Deep Learning for Design and Verification Engineers

John Aynsley
Deep Learning for Design and Verification Engineers

What is Deep Learning?

Neural Networks

How a Network Learns

Getting Started
"Giving computers the ability to learn without being explicitly programmed"
Machine Learning Algorithms

Classification
Regression
Neural networks
Support vector machines
Dimensionality reduction
Bayesian statistics

Hebbian learning
Markov models
Decision tree learning
Random forests
Reinforcement learning
Evolutionary algorithms

Used in statistics and data science
Deep Learning

Deep learning is very simple algorithms
applied very intensively to massive datasets
and often applied end-to-end

This is not a definition, just an intuitive description!
Machine Learning and Deep Learning

Machine learning is not new

Deep learning is new

Deep learning is growing fast because it really works!
Supervised Learning

Input

Output

Learn to predict the output from the input

Deployment

Model

Inference

Novel input

Predicted output

Platform

GPU, IC, ASIC, or FPGA

GPU
Unsupervised Learning

Examples

Partition the data into clusters based on similarity
Find hidden patterns in the data
Pick out anomalous data samples
Supervised Learning with a Neural Network

Labels
- Persian
- Abyssian
- Siamese
- British Shorthair
- Scottish Fold
- Burmese

Training data

Ground Truth

Compare

Cost Function

Neural Network

Prediction

Deep Learning: >1 hidden layer
Why Neural Networks Now?

2012

Bigger datasets
Faster computers

Since 2012

Improved neural network architectures
Neural networks often outperforming previous state-of-the-art
Image Classification
The ImageNet Challenge (ILSVRC)

ImageNet Large Scale Visual Recognition Challenge: 1.2M images in 1000 categories

Depends what you measure!

<table>
<thead>
<tr>
<th>Year</th>
<th>Network</th>
<th>#Layers</th>
<th>Top-5 Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012 winner</td>
<td>AlexNet (CNN)</td>
<td>8</td>
<td>15.3%</td>
</tr>
<tr>
<td>2013 winner</td>
<td>ZFNet (CNN)</td>
<td>8</td>
<td>14.8%</td>
</tr>
<tr>
<td>2014</td>
<td>VGGNet (CNN)</td>
<td>19</td>
<td>7.3%</td>
</tr>
<tr>
<td>2014 winner</td>
<td>GoogLeNet (Inception)</td>
<td>22</td>
<td>6.7%</td>
</tr>
<tr>
<td>2015 winner</td>
<td>ResNet (residual)</td>
<td>152</td>
<td>3.6%</td>
</tr>
<tr>
<td>2016 winner</td>
<td>CUImage (ensemble)</td>
<td>-</td>
<td>3.0%</td>
</tr>
</tbody>
</table>

Dramatic improvement

Human error rate ~ 5%

3% bad labels

Training typically takes a few weeks on a few GPUs
Natural Language Translation

Try the API:

Source Language: English (en)

Target Language: French (fr)

It was raining cats and dogs and I had forgotten my umbrella.

Il y avait des chiens et des chiens et j'avais l'idée d'être un monstre.
Other Exciting Applications

Image segmentation
Adding captions to images
Adding color to black-and-white images
Generating images that mimic other artists or styles
Spotting significant events in videos
Human pose estimation to analyze behavior
Handwriting recognition
Voice recognition and voice generation
Predicting patterns in natural phenomena
Playing games
Cool Stuff

http://www.yaronhadad.com/deep-learning-most-amazing-applications/

https://deepdreamgenerator.com
Deep Learning for Design and Verification Engineers

What is Deep Learning?

Neural Networks

How a Network Learns

Getting Started
A Neural Network?

In the academic literature
E.g. GoogLeNet
An Artificial Neuron

Input units

Hidden unit

\[ u = \sum_{i=1}^{3} w_i x_i + b \]

\[ y = \text{activation}(u) \]
Common Activation Functions

- **Step**
  - The Perceptron

- **Sigmoid aka logistic**
  - Early neural networks and still used today

- **ReLU**
  - Popular today
A Deep Neural Network

\[ y_j = \text{RELU} \left( \sum_{i=1}^{3} w_{ji}x_i + b_j \right) \]

\[ y_j = \text{RELU} \left( \sum_{i=1}^{n} w_{ji}x_i + b_j \right) \]
Kinds of Neural Network

ANN – Artificial Neural Network

CNN – Convolutional Neural Network (e.g. image processing)

R-CNN – Regional CNN (image segmentation)

RNN – Recursive Neural Network (e.g. natural language processing)

GAN – Generative Adversarial Network
Deep Learning for Design and Verification Engineers

What is Deep Learning?

Neural Networks

How a Network Learns

Getting Started
Regression Task

![Graph showing a scatter plot with input x and output y. The graph visually represents the relationship between input and output values.]
Define a Hypothesis / Model / Network

\[ h_{\text{slope,offset}}(x) = \text{slope} \cdot x + \text{offset} \]

```python
def hypothesis(x, slope, offset):
    return slope * x + offset
```
Cost Function

\[ J(slope, offset) = \frac{1}{2m} \sum_i (h(x_i) - y_i)^2 \]

```python
def cost(x, slope, offset, y):
    return np.mean(np.square(hypothesis(x, slope, offset) - y)) / 2.0
```
Cost as a Function of Slope and Offset
Contour Plot of Cost Function
Gradient Descent

\[ \frac{\partial J}{\partial \text{slope}} \]

slope \leftarrow \text{slope} - \alpha \cdot \frac{\partial J}{\partial \text{slope}}

\[ \frac{\partial J}{\partial \text{offset}} \]

offset \leftarrow \text{offset} - \alpha \cdot \frac{\partial J}{\partial \text{offset}}
Gradient Descent Algorithm

```
num_steps = 3000
learning_rate = 0.001

slope = 4.0
offset = -4.0

def derivative_wrt_slope():
    return np.mean((hypothesis(x, slope, offset) - y) * x)

def derivative_wrt_offset():
    return np.mean((hypothesis(x, slope, offset) - y))

for step in range(num_steps):
    new_slope = slope - learning_rate * derivative_wrt_slope()
    new_offset = offset - learning_rate * derivative_wrt_offset()
    slope = new_slope
    offset = new_offset
```
Converging on the Minimum

Final slope = -2.31499114425  offset = 4.38960555415
Cost, Slope, Offset against Step

Ground truth = 5

Ground truth = -2
Stochastic Gradient Descent

Use a subset of the training data at each step
Forward and Back-Propagation

Forward propagation calculates weighted sums and activation function.

Back propagation calculates gradients.
Classification Task

Given a new point, classify it as red, green, or blue
One-Hot Labels

Category:

Red  Green  Blue

Numeric label (single output):

1  2  3

One-hot label (3 outputs):

\[
\begin{pmatrix}
1 \\
0 \\
0 \\
\end{pmatrix}
\quad \begin{pmatrix}
0 \\
1 \\
0 \\
\end{pmatrix}
\quad \begin{pmatrix}
0 \\
0 \\
1 \\
\end{pmatrix}
\]

Would introduce a bias

No bias
The Hypothesis or Model

\[ \vec{y} = W \vec{x} + \vec{b} \]

\[ y_j = \sum_{i=1}^{3} w_{ji} x_i + b_j \]

Trainable parameters

Weights

Biases

3 labels red, green, blue

2 features
Calculating the Cost Function

Inputs

\[ x_1 \]
\[ x_2 \]

Outputs

\[ y_1 \]
\[ y_2 \]
\[ y_3 \]

Possible predictions

\[
\begin{pmatrix}
5.2 \\
0.01 \\
-0.3
\end{pmatrix}
\begin{pmatrix}
8.1 \\
0.2 \\
3.9
\end{pmatrix}
\begin{pmatrix}
0.1 \\
0.2 \\
0.1
\end{pmatrix}
\]

Labels

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]
Converting Scores to Probabilities

The probabilities of a given label occurring
The Softmax Function

\[
\sigma(y)_j = \frac{e^{y_j}}{\sum_k e^{y_k}}
\]

\[
\begin{align*}
0.0 & \rightarrow (0.301) \quad \text{compare} \quad (0) \\
+0.1 & \rightarrow (0.332) \quad \text{compare} \quad (1)
\end{align*}
\]

\[
\begin{align*}
1.0 & \rightarrow (0.090) \quad \text{compare} \quad (0) \\
2.0 & \rightarrow (0.245) \quad \text{compare} \quad (0) \\
3.0 & \rightarrow (0.665) \quad \text{compare} \quad (1)
\end{align*}
\]

\[
\begin{align*}
10 & \rightarrow (2 \times 10^{-9}) \quad \text{compare} \quad (0) \\
20 & \rightarrow (5 \times 10^{-5}) \quad \text{compare} \quad (0) \\
30 & \rightarrow (9.999) \quad \text{compare} \quad (1)
\end{align*}
\]
Compare using Cross-Entropy

$$H(\sigma, \ell) = -\sum_i \ell_i \log(\sigma_i)$$

The closer the softmax value corresponding to the given label is to 1, the closer the cost is to 0.
A Neural Network for Classification

Inputs

Labels

Logits

Outputs

Probabilities

Cost

softmax

cross-entropy

$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$

$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$

$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$
Predictions, Confidence > 0.8
The Decision Boundary
Classes are Linearly Separable
Non-Linear Regression and Classification
Piecewise Linear Approximation

\[ y = \sum_{i=1}^{4} w_i x_i + b \]

\[ y_j = \text{RELU}(w_j x + b_j) \]
The Predicted Output

Such a network is hard to train
The Landscape of the Cost Function

Local and global minima
A Deep Neural Network

Input unit  16 hidden units  16 hidden units  Output unit

\[ y = \sum_{i=1}^{4} w_i x_i + b \]

\[ y_j = \text{RELU}(w_j x + b_j) \]

Cost function (mean squared error)

Ground truth
The Predicted Output
Deep neural networks have many degrees of freedom / degenerate / redundant

Gradient descent tends not to get stuck in local minima

Gradient descent tends to find a good global minimum

Why?

Most stationary points are saddle points, not local minima?
What is Deep Learning?

Neural Networks

How a Network Learns

Getting Started
# Libraries and Frameworks for Training

<table>
<thead>
<tr>
<th>Organisation/Company</th>
<th>Framework</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>University of Montreal</td>
<td><strong>Scikit-learn</strong></td>
<td>Not deep learning</td>
</tr>
<tr>
<td>University of Montreal</td>
<td><strong>Theano</strong></td>
<td>Runs on GPU</td>
</tr>
<tr>
<td></td>
<td><strong>Pylearn2</strong></td>
<td>(Theano)</td>
</tr>
<tr>
<td></td>
<td><strong>Lasagne</strong></td>
<td>(Theano)</td>
</tr>
<tr>
<td>Berkeley AI Research</td>
<td><strong>Caffe</strong></td>
<td>Runs on GPU</td>
</tr>
<tr>
<td>Facebook, Twitter, ...</td>
<td><strong>Torch, PyTorch</strong></td>
<td>Runs on GPU</td>
</tr>
<tr>
<td>Google</td>
<td><strong>TensorFlow</strong></td>
<td>Runs on GPU</td>
</tr>
<tr>
<td>Microsoft</td>
<td><strong>CNTK</strong></td>
<td>Cognitive Tool Kit</td>
</tr>
<tr>
<td></td>
<td><strong>Keras</strong></td>
<td>(Theano, TensorFlow, CNTK)</td>
</tr>
<tr>
<td>Skymind</td>
<td><strong>DL4J</strong></td>
<td>Deep Learning for Java</td>
</tr>
<tr>
<td>Apache</td>
<td><strong>MXNet</strong></td>
<td></td>
</tr>
<tr>
<td>Intel</td>
<td><strong>Neon</strong></td>
<td>Optimized for Intel CPUs</td>
</tr>
<tr>
<td>MathWorks</td>
<td><strong>MATLAB</strong></td>
<td>Various toolboxes</td>
</tr>
</tbody>
</table>
Deep Learning Platforms and Toolkits

- Amazon Deep Learning AMIs
- Au-Zone DeepView
- Google Cloud Machine Learning Engine
- IBM Watson
- Intel Nervana Cloud
- Microsoft Azure Machine Learning Studio
- MVTec HALCON
- NVIDIA TensorRT
- Qualcomm Snapdragon NPE SDK
- Xilinx reVISION

And many more! All trademarks acknowledged
Deep Learning IP and Chips

BrainChip
Cadence Tensilica Vision DSP
Google TPU
Graphcore
IBM TrueNorth

Intel Loihi
Intel Movidius
KAIST MVLSI Laboratory
KALRAY MPPA
Synopsys DesignWare EV6x

And many more! All trademarks acknowledged
Python and Jupyter Notebook

Simple Example of Non-Linear Regression using Keras

Keras is an API on top of TensorFlow (or Theano, another ML library) that hides and abstracts a lot of the detail when building deep neural network models. Keras models are a lot more compact and readable than low-level TensorFlow models, although because a lot of the detail is obscured, it might not be so clear to beginners what is going on under-the-hood.

First run the code to generate and plot the dataset.

```python
In [43]:
import numpy as np
def generate_data(m=200, train_slope=0.1, train_offset=-1.0):
    X = np.linspace(-10, 10, m)
    y = X * train_slope + train_offset
    return X, y
X, y = generate_data()
import matplotlib.pyplot as plt
plt.plot(X, y, 'o')
```
```python
5  train_slope = 0.1
6  train_offset = -1.0
7
8  train_x = np.linspace(-10, 10, m).astype(np.float32)
9  rng = np.random.RandomState(seed=42)
10  train_y = (train_slope * train_x + np.sin(train_x / 1.5) + train_offset +
              rng.normal(0.0, 0.2, size=len(train_x))).astype(np.float32)
11
13  plt.rcParams["figure.figsize"] = (12, 6)
14  plt.plot(train_x, train_y, 'o');
15  plt.show()
```
Jupyter Notebook Architecture

You

Local computer

Browser

AWS Machine

HTTP

SSH

WinSCP

SSH

Notebook server

IPython Kernel

Notebook files .ipynb

Big GPU
AWS Deep Learning AMI

Apache MXNet, Caffe2, TensorFlow, PyTorch, Keras, CNTK, Theano
SSH Port Forwarding

Jupyter Notebook in local web browser

Amazon EC2 instance with NVIDIA Tesla GPU
Build a neural network model with two hidden layers using the Keras API.

```python
In [47]:
1. import keras
2. from keras.models import Sequential
3. from keras.layers import Dense
4. keras.backend.clear_session()
5. n_hidden = 32
6. model = Sequential()
7. model.add(Dense(input_dim=1, units=n_hidden, activation='relu'))
8. model.add(Dense(units=n_hidden, activation='relu'))
9. model.add(Dense(units=1))
10. model.summary()
11. model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])
```

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 32)</td>
<td>64</td>
</tr>
<tr>
<td>dense_2 (Dense)</td>
<td>(None, 32)</td>
<td>1056</td>
</tr>
<tr>
<td>dense_3 (Dense)</td>
<td>(None, 1)</td>
<td>33</td>
</tr>
</tbody>
</table>

Total params: 1,153
Trainable params: 1,153
Non-trainable params: 0

```python
In [44]:
1. model.fit(train_x, train_y, epochs=10000, batch_size=m, verbose=0)
```
Plot the predicted curve learnt by the model.

In [42]:
1. y = model.predict(train_x)
2. plt.plot(train_x, train_y, 'o')
3. plt.plot(train_x, y, linewidth=5)
4. plt.show()
Classification

Simple Example of Non-Linear Classification using Keras

Generate and plot the dataset, which consists of clusters of colored points.

In [56]:
import random
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import cm
n_features = 2  # The 2 dimensions of each training data point
n_labels = 4  # The number of categories, shown in various colors (up to 6)
n_clusters = 3  # The number of clusters of each color
spread = 10  # The maximum distance between the clusters
m = 202  # The number of datapoints
```python
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import sgd

keras.backend.clear_session()

n_hidden = 16

model = Sequential()
model.add(Dense(input_dim=n_features, units=n_hidden, activation='relu'))
model.add(Dense(units=n_hidden, activation='relu'))
model.add(Dense(units=n_hidden, activation='relu'))
model.add(Dense(units=n_labels, activation='softmax'))
model.summary()

model.compile(loss='categorical_crossentropy', optimizer=sgd(lr=0.05), metrics=['accuracy'])
```

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 16)</td>
<td>48</td>
</tr>
<tr>
<td>dense_2 (Dense)</td>
<td>(None, 16)</td>
<td>272</td>
</tr>
<tr>
<td>dense_3 (Dense)</td>
<td>(None, 16)</td>
<td>272</td>
</tr>
<tr>
<td>dense_4 (Dense)</td>
<td>(None, 4)</td>
<td>68</td>
</tr>
</tbody>
</table>

Total params: 660
Run and Evaluate the Model

Run gradient descent.

```python
In [101]:
1   model.fit(train_x, train_y, epochs=5000, batch_size=m, shuffle=False, verbose=0)
```

```
Out[101]: <keras.callbacks.History at 0x7f416fe2d390>
```

Evaluate the trained model on the training data.

```python
In [102]:
1   loss_and_acc = model.evaluate(train_x, train_y, batch_size=m, verbose=0)
2   print('Accuracy = {:.2f}'.format(loss_and_acc[1]))
```

```
Accuracy = 0.98
```
The Decision Boundary
Softmax Probability for each Label
Training versus Inference

Input

Output

Learn to predict the output from the input

Deployment

Inference

Model

Novel input

Predicted output

Platform

GPU

GPU, IC, ASIC, or FPGA
Weight Quantization

Model trained in 32-bit floating point

Trained on GPUs

TensorFlow Script

Quantized internally to 8 bits

Deployed in target system

Externally equivalent models
For More Information

Example Jupyter Notebook:

www.doulos.com/downloads/dvcon_dl.ipynb

john.aynsley@doulos.com