Machine Learning with TensorFlow

https://github.com/lightcycle/MachineLearningWithTensorFlow

Mike Davis (mike.davis@asynchrony.com)

What is Machine Learning?

- Methods for finding structure in or making predictions from data without explicitly programmed logic
- Examples:
  - Image classification
  - Speech to text
  - Natural language processing
  - Spam detection
  - Recommendation engines
- We'll focus on *supervised learning*, finding predictive functions that fit provided input and output data

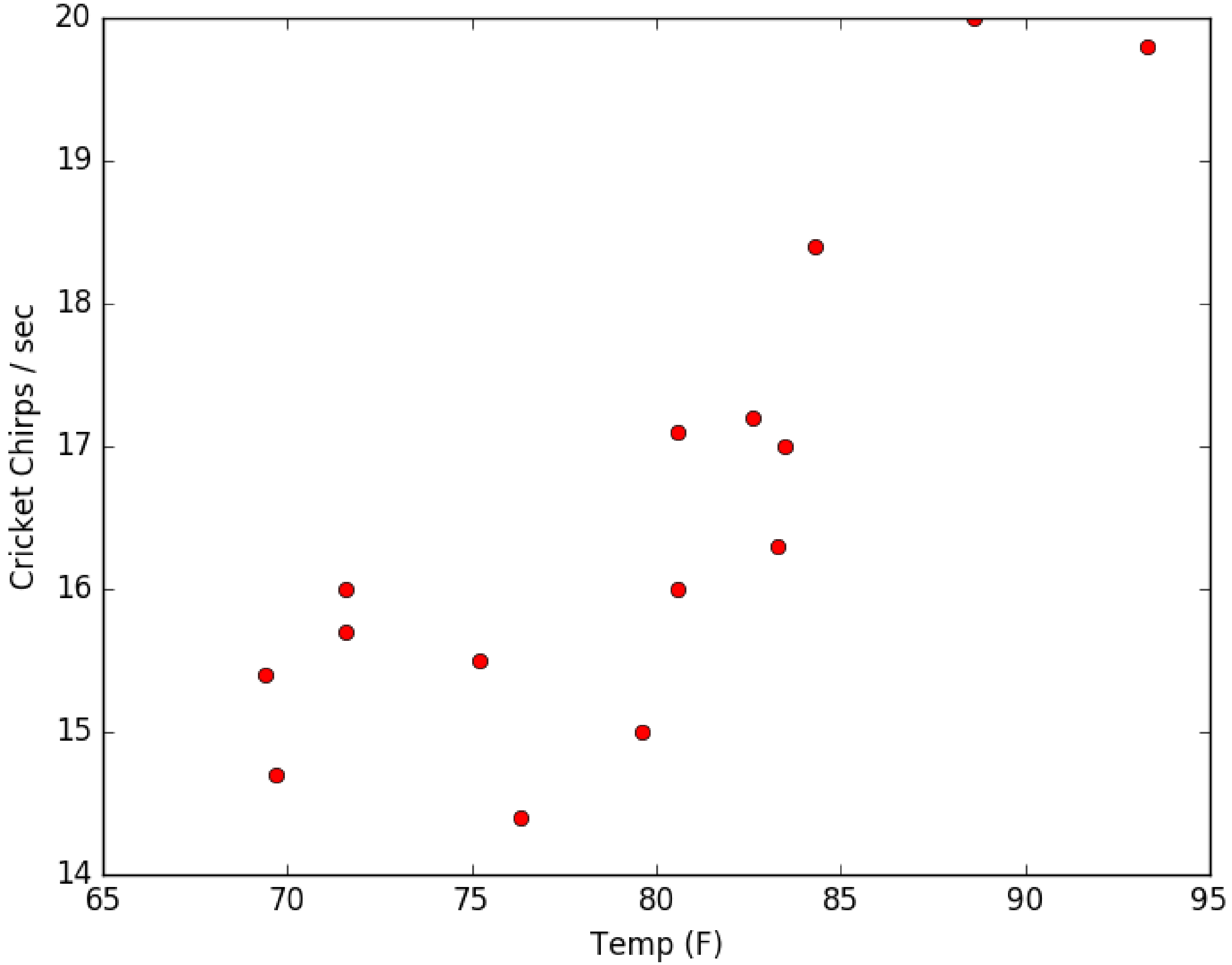
Components of a Supervised Learning Solution

Supervised learning setups commonly include these components:

- A **model function** mapping inputs to outputs, including configurable **parameters** that alter how the model works.
- A **loss function** that quantifies how accurate the model is for given parameters, inputs, and outputs
- An **optimization algorithm** that repeatedly tweaks the parameters and evaluates the loss function to improve the accuracy of the model

Example: Linear Regression

Imagine we have these measurements of cricket chirp frequency at certain temperatures:



Example: Linear Regression

Model

Since we think there's a linear relationship, our model is the equation for a line:

*predicted_outputs = weight * inputs + bias*

Parameters

- weight controls the slope of the line
- bias shifts the line up and down

Example: Linear Regression

Loss Function

Mean squared error (MSE) is a common choice for regression loss functions. For each input temperature, we'll square the difference between the model's predicted chirp frequency and the measured chirp frequency. Then we'll take the average of those squared differences.

$$\text{loss} = \text{average of } (\text{predicted\_output} - \text{output})^2$$

Squaring the differences has the effect of:

- Ensuring that the loss is positive
- Penalizing outliers

Example: Linear Regression

Optimization Algorithm

Our optimization algorithm needs to determine the weight and bias values that result in a line best fitting our data.

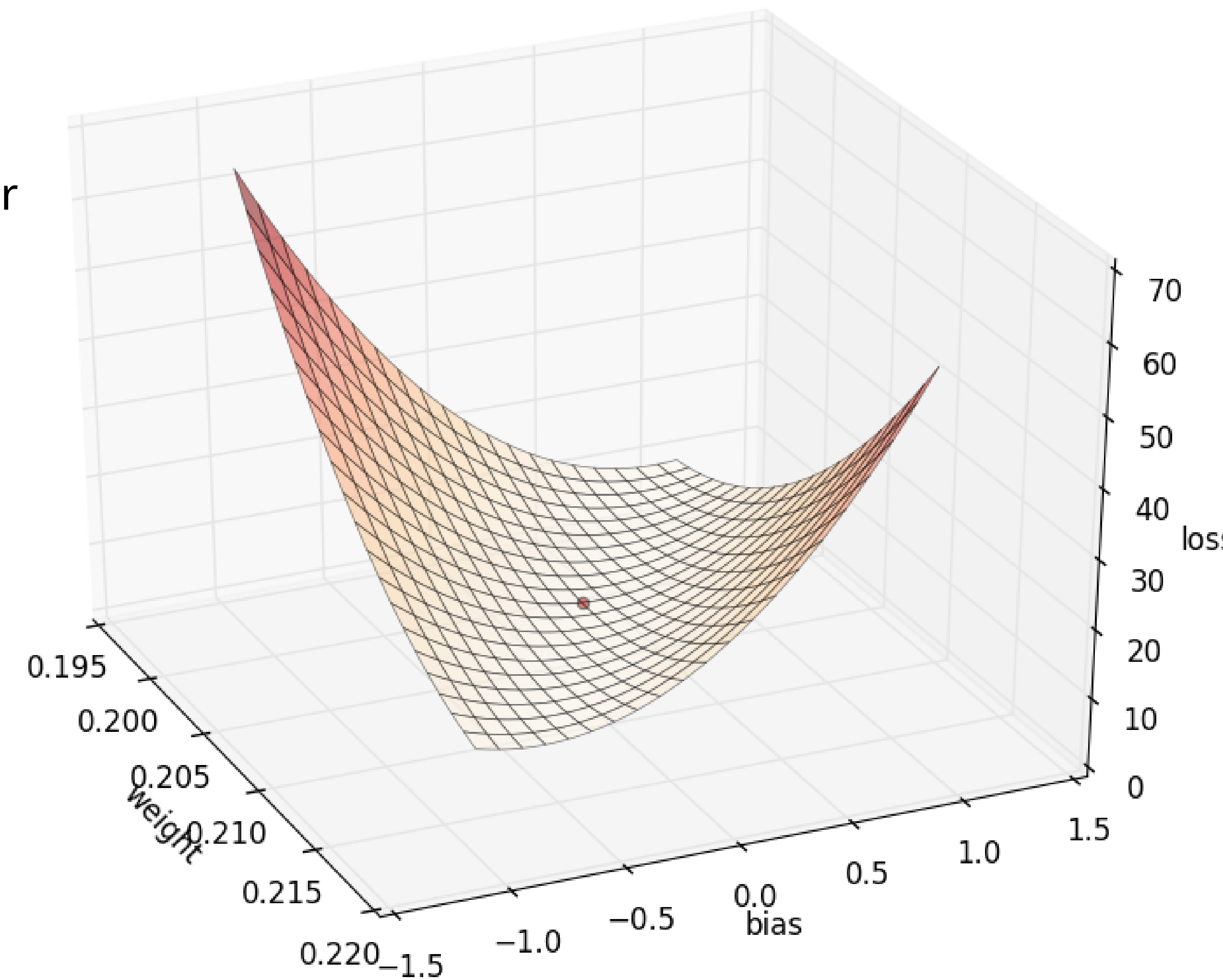
Gradient descent is a common strategy:

1. Start from a reasonable, random set of parameter values
2. Determine the direction most likely to result in improvement, using partial derivatives of the loss function over each parameter
3. Update the parameters slightly in the determined direction and repeat at step 2.

Note: Linear regression is a silly example for machine learning, since we could simply solve for the minimum of the loss function. But this isn't practical for more complex models with thousands of parameters.

Example: Linear Regression

Starting from a random spot on this surface, gradient descent will eventually discover the optimal parameters.



TensorFlow Introduction

TensorFlow is Google's open-sourced library for these kinds of computations, first released Nov 2015.

Similar popular libraries are Theano, Torch, and Caffe. More recently released libraries include CNTK (Microsoft) and DSSTNE (Amazon).

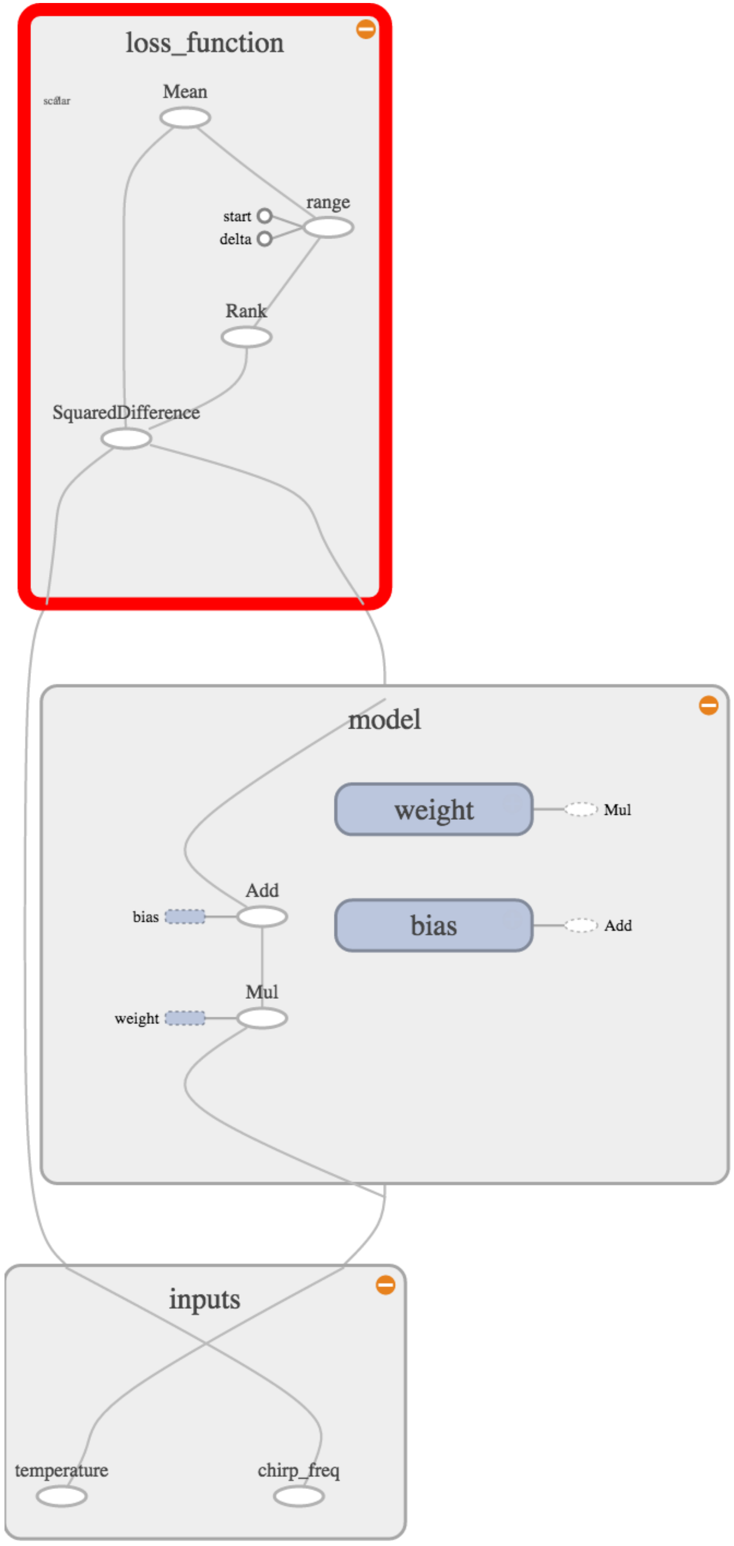
TensorFlow Introduction

- Data is stored in n-dimensional arrays called **tensors**
  - 0-dim for scalars
  - 1-dim for vectors
  - 2-dim for matrices
  - etc
- Computations are defined as a **data flow graph** passing tensors between mathematical operations
  - Graph defined in a high-level language (only Python currently)
  - Graph computed by native code for performance

Example: Linear Regression

Defining the data flow graph:

```
# Placeholders for providing data to graph
X = tf.placeholder(tf.float32,
                   name = "temperature")
Y = tf.placeholder(tf.float32,
                   name = "chirp_freq")
# Model (Linear)
weight = tf.Variable(0., name = "weight")
bias = tf.Variable(0., name = "bias")
modeled_Y = tf.add(tf.mul(X, weight), bias)
# Loss Function (Mean Squared Error)
loss = tf.reduce_mean(
        tf.squared_difference(Y, modeled_Y))
```



Example: Linear Regression

Input to be fed to the graph for training:

```
temp_f = [88.6, 71.6, 93.3, 84.3, 80.6, 75.2, 69.7, 71.6, 69.4,
          83.3, 79.6, 82.6, 80.6, 83.5, 76.3]
cricket_chirps_per_s = [20, 16, 19.8, 18.4, 17.1, 15.5, 14.7,
                        15.7, 15.4, 16.3, 15, 17.2, 16, 17, 14.4]
```

Creating the optimizer operation with set learning rate:

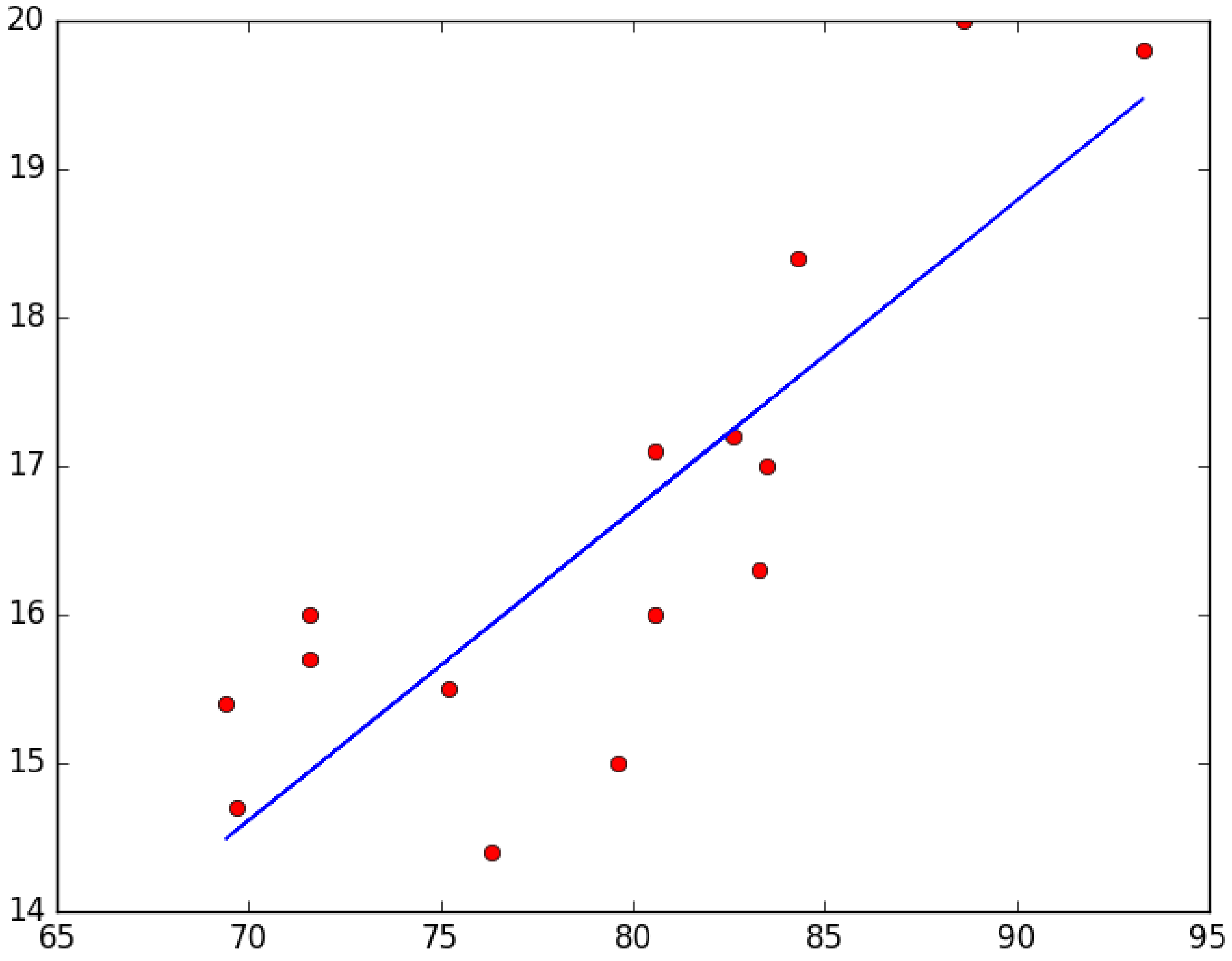
```
training_op = tf.train.GradientDescentOptimizer(0.0001).minimize(loss)
```

Training the model for 50 steps:

```
with tf.Session() as session:
    tf.initialize_all_variables().run()
    for step in range(50):
        session.run([training_op],
                    feed_dict={X: temp_f, Y: cricket_chirps_per_s})
```

Example: Linear Regression

Trained linear model results:



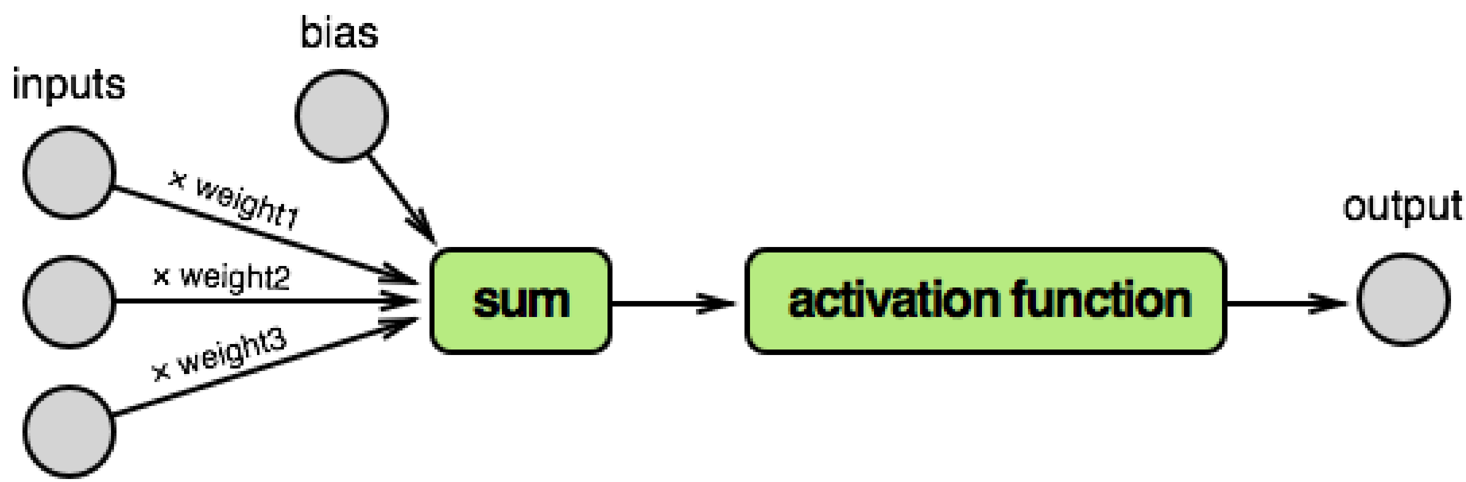
Example: Linear Regression

- TensorFlow takes care of several details for us, by examining the data flow graph:
  - Identifying the trainable weight and bias parameters we created with tf.Variable()
  - Performing automatic differentiation to support gradient descent optimization
  - Distributing execution of the graph over available CPUs and GPUs

- Additional features that take advantage of the graph include:
  - Saving and restoring all parameter values
  - Operations for logging values, images, and audio to Tensorboard
  - Distributing execution over multiple machines
  - Exporting trained model for use in production

Example: Neural Net

Neural net models are composed of nodes inspired by biological neurons:



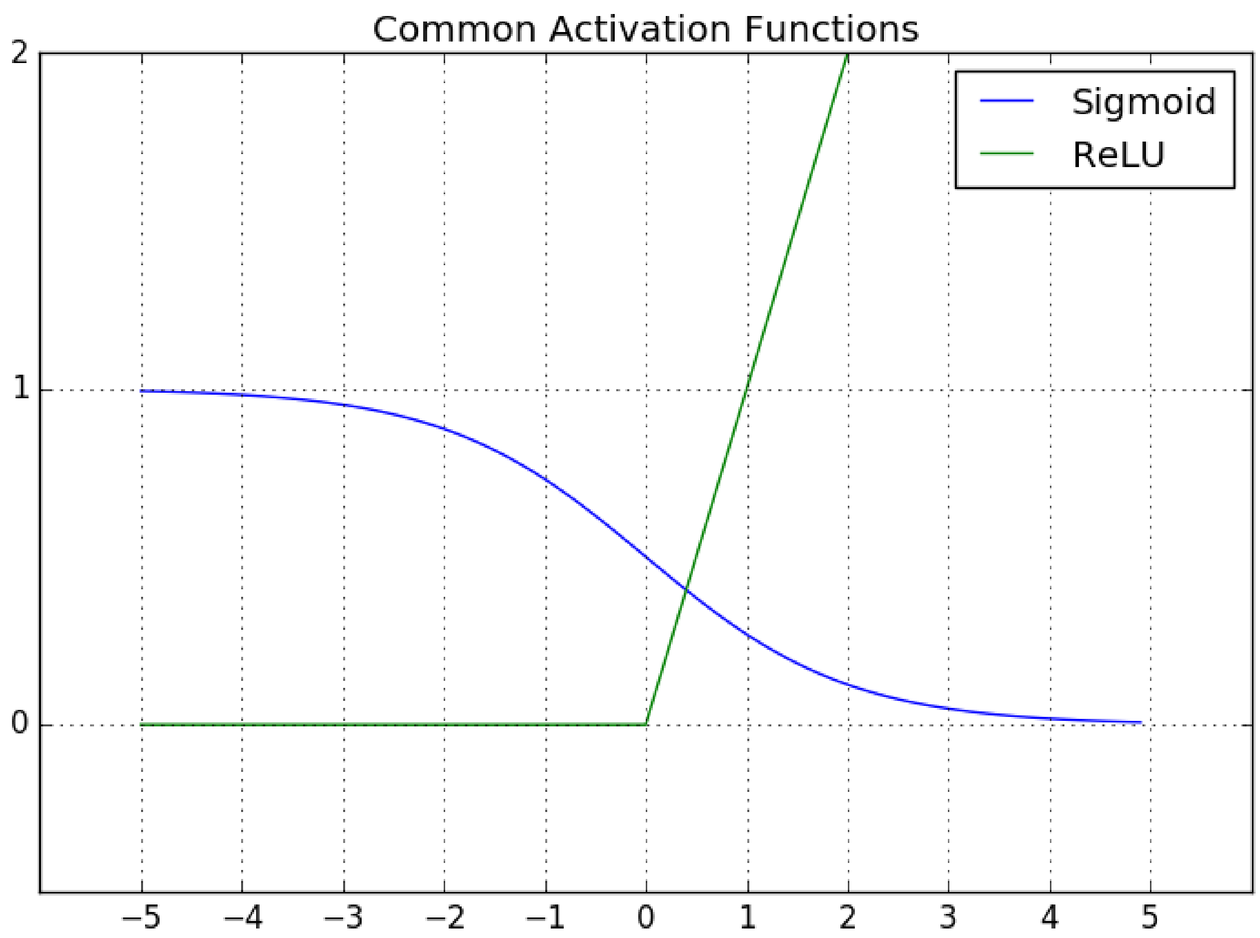
Each input node's value is multiplied by a separate weight parameter, and the result is summed together with a bias parameter.

Finally, the result is transformed by an **activation function**.

Note: The activation function introduces *non-linearity*. Without it, the math performed by the node (or even a network of nodes) could be algebraically reduced to a simple linear equation. Which would produce a convoluted linear regression model.

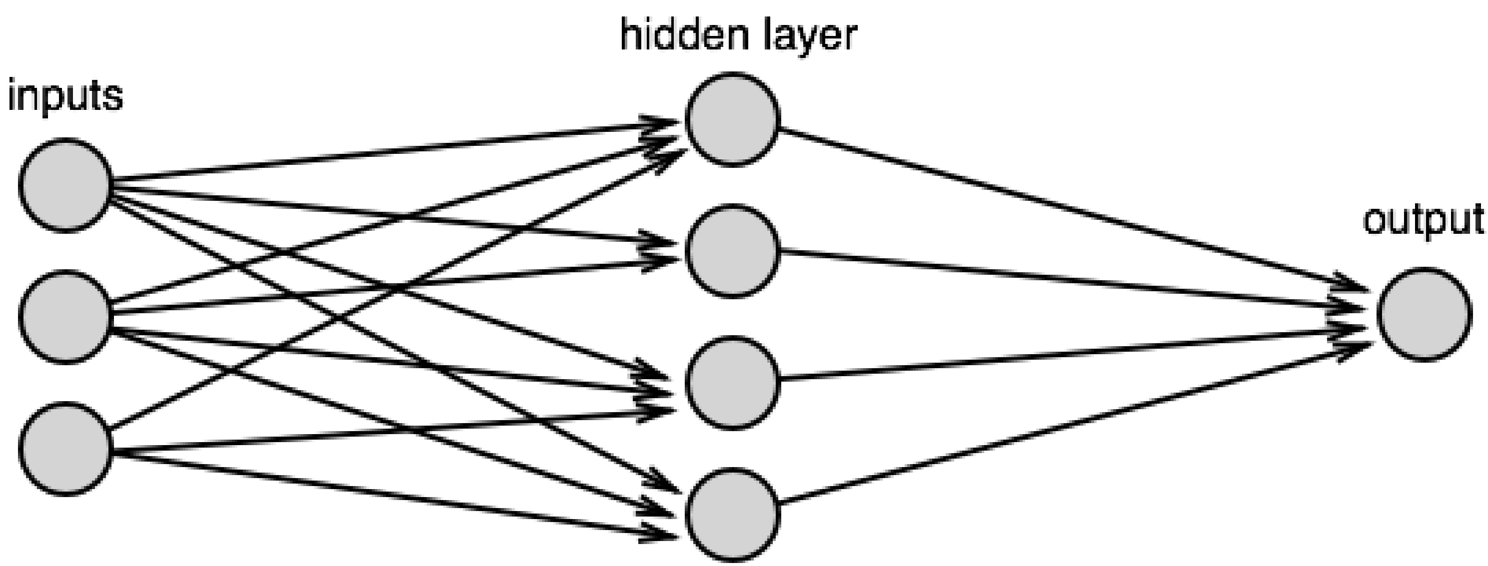
Example: Neural Net

The activation function can be anything, but two common choices are the *sigmoid* and *rectified linear units (ReLU)* functions.



Example: Neural Net

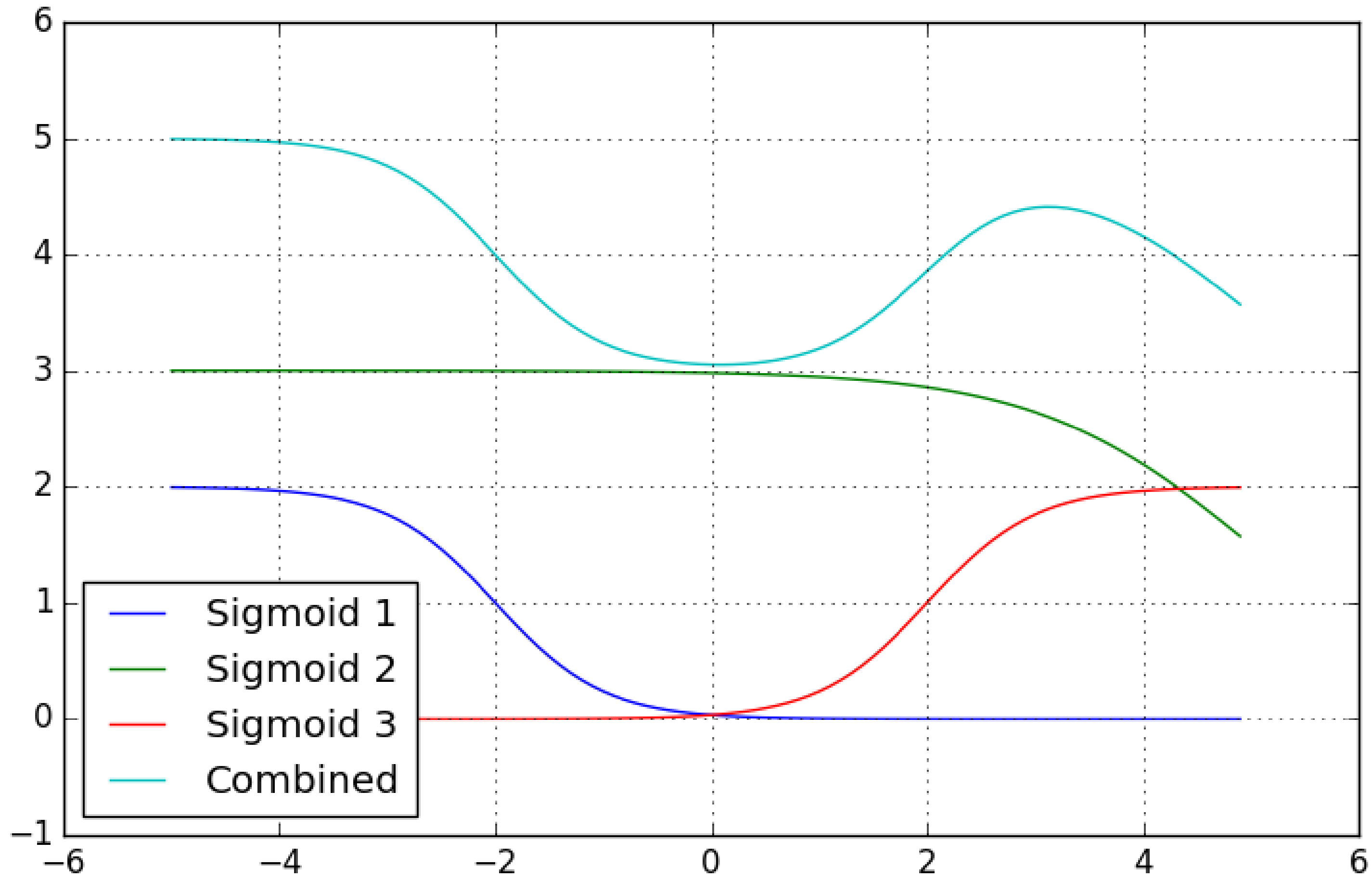
Neural networks are constructed with one or more hidden layers of nodes between the input nodes and output nodes.



With enough nodes and an appropriate activation function, a single layer can approximate any function.

Example: Neural Net

Each hidden layer essentially combines instances of the activation function shape, scaled and shifted by node weights and bias.



Example: Neural Net

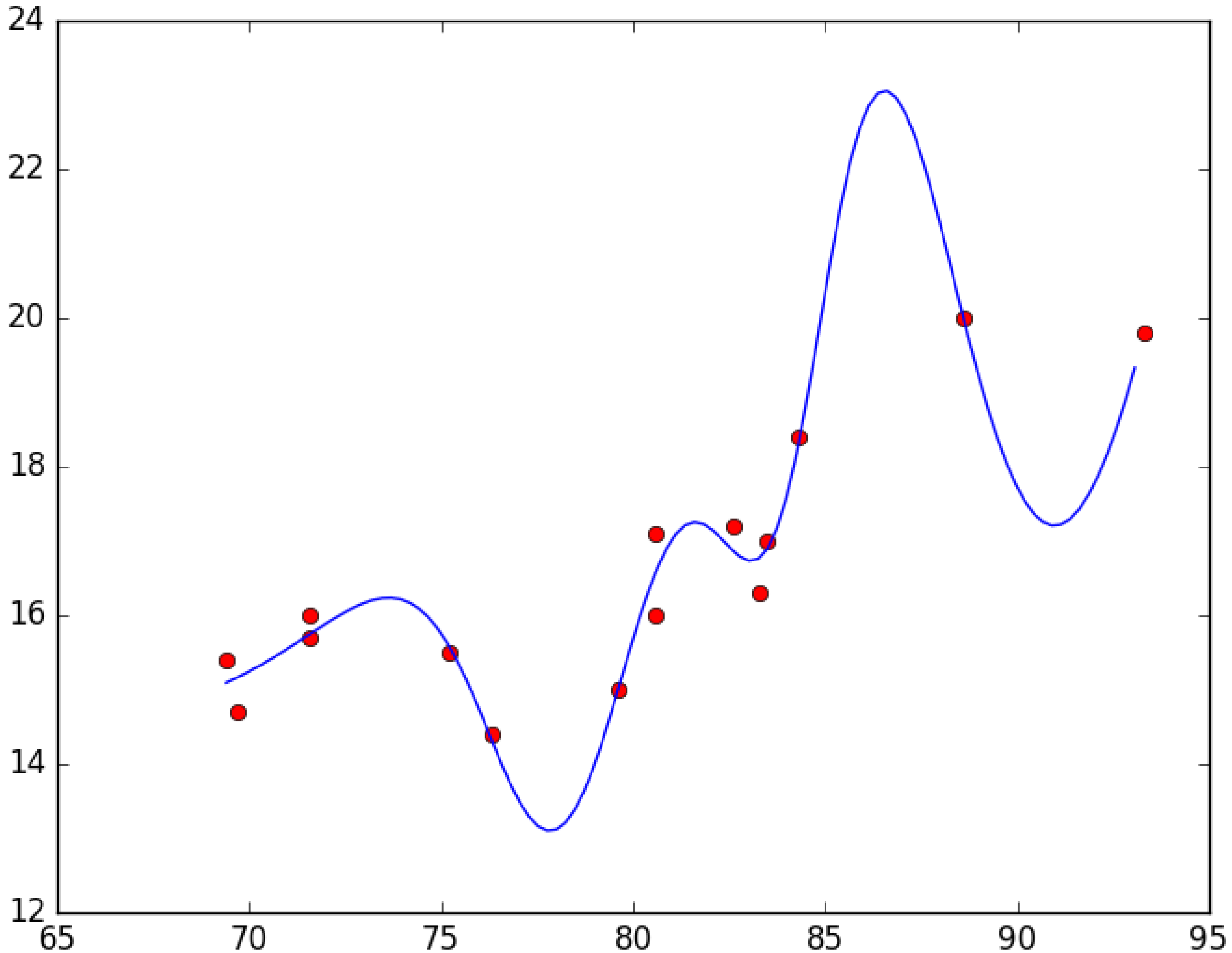
Model

Neural net data flow graph definition (*weight()* and *bias()* functions generate variables):

```
# Model (Neural Net with one hidden layer)
input_layer = tf.expand_dims(X, 1)
hidden_layer_nodes = 10
hidden_layer_weight = weight([1, hidden_layer_nodes])
hidden_layer_bias = bias([1, hidden_layer_nodes])
hidden_layer = tf.nn.sigmoid(tf.matmul(input_layer, hidden_layer_weight)
               + hidden_layer_bias)
output_layer_weight = weight([hidden_layer_nodes, 1])
output_layer_bias = bias([1])
modeled_Y = tf.matmul(hidden_layer, output_layer_weight)
            + output_layer_bias
```

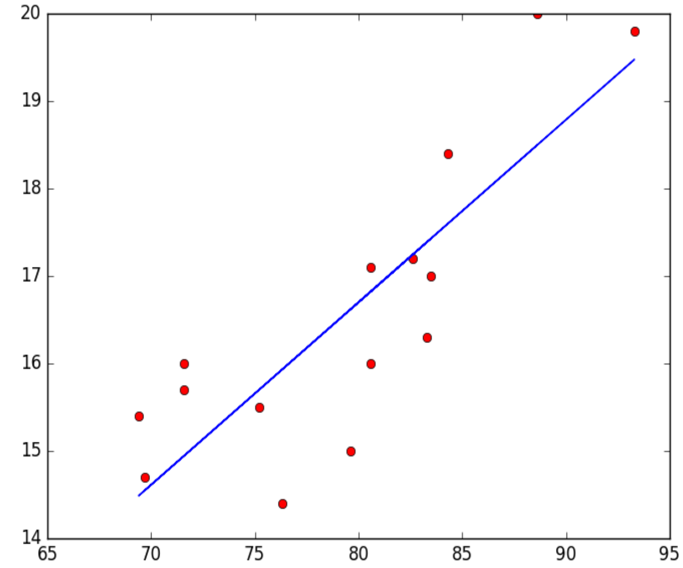
Example: Neural Net

Trained neural net model results:



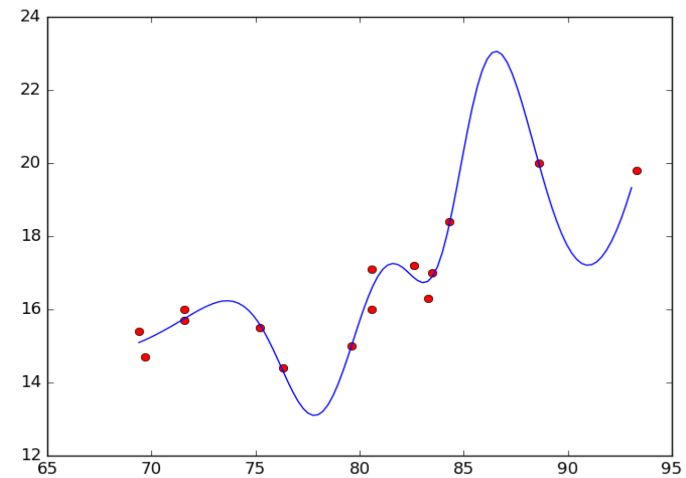
Overfitting and Testing

The neural net model fit the data more closely, but that doesn't make it a better choice of model for this data.



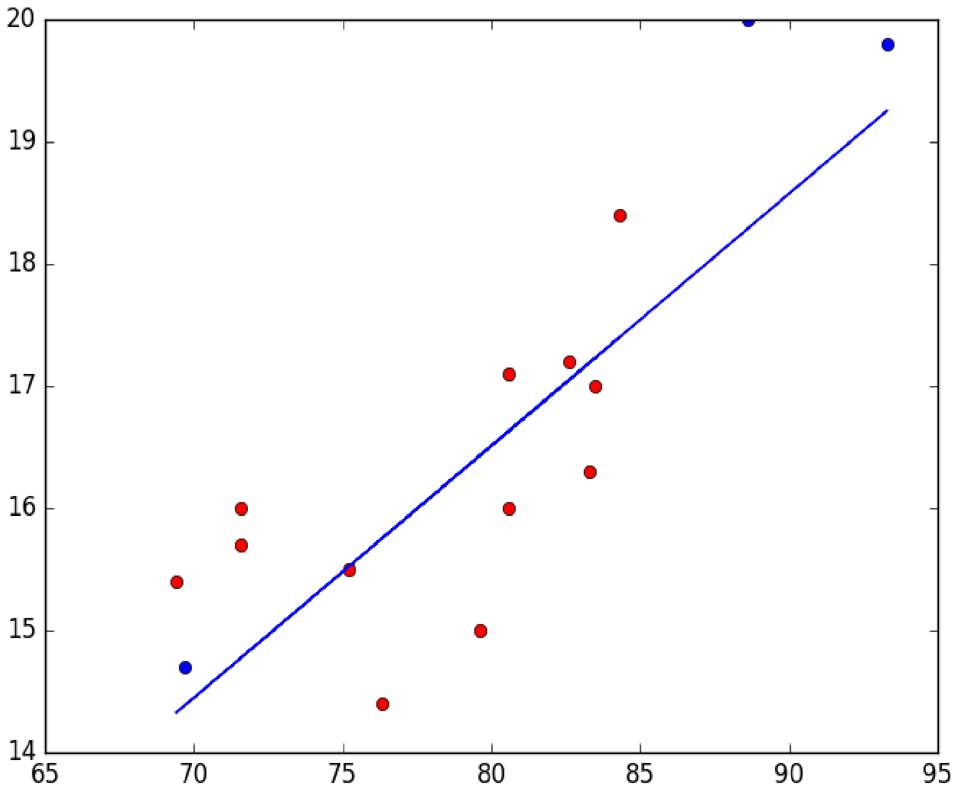
Our goal is to train a model that makes reasonable predictions.

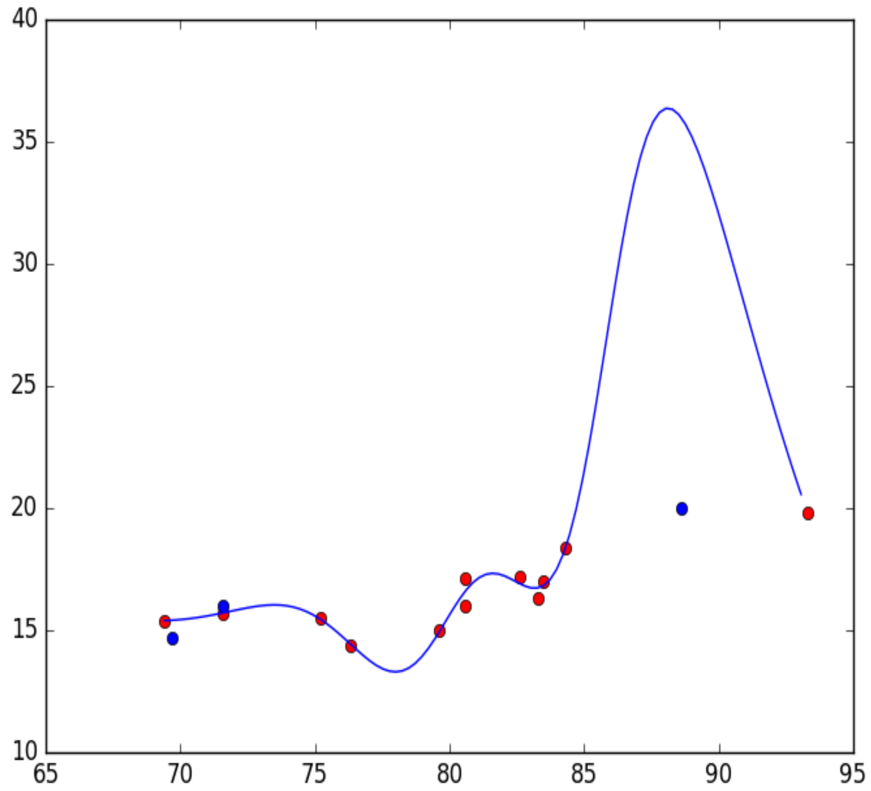
There's always the risk of training a model that **overfits** the data. Such models essentially memorize the answers to the test, but can't answer new questions.



Overfitting and Testing

To detect overfitting, split the available data into training and testing data sets. After training the model with the training data, we can check to loss on the testing data to determine how well the trained model is generalizing.





Example: Convolutional Neural Net

Machine learning can be applied to computer vision problems as well.

For this example, we'll train a model that can determine which direction is upwards in an input photograph.

This problem has a couple of significant differences from the previous two examples:

- The output needs to be a selection from a set of classes (up, right, down, left), rather than a number. This is a **classification** problem.
- The input needs to be an image.

Example: Convolutional Neural Net

The model will have four output numbers, one for each class. The classes will be encoded as **one-hot vectors**:

- [1, 0, 0, 0] is up
- [0, 1, 0, 0] is right
- [0, 0, 1, 0] is down
- [0, 0, 0, 1] is left

Our model is likely to output wide-ranging numbers. The **softmax** function is commonly used for normalization in classification problems. After normalization, we'll consider the closest class match to be the model's selection.

softmax([1,2,3,4]) = [0.0320586, 0.08714432, 0.23688284, 0.64391428] →

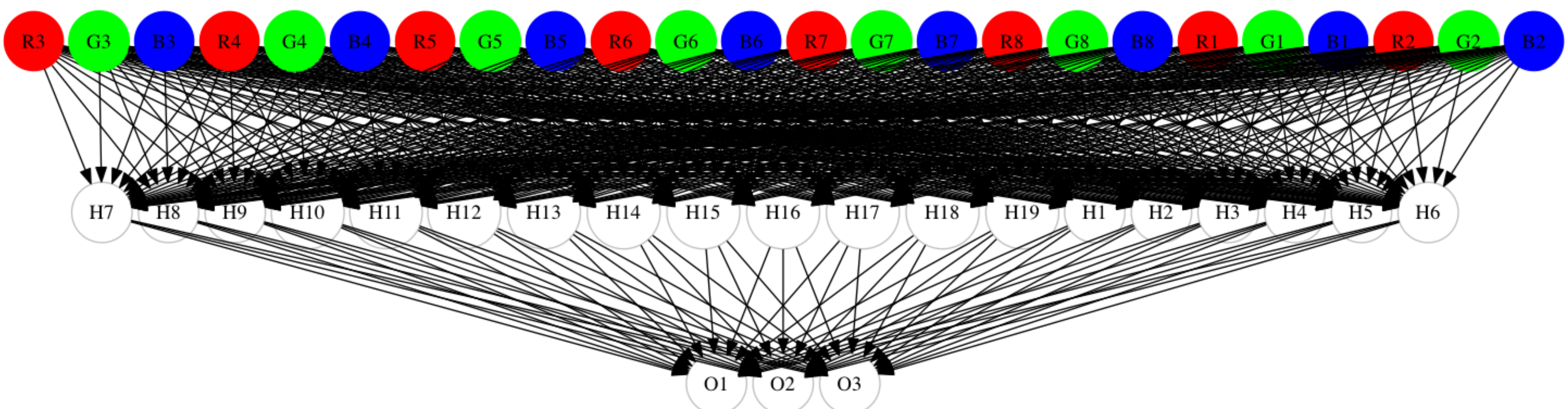
[0, 0, 0, 1] → left

Example: Convolutional Neural Net

To process images, we could have a separate input for the color channel value at each pixel. But this approach has disadvantages:

- Many inputs (width × height × 3 for color) means more computational complexity
- Discards information about the values' relative locations to each other

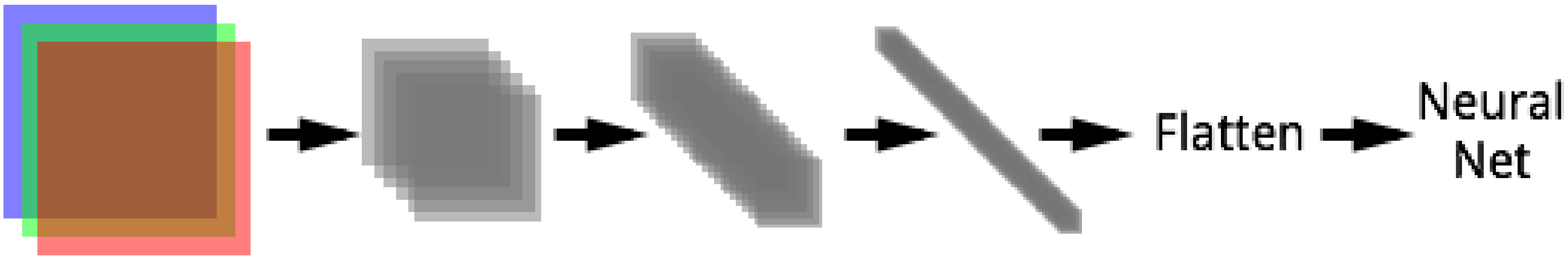
For a tiny 3 × 3 RGB image input to a neural net:



Example: Convolutional Neural Net

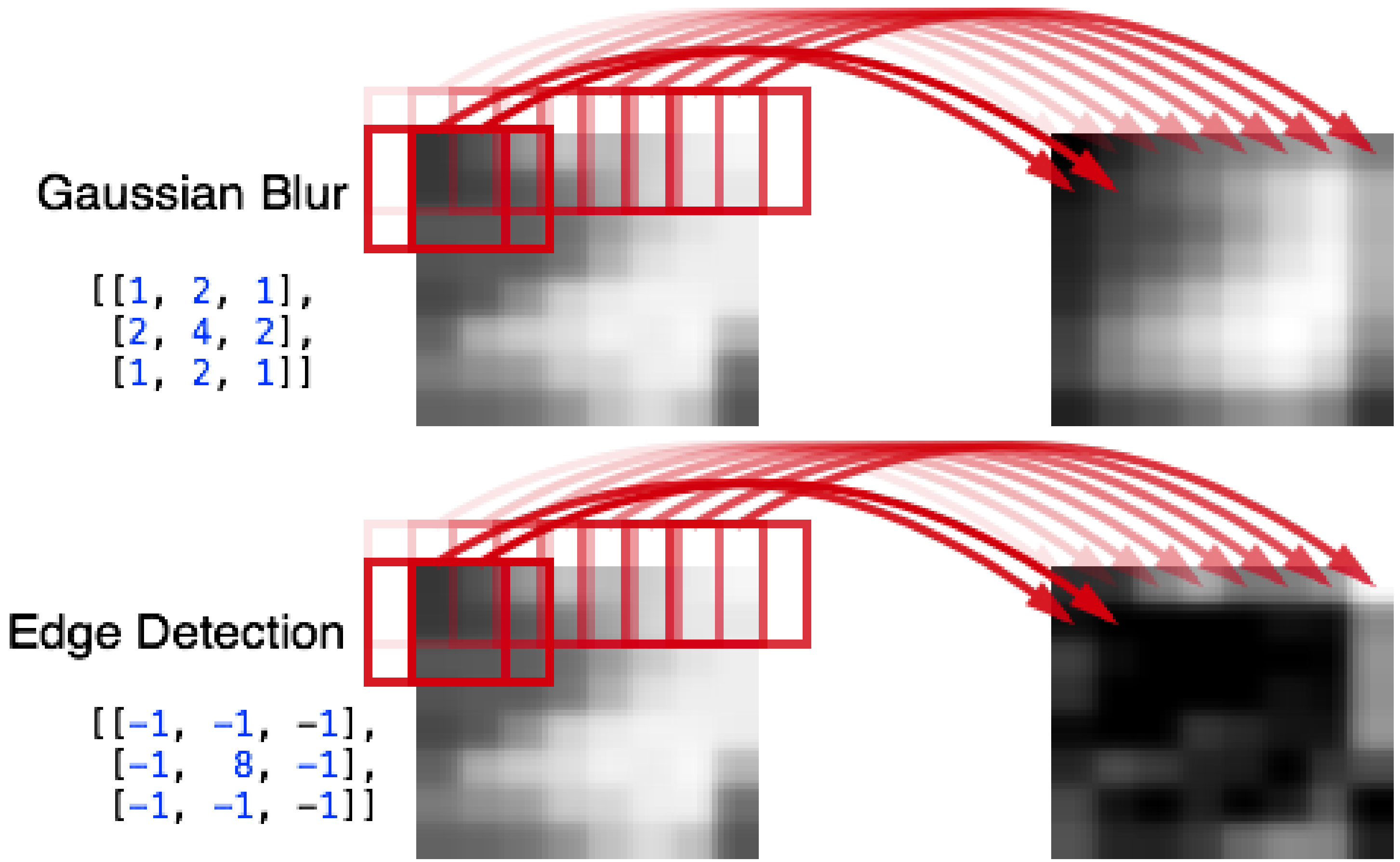
Convolutional neural nets are a newer approach to handling images:

1. Sweep (convolve) across the input image's layers (R, G, and B initially), multiplying with small matrices called **kernels** to create a new set of images
2. Optionally, scale the new images.
3. Either repeat step 1 with the new images and additional kernels, or
4. Flatten the remaining images into numeric inputs, and pass them into a regular neural net.



Example: Convolutional Neural Net

Many of the filters in graphics programs like Photoshop are implemented as convolutions using 2D matrix kernels applied to each color channel.



Example: Convolutional Neural Net

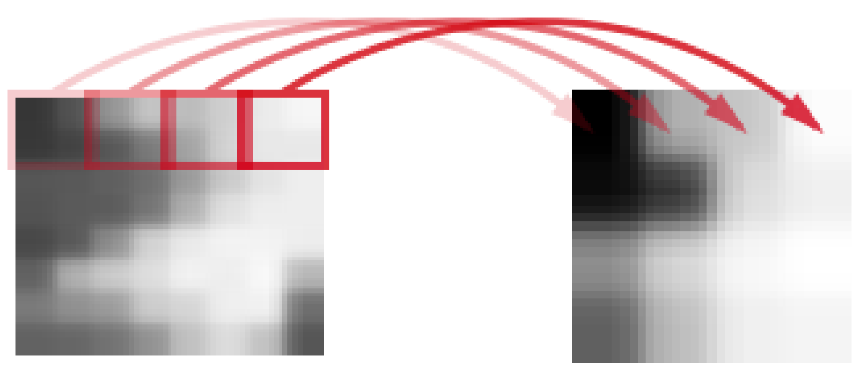
We’ll be using 3D matrix kernels, to produce a new grayscale image from multiple input images. Below the input images are the RGB components. But they could be multiple grayscale images as well.

Example: Convolutional Neural Net

Pooling is commonly used after convolution to scale down the output images. Pooling converts each n×n block of values to a single value, often by taking the maximum or average.

Pooling offers two major benefits:

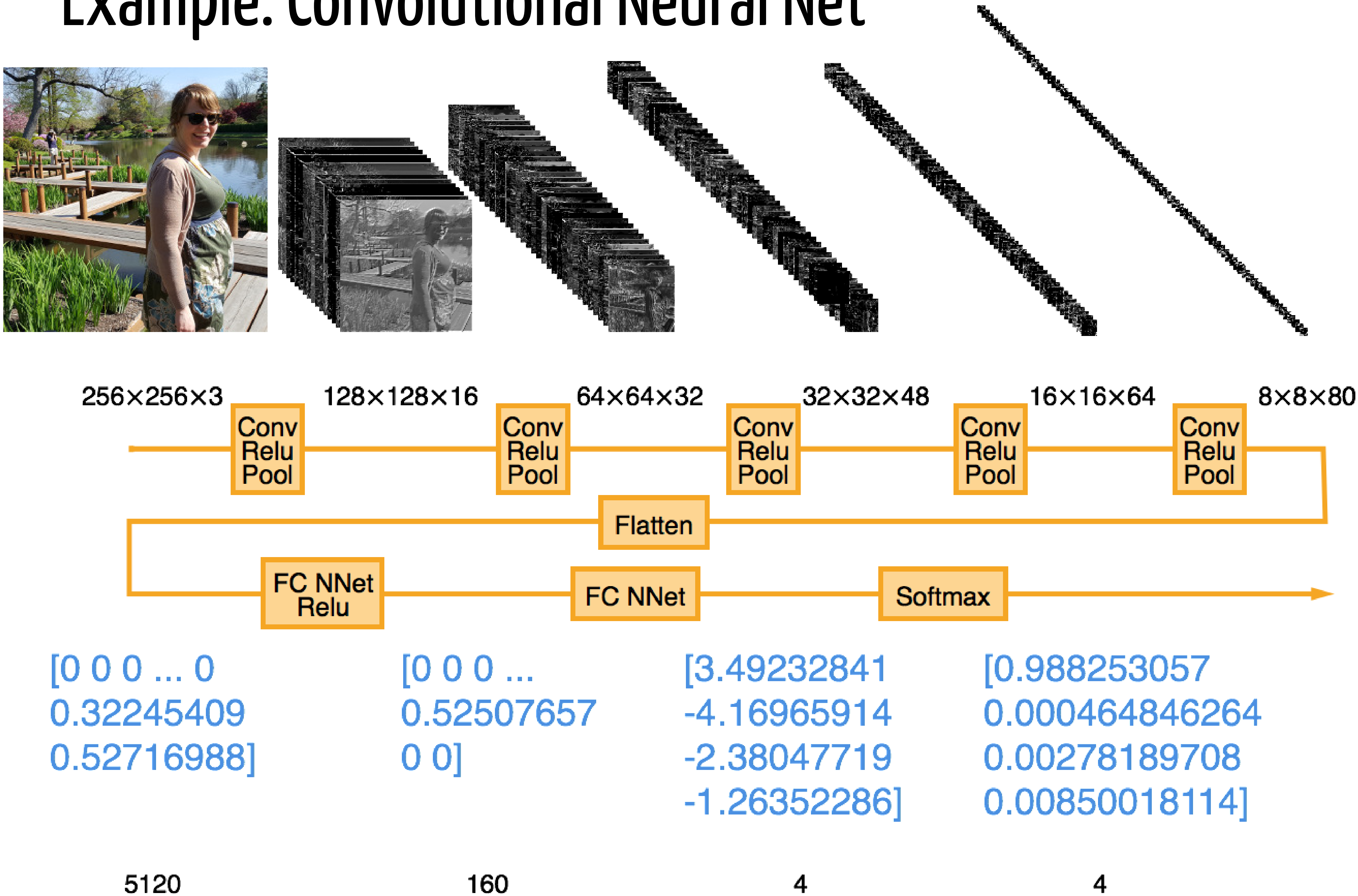
- Reducing the size of the inputs to the next layer reduces computational cost
- Adds some **translation invariance**



Example: Convolutional Neural Net

Convolution and pooling layers are typically stacked together. As images pass through the multiple filters, complex features can be identified. This is where the term **deep learning** comes from.

Example: Convolutional Neural Net



Example: Convolutional Neural Net

Training a model this complex requires a great deal of training data. I used 195000 images from the Microsoft COCO datasets.

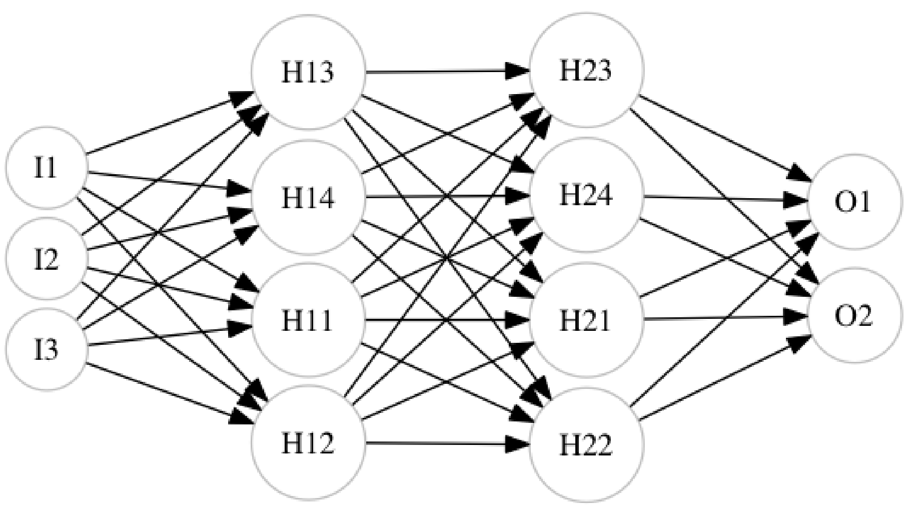
Processing that many images during each training step isn't feasible. So we use **batching** instead, shuffling the image order and using 100 for each training step.

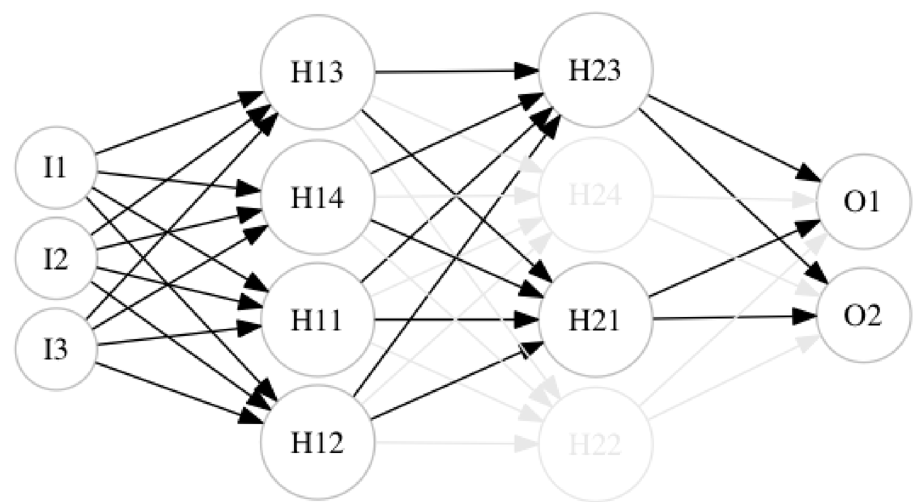
The full training dataset can be used multiple times. Each full use is called an **epoch**.

Example: Convolutional Neural Net

One last trick is to apply **dropout** to a layer of the neural net.

Dropout disables a random subset of neurons during each training step. This ends up having a protective effect against overfitting.





Example: Convolutional Neural Net

Some convenience methods for building the model data flow graph:

```
# Create weight tensor, with random initial normal values
def __weight_variable(cls, shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

# Create bias tensor, with constant initial values
def __bias_variable(cls, shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)
```

Example: Convolutional Neural Net

Some more convenience methods for building the model data flow graph:

```
def __conv_layer(cls, input_shape, input,
                 k_width, k_height, num_outputs):
    weight = cls.__weight_variable([k_width, k_height,
                                    input_shape[2], num_outputs])
    bias = cls.__bias_variable([num_outputs])
    return tf.nn.conv2d(input, weight, strides=[1, 1, 1, 1],
                        padding='SAME') + bias,\
           (input_shape[0], input_shape[1], num_outputs)

def __relu_layer(cls, input_shape, input):
    return tf.nn.relu(input),\
           input_shape
```

Example: Convolutional Neural Net

Even more convenience methods for building the model data flow graph:

```
def __maxpool_layer(cls, input_shape, input, pool_size):
    return tf.nn.max_pool(input, ksize=[1, pool_size, pool_size, 1],
                          strides=[1, pool_size, pool_size, 1],
                          padding='SAME'),\
           (input_shape[0] / pool_size, input_shape[1] / pool_size,
            input_shape[2])

def __fully_connected_layer(cls, input_shape, input, num_outputs):
    flattened_size = reduce(mul, input_shape, 1)
    weight = cls.__weight_variable([flattened_size, num_outputs])
    bias = cls.__bias_variable([num_outputs])
    input_flat = tf.reshape(input, [-1, flattened_size])
    return tf.matmul(input_flat, weight) + bias,\
           (num_outputs,)
```

Example: Convolutional Neural Net

Last of the convenience methods for building the model data flow graph:

```
def __dropout(cls, input_shape, input, keep_prob):
    return tf.nn.dropout(input, keep_prob), \
           input_shape

def __softmax(cls, input_shape, model):
    return tf.nn.softmax(model), \
           input_shape
```

Example: Convolutional Neural Net

Building the model data flow graph:

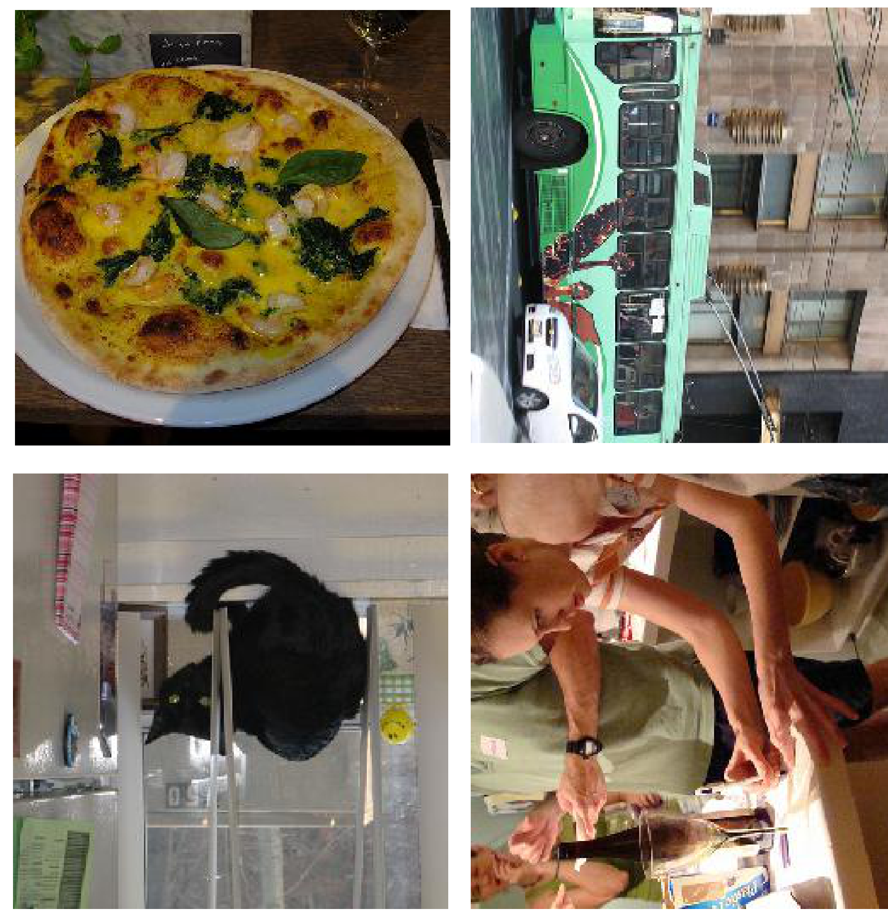
```
def create_graph(cls, input, keep_prob):
    # Normalize [0,255] ints to [0,1] floats
    input_float = tf.div(tf.cast(input, tf.float32), 255)

    # Build model
    model = input_float
    shape = (256, 256, 3)
    model, shape = cls.__conv_layer(shape, model, 3, 3, 16)
    model, shape = cls.__relu_layer(shape, model)
    model, shape = cls.__maxpool_layer(shape, model, 2)
    model, shape = cls.__conv_layer(shape, model, 3, 3, 32)
    model, shape = cls.__relu_layer(shape, model)
    model, shape = cls.__maxpool_layer(shape, model, 2)
    model, shape = cls.__conv_layer(shape, model, 3, 3, 48)
    model, shape = cls.__relu_layer(shape, model)
    model, shape = cls.__maxpool_layer(shape, model, 2)
    model, shape = cls.__conv_layer(shape, model, 3, 3, 64)
    model, shape = cls.__relu_layer(shape, model)
    model, shape = cls.__maxpool_layer(shape, model, 2)
    model, shape = cls.__conv_layer(shape, model, 3, 3, 80)
    model, shape = cls.__relu_layer(shape, model)
    model, shape = cls.__maxpool_layer(shape, model, 2)
    model, shape = cls.__fully_connected_layer(shape, model, 160)
    model, shape = cls.__relu_layer(shape, model)
    model, shape = cls.__dropout(shape, model, keep_prob)
    model, shape = cls.__fully_connected_layer(shape, model, 4)
    model, shape = cls.__softmax(shape, model)

    return model
```

Example: Convolutional Neural Net

After training this model with 40 epochs of 195000 images, the model's accuracy is roughly 83% when testing with 40000 new images.

Pass

Fail



Performance Considerations

Machines with one or more GPUs with CUDA support (specifically Nvidia chips) greatly reduce the training time for complex models. GPUs have hundreds or thousands of cores capable of performing calculations in parallel.

If you have a gaming rig at home, you may be well equipped to try machine learning with complex models.

I'm on a MacBook, so I used an Amazon EC2 instance with a single, modest GPU. With a g2.2xlarge instance, training took approximately 12 hours. Many ABI images that include TensorFlow are available; I used "Bitfusion Ubuntu 14 TensorFlow".

Building New Models

Throughout these examples, several “magic number” values have been left unexplained:

- Learning rate
- Number of training steps or epochs
- How many and which layers to use in a convolutional neural network
- Which activation function to use
- Which loss function to use
- How much training data is needed

These choices are referred to as **hyperparameters**.

How do we know what to do?

Building New Models

A few strategies for building new models:

- ~~Copy~~ Research published models
  - Numerous working models for computer vision, language processing, audio processing, and other problems are available in public academic papers
- Rules of thumb and current fashion
  - These change rapidly, but as interest grows there are more sources than ever
  - E.g. convolutional neural net best practices:
    - Deeper models and more training data always help
    - Use small kernels, ReLU, small pooling, softmax, and cross-entropy
- Experiment, in parallel if possible
  - During model development, try different sets of hyperparameters

Applying Machine Learning to Customer Problems

The most expensive part of building a supervised learning solution is often collecting enough data for training and testing.

In academic settings, hordes of unfortunate graduate students can be exploited to manually label data.

In business settings, look for situations where ample data is already available:

- Identifying abusive / pornographic social media content based on user reporting
- Flagging spam based on user deletion
- Predicting user behavior based on browsing statistics
- Recommendations based on users' past choices

Any case where a lot of data is available!

Ethical Considerations

We need to be especially cautious when building predictive models that have a significant, real-life impact on individuals.

For example, imagine if HR requested a model that filter new candidates by comparing their resumes to the resumes of existing employees.

Without care, the trained model might make unintended inferences and perpetuate unfortunate biases in an attempt to make the hiring pool homogenous:

- Inferring age based on education dates
- Inferring nationality or race based on name
- Inferring gender based on name
- Inferring socio-economic status based on location

Resources

- Links
  - https://www.oreilly.com/learning/hello-tensorflow
  - http://cs231n.github.io/convolutional-networks/
  - https://github.com/jtoy/awesome-tensorflow
- Books
  - TensorFlow For Machine Intelligence (Sam Abrahams, Danijar Hafner, Erik Erwitt, Ariel Scarpinelli
  - Data Science from Scratch: First Principles with Python (Joel Grus)
  - Programming Collective Intelligence (Toby Segaran)