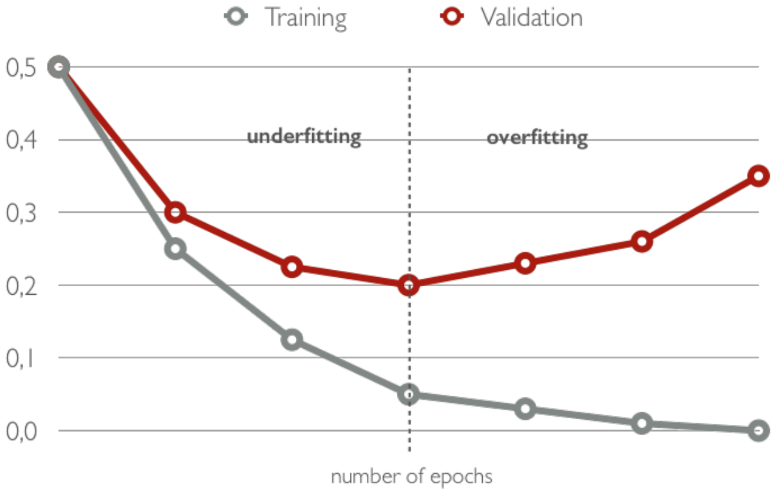
Training Curve

- **x-axis**: epochs or iterations
- **y-axis**: loss, error, or accuracy

Typical Training Curve



Assessing the Fit



Hyperparameters

- Size of network
  - Number of layers
  - Number of neurons in each layer
- Choice of Activation Function
- Learning Rate
- Batch Size

Q: How do we tune hyperparameters?

Lab 2

- Distinguishing cats and dogs
- You have pretty much everything you need to begin assignment 2!