Machine Learning
Neural Networks

Dr Ioannis Patras

Slide thanks: Tim Hospedales

Course Context

- Supervised Learning
  - (Linear) regression
  - (Linear) Classifiers and Logistic Regression
  - Neural Networks
- Unsupervised
  - Clustering
  - Density Estimation
  - HMMs

Overview

- Neural Nets Intro
- From Neurons to Neural Nets
- Multi-Layer Neural Nets for Non-Linear Functions
- Neural Net Prediction Details
- Training Neural Nets
- More considerations

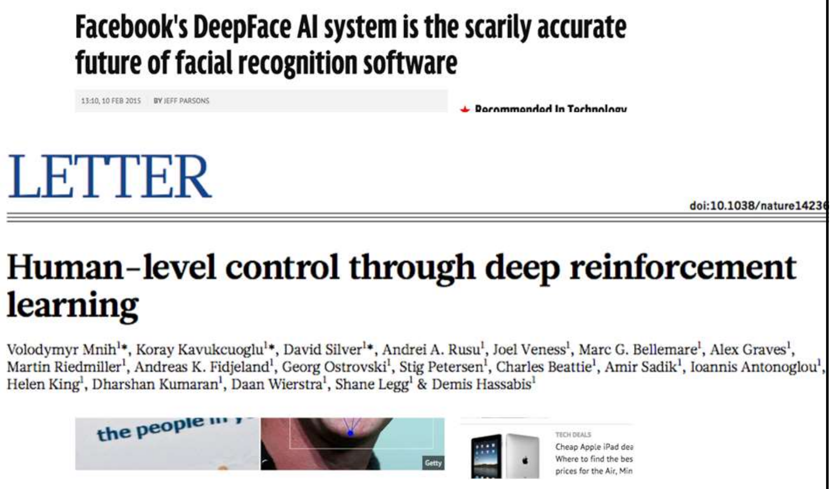
Intro to Neural Nets

- *The term '**neural network**' has origins in attempts to find mathematical representations of information processing in biological systems.*
    - *(McCulloch and Pitts, 1943; Widrow and Hoff, 1960; Rosenblatt, 1962; Rumelhart et al., 1986).*
- Neural Networks:
    - Inspired from algorithms that try to mimic the brain
    - Formalized as systems of interconnected "neurons"
    - Each neuron does a very simple calculation
    - Many layers of neurons put together.
    - The collective can do qualitatively more powerful computation than any individual unit.

Neural Nets in 2017!

- Until late 00's, they were largely shunned.
  - Now they are back!
- Superhuman performance at:
  - Character recognition
  - Face recognition
  - Object recognition
  - Atari video game playing
  - Go playing(!)
- Pervasive:
  - Run OCR in banks/mail.
  - Every facebook photo upload (millions per day) goes through a Deep CNN.
  - Run Cortana, Google's, Siri speech recognition.

In the Media



Facebook's DeepFace AI system is the scarily accurate future of facial recognition software

13:10, 10 FEB 2015 BY JEFF PARSONS

Recommended In Technology

LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

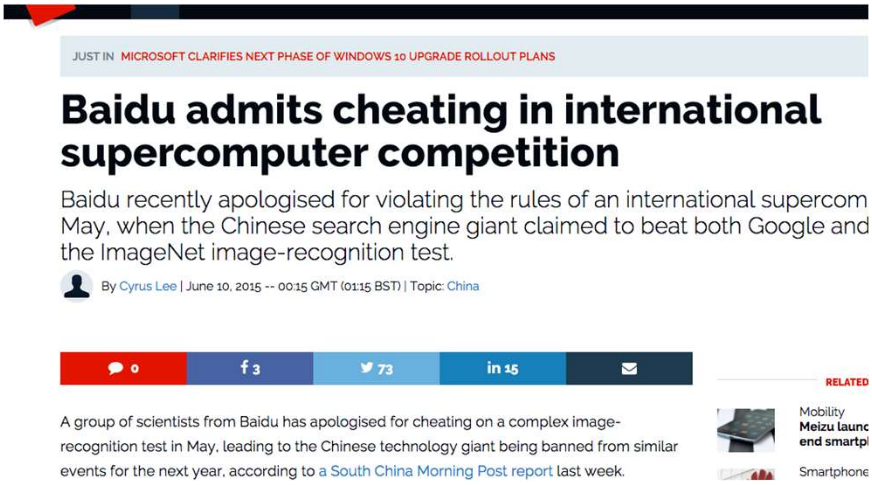
Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

TECH DEALS

Cheap Apple iPad dea
Where to find the bes
prices for the Air, Min

Deepmind Neural Atari

…and scandal!



JUST IN MICROSOFT CLARIFIES NEXT PHASE OF WINDOWS 10 UPGRADE ROLLOUT PLANS

Baidu admits cheating in international supercomputer competition

Baidu recently apologised for violating the rules of an international supercom
May, when the Chinese search engine giant claimed to beat both Google and
the ImageNet image-recognition test.

By Cyrus Lee | June 10, 2015 -- 00:15 GMT (01:15 BST) | Topic: China

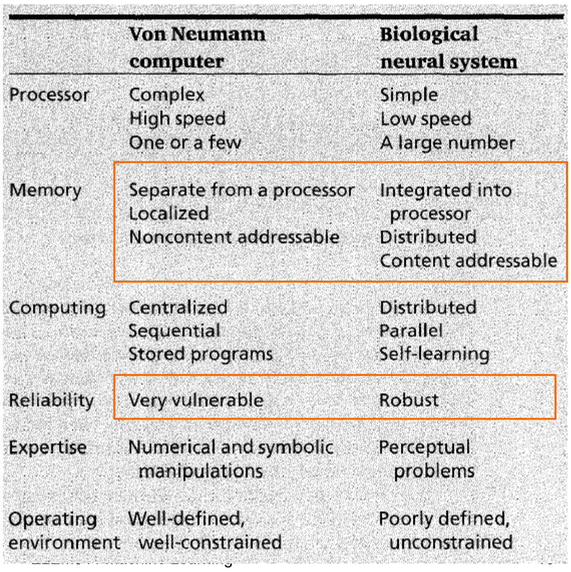
A group of scientists from Baidu has apologised for cheating on a complex image-recognition test in May, leading to the Chinese technology giant being banned from similar events for the next year, according to a South China Morning Post report last week.

Brain vs Computers

- Brain is massively parallel, but slow
  - 100 billion (10^{11}) neurons
  - 100 trillion ($10^{14\text{-}15}$) connections (synapses)
  - 10^2 operations per second
- Contrast fast serial computers
  - 1-10^3 processors
  - 1-10 billion operators per second

Von Neuman computer versus biological neural system

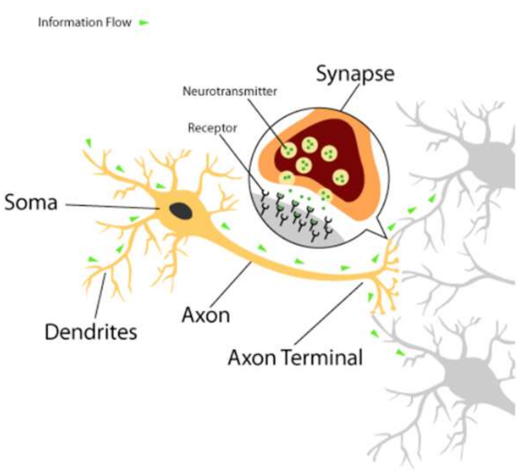
Artificial neural networks have an architecture roughly similar to the biological neural system



	Von Neumann computer	Biological neural system
Processor	Complex High speed One or a few	Simple Low speed A large number
Memory	Separate from a processor Localized Noncontent addressable	Integrated into processor Distributed Content addressable
Computing	Centralized Sequential Stored programs	Distributed Parallel Self-learning
Reliability	Very vulnerable	Robust
Expertise	Numerical and symbolic manipulations	Perceptual problems
Operating environment	Well-defined, well-constrained	Poorly defined, unconstrained

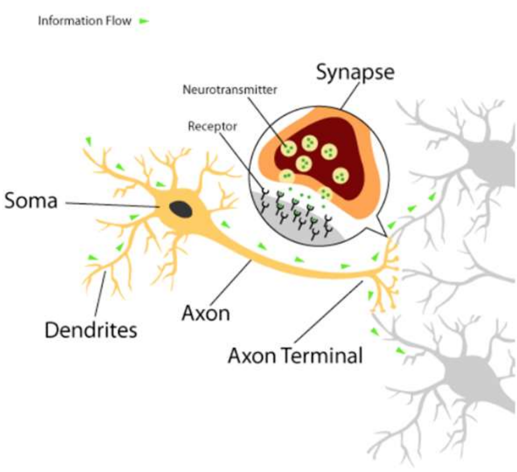
Real Neurons in Real Brains

- The Brain inspires NNs.
- In turn NNs and ML could (??) help us understand how the brain works
- Brains do massively parallel computation.
- Synapses change strength (cf weight change in ML model)
- Local signals used to change the strength



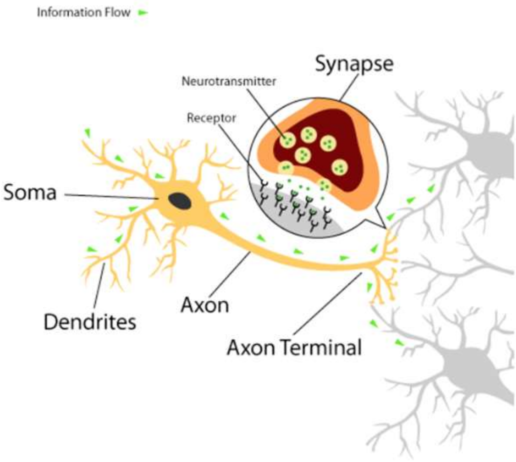
Real Neurons in Real Brains

- Dendrites collect incoming signals
- Polarization builds up in cell…
- After reaching a charge threshold, neuron fires
- Impluse travels down Axon
- And into other neurons via synapses
- …and repeat



Real Neurons in Real Brains

- Real neurons use spikes not continuous outputs
- However, they have a minimum (0) and maximum firing rate
- Artificial neurons can roughly be considered to simulate natural neurons at firing rate level



Artificial Neural Networks

- Nodes (cf neurons)
- Adaptive Weights (cf synaptic strength)
- Defined by:
  - Interconnection pattern between layers of neurons
  - The activation function that converts a neuron’s weighted input to its output activation.
  - The learning process for updating the interconnection weights

- Can model arbitrarily complex non-linear functions of inputs.

Fixed Versus Adaptive Basis Functions

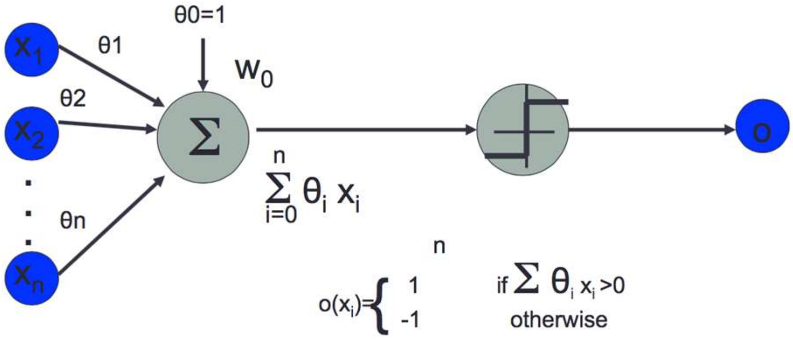
- Previously we used fixed linear combinations of features.
- Limited by the curse of dimensionality
- In contrast, Neural Nets adapt the parameters of the basis functions during training.

Overview

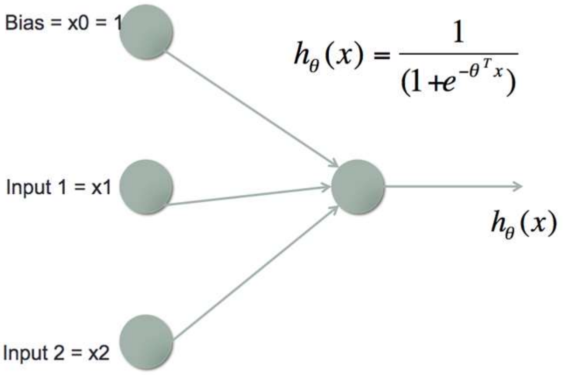
- Neural Nets Intro
- From Neurons to Neural Nets
- Multi-Layer Neural Nets for Non-Linear Functions
- Neural Net Prediction Details
- Training Neural Nets
- More considerations

Perceptron: A Simple Model Neuron

- Uses the step function as activation
  - Linear Threshold Unit (LTU)


θ0=1
x_1
θ1
w_0
θ2
x_2
Σ
$\sum_{i=0}^{n} \theta_i x_i$
θn
x_n
o
$o(x_i)=\begin{cases} 1 & \text{if } \sum^{n} \theta_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$

Neuron as a Logistic Regression Unit



$$h_\theta(x) = \frac{1}{(1+e^{-\theta^T x})}$$

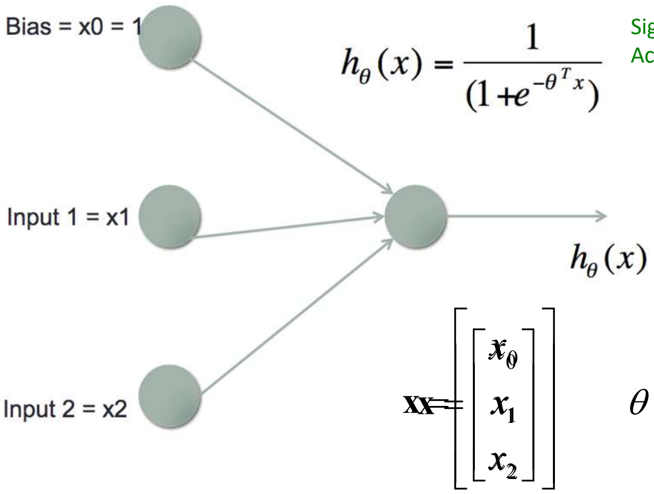
Can think of a logistic regression unit as a neuron (function) that multiplies the input by the parameters (weights) and squashes the resulting sum through the sigmoid.

- Logistic Regression Unit => Perception if weight strength goes to infinity, so sigmoid => step.

Feed Forward Neural Net

- Connected set of logistic regression units
  - Arranged in layers.
  - Each unit's output is a non-linear function (e.g., sigmoid, step function) of a linear combination of its inputs.
- Input layer size:
  - Size of input to classify/regress
- Output layer size
  - Size of target for classification/regression
- Any number of hidden neurons
- Any number of hidden layers

A Sigmoid/Logistic Neuron



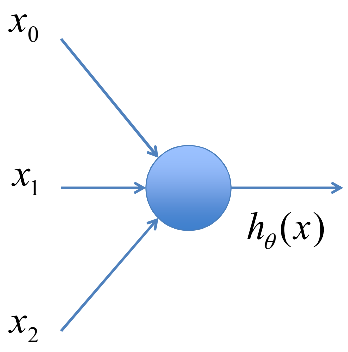
$$h_\theta(x) = \frac{1}{(1 + e^{-\theta^T x})}$$

Sigmoid Activation

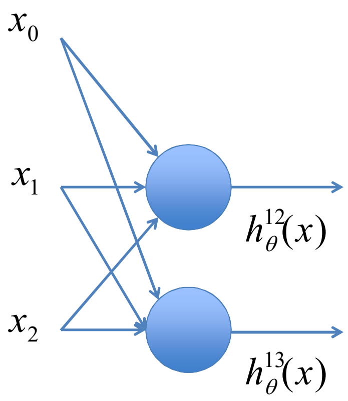
Weights

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

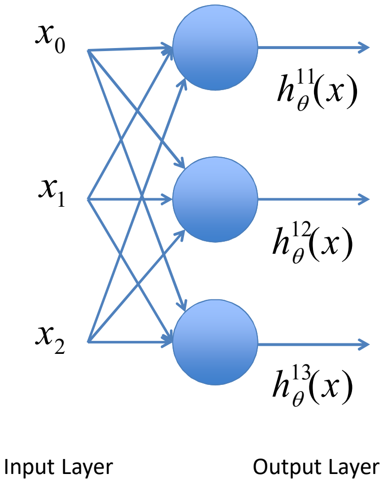
A Single Neuron



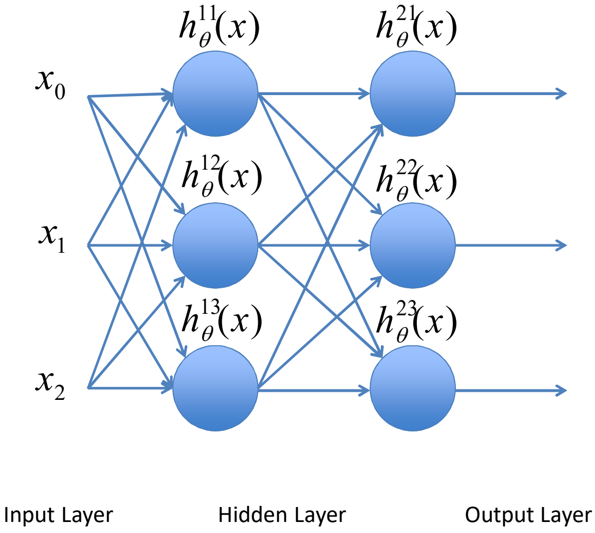
A Neural Network



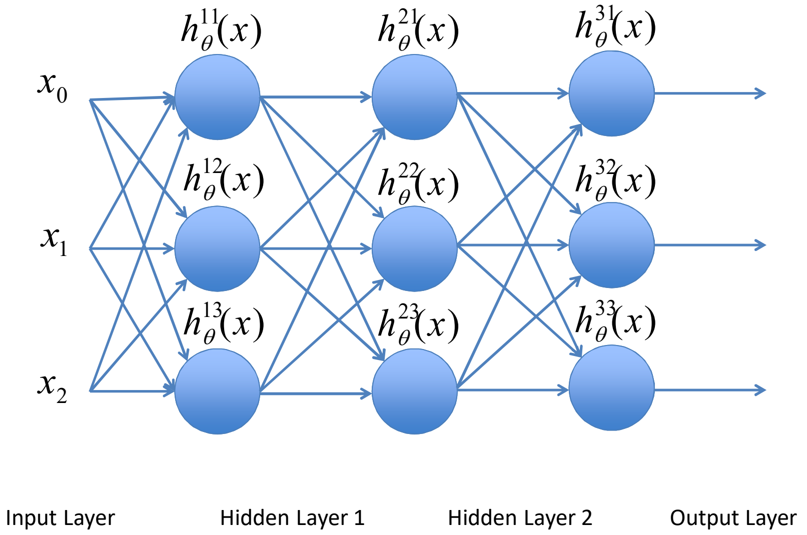
A Neural Network



A Multi-Layer Neural Network



A Deep Neural Network



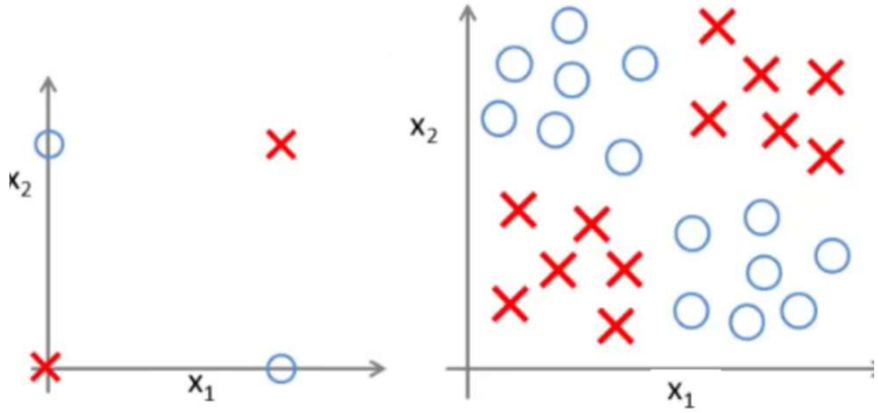
Summary:

- Neural nets as interconnected layers of (e.g., sigmoid / logistic regression) neurons/units.
- Layer l+1's inputs are layer l's outputs.
- Can have any number of layers.

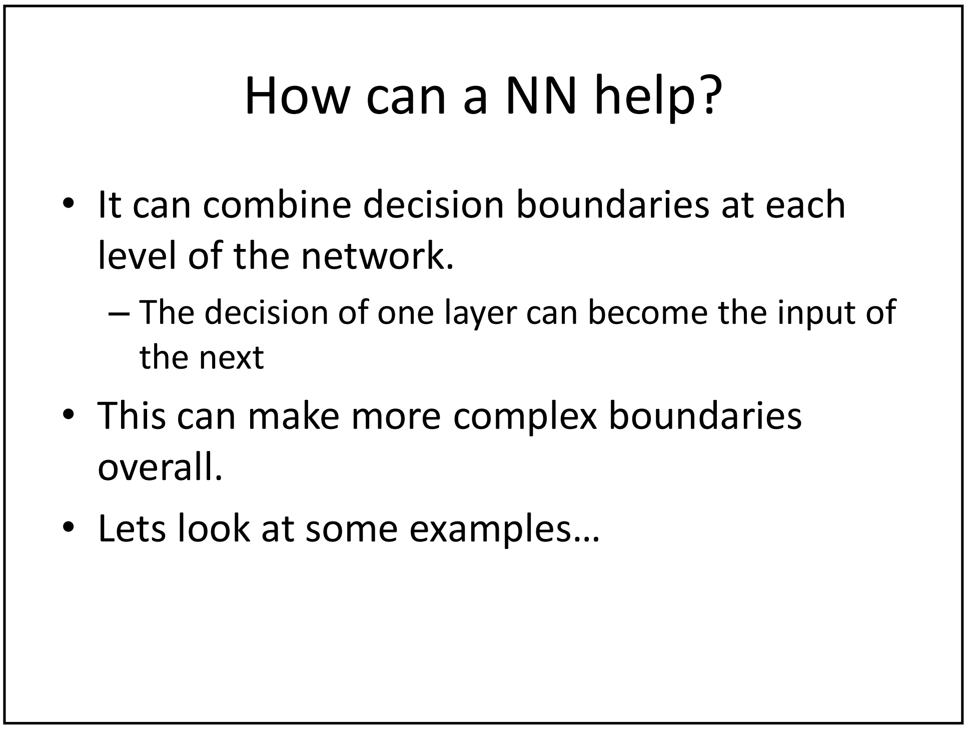
Overview

- Neural Nets Intro
- From Neurons to Neural Nets
- Multi-Layer Neural Nets for Non-Linear Functions
- Neural Net Prediction Details
- Training Neural Nets
- More considerations

Linearly Non-Separable Problems
E.g., the famous XOR problem

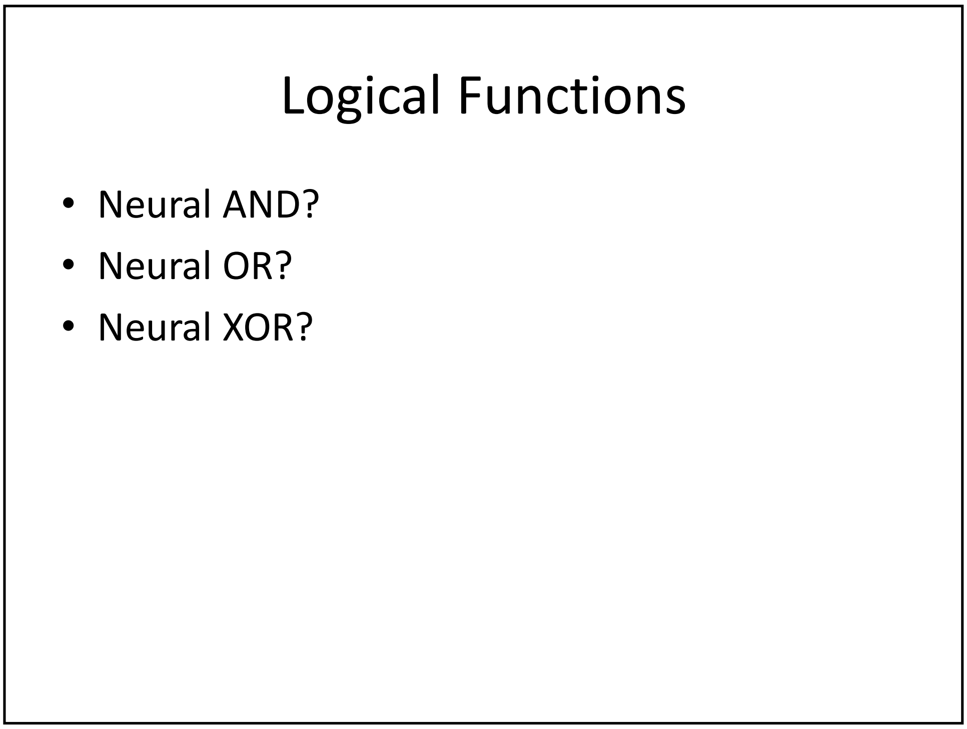


Can you draw a linear decision boundary to separate the two classes?



How can a NN help?

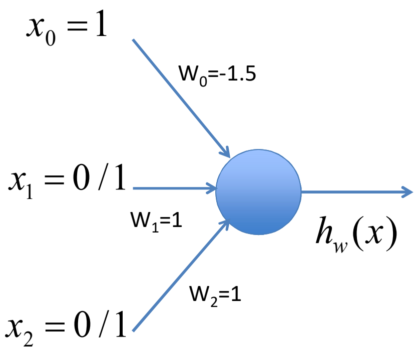
- It can combine decision boundaries at each level of the network.
  – The decision of one layer can become the input of the next
- This can make more complex boundaries overall.
- Lets look at some examples…



Logical Functions

- Neural AND?
- Neural OR?
- Neural XOR?

Logistic Unit AND

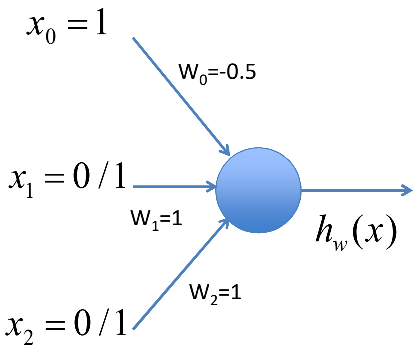


A	B	A AND B
0	0	<0
0	1	<0
1	0	<0
1	1	>0 (class 1)

What weights will give AND?

Recall $H_w(x)$=1 if $w_1x_1+w_2x_2+w_0>0$
Otherwise $H_w(x)$=0

Logistic Unit OR

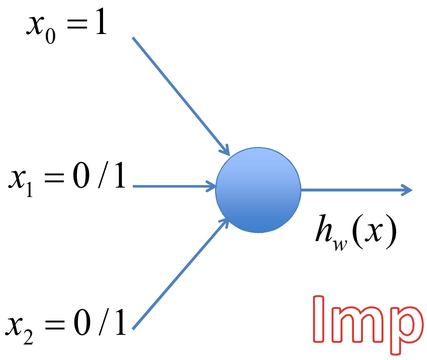


A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

What weights will give OR?

Recall $H_w(x)$=1 if $w_1x_1+w_2x_2+w_0>0$
Otherwise $H_w(x)$=0

Logistic Unit XOR?



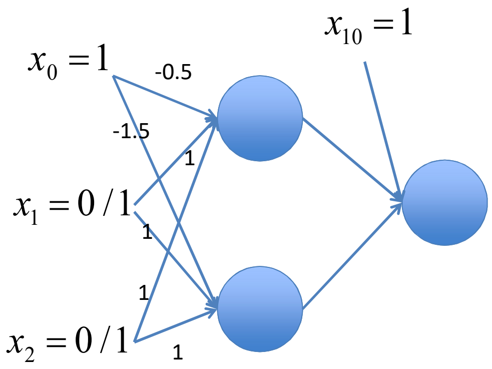
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Impossible!

What weights will give XOR?

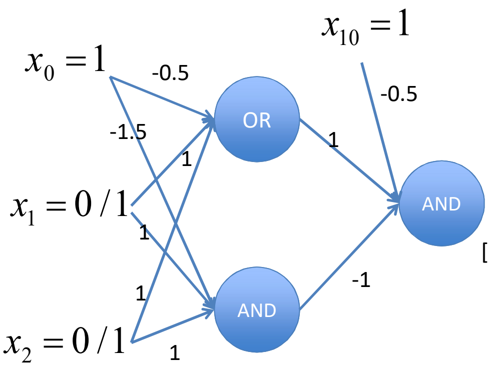
Recall $H_w(x)=1$ if $w_1x_1+w_2x_2+w_0>0$
Otherwise $H_w(x)=0$

Neural Net XOR



A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

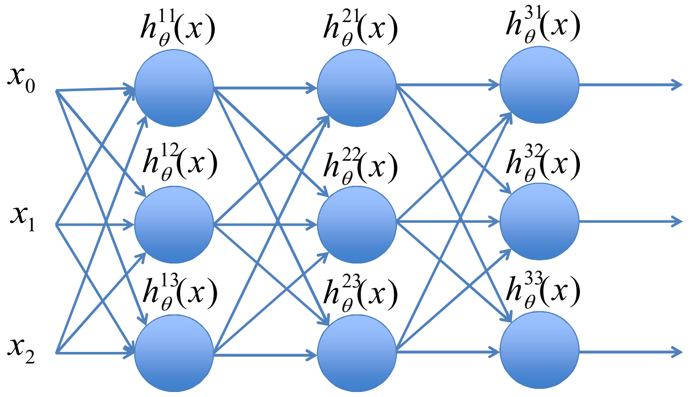
Neural Net XOR



[OR] AND [NOT AND]

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

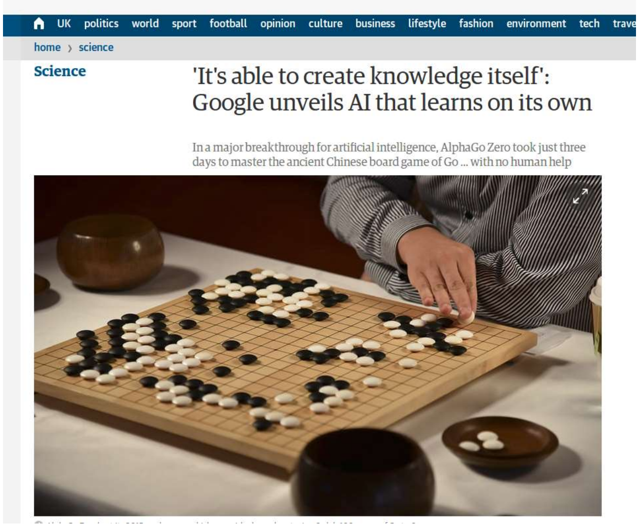
How to Come up With these Weights?



Input Layer Hidden Layer 1 Hidden Layer 2 Output Layer

Backpropagation Algorithm. Coming up next…

More Deepmind



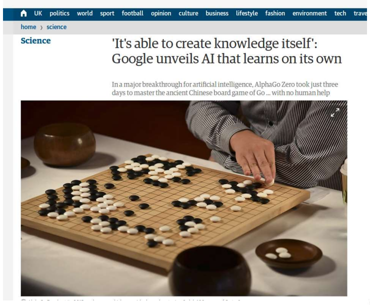
home › science

Science

'It's able to create knowledge itself': Google unveils AI that learns on its own

In a major breakthrough for artificial intelligence, AlphaGo Zero took just three days to master the ancient Chinese board game of Go ... with no human help

More Deepmind



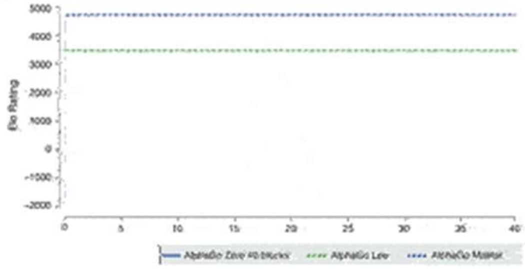
UK politics world sport football opinion culture business lifestyle fashion environment tech trave

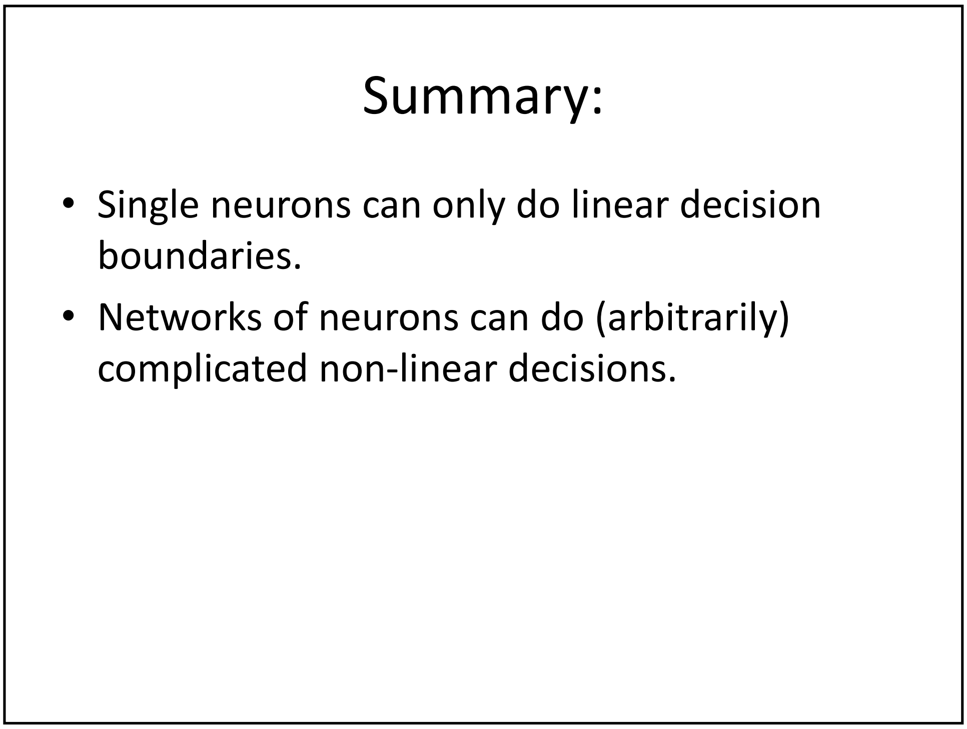
home › science

Science

'It's able to create knowledge itself': Google unveils AI that learns on its own

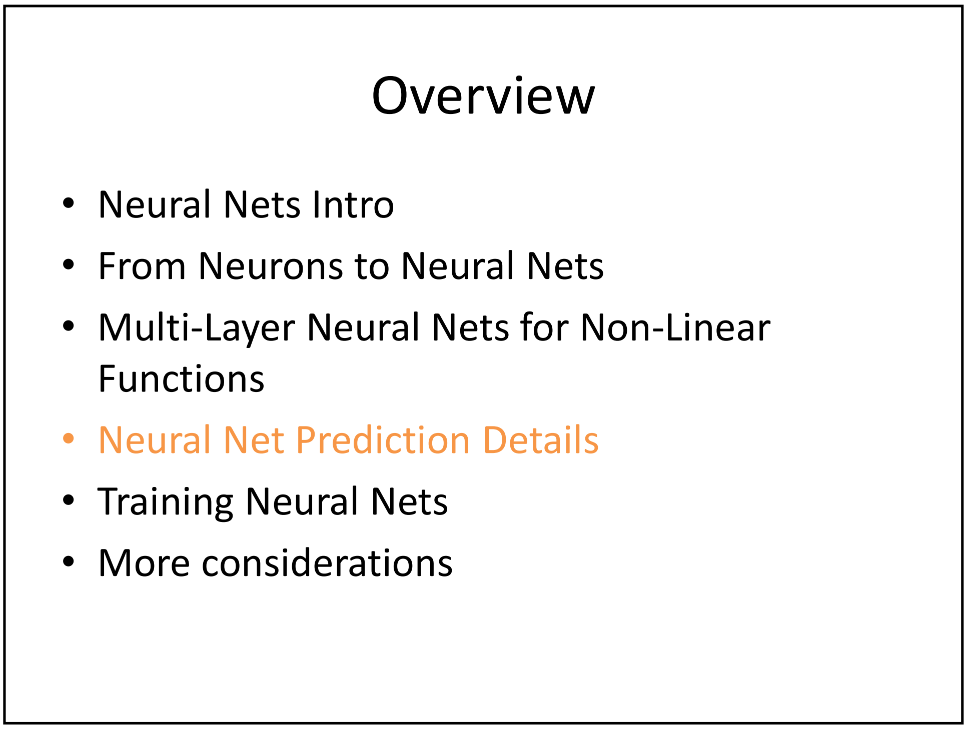
In a major breakthrough for artificial intelligence, AlphaGo Zero took just three days to master the ancient Chinese board game of Go ... with no human help





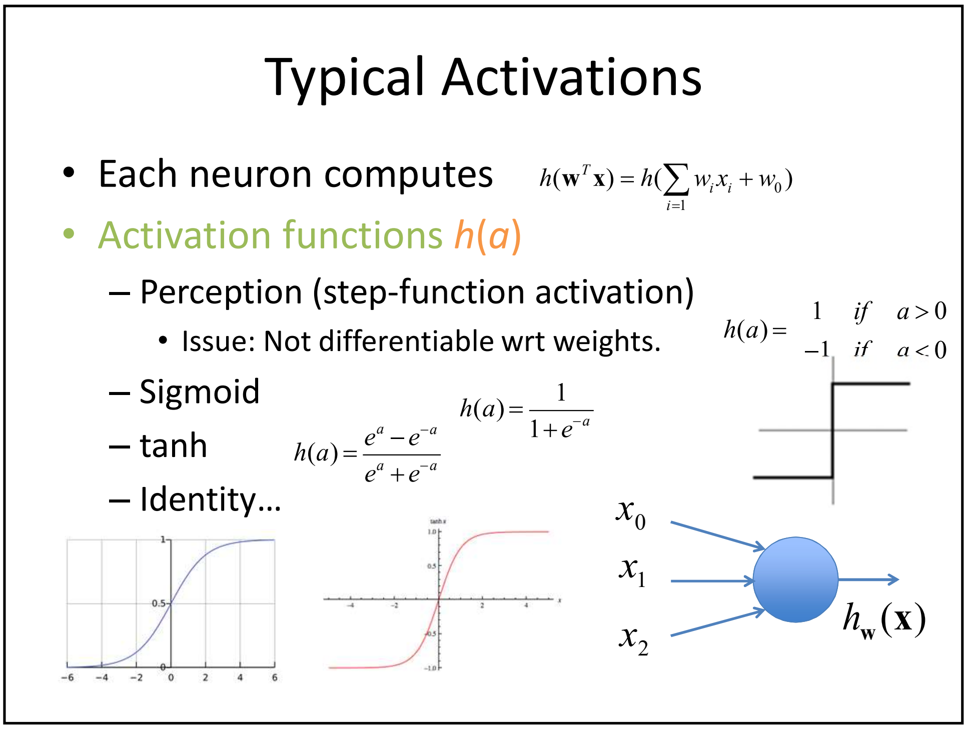
Summary:

- Single neurons can only do linear decision boundaries.
- Networks of neurons can do (arbitrarily) complicated non-linear decisions.



Overview

- Neural Nets Intro
- From Neurons to Neural Nets
- Multi-Layer Neural Nets for Non-Linear Functions
- Neural Net Prediction Details
- Training Neural Nets
- More considerations



Typical Activations

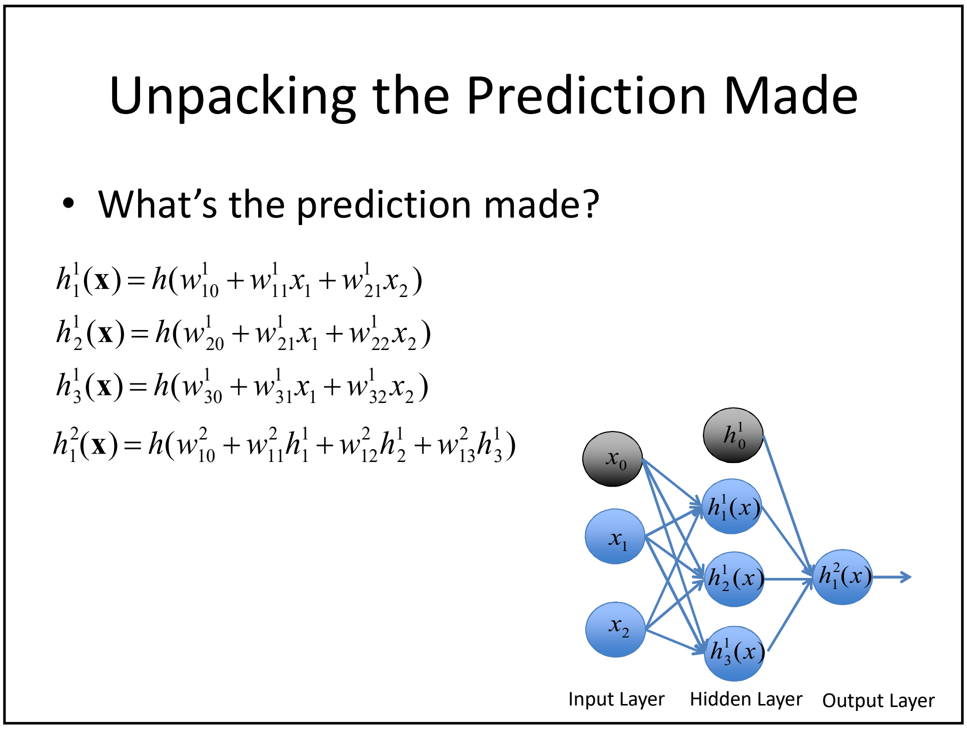
- Each neuron computes $h(\mathbf{w}^T\mathbf{x}) = h(\sum_{i=1} w_i x_i + w_0)$
- Activation functions *h*(*a*)
  - Perception (step-function activation)
    - Issue: Not differentiable wrt weights.
  - Sigmoid
  - tanh
  - Identity…

$$h(a) = \begin{matrix} 1 & if & a>0 \\ -1 & if & a<0 \end{matrix}$$

$$h(a) = \frac{1}{1+e^{-a}}$$

$$h(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$$

1
0.5
0
−6
−4
−2
0
2
4
6
tanh x
1.0
0.5
−0.5
−1.0
−4
−2
2
4
x
x_0
x_1
x_2
$h_{\mathbf{w}}(\mathbf{x})$



Unpacking the Prediction Made

- What's the prediction made?

$$h_1^1(\mathbf{x}) = h(w_{10}^1 + w_{11}^1 x_1 + w_{21}^1 x_2)$$
$$h_2^1(\mathbf{x}) = h(w_{20}^1 + w_{21}^1 x_1 + w_{22}^1 x_2)$$
$$h_3^1(\mathbf{x}) = h(w_{30}^1 + w_{31}^1 x_1 + w_{32}^1 x_2)$$
$$h_1^2(\mathbf{x}) = h(w_{10}^2 + w_{11}^2 h_1^1 + w_{12}^2 h_2^1 + w_{13}^2 h_3^1)$$

x_0
x_1
x_2
h_0^1
$h_1^1(x)$
$h_2^1(x)$
$h_3^1(x)$
$h_1^2(x)$
Input Layer
Hidden Layer
Output Layer

Unpacking the Prediction Made

- What’s the prediction made?

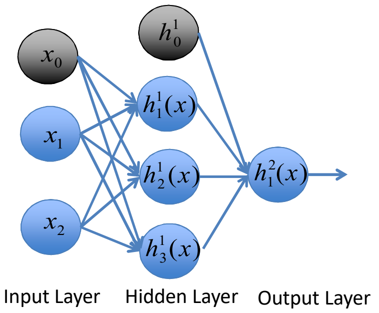
$$h_1^1(\mathbf{x}) = h(w_{10}^1 + w_{11}^1 x_1 + w_{21}^1 x_2)$$

$$h_2^1(\mathbf{x}) = h(w_{20}^1 + w_{21}^1 x_1 + w_{22}^1 x_2)$$

$$h_3^1(\mathbf{x}) = h(w_{30}^1 + w_{31}^1 x_1 + w_{32}^1 x_2)$$

$$h_1^2(\mathbf{x}) = h(w_{10}^2 + w_{11}^2 h_1^1 + w_{12}^2 h_2^1 + w_{13}^2 h_3^1)$$

$$h^2{}_k(\mathbf{x}) = h_k\left(\sum_{j=1}^{M} w^{(2)}{}_{kj} \cdot h_j\big(\sum_{i=1}^{D} w^{(1)}{}_{ji} x_i\big)\right)$$



Unpacking the Prediction Made

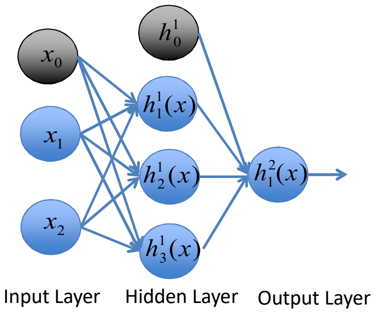
- What’s the prediction made?

$$h_1^1(\mathbf{x}) = h(w_{10}^1 + w_{11}^1 x_1 + w_{21}^1 x_2)$$

$$h_2^1(\mathbf{x}) = h(w_{20}^1 + w_{21}^1 x_1 + w_{22}^1 x_2)$$

$$h_3^1(\mathbf{x}) = h(w_{30}^1 + w_{31}^1 x_1 + w_{32}^1 x_2)$$

$$h_1^2(\mathbf{x}) = h(w_{10}^2 + w_{11}^2 h_1^1 + w_{12}^2 h_2^1 + w_{13}^2 h_3^1)$$



$$\mathbf{h}^2(\mathbf{x}) = \mathbf{h}\left(\mathbf{W}^{(2)} \cdot \mathbf{h}(\mathbf{W}^{(1)}\ \mathbf{x})\right)$$

Aside: Non-linearities are important.

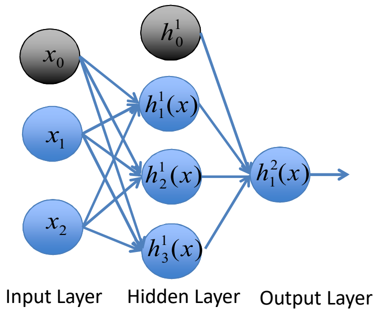
- What's the prediction made?
  - Suppose we used linear activation h. $h(a)=a$
    - What happens to the prediction?

$$\mathbf{h}^2(\mathbf{x}) = \mathbf{h}\left(\mathbf{W}^{(2)} \cdot \mathbf{h}(\mathbf{W}^{(1)} \mathbf{x})\right)$$

$$\mathbf{h}^2(\mathbf{x}) = \mathbf{W}^{(2)}\mathbf{W}^{(1)} \mathbf{x}$$

$$\mathbf{h}^2(\mathbf{x}) = \mathbf{W}'\mathbf{x}$$

- No matter how many layers….
  - Still a simple linear model



Neural Nets can do many tasks

- Single output regression
- Multiple output regression
- Binary classification.
- Multiclass classification.
- Binary multi-label classification.

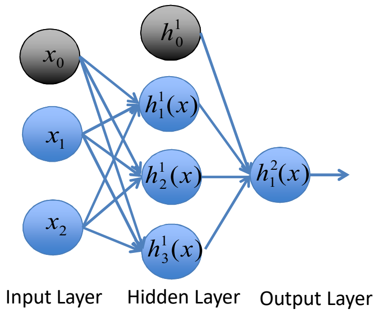
- … by changing the activation function of output layer.

Neural Nets can do many tasks: Single output regression

- Single output regression
  - Identity activation $h^2(a) = a$
    - Linear combination of previous layer is range +/- inf.

$$h_1^1(\mathbf{x}) = h(w_{10}^1 + w_{11}^1 x_1 + w_{21}^1 x_2)$$
$$h_2^1(\mathbf{x}) = h(w_{20}^1 + w_{21}^1 x_1 + w_{22}^1 x_2)$$
$$h_3^1(\mathbf{x}) = h(w_{30}^1 + w_{31}^1 x_1 + w_{32}^1 x_2)$$

$$h_1^2(\mathbf{x}) = w_{10}^2 + w_{11}^2 h_1^1 + w_{12}^2 h_2^1 + w_{13}^2 h_3^1$$

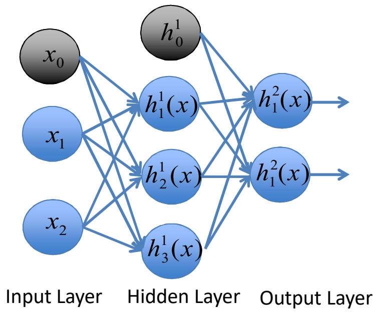


Neural Nets can do many tasks: Multiple output regression

- Multiple output regression
  - Identity activation $h^2(a) = a$
    - Linear combination of previous layer is range +/- inf.
  - Use multiple output units.

$$h_1^2(\mathbf{x}) = w_{10}^2 + w_{11}^2 h_1^1 + w_{12}^2 h_2^1 + w_{13}^2 h_3^1$$

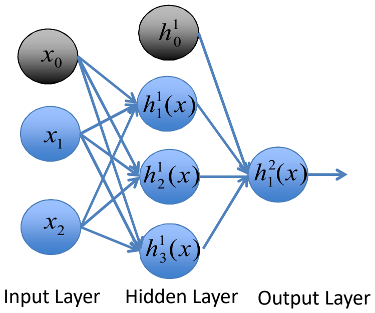
$$h_2^2(\mathbf{x}) = w_{20}^2 + w_{21}^2 h_1^1 + w_{22}^2 h_2^1 + w_{23}^2 h_3^1$$



Neural Nets can do many tasks: Binary classification

- Binary classification
  - – Sigmoid activation $h^2(a) = 1/(1+e^{-a})$
    - Linear combination of previous layer squashed into range [0,1].

$$h_1^2(\mathbf{x}) = \sigma(w_{10}^2 + w_{11}^2 h_1^1 + w_{12}^2 h_2^1 + w_{13}^2 h_3^1)$$



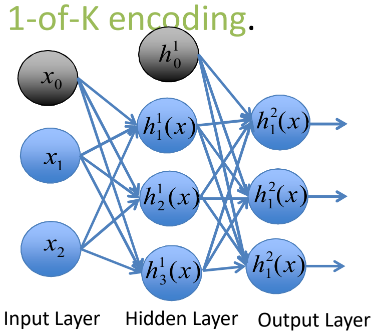
Neural Nets can do many tasks: Multiclass/Multi-label classification

- Multiclass classification
  - – Sigmoid activation $h^2(a) = 1/(1+e^{-a})$
    - Linear combination of previous layer squashed into range [0,1]
  - – Use multiple output units in 1-of-K encoding.

$$h_1^2(\mathbf{x}) = \sigma(w_{10}^2 + w_{11}^2 h_1^1 + w_{12}^2 h_2^1 + w_{13}^2 h_3^1)$$

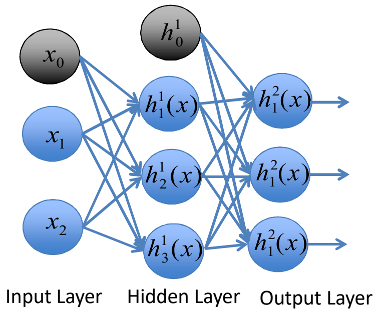
$$h_2^2(\mathbf{x}) = \sigma(w_{20}^2 + w_{21}^2 h_1^1 + w_{22}^2 h_2^1 + w_{23}^2 h_3^1)$$

$$h_3^2(\mathbf{x}) = \sigma(w_{30}^2 + w_{31}^2 h_1^1 + w_{32}^2 h_2^1 + w_{33}^2 h_3^1)$$



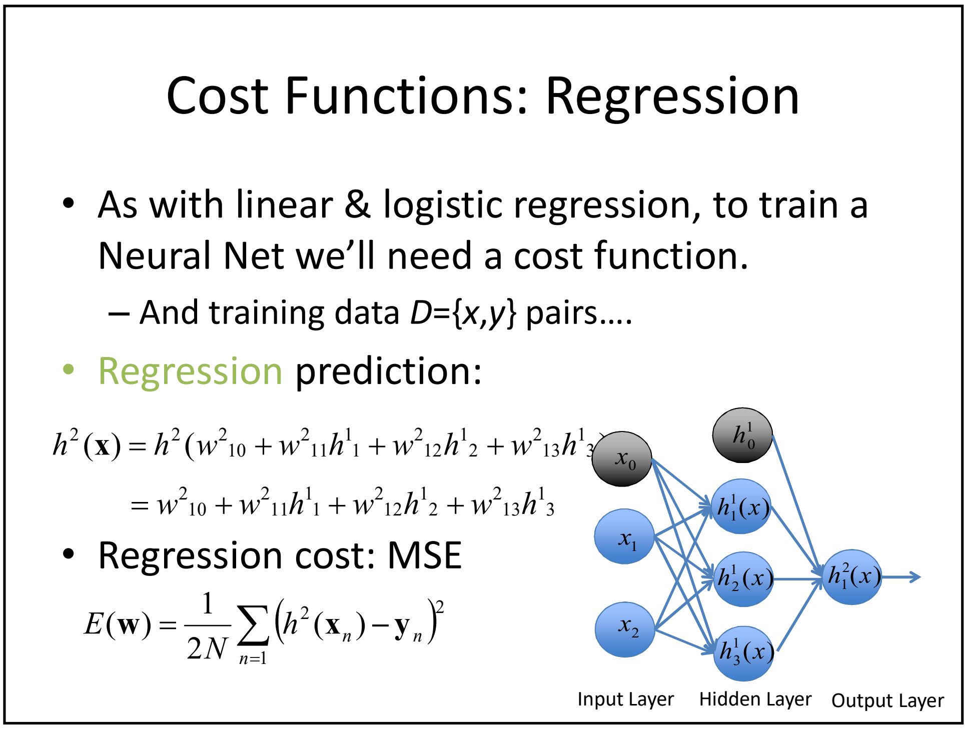
Neural Nets can do many tasks: Multiclass/Multi-label classification

- Multiclass classification
  - Sigmoid activation $h^2(a) = 1/(1+e^{-a})$
    - Linear combination of previous layer squashed into range [0,1]
  - Use multiple output units in 1-of-K encoding.
- I.e., Train set would be
  - y=[1,0,0] : person
  - y=[0,1,0] : car
  - y=[0,0,1] : truck



Overview

- Neural Nets Intro
- From Neurons to Neural Nets
- Multi-Layer Neural Nets for Non-Linear Functions
- Neural Net Prediction Details
- Training Neural Nets
- More considerations



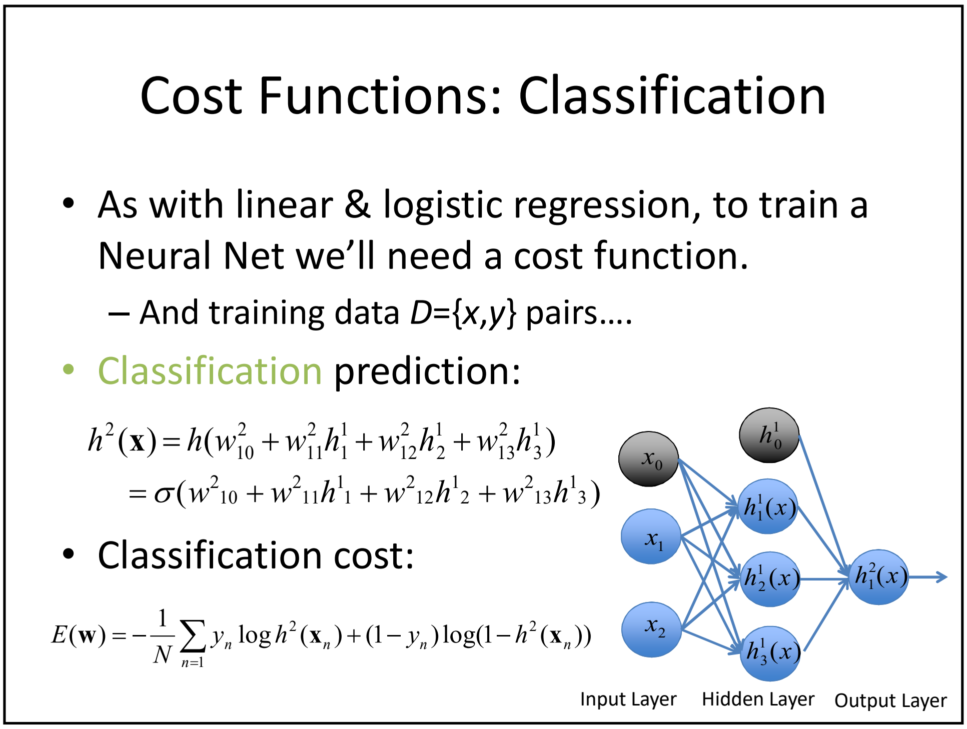
Cost Functions: Regression

- As with linear & logistic regression, to train a Neural Net we'll need a cost function.
  - And training data *D*={*x*,*y*} pairs….
- Regression prediction:

$$h^2(\mathbf{x}) = h^2(w^2{}_{10} + w^2{}_{11}h^1{}_1 + w^2{}_{12}h^1{}_2 + w^2{}_{13}h^1{}_3)$$
$$= w^2{}_{10} + w^2{}_{11}h^1{}_1 + w^2{}_{12}h^1{}_2 + w^2{}_{13}h^1{}_3$$

- Regression cost: MSE

$$E(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}\left(h^2(\mathbf{x}_n) - \mathbf{y}_n\right)^2$$



Cost Functions: Classification

- As with linear & logistic regression, to train a Neural Net we'll need a cost function.
  - And training data *D*={*x*,*y*} pairs….
- Classification prediction:

$$h^2(\mathbf{x}) = h(w_{10}^2 + w_{11}^2 h_1^1 + w_{12}^2 h_2^1 + w_{13}^2 h_3^1)$$
$$= \sigma(w^2{}_{10} + w^2{}_{11}h^1{}_1 + w^2{}_{12}h^1{}_2 + w^2{}_{13}h^1{}_3)$$

- Classification cost:

$$E(\mathbf{w}) = -\frac{1}{N}\sum_{n=1} y_n \log h^2(\mathbf{x}_n) + (1 - y_n)\log(1 - h^2(\mathbf{x}_n))$$

Computing the Cost

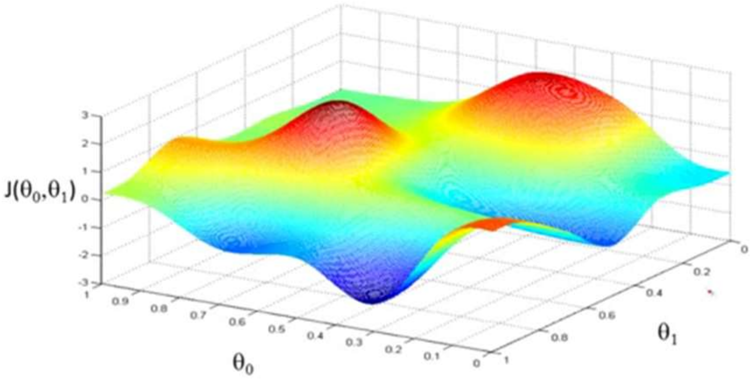
- To train NN, find weights to minimise the cost.
- We know how to write code to compute E(**w**)
  - 1. Do the "forward propagation" and make predictions.
  - 2. Use predictions to compute cost.
- …. But how to minimise it efficiently?

$$h^2(\mathbf{x}) = \mathbf{w}^2 h^1(\mathbf{W}^1\mathbf{x}) \qquad h^2(\mathbf{x}) = \sigma(\mathbf{w}^2\ h^1(\mathbf{W}^1\mathbf{x}))$$

$$E(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}\left(h^2(\mathbf{x}_n) - \mathbf{y}_n\right)^2 \qquad E(\mathbf{w}) = -\frac{1}{N}\sum_{n=1} y_n \log h^2(\mathbf{x}_n) + (1-y_n)\log(1-h^2(\mathbf{x}_n))$$

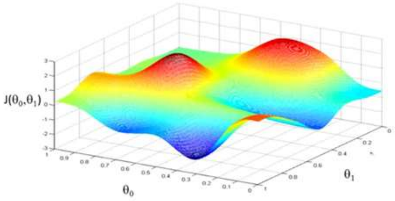
Minimising the Cost

- Cost is Non-Convex



Neural Net Cost is Non Convex

- Cost is Non-Convex:
  - No global minima.
  - Require iterative gradient methods to train.
  - Gradient descent will only converge to local minima.



How to Optimise Cost?

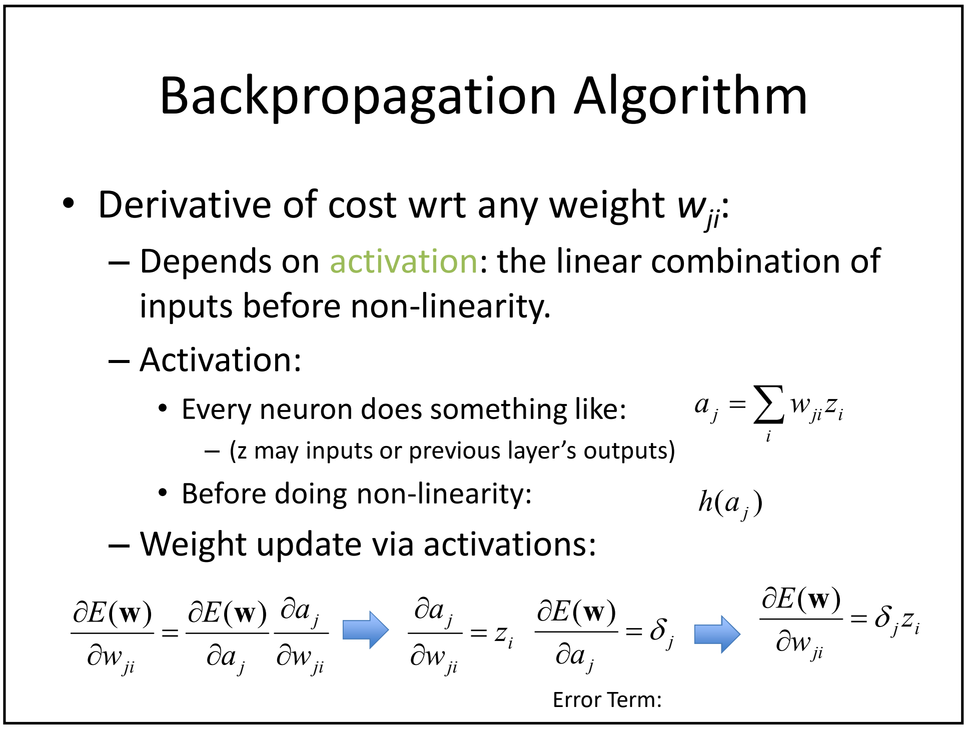
- For Linear/Logistic regression, each weight had a straightforward derivative.
- For neural net, complicated by many layers of interdependent weights.

$$h^2(\mathbf{x}) = \mathbf{w}^2 h^1(\mathbf{W}^1\mathbf{x})$$

$$h^2(\mathbf{x}) = \sigma(\mathbf{w}^2\ h^1(\mathbf{W}^1\mathbf{x}))$$

$$E(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}\left(h^2(\mathbf{x}_n) - \mathbf{y}_n\right)^2$$

$$E(\mathbf{w}) = -\frac{1}{N}\sum_{n=1} y_n \log h^2(\mathbf{x}_n) + (1-y_n)\log(1-h^2(\mathbf{x}_n))$$

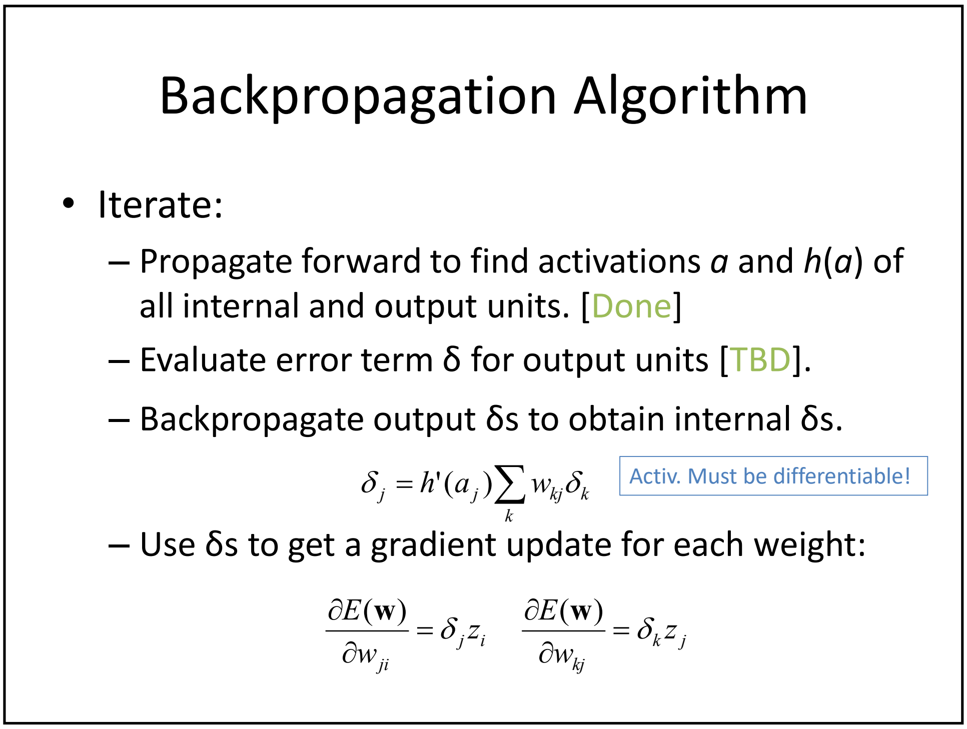


Backpropagation Algorithm

- Derivative of cost wrt any weight w_{ji}:
  - Depends on activation: the linear combination of inputs before non-linearity.
  - Activation:
    - Every neuron does something like: $a_j = \sum_i w_{ji} z_i$
      - (z may inputs or previous layer's outputs)
    - Before doing non-linearity: $h(a_j)$
  - Weight update via activations:

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \frac{\partial E(\mathbf{w})}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \Rightarrow \frac{\partial a_j}{\partial w_{ji}} = z_i \quad \frac{\partial E(\mathbf{w})}{\partial a_j} = \delta_j \Rightarrow \frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \delta_j z_i$$

Error Term:



Backpropagation Algorithm

- Iterate:
  - Propagate forward to find activations *a* and *h*(*a*) of all internal and output units. [Done]
  - Evaluate error term δ for output units [TBD].
  - Backpropagate output δs to obtain internal δs.

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$

Activ. Must be differentiable!

  - Use δs to get a gradient update for each weight:

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \delta_j z_i \qquad \frac{\partial E(\mathbf{w})}{\partial w_{kj}} = \delta_k z_j$$

Output Error Term: Regression

- Output error term: $\frac{\partial E(\mathbf{w})}{\partial a_k} = \delta_k$
- For linear regression:
  - Cost: Square Deviation
  - Activation: Identity.

$$E_n(\mathbf{w}) = 0.5\left(h(a_j) - y\right)^2 \qquad \frac{\partial E_n(\mathbf{w})}{\partial a_k} = h'(a_k)\left(h(a_k) - y\right)$$

$$h(a_k) = a_k \quad h'(a_k) = 1 \qquad \frac{\partial E_n(\mathbf{w})}{\partial a_k} = \delta_k = \left(h(a_k) - y\right)$$

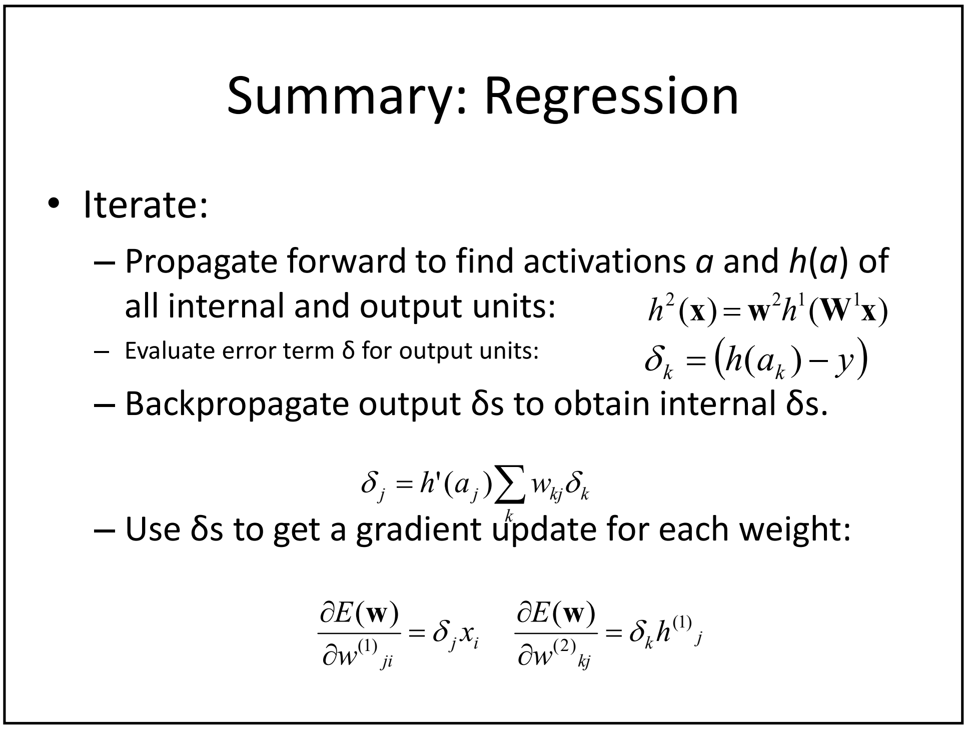
Output Error Term: Binary Classifier

- Output error term: $\frac{\partial E(\mathbf{w})}{\partial a_k} = \delta_k$
- For binary classifier
  - Cost: Square Devation (or cross-entropy)
  - Activation: Sigmoid.

$$E_n(\mathbf{w}) = 0.5\left(h(a_k) - y\right)^2 \qquad \frac{\partial E_n(\mathbf{w})}{\partial a_k} = h'(a_k)\left(h(a_k) - y\right)$$

$$h(a) = \sigma(a) = 1/(1 + e^{-a})$$

$$h'(a) = \sigma'(a) = \sigma(a)(1 - \sigma(a)) \qquad \frac{\partial E_n(\mathbf{w})}{\partial a_k} = \sigma'(a_k)\left(\sigma(a_k) - y\right)$$



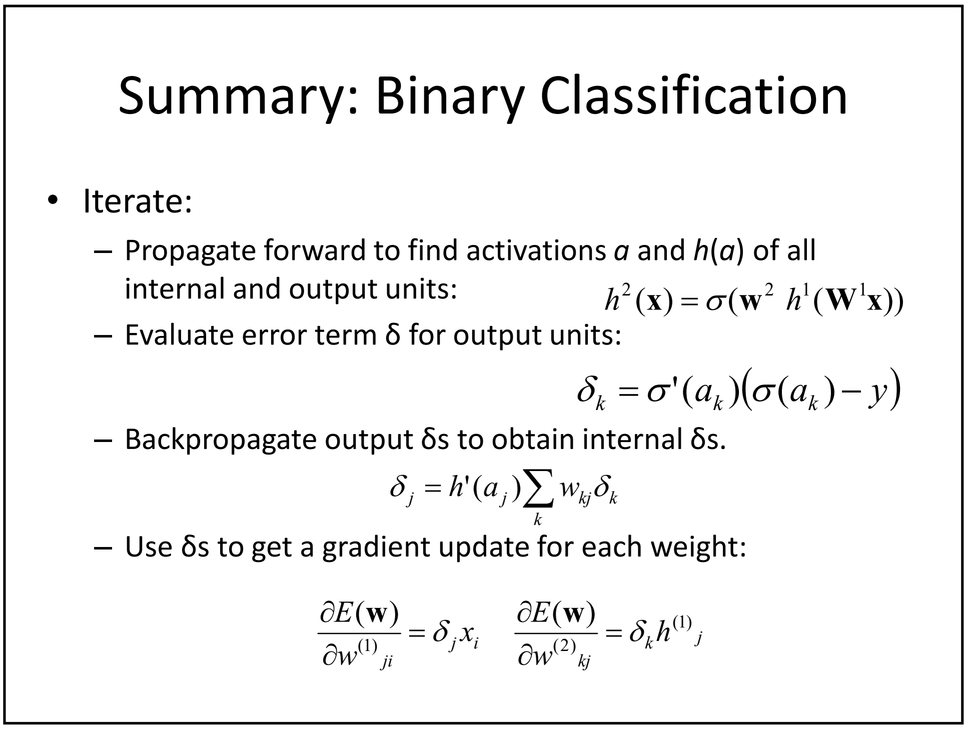
Summary: Regression

- Iterate:
  - Propagate forward to find activations *a* and *h*(*a*) of all internal and output units: $h^2(\mathbf{x}) = \mathbf{w}^2 h^1(\mathbf{W}^1\mathbf{x})$
  - Evaluate error term δ for output units: $\delta_k = \left(h(a_k) - y\right)$
  - Backpropagate output δs to obtain internal δs.

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$

  - Use δs to get a gradient update for each weight:

$$\frac{\partial E(\mathbf{w})}{\partial w^{(1)}{}_{ji}} = \delta_j x_i \quad \frac{\partial E(\mathbf{w})}{\partial w^{(2)}{}_{kj}} = \delta_k h^{(1)}{}_j$$



Summary: Binary Classification

- Iterate:
  - Propagate forward to find activations *a* and *h*(*a*) of all internal and output units: $h^2(\mathbf{x}) = \sigma(\mathbf{w}^2\ h^1(\mathbf{W}^1\mathbf{x}))$
  - Evaluate error term δ for output units:

$$\delta_k = \sigma'(a_k)\left(\sigma(a_k) - y\right)$$

  - Backpropagate output δs to obtain internal δs.

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$

  - Use δs to get a gradient update for each weight:

$$\frac{\partial E(\mathbf{w})}{\partial w^{(1)}{}_{ji}} = \delta_j x_i \quad \frac{\partial E(\mathbf{w})}{\partial w^{(2)}{}_{kj}} = \delta_k h^{(1)}{}_j$$

Training Overview

- Implement forward propagation to get h(x) for any x.
- Implement cost function computation.
- Implement backpropagation to compute partial derivatives.
- Iterate forward & backward propagation.

Summary

- Training neural nets:
  - Use backpropagation to minimise their cost function.
  - Errors at later nodes are backpropagated to compute errors at earlier nodes.
  - Errors at each node give gradient update for that node.

Overview

- Neural Nets Intro
- From Neurons to Neural Nets
- Multi-Layer Neural Nets for Non-Linear Functions
- Neural Net Prediction Details
- Training Neural Nets
- More considerations

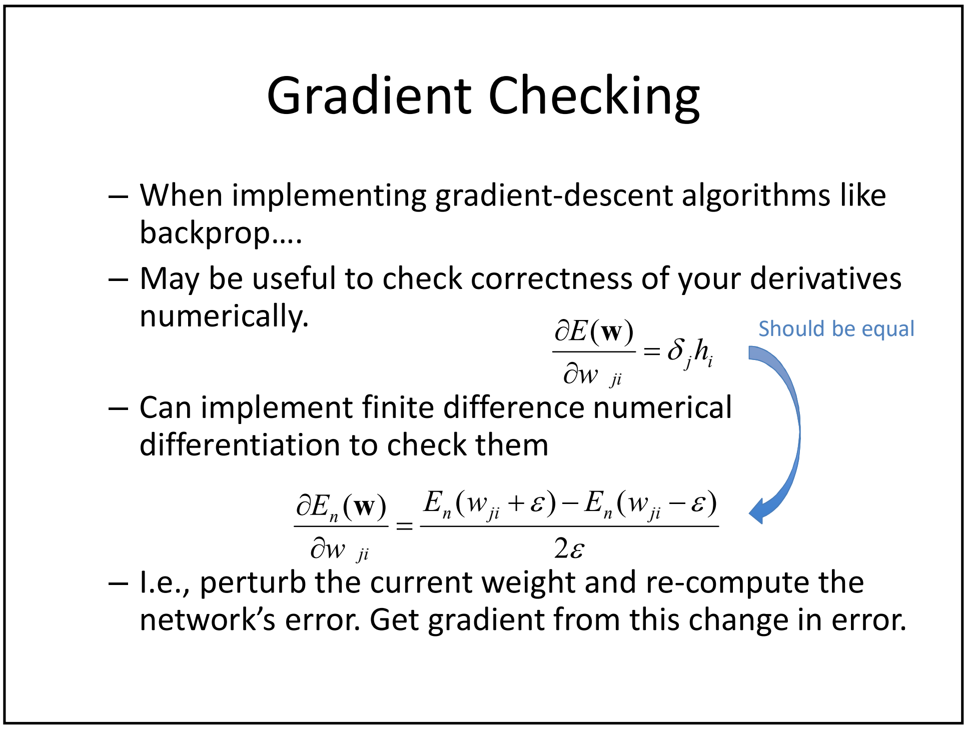
Batch vs Online

- Updates in past couple slides are gradients WRT one single input *n*.
  - Online Gradient Descent:
    - Iterate over data *n*, and weights *w*(*i*,*j*):

$$w_{ji} = w_{ji} - \alpha \frac{\partial E_n(\mathbf{w})}{\partial w_{ji}}$$

  - Batch Gradient Descent:
    - Iterate over weights *w*(*i*,*j*):

$$w_{ji} = w_{ji} - \alpha \sum_n \frac{\partial E_n(\mathbf{w})}{\partial w_{ji}}$$



Gradient Checking

- When implementing gradient-descent algorithms like backprop….
- May be useful to check correctness of your derivatives numerically.

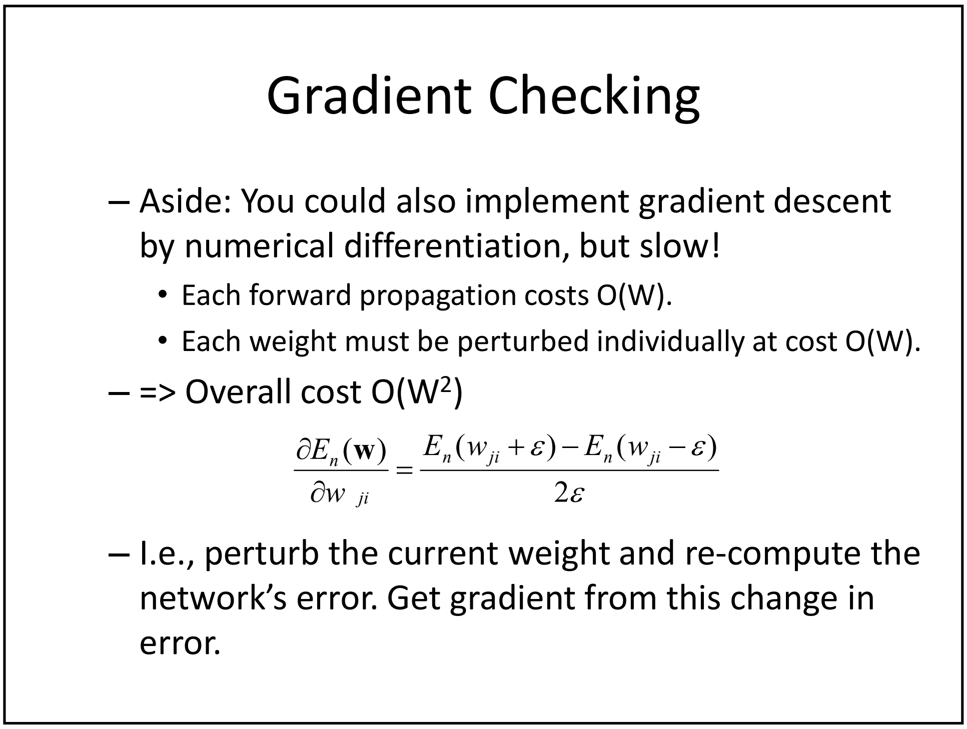
$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \delta_j h_i$$

Should be equal

- Can implement finite difference numerical differentiation to check them

$$\frac{\partial E_n(\mathbf{w})}{\partial w_{ji}} = \frac{E_n(w_{ji}+\varepsilon)-E_n(w_{ji}-\varepsilon)}{2\varepsilon}$$

- I.e., perturb the current weight and re-compute the network's error. Get gradient from this change in error.



Gradient Checking

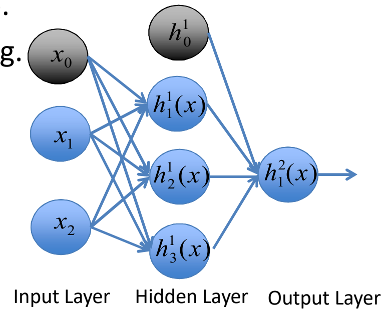
- Aside: You could also implement gradient descent by numerical differentiation, but slow!
  - Each forward propagation costs O(W).
  - Each weight must be perturbed individually at cost O(W).
- => Overall cost O(W^2)

$$\frac{\partial E_n(\mathbf{w})}{\partial w_{ji}} = \frac{E_n(w_{ji}+\varepsilon)-E_n(w_{ji}-\varepsilon)}{2\varepsilon}$$

- I.e., perturb the current weight and re-compute the network's error. Get gradient from this change in error.

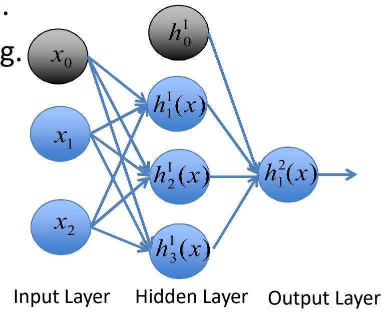
Initialization

- For gradient methods need to pick initial weight vectors ***w***.
  - Zero initialization:
    - Every hidden unit gets the same input.
    - Nothing different to backprop.
    - Network never learns anything.



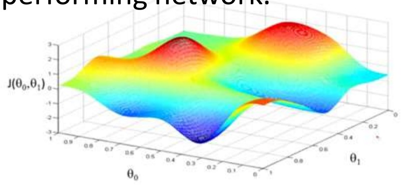
Initialization => Symmetry Breaking

- For gradient methods need to pick initial weight vectors ***w***.
  - Zero initialization:
    - Every hidden unit gets the same input.
    - Nothing different to backprop.
    - Network never learns anything.
- Solution:
  - Init randomly in [-ε,ε]



Local Minima

- Neural Net costs are not Convex
- Solutions:
  1. Accept local minima.
  2. Online rather than batch gradient descent may help jitter out of minima.
  3. Repeatedly restart from different random initial conditions, take the best performing network. (Expensive)
  4. …momentum, etc.



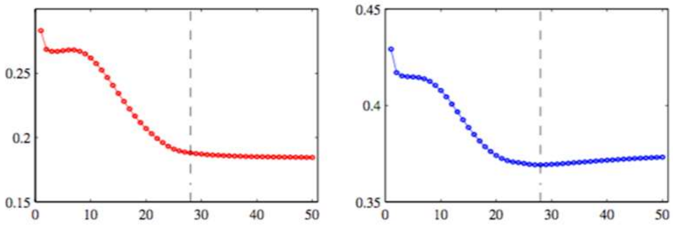
Overfitting in Neural Nets

- Neural Nets can have lots of parameters (weights)
  - 10s of millions in practice!
  - Overfitting is a risk, especially if limited data.
- Solutions: Regularization:
  - Use L2 regularizer/weight decay as we did with linear/logistic regression (needs update to gradients)

Overfitting in Neural Nets

- Neural Nets can have lots of parameters (weights)
  - 10s of millions in practice!
  - Overfitting is a risk, especially if limited data.
- Solutions: Early Stopping:
  - At each iteration of gradient descent, check the network cost on validation set.
  - Stop once validation error starts to increase (although train error will still be decreasing)

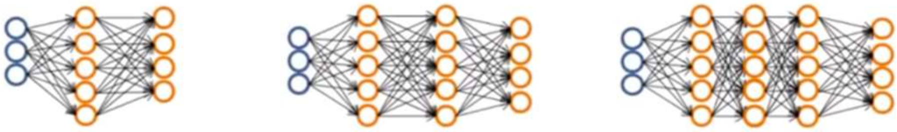
Overfitting in Neural Nets



- Solutions: Early Stopping:
  - At each iteration of gradient descent, check the network cost on validation set.
  - Stop once validation error starts to increase (although train error will still be decreasing)

NN: Design Decisions

- Network Architecture
  - # of Hidden Layers.
  - # of Hidden Nodes.
- (# of input & output relatively easy)



- Cost function choice
- Activation function choice
- Learning rate

Training Overview (2)

- Choose architecture
- Initialize weights to small random numbers
- Implement forward propagation to get h(x) for any x.
- Implement cost function computation.
- Implement backpropagation to compute partial derivatives.
- Iterate forward & backward propagation.
- Use gradient checking to debug.
- Disable gradient checking once debugged.
- Early Stop training once validation error increasing.
- (Repeatedly retrain to try and find different local minima)

Overview

- Neural Nets Intro
- From Neurons to Neural Nets
- Multi-Layer Neural Nets for Non-Linear Functions
- Neural Net Prediction Details
- Training Neural Nets
- More considerations
- Resurgence and Deep Neural Nets

Why were NNs out of favor?

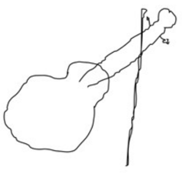
- Slow to train:
  - Vanilla gradient descent inefficient compared to convex optimisers (e.g., Kernel-SVM).
- Local minima & no better than alternatives:
  - Kernel-SVMs can do “equally good” non-linear classification and get global optimum quickly.
- Over-fitting:
  - Neural nets have lots of parameters.
  - They seemed easy to overfit, and tricky to regularize them.

Why are NNs back?

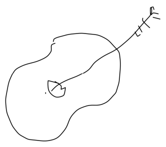
- Slow to train: Solved?
  - In ~2012, GPU computation used to speed up operations required for backprop x10-20 times (per GPU!) over CPU.
- Over-fitting: Solved?
  - “Big Data” in 2017 vast available databases can constrain vast number of parameters.
  - New regularization techniques: Dropout.
- Local minima: Doesn’t matter?
  - With other stuff taken care of, local minima are usually pretty good.
- Technical advances:
  - RELU activation.
  - Deep neural nets can be better than shallow. (wide as well)
- Feature learning
  - Can input “raw data” an learn the features.

NN Example - EECS Research ☺

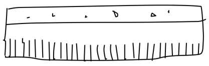
Let’s Play Pictionary!



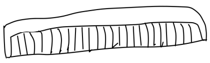
violin



guitar



piano

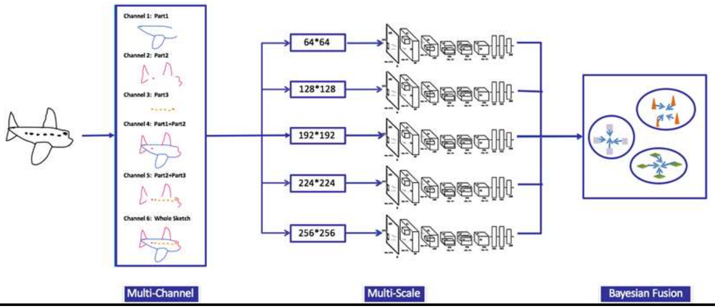


comb

NN Example - EECS Research ☺

Best Paper Prize: BMVC 2015!

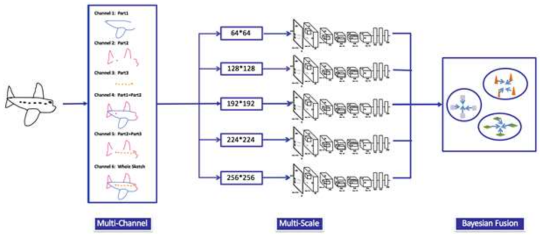
- Our AI system plays superhuman level of pictionary!
  - Key components: Neural Net => Metric Learning => Nearest Neighbor.



NN Example - EECS Research ☺

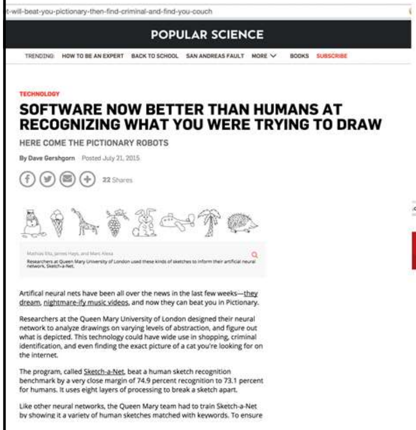
Best Paper Prize: BMVC 2015!

- Specs:
  - Eight layers (five convolutional).
  - “Only” 8.5 million parameters. (CF 144 mil).
  - Trained on 300M images with 32C Xenon CPU: 80 hours, or K40 GPU: 10 hours.



NN Example - EECS Research ☺

Reported in BBC, Popular Science, Business Insider, and many more….



t-will-beat-you-pictionary-then-find-criminal-and-find-you-couch

POPULAR SCIENCE

TRENDING: HOW TO BE AN EXPERT BACK TO SCHOOL SAN ANDREAS FAULT MORE BOOKS SUBSCRIBE

TECHNOLOGY

SOFTWARE NOW BETTER THAN HUMANS AT RECOGNIZING WHAT YOU WERE TRYING TO DRAW

HERE COME THE PICTIONARY ROBOTS

By Dave Gershgorn Posted July 21, 2015

22 Shares

Mathias Eitz, James Hays, and Marc Alexa

Researchers at Queen Mary University of London used these kinds of sketches to inform their artificial neural network, Sketch-a-Net.

Artifical neural nets have been all over the news in the last few weeks—they dream, nightmare-ify music videos, and now they can beat you in Pictionary.

Researchers at the Queen Mary University of London designed their neural network to analyze drawings on varying levels of abstraction, and figure out what is depicted. This technology could have wide use in shopping, criminal identification, and even finding the exact picture of a cat you're looking for on the internet.

The program, called Sketch-a-Net, beat a human sketch recognition benchmark by a very close margin of 74.9 percent recognition to 73.1 percent for humans. It uses eight layers of processing to break a sketch apart.

Like other neural networks, the Queen Mary team had to train Sketch-a-Net by showing it a variety of human sketches matched with keywords. To ensure

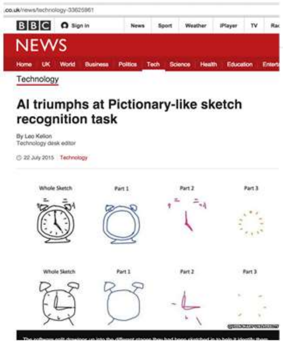
uk.businessinsider.com/queen-mary-neural-network-sketch-recognition-pictionary-2015-7

BUSINESS INSIDER UK TECH

Researchers have developed a programme that can identify these sketches better than a human

LUCY ENGLAND

Jul. 24, 2015, 1:52 PM 8,031



.co.uk/news/technology-33625961

BBC Sign in News Sport Weather iPlayer TV Ra

NEWS

Home UK World Business Politics Tech Science Health Education Entert

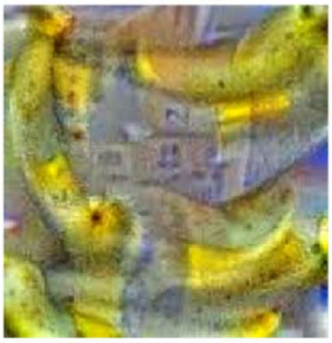
Technology

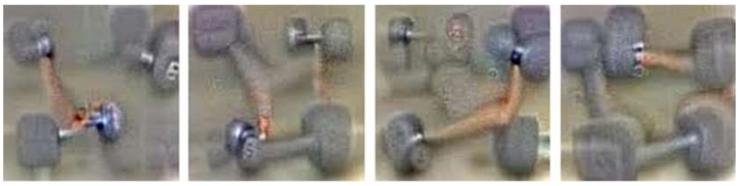
AI triumphs at Pictionary-like sketch recognition task

By Leo Kelion
Technology desk editor

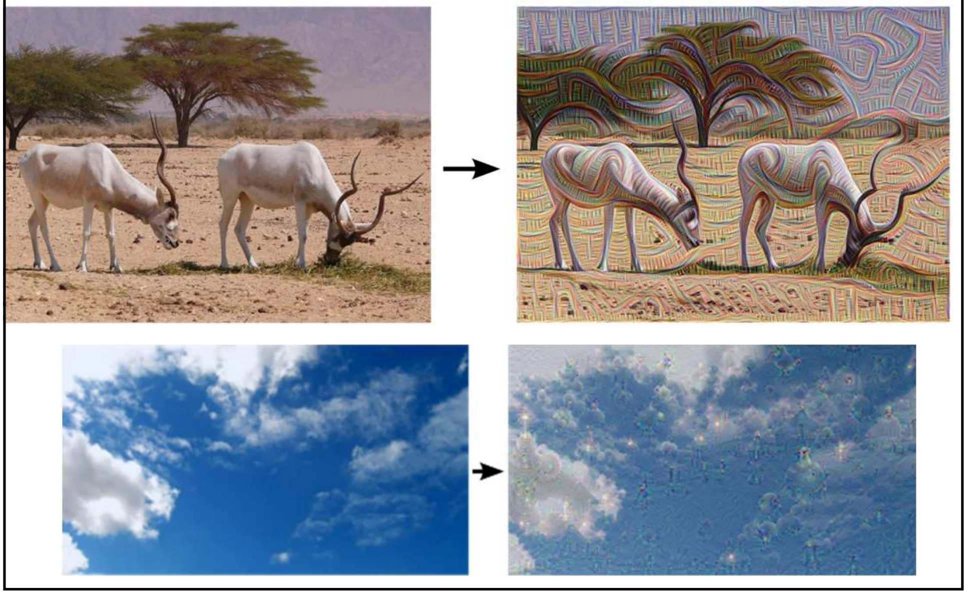
22 July 2015 Technology

Visualising Deep Neural Net Knowledge





How the network sees?



Looking for Patterns in Clouds

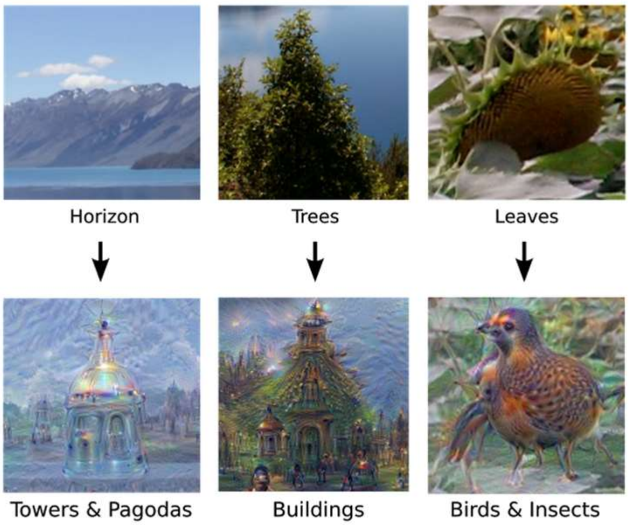
"Admiral Dog!"

"The Pig-Snail"

"The Camel-Bird"

"The Dog-Fish"

Looking for Patterns in Clouds



Scary ones…