What is Deep Learning?
Data and Machine Learning

Graph showing the relationship between performance and amount of data, with two lines: one for New AI methods (deep learning) and another for Most learning algorithms.
The idea

- Build learning algorithms that mimic the brain!
Neurons in the brain
Neural Networks
(Deep Learning)
Deep Learning Trends

Now

0-2 years
Tagged data

3-5 years
Tagged & untagged data
Learning from Tagged Data
Learning from Tagged Data

In 1917, Einstein applied the general theory of relativity to model the large-scale structure of the universe. He was visiting the United States when Adolf Hitler came to power in 1933 and did not go back to Germany, where he had been a professor at the Berlin Academy of Sciences. He settled in the U.S., becoming an American citizen in 1940. On the eve of World War II, he endorsed a letter to President Franklin D. Roosevelt alerting him to the potential development of "extremely powerful bombs of a new type" and recommending that the U.S. begin similar research. This eventually led to what would become the Manhattan Project. Einstein supported defending the Allied forces, but largely denounced using the new discovery of nuclear fission as a weapon. Later, with the British philosopher Bertrand Russell, Einstein signed the Russell–Einstein Manifesto, which highlighted the danger of nuclear weapons. Einstein was affiliated with the Institute for Advanced Study in Princeton, New Jersey, until his death in 1955.

Tag colours:
- LOCATION
- TIME
- PERSON
- ORGANIZATION
- MONEY
- PERCENT
- DATE
Deep Learning Applications

Speech recognition

Image Search

Ads; Web search
Colorization of Black and White Images

Colorization of Black and White Photographs
Image taken from Richard Zhang, Phillip Isola and Alexei A. Efros.
Automatic Machine Translation

Instant Visual Translation
Example of instant visual translation, taken from the Google Blog.
Deep Learning for Recommender Systems

Image courtesy of Netflix
Recommender System
The model is capable of learning how to spell, punctuate, form sentences and even capture the style of the text in the corpus.
AI as a computer systems problem

10 million connections

↓

1 billion connections
16,000 CPUs is expensive
GPUs (Graphics Processor Unit)
Building Huge Neural Networks

- 10 million connections
- 1 billion connections
- 10 billion connections
What is the idea?

- Biologically inspired model
- Huge amount of training samples
- General and suitable for any input
- Supervised, unsupervised
What has been achieved?

loosely biologically inspired models

huge amount of training samples when available

general and suitable for many kinds of inputs after adaptation

supervised learning in a product

unsupervised and reinforcement learning – work in progress
A bit of history

early 1960s
Alexey Ivakhnenko
first works on deep neural networks

1986
Geoffrey Hinton
backpropagation algorithm in its current form
A bit of history

Computer vision and speech communities do not use neural nets anymore.

- Hard to train
- No computational resources
- Small datasets

- Efficient training
- GPU / multi-core processors
- Large datasets

2006
What is a Neural Network like?
Biological Inspiration

Idea: To make the computer more robust, intelligent, and learn, ... Let’s model our computer software (and/or hardware) after the brain.
Neurons in the Brain

• Although heterogeneous, at a low level the brain is composed of neurons
  – A neuron receives input from other neurons (generally thousands) from its synapses
  – Inputs are approximately summed
  – When the input exceeds a threshold the neuron sends an electrical spike that travels down the axon, to the next neuron(s)
Comparison of Brains and Traditional Computers

<table>
<thead>
<tr>
<th></th>
<th>Personal Computer</th>
<th>Human Brain</th>
</tr>
</thead>
<tbody>
<tr>
<td>processing units</td>
<td>1 CPU, 2–10 cores&lt;br&gt;(10^{10}) transistors&lt;br&gt;1–2 graphics cards/GPUs,&lt;br&gt;(10^3) cores/shaders&lt;br&gt;(10^{10}) transistors</td>
<td>(10^{11}) neurons</td>
</tr>
<tr>
<td>storage capacity</td>
<td>(10^{10}) bytes main memory (RAM)&lt;br&gt;(10^{12}) bytes external memory</td>
<td>(10^{11}) neurons&lt;br&gt;(10^{14}) synapses</td>
</tr>
<tr>
<td>processing speed</td>
<td>(10^{-9}) seconds&lt;br&gt;(10^9) operations per second</td>
<td>(&gt;10^{-3}) seconds&lt;br&gt;&lt; 1000 per second</td>
</tr>
<tr>
<td>bandwidth</td>
<td>(10^{12}) bits/second</td>
<td>(10^{14}) bits/second</td>
</tr>
<tr>
<td>neural updates</td>
<td>(10^6) per second</td>
<td>(10^{14}) per second</td>
</tr>
</tbody>
</table>
A Biological Neuron

- The human brain is made up of about 100 billion neurons

- Neurons receive electric signals at the dendrites and send them to the axon
The Brain vs Artificial Neural Networks

- Similarities
  - Neurons, connections between neurons
  - Learning = change of connections, not change of neurons
  - Massive parallel processing

- But artificial neural networks are much simpler
  - computation within neuron vastly simplified
  - discrete time steps
  - typically some form of supervised learning with massive number of stimuli
An Artificial Neuron

A **Threshold Logic Unit (TLU)** is a processing unit for numbers with \( n \) inputs \( x_1, \ldots, x_n \) and one output \( y \). The unit has a **threshold** \( \theta \) and each input \( x_i \) is associated with a **weight** \( w_i \). A threshold logic unit computes the function

\[
y = \begin{cases} 
1, & \text{if } \sum_{i=1}^{n} w_i x_i \geq \theta, \\
0, & \text{otherwise.}
\end{cases}
\]

TLUs mimic the thresholding behavior of biological neurons in a (very) simple fashion.
The Neuron Computation

Threshold logic unit for the conjunction $x_1 \land x_2$.

\[ x_1 \quad 3 \quad \rightarrow \quad 4 \quad \rightarrow \quad y \]

\[ x_2 \quad 2 \quad \rightarrow \quad y \]

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$3x_1 + 2x_2$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Threshold logic unit for the implication $x_2 \rightarrow x_1$.

\[ x_1 \quad 2 \quad \rightarrow \quad 1 \quad \rightarrow \quad y \]

\[ x_2 \quad -2 \quad \rightarrow \quad y \]

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$2x_1 - 2x_2$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Geometric Interpretation

Threshold logic unit for $x_1 \land x_2$.

Threshold logic unit for $x_2 \rightarrow x_1$. 
Geometric Interpretation

Visualization of 3-dimensional Boolean functions:

Threshold logic unit for \((x_1 \land \overline{x_2}) \lor (x_1 \land x_3) \lor (\overline{x_2} \land x_3)\).
Networks of Neurons

Idea: logical decomposition

\[ x_1 \leftrightarrow x_2 \equiv (x_1 \rightarrow x_2) \land (x_2 \rightarrow x_1) \]

-1

computes \( y_1 = x_1 \rightarrow x_2 \)

\[ x_1 \rightarrow -2 \]

\[ x_2 \rightarrow -2 \]

\[ x_1 \rightarrow 2 \]

\[ x_2 \rightarrow 2 \]

\[ 2 \]

\[ 2 \]

3

computes \( y = y_1 \land y_2 \)

\[ y = x_1 \leftrightarrow x_2 \]

computes \( y_2 = x_2 \rightarrow x_1 \)
Linear Models

- We used before weighted linear combination of feature values $h_j$ and weights $\lambda_j$

$$\text{score}(\lambda, d_i) = \sum_{j} \lambda_j h_j(d_i)$$

- Such models can be illustrated as a "network"
XOR

- Linear models cannot model XOR
Multiple Layers

- Add an intermediate ("hidden") layer of processing (each arrow is a weight)

- Have we gained anything so far?
Non-Linearity

- Instead of computing a linear combination
  \[ \text{score}(\lambda, d_i) = \sum_{j} \lambda_j \ h_j(d_i) \]

- Add a non-linear function
  \[ \text{score}(\lambda, d_i) = f\left(\sum_{j} \lambda_j \ h_j(d_i)\right) \]

- Popular choices
  - \( \tanh(x) \)
  - \( \text{sigmoid}(x) = \frac{1}{1+e^{-x}} \)
  - \( \text{relu}(x) = \max(0, x) \)

(sigmoid is also called the "logistic function")
More Layers = Deep Learning

General structure of a multi-layer perceptron

- $x_1, x_2, \ldots, x_n$ are input nodes.
- $U_{in}, U_{hidden}^{(1)}, U_{hidden}^{(2)}, \ldots, U_{hidden}^{(r-2)}, U_{out}$ are layers in the network.
- $y_1, y_2, \ldots, y_m$ are output nodes.
Example

- Try out two input values

- Hidden unit computation

\[
\text{sigmoid}(1.0 \times 3.7 + 0.0 \times 3.7 + 1 \times -1.5) = \text{sigmoid}(2.2) = \frac{1}{1 + e^{-2.2}} = 0.90
\]

\[
\text{sigmoid}(1.0 \times 2.9 + 0.0 \times 2.9 + 1 \times -4.5) = \text{sigmoid}(-1.6) = \frac{1}{1 + e^{1.6}} = 0.17
\]
Example

- Try out two input values
- Hidden unit computation

\[
sigmoid(1.0 \times 3.7 + 0.0 \times 3.7 + 1 \times -1.5) = sigmoid(2.2) = \frac{1}{1 + e^{-2.2}} = 0.90
\]

\[
sigmoid(1.0 \times 2.9 + 0.0 \times 2.9 + 1 \times -4.5) = sigmoid(-1.6) = \frac{1}{1 + e^{1.6}} = 0.17
\]
Example

- Output unit computation

\[
sigmoid(.90 \times 4.5 + .17 \times -5.2 + 1 \times -2.0) = sigmoid(1.17) = \frac{1}{1 + e^{-1.17}} = 0.76
\]
Example

Output unit computation

\[
sigmoid(0.90 \times 4.5 + 0.17 \times -5.2 + 1 \times -2.0) = sigmoid(1.17) = \frac{1}{1 + e^{-1.17}} = 0.76
\]
Backpropagation in Training
Error

- Computed output: $y = 0.76$
- Correct output: $t = 1.0$

⇒ How do we adjust the weights?
Key Concepts

• Gradient descent
  – error is a function of the weights
  – we want to reduce the error
  – gradient descent: move towards the error minimum
  – compute gradient $\rightarrow$ get direction to the error minimum
  – adjust weights towards direction of lower error

• Back-propagation
  – first adjust last set of weights
  – propagate error back to each previous layer
  – adjust their weights
Gradient Descent
Derivative of Sigmoid

- Sigmoid
  \[ \text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \]

- Reminder: quotient rule
  \[ \left( \frac{f(x)}{g(x)} \right)' = \frac{g(x)f'(x) - f(x)g'(x)}{g(x)^2} \]

- Derivative
  \[ \frac{d}{dx} \text{sigmoid}(x) = \frac{d}{dx} \frac{1}{1 + e^{-x}} = \frac{0 \times (1 - e^{-x}) - (-e^{-x})}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}} \left( \frac{e^{-x}}{1 + e^{-x}} \right) = \frac{1}{1 + e^{-x}} \left( 1 - \frac{1}{1 + e^{-x}} \right) = \text{sigmoid}(x)(1 - \text{sigmoid}(x)) \]
Final Layer Update

- Linear combination of weights $s = \sum_k w_k h_k$
- Activation function $y = \text{sigmoid}(s)$
- Error (L2 norm) $E = \frac{1}{2}(t - y)^2$
- Derivative of error with regard to one weight $w_k$

$$\frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k}$$
Final Layer Update (1)

- Linear combination of weights $s = \sum_k w_k h_k$
- Activation function $y = \text{sigmoid}(s)$
- Error (L2 norm) $E = \frac{1}{2}(t - y)^2$
- Derivative of error with regard to one weight $w_k$
  \[
  \frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k}
  \]
- Error $E$ is defined with respect to $y$
  \[
  \frac{dE}{dy} = \frac{d}{dy} \frac{1}{2}(t - y)^2 = -(t - y)
  \]
Final Layer Update (2)

- Linear combination of weights $s = \sum_k w_k h_k$
- Activation function $y = \text{sigmoid}(s)$
- Error (L2 norm) $E = \frac{1}{2}(t - y)^2$
- Derivative of error with regard to one weight $w_k$
  \[
  \frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k}
  \]
- $y$ with respect to $x$ is $\text{sigmoid}(s)$
  \[
  \frac{dy}{ds} = \frac{d \text{sigmoid}(s)}{ds} = \text{sigmoid}(s)(1 - \text{sigmoid}(s)) = y(1 - y)
  \]
Final Layer Update (3)

- Linear combination of weights \( s = \sum_k w_k h_k \)
- Activation function \( y = \text{sigmoid}(s) \)
- Error (L2 norm) \( E = \frac{1}{2}(t - y)^2 \)
- Derivative of error with regard to one weight \( w_k \)
  \[
  \frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k}
  \]

- \( x \) is weighted linear combination of hidden node values \( h_k \)
  \[
  \frac{ds}{dw_k} = \frac{d}{dw_k} \sum_k w_k h_k = h_k
  \]
Putting it All Together

- Derivative of error with regard to one weight $w_k$
  \[
  \frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k} = -(t - y) \ y(1 - y) \ h_k
  \]

- error
- derivative of sigmoid: $y'$

- Weight adjustment will be scaled by a fixed learning rate $\mu$
  \[
  \Delta w_k = \mu \ (t - y) \ y' \ h_k
  \]
Multiple Output Nodes

- Our example only had one output node
- Typically neural networks have multiple output nodes
- Error is computed over all $j$ output nodes

$$E = \sum_j \frac{1}{2} (t_j - y_j)^2$$

- Weights $k \rightarrow j$ are adjusted according to the node they point to

$$\Delta w_{j \leftarrow k} = \mu (t_j - y_j) y'_j h_k$$
Hidden Layer Update

• In a hidden layer, we do not have a target output value

• But we can compute how much each node contributed to downstream error

• Definition of error term of each node

\[ \delta_j = (t_j - y_j) y'_j \]

• Back-propagate the error term

(why this way? there is math to back it up...)

\[ \delta_i = \left( \sum_j w_{j \leftarrow i} \delta_j \right) y'_i \]

• Universal update formula

\[ \Delta w_{j \leftarrow k} = \mu \delta_j h_k \]
Our Example

- Computed output: $y = .76$
- Correct output: $t = 1.0$
- Final layer weight updates (learning rate $\mu = 10$)
  - $\delta_G = (t - y) y' = (1 - .76) \times .181 = .0434$
  - $\Delta w_{GD} = \mu \delta_G h_D = 10 \times .0434 \times .90 = .391$
  - $\Delta w_{GE} = \mu \delta_G h_E = 10 \times .0434 \times .17 = .074$
  - $\Delta w_{GF} = \mu \delta_G h_F = 10 \times .0434 \times 1 = .434$
Our Example

- Computed output: $y = .76$
- Correct output: $t = 1.0$
- Final layer weight updates (learning rate $\mu = 10$)
  - $\delta_G = (t - y) y' = (1 - .76) 0.181 = .0434$
  - $\Delta w_{GD} = \mu \delta_G h_D = 10 \times .0434 \times .90 = .391$
  - $\Delta w_{GE} = \mu \delta_G h_E = 10 \times .0434 \times .17 = .074$
  - $\Delta w_{GF} = \mu \delta_G h_F = 10 \times .0434 \times 1 = .434$
Hidden Layer Updates

- **Hidden node D**
  - $\delta_D = \left( \sum_j w_{j\rightarrow i} \delta_j \right) y_D' = w_{GD} \delta_G y_D' = 4.5 \times 0.0434 \times 0.0898 = 0.0175$
  - $\Delta w_{DA} = \mu \delta_D h_A = 10 \times 0.0175 \times 1.0 = 0.175$
  - $\Delta w_{DB} = \mu \delta_D h_B = 10 \times 0.0175 \times 0.0 = 0$
  - $\Delta w_{DC} = \mu \delta_D h_C = 10 \times 0.0175 \times 1 = 0.175$

- **Hidden node E**
  - $\delta_E = \left( \sum_j w_{j\rightarrow i} \delta_j \right) y_E' = w_{GE} \delta_G y_E' = -5.2 \times 0.0434 \times 0.2055 = -0.0464$
  - $\Delta w_{EA} = \mu \delta_E h_A = 10 \times -0.0464 \times 1.0 = -0.464$
  - etc.
Additional Aspects
GPU

- Neural network layers may have, say, 200 nodes

- Computations such as $\vec{W}\hat{h}$ require $200 \times 200 = 40,000$ multiplications

- Graphics Processing Units (GPU) are designed for such computations
  - image rendering requires such vector and matrix operations
  - massively multi-core but lean processing units
  - example: NVIDIA Tesla K20c GPU provides 2496 thread processors

- Extensions to C to support programming of GPUs, such as CUDA
Toolkits

- Theano
- Tensorflow (Google)
- PyTorch (Facebook)
- MXNet (Amazon)
- DyNet
References

- Introduction to Neural Networks, Philipp Koehn, 3 October 2017
- Artificial Neural Networks and Deep Learning, Christian Borgelt, University of Konstanz, Germany