Lecture 20: Introduction to Neural Networks and Deep Learning

COMP90049
Knowledge Technologies

Sarah Erfani, Vinh Nguyen, Rao Kotagiri SCIS

Semester 1, 2018

THE UNIVERSITY OF MELBOURNE

What is Deep Learning?

A state-of-the-art machine learning technique that has consistently broken record in a range of difficult AI problems

- Computer vision: image/video recognition/understanding
- Natural language processing: speech recognition, machine translation, language understanding from scratch
- AI: GO playing @Grand Master level (Google DeepMind's AlphaGo vs. Lee Sedol)

Often involves training neural networks comprising many hidden layers

What is Deep Learning?

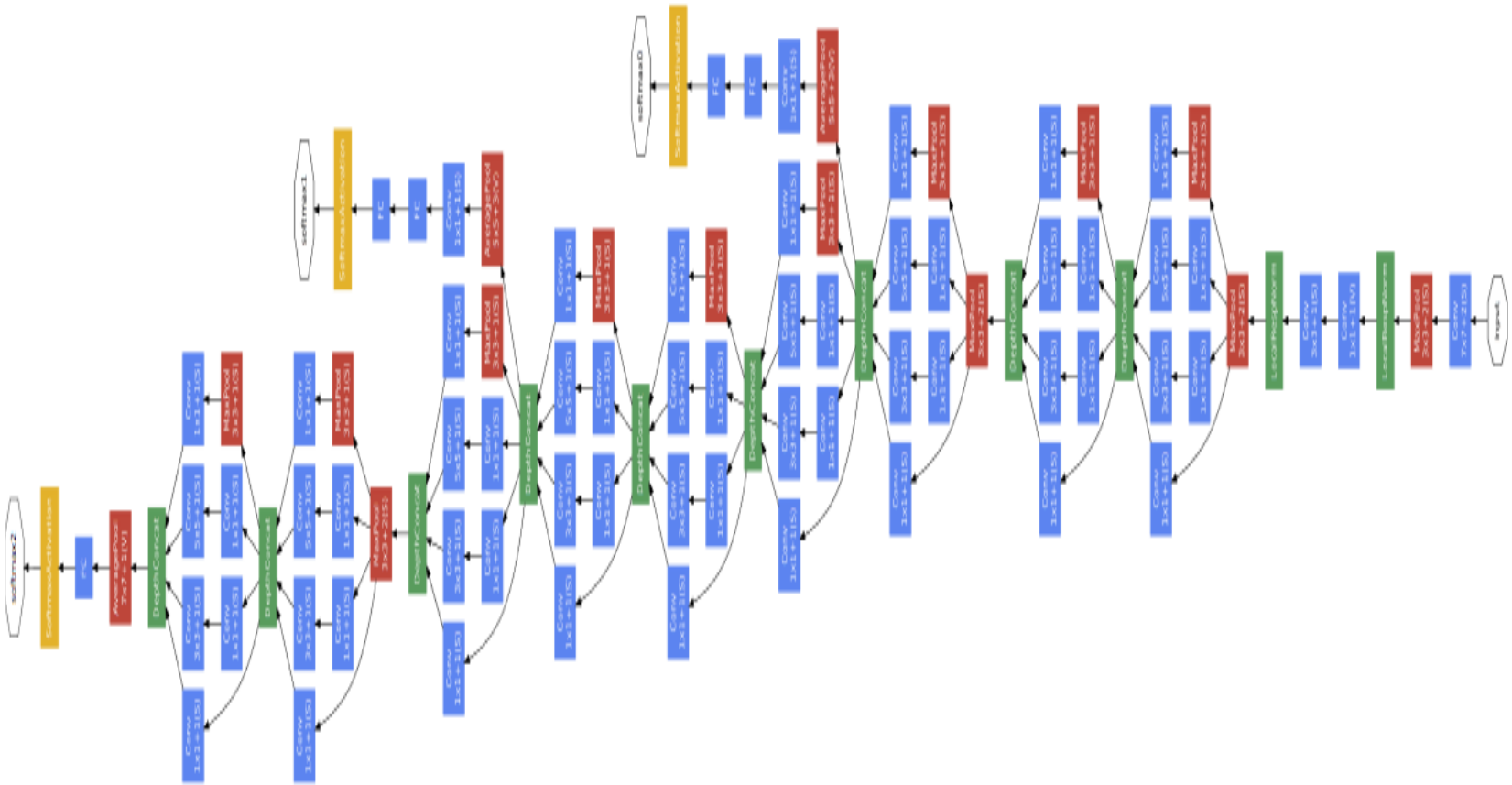


Figure 1: Google - Inception

What Deep Learning Can Do?

Image Captioning

http://deeplearning.cs.toronto.edu/i2t

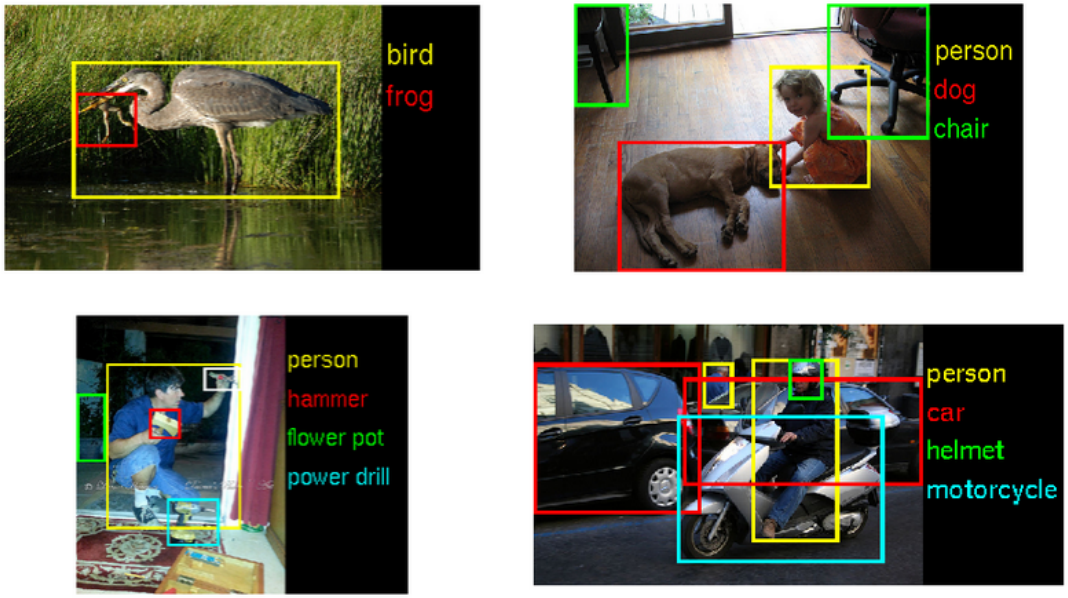
What Deep Learning Can Do?

Predicting age and gender

http://how-old.net/

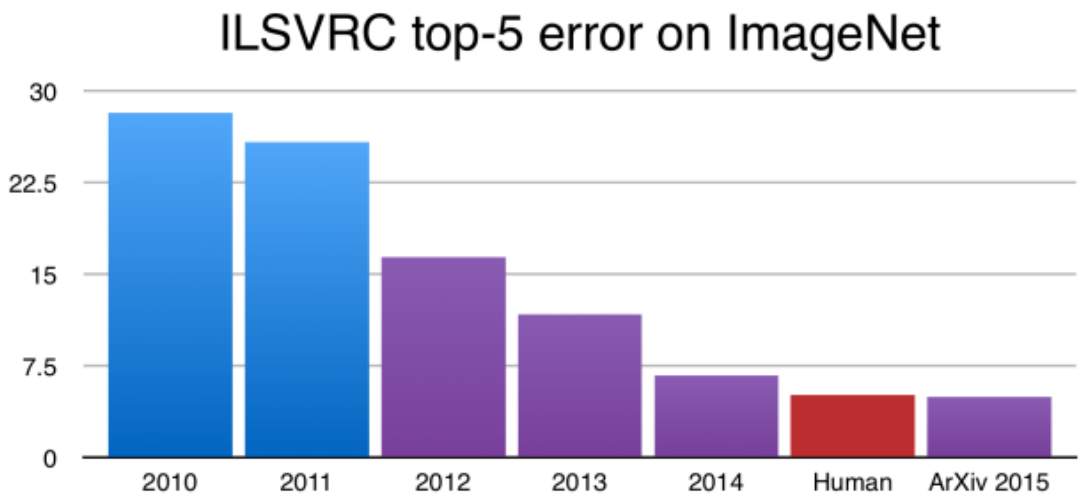
What Deep Learning Can Do?

ImageNet Large Scale Visual Recognition Competition (ILSVRC)



Source: http://image-net.org/challenges/LSVRC/2014/index

Benchmark in Computer Vision



- - 2011: Hand-crafted features (SIFT, HOG...) + traditional classifiers (SVM, Boosting...)
- 2012 - : Deep learning. Now DNNs perform better than Humans

Applications of Deep Learning-Powered Computer Vision

Vision is an essential component of AI systems

- Robotics: self driving car, robots
- Image search & tagging (Facebook DeepFace)
- Augmented reality

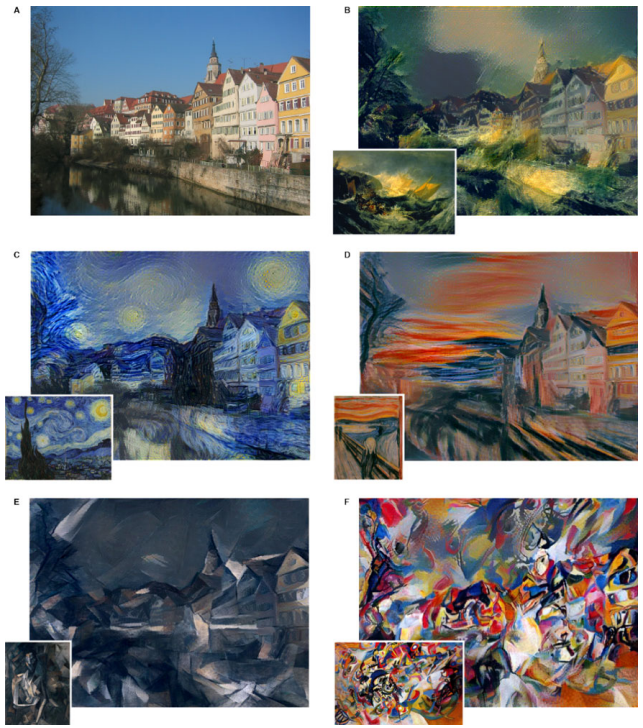
Other Domains

Speech is also an essential component of AI systems: Human-Computer Interface

Deep learning is making breakthrough in

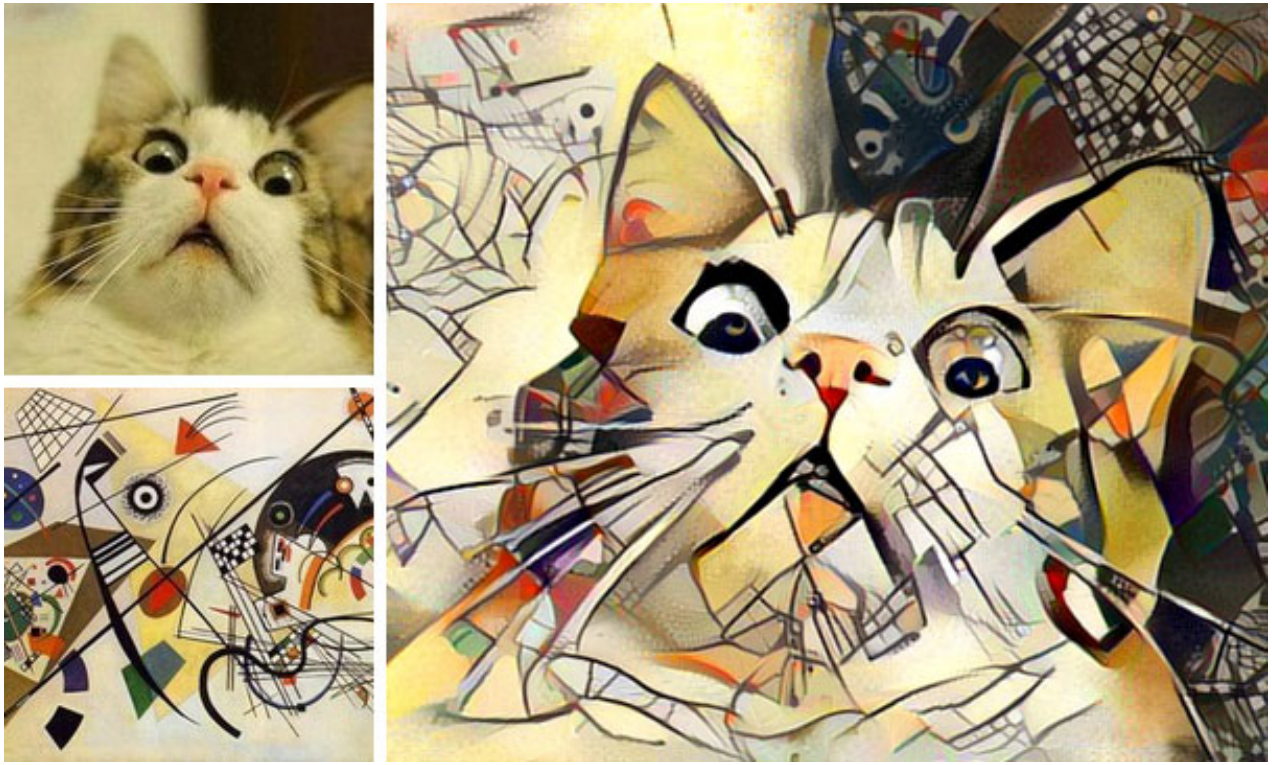
- Natural language processing: speech recognition/understanding, machine translation
  - Dictation, Siri
  - Live translation: Microsoft Skype https://www.youtube.com/watch?v=cJIILew6l28

Combining pictures



http://www.boredpanda.com/computer-deep-learning-algorithm-painting-masters/

Combining pictures



http://www.boredpanda.com/inceptionism-neural-network-deep-dream-art/

Combining pictures

Combining pictures

Combining pictures

Generating Text

SEED: Yes we can

Good morning. And as we mark the fact that they can stand with their companies that are consistent to the state of Pakistan and the United States of America.

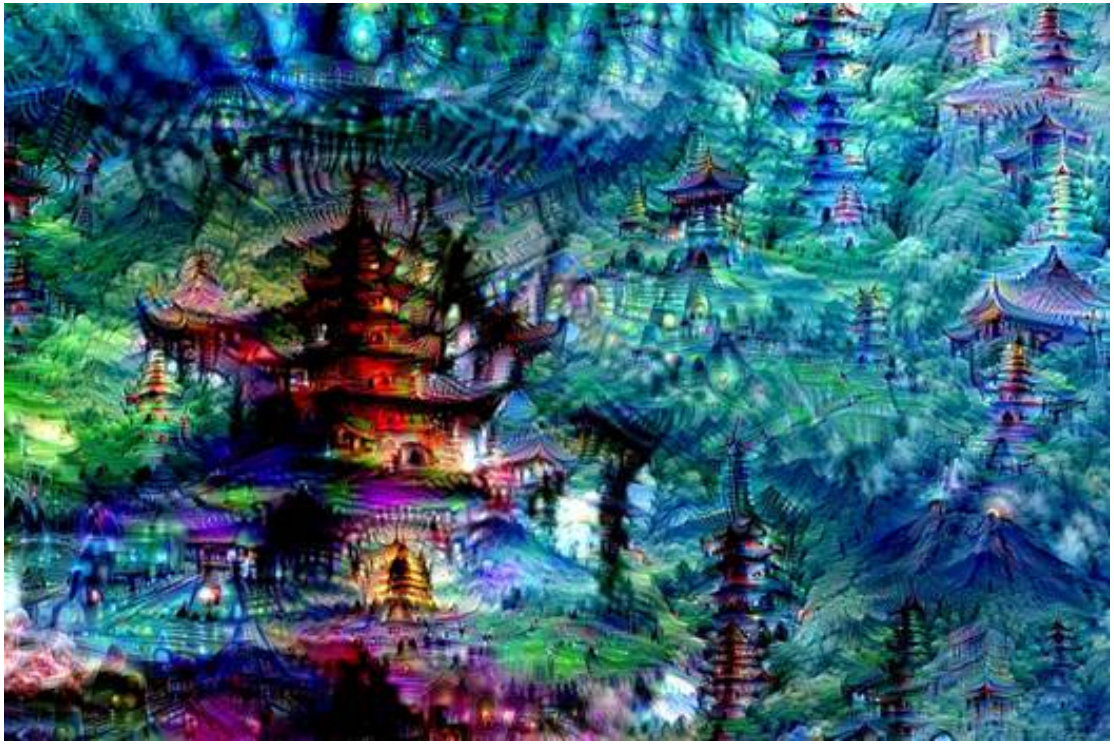
With the financial system we can do that. And the people of the United States will not be able to continue to support the people of the greatest problem of the American people to stay in the White House. And that's why I've got to recognize the private sector that there is no doubt that we've got to continue to shape the painful realisation that we are the United States of America. And these are the people of all Americans to be recognising the continent of the reason that they don't have to stay on the law.

It was the first time in Muslim Americans to see your own lives. And the United States will be better profitable and security and prosperity to the American people —she was a candidate for the same recession that they have to make sure that they take a state in the United States of America.

Thank you very much. God bless you. God bless you. God bless you. God bless you.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

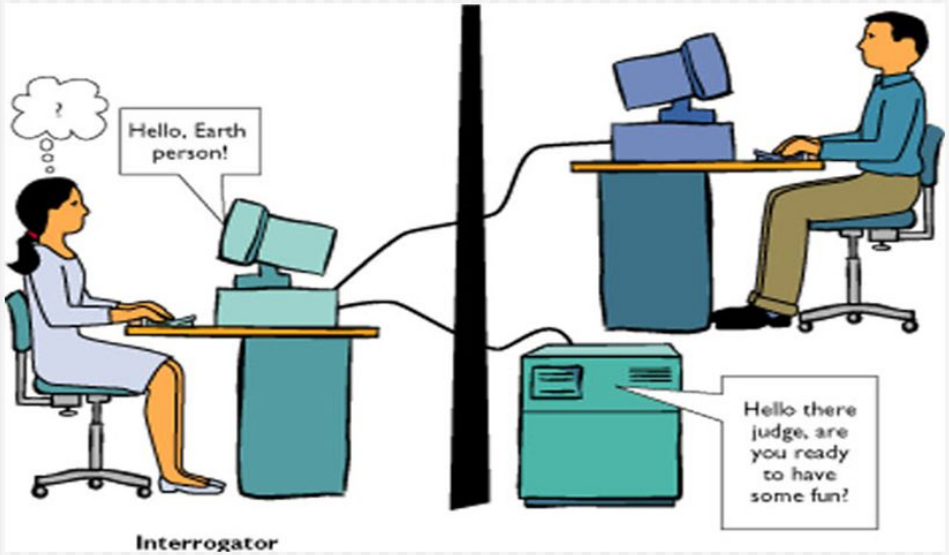
Generating Images



http://deepdreamgenerator.com/

Generating Music

Turing Test Example

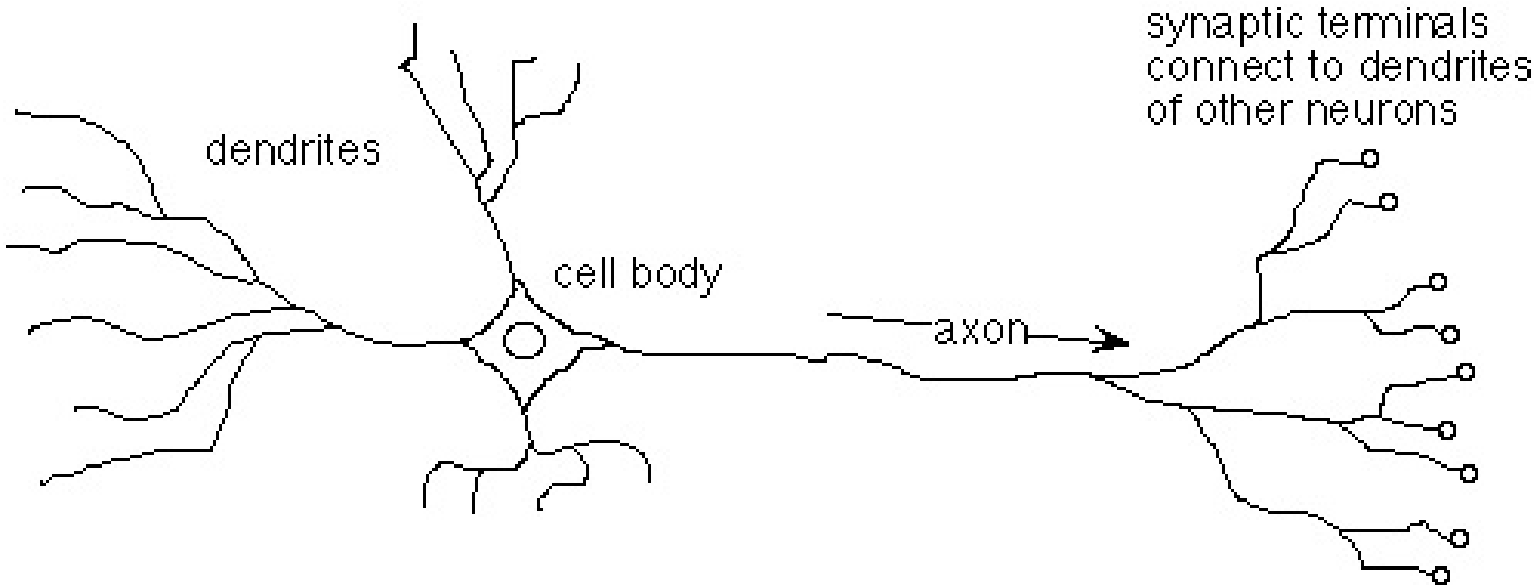


Neural Network Recap

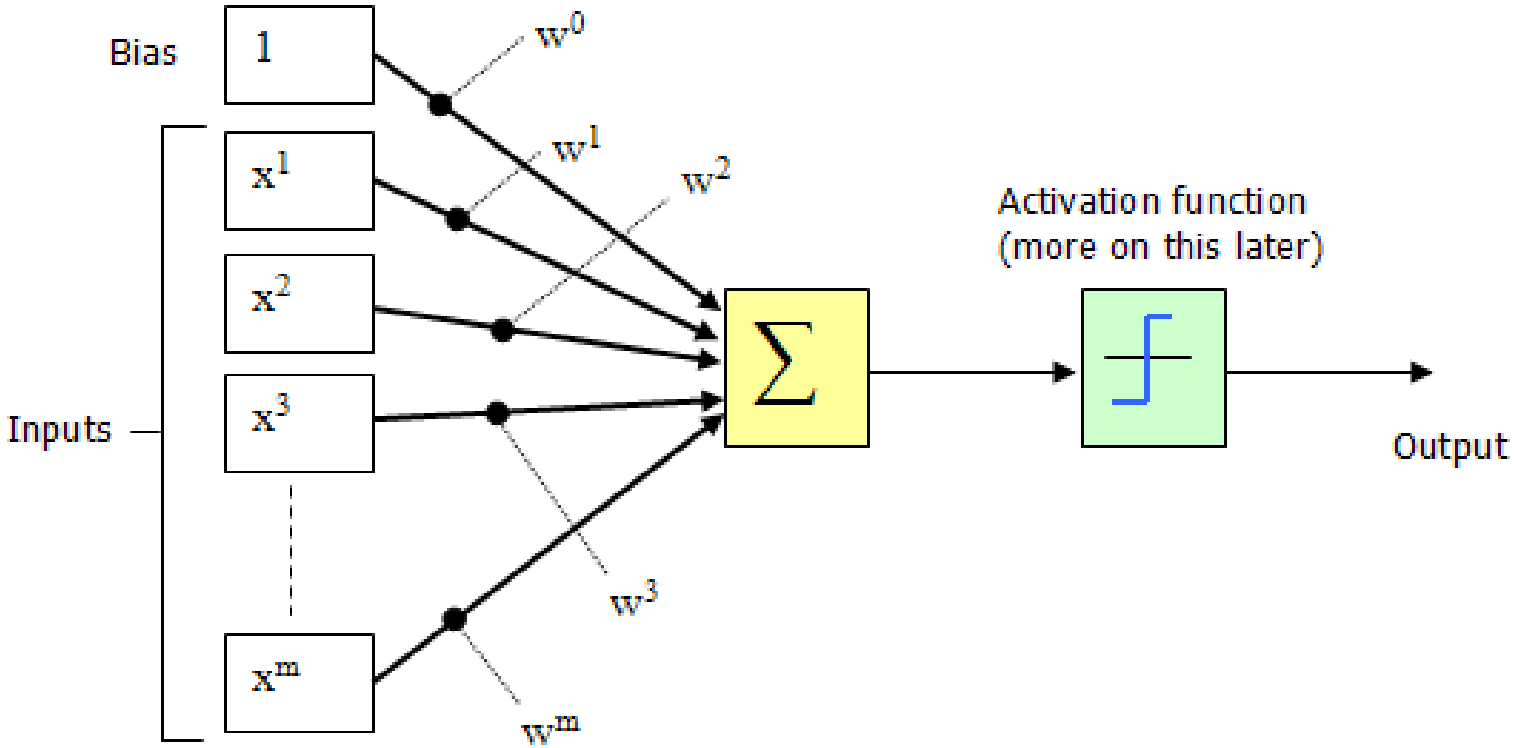
- Versatile models: applicable to many machine learning tasks
- Best suited to data that has temporal or spatial order, e.g., image, text, sequences (time series, DNA...)
- State of the art performance that pushes the limit of machine learning and AI recently

A Historical Perspective

- The first artificial neural networks were invented in 1950's
- Inspired by the biological brain



A Historical Perspective



http://www.codeproject.com/Articles/16419/AI-Neural-Network-for-beginners-Part-of

What does an artificial neurone do?

- Receive signals from input neurones: $x_1, x_2, \ldots, x_m$
- Weight signals according to the link strength between neurons: $w_1 x_1, w_2 x_2, \ldots, w_m x_m$
- Add the input signals: $w_1 x_1 + w_2 x_2 + \ldots + w_m x_m = \sum_{i=1}^{m} w_i x_i$
- Emit an output signal: activation function $f_A(.)$

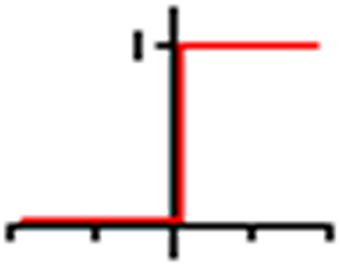
$$f_A\left(\sum_{i=1}^{m} w_i x_i\right)$$

Activation functions

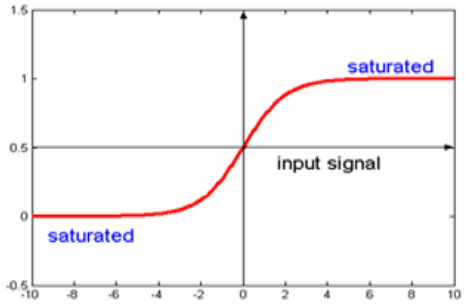
- Step function

$$f_{step}(x) = \begin{cases} 1 & \text{if } x \geq b; \\ 0 & \text{if } x < b. \end{cases}$$

- Sigmoid function: $f_{sigm}(x) = \frac{1}{1+e^{-\beta x}}$



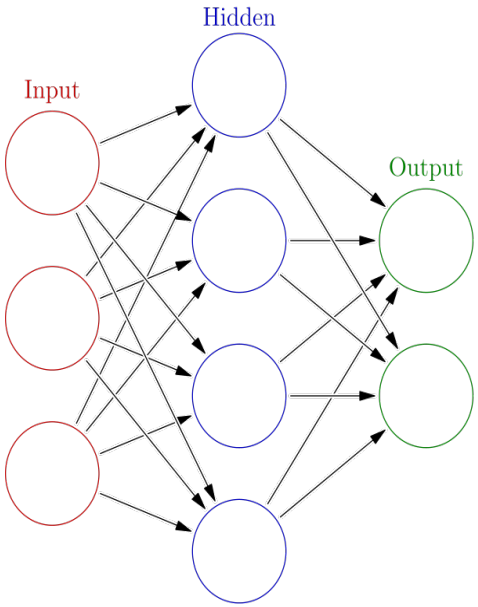
(a) Step



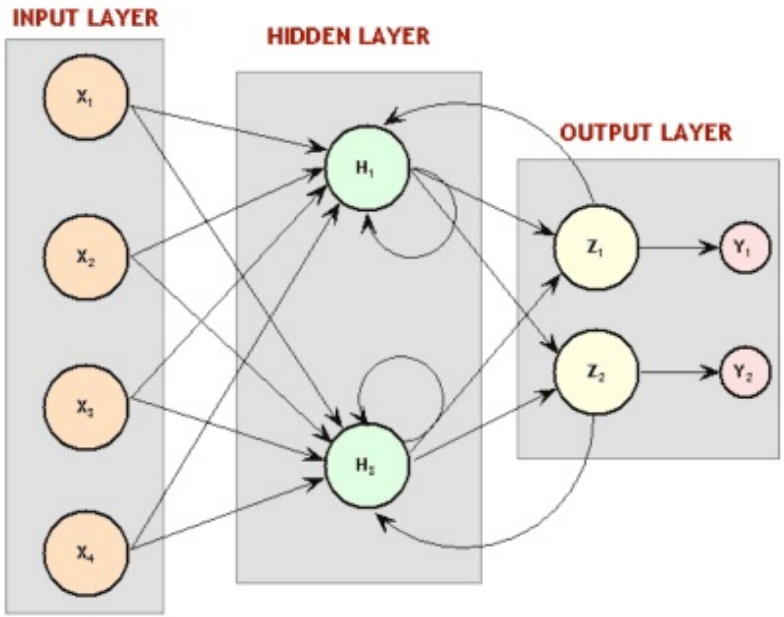
(b) Sigmoid

Network Architecture

- Feed-forward NN
- Recurrent NN



(c) Feed-forward



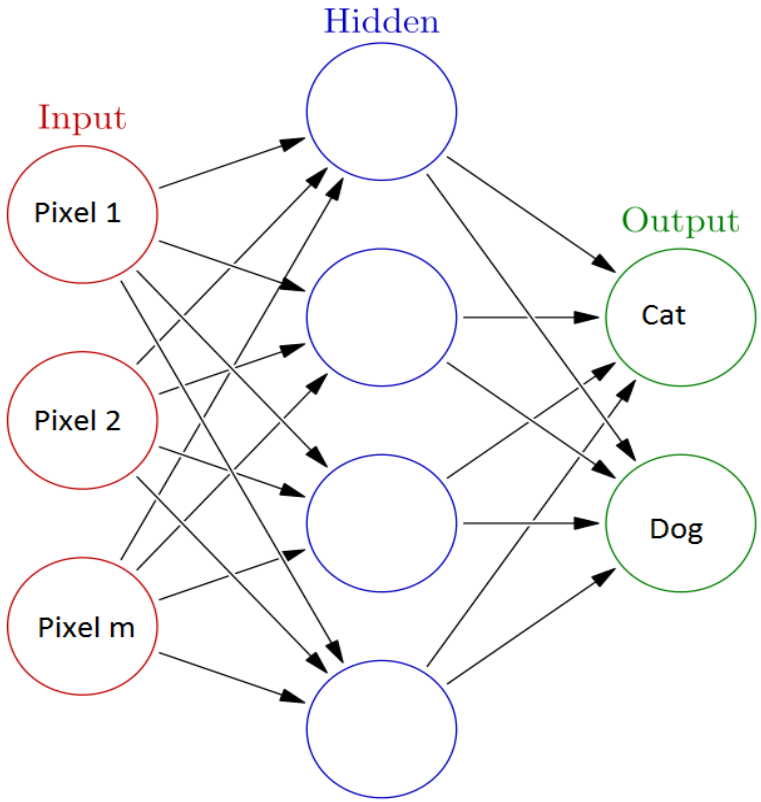
(d) Recurrent

Network Training

- Find a set of weights so that the the network exhibits the desired behaviour
- Example: cat vs. dog

https://www.kaggle.com/c/dogs-vs-cats/data

Training Data: input & label



Input: pixels, Output: {1-1} for cat & dog, {1-0} for cat only, {0-1} for dog only, {0-0} if no cat & dog

Error Function

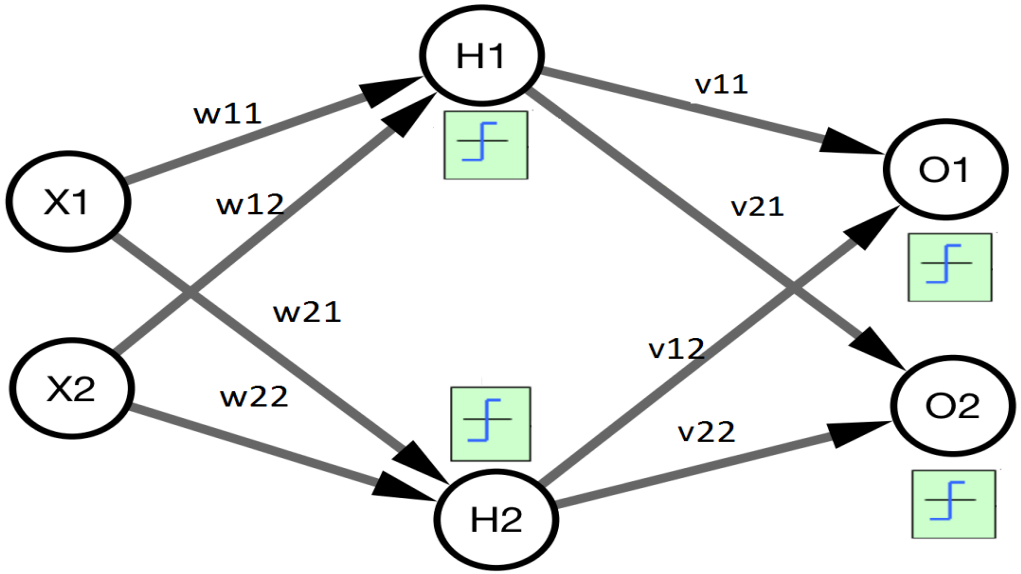
- Measure the difference between actual output and expected output
- One popular measure: sum of squared error

$$E_{NN}(input, weights, label) = \sum (output - label)^2$$

$$output = f_{NN}(input, weights)$$

- Note: Neural network is a composite/nested function that map the input to the output. f_{NN}: transfer function

Functional Form of Neural Networks



$$
\begin{aligned}
H_1 &= f_A(X_1 w_{11} + X_2 w_{12}) && (1)\\
H_2 &= f_A(X_1 w_{21} + X_2 w_{22}) && (2)\\
O_1 &= f_B(H_1 v_{11} + H_2 v_{12}) && (3)\\
&= f_B(f_A(X_1 w_{11} + X_2 w_{12}) v_{11} + f_A(X_1 w_{21} + X_2 w_{22}) v_{12}) \\
O_2 &= f_B(H_1 v_{21} + H_2 v_{22}) && (4)\\
&= f_B(f_A(X_1 w_{11} + X_2 w_{12}) v_{21} + f_A(X_1 w_{21} + X_2 w_{22}) v_{22}) \\
output &= f_{NN}(input, weights) && (5)
\end{aligned}
$$

Minimizing the Error Function

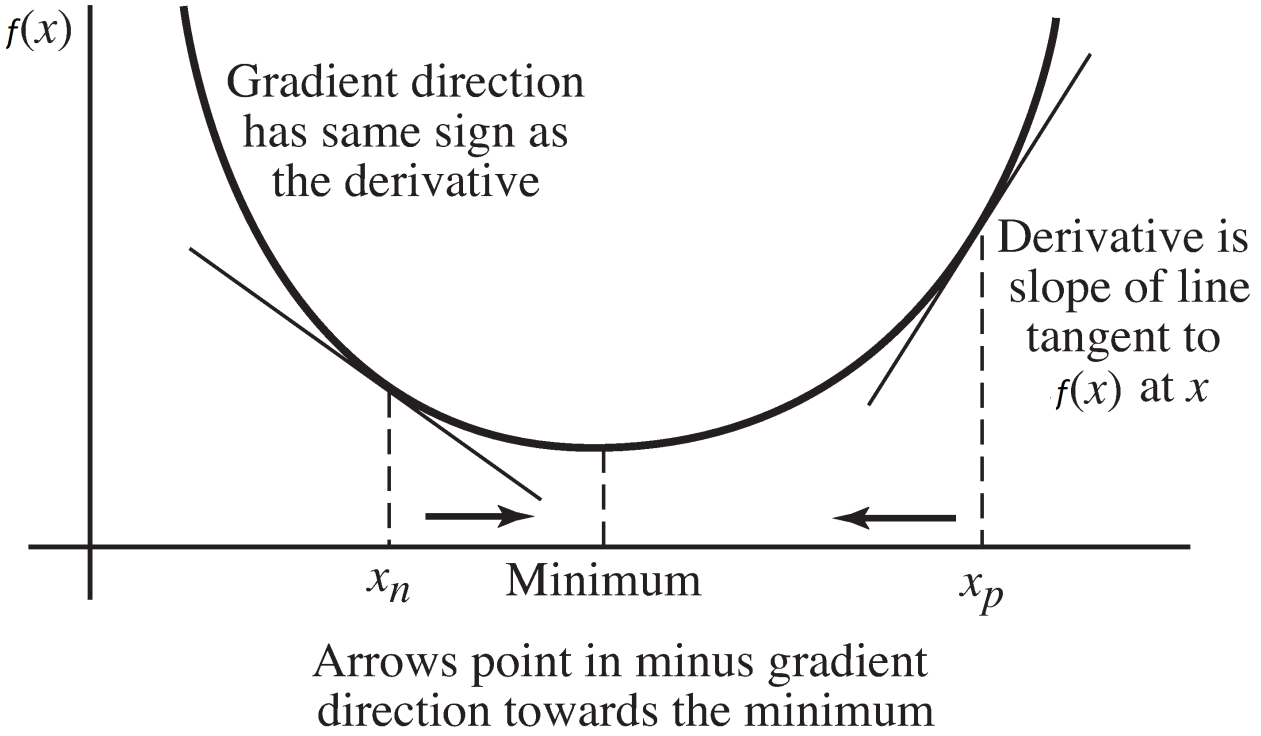
- Training Neural network is to minimize the error function with respects to the edge weights: $\min\ E_{NN}(\mathbf{w})$
- Gradient descent: iterative algorithm for finding a local minimum of a function
  - Compute the gradient of the error function

$$\frac{\partial E_{NN}}{\partial \mathbf{w}} = \left(\frac{\partial E_{NN}}{\partial w_{11}}, \frac{\partial E_{NN}}{\partial w_{12}}, \dots \right)$$

  - Weights update: take small step in the direction of negative gradient

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial E_{NN}}{\partial \mathbf{w}}$$

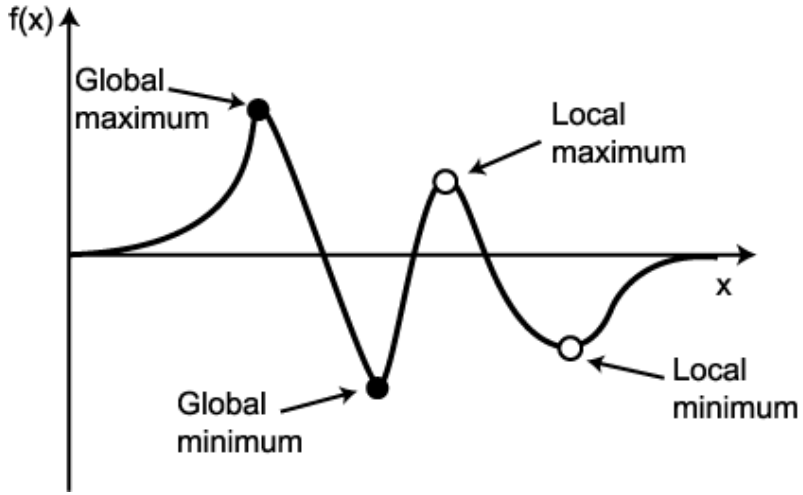
Intuition behind Gradient Descent



Negative gradient points towards a local minimum.

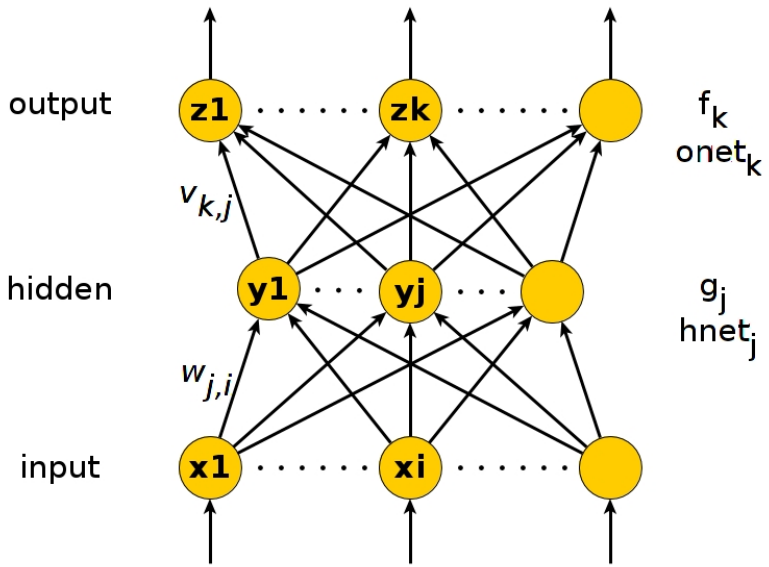
Intuition behind Gradient Descent

The iterative algorithm might converge to one of the many local minima



In neural networks, $\mathbf{w}$ is updated through a procedure called back propagation which computes gradients with respect to each weight in the network.

Training Feed Forward Networks



$hnet_j$: pre-activation value of hidden nodes; $g_j(\cdot)$: hidden node activation function

$onet_k$: pre-activation value of output nodes; $f_k(\cdot)$: output activation function

Feed forward phase

How is the output z_k related to the input x's?

$$
\begin{aligned}
z_k &= f_k(onet_k) \\
&= f_k(\sum_j v_{k,j} y_j) \\
&= f_k[\sum_j v_{k,j} g_j(hnet_j)] \\
&= f_k[\sum_j v_{k,j} g_j(\sum_i w_{j,i} x_i)]
\end{aligned}
$$

How to train the networks?

- A training example is a vector of inputs and the desired output(s)
  i.e. $(\{x_1, ..., x_n\}, \{t_1, \ldots, t_c\})$.
  $t_i = 1$ iff the data point $\{x_1, ..., x_n\}$ belongs to the i-th class (one-hot encoding).
- Recall we want to train the weights $v_{j,i}$ and $w_{k,j}$ to minimise the difference between z_k and t_k for each of our training inputs.
- define an **error function** E to be the *sum of squared errors*.

$$E = \frac{1}{2}\sum_{k}^{c}(t_k - z_k)^2$$

- If we think of E as height, it defines an error **landscape** on the weight space. The aim is to find a set of weights for which E is very low.
- This is done by moving in the steepest downhill direction (**Gradient descent:**), i.e., $-\frac{\partial E}{\partial w_i}$

$$w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}$$

Gradient Descent

- The gradient of the error function is computed via the **backpropagation algorithm**.
- We can calculate $\frac{\partial E}{\partial w_i}$ only if we use *differentiable* transfer functions.
- Use the **sigmoid** function $f(x) = \frac{1}{1+e^{-x}}$
- This transfer function has the nice property

$$\frac{\partial f}{\partial x} = f(x)(1 - f(x))$$

Backpropagation – Basic Algorithm

Backpropagation(network, training examples D)

Initialise the weights $v_{k,j}$ and $w_{j,i}$ to small random values

Repeat

For each $d \in D, d = \langle x_1, ..., x_n, t_1, \ldots, t_c \rangle$

Forward pass: calculate z_k, $onet_k$, y_j, $hnet_j$

Backward pass:

Calculate error between z_k and t_k at output

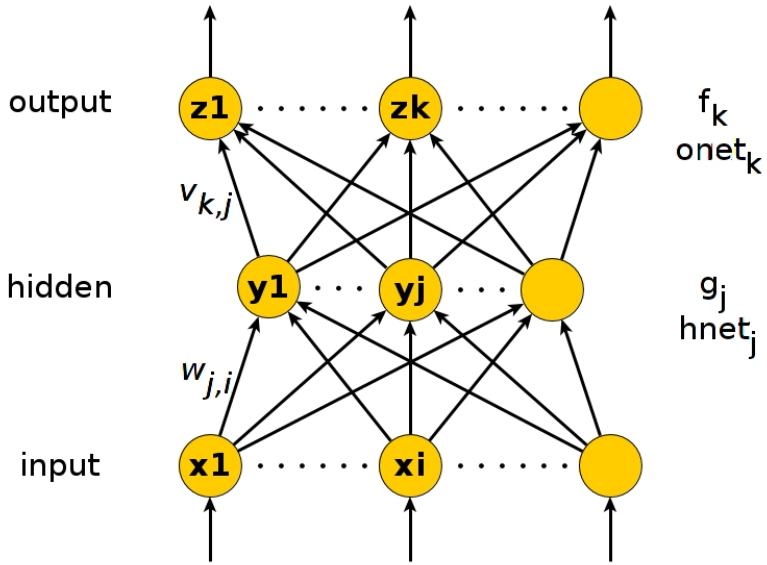
Update the weights in each layer, $v_{k,j}$ and $w_{j,i}$

in proportion to their effect on the error

using gradient descent: $w_{j,i} \leftarrow w_{j,i} - \eta \frac{\partial}{\partial}$

Until network has *converged*

Computing gradient: Repeated application of chain rule



Systematic application of the chain rule, layer by layer: $y(x) = y(u(x)) \rightarrow \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \frac{\partial u}{\partial x}$

Gradient w.r.t. Output Layer Weights

For a single example d, the effect of changes in output weight $v_{k,j}$ on the output error $E = \frac{1}{2}(t_k - z_k)^2$ is:

$$\frac{\partial E}{\partial v_{k,j}} = \delta_k \, y_j$$

where

$$\delta_k = -(t_k - z_k) \, f_k'(onet_k)$$

Using gradient descent, $\Delta v_{k,j} = -\eta \frac{\partial E}{\partial v_{k,j}}$.

Hence, update rule for output layer weights is

$$\boxed{v_{k,j} \leftarrow v_{k,j} - \eta \, \delta_k \, y_j}$$

Derivation – Gradient w.r.t. output Layer Weights

Remember the chain rule: $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}$

$$\frac{\partial E}{\partial v_{k,j}} = \frac{\partial}{\partial z_k}[\frac{1}{2}(t_k - z_k)^2]\frac{\partial z_k}{\partial v_{k,j}} \tag{6}$$

$$= -(t_k - z_k)\frac{\partial f_k(onet_k)}{\partial v_{k,j}} \tag{7}$$

$$= -(t_k - z_k)\frac{\partial f_k(onet_k)}{\partial onet_k}\frac{\partial onet_k}{\partial v_{k,j}} \tag{8}$$

$$= -(t_k - z_k)f'_k(onet_k)\frac{\partial \sum_j v_{k,j}y_j}{\partial v_{k,j}} \tag{9}$$

$$= -(t_k - z_k)f'_k(onet_k)y_j \tag{10}$$

Gradient w.r.t. hidden Layer Weights

Similarly for hidden layer weights

$$\frac{\partial E}{\partial w_{ji}} = \delta_j \, x_k$$

where

$$\delta_j = g_j'(hnet_j) \sum_k^c \delta_k v_{k,j}$$

Using gradient descent, $\Delta w_{kj} = -\eta \, \frac{\partial E}{\partial w_{kj}}$.

Hence, update rule for hidden layer weights is

$$\boxed{w_{kj} \leftarrow w_{kj} - \eta \, \delta_j \, x_i}$$

Derivation – Gradient w.r.t. hidden Layer Weights

$$\frac{\partial E}{\partial w_{j,i}} = \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial w_{j,i}} = \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial onet_k}\frac{\partial onet_k}{\partial w_{j,i}} \tag{11}$$

$$= \sum_k^c \left(-(t_k - z_k) f'(onet_k) \frac{\partial(\sum_k v_{k,j} y_j)}{\partial y_j} \right) \frac{\partial y_j}{\partial w_{j,i}} \tag{12}$$

$$= \sum_k^c \left(\delta_k v_{k,j} \right) \frac{\partial y_j}{\partial hnet_j}\frac{\partial hnet_j}{\partial w_{j,i}} \tag{13}$$

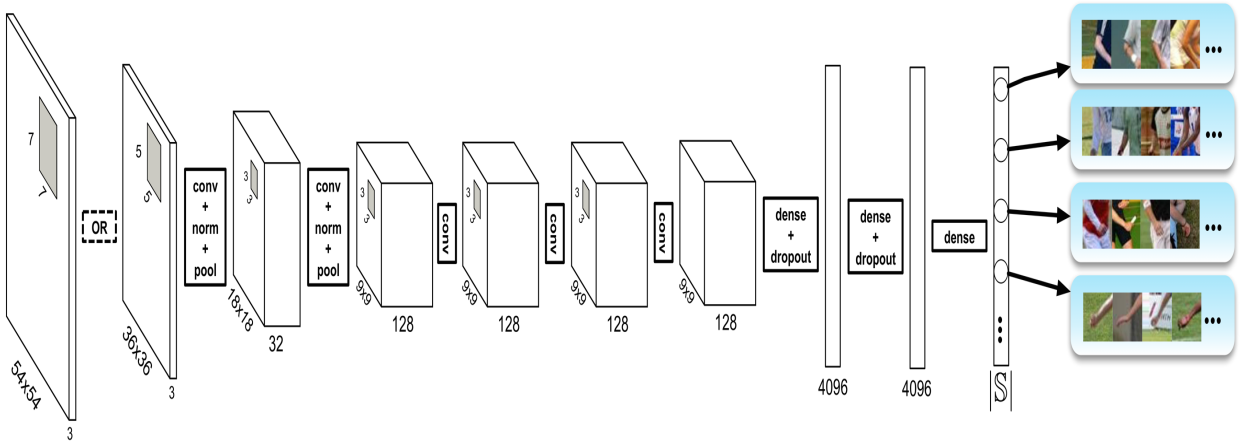
$$= \sum_k^c \left(\delta_k v_{k,j} \right) g_j'(hnet_j) \frac{\partial(\sum_i w_{j,i} x_i)}{\partial w_{j,i}} \tag{14}$$

$$= \sum_k^c \left(\delta_k v_{k,j} \right) g_j'(hnet_j) x_i \tag{15}$$

Why Deep Learning Works Now? (but not 30 years ago)

- Lots more computing power
  - GPU: mini-super computer: 3000 cores, 12GB RAM, 100x speed up over CPU
  - Much deeper and larger networks, trained for a few days-a week

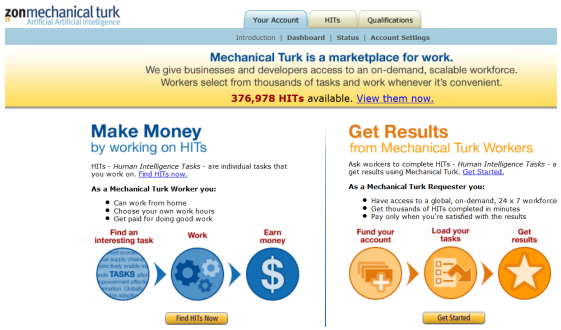
(e) GTX Titan X



(f) A deep network

Why Deep Learning Works Now? (but not 30 years ago)

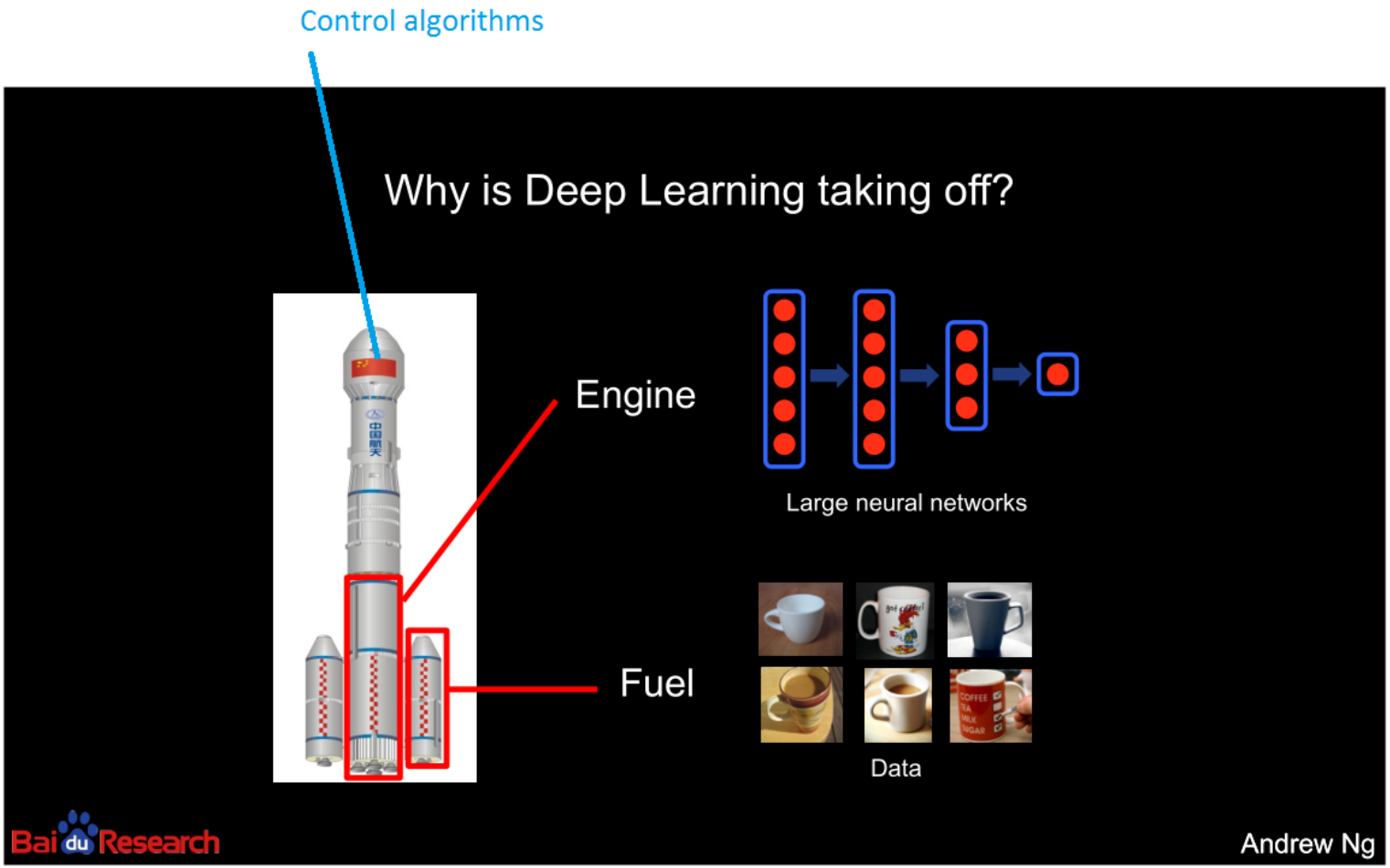
- Lots more data
  - phone, camera, sensors
  - Internet, Croud-Sourcing (Amazon Mechanical Turk) $\rightarrow$ huge labeled data sets of millions of training samples



Why Deep Learning Works Now? (but not 30 years ago)

- More sophisticated algorithms
  - Dropout: strong regularization that prevent deep nets from overfitting
  - Clever initialization: initial weights are close to a good minimum
  - Stochastic gradient descent: fast approximate gradient descent algorithms
  - Activation function that facilitate training of deep nets, e.g., Rectified Linear Unit: $\text{ReLU}(x) = \max(0, x)$

Why Deep Learning Works Now? (but not 30 years ago)



How to Apply Neural Networks

Many readily available implementations:

- Scikit-learn, Matlab, Weka: Suitable for small scale problems
- Modern implementations: TensorFlow (Google), Torch (Facebook), CNTK (Microsoft), Theano (U. Toronto), Keras (Python wrapper)
  - Support training on (multiple) GPUs
  - Support advanced network structures: convolutional neural networks (for image recognition), recurrent neural networks (for sequential data, e.g., text)

Network architectures/parameters

- Number of input nodes = number of features
- Number of output nodes = number of classes
  $d = (\{x_1, ..., x_n\}, \{t_1, \ldots, t_c\})$ where $t_i = 1$ iff the training data belong to the i-th class (one-hot encoding).
  Neural networks can deal naturally with multi-class classification problems (compare to SVM?)
- Number of hidden layers
- Number of nodes in each hidden layers
- Regularization parameters (similar to C in SVM): control the complexity of the model, preventing overfitting.

Reading

http://neuralnetworksanddeeplearning.com/chap1.html

http://neuralnetworksanddeeplearning.com/chap2.html

Further Resources

http://www.wired.com/2014/01/geoffrey-hinton-deep-learning
http://chronicle.com/article/The-Believers/190147/

Courses

- https://class.coursera.org/neuralnets-2012-001
- https://www.coursera.org/course/ml

Book

- http://www.deeplearningbook.org/

Summary

- dL