

Prototyping and Deploying Models to Production

Keras Model Import into DeepLearning4J

Tom Hanlon

tom@skymind.io



Getting Help

- Join us on Gitter Chat
 - <https://gitter.im/deeplearning4j/deeplearning4j>
- Email Tom@skymind.io



Training Information

- <https://skymind.ai/academy>
 - Contact us for information on training for your organization !



Course Content



Introductions

- Why are you here?
- What do you want to learn?
- What chapter interests you?
- What is your background?



What is Deep Learning

This section describes the fields of Artificial Intelligence, Machine Learning and Deep Learning and how they are related.

What is Deep Learning?

The relationship between Deep Learning and Machine Learning

- Machine Learning
- Data Science / Data Mining
- Deep Learning



Introduction to Machine Learning

- ⇒ Machine Learning
- Data Science / Data Mining
- Deep Learning



Machine Learning

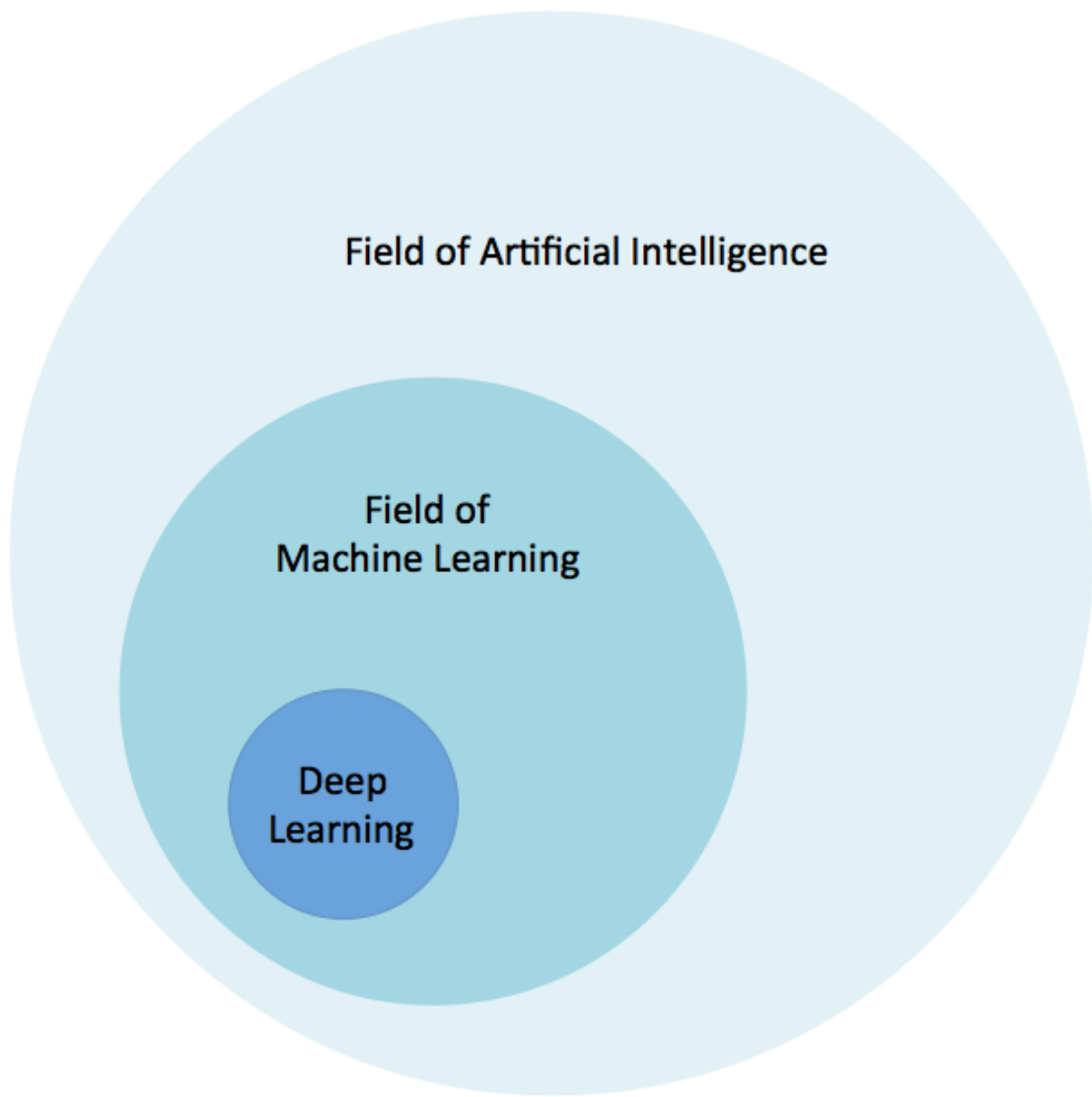


What is Machine Learning?

- Extracting Knowledge from raw data in the form of a model * Decision trees * Linear Models * Neural Networks
- Arthur Samuel quote:
 - "Field of study that gives computers the ability to learn without being explicitly programmed"



A Diagram



Data Science / Data Mining

- Machine Learning
- \Rightarrow Data Science / Data Mining
- Deep Learning



Machine Learning Compared to Data Science/Mining

- Data Mining
 - The process of extracting information from the data
 - Uses Machine Learning
- Data Science * Data Mining from the lens of a statistician * Venn Diagrams * A way to get a raise * A more agreeable Actuary * A statistician using a Mac



Deep Learning

- Machine Learning
- Data Science / Data Mining
- \Rightarrow Deep Learning



A Definition of Deep Learning

- Deep learning (also known as deep structured learning, hierarchical learning or deep machine learning) is a branch of machine learning based on a set of algorithms that attempt to model high level abstractions in data.

source - wikipedia



Neural Networks

- A computational approach patterned on the human brain and nervous system



Biological Neurons

- Biological Neuron: An electrically excitable cell that processes and transmits information through electrical and chemical signals
- Biological Neural Network: An interconnected group of neurons

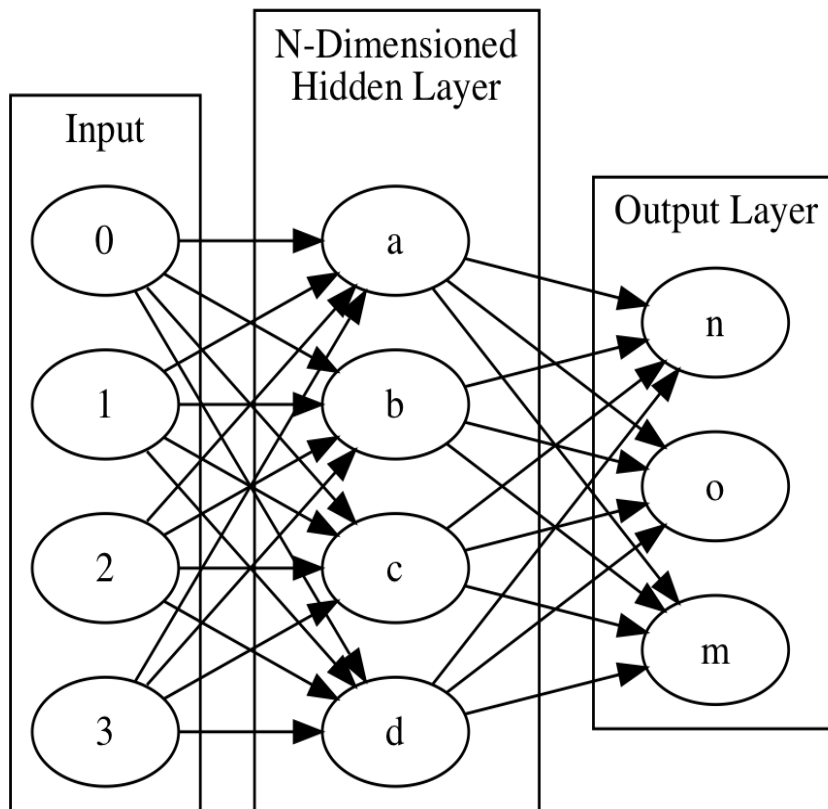


Role of Artificial Neural Network

- *Learns or Trains to perform tasks that traditional programming methods find rather challenging.*
Speech recognition, object recognition, computer vision, pattern recognition.



A Neural Network



Introduction to Keras

What is Keras

- Deep Learning Toolkit for building Neural Networks
- Uses Theano or Tensorflow for graph processing
- Uses Numpy Scikit learn for data wrangling



Keras Diagram



Introduction to Tensorflow and Theano

Tensorflow and Theano have similar Functionality

- Both provide a Python API to operations over NDArrays(Tensors)
- Both allow building a graph of operations and executing that graph in C++
- Both are pretty low level, not C++ but flexible and complex enough to do interesting things.



Tensorflow

- Developed by Google Brain Research team
- Written in Python C++
- Python API
 - C++ core



Tensorflow Core

- dataflow graph representing computations
- The entire dataflow graph is a complete description of computations, which occur within a session, and are executed on devices (CPUs or GPUs)
 - Nodes represent operations
 - edges represent tensors (multi-dimensional arrays, the backbone of TensorFlow)



Linear Regression example

```
import numpy as np
import tensorflow as tf

# Model parameters
W = tf.Variable([.3], tf.float32)
b = tf.Variable([-0.3], tf.float32)
# Model input and output
x = tf.placeholder(tf.float32)
linear_model = W * x + b
y = tf.placeholder(tf.float32)
# loss
loss = tf.reduce_sum(tf.square(linear_model - y)) # sum of the squares
# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
```



Linear Regression example...

```
# training data
x_train = [1,2,3,4]
y_train = [0,-1,-2,-3]
# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
for i in range(1000):
    sess.run(train, {x:x_train, y:y_train})
# evaluate training accuracy
curr_W, curr_b, curr_loss = sess.run([W, b, loss], {x:x_train, y:y_train})
print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))
```



Theano

- Numerical Computation Library for Python
 - Define, optimize and evaluate mathematical expressions over NDArrays
-

It is good to think of `theano.function` as the interface to a compiler which builds a callable object from a purely symbolic graph.

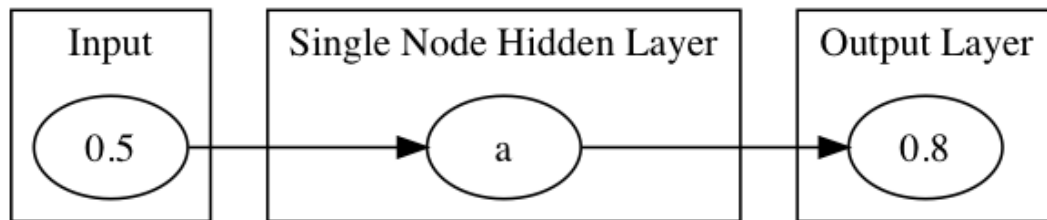
One of Theano's most important features is that `theano.function` can optimize a graph and even compile some or all of it into native machine instructions.



The Instructor will Demonstrate a Neural Network

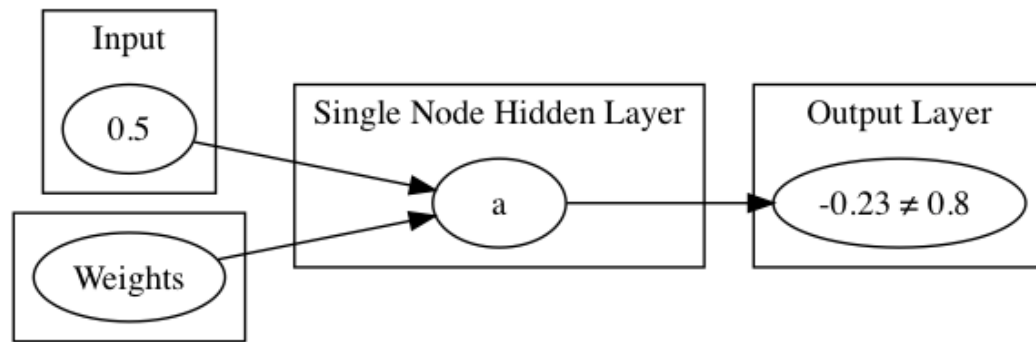
Simple Neural Network Details

The Goal: Input=0.5 Output=0.8



The Design:

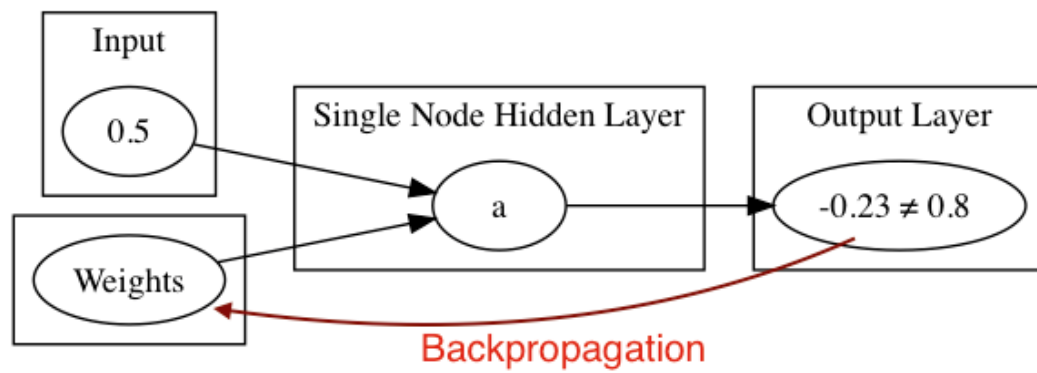
- Apply Random weights, plus activation function per neuron => output



- Test output: does $-0.23 = 0.8$?

Calculate error

- * backpropagate to adjust weights
- * repeat



How is Input Handled?

Input to a Neural Network is always Numeric. Typically a MultiDimensional Array, in this case.. not so many dimensions.

```
INDArray input = Nd4j.create(new float[] {(float) 0.5}, new int[] {1, 1});
```

Create a 1*1 INDArray with the value 0.5



How is Output Generated?

- The activation function combined with the number of neurons in output layer determine the output



Simple Network Output Layer

```
.layer(1, new OutputLayer.Builder(LossFunctions.LossFunction.MSE)  
.activation(Activation.IDENTITY) .nIn(nHidden).nOut(1).build())
```



Output Layer Explained

- Activation Function Identity
 - Emits what it receives
- LossFunction
 - How error is calculated



Output Layer of a Neural Network Can be Configured for Different Purposes:

- One of x possible classes
- True/False
- Range of values
- More...



Settings important to all Neural Networks

- Learning Rate
 - Applies to whole Network
 - If output is 10 and expected output is 1 how big of a step to take to correct error
- Activation Function
 - Set per layer
 - Sets the "trigger" on what output to emit
 - examples, sigmoid, ReLU, tanh
 - More on these later



Settings important to all Neural Networks...

- Loss Function
 - Set on Output Layer
 - How error is calculated
 - Example Mean Squared Error
 - More on this later
- Updater
 - Applies to whole Network
 - How weights are updated
 - Example Stochastic Gradient Descent



Simple Network Setting

- LearningRate 0.005
- Updater Nesterovs variant of Stochastic Gradient Descent with Momentum
- Output Activation Identity
- Hidden Layer Activation TANH
- Number of Epochs 500



Extrapolate from Simple Network

- Simple Network took one value in and one value out and trained the network to learn the correct value
- More complex data same process
- Input can be images, row data, documents
- Output can be T/F, range of values, labels of a class



VGG-16 Demo

- Instructor demonstration of a more complex Neural Network



Questions

- What is the Learning Rate and what effect does it have on the Neural Network?
- Activation Function is set per Layer or for the whole Network?
- The error of the output compared to the expected output is? A. Loss Function B. Sigmoid Function C. Learning Rate
- Stochastic Gradient Descent is an example of A. Updater B. Activation Function C. Loss Function



Numpy Scikit-learn basics

Numpy and Scipy

- Took python from a general programming language to a very powerful matrix-oriented one.



Pandas

- Pandas brought dataframes to python.
 - Dataframes are one of the core concepts in modern data analysis.



Scikit-learn

- Implementations of best-of-breed algorithms
- Standardized library.



Numpy, Scipy, Scikit-learn, PANDAS

- Help make python the programming language of choice of data scientists



Preprocessing with PANDAS

- Read Data
- Select columns and rows
- Filtering
- Time Series



NumPY, SciPY

- Arrays
- Indexing, Slicing, Iterating
- Reshaping
- Matrices



SCIKIT-LEARN

- Machine Learning
 - Feature extraction
 - Classification
 - Regression
 - Clustering
 - Dimension reduction
 - Model selection



Keras Building a Simple Model

The Steps of Defining and using a Neural Network

1. Load Data
2. Describe the model
3. Build the model
4. Train the model
5. Use model for inference



Data Description

- Iris Dataset

```
5.1,3.5,1.4,0.2,Iris-setosa  
7.0,3.2,4.7,1.4,Iris-versicolor  
7.7,2.8,6.7,2.0,Iris-virginica
```



Load Data with Pandas

```
dataframe = pandas.read_csv("iris.csv", header=None)
dataset = dataframe.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]
print(X)
print(Y)
```



Encode Class Values as Integers

```
encoder = LabelEncoder()  
encoder.fit(Y)  
encoded_Y = encoder.transform(Y)
```



convert integers to dummy variables (hot encoded)

```
dummy_y = np_utils.to_categorical(encoded_Y)  
print(dummy_y)
```



Describe the model

- Keras

```
model = Sequential()  
model.add(Dense(4, input_dim=4, activation='relu'))  
model.add(Dense(3, activation='sigmoid'))
```



Build the model

- Keras

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=
```



Train the model

- Keras

```
model.fit(X, dummy_y, nb_epoch=200, batch_size=5)
```



Use the Model for Inference

- Keras

```
prediction = model.predict(numpy.array([[4.6,3.6,1.0,0.2]]));
```



Neural Network Internals

This section describes key configuration details of Neural Networks

- Activation Functions
- BackPropagation
- Loss Functions
- Weight Initialization
- Data Normalization and Standardization



Activation Functions

What is an Activation Function

- Determines output of Neuron Based on Inputs
- Non-Linear Transform function at each node
- Defined per layer
- Allow neural networks to make complex boundary decisions for features at various levels of abstraction.



Common Activation Functions

DeepLearning4J and Keras support the following Activation functions

- CUBE
- ELU
- HARDSIGMOID
- HARDTANH
- IDENTITY
- LEAKYRELU



Supported Activation Functions...

- RATIONALTANH
- RELU
- RRELU
- SIGMOID
- SOFTMAX
- SOFTPLUS
- SOFTSIGN
- TANH












Commonly Used Activation Functions

- Sigmoid
- TanH
- Relu



Activation Functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU)		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Activation Function and output

- Activation Function on output is special
 - Classification = softmax
 - regression = identity



Output Layer Activation Guidelines

- Classification
 - softmax activation
 - Negative Log Likelihood for loss Function
 - MXCENT
 - Multi Class Cross Entropy
- Softmax
 - Probability Distribution over classes
 - Outputs sum to 1.0
- Regression
 - Identity Activation
 - MSE(Mean Squared Error) Loss Function

activation function for the output layer: this is usually application specific. For classification problems, you generally want to use the softmax activation function, combined with the negative log likelihood / MCXENT (multi-class cross entropy).

The softmax activation function gives you a probability distribution over classes (i.e., outputs sum to 1.0). For regression problems, the “identity” activation function is frequently a good choice, in conjunction with the MSE (mean squared error) loss function.



Back Propagation and Updaters/optimizers

What is BackPropagation

- The process of updating the weights of a Neural Network to reduce error
- The Forward Pass generates an output
- The Loss Function calculates the error
- Gradient Descent is used to minimize the error
- Steps are repeated and network continues to improve



Updaters/Optimizers

- List of Keras Supported Optimizers
 - <https://keras.io/optimizers/>

Backpropagation learning is the same general idea as the perceptron learning algorithm. We want to compute the input example's output with a forward pass through the network. If the output matches the label then we don't do anything. If the output does not match the label, then we need to adjust the weights on the connections in the neural network.



Gradient Descent in Weight Space

- Use Josh's picture here

We consider backpropagation to be doing gradient descent in weights space where the gradient is on the error surface. This error surface describes the error of the input features as a function of the weight values in the neural network.

Backpropagation and Fractional Error Responsibility The general idea with backpropagation is how we distribute the “blame” backwards through the network. Each hidden node sending input to the current node is somewhat “responsible” for some portion of the error in each of the neurons it has a forward connection to. The first hidden layer uses the input from the raw feature vector as input. All subsequent layers use the activation value of the previous layer’s neurons as input. However, for hidden layers before the output layer we have to divide up this error appropriately. With error backpropagation we divide the Δ_i values according to the connection weight between the hidden node and the output node. To calculate Δ_j in the code above we see the equation: $\Delta_j = g' \text{ input_sum}_j \sum_i W_{ji} \Delta_i$. This equation takes a look at each node i in the current layer and takes the current error value Δ_i and multiplies it by the weight on the incoming connection times the derivative of the activation function. This produces the fractional error value for the node in the previous layer which is used as we update the incoming connections to that layer. We progressively perform this algorithm

Backpropagation and Mini-Batch Stochastic Gradient Descent In chapter 1 we learned about a variant of stochastic gradient descent called “minibatch” where we train the model on multiple examples at once as opposed to a single example at a time. We see mini-batch used with backpropagation and stochastic gradient descent in neural networks as well to improve training. Under the hood we’re computing the average of the gradient across all the examples inside the mini-batch. Specifically we compute the forward pass for all of the examples to get their output scores as a batch linear algebra matrix operation. During the backward pass for each later we are computing the average of the gradient (for the 88 | Chapter 2: Foundations of Neural Networks layer). By doing backpropagation this way we’re able to get a better gradient approximation and use our hardware more efficiently at the same time.

Adapting to Changing Error

- Initially error will be large
- Output more or less random
- Two approaches
 - Dynamic Learning Rate
 - Use Adaptive Optimizer



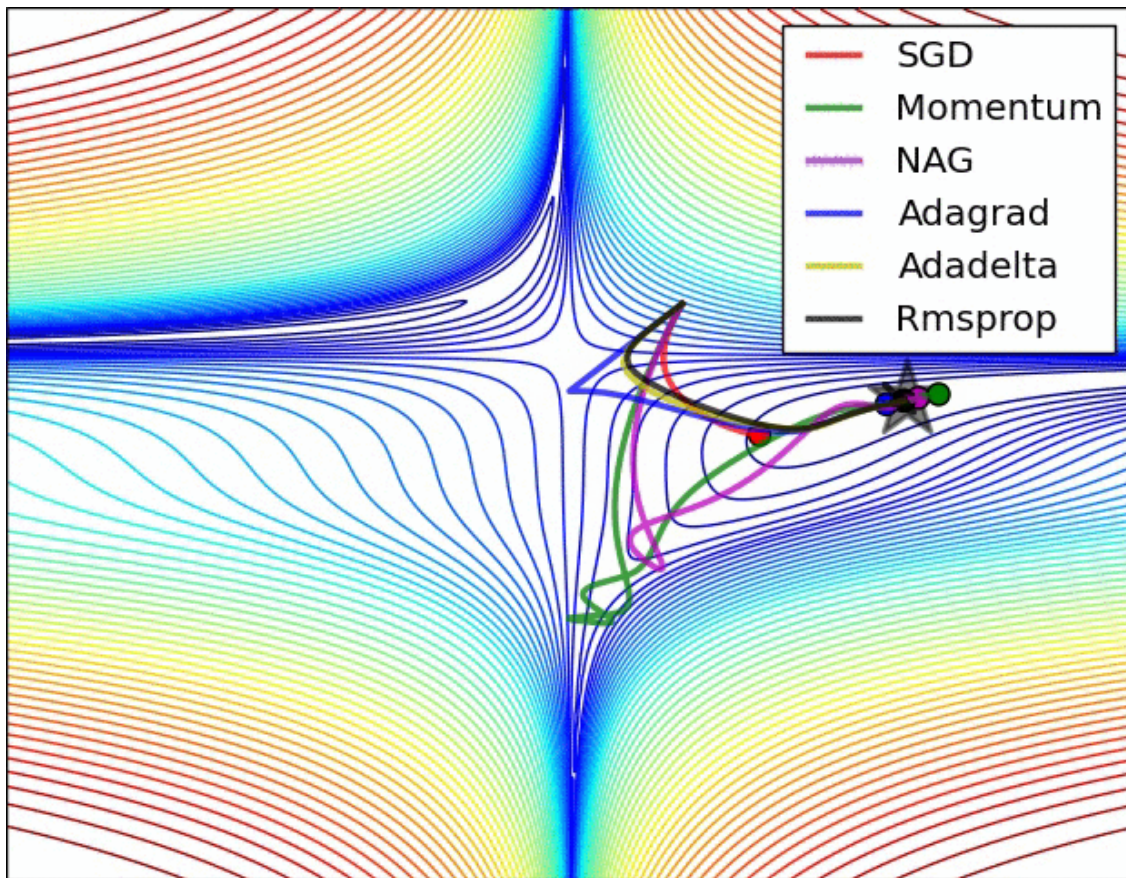
Adaptive Optimizers / Dynamic Learning Rate a Simplified View

- Initially you want the network to train quickly
 - Error is large
 - Take Large Steps
- As Error decreases
 - Smaller steps are better



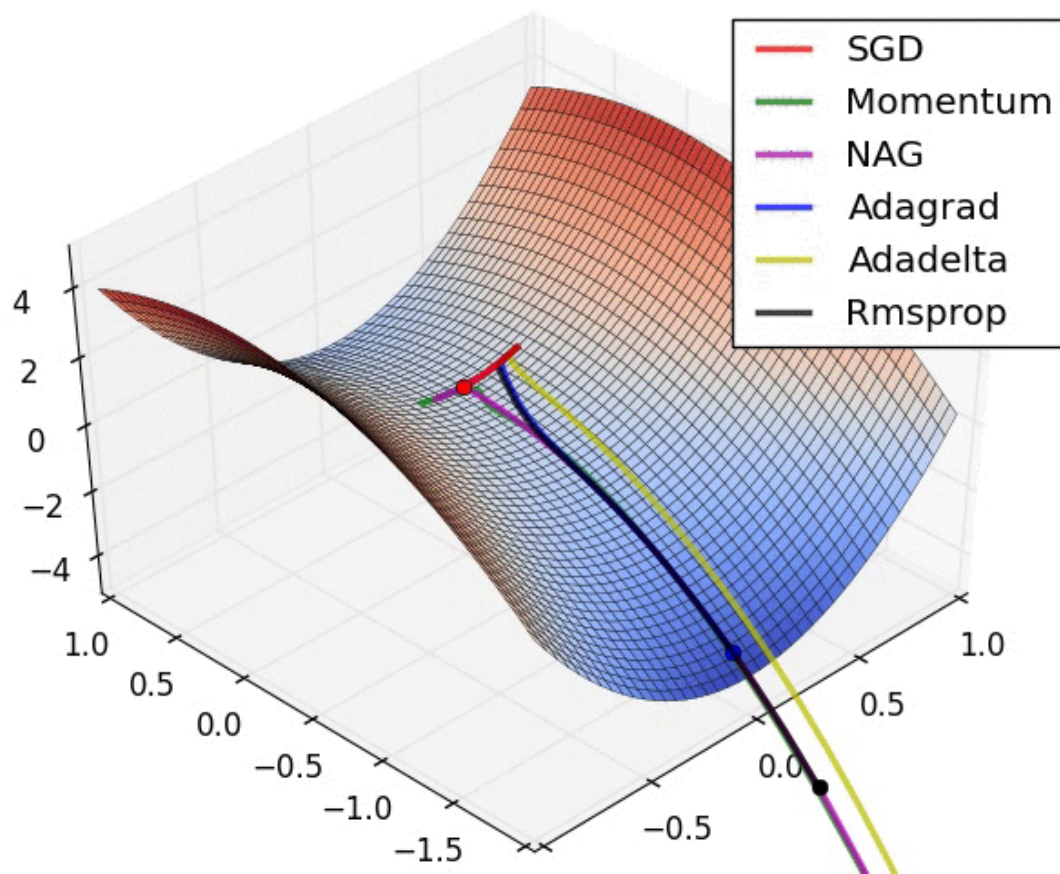
Updater Animation

- Thanks to Alec Radford



Updater Animation

- Thanks to Alec Radford



Loss Functions

Loss functions quantify how close a given neural network is to the ideal it is training towards.

The three important functions at work in machine learning optimization:

- parameters
 - transform input to help determine the classifications a network infers
- loss function
 - gauges how well it classifies (minimizes error) at each step
- optimization function
 - guides it toward the points of least error.



AVAILABLE LOSS FUNCTIONS

- For Regression
- MSE(Mean Squared Error)
- Mean Absolute Error(MAE)
- Mean Squared Log Error (MSLE)
- Mean Absolute Percentage Error(MAPE)



Regression Loss Function Common Usage

- MSLE and MAPE handle large ranges,
 - common practice to normalize input to suitable range and use MSE or MAE



Loss Functions for Classification

- Hinge Loss
 - Hard Classification 0,1 a -1,1 classifier
- Logistic Loss
 - Probabilities per class

Logistic loss functions are used when probabilities are of greater interest than hard classifications. Great examples of these would be flagging potential fraud, with a human-in-the-loop solution or predicting the “probability of someone clicking on an ad”, which can then be linked to a currency number.

Predicting valid probabilities means generating numbers between 0 and 1. Predicting valid probabilities also mean making sure the probability of mutually exclusive outcomes should sum to one.

For this reason, it is essential that the very last layer of a neural network used in classification is a softmax. Note that the sigmoid activation function will also give valid values between 0 and 1.

However it cannot be used in scenarios where the outputs are mutually exclusive, as it does not model the dependencies between the output values.



Negative Log Likelihood

- Likelihood
 - between 0 and 1
- Log likelihood
 - between negative infinity and 0
- Negative log Likelihood
 - between 0 and infinity

For the sake of mathematical convenience, when dealing with the product of probabilities it is customary to convert them to the log of the probabilities.

And hence the product of the probabilities transforms to the sum of the log of the probabilities.



Weight Initialization

Weight Initialization: The Challenge

- Weights too small
 - Signal shrinks as it passes through layers
 - Becomes too small to be useful
- Weights too large
 - Signal Grows as it passes through Layers
 - Becomes too large to be useful



Xavier Distribution

- 0 mean and a specific variance
 - $\text{Var}(W)=1/n_{\text{In}}$

where W is the initialization distribution for the neuron in question, and n_{In} is the number of neurons feeding into it. The distribution used is typically Gaussian or uniform.



Benefits of Xavier

- Xavier Enabled Full Network Training vs per-Layer Pre-Training
- Big Breakthrough

Xavier initialization was one of the big enablers of the move away from per-layer generative pre-training.



Alternatives to Xavier

- Relu
- Works well with CNN's and Relu activations



Data Normalization and Standardization

Normalization

- Convert to range of values 0-1
 - Max Value becomes 1
 - Min Value becomes 0



Standardization

- Convert Values to
 - mean 0
 - Standard Deviation of 1



Main Topologies of Neural Networks

Common Neural Network Architectures

- MLP(Multi Layer Perceptron) or FeedForward Neural Network
- Recurrent Neural Network
- Convolutional Neural Network



Multi Layer Perceptron

- Good for General Use
- single input maps to single output
- Number of classes = Number of output Nodes



Recurrent Neural Networks

- LSTM (Long Short Term Memory)
- Goodwith Sequence Data
- Ability to Learn time dependencies that effect output
- One to many, many to one



Convolutional Neural Networks

- Convolutional Layers
- 3 dimensional input
- Learned filters Convolve over image



Tuning Neural Networks

Hyper-Parameters that may need tuning

- Learning Rate
- Batch Size



Learning Rate Guidelines

- 0.1 to 0.000001



Learning Rate

- Adaptive Learning Rate
- Adjust optimizer based on previous updates
 - Nesterovs Momentum
 - Adagrad, Adadelata, Adam, RMSProp



Learning Rate Schedules

- Tune Learning Rate as Learning Progresses
- Based on Schedule or other metrics



Early Stopping

- Stop Training once overfitting is detected



Avoiding Overfitting

- What is Overfitting
 - Scores well on test
 - Scores poorly on unseen examples
- Has "memorized" training data
- Fails to generalize



Regularization

- L1 and L2 Regularization
 - penalizes large network weights
 - avoids weights becoming too large
- Risks
 - Coefficients too high
 - Network stops learning
- Common values for L2 regularization
 - 10^{-3} to 10^{-6} .

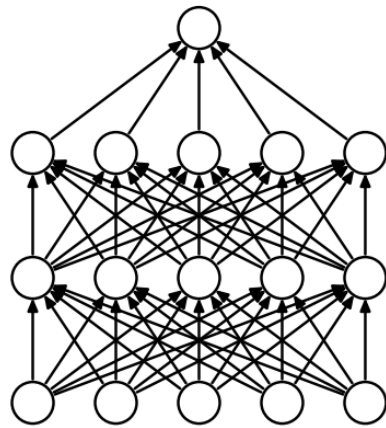


Dropout

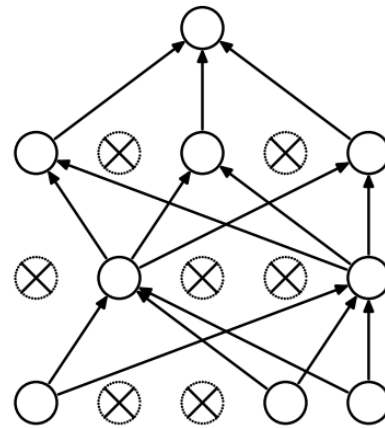
- Set probability that a Neuron will be de-activated, set activation to 0
- Forces Network to learn different redundant representations
- Commonly used dropout rate of 0.5.



Dropout



(a) Standard Neural Net



(b) After applying dropout.

So, if you set half of the activations of a layer to zero, the neural network won't be able to rely on particular activations in a given feed-forward pass during training.

As a consequence, the neural network will learn different, redundant representations; the network can't rely on the particular neurons and the combination (or interaction) of these to be present.

Another nice side effect is that training will be faster.

Take this from lightning talk

Importing Keras

Key topics

- Model Import
- Transfer Learning
- Model Serializer



Ecosystem overview

Subsections

- GPU's
- Spark
- Hadoop Integration

