Spark Technology
Center

Deep Neural Network Regression at Scale in MLlib

Jeremy
Nixon

Acknowledgements - Built off of work by
Alexander Ulanov and Xiangrui Meng

Spark.tc

Structure

1. Introduction / About
2. Motivation
   a. Regression
   b. Comparison with prominent MLlib algorithms
3. Properties
   a. Automated Feature Generation
   b. Capable of Learning Non-Linear Structure
   c. Non-Local Generalization
4. Framing Deep Learning
5. The Model
6. Applications
7. Features / Usage
8. Optimization
9. Future Work

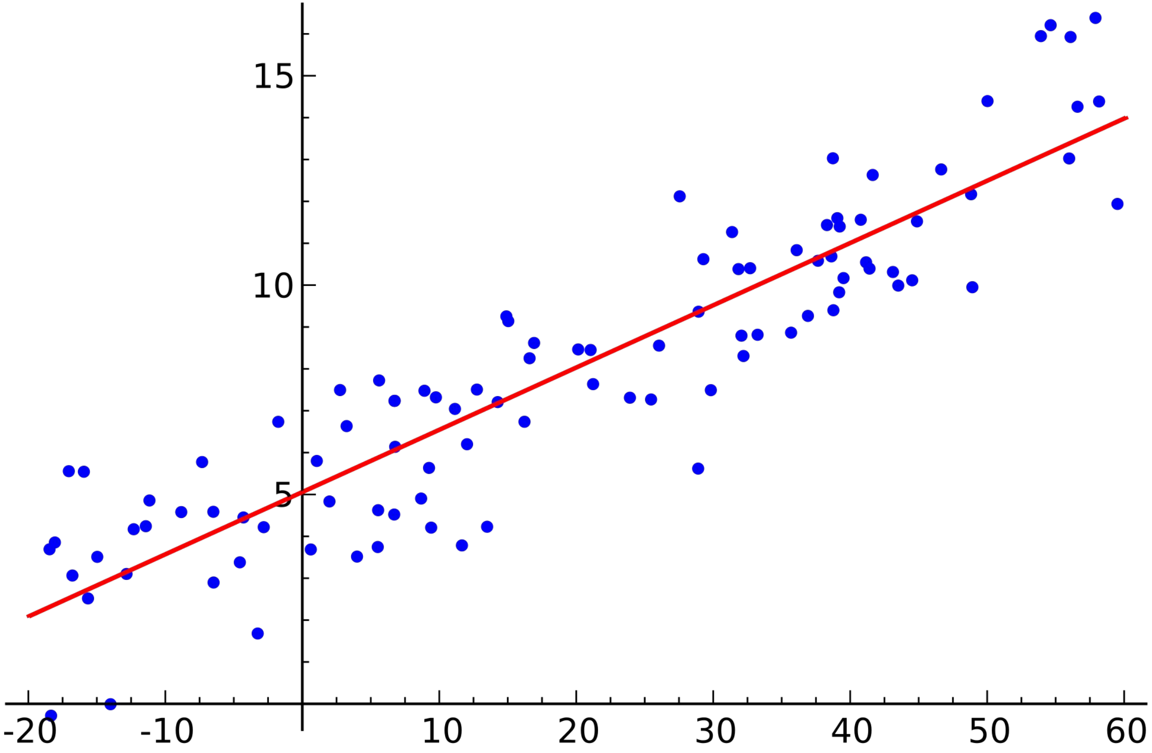
Jeremy Nixon

- Machine Learning Engineer at the Spark Technology Center
- Contributor to MLlib, scalable-deeplearning
- Previously, studied Applied Mathematics to Computer Science / Economics at Harvard
- www.github.com/JeremyNixon

Regression Models are Valuable For:

- Location Tracking in Images
- Housing Price Prediction
- Predicting Lifetime value of a customer
- Stock market stock evaluation
- Forecasting Demand for a product
- Pricing Optimization
- Price Sensitivity
- Dynamic Pricing
- Many, many other applications.

Ever trained a Linear Regression Model?



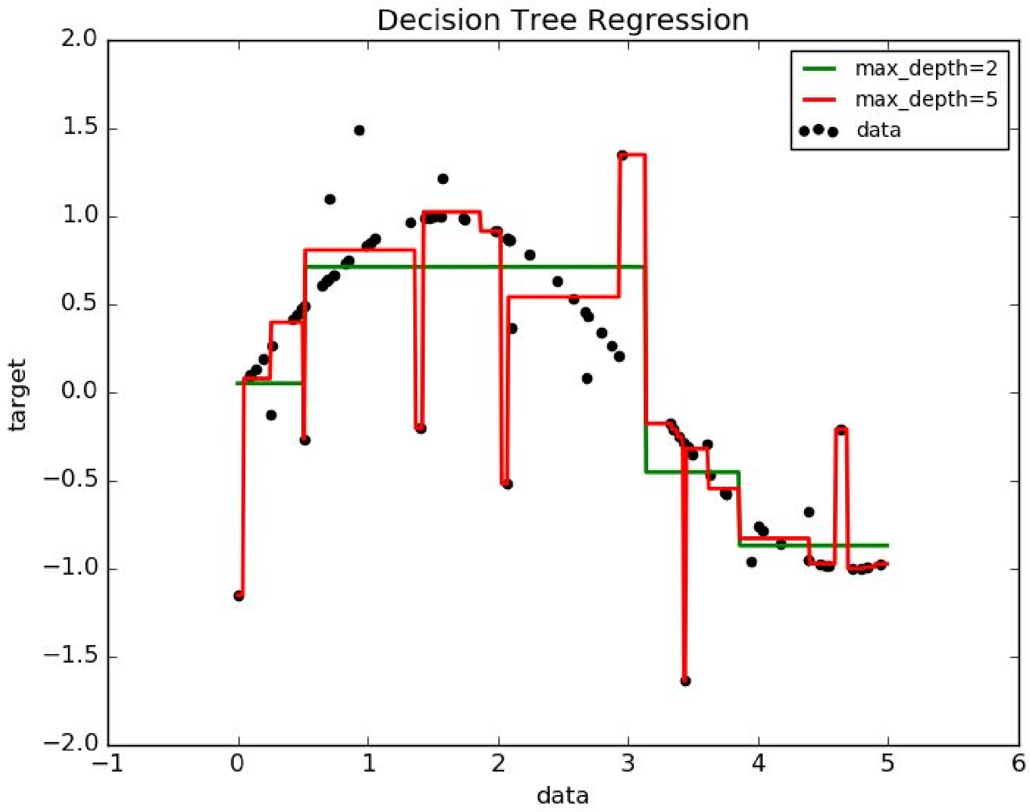
Linear Regression Models

Major Downsides:

Cannot discover non-linear structure in data.

Manual feature engineering by the Data Scientist. This is time consuming and can be infeasible for high dimensional data.

Decision Tree Based Model? (RF, GB)



Decision Tree Models

Upside:

Capable of automatically picking up on non-linear structure.

Downsides:

Incapable of generalizing outside of the range of the input data.

Restricted to cut points for relationships.

Thankfully, there's an algorithmic solution.

Multilayer Perceptron Regression

- New Algorithm on Spark MLlib -

Deep Feedforward Neural Network for Regression.

```
val data = sqlContext.read.format("libsvm").load("../data/mllib/sample_mlpr_data.txt")
val Array(train, test) = data.randomSplit(Array(0.7, 0.3))
val layers = Array[Int](12, 100, 100, 100, 100, 100, 1)
val trainer = new MultilayerPerceptronRegressor().setLayers(layers).setSolver("l-bfgs").setSeed(1234L)
val model = trainer.fit(train)
val result = model.transform(test)
```

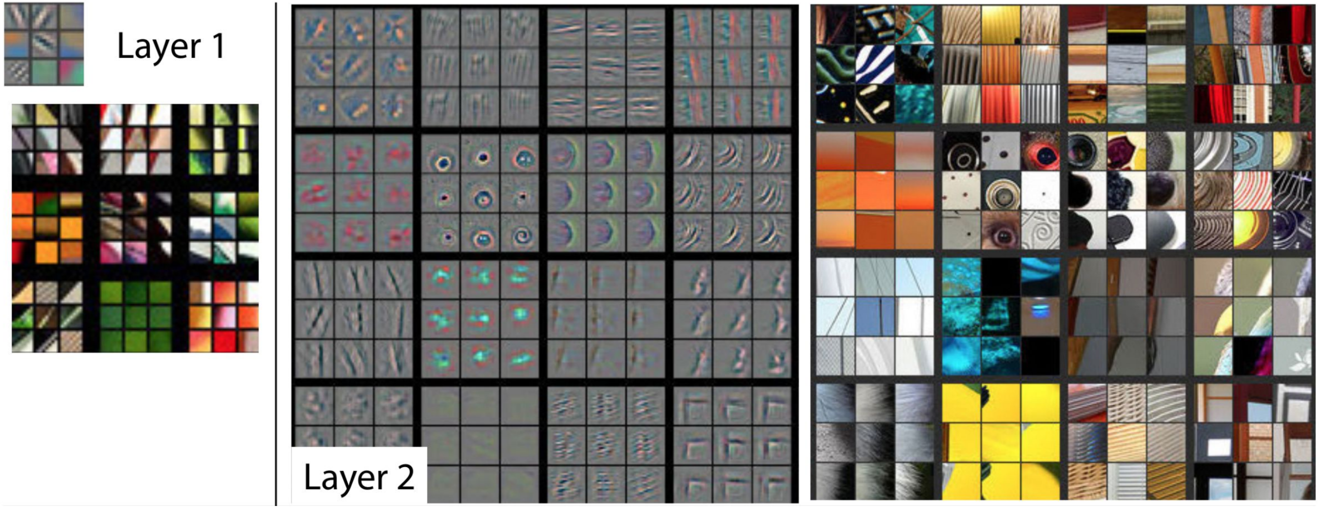
Properties

Overview

1. Automated Feature Generation
2. Capable of Learning Non-linear Structure
3. Generalization outside input data range

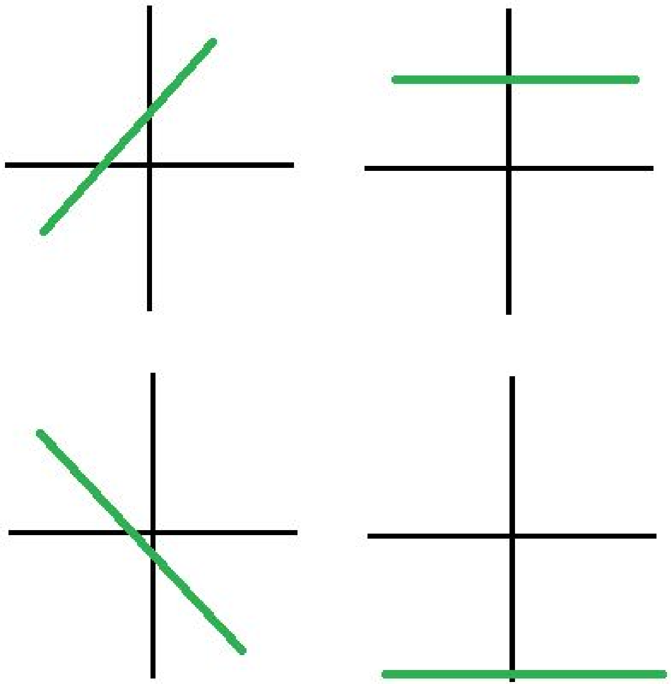
Automated Feature Generation

- Pixel - Edges - Shapes - Parts - Objects : Prediction
- Learns features that are optimized for the data

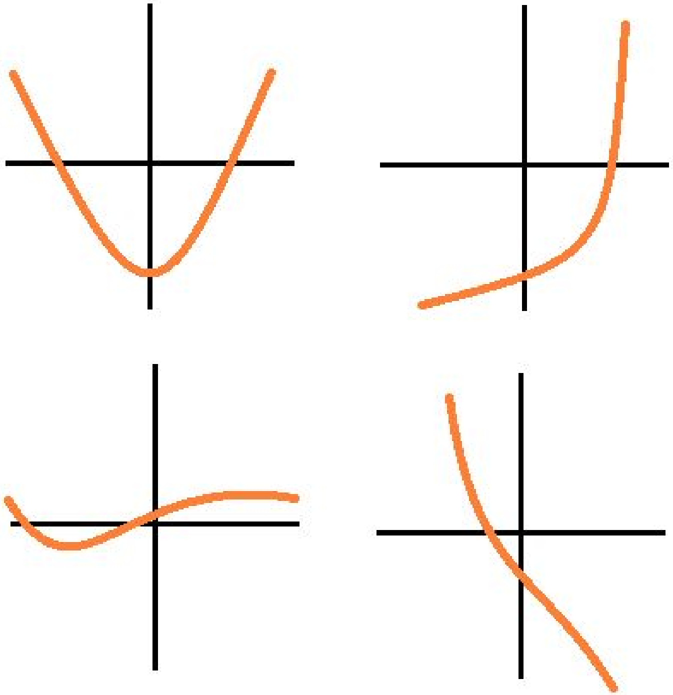


Visualizing & Understanding Convolutional Networks , ECCV 2014, Matthew Zeiler, Rob Fergus

Capable of Learning Non-Linear Structure

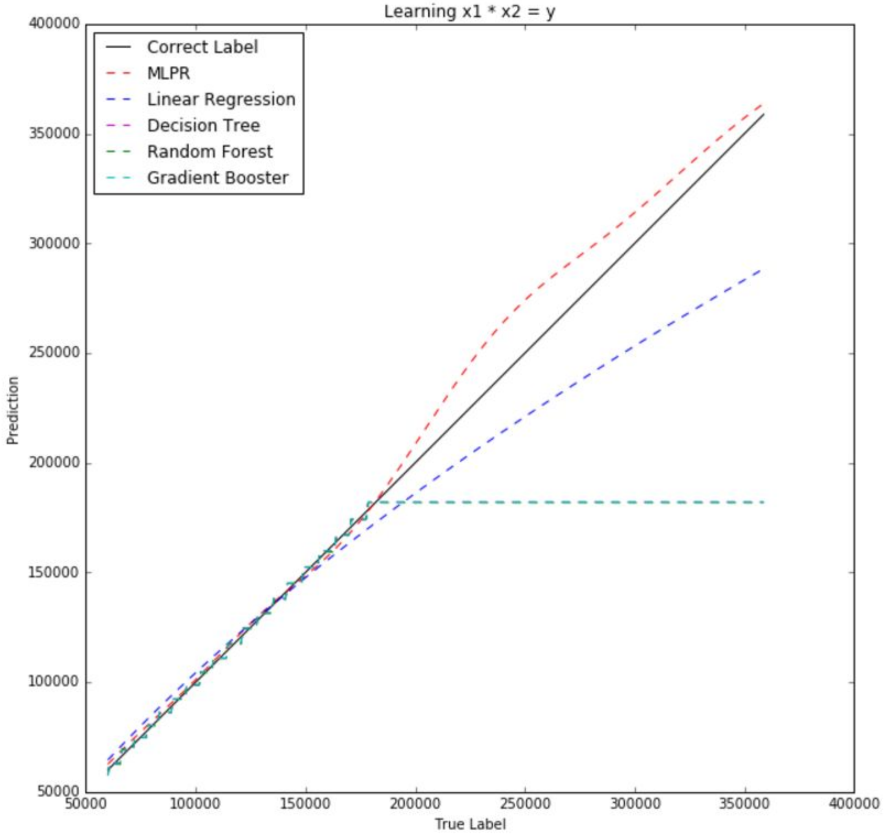


Linear Functions



Nonlinear Functions

Generalization Outside Data Range



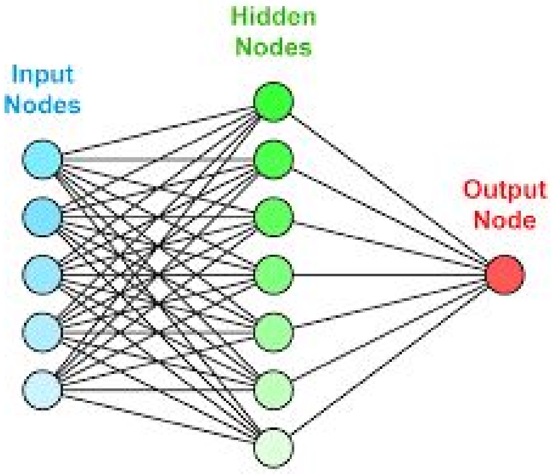
Many Successes of Deep Learning

1. CNNs - State of the art
    a. Object Recognition
    b. Object Localization
    c. Image Segmentation
    d. Image Restoration
2. RNNs (LSTM) - State of the Art
    a. Speech Recognition
    b. Question Answering
    c. Machine Translation
    d. Text Summarization
    e. Named Entity Recognition
    f. Natural Language Generation
    g. Word Sense Disambiguation
    h. Image / Video Captioning
    i. Sentiment Analysis

Many Ways to Frame Deep Learning

1. Automated Feature Engineering
2. Non-local generalization
3. Manifold Learning
4. Exponentially structured flexibility countering curse of dimensionality
5. Hierarchical Abstraction
6. Learning Representation / Input Space Contortion / Transformation for Linear Separability
7. Extreme model flexibility leading to the ability to absorb much larger data without penalty

The Model



X = Normalized Data, W_1, W_2 = Weights, b = Bias

Forward:

1. Multiply data by first layer weights | $(X*W_1 + b_1)$
2. Put output through non-linear activation | $\max(0, X*W_1 + b_1)$
3. Multiply output by second layer weights | $\max(0, X*W_1 + b) * W_2 + b_2$
4. Return predicted output

```
override def eval(data: BDM[Double], output: BDM[Double]): Unit = {
  output(::, *) := b
  BreezeUtil.dgemm(1.0, w, data, 1.0, output)
}
```

```
private[ann] class ReluFunction extends ActivationFunction {

  override def eval: (Double) => Double = x => {
    if (x > 0) x
    else 0
  }
```

DNN Regression Applications

Great results in:

- Computer Vision
  - Object Localization / Detection as DNN Regression
  - Self-driving Steering Command Prediction
  - Human Pose Regression
- Finance
  - Currency Exchange Rate
  - Stock Price Prediction
  - Forecasting Financial Time Series
  - Crude Oil Price Prediction

DNN Regression Applications

Great results in:

- Atmospheric Sciences
  - Air Quality Prediction
  - Carbon Dioxide Pollution Prediction
  - Ozone Concentration Modeling
  - Sulphur Dioxide Concentration Prediction
- Infrastructure
  - Road Tunnel Cost Estimation
  - Highway Engineering Cost Estimation
- Geology / Physics
  - Meteorology and Oceanography Application
  - Pacific Sea Surface Temperature Prediction
  - Hydrological Modeling

Features of DNNR

1. Automatically Scaling Output Labels
2. Pipeline API Integration
3. Save / Load Models Automatically
4. Gradient Descent and L-BFGS
5. Tanh and Relu Activation Functions

Optimization

Loss Function

We compute our errors (difference between our predictions and the real outcome) using the mean squared error function:

$$\mathrm{MSE}_{\mathrm{test}} = \frac{1}{m} \sum_i (\hat{\boldsymbol{y}}^{(\mathrm{test})} - \boldsymbol{y}^{(\mathrm{test})})_i^2 .$$

Optimization

Parallel implementation of backpropagation:

1. Each worker gets weights from master node.
2. Each worker computes a gradient on its data.
3. Each worker sends gradient to master.
4. Master averages the gradients and updates the weights.

```scala
override def computePrevDelta(
  delta: BDM[Double],
  output: BDM[Double],
  prevDelta: BDM[Double]): Unit = {
  BreezeUtil.dgemm(1.0, w.t, delta, 0.0, prevDelta)
}

override def grad(delta: BDM[Double], input: BDM[Double], cumGrad: BDV[Double]): Unit = {
  // compute gradient of weights
  val cumGradientOfWeights = new BDM[Double](w.rows, w.cols, cumGrad.data, cumGrad.offset)
  BreezeUtil.dgemm(1.0 / input.cols, delta, input.t, 1.0, cumGradientOfWeights)
  if (ones == null || ones.length != delta.cols) ones = BDV.ones[Double](delta.cols)
  // compute gradient of bias
  val cumGradientOfBias = new BDV[Double](cumGrad.data, cumGrad.offset + w.size, 1, b.length)
  BreezeUtil.dgemv(1.0 / input.cols, delta, ones, 1.0, cumGradientOfBias)
}
```

```scala
override def derivative: (Double) => Double = z => {
  if (z > 0) 1
  else 0
}
```

Performance

- Parallel MLP on Spark with 7 nodes ~= Caffe w/GPU (single node).
- Advantages to parallelism diminish with additional nodes due to communication costs.
- Additional workers are valuable up to ~20 workers.
- See https://github.com/avulanov/ann-benchmark for more details

Future Work

1. Convolutional Neural Networks
    a. Convolutional Layer Type
    b. Max Pooling Layer Type
2. Flexible Deep Learning API
3. More Modern Optimizers
    a. Adam
    b. Adadelta + Nesterov Momentum
4. More Modern activations
5. Dropout / L2 Regularization
6. Batch Normalization
7. Tensor Support
8. Recurrent Neural Networks (LSTM)

References

- Detection as DNN Regression: http://papers.nips.cc/paper/5207-deep-neural-networks-for-object-detection.pdf
- Object Localization: http://arxiv.org/pdf/1312.6229v4.pdf
- Pose Regression: https://www.robots.ox.ac.uk/~vgg/publications/2014/Pfister14a/pfister14a.pdf
- Currency Exchange Rate: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.2442
- Stock Price Prediction: https://arxiv.org/pdf/1003.1457.pdf
- Forcasting Financial Time Series: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.15.8688&rep=rep1&type=pdf
- Crude Oil Price Prediction: http://www.sciencedirect.com/science/article/pii/S0140988308000765
- Air Quality Prediction: https://www.researchgate.net/profile/VR_Prybutok/publication/8612909_Prybutok_R._A_neural_network_model_forecasting_for_prediction_of_daily_maximum_ozone_concentration_in_an_industrialized_urban_area._Environ._Pollut._92(3)_349-357/links/0deec53babcab9c32f000000.pdf
- Air Pollution Prediction - Carbon Dioxide http://202.116.197.15/cadalcanton/Fulltext/21276_2014319_102457_186.pdf
- Atmospheric Sulphyr Dioxide Concentrations http://cdn.intechweb.org/pdfs/17396.pdf
- Oxone Concentration Comparison https://www.researchgate.net/publication/263416130_Statistical_Surface_Ozone_Models_An_Improved_Methodology_to_Account_for_Non-Linear_Behaviour
- Road Tunnel Cost Estimationhttp://ascelibrary.org/doi/abs/10.1061/(ASCE)CO.1943-7862.0000479
- Highway Engineering Cost Estimationhttp://www.jcomputers.us/vol5/jcp0511-19.pdf
- Pacific Sea Surface Temperature http://www.ncbi.nlm.nih.gov/pubmed/16527455
- Meteorology and Oceanography https://open.library.ubc.ca/cIRcle/collections/facultyresearchandpublications/32536/items/1.0041821
- Hydrological Modeling: http://hydrol-earth-syst-sci.net/13/1607/2009/hess-13-1607-2009.pdf

Thank You!

Questions?

Acknowledgements:

Built off of work by

Alexander Ulanov and Xiangrui Meng