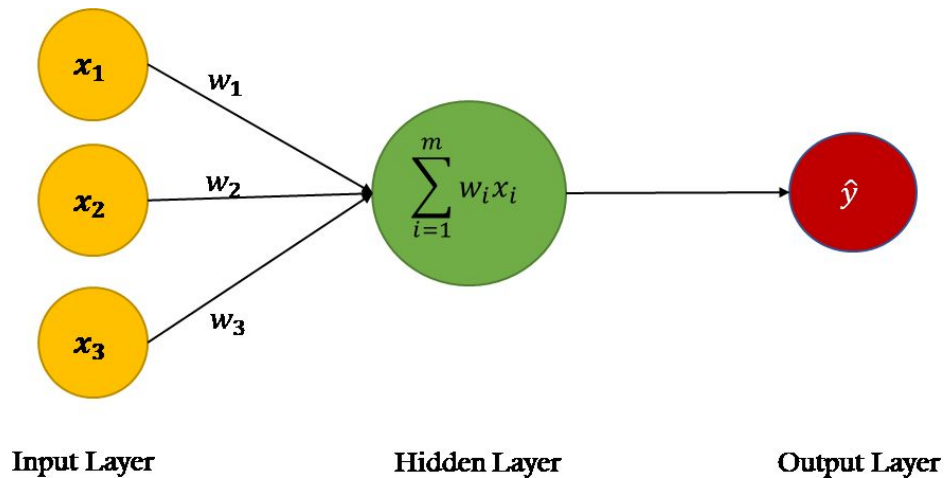




Keras/Apache
Spark



Neuron



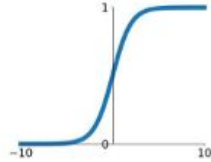
- Takes in any number of inputs and creates a weighted average of them
- Can apply an “activation function” to induce non-linearity

Activation Function

Activation Functions

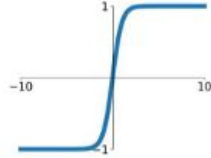
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



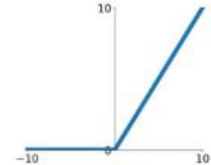
tanh

$$\tanh(x)$$



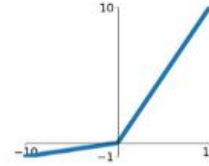
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

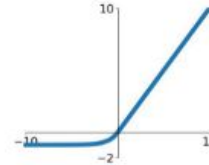


Maxout

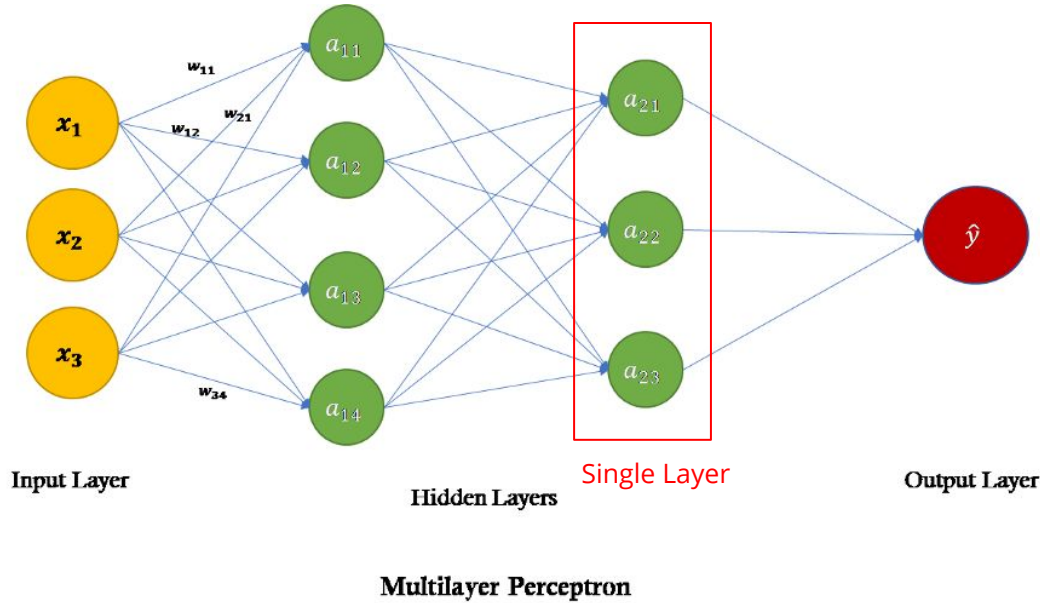
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neural Network



- With many neurons, organized in layers, you can theoretically model highly non-linear behavior

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

Actual Solution:

- Differentiation
- Backpropagation
- Gradient Descent

Easier Solution:



Example Problem: Pythagorean Distance

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

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 = tensorflow.keras.models.Sequential()
```

```
#Add input layer
```

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

```
#Add hidden layers
```

```
model.add(tensorflow.keras.layers.Dense(units=10, activation='relu'))
```

```
model.add(tensorflow.keras.layers.Dense(units=5, activation='tanh'))
```

```
#Add output layer
```

```
model.add(tensorflow.keras.layers.Dense(units=1))
```


Compiling and Training the Model

```
#Compile the model
```

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

```
#Train the model
```

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

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*

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
 - Designed to do matrix operations faster and can therefore train faster

Classification

Softmax Layer

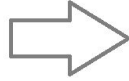
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

- Takes any input and outputs a probability distribution
 - Motivation: The training outputs can now just be one-hot encoded vectors of the class
- To classify image, choose class with highest probability

Image Classification: Basic

1	1	0
4	2	1
0	2	1

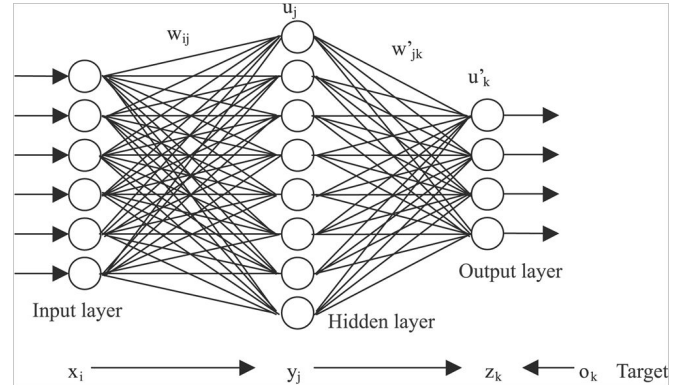
Image



1
1
0
4
2
1
0
2
1

Flattened

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



Fully Connected Layers

Image Classification: Convolutional Neural Network

Convolution

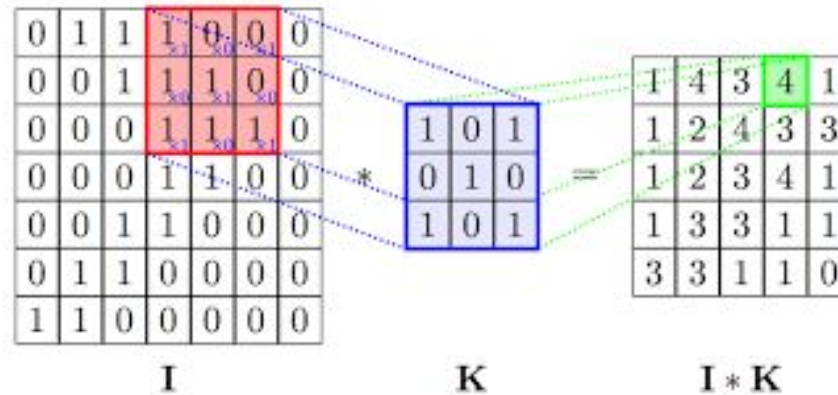


Image Classification: Convolutional Neural Network

Pooling

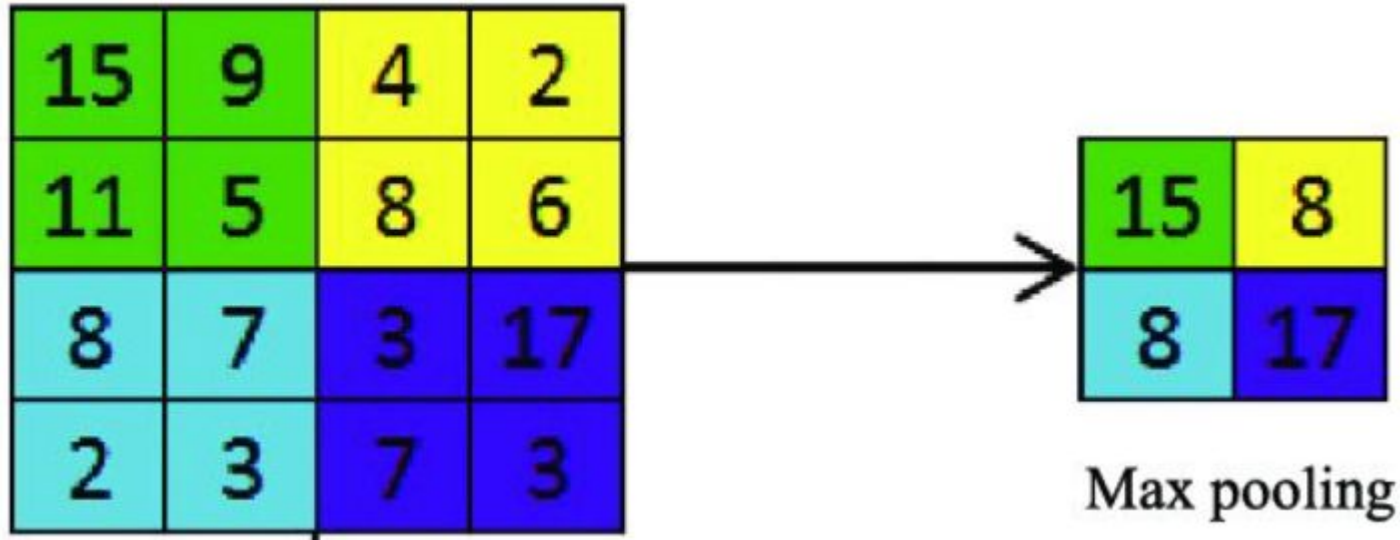


Image Classification: Convolutional Neural Network

Multi-Layer

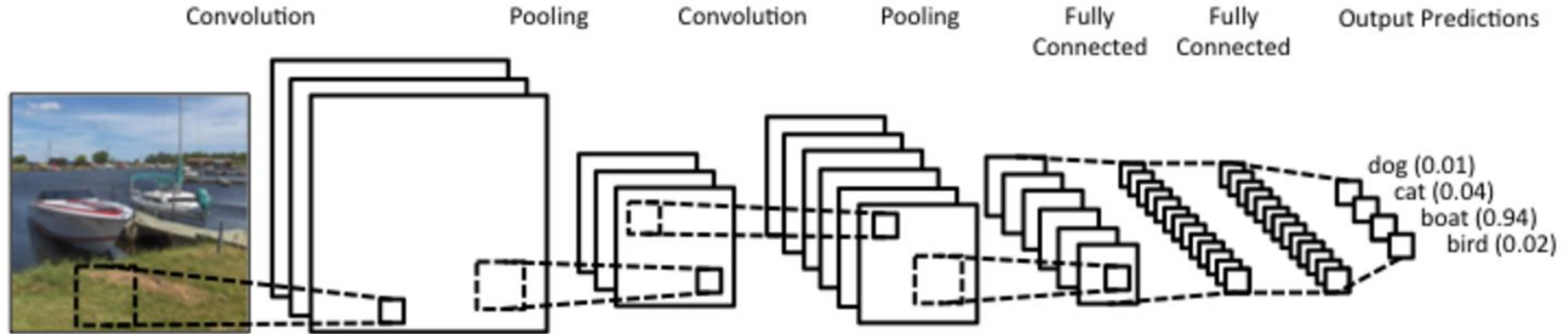


Image Classification: Convolutional Neural Network

Dropout:

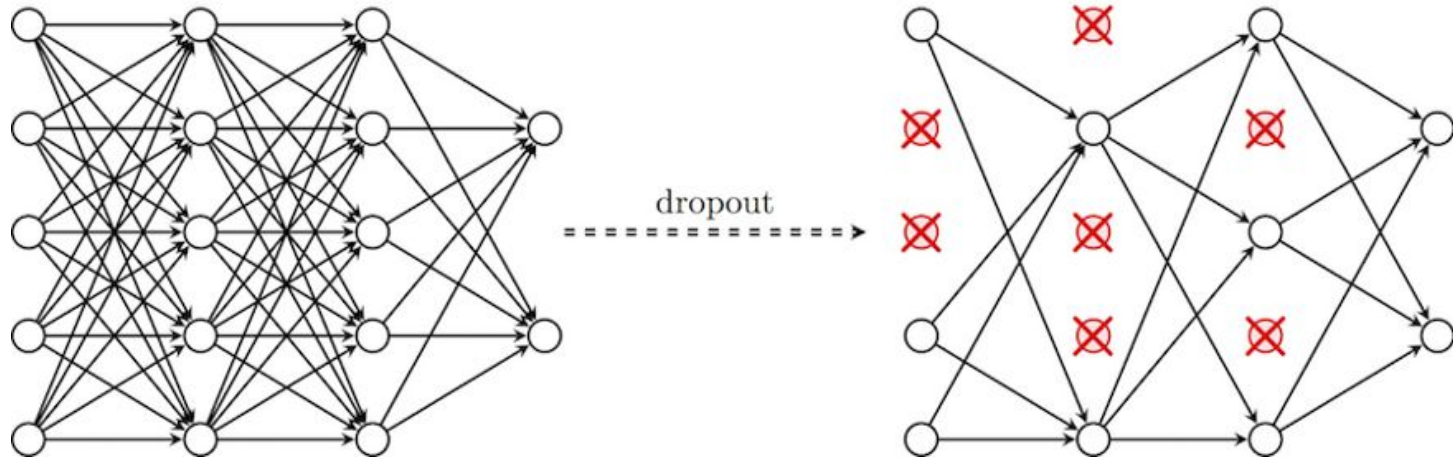


Image Classification: Example Model

Harder Challenges

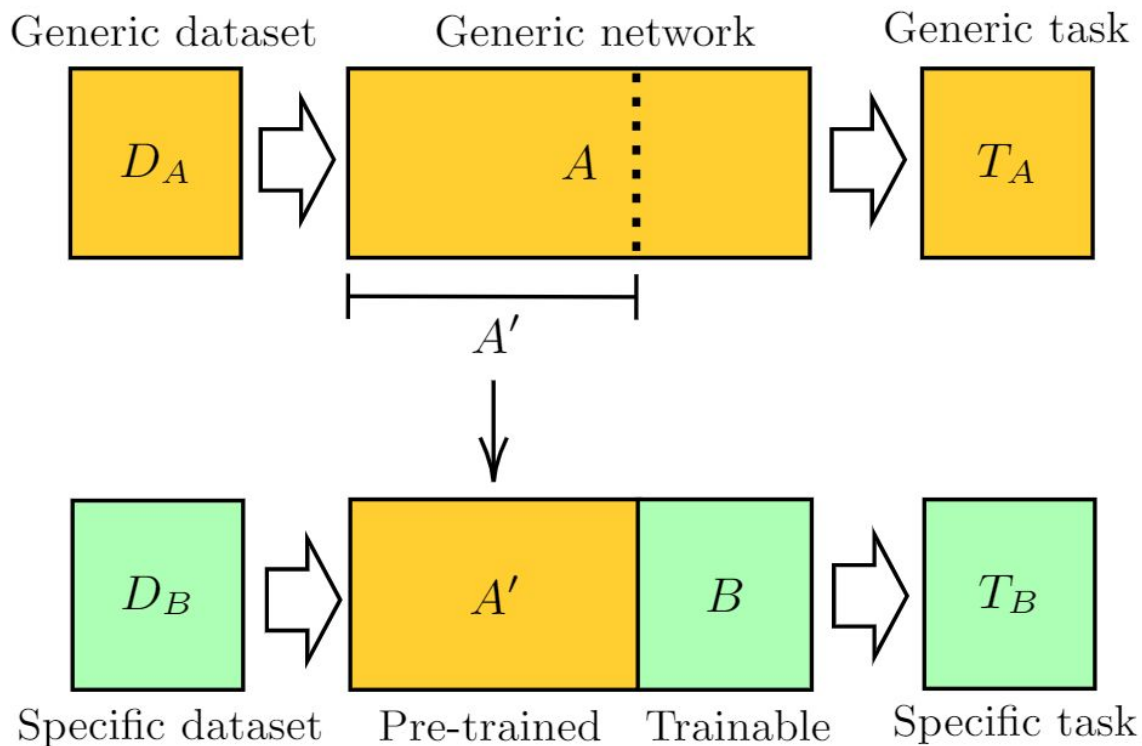
Complicated Problems + Not much computation available

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

Data set very large

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

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)`

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')
```

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

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)))`

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)
- Example Uses
 - 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://...")
```

Mlib

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

Mlib: 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)]))

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

Mlib: 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)

predictions = model.transform(testData)

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

Mlib: Classification

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)

# 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()
,
```

SystemML: Training

```
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)
```

SystemML: Check Model Fit

- After training, use a test set to check how well the model fits our data:

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