A Gentle Introduction to Neural Networks (with Python)

Tariq Rashid @postenterprise

EuroPython Bilbao July 2016
… and a live demo!
Background
locate people in this photo

add these numbers

2403343781289312
+ 2843033712837981
+ 2362142787897881
+ 3256541312323213
+ 9864479802118978
+ 897667987987897
+ 8981257890087988
= ?
AI is Huge!

**Google’s DeepMind chalks up AI landmark after beating Go world champion Lee Sedol**

**New Scientist**

**Revealed: Google AI has access to huge haul of NHS patient data**

A data-sharing agreement obtained by *New Scientist* shows that Google’s collaboration with the NHS goes far beyond what it has publicly announced.

**Self-driving cars set to disrupt UK’s £14bn motor insurance industry**

Self-driving cars will result in an 80pc drop in crashes by 2035, experts say.
Mastering the game of Go with deep neural networks and tree search

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel & Demis Hassabis
Ideas
Simple Predicting Machine

question → think → answer
Simple Predicting Machine

input → process (calculate) → output
Kilometres to Miles

- Try a model - this one is linear

```
kilometres
100
```

```
miles = 0.5 
kilometres
```

```
miles
50
```
Kilometres to Miles

kilometres
100

miles = kilometres \times 0.5

calculated miles
50

correct miles
62.137

error
12.137

not great
Kilometres to Miles

kilometres
100

miles = kilometres x 0.6

calculated miles
60

correct miles
62.137

error
2.137

better
Kilometres to Miles

kilometres
100

miles = kilometres x 0.7

calculated miles
70

correct miles
62.137

error
-7.863

worse!
Kilometres to Miles

kilometres 100

miles = kilometres x 0.61

calculated miles 61

correct miles 62.137

error 1.137

best yet!
Key Points

1. Don’t know how something works exactly? Try a model with adjustable parameters.

2. Use the error to refine the parameters.
Garden Bugs

Widths and Lengths of Garden Bugs

- Caterpillars
- Ladybirds
Classifying Bugs

Widths and Lengths of Garden Bugs

length

width

separating line
Classifying Bugs

Widths and Lengths of Garden Bugs

Separating line
Classifying Bugs

Widths and Lengths of Garden Bugs

- Red circles: Group 1
- Green circles: Group 2

Separating line: 

Axes:
- Length
- Width
Classifying Bugs

Classifying an Unknown Bug

unknown bug
Key Points

1. **Classifying** things is kinda like **predicting** things.
<table>
<thead>
<tr>
<th>Example</th>
<th>Width</th>
<th>Length</th>
<th>Bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.0</td>
<td>1.0</td>
<td>ladybird</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>3.0</td>
<td>caterpillar</td>
</tr>
</tbody>
</table>
Training Data for Classifying Bugs

caterpillar

ladybird
Learning from Data

Training Data for Classifying Bugs

$y = (0.25)x$

not a good separator
Learning from Data

Shift the line up just above the training data point.

Training Data for Classifying Bugs

\[ y = (0.25)x \]
Learning from Data

- **Final Refinement**
  \[ y = (2.90) \times \]

- **Refined**
  \[ y = (0.3667) \times \]

- **Initial**
  \[ y = (0.25) \times \]
How Do We Update The Parameter?

\[ \text{error} = \text{target} - \text{actual} \]

\[ E = (A + \Delta A)x - Ax \]

\[ \Delta A = \frac{E}{x} \]
Hang On!

Oh no! Each update ignores previous examples.

- **Final refinement**: $y = (2.90)x$
- **Refined**: $y = (0.3667)x$
- **Initial**: $y = (0.25)x$
Calm Down the Learning

\[ \Delta A = L \cdot \left( \frac{E}{x} \right) \]

Diagram:
- \( y = Ax \)
- \( t = (A + \Delta A)x \)
- Learning rate
Calm Down the Learning

Learning rate $= 0.5$

- Second moderated refinement
  $y = (1.6042) \times$

- First moderated refinement
  $y = (0.3083) \times$

- Initial
  $y = (0.25) \times$
1. **Moderating** your learning is good - ensures you learn from all your data, and reduces impact of outliers or noisy training data.
### Boolean Logic

**IF** I have eaten my vegetables **AND** I am still hungry **THEN** I can have ice cream.

**IF** it’s the weekend **OR** I am on annual leave **THEN** I’ll go to the park.

<table>
<thead>
<tr>
<th>Input A</th>
<th>Input B</th>
<th>AND</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Boolean Logic

input A

logical function

output

input B
Boolean Logic

**logical AND**

- (0,0)
- (1,0)
- (0,1)
- (1,1)

**dividing line**

**logical OR**

- (0,0)
- (1,0)
- (0,1)
- (1,1)

**dividing line**
XOR Puzzle!

<table>
<thead>
<tr>
<th>Input A</th>
<th>Input B</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
XOR Solution!

... use more than one node!
1. Some problems can't be solved with just a single simple linear classifier.

2. You can use **multiple nodes** working together to solve many of these problems.
Brains in Nature
Brains in Nature

brain 0.4 grams

11,000 neurons

nature's brains can eat, fly, navigate, fight,
communicate, play, learn ...

.. and they're resilient

302 neurons

37 billion neurons

(humans 20 billion)

https://faculty.washington.edu/chudler/facts.html
Brains in Nature

- **no output**

- **output on**

- **dial**

- **threshold**
Brains in Nature

logistic function

\[ y = \frac{1}{1 + e^{-x}} \]
Brains in Nature

neurons

signals
Artificial Neuron

input a

input b

input c

Sum inputs
\[ x = a + b + c \]

sigmoid threshold function
\[ y(x) \]

output y
Artificial Neural Network .. finally!
Pause.
Where Does The Learning Happen?

- link weight?
- sigmoid function slope?
1. Natural brains can do sophisticated things, and are incredibly resilient to damage and imperfect signals .. unlike traditional computing.


3. Link weights are the adjustable parameter - it’s where the learning happens.
Feeding Signals Forward
Feeding Signals Forward

Sum inputs:

\[ x = a \cdot w_a + b \cdot w_b + c \cdot w_c \]

Sigmoid threshold function:

\[ y(x) \]

Output:

\[ y \]
Feeding Signals Forward

Inputs: 1.0, 0.5

Layer 1:
- Node 1 with weights:
  - $w_{1,1} = 0.9$
  - $w_{1,2} = 0.2$
  - $w_{2,1} = 0.3$

Layer 2:
- Node 1 with weights:
  - $w_{1,1} = 0.8$

Outputs:
- Node 1 with a value of 0.7408
- Node 2 with a value of 0.6457
Matrix Multiplication

\[ x = (\text{input}_1 \times w_{1,1}) + (\text{input}_2 \times w_{2,1}) \]

\[ x = (\text{input}_1 \times w_{1,2}) + (\text{input}_2 \times w_{2,2}) \]
Matrix Multiplication

weights

\[
\begin{pmatrix}
w_{1,1} & w_{2,1} \\
w_{1,2} & w_{2,2}
\end{pmatrix}
\]

incoming signals

\[
\begin{pmatrix}
input_1 \\
input_2
\end{pmatrix}
\]

\[
\begin{pmatrix}
(input_1 \times w_{1,1}) + (input_2 \times w_{2,1}) \\
(input_1 \times w_{1,2}) + (input_2 \times w_{2,2})
\end{pmatrix}
\]

\[
\omega \cdot 1 = X
\]

dot product
1. The many feedforward calculations can be expressed *concisely* as *matrix multiplication*, no matter what shape the network.

2. Some programming languages can do matrix multiplication really *efficiently* and *quickly*. 


Network Error

\[ w_{1,1} = 3.0 \]

\[ w_{2,1} = 1.0 \]
Matrices Again!

\[
\text{error}_{\text{hidden}} = \begin{pmatrix}
w_{11} & w_{12} \\
w_{21} & w_{22}
\end{pmatrix} \cdot \begin{pmatrix}
e_1 \\
e_2
\end{pmatrix}
\]

\[
\text{error}_{\text{hidden}} = w^T_{\text{hidden-output}} \cdot \text{error}_{\text{output}}
\]
1. Remember we use the **error** to guide how we refine a model’s parameter - link weights.

2. The error at the output nodes is easy - the difference between the **desired** and **actual** outputs.

3. The error at internal nodes isn’t obvious. A **heuristic** approach is to split it in **proportion** to the link weights.

4. … and back propagating the error can be expressed as a **matrix** multiplication too!
Yes, But How Do We Actually Update The Weights?

\[ o_k = \frac{1}{1 + e^{-\sum_{j=1}^{3} (w_{j,k} \cdot \frac{1}{1 + e^{-\sum_{i=1}^{3} (w_{i,j} \cdot x_i)}})}} \]

Aaarrggghhh !!
Perfect is the Enemy of Good

landscape is a complicated difficult mathematical function ..
... with all kinds of lumps, bumps, kinks ...
Gradient Descent

smaller gradient .. you're closer to the bottom ... take smaller steps?
1. **Gradient descent** is a practical way of finding the minimum of **difficult** functions.

2. You can avoid the chance of **overshooting** by taking smaller steps if the gradient gets shallower.

3. The error of a neural network is a **difficult** function of the link weights … so maybe gradient descent will help …
Climbing Down the Network Error Landscape

we need to find this gradient
$E = (\text{desired} - \text{actual})^2$

$$\frac{dE}{dw_j} = -e_j \cdot o_j \cdot (1 - o_j) \cdot o_i$$

A gentle intro to calculus

http://makeyourownneuralnetwork.blogspot.co.uk/2016/01/a-gentle-introduction-to-calculus.html
Updating the Weights

move $w_{jk}$ in the opposite direction to the slope

$$\text{new } w_{jk} = \text{old } w_{jk} - \alpha \cdot \frac{\partial E}{\partial w_{jk}}$$

remember that learning rate
Neural Network Class

- Initialise: set size, initial weights
- Train: do the learning
- Query: query for answers
Python has Cool Tools

- numpy
- scipy
- matplotlib
- notebook

matrix maths
# initialise the neural network

```python
def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
    # set number of nodes in each input, hidden, output layer
    self.inodes = inputnodes
    self.hnodes = hiddennodes
    self.onodes = outputnodes

    # link weight matrices, wih and who
    # weights inside the arrays are w_i__j, where link is from node i to node j in the next layer
    # w11 w21
    # w12 w22 etc
    self.wih = numpy.random.normal(0.0, pow(self.hnodes, -0.5), (self.hnodes, self.inodes))
    self.who = numpy.random.normal(0.0, pow(self.onodes, -0.5), (self.onodes, self.hnodes))

    # learning rate
    self.lr = learningrate

    # activation function is the sigmoid function
    self.activation_function = lambda x: scipy.special.expit(x)

    pass
```

random initial weights `numpy.random.normal()`
# query the neural network

def query(self, inputs_list):
    # convert inputs list to 2d array
    inputs = numpy.array(inputs_list, ndmin=2).T

    # calculate signals into hidden layer
    hidden_inputs = numpy.dot(self.wih, inputs)
    # calculate the signals emerging from hidden layer
    hidden_outputs = self.activation_function(hidden_inputs)

    # calculate signals into final output layer
    final_inputs = numpy.dot(self.who, hidden_outputs)
    # calculate the signals emerging from final output layer
    final_outputs = self.activation_function(final_inputs)

    return final_outputs
# train the neural network
def train(self, inputs_list, targets_list):
    # convert inputs list to 2d array
    inputs = numpy.array(inputs_list, ndmin=2).T
    targets = numpy.array(targets_list, ndmin=2).T

    # calculate signals into hidden layer
    hidden_inputs = numpy.dot(self.wih, inputs)
    # calculate the signals emerging from hidden layer
    hidden_outputs = self.activation_function(hidden_inputs)

    # calculate signals into final output layer
    final_inputs = numpy.dot(self.who, hidden_outputs)
    # calculate the signals emerging from final output layer
    final_outputs = self.activation_function(final_inputs)

    # output layer error is the (target - actual)
    output_errors = targets - final_outputs
    # hidden layer error is the output_errors, split by weights, recombined at hidden nodes
    hidden_errors = numpy.dot(self.who.T, output_errors)

    # update the weights for the links between the hidden and output layers
    self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 - final_outputs)), numpy.transpose(hidden_outputs))

    # update the weights for the links between the input and hidden layers
    self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)), numpy.transpose(inputs))

    pass
Handwriting
Handwritten Numbers Challenge
MNIST dataset:

60,000 training data examples

10,000 test data examples
MNIST Datasets

In [9]:
```
data_file = open("mnist_dataset/mnist_train_100.csv", 'r')
data_list = data_file.readlines()
data_file.close()
```

Out[9]:
```
100
```

In [10]:
```
data_list[0]
```

Out[10]:
```
'5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

In [32]:
```
all_values = data_list[0].split(',')
image_array = numpy.asarray(all_values[1:]).reshape((28,28))
matplotlib.pyplot.imshow(image_array, cmap='Greys', interpolation='None')
```

Out[32]:
```
<matplotlib.image.AxesImage at 0x108818cc0>
```

Label

784 Pixels Values

28 by 28 pixel image
<table>
<thead>
<tr>
<th>output layer</th>
<th>label</th>
<th>example “5”</th>
<th>example “0”</th>
<th>example “9”</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.95</td>
<td>0.02</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.00</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0.01</td>
<td>0.02</td>
<td>0.40</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0.99</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>0.02</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>0.01</td>
<td>0.02</td>
<td>0.86</td>
</tr>
</tbody>
</table>
Experiments

96% is very good!
we've only used simple ideas and code

Performance and Hidden Nodes
MNIST dataset with 3-layer neural network

Performance and Learning Rate
MNIST dataset with 3-layer neural network

Performance and Epoch
MNIST dataset with 3-layer neural network

random processes do go wonky!
More Experiments

98% is amazing!
Thoughts
Peek Inside The Mind Of a Neural Network?

**normal forward query**

- Image → Neural Network → Label

**reverse back query**

- Image ← Neural Network ← Label
Peek Inside The Mind Of a Neural Network?

this isn't done very often
Thanks!

live demo!
Finding Out More

makeyourownneuralnetwork.blogspot.co.uk

github.com/makeyourownneuralnetwork

www.amazon.co.uk/dp/B01EER4Z4G

twitter.com/myoneuralnet

slides goo.gl/JKsb62
Raspberry Pi Zero

It all works on a Raspberry Pi Zero ... and it only costs £4 / $5 !!