Neural Networks 2

Shivaram Kalyanakrishnan

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

February 2023

This Lecture

- Backpropagation: forward and backward passes
- Practical issues with neural networks
- Overview of models and application domains

This Lecture

- Backpropagation: forward and backward passes
- Practical issues with neural networks
- Overview of models and application domains

Backpropagation Algorithm (Rumelhart *et al.*, 1986)

We are given a training data set $\{(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)\}$.
Let us start with some initial weights $\mathbf{w}$.

For each data point r,

the true label is y^r,

the prediction is $p^r(\mathbf{w})$; and

thus, the error is $(y^r - p^r(\mathbf{w}))^2$.

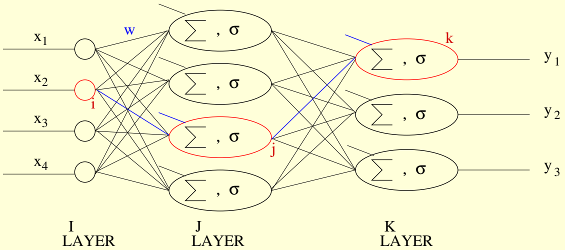
The aggregate error $E(\mathbf{w})$ is $\sum_{r=1}^{n}(y^r - p^r(\mathbf{w}))^2$.
We move a step in the direction minimising error:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} E(\mathbf{w}),$$

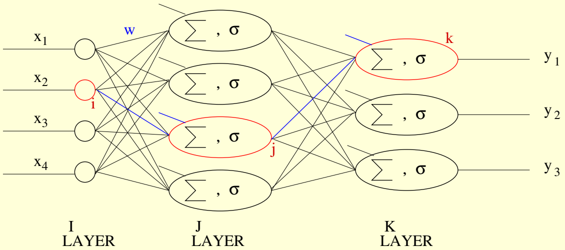
and iterate until convergence.

For a given neural network, $\nabla_{\mathbf{w}} E(\mathbf{w})$ can be easily computed (coming up).

Backprop: Forward Pass (Function Calculation)

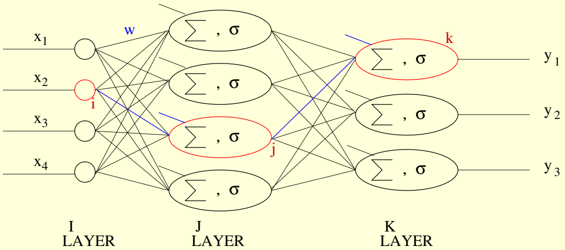


Backprop: Forward Pass (Function Calculation)



$$out_I(i) = x_i.$$

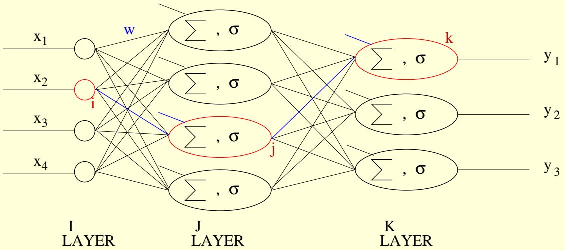
Backprop: Forward Pass (Function Calculation)



$$out_I(i) = x_i.$$

$$in_J(j) = \sum_{i \in I} w_{ij} out_I(i) + b_j.$$

Backprop: Forward Pass (Function Calculation)

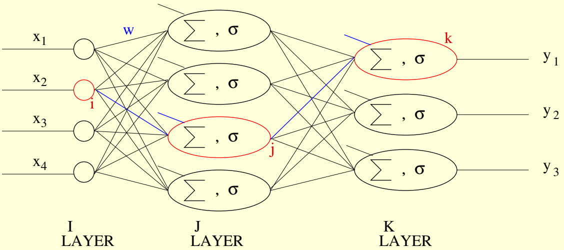


$$out_I(i) = x_i.$$

$$in_J(j) = \sum_{i \in I} w_{ij} out_I(i) + b_j.$$

$$out_J(j) = \sigma(in_J(j)).$$

Backprop: Forward Pass (Function Calculation)



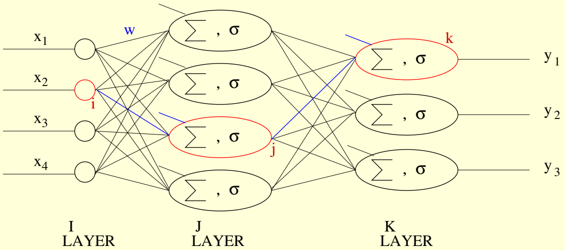
$$out_I(i) = x_i.$$

$$in_J(j) = \sum_{i \in I} w_{ij} out_I(i) + b_j.$$

$$out_J(j) = \sigma(in_J(j)).$$

$$in_K(k) = \sum_{j \in J} w_{jk} out_J(j) + b_k.$$

Backprop: Forward Pass (Function Calculation)



$$out_I(i) = x_i.$$

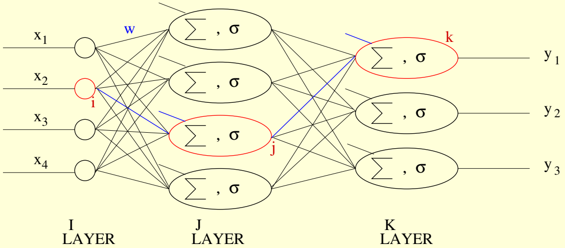
$$in_J(j) = \sum_{i \in I} w_{ij} out_I(i) + b_j.$$

$$out_J(j) = \sigma(in_J(j)).$$

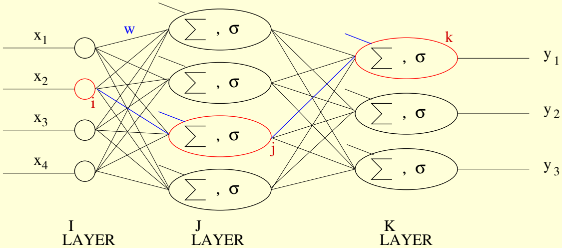
$$in_K(k) = \sum_{j \in J} w_{jk} out_J(j) + b_k.$$

$$out_K(k) = \sigma(in_K(k)).$$

Backprop: Backward Pass (Gradient Descent to Adjust Weights)



Backprop: Backward Pass (Gradient Descent to Adjust Weights)

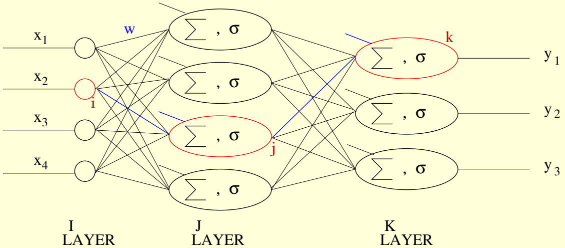


For data point (x^r, y^r), Δ^r for a node denotes its contribution to the error:

$$\Delta_k^r = (y^r - out_K(k)|_{x=x^r})\sigma'(in_K(k)|_{x=x^r}).$$

$$\Delta_j^r = \left(\sum_{k=1}^{K} w_{jk}\Delta_k^r\right)\sigma'(in_J(j)|_{x=x^r}).$$

Backprop: Backward Pass (Gradient Descent to Adjust Weights)



For data point (x^r, y^r), Δ^r for a node denotes its contribution to the error:

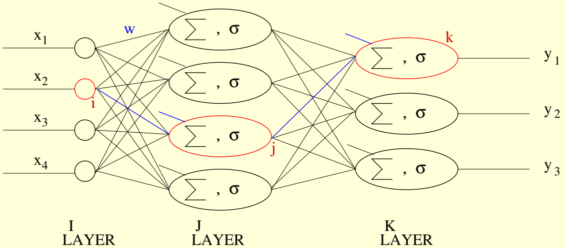
$$\Delta_k^r = (y^r - out_K(k)|_{x=x^r})\sigma'(in_K(k)|_{x=x^r}).$$

$$\Delta_j^r = \left(\sum_{k=1}^{K} w_{jk}\Delta_k^r\right)\sigma'(in_J(j)|_{x=x^r}).$$

Weights are updated based on Δ's of the target node:

$$w_{ij} \leftarrow w_{ij} + \alpha \sum_{r=1}^{n} out_I(i)|_{x=x^r}\Delta_j^r; \quad w_{jk} \leftarrow w_{jk} + \alpha \sum_{r=1}^{n} out_J(j)|_{x=x^r}\Delta_k^r.$$

Backprop: Backward Pass (Gradient Descent to Adjust Weights)



For data point (x^r, y^r), Δ^r for a node denotes its contribution to the error:

$$\Delta_k^r = (y^r - out_K(k)|_{x=x^r})\sigma'(in_K(k)|_{x=x^r}).$$

$$\Delta_j^r = \left(\sum_{k=1}^{K} w_{jk}\Delta_k^r\right)\sigma'(in_J(j)|_{x=x^r}).$$

Weights are updated based on Δ's of the target node:

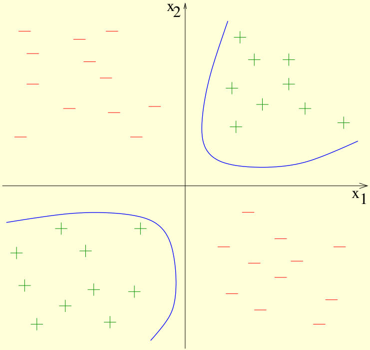
$$w_{ij} \leftarrow w_{ij} + \alpha \sum_{r=1}^{n} out_I(i)|_{x=x^r}\Delta_j^r; \quad w_{jk} \leftarrow w_{jk} + \alpha \sum_{r=1}^{n} out_J(j)|_{x=x^r}\Delta_k^r.$$

Why is this method called "Backprop"?

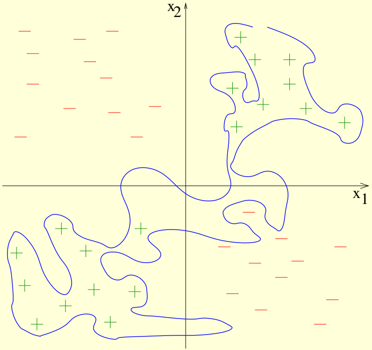
This Lecture

- Backpropagation: forward and backward passes
- Practical issues with neural networks
- Overview of models and application domains

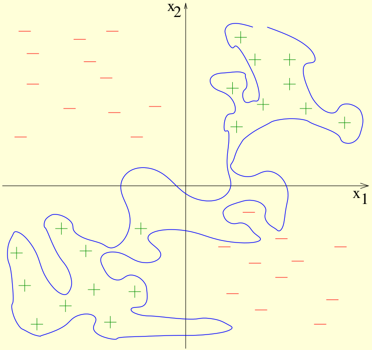
Overfitting

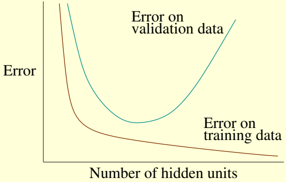


Overfitting

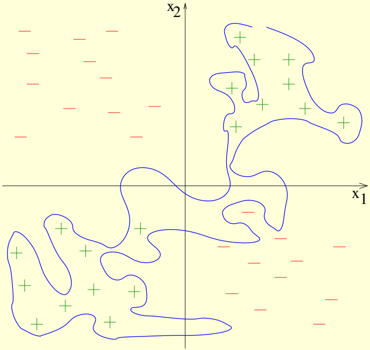


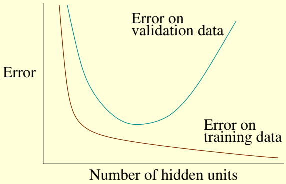
Overfitting





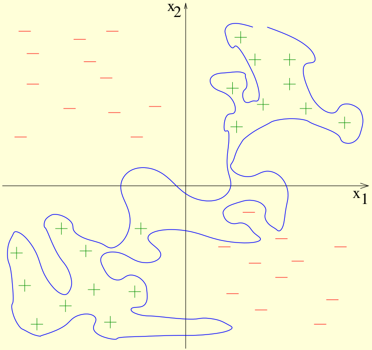
Overfitting

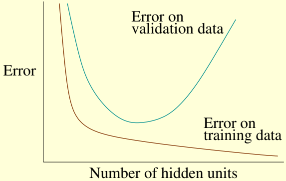




How to tune the number of hidden units?

Overfitting





How to tune the number of hidden units? Cross-validation!

Topology and Initialisation

- For a given task, how many layers needed; how many nodes per layer?

Topology and Initialisation

- For a given task, how many layers needed; how many nodes per layer?

  No conclusive answer. Good idea to start with 1–2 layers and increment (similar with nodes per layer).

Topology and Initialisation

- For a given task, how many layers needed; how many nodes per layer?

  No conclusive answer. Good idea to start with 1–2 layers and increment (similar with nodes per layer).

  To prevent overfitting, number of weights must grow proportionally with size of training data. Today's data sets often allow for tens of layers (deep learning).

Topology and Initialisation

- For a given task, how many layers needed; how many nodes per layer?

  No conclusive answer. Good idea to start with 1–2 layers and increment (similar with nodes per layer).

  To prevent overfitting, number of weights must grow proportionally with size of training data. Today's data sets often allow for tens of layers (deep learning).

- Which activation function to use?

Topology and Initialisation

- For a given task, how many layers needed; how many nodes per layer?

  No conclusive answer. Good idea to start with 1–2 layers and increment (similar with nodes per layer).

  To prevent overfitting, number of weights must grow proportionally with size of training data. Today's data sets often allow for tens of layers (deep learning).

- Which activation function to use?

  $\sigma()$ and "`tanh()`" quite popular. Also "ReLU" in some applications.

Topology and Initialisation

- For a given task, how many layers needed; how many nodes per layer?

  No conclusive answer. Good idea to start with 1–2 layers and increment (similar with nodes per layer).

  To prevent overfitting, number of weights must grow proportionally with size of training data. Today's data sets often allow for tens of layers (deep learning).

- Which activation function to use?

  $\sigma()$ and "$\mathtt{tanh}()$" quite popular. Also "ReLU" in some applications.

- How to initialise the weights?

Topology and Initialisation

- For a given task, how many layers needed; how many nodes per layer?

  No conclusive answer. Good idea to start with 1–2 layers and increment (similar with nodes per layer).

  To prevent overfitting, number of weights must grow proportionally with size of training data. Today's data sets often allow for tens of layers (deep learning).

- Which activation function to use?

  $\sigma()$ and "$\mathtt{tanh}()$" quite popular. Also "ReLU" in some applications.

- How to initialise the weights?

  Typically drawn at random from $N(0, 1)$. "Unsupervised pre-training" also possible.

Mini-batches; Stochastic Gradient Descent

- Recall that we update weights by gradient descent as follows.

$$w_{ij} \leftarrow w_{ij} + \alpha \sum_{r=1}^{n} out_I(i)|_{x=x^r} \Delta_j^r; \quad w_{jk} \leftarrow w_{jk} + \alpha \sum_{r=1}^{n} out_J(j)|_{x=x^r} \Delta_k^r.$$

Mini-batches; Stochastic Gradient Descent

- Recall that we update weights by gradient descent as follows.

$$w_{ij} \leftarrow w_{ij} + \alpha \sum_{r=1}^{n} out_I(i)|_{x=x^r} \Delta_j^r; \quad w_{jk} \leftarrow w_{jk} + \alpha \sum_{r=1}^{n} out_J(j)|_{x=x^r} \Delta_k^r.$$

  Each update needs a scan through the entire training data set!

Mini-batches; Stochastic Gradient Descent

- Recall that we update weights by gradient descent as follows.

$$w_{ij} \leftarrow w_{ij} + \alpha \sum_{r=1}^{n} out_I(i)|_{x=x^r} \Delta_j^r; \quad w_{jk} \leftarrow w_{jk} + \alpha \sum_{r=1}^{n} out_J(j)|_{x=x^r} \Delta_k^r.$$

  Each update needs a scan through the entire training data set!

- Instead we can randomly partition D into m mini-batches of size b (thus $m = n/b$), say $D_1, D_2, \ldots, D_m$.

Mini-batches; Stochastic Gradient Descent

- Recall that we update weights by gradient descent as follows.

$$w_{ij} \leftarrow w_{ij} + \alpha \sum_{r=1}^{n} out_I(i)|_{x=x^r} \Delta_j^r; \quad w_{jk} \leftarrow w_{jk} + \alpha \sum_{r=1}^{n} out_J(j)|_{x=x^r} \Delta_k^r.$$

  Each update needs a scan through the entire training data set!

- Instead we can randomly partition D into m mini-batches of size b (thus $m = n/b$), say $D_1, D_2, \ldots, D_m$.

  Then we make m updates while scanning through the full data set; each update uses a mini-batch of data.

  For $s = 1, 2, \ldots, m$:

$$w_{ij} \leftarrow w_{ij} + \alpha \sum_{x^r \in D_s} out_I(i)|_{x=x^r} \Delta_j^r; \quad w_{jk} \leftarrow w_{jk} + \alpha \sum_{x^r \in D_s} out_J(j)|_{x=x^r} \Delta_k^r.$$

Mini-batches; Stochastic Gradient Descent

- Recall that we update weights by gradient descent as follows.

$$w_{ij} \leftarrow w_{ij} + \alpha \sum_{r=1}^{n} out_I(i)|_{x=x^r} \Delta_j^r; \quad w_{jk} \leftarrow w_{jk} + \alpha \sum_{r=1}^{n} out_J(j)|_{x=x^r} \Delta_k^r.$$

  Each update needs a scan through the entire training data set!

- Instead we can randomly partition D into m mini-batches of size b (thus $m = n/b$), say $D_1, D_2, \ldots, D_m$.

  Then we make m updates while scanning through the full data set; each update uses a mini-batch of data.

  For $s = 1, 2, \ldots, m$:

$$w_{ij} \leftarrow w_{ij} + \alpha \sum_{x^r \in D_s} out_I(i)|_{x=x^r} \Delta_j^r; \quad w_{jk} \leftarrow w_{jk} + \alpha \sum_{x^r \in D_s} out_J(j)|_{x=x^r} \Delta_k^r.$$

- Batch size b is usually small compared to n.

Mini-batches; Stochastic Gradient Descent

- Recall that we update weights by gradient descent as follows.

$$w_{ij} \leftarrow w_{ij} + \alpha \sum_{r=1}^{n} out_I(i)|_{x=x^r} \Delta_j^r; \quad w_{jk} \leftarrow w_{jk} + \alpha \sum_{r=1}^{n} out_J(j)|_{x=x^r} \Delta_k^r.$$

  Each update needs a scan through the entire training data set!

- Instead we can randomly partition D into m mini-batches of size b (thus $m = n/b$), say $D_1, D_2, \ldots, D_m$.

  Then we make m updates while scanning through the full data set; each update uses a mini-batch of data.

  For $s = 1, 2, \ldots, m$:

$$w_{ij} \leftarrow w_{ij} + \alpha \sum_{x^r \in D_s} out_I(i)|_{x=x^r} \Delta_j^r; \quad w_{jk} \leftarrow w_{jk} + \alpha \sum_{x^r \in D_s} out_J(j)|_{x=x^r} \Delta_k^r.$$

- Batch size b is usually small compared to n.

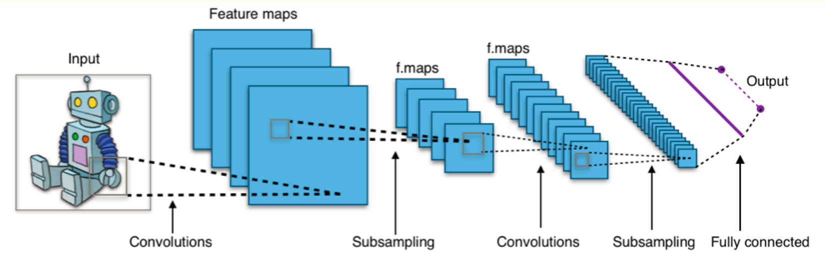
- Using only a random subset of data for each update leads to stochastic gradient descent, which also converges to a local minimum.

This Lecture

- Backpropagation: forward and backward passes
- Practical issues with neural networks
- Overview of models and application domains

Convolutional Neural Networks

- A convolution operator is a small “filter” scanned over the entire image.
- The parameters of this filter are learned, as usual, using backprop.
- Useful for detecting localised patterns, edges, etc.
- Very effective in tasks based on visual perception (images, video).

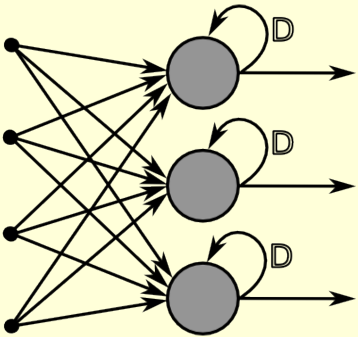


[1]

1. https://upload.wikimedia.org/wikipedia/commons/6/63/Typical_cnn.png. CC image courtesy of Aphex34 on WikiMedia Commons licensed under CC-BY-SA-4.0.

Recurrent Neural Networks

- Feedback networks, persist previous stage(s) of signal.
- Especially effective for speech processing and sequential tasks.

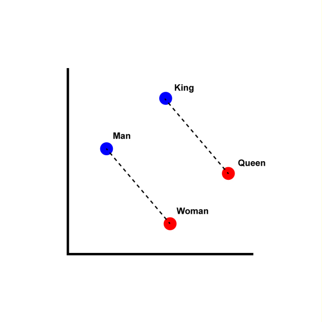


[1]

1. https://upload.wikimedia.org/wikipedia/commons/d/dd/RecurrentLayerNeuralNetwork.png. CC image courtesy of Chrislb on WikiMedia Commons licensed under CC-BY-SA-3.0.

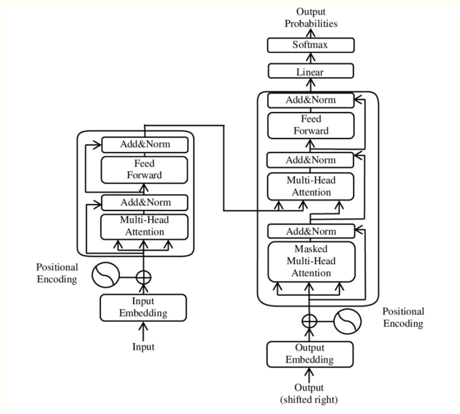
Special Architectures for Natural Language Processing

Word2vec



[1]

Transformers



[2]

1. https://commons.wikimedia.org/wiki/File:Word_vector_illustration.jpg. CC image courtesy of Singerep on WikiMedia Commons licensed under CC-BY-SA-4.0.
2. https://upload.wikimedia.org/wikipedia/commons/8/8f/The-Transformer-model-architecture.png. CC image courtesy of Yuening Jia on WikiMedia Commons licensed under CC-BY-SA-3.0.

Questions

- Are neural networks inspired by biological brains?

Questions

- Are neural networks inspired by biological brains?

  Yes. Their massively parallel architecture is more human-like and less like the classical, sequential von Neumann computing model. Edge weights are similar to the strength of synapse connections between nodes. Popular CNNs have parallels with the layered mammalian visual cortex. But there are also many differences.

Questions

- Are neural networks inspired by biological brains?

  Yes. Their massively parallel architecture is more human-like and less like the classical, sequential von Neumann computing model. Edge weights are similar to the strength of synapse connections between nodes. Popular CNNs have parallels with the layered mammalian visual cortex. But there are also many differences.

- Is deep learning the answer to AI?!

Questions

- Are neural networks inspired by biological brains?

  Yes. Their massively parallel architecture is more human-like and less like the classical, sequential von Neumann computing model. Edge weights are similar to the strength of synapse connections between nodes. Popular CNNs have parallels with the layered mammalian visual cortex. But there are also many differences.

- Is deep learning the answer to AI?!

  Far from! It is well-suited to certain key tasks in perception, but is not the method of choice for learning in many domains. It has little bearing on faculties such as reasoning and decision making.

Questions

- Are neural networks inspired by biological brains?

  Yes. Their massively parallel architecture is more human-like and less like the classical, sequential von Neumann computing model. Edge weights are similar to the strength of synapse connections between nodes. Popular CNNs have parallels with the layered mammalian visual cortex. But there are also many differences.

- Is deep learning the answer to AI?!

  Far from! It is well-suited to certain key tasks in perception, but is not the method of choice for learning in many domains. It has little bearing on faculties such as reasoning and decision making.

- Why are GPU's used for deep learning?

  Data sizes are large; many passes are needed to minimise error. Weight updates in Backprop can be parallelised.

Questions

- Are neural networks inspired by biological brains?

  Yes. Their massively parallel architecture is more human-like and less like the classical, sequential von Neumann computing model. Edge weights are similar to the strength of synapse connections between nodes. Popular CNNs have parallels with the layered mammalian visual cortex. But there are also many differences.

- Is deep learning the answer to AI?!

  Far from! It is well-suited to certain key tasks in perception, but is not the method of choice for learning in many domains. It has little bearing on faculties such as reasoning and decision making.

- Why are GPU's used for deep learning?

  Data sizes are large; many passes are needed to minimise error. Weight updates in Backprop can be parallelised.

- Are there off-the-shelf implementations of neural networks?

  Tonnes. Popular libraries include Theano, TensorFlow, Caffe, PyTorch, and Keras.

References

- Chapter 10, **A Course in Machine Learning**, Hal Daumé III. Available on-line at `http://ciml.info/`.