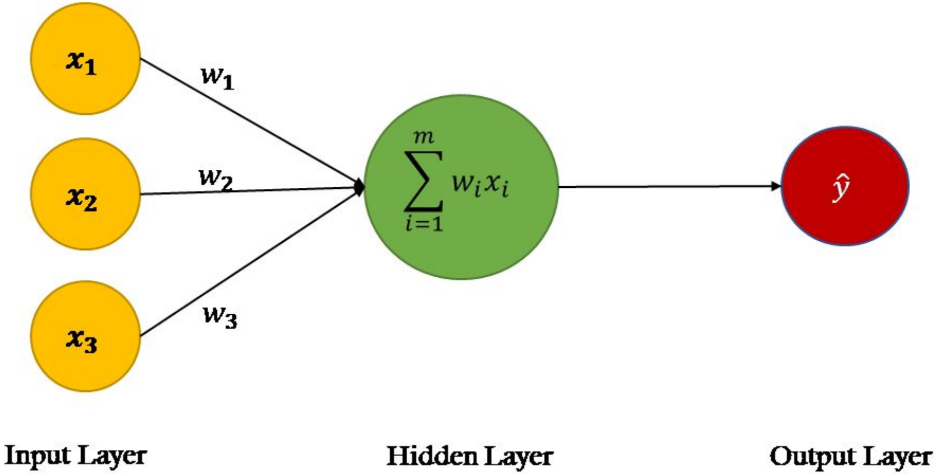
Keras/Apache Spark

Mehul Raheja, Andrew Lin,
Anirudhan Badrinath

Keras

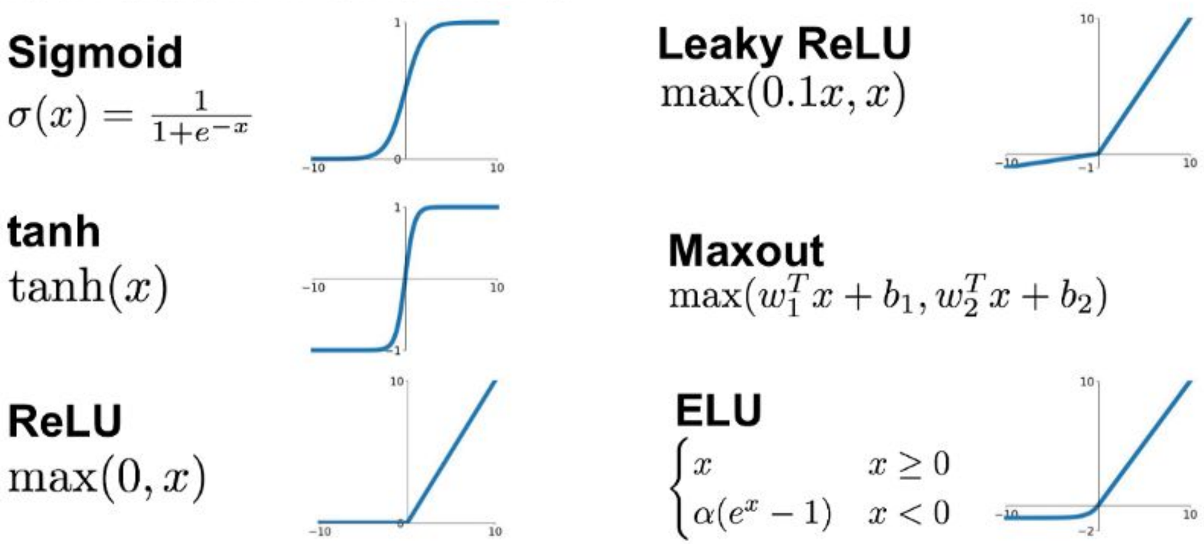
Neuron



- Takes in any number of inputs and creates a weighted average of them
- Can apply an "activation function" on weighted sum to induce non-linearity

towardsdatascience.com

Activation Function



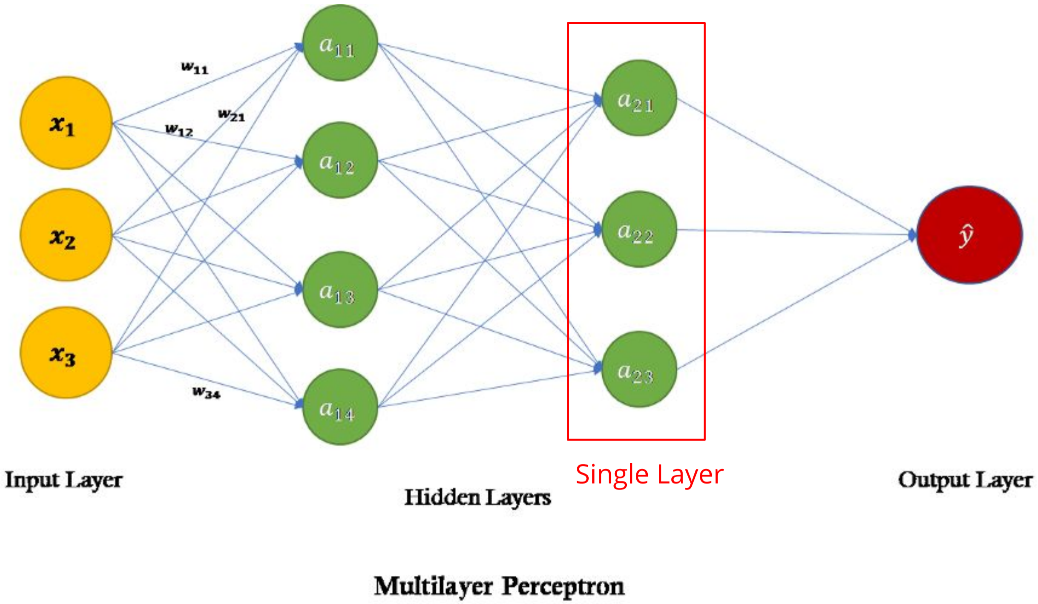
RELU: Typical in most neural networks

Sigmoid/tanh: Typically used in final layer

LeakyReLU: Solves vanishing gradient problem

https://medium.com/@shrutijadon10104776

Neural Network



- With many neurons, organized in layers, you can theoretically model highly non-linear behavior
  - *Using what kind of function?*

towardsdatascience.com

But how do we create a neural network that models our data?

Actual Solution:

- Partial Differentiation
- Backpropagation
- Gradient Descent
- Optimization Algorithms
- Learning Rates
- etc.

Easier Solution:

K Keras

Example Problem: Pythagorean Distance

$$(\text{x,y}) \rightarrow \sqrt{x^2 + y^2}$$

Can we model this with linear regression (OLS)?
What kinds of features could we possibly use to model this relationship?

Training Data

X

```
array([[1.81677971, 3.06033363],
       [2.14512171, 5.4058173 ],
       [6.49606482, 8.07453646],
       ...,
       [4.65661968, 6.94611321],
       [6.21610875, 7.49327311],
       [5.12444694, 6.7375422 ]])
```

y

```
array([[ 3.55897884],
       [ 5.8158755 ],
       [10.36325225],
       ...,
       [ 8.36257111],
       [ 9.73597195],
       [ 8.46489406]])
```

Creating the Neural Network

```
#Initialize our Model

model = Sequential()

#Add input layer

model.add(Dense(units=4, activation='relu', input_shape=[2]))

#Add hidden layers

model.add(Dense(units=10, activation='relu'))

model.add(Dense(units=5, activation='tanh'))

#Add output layer

model.add(Dense(units=1))
```

Creating the Neural Network

```
#Initialize our Model

model = Sequential()

#Add input layer

model.add(Dense(units=4, activation='relu', input_shape=[2]))

#Add hidden layers

model.add(Dense(units=10, activation='relu'))

model.add(Dense(units=5, activation='tanh'))

#Add output layer

model.add(Dense(units=1))
```

Why do we have the activations for these fully-connected layers?

Compiling and Training the Model

```
#Compile the model

model.compile(optimizer='sgd',  loss='mse')

#Train the model

model.fit(X, y, epochs = 10)
```

Losses:
- MSE
- MAE
- Binary CE
- Categorical CE

Optimizers:
- Adam
- SGD
- RMSProp
- AdaGrad

Compiling and Training the Model

```
Epoch 1/10
313/313 [==============================] - 0s 1ms/step - loss: 0.3406
Epoch 2/10
313/313 [==============================] - 0s 1ms/step - loss: 0.3402
Epoch 3/10
313/313 [==============================] - 0s 1ms/step - loss: 0.3414
```

...

Testing the Model

- Create new X_test with a randomly sampled distribution of (x,y) pairs
- Create y_test based on X_test
- Use:

```
y_pred = model.predict(X_test)
```

- Compute MSE between y_pred and y_test

Observations

- Required a lot of training data (compared to linear regression + feature engineering)
- The activation function/epochs/batch size were all arbitrarily decided
    - Sometimes there are intuitive approaches to this but a lot of the times, you test various combinations through *Hyper Parameter Optimization*
    - Domain knowledge can help with these decisions!
        - i.e. in weather forecasting, we may want to use an exponential activation function at some point because of the unlikely nature of extreme events

Potential Improvements

- validation_split = 0.2
  - Allows you to calculate “validation error” at each epoch
  - Will allow us to do hyper-parameter optimization
- optimizer = ‘Adam’
  - Generally solves problems faster than SGD
- GPU Utilization
  - Designed to do matrix operations faster and can therefore train faster
  - Read about CUDA!

Classification

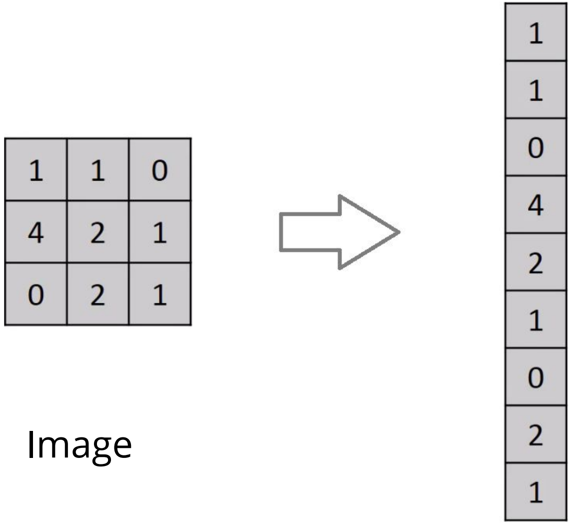
Softmax Layer

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- Takes any input and outputs a probability distribution: [0.1 0.8 0.05 0.05]
  - Motivation: The training outputs can now just be one-hot encoded vectors of the class
  - One-Hot Encoding entails placing a 1 where we have a true output: [0 1 0 0]
- To classify image, choose class with highest probability

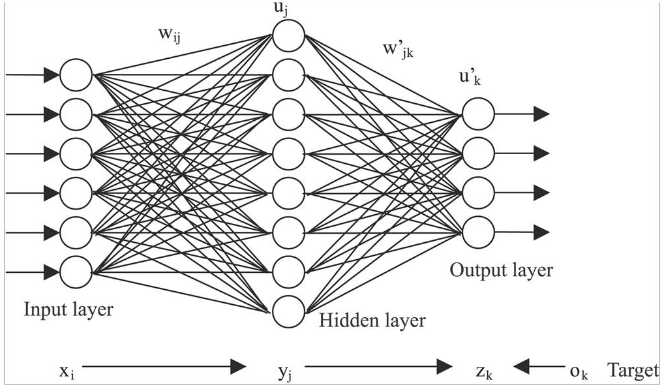
Image Classification: Basic

But what happens when we shift the image by one pixel? Yikes.



Image

Flattened

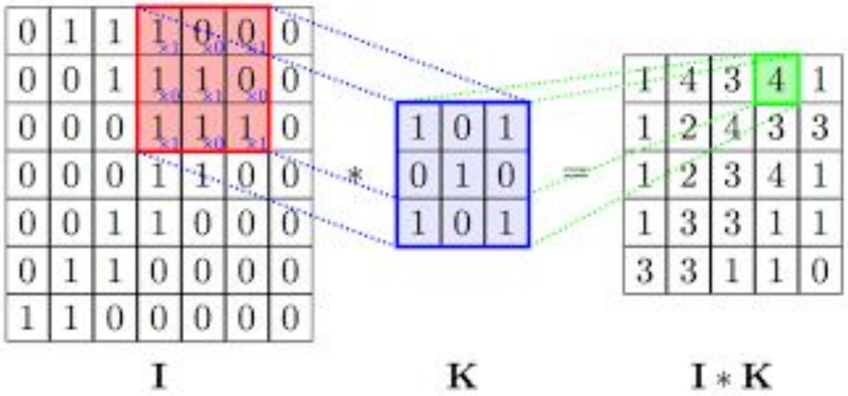


Fully Connected Layers

https://medium.com/@cdabakoglu

Image Classification: Convolutional Neural Network

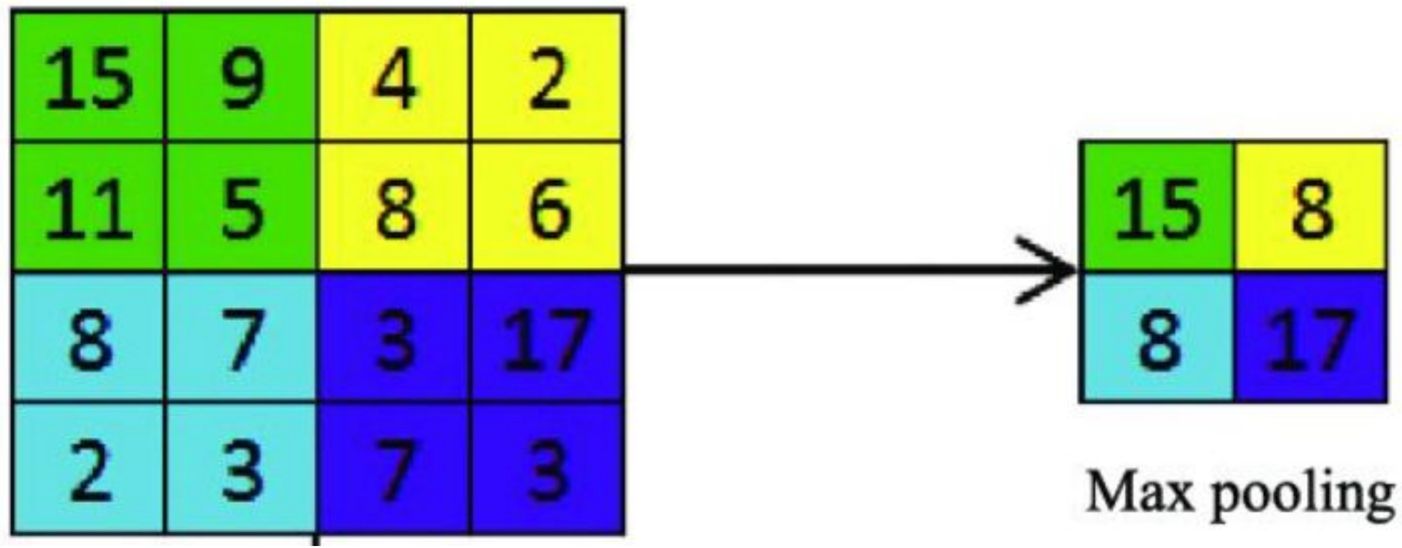
Convolution… a fancier term for pairwise multiplication and summing



https://medium.com/@cdabakoglu

Image Classification: Convolutional Neural Network

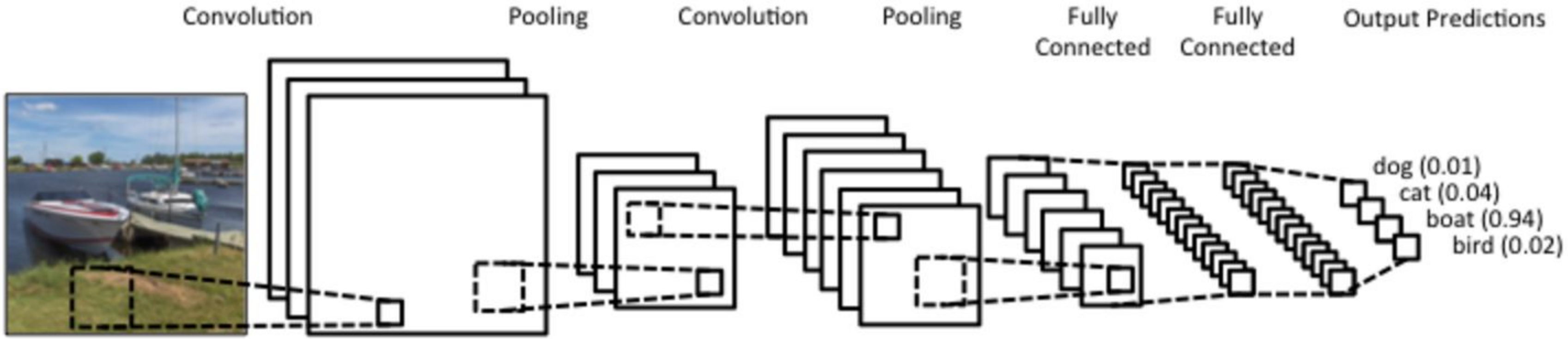
Pooling… a fancy word for applying a function on a block of values to combine or "pool" them; in this case, we apply the `max` function



https://medium.com/@cdabakoglu

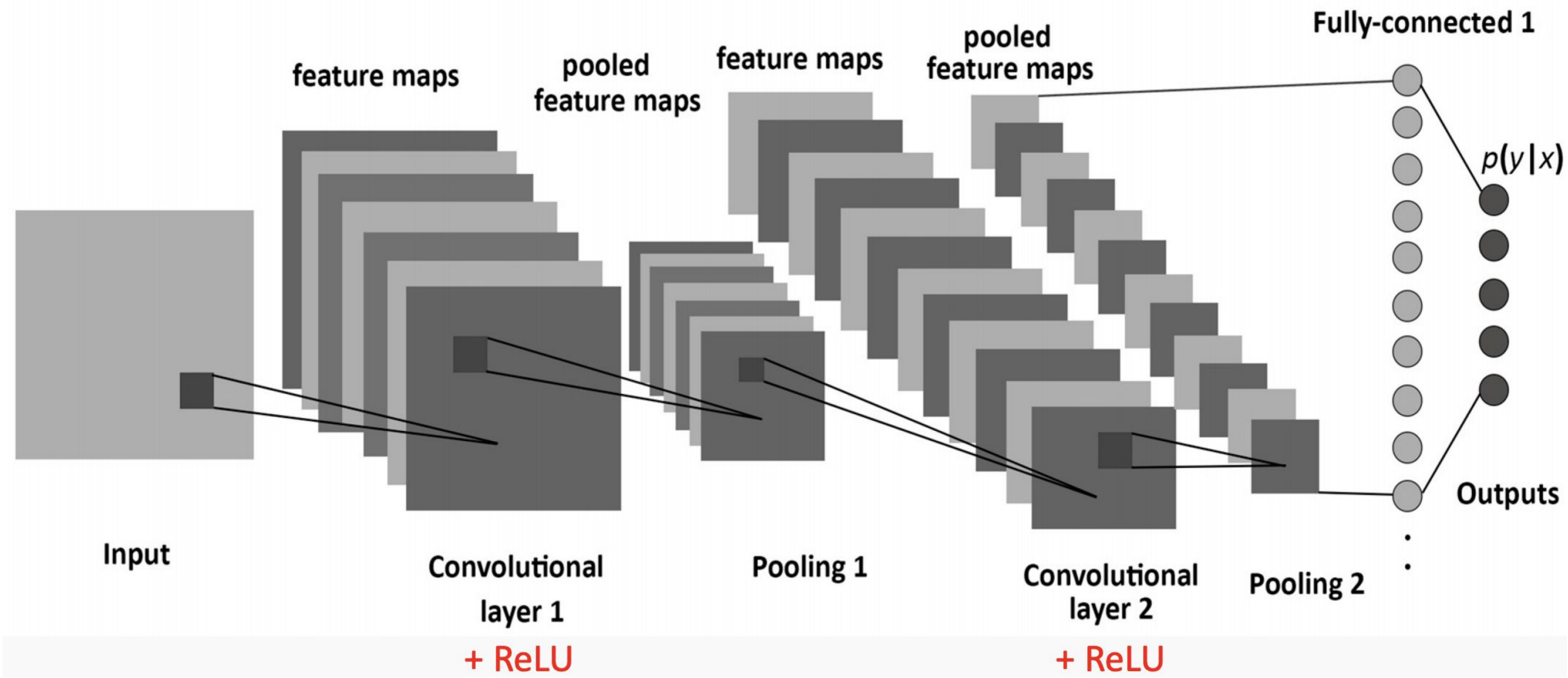
Image Classification: Convolutional Neural Network

Multi-Layer Network



https://medium.com/@cdabakoglu

Image Classification: Example Model



https://medium.com/@cdabakoglu

Harder Challenges

Computational Resources and Complex Tasks

- Transfer learning allows you to use models that have already been mostly trained
- Results nearly as good as full training

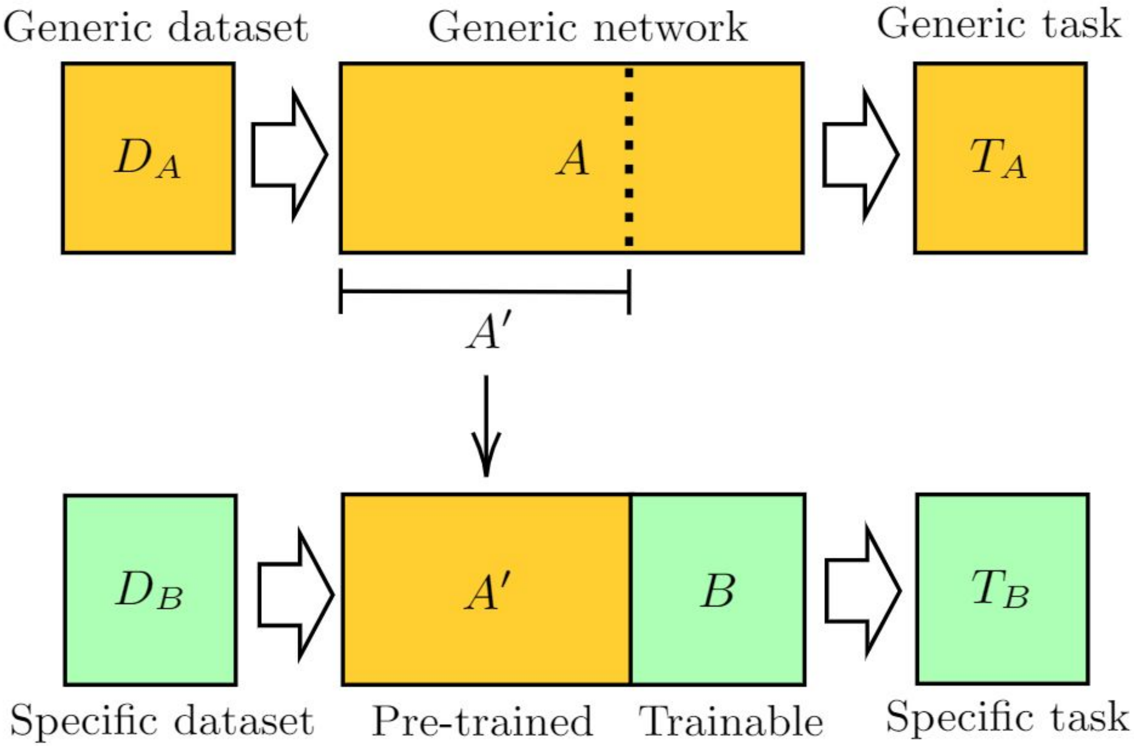
Data Set Intractability

- Keras has data generators so each batch can be loaded in separately
- Time required in near equivalent but memory needed is reduced drastically

Skipping Layers

- Keras has a functional model provided alongside the Sequential API
- Read about it in the notes!

Transfer Learning



Transfer Learning: Example

ResNet 50:

- One of the high performing models for image classification on ImageNet data
- Takes quite a long time to train

Fold	Accuracy	Time (s)
0	0.86	16,888.32
1	0.82	16,895.00
2	0.85	16,897.95
3	0.77	16,879.80
4	0.80	16,884.62
Average	**0.82**	**16,889.14**

Transfer Learning: Example

ResNet 50:

- One of the high performing models for image classification on ImageNet data
- Takes quite a long time to train

Goal:

- Use resnet to classify between images of cats and dogs

Solution:

- Use all of ResNet50 except the last layer and re-train ONLY the last layer to get the best training error between cats and dogs

Fold	Accuracy	Time (s)
0	0.86	16,888.32
1	0.82	16,895.00
2	0.85	16,897.95
3	0.77	16,879.80
4	0.80	16,884.62
Average	**0.82**	**16,889.14**

Transfer Learning: Example

```
resnet_weights_path = ... #Can be found online

model = Sequential()
model.add(ResNet50(include_top=False, pooling='avg',
     weights=resnet_weights_path))
model.add(Dense(2, activation='softmax'))
```

Keras Generator

- Works similar to Python Generators
- “Generates” one batch at a time so RAM is not overloaded
- Simplest example:
  - `ImageDataGenerator().flow_from_directory(directory_name)`
- Can write your own Python generator function:

```
def data_generator(...):

    while True:

        # … various processing ...

        yield X_batch, y_batch
```

Keras Generator: Example

File Storage

```
data/
    train/
        dogs/
            dog001.jpg
            dog002.jpg
            ...
        cats/
            cat001.jpg
            cat002.jpg
            ...
    validation/
        dogs/
            dog001.jpg
            dog002.jpg
            ...
        cats/
            cat001.jpg
            cat002.jpg
            ...
```

Generator Code

```
test_datagen = ImageDataGenerator()

train_generator = train_datagen.flow_from_directory(
    'data/train',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
    'data/validation',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')
```

towardsdatascience.com

Keras Generator: Training

```
model.fit_generator(
    train_generator,
    steps_per_epoch=2000,
    epochs=50,
    validation_data=validation_generator,
    validation_steps=800)
```

Apache Spark

Download Apache Spark

- https://spark.apache.org/downloads.html
- Or run `pip install pyspark` in your terminal
    - Installation should be fairly straightforward.

Uses of Apache Spark

- In-memory data processing
- Process large datasets
- Data streaming
- Mlib: Machine Learning library
- SystemML: Optimization language to speed up ML computations

MapReduce: Basic Ideas

- Two step method used to process data
- Map: applies a function to every element in a dataset
- Reduce: combine results from map
- Example (CS 61A): (reduce * (map (lambda (x) (+ x 2)) ‘(1 2 3 4 5))
    - Applies (lambda (x) (+ x 2)) to every element of the list ‘(1 2 3 4 5), then combines the resulting elements through multiplication

MapReduce

- Input: dataset of (key, value) pairs
- Map: function sending (key, value) to (mapped key, mapped value)
- Reduce: takes the set of mapped values and possibly a key, and merges the values which are returned
- Output: result from reduce (can be either a single value or a set of values)
- MapReduce has many diverse applications, including the following:
    - Word/character count
    - Maximum and minimum temperatures around the world
    - Presidential election results by congressional district
    - K-means algorithm

MapReduce in Spark

- Map and Reduce functions are given in Spark
- Up to 100 times faster than MapReduce in Hadoop
- Example (word count):

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

http://spark.apache.org/examples.html

MLlib

- Built-in scalable machine learning library in Spark
- Includes the following algorithms and utilities:
  - Regression
  - Classification
  - Clustering
  - Dimensionality reduction
  - Optimization methods
  - Filtering
  - Featurization
  - Statistical testing

https://spark.apache.org/docs/1.1.1/mllib-guide.html?utm_source=xp&utm_medium=blog&utm_campaign=content

MLlib: Regression

Linear Least Squares:

```
# Refer to https://spark.apache.org/docs/1.1.1/mllib-linear-methods.html
# import algorithms
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD
from numpy import array

# Load and parse the data
data = sc.textFile("data/mllib/ridge-data/lpsa.data")
parsedData = data.map(parsePoint)
```

Import all required libraries, load data and map preprocessing functions

https://spark.apache.org/docs/1.1.1/mllib-linear-methods.html

https://docs.databricks.com/applications/machine-learning/train-model/mllib/index.html#decision-trees-examples

MLlib: Regression

Linear Least Squares:

```
# Refer to https://spark.apache.org/docs/1.1.1/mllib-linear-methods.html
# import algorithms
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD
from numpy import array

# Load and parse the data
data = sc.textFile("data/mllib/ridge-data/lpsa.data")
parsedData = data.map(parsePoint)

# Build the model
model = LinearRegressionWithSGD.train(parsedData)
```

Build and train!

https://spark.apache.org/docs/1.1.1/mllib-linear-methods.html

https://docs.databricks.com/applications/machine-learning/train-model/mllib/index.html#decision-trees-examples

MLlib: Regression

Linear Least Squares:

```python
# Refer to https://spark.apache.org/docs/1.1.1/mllib-linear-methods.html
# import algorithms
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD
from numpy import array

# Load and parse the data
data = sc.textFile("data/mllib/ridge-data/lpsa.data")
parsedData = data.map(parsePoint)

# Build the model
model = LinearRegressionWithSGD.train(parsedData)

# Evaluate model
valuesAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
MSE = valuesAndPreds.map(lambda (v, p): (v - p)**2) \
                    .reduce(lambda x, y: x + y) / valuesAndPreds.count()
```

Evaluate using MapReduce!

https://spark.apache.org/docs/1.1.1/mllib-linear-methods.html

https://docs.databricks.com/applications/machine-learning/train-model/mllib/index.html#decision-trees-examples

MLlib: Classification

Decision Tree:

```
# Refer to https://spark.apache.org/docs/1.5.2/ml-decision-tree.html
# import algorithms
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import StringIndexer, VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.mllib.util import MLUtils

# Load, parse, convert data to dataframe
data = MLUtils.loadLibSVMFile(sc, "data.txt").toDF()

# Index labels, adding metadata to the label column.
labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)
# Identify categorical features, and index them.
featureIndexer = VectorIndexer(inputCol="features",
                               outputCol="indexedFeatures",
                               maxCategories=5).fit(data)
# Split the data into training and test sets
(trainingData, testData) = data.randomSplit([0.75, 0.25])
```

Load data, use SKLearn'esque helper data processors, and perform a data split.

MLlib: Classification

Decision Tree:

```
# Refer to https://spark.apache.org/docs/1.5.2/ml-decision-tree.html
# import algorithms
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import StringIndexer, VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.mllib.util import MLUtils

# Load, parse, convert data to dataframe
data = MLUtils.loadLibSVMFile(sc, "data.txt").toDF()

# Index labels, adding metadata to the label column.
labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)
# Identify categorical features, and index them.
featureIndexer = VectorIndexer(inputCol="features",
                               outputCol="indexedFeatures",
                               maxCategories=5).fit(data)
# Split the data into training and test sets
(trainingData, testData) = data.randomSplit([0.75, 0.25])

# Train a DecisionTree model.
dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")

# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dt])

# Train model
model = pipeline.fit(trainingData)
```

Create a DecisionTree and an MLlib Pipeline that processes and trains the model.

MLlib: Classification

Decision Tree:

```python
# Refer to https://spark.apache.org/docs/1.5.2/ml-decision-tree.html
# import algorithms
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import StringIndexer, VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.mllib.util import MLUtils

# Load, parse, convert data to dataframe
data = MLUtils.loadLibSVMFile(sc, "data.txt").toDF()

# Index labels, adding metadata to the label column.
labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)
# Identify categorical features, and index them.
featureIndexer = VectorIndexer(inputCol="features",
                               outputCol="indexedFeatures",
                               maxCategories=5).fit(data)
# Split the data into training and test sets
(trainingData, testData) = data.randomSplit([0.75, 0.25])

# Train a DecisionTree model.
dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")

# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dt])

# Train model
model = pipeline.fit(trainingData)

predictions = model.transform(testData)

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="precision")
accuracy = evaluator.evaluate(predictions)
```

Predict and Evaluate

MLlib: K-means clustering

```
from numpy import array
parsedData = data.map(
    lambda line: array([float(x) for x in line.split(' ')])
).cache()
parsedData.foreach(show)
```

MapReduce! Helpful for any massively parallelizable operation.

http://vargas-solar.com/big-data-analytics/hands-on/k-means-with-spark-hadoop/

MLlib: K-means clustering

```
from numpy import array
parsedData = data.map(
    lambda line: array([float(x) for x in line.split(' ')])
).cache()
parsedData.foreach(show)
```

```
from pyspark.mllib.clustering import KMeans
clusters = KMeans.train(parsedData, 2, maxIterations=10, runs=10, initializationMode='random')
```

Import and train!
Similar to SKLearn's interface

http://vargas-solar.com/big-data-analytics/hands-on/k-means-with-spark-hadoop/

MLlib: K-means clustering

```
from numpy import array
parsedData = data.map(
    lambda line: array([float(x) for x in line.split(' ')])
).cache()
parsedData.foreach(show)
```

```
from pyspark.mllib.clustering import KMeans
clusters = KMeans.train(parsedData, 2, maxIterations=10, runs=10, initializationMode='random')
```

```
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))
```

Prediction Error Metric

```
WSSSE = (parsedData.map(lambda point:error(point)).reduce(lambda x, y: x+y))
print('Within Set Sum of Squared Error = ' + str(WSSSE))
```

Use MapReduce again to calculate the overall error!

http://vargas-solar.com/big-data-analytics/hands-on/k-means-with-spark-hadoop/

SystemML: Introduction

- SystemML provides a bridge between Keras and Spark!
  - Provides parallelization capabilities
- Typically used in tasks with complexity in data or task
  - Hence… why it’s not covered as much in normal machine learning frameworks
- Advanced material that is not going to be covered in great detail since it is difficult to implement in practise without existing infrastructure
- Training and testing is simple though, so let’s take a look.

SystemML: Example

```
from systemml.mllearn import Keras2DML

epochs = 5
batch_size = 100
samples = 60000
max_iter = int(epochs*math.ceil(samples/batch_size))

sysml_model = Keras2DML(spark, keras_model, input_shape=(1,28,28),
weights='weights_dir', batch_size=batch_size, max_iter=max_iter,
test_interval=0, display=10)
```

Create model (using Keras)

https://towardsdatascience.com

SystemML: Example

```
from systemml.mllearn import Keras2DML


epochs = 5
batch_size = 100
samples = 60000
max_iter = int(epochs*math.ceil(samples/batch_size))


sysml_model = Keras2DML(spark, keras_model, input_shape=(1,28,28),
weights='weights_dir', batch_size=batch_size, max_iter=max_iter,
test_interval=0, display=10)


sysml_model.fit(X_train, y_train)
```

Train model using SystemML

https://towardsdatascience.com

SystemML: Example

```
from systemml.mllearn import Keras2DML

epochs = 5
batch_size = 100
samples = 60000
max_iter = int(epochs*math.ceil(samples/batch_size))

sysml_model = Keras2DML(spark, keras_model, input_shape=(1,28,28),
weights='weights_dir', batch_size=batch_size, max_iter=max_iter,
test_interval=0, display=10)

sysml_model.fit(X_train, y_train)
```

```
sysml_model.score(X_test, y_test)
```

Check model accuracy

https://towardsdatascience.com