DEEP LEARNING FOR TRADING

Yam Peleg
OUR GOAL
We want to predict the future.
OUR GOAL

We want to predict the future.

DEEP LEARNING FOR TRADING
Each example in the training data is a *pair* consisting of an input vector (features) and a desired output value (labels).

A supervised learning algorithm analyzes the training data and approximate a function, which can be used for mapping new unlabeled examples.
FINANCIAL PREDICTION PITFALLS

The longer the time frame, the more difficult it will be to accurately forecast financial results.

**Importance**

Data Importance is questionable and determination of meaningful data is hard.

**Overfitting**

Overfitted easily, most models have poor predictive capabilities on financial data.

**Noisy Data**

Noise in financial data is very common and sometimes distinguishing noise from behavior is hard.

**Behavior**

Behavior of financial markets change all the time and can be really unpredictable.

**Much Data**

Possible relevant data from many markets is incredibly large.

**No Theory**

Complex non-linear interactions in the data are not well specified by financial theory.

DEEP LEARNING FOR TRADING
WHY DEEP LEARNING?

This is why.

**Importance**

Data Importance is questionable and determination of meaningful data is hard.

**No Theory**

Complex non-linear interactions in the data are not well specified by financial theory.

**Overfitting**

Overfitted easily, most models have poor predictive capabilities on financial data.

**Noisy Data**

Noise in financial data is very common and sometimes distinguishing noise from behavior is hard.

**Behavior**

Behavior of financial markets change all the time and can be really unpredictable.

**Much Data**

Possible relevant data from many markets is incredibly large.
LINEAR REGRESSION
\[ y_i = \beta_0 + \beta_1 x_{i_1} + \beta_2 x_{i_2} + \cdots + \beta_n x_{i_n} + \varepsilon \]
Perceptron
The Artificial Neuron

Input \( W \) \( W \) \( W \)

\[ \sum \]

\[ f \]

Output

Yam Peleg
DEEP LEARNING FOR TRADING
Neural Network

One Layer Perceptron

Perceptron Layer
(Hidden Layer)

Input Layer

Output Layer

DEEP LEARNING FOR TRADING
GRADIENT BASED MODELS

1: Forward Propagation

\[ f_n(x_n, w_n) = \hat{y} \]

2: Loss Calculation

\[ E = l(\hat{y}, y) \]

3: Optimization

\[ \frac{\partial E}{\partial x_n} = \frac{\partial l(\hat{y}, y)}{\partial x_n} \]

\[ \frac{\partial E}{\partial w_n} = \frac{\partial E}{\partial f_n(x_{n-1}, w_n)} \frac{\partial f_n(x_{n-1}, w_n)}{\partial w_n} \]

\[ \frac{\partial E}{\partial w_{n-1}} = \frac{\partial E}{\partial f_n(x_{n-2}, w_{n-1})} \frac{\partial f_n(x_{n-2}, w_{n-1})}{\partial w_{n-1}} \]

Legend

\( y \) – Ground Truth
\( x_0 \) – Features Vector
\( x_i \) – Output of \( i \) layer
\( w_i \) – Weights of \( i \) layer
\( \hat{y} \) – Model Output
\( l(\hat{y}, y) \) – Loss Function
\( E \) – Loss Surface

Classic SGD

\[ v_t = \mu v_{t-1} - \alpha L_t(w_{t-1}) \]

\[ w_t = w_{t-1} - v_t \]

AdaGrad

\[ w_t = w_{t-1} - \alpha \frac{v^T_t v_t}{\sqrt{v^T_t v_t} + \epsilon} \]

RMSProp

\[ R_t = \gamma R_{t-1} + (1 - \gamma) |L_t(w_{t-1})|^2 \]

\[ w_t = w_{t-1} - \alpha \frac{v^T_t v_t}{\sqrt{v^T_t v_t} + \epsilon} \]

Adam

\[ R_t = \beta_1 R_{t-1} + (1 - \beta_1) |L_t(w_{t-1})|^2 \]

\[ w_t = w_{t-1} - \alpha \frac{v^T_t v_t}{\sqrt{v^T_t v_t} + \epsilon} \]

Yam Peleg

DEEP LEARNING FOR TRADING
Perceptron: It is a type of linear classifier, a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. The algorithm allows for online learning, in that it processes elements in the training set one at a time.

**FEED FORWARD**

**Feed Forward Network**
- Sometimes referred to as MLP, is a fully connected dense model used as a simple classifier.

**Convolutional Network**
- Assumes that highly correlated features located close to each other in the input matrix can be pooled and treated as one in the next layer.
- Known for superior Image classification capabilities.

**SUPERVISED**

**Simple Recurrent Neural Network**
- A class of artificial neural network where connections between units form a directed cycle.

**Hopfield Recurrent Neural Network**
- It is a RNN in which all connections are symmetric. It requires stationary inputs.

**Long Short Term Memory Network**
- Contains gates that determine if the input is significant enough to remember, when it should continue to remember or forget the value, and when it should output.

**RECURRENT**

**UNSUPERVISED**

**Auto Encoder**
- Aims to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction.

**Restricted Boltzmann Machine**
- Can learn a probability distribution over its set of inputs.

**Deep Belief Net**
- Is a composition of simple, unsupervised networks such as restricted Boltzmann machines, where each sub-network’s hidden layer serves as the visible layer for the next.

---

Yam Peleg

**DEEP LEARNING FOR TRADING**
DEEP LEARNING SUPERIORITY
Deep Learning is better than humans on certain Image recognition tasks.

Human: 94.9%
Deep Learning: 96.92%

ref: http://www.image-net.org/challenges/LSVRC/
Implementing deep neural networks for financial market prediction, Dixon et al, 2015
Each example in the training data is a pair consisting of an input vector and again the input vector.

The goal is to learn a function that describes the hidden structure from unlabeled data.
Auto Encoders

For learning the distribution of the features

Features
- Past Prices
- Correlations
- Technical Analysis
- Z Score
- Time Features

Recommended Papers
Unsupervised Pretraining

For better approximation

Recommended Papers


Features
- Past Prices
- Correlations
- Technical Analysis
- Z Score
- Time Features

Gr Leaming for Trading

Input Layer
- Hidden Layers
- Output Layer

Past Prices
- Correlations
- Technical Analysis
- Z Score
- Time Features

Future Prices
- Regression
- Up or Down Analysis

Ground Truth
Deep Learning Hardware

Deep learning is often done on the GPU or other powerful devices.

- Better for Matrix algebra
- Parallel calculations
- Much more powerful
Deep Learning Framework

Deep learning is often done on the GPU or other powerful devices

Caffe
Theano
TensorFlow™
Lua

Framework
Framework
Driver + Lib
Hardware
Deep Learning Using Python

Deep learning is often done on the GPU or other powerful devices.

- Language: Python
- Framework: Theano, Lasagne, nolearn
- Framework: cuDNN
- Driver + Lib: NVIDIA CUDA
- Hardware: GPU

Yam Peleg
DEEP LEARNING FOR TRADING
Python Stays Python

Deep learning is often done on the GPU or other powerful devices

```python
import theano.sandbox.cuda
theano.sandbox.cuda.use("gpu")
```
Theano

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.
YO DAWG, I HEARED YOU LIKE LANGUAGES

SO I PUT A LANGUAGE INSIDE YOUR LANGUAGE
Theano Tutorial

Shared Variables

In [1]: import numpy, theano
np_array = numpy.ones(2, dtype='float32')

s_false   = theano.shared(np_array, borrow=False)
s_true    = theano.shared(np_array, borrow=True)

np_array += 1
print(s_false.get_value())
print(s_true.get_value())

Out [1]:
[ 1.  1.]
[ 2.  2.]

Variables
A Theano Variable is a Variable with storage that is shared between functions that it appears in.
Theano Tutorial

When using theano.function you're compiling C code performing your tasks under the hood. This is what makes Theano fast.

In [1]: import theano
   x = theano.tensor.dscalar()
   f = theano.function([x], 2*x)
   f(4)

Out [1]: array(8.0)

Functions
The idea here is that we've compiled the symbolic graph (2*x) into a function that can be called on a number and will do some computations.
Theano Tutorial

Gradients: computes the derivative of some expression

In [1]: import numpy
   import theano
   import theano.tensor as T
   from theano import pp
   x = T.dscalar('x')
   y = x ** 2
   gy = T.grad(y, x)
   pp(gy)  # print out the gradient prior to optimization

Out [1]: '((fill((x ** TensorConstant{2}), TensorConstant{1.0}) * TensorConstant{2}) * (x ** (TensorConstant{2} - TensorConstant{1})))'

In [2]: f = theano.function([x], gy)
   f(4)

Out [2]: array(8.0)

Gradients
Now let’s use Theano for a slightly more sophisticated task: create a function which computes the derivative of some expression \( y \) with respect to its parameter \( x \).
LINEAR REGRESSION

\[ \hat{y} = wx \]

\[ l = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \]

\[ w = w - \alpha \frac{\partial l}{\partial w} \]
**Theano Tutorial**

**Gradient based linear regression**

In [1]: def model(X, weights):
    return X * weights

w = theano.shared(np.asarray(0., dtype=theano.config.floatX))
y = model(X, weights)

Loss = T.mean(T.sqr(y - Y))
gradient = T.grad(loss, weights)
updates = [[weights, weights - gradient * learning_rate]]

train = theano.function(inputs=[X, Y], outputs=loss, updates=updates, allow_input_downcast=True)

for i in range(epoches):
    for x, y in zip(X, Y):
        train(x, y)
GRADIENT BASED MODELS

1: Forward Propagation

def model(X, weights)
    ...

2: Loss Calculation

    Loss = ...
    gradient = T.grad(loss, weights)

3: Optimization

    updates = [[weights, weights - gradient]]

Legend

\( l \) - Ground Truth
\( x_0 \) - Features Vector
\( x_i \) - Output of \( i \) layer
\( w_i \) - Weights of \( i \) layer
\( \hat{y} \) - Model Output
\( l(\hat{y}, y) \) - Loss Function
\( E \) - Loss Surface
\( f \) - Activation Function
The core data structure of Keras is a model, a way to organize layers. The main type of model is the Sequential model, a linear stack of layers.

In [1]: from keras.models import Sequential
from keras.layers.core import Dense, Activation

model = Sequential()

model.add(Dense(output_dim=..., input_dim=...))
model.add(Activation(...))
In [1]: from keras.models import Sequential
   from keras.layers.core import Dense, Activation

model = Sequential()
model.add(Dense(output_dim=64, input_dim=100))
model.add(Activation("relu"))
model.add(Dense(output_dim=24, input_dim=64))
model.add(Activation("relu"))
model.add(Dense(output_dim=10))
model.add(Activation("softmax"))

model.compile(loss='categorical_crossentropy', optimizer='sgd')
model.fit(X,Y)
Implementing deep neural networks for financial market prediction, Dixon et al, 2015
Deep Neural Networks

For complex function approximation

Features
- Past Prices
- Correlations
- Technical Analysis
- Z Score
- Time Features

Future Prices
- Regression
- Up or Down Classification

Ground Truth

Deep Learning for Trading

model = Sequential()
model.add(Dense(output_dim=64, input_dim=100))
model.add(Activation("relu"))
model.add(Dense(output_dim=24, input_dim=64))
...
model.compile(loss='categorical_crossentropy', optimizer='sgd')
model.fit(X,Y)
Auto Encoders
For learning the distribution of the features

Recommended Papers
Auto Encoders

For learning the distribution of the features

Features
- Past Prices
- Correlations
- Technical Analysis
- Z Score
- Time Features

Input Layer → Hidden Layers → Output Layer

model = Sequential()
model.add(Dense(output_dim=64, input_dim=100))
model.add(Activation("relu"))
model.add(Dense(output_dim=24, input_dim=64))
...
model.compile(loss='categorical_crossentropy', optimizer='sgd')
model.fit(X,X)
Questions?