

Introduction to Deep Learning and Neural Network

Poo Kuan Hoong

Learning Objectives

- At the end of today's class, you will be able to:
 - Learn about artificial neural networks and deep learning
 - Understand how both techniques can be used for machine learning

Where are we now..

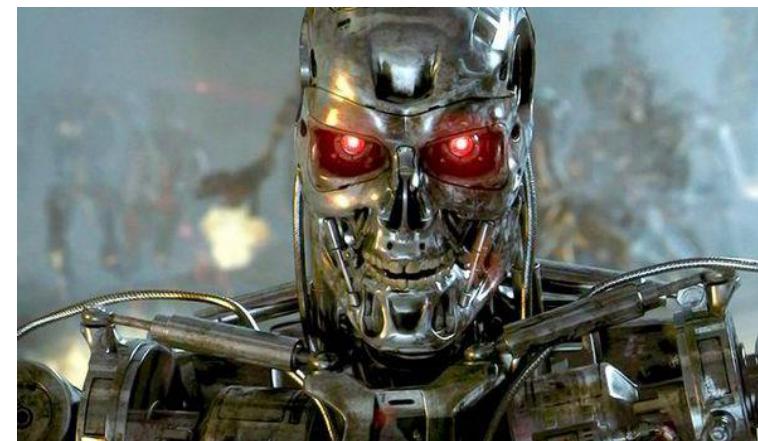
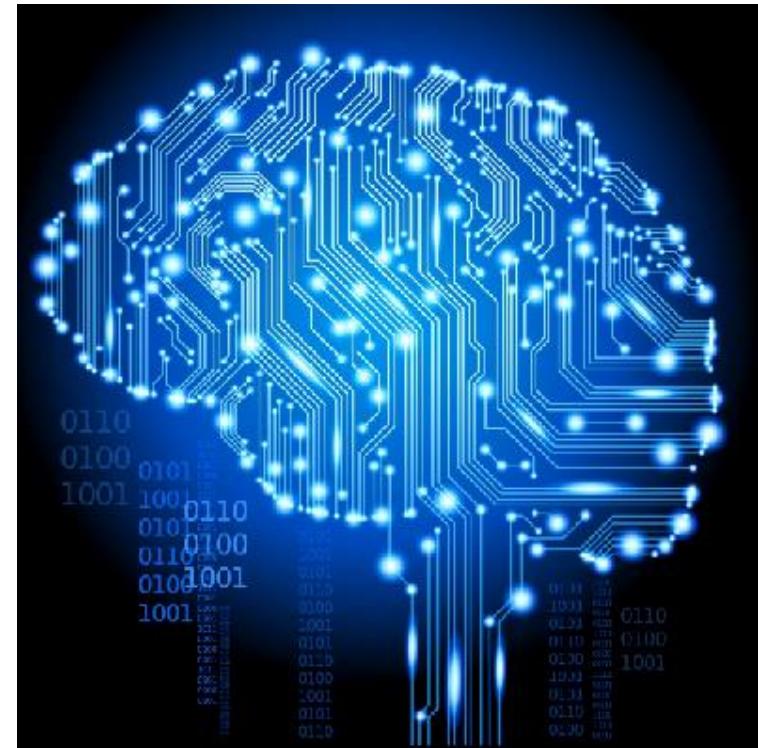


Where are we now..



Introduction

- **Deep Learning:** The fastest growing sub-field in A.I.
 - Computers can now make sense of **difficult tasks** by crunching **large amounts of data**
- **See, Hear, Learn, React**
- Many industries and businesses have been impacted by DL, spurring new products and services that are **data-driven**
- **Machine Learning:** Machines that can learn!

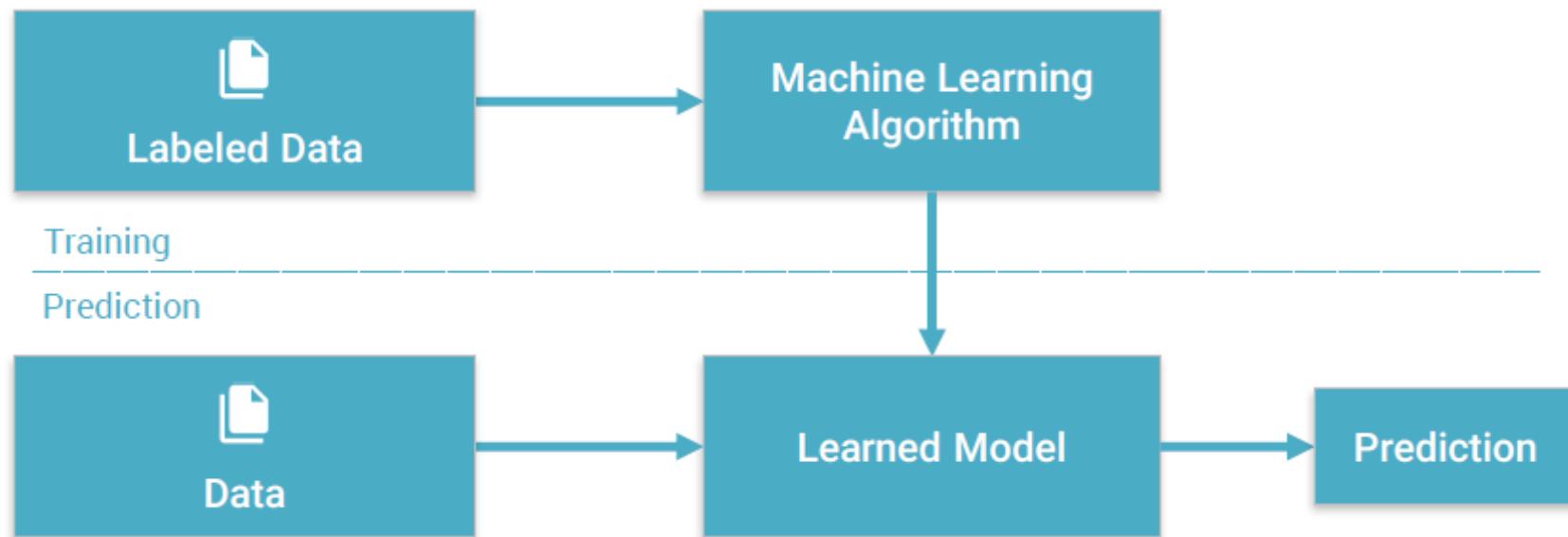




How do we learn?

Machine Learning

A type of Artificial Intelligence that provides computers with the ability to learn **without being explicitly programmed**



Machine Learning - Approaches



Supervised Learning

Learning from a labeled training set



Unsupervised Learning

Discovering patterns in unlabeled data



Reinforcement Learning

Learning based on feedback or reward

Machine Learning - Types of Problems



Classification



Regression



Clustering



Anomaly Detection

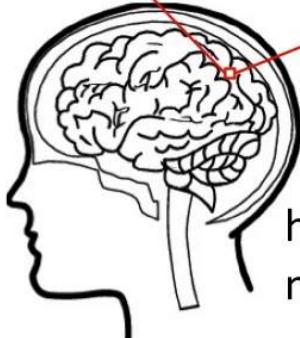
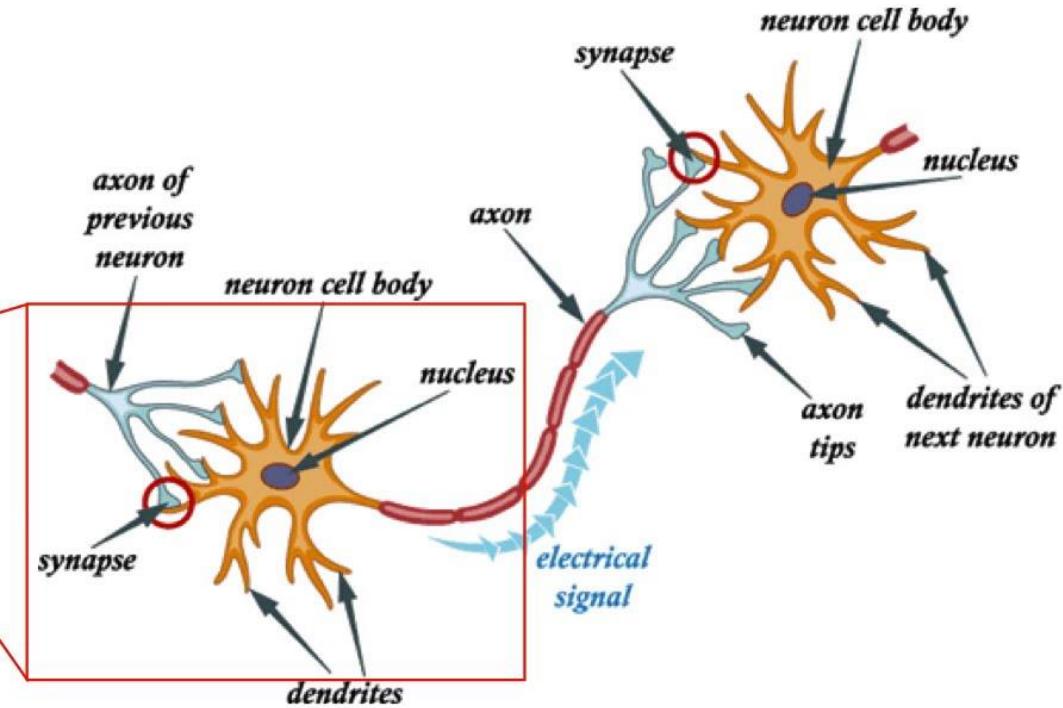
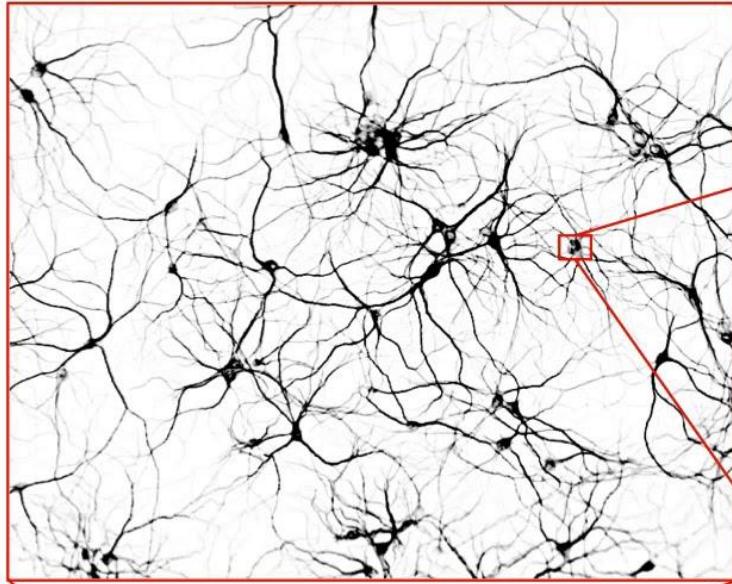
Neural Network

Biological Inspiration

- Animals are able to react adaptively to changes in their external and internal environment, and they use their nervous system to perform these behaviours.
- An appropriate model/simulation of the nervous system should be able to produce similar responses and behaviours in artificial systems.
- The nervous system is build by relatively simple units, the neurons, so copying their behaviour and functionality should be the solution.



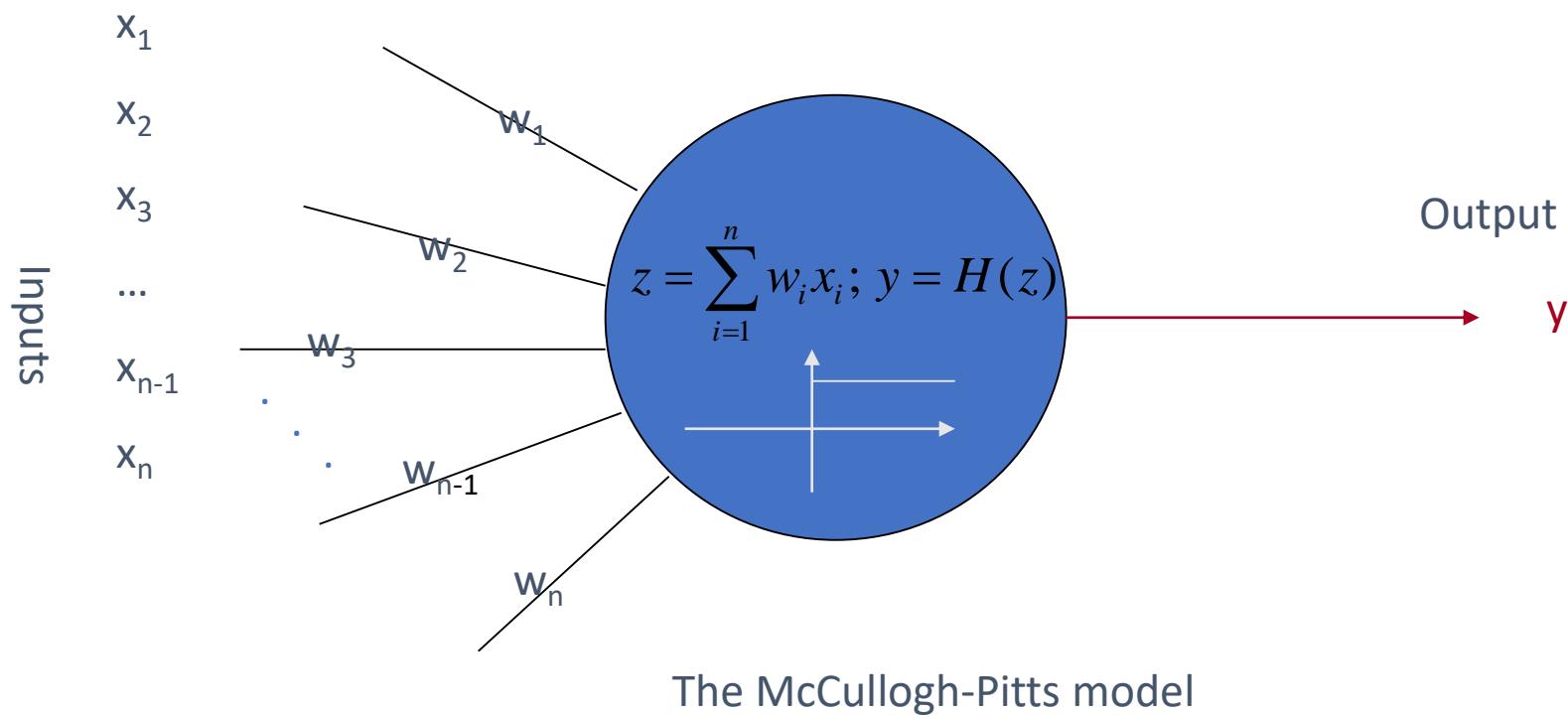
Neurons and the brain



humans don't
need features

Artificial Neurons

- Neurons work by processing information. They receive and provide information in form of spikes.



Artificial Neurons

Nonlinear generalization of the McCulloch-Pitts neuron:

$$y = f(x, w)$$

y is the neuron's output, x is the vector of inputs, and w is the vector of synaptic weights.

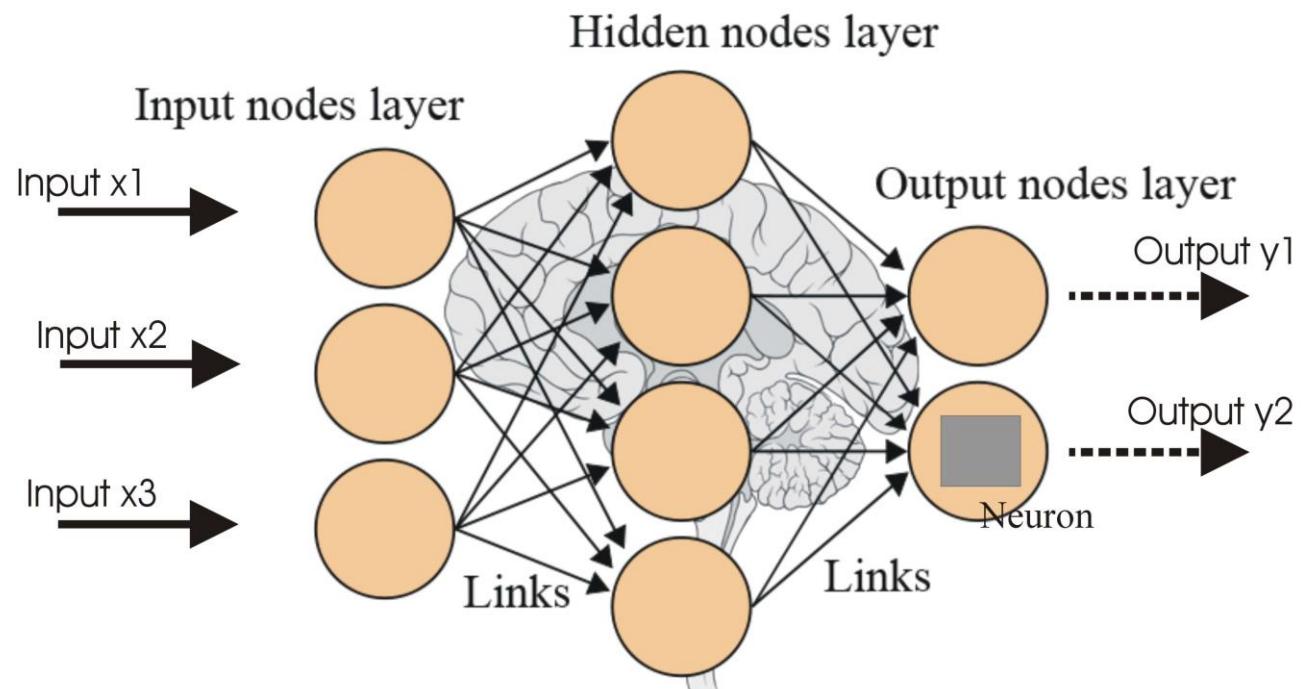
Examples:

$$y = \frac{1}{1 + e^{-w^T x - a}}$$
 sigmoidal neuron

$$y = e^{-\frac{\|x-w\|^2}{2a^2}}$$
 Gaussian neuron

Artificial Neural Networks

- An artificial neural network is composed of many artificial neurons that are linked together according to a specific network architecture. The objective of the neural network is to transform the inputs into meaningful outputs.



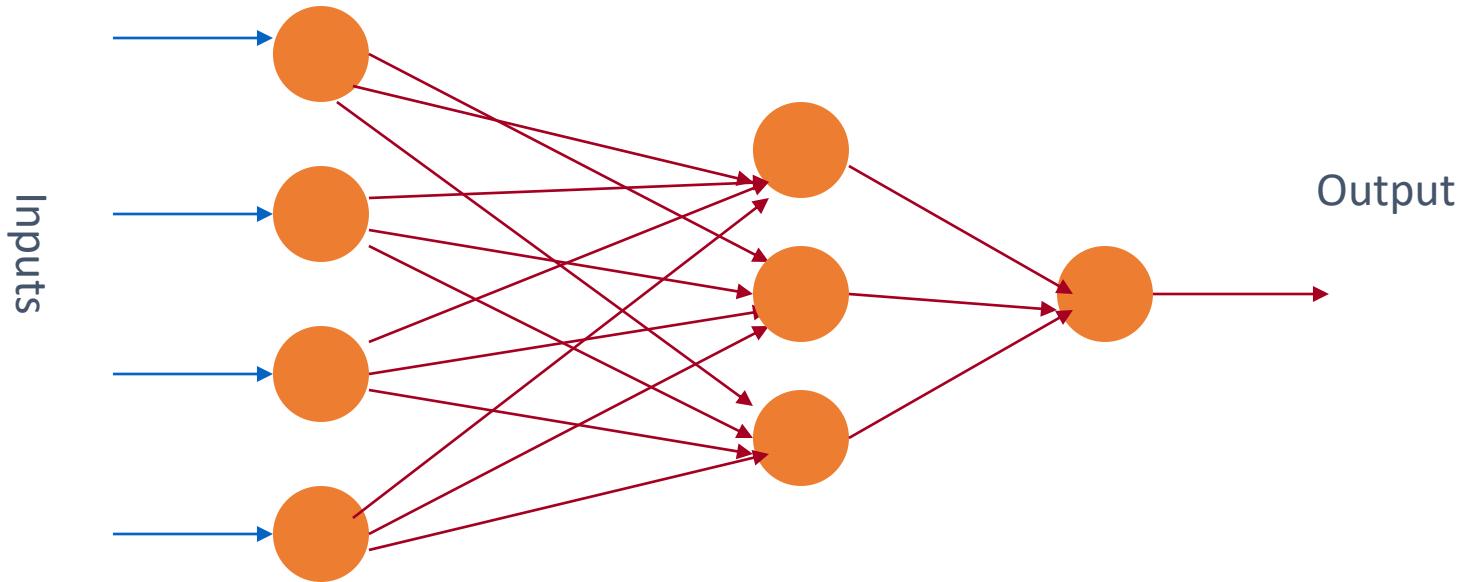
Artificial neural networks

- Tasks to be solved by artificial neural networks:
 - controlling the movements of a robot based on self-perception and other information (e.g., visual information);
 - deciding the category of potential food items (e.g., edible or non-edible) in an artificial world;
 - recognizing a visual object (e.g., a familiar face);
 - predicting where a moving object goes, when a robot wants to catch it.

Learning in biological neural networks

- The learning rules of Hebb:
 - synchronous activation increases the synaptic strength;
 - asynchronous activation decreases the synaptic strength.
- These rules fit with energy minimization principles.
- Maintaining synaptic strength needs energy, it should be maintained at those places where it is needed, and it shouldn't be maintained at places where it's not needed.

Neural network mathematics



$$\begin{aligned}y_1^1 &= f(x_1, w_1^1) \\y_2^1 &= f(x_2, w_2^1) \\y_3^1 &= f(x_3, w_3^1) \\y_4^1 &= f(x_4, w_4^1)\end{aligned} \quad y^1 = \begin{pmatrix} y_1^1 \\ y_2^1 \\ y_3^1 \\ y_4^1 \end{pmatrix} \quad \begin{aligned}y_1^2 &= f(y^1, w_1^2) \\y_2^2 &= f(y^1, w_2^2) \\y_3^2 &= f(y^1, w_3^2)\end{aligned} \quad y^2 = \begin{pmatrix} y_1^2 \\ y_2^2 \\ y_3^2 \end{pmatrix} \quad y_{Out} = f(y^2, w_1^3)$$

Neural network mathematics

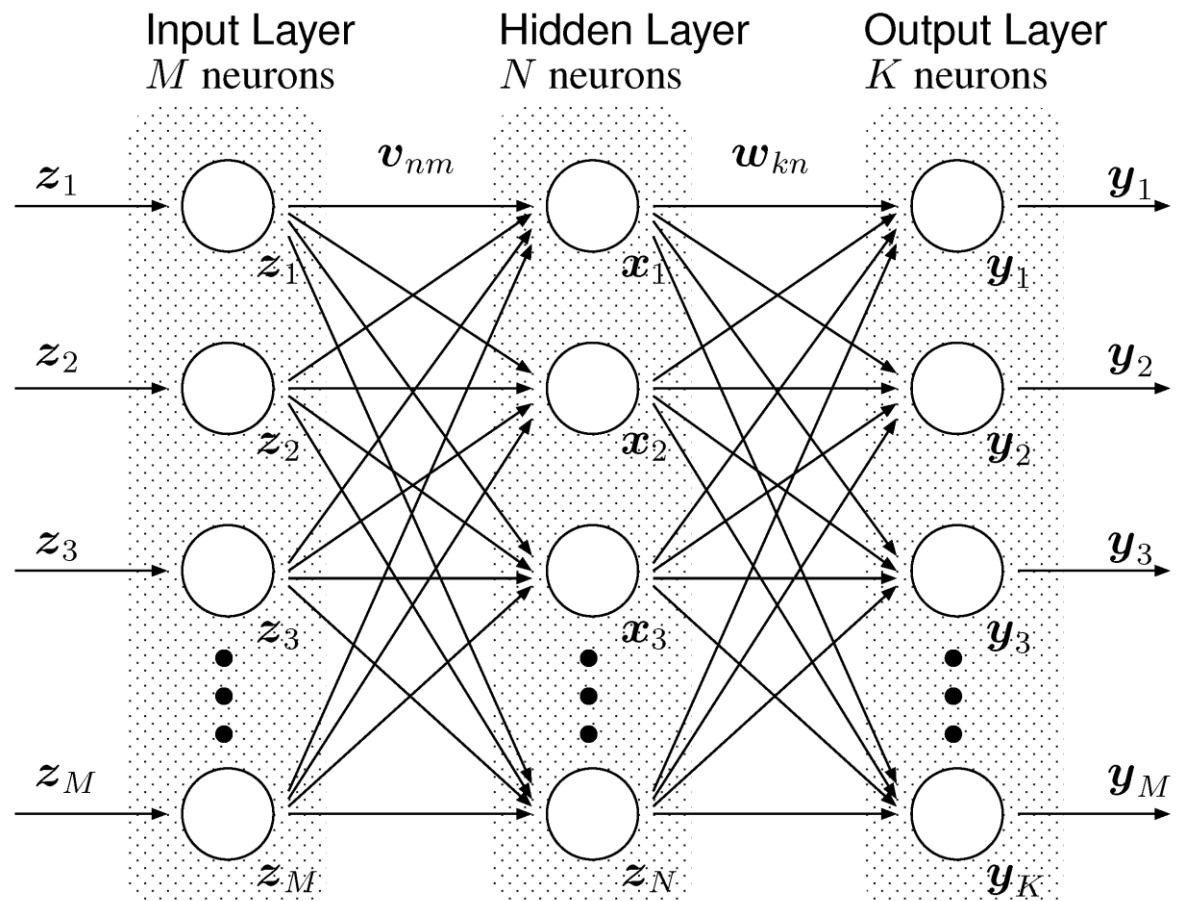
- Neural network: input / output transformation

$$y_{out} = F(x, W)$$

- W is the matrix of all weight vectors.

Multi-layer Perceptron (MLP)

- A **multi-layer perceptron (MLP)** is a class of feedforward artificial neural network.
- An MLP consists of at least three layers of nodes.
- Except for the input nodes, each node is a neuron that uses a nonlinear activation function.
- MLP utilizes a supervised learning technique called backpropagation for training.

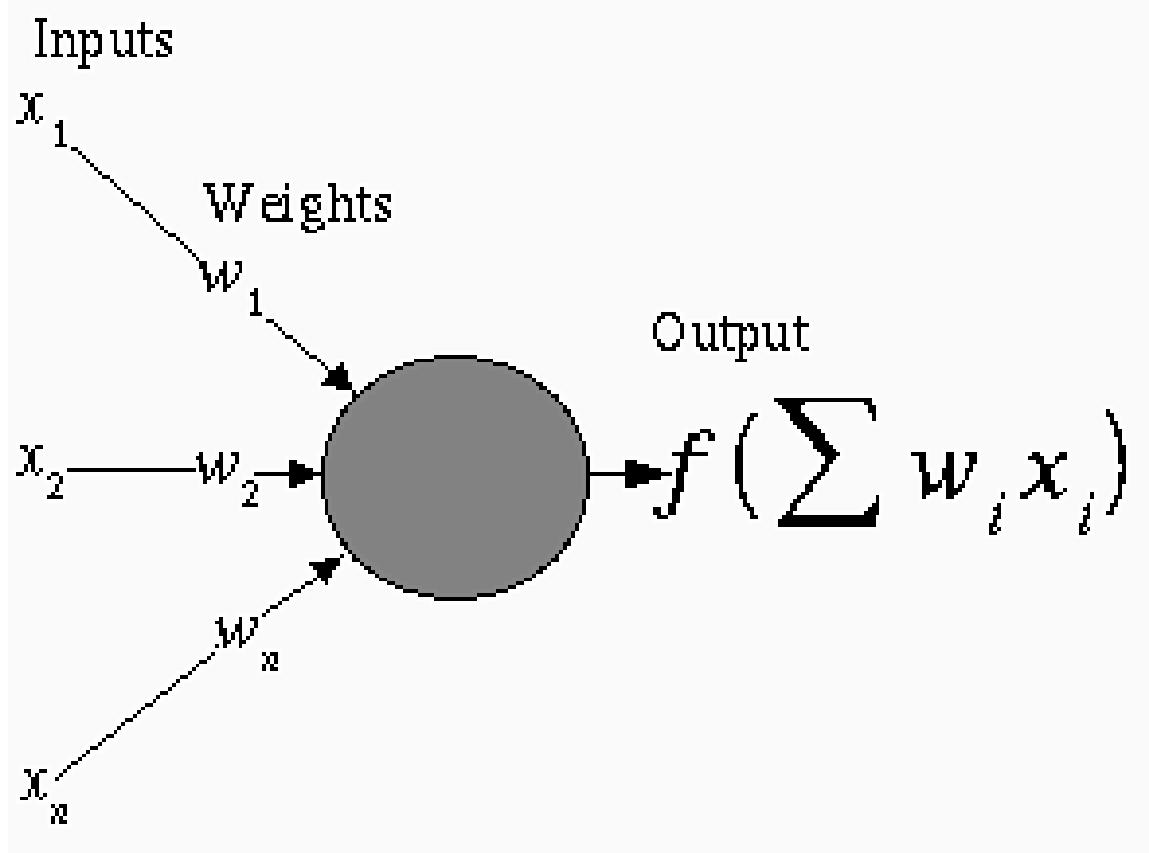


Backpropagation

- Solution of the complicated learning:
 - calculate first the changes for the synaptic weights of the output neuron;
 - calculate the changes backward starting from layer p-1, and propagate backward the local error terms.
- The method is still relatively complicated but it is much simpler than the original optimisation problem.

Backpropagation

- **Inputs x_i** , arrive through pre-synaptic connections
- Synaptic efficacy is modeled using real **weights w_i**
- The response of the neuron is a **nonlinear function f** of its weighted inputs



Inputs To Neurons

- Arise from other neurons or from outside the network
- Nodes whose inputs arise outside the network are called *input nodes* and simply copy values
- An input may *excite* or *inhibit* the response of the neuron to which it is applied, depending upon the weight of the connection

Weights

- Represent synaptic efficacy and may be *excitatory* or *inhibitory*
- Normally, positive weights are considered as excitatory while negative weights are thought of as inhibitory
- **Learning** is the process of modifying the weights in order to produce a network that performs some function

Outputs

- The response function is normally nonlinear
- Samples include
 - Sigmoid

$$f(x) = \frac{1}{1 + e^{-\lambda x}}$$

- Piecewise linear

$$f(x) = \begin{cases} x, & \text{if } x \geq \theta \\ 0, & \text{if } x < \theta \end{cases}$$

Backpropagation Preparation

- **Training Set**
A collection of input-output patterns that are used to train the network
- **Testing Set**
A collection of input-output patterns that are used to assess network performance
- **Learning Rate- η**
A scalar parameter, analogous to step size in numerical integration, used to set the rate of adjustments

Network Error

- Total-Sum-Squared-Error (TSSE)

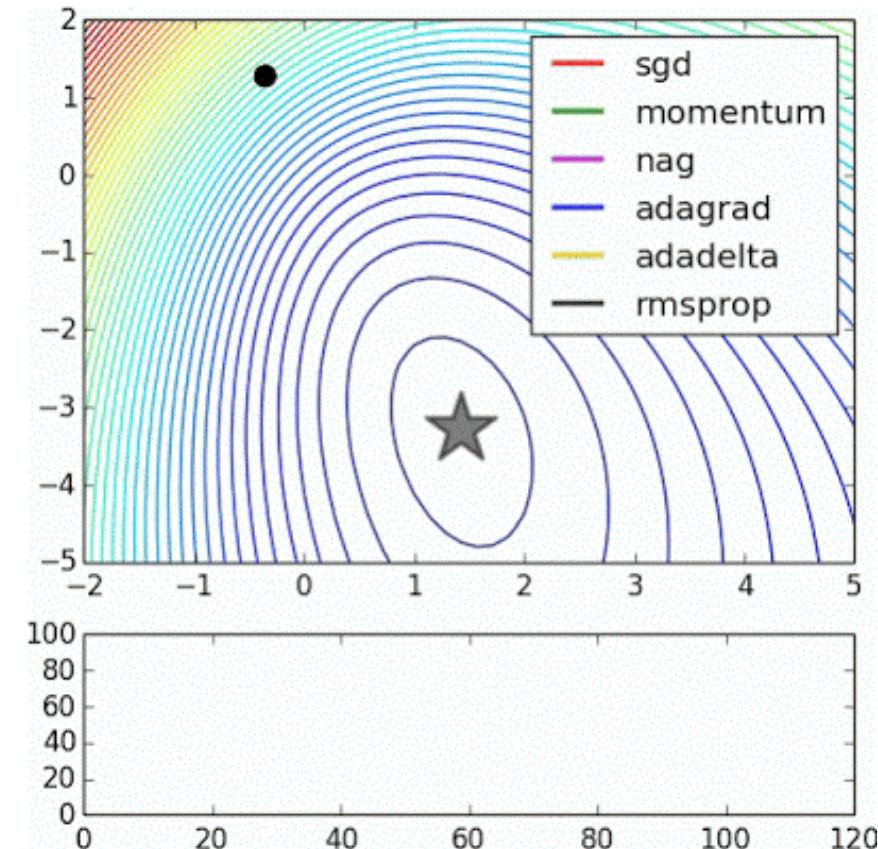
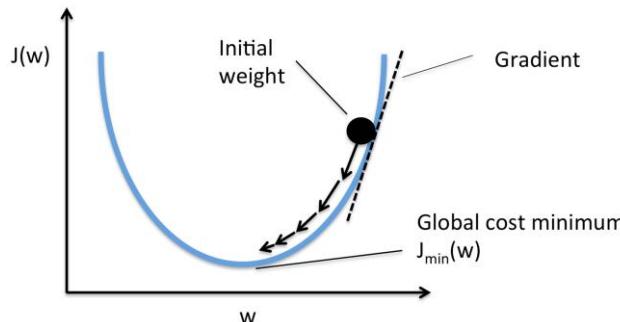
$$TSSE = \frac{1}{2} \sum_{\text{patterns}} \sum_{\text{outputs}} (\text{desired} - \text{actual})^2$$

- Root-Mean-Squared-Error (RMSE)

$$RMSE = \sqrt{\frac{2 * TSSE}{\# \text{patterns} * \# \text{outputs}}}$$

NN Basics: Optimizing the Cost Function

- Re-adjusting the weights and biases is an **optimization** problem
- A **cost function** is defined to measure overall error which needs to be minimized
- **Gradients** are used to update weight parameters

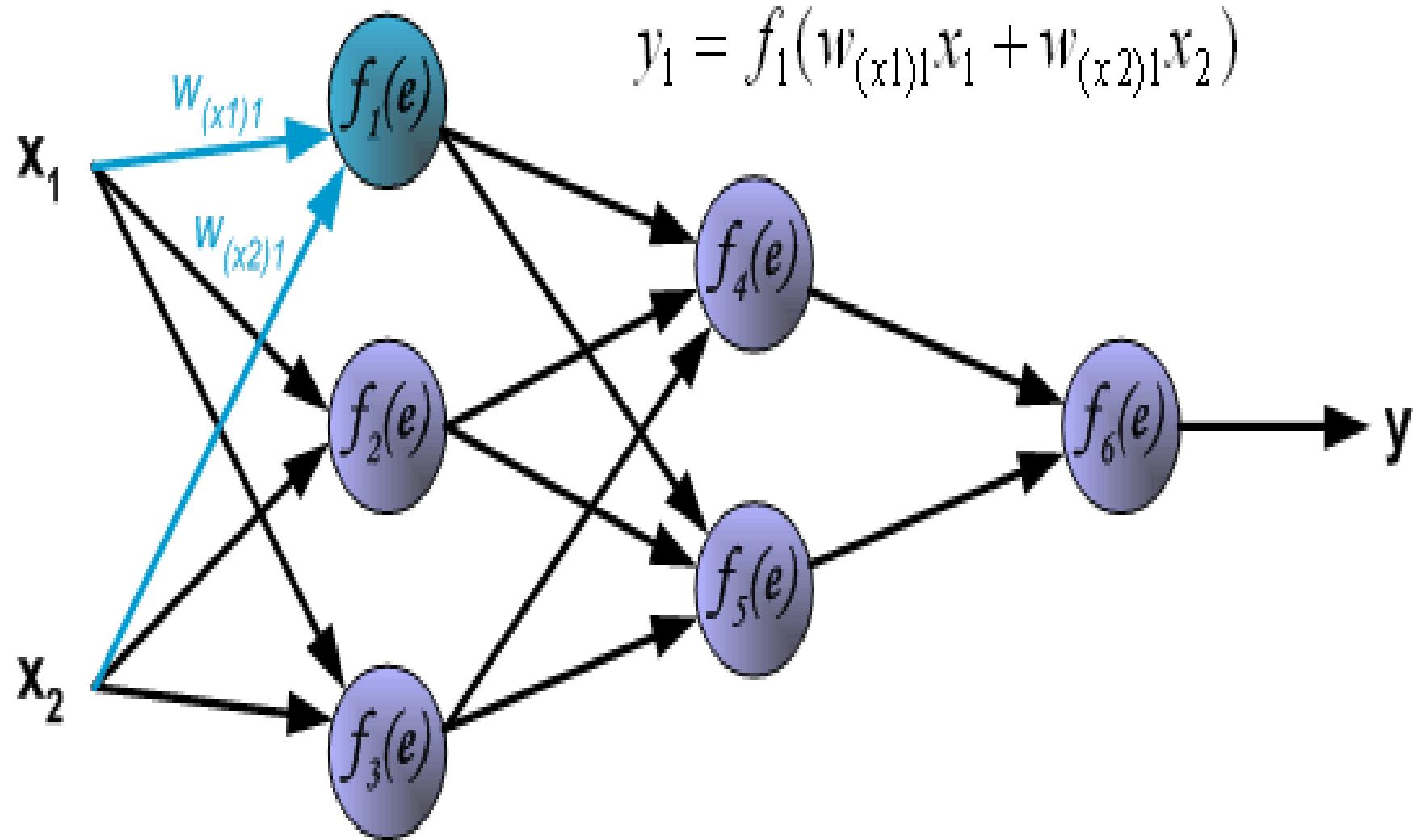


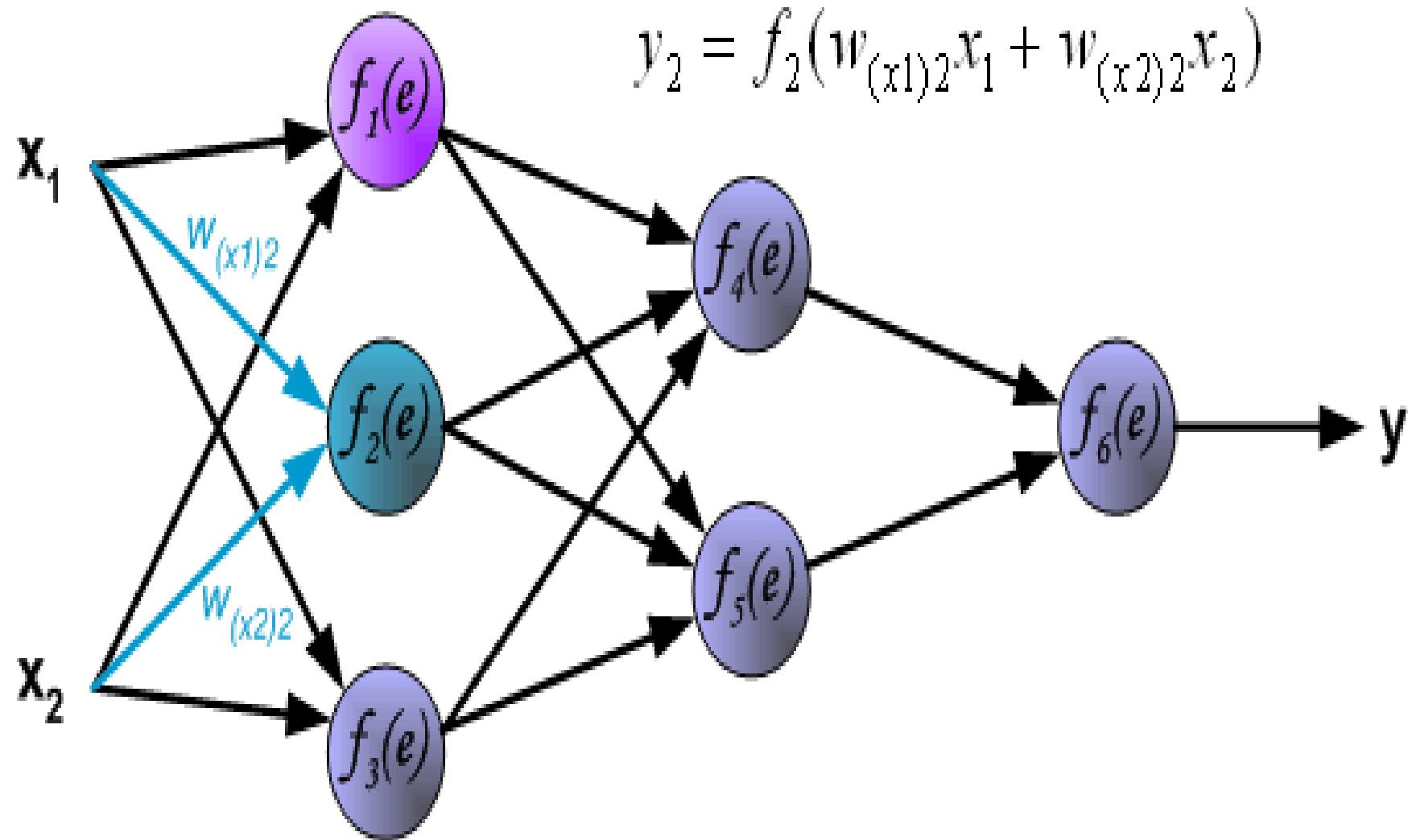
A Pseudo-Code Algorithm

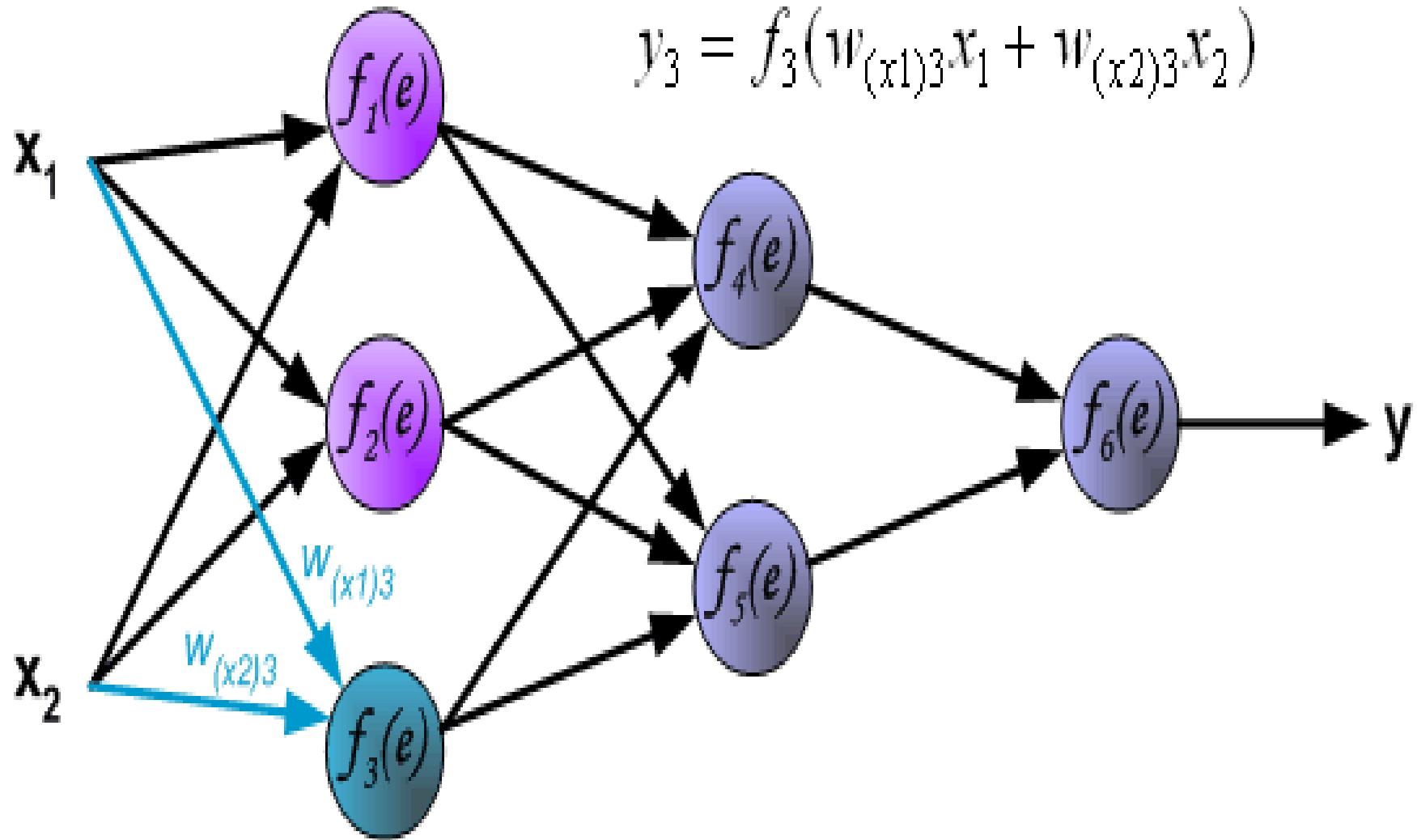
- Randomly choose the initial weights
- While error is too large
 - For each training pattern (presented in random order)
 - Apply the inputs to the network
 - Calculate the output for every neuron from the input layer, through the hidden layer(s), to the output layer
 - Calculate the error at the outputs
 - Use the output error to compute error signals for pre-output layers
 - Use the error signals to compute weight adjustments
 - Apply the weight adjustments
 - Periodically evaluate the network performance

Feedforward Network Training by Backpropagation: Process Summary

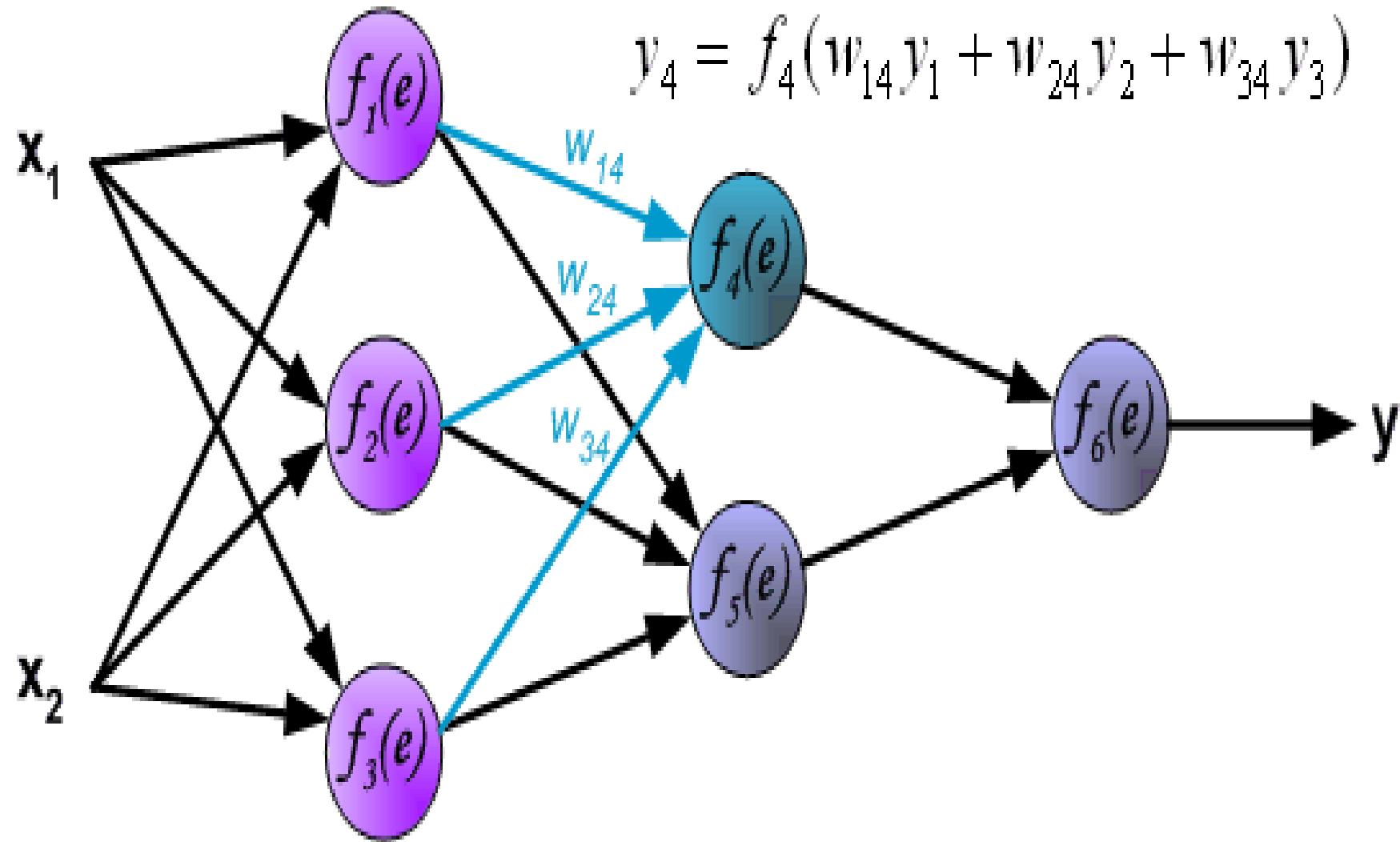
- Select an architecture
- Randomly initialize weights
- While error is too large
 - Select training pattern and feedforward to find actual network output
 - Calculate errors and backpropagate error signals
 - Adjust weights
- Evaluate performance using the test set



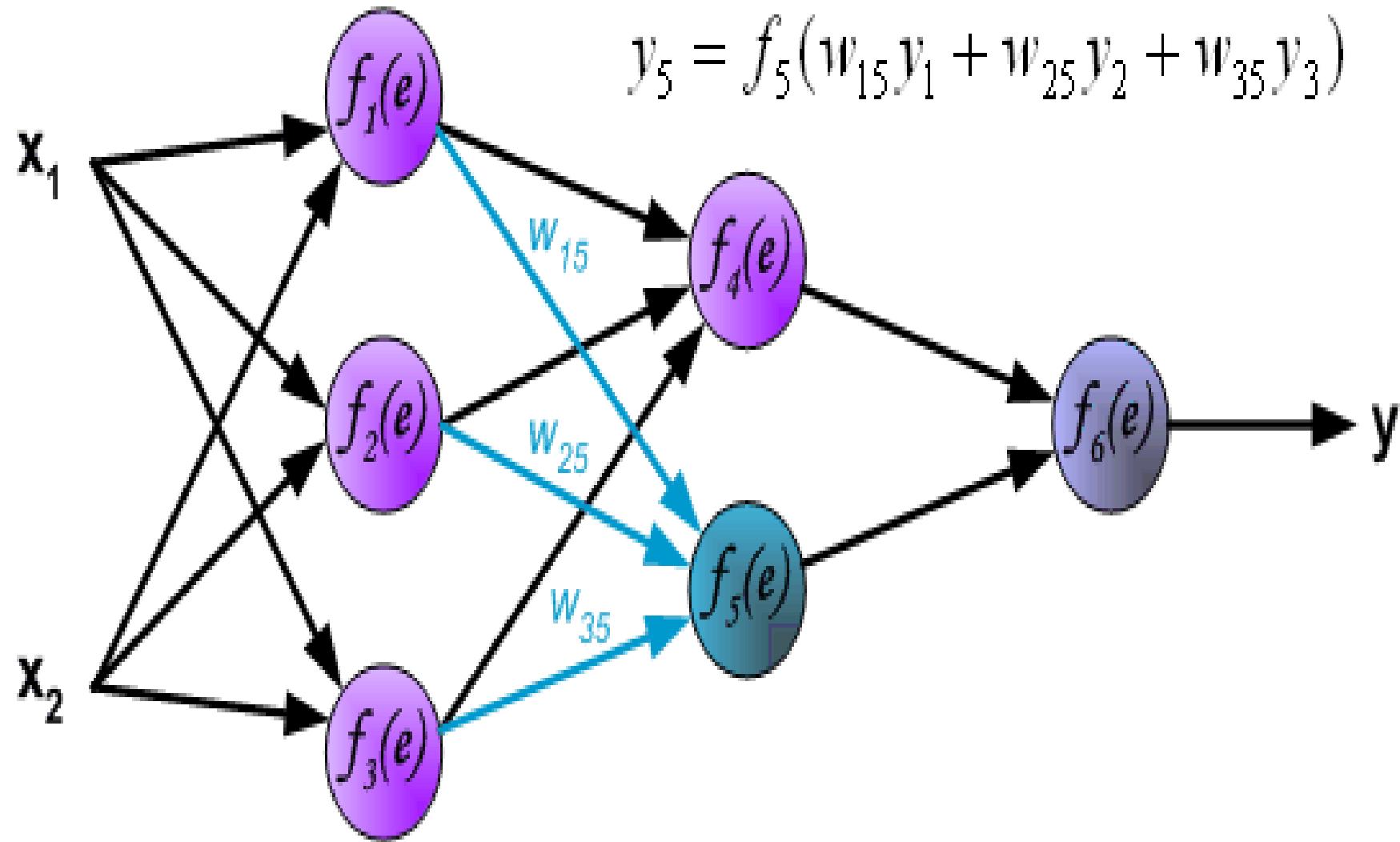


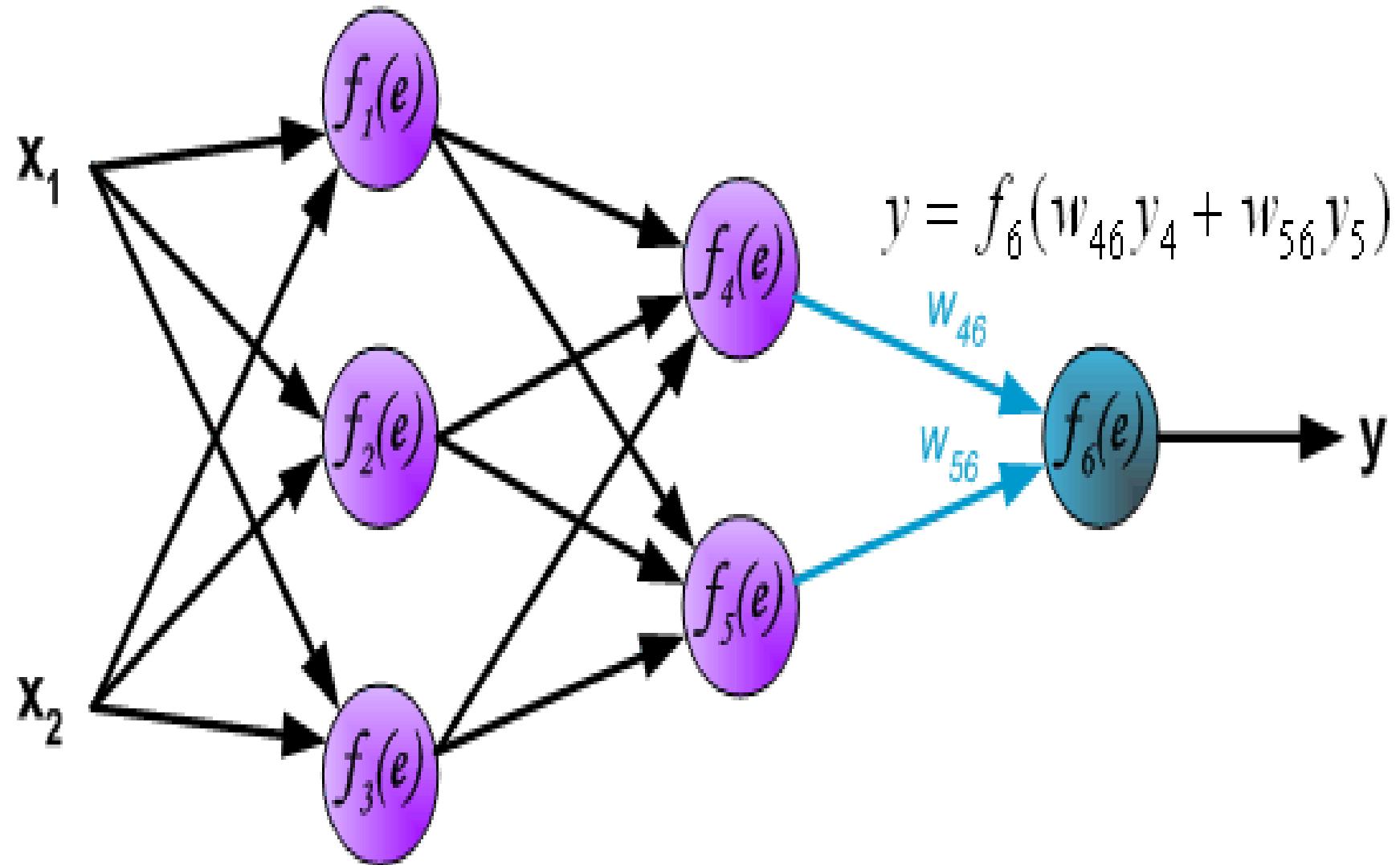


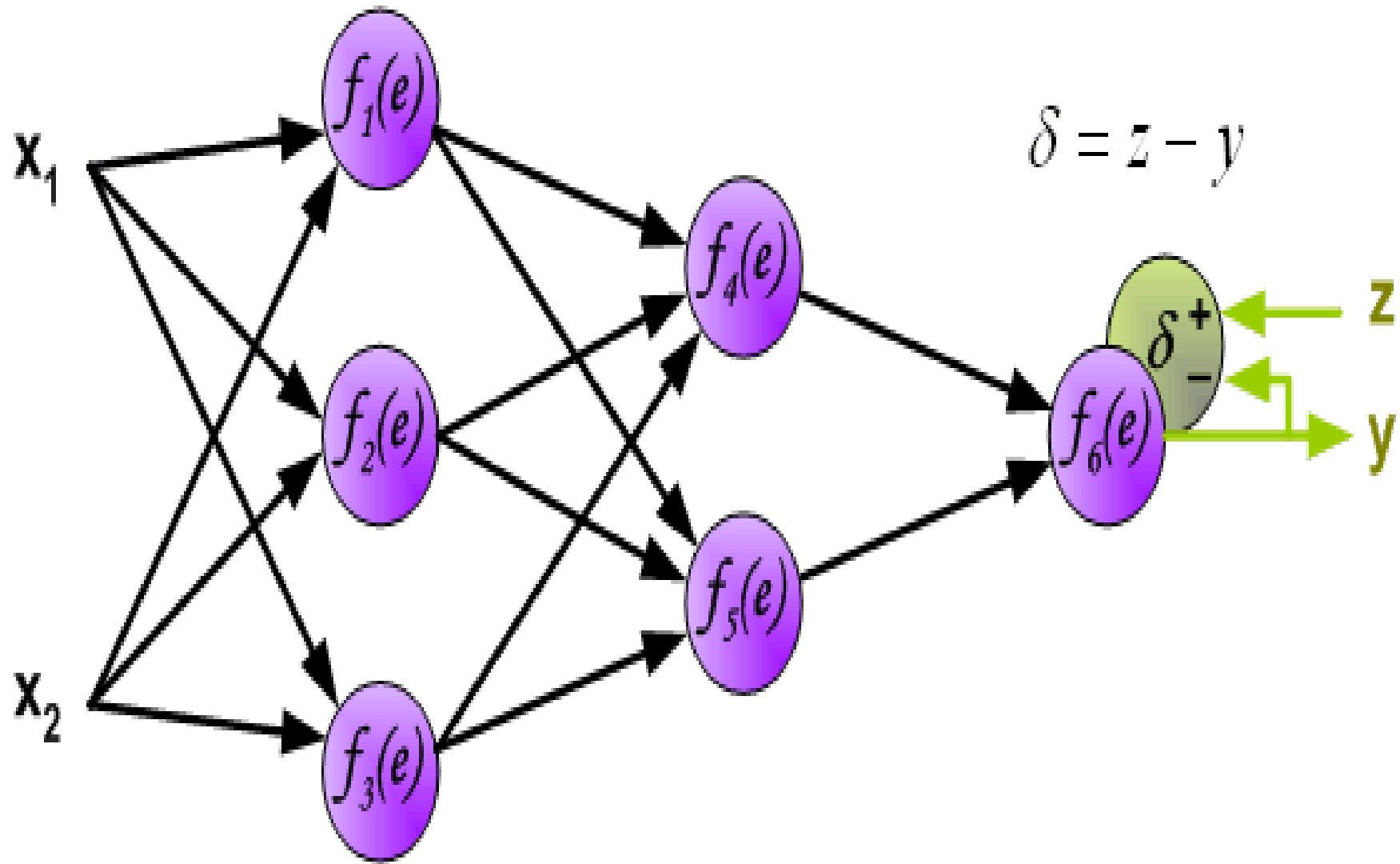
$$y_3 = f_3(w_{(x1)3}x_1 + w_{(x2)3}x_2)$$



$$y_4 = f_4(w_{14}y_1 + w_{24}y_2 + w_{34}y_3)$$



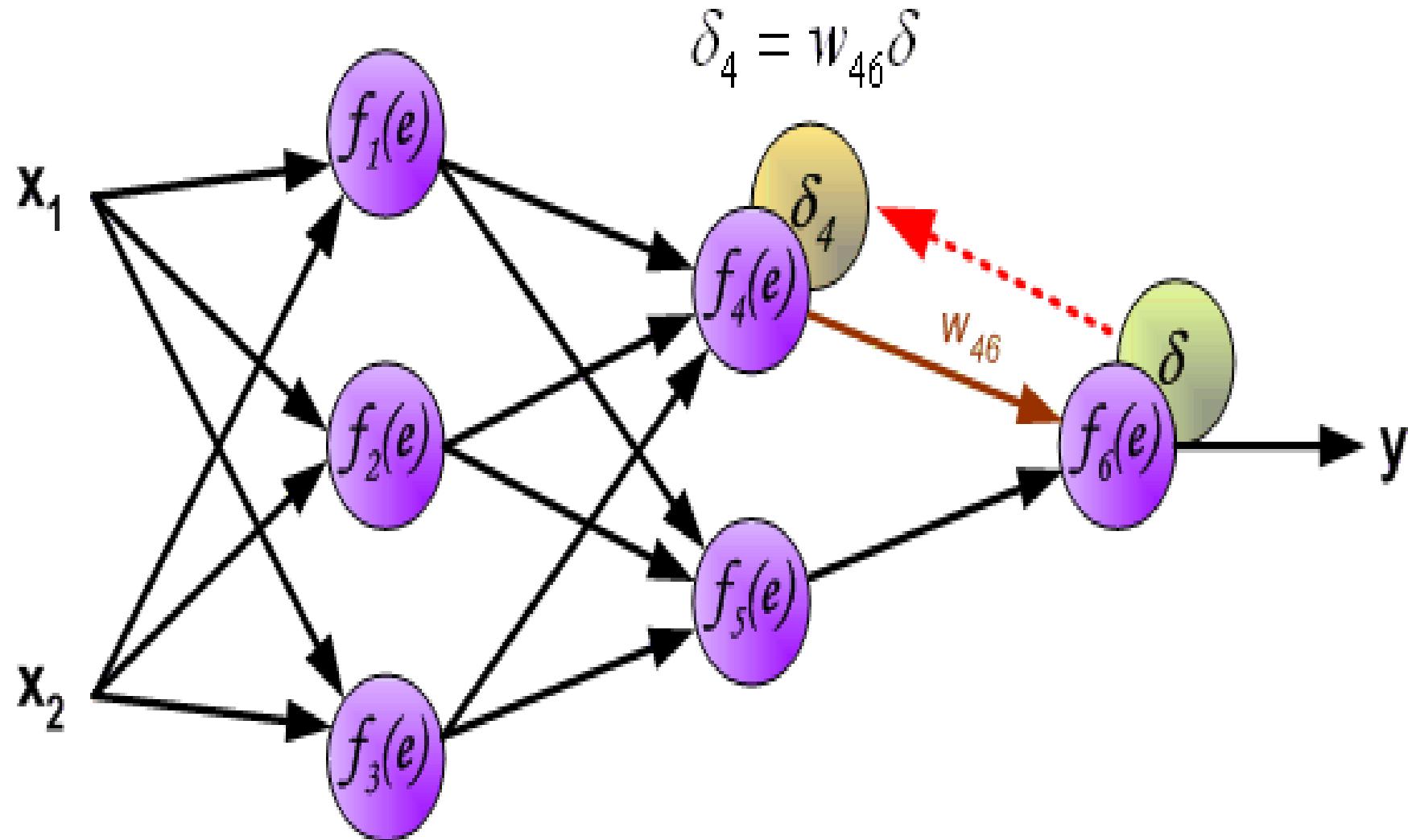


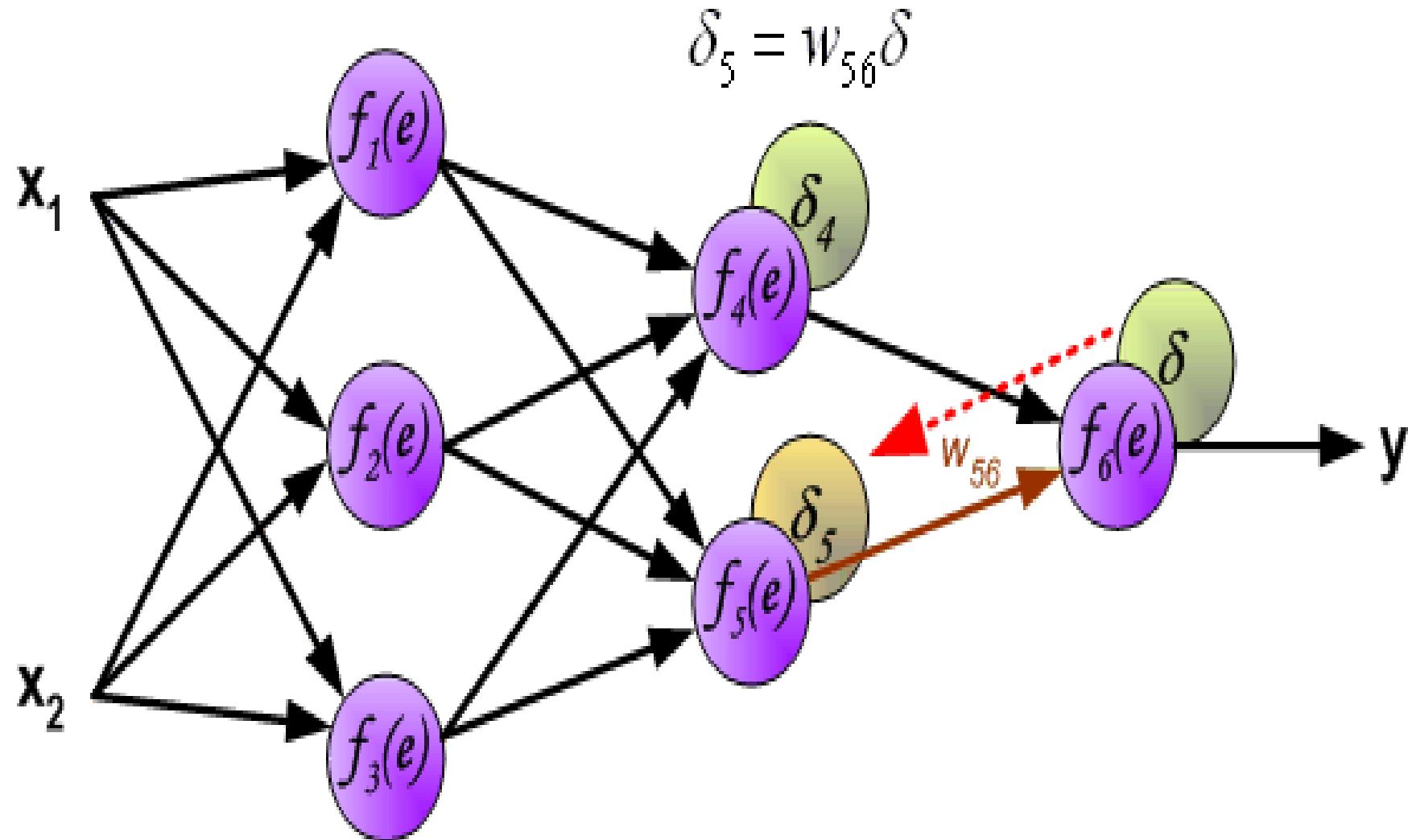


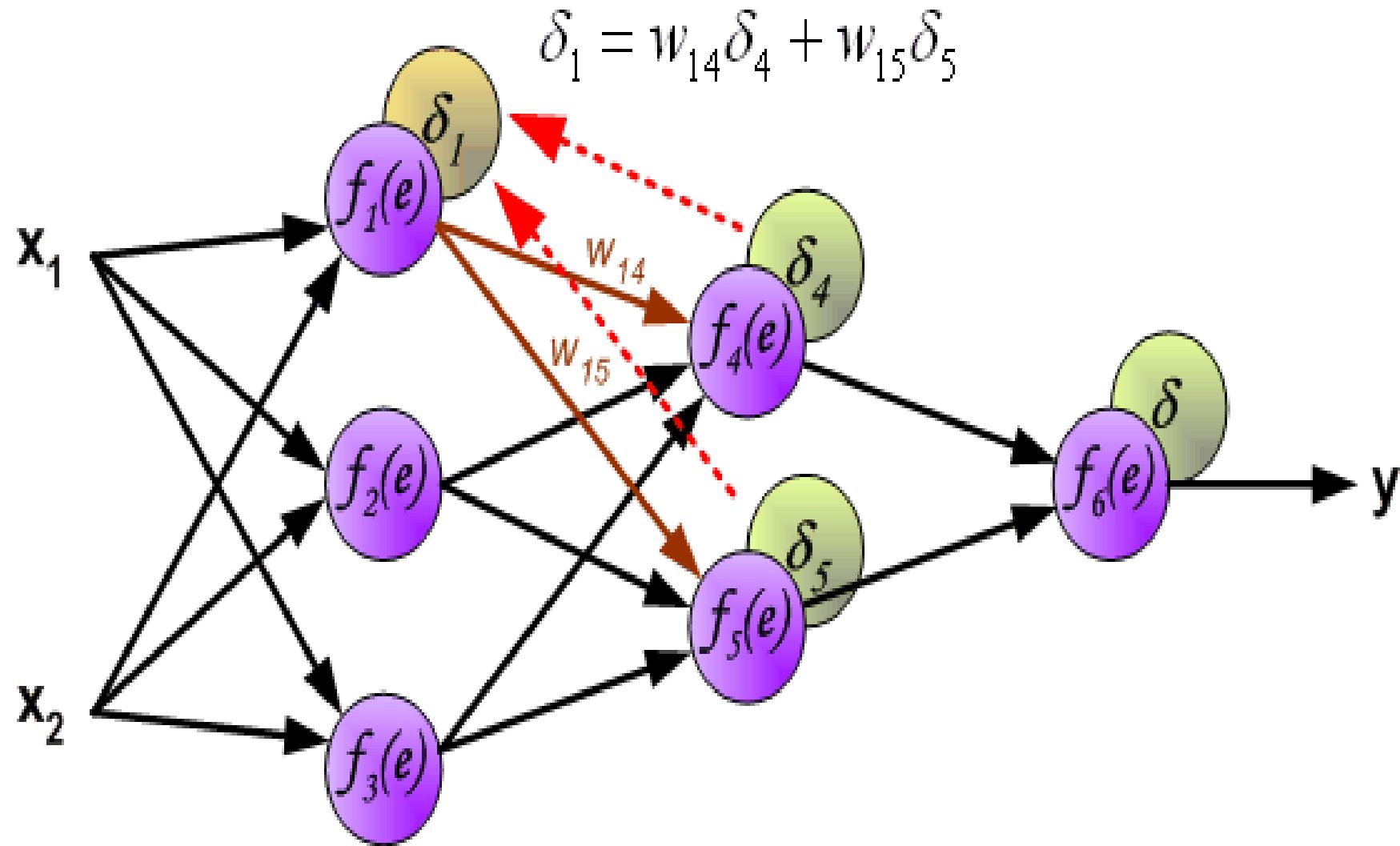
$$\delta = z - y$$

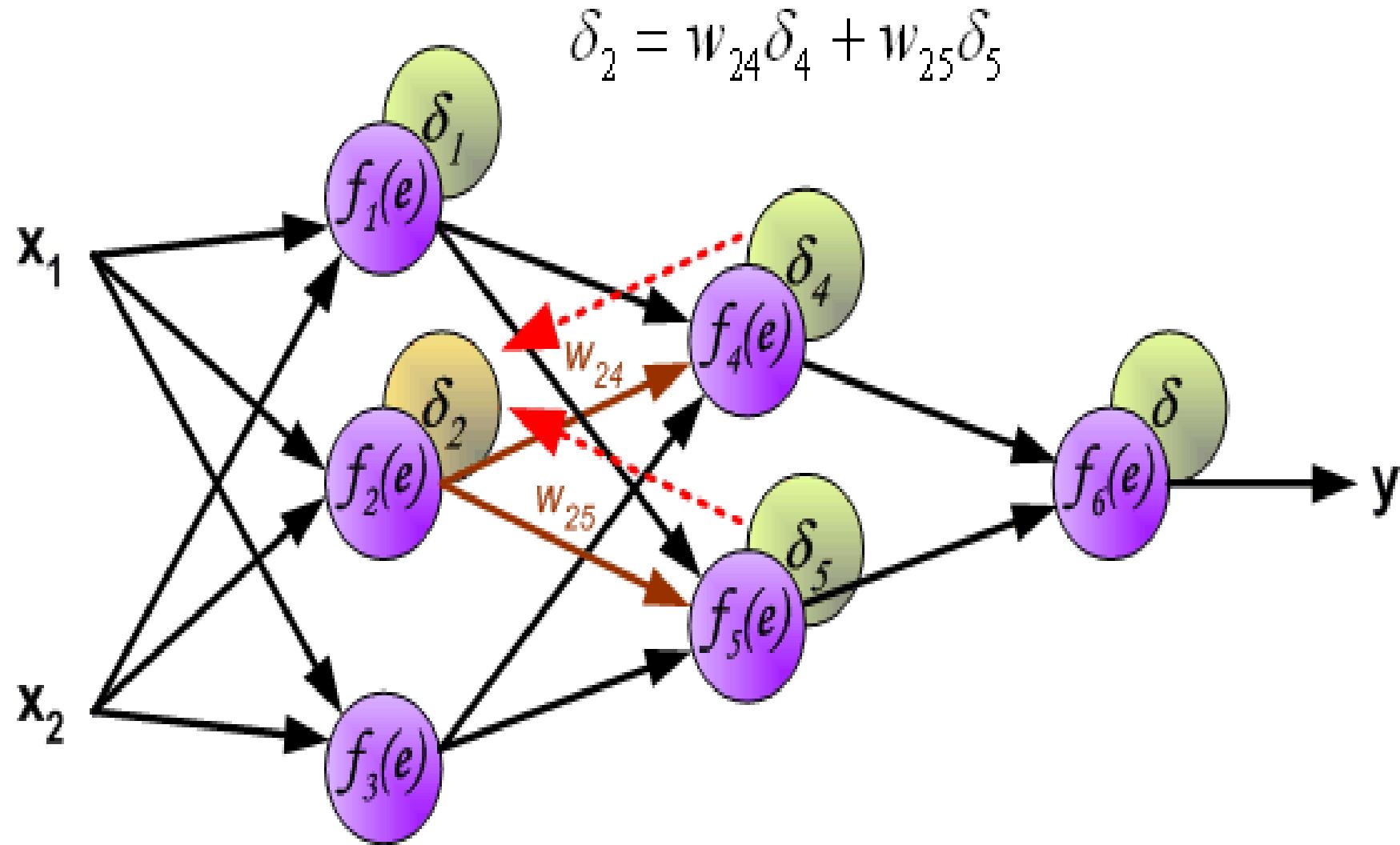
z

y

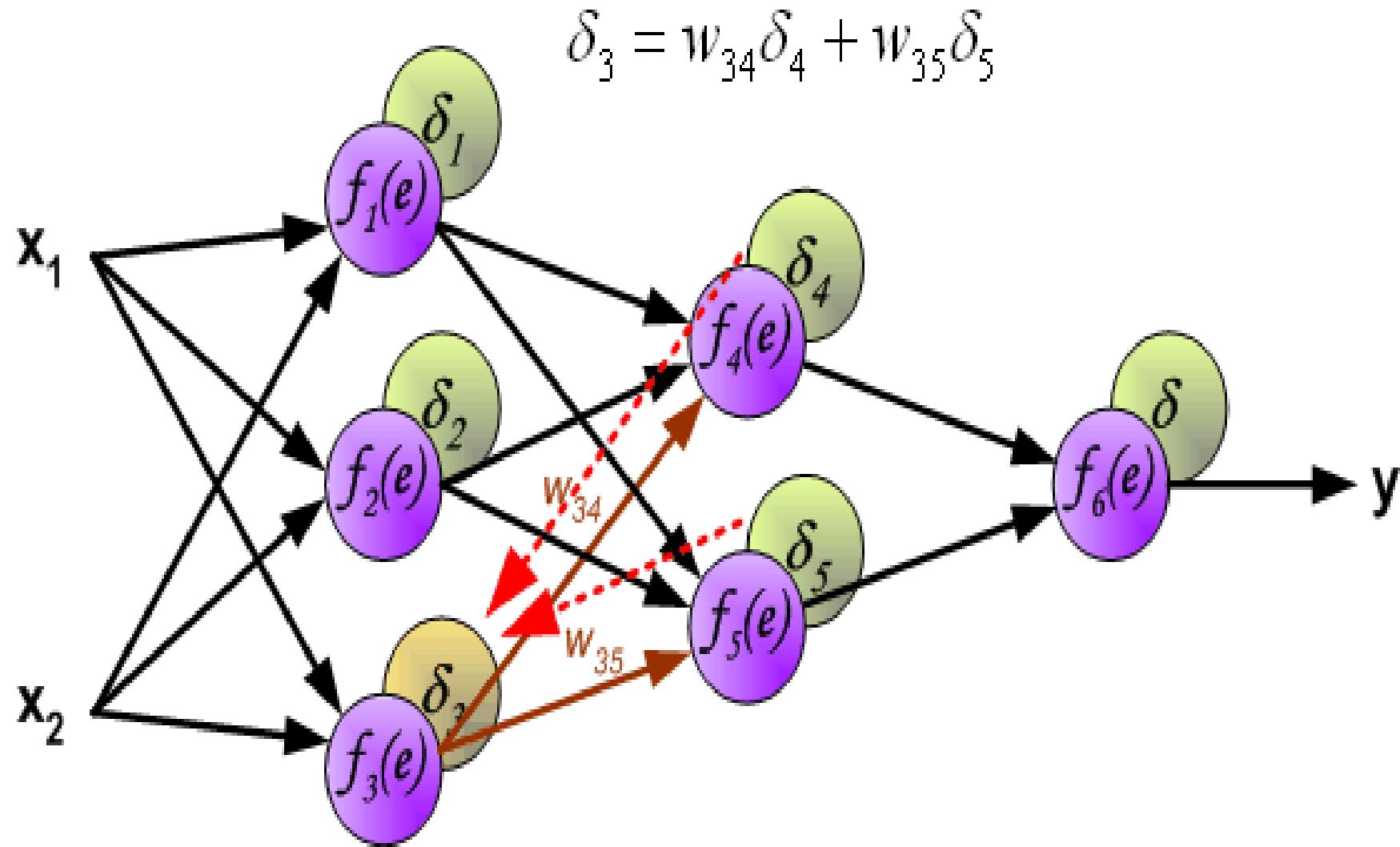


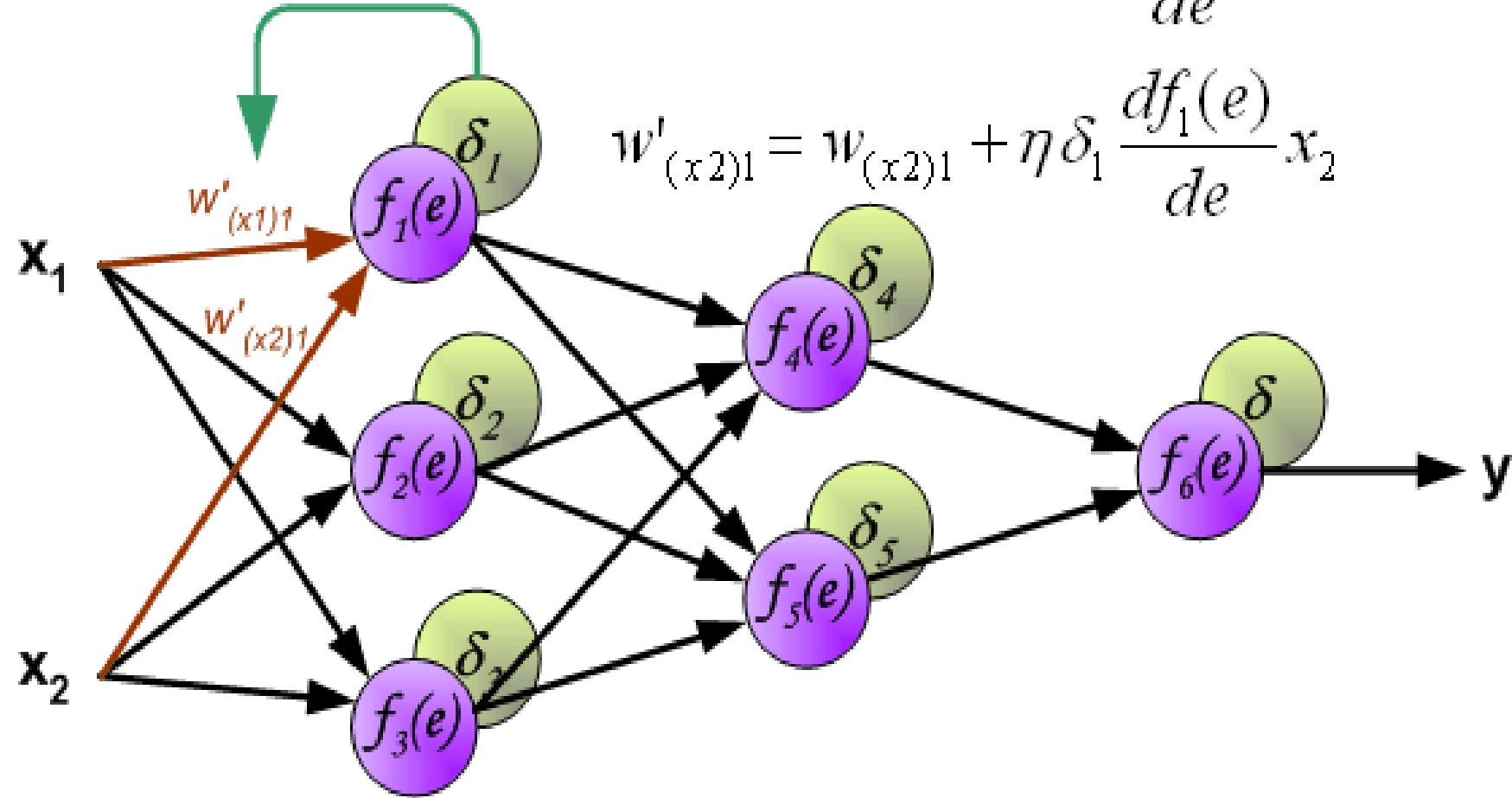






$$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5$$



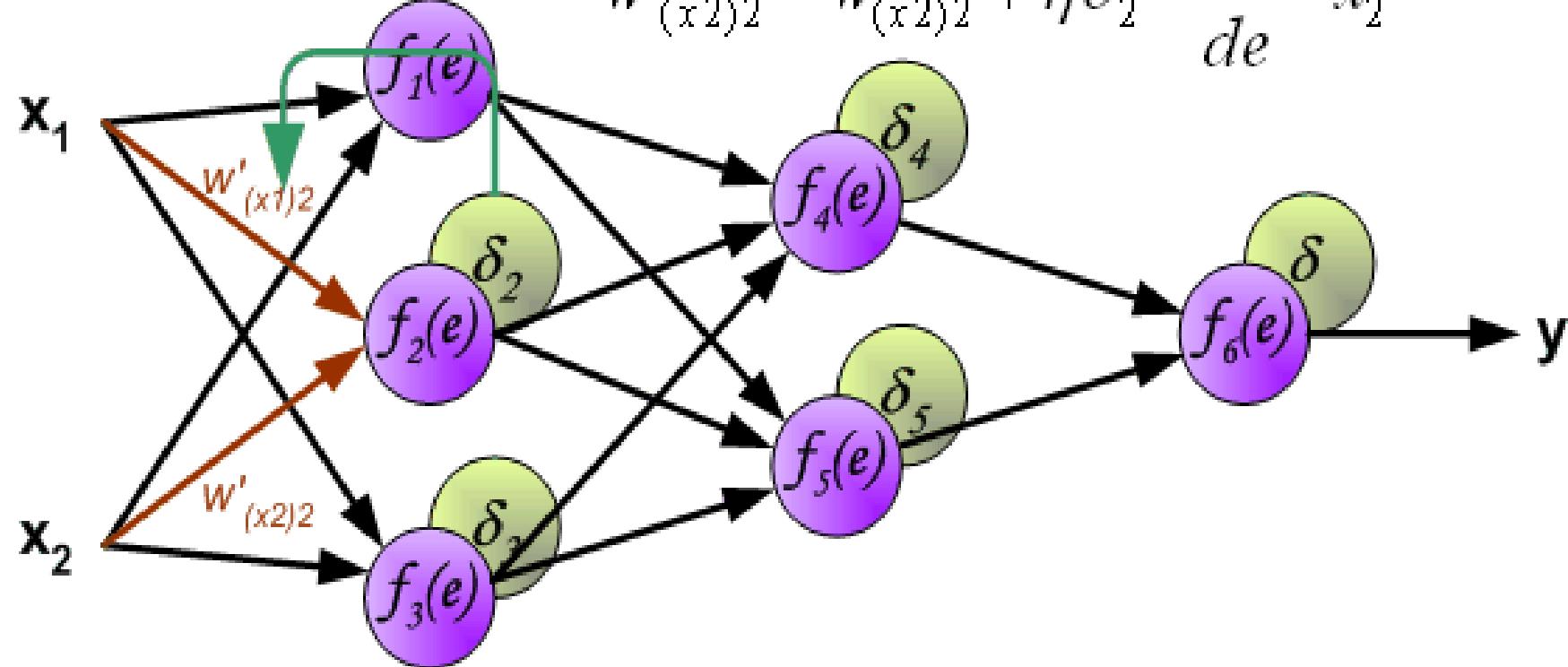


$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$

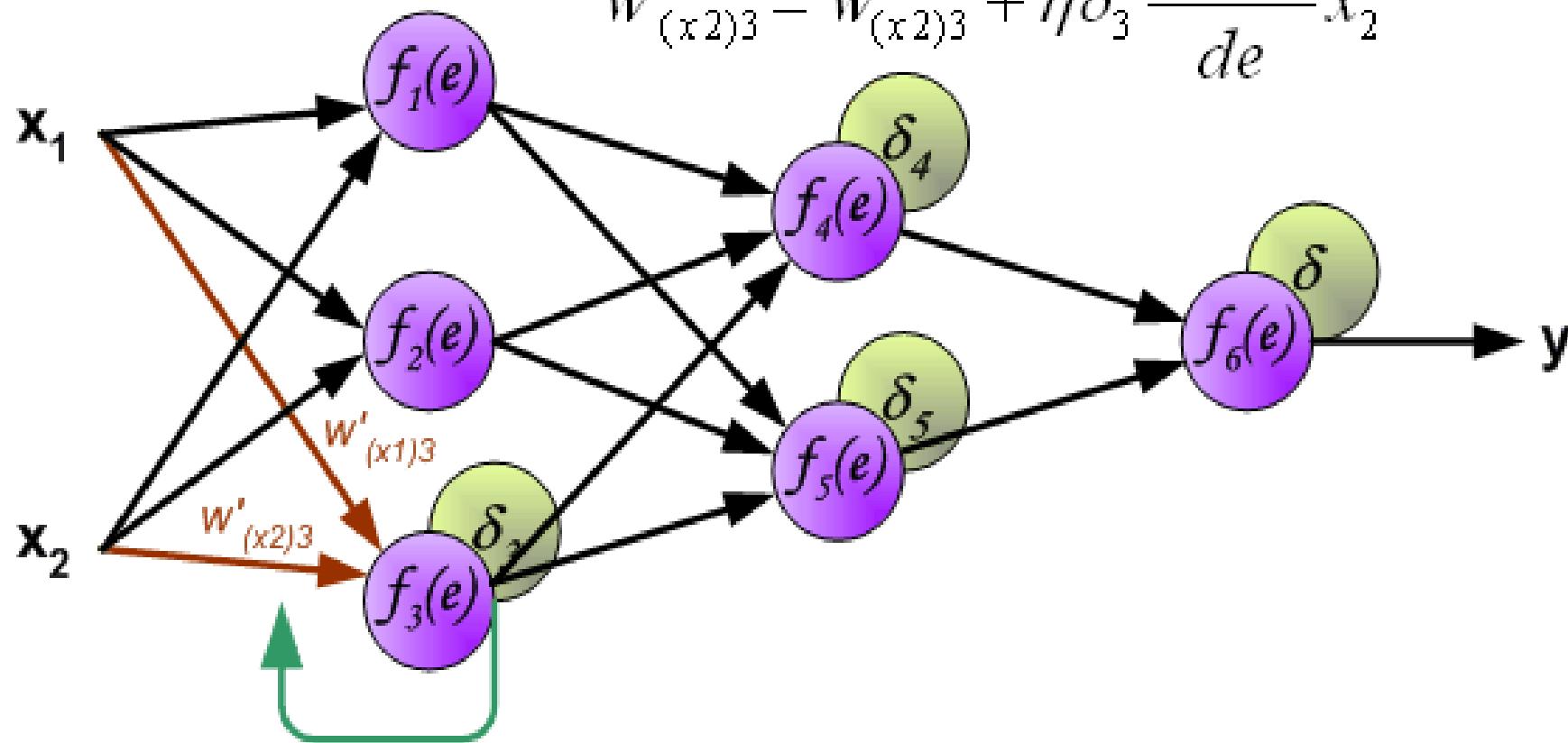
$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



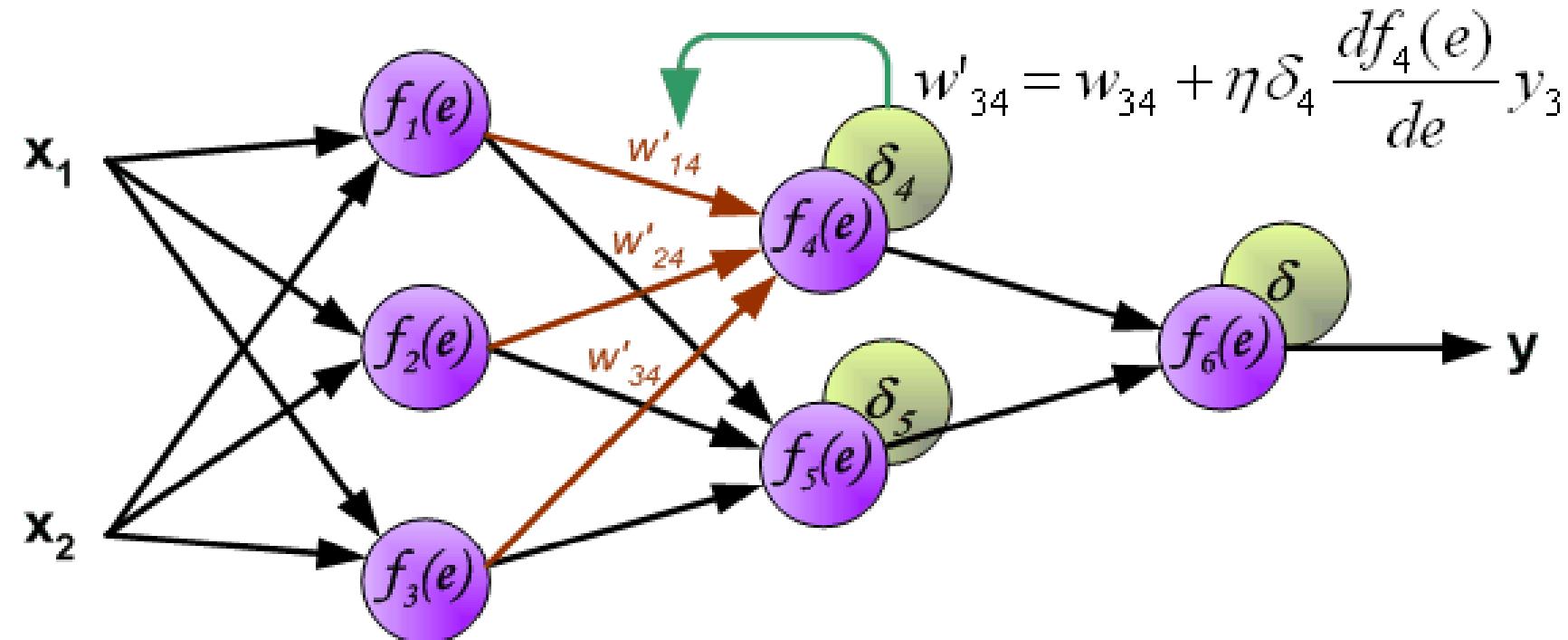
$$w'_{(x1)3} = w_{(x1)3} + \eta \delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} + \eta \delta_3 \frac{df_3(e)}{de} x_2$$



$$w'_{14} = w_{14} + \eta \delta_4 \frac{df_4(e)}{de} y_1$$

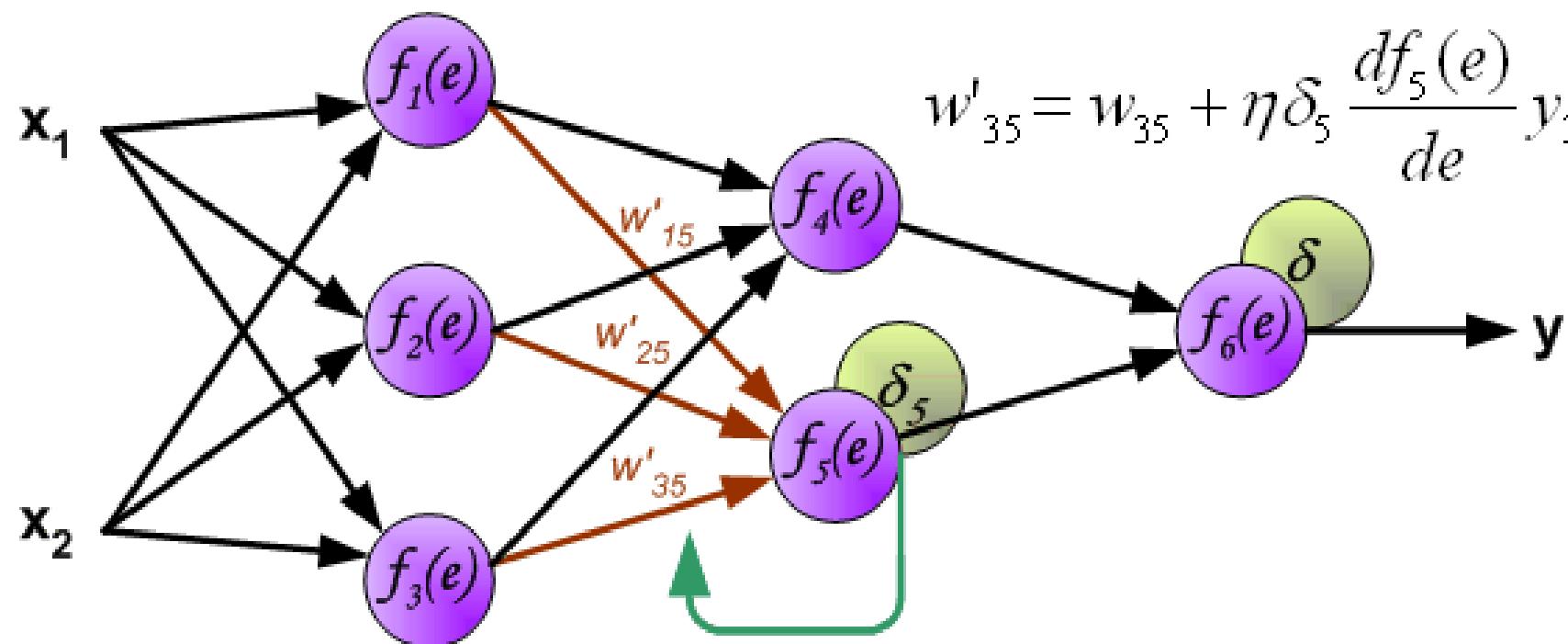
$$w'_{24} = w_{24} + \eta \delta_4 \frac{df_4(e)}{de} y_2$$



$$w'_{15} = w_{15} + \eta \delta_5 \frac{df_5(e)}{de} y_1$$

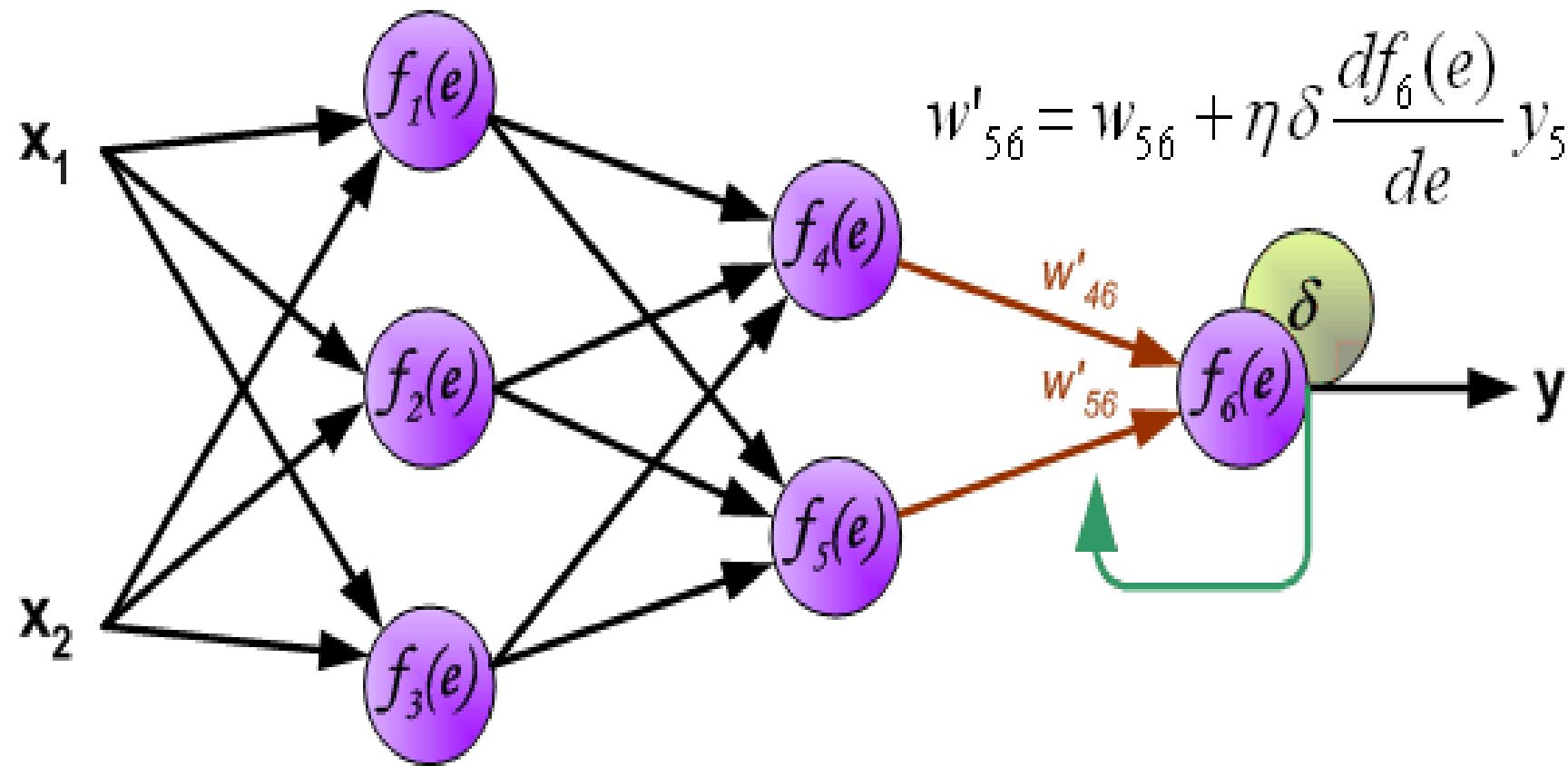
$$w'_{25} = w_{25} + \eta \delta_5 \frac{df_5(e)}{de} y_2$$

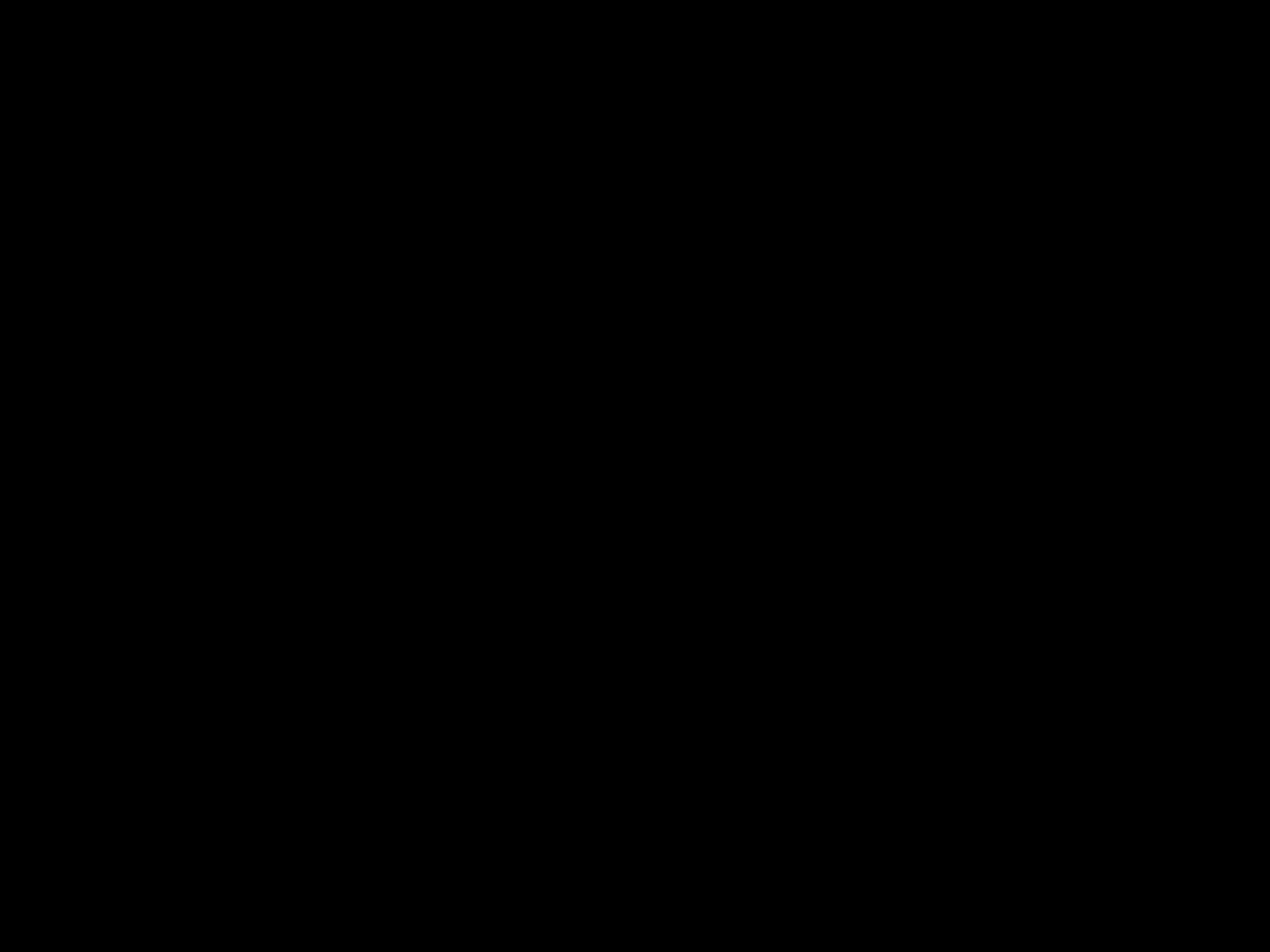
$$w'_{35} = w_{35} + \eta \delta_5 \frac{df_5(e)}{de} y_3$$



$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta \delta \frac{df_6(e)}{de} y_5$$





What's wrong with NN and back-propagation?

- It **requires a lot of labelled** training data
 - Almost all data is unlabelled
- The learning time does not scale well
 - It is **very slow** in networks with multiple hidden layers
- It can get stuck in poor **local optimas**
 - Works well most of the time, but suffers when complexity...

Use unsupervised
generative methods

Bring out the GPUs!

Use more robust
activation functions
like RELU, dropout
layers, etc.

sklearn: Neural Network

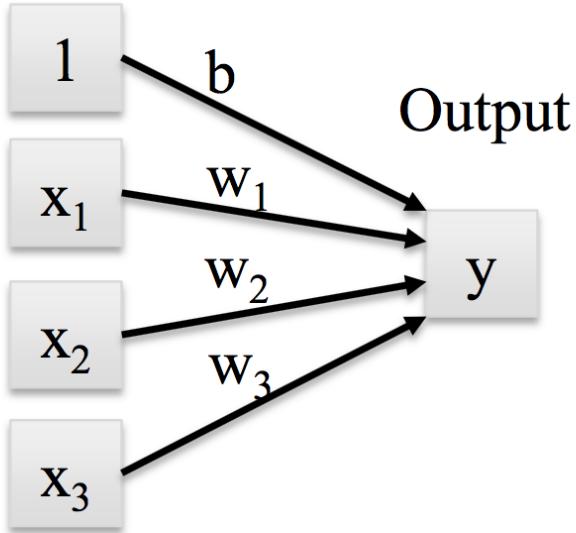
sklearn

- Class `MLPClassifier` implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation.
- MLP trains using Stochastic Gradient Descent, Adam, or L-BFGS. Stochastic Gradient Descent (SGD) updates parameters using the gradient of the loss function with respect to a parameter that needs adaptation.
- Multi-layer Perceptron is sensitive to feature scaling, so it is highly recommended to scale your data.
- L-BFGS converges faster and with better solutions on small datasets. For relatively large datasets, however, Adam is very robust.

Review: Linear and Logistic Regression

Linear regression

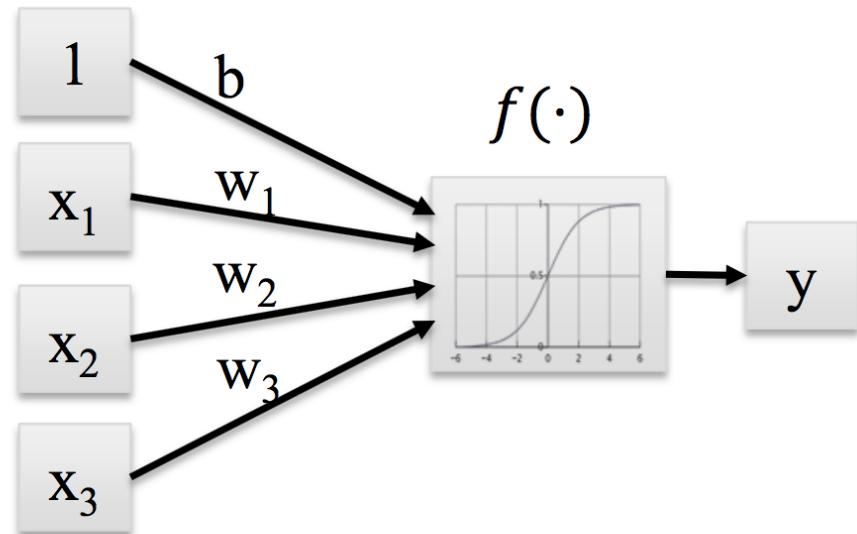
Input features



$$\hat{y} = \hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n$$

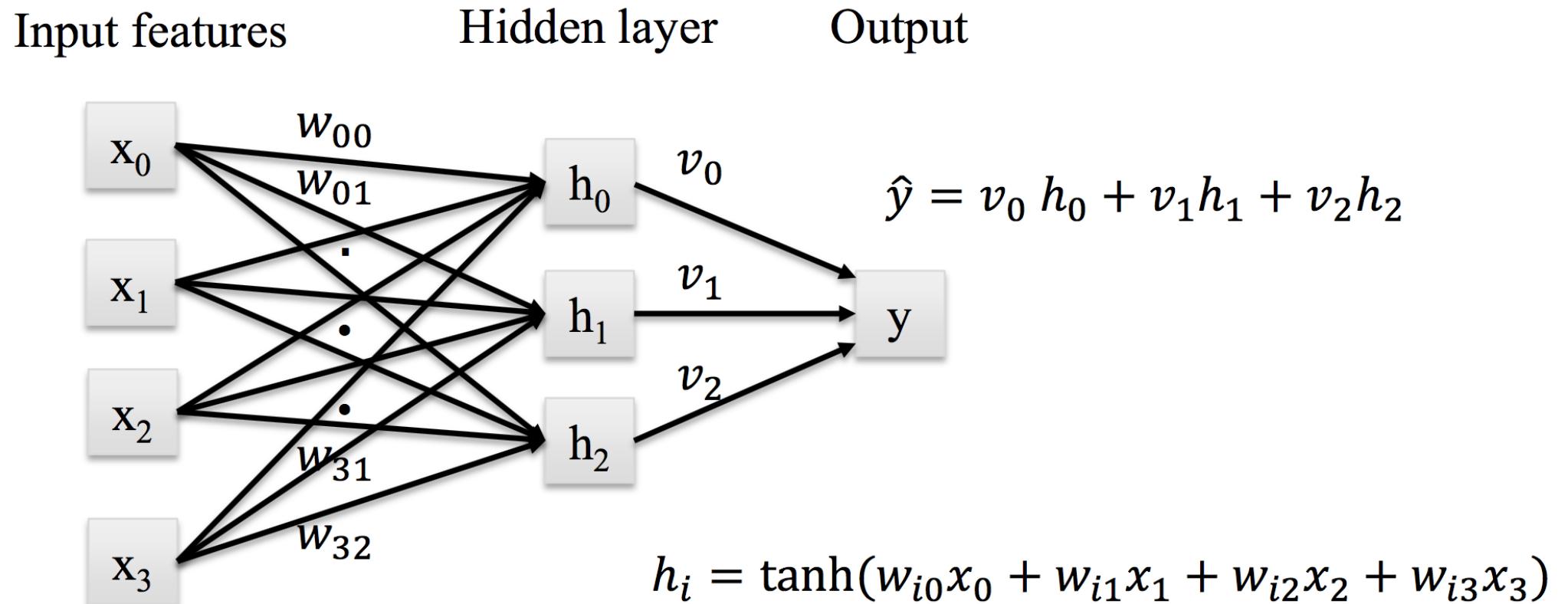
Logistic regression

Input features

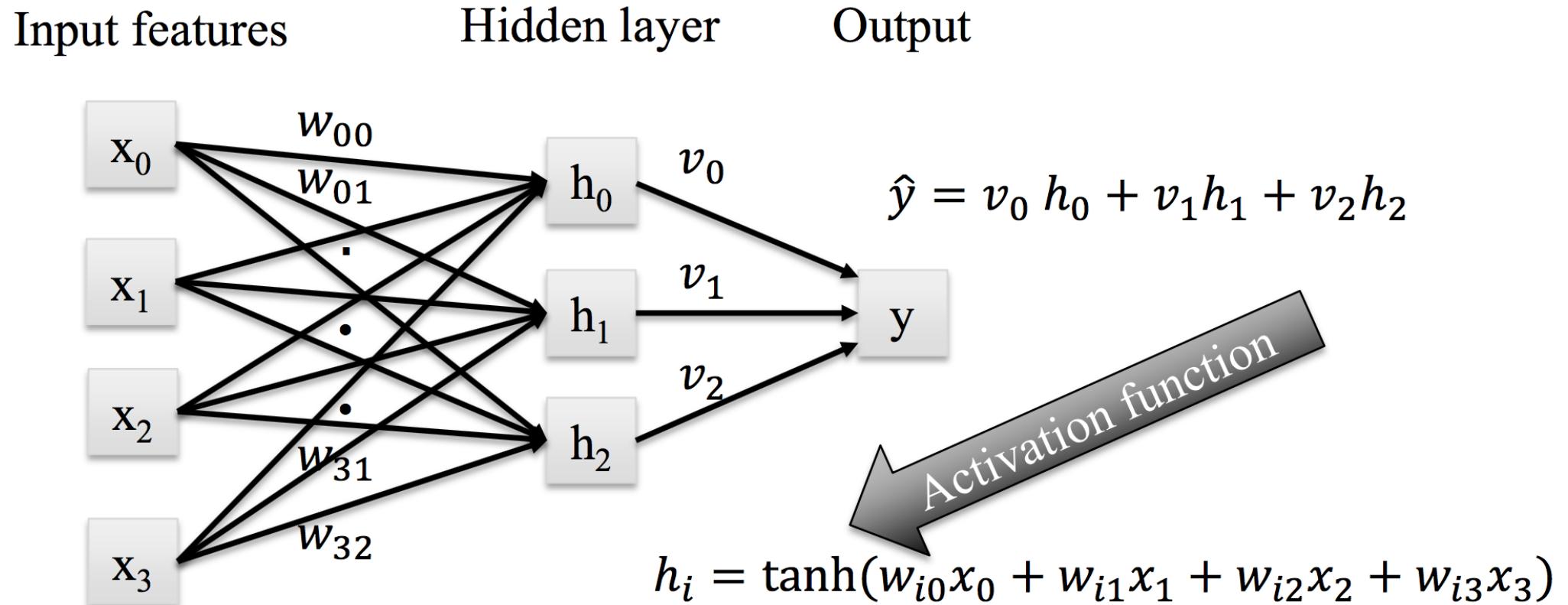


$$\hat{y} = \text{logistic}(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)$$

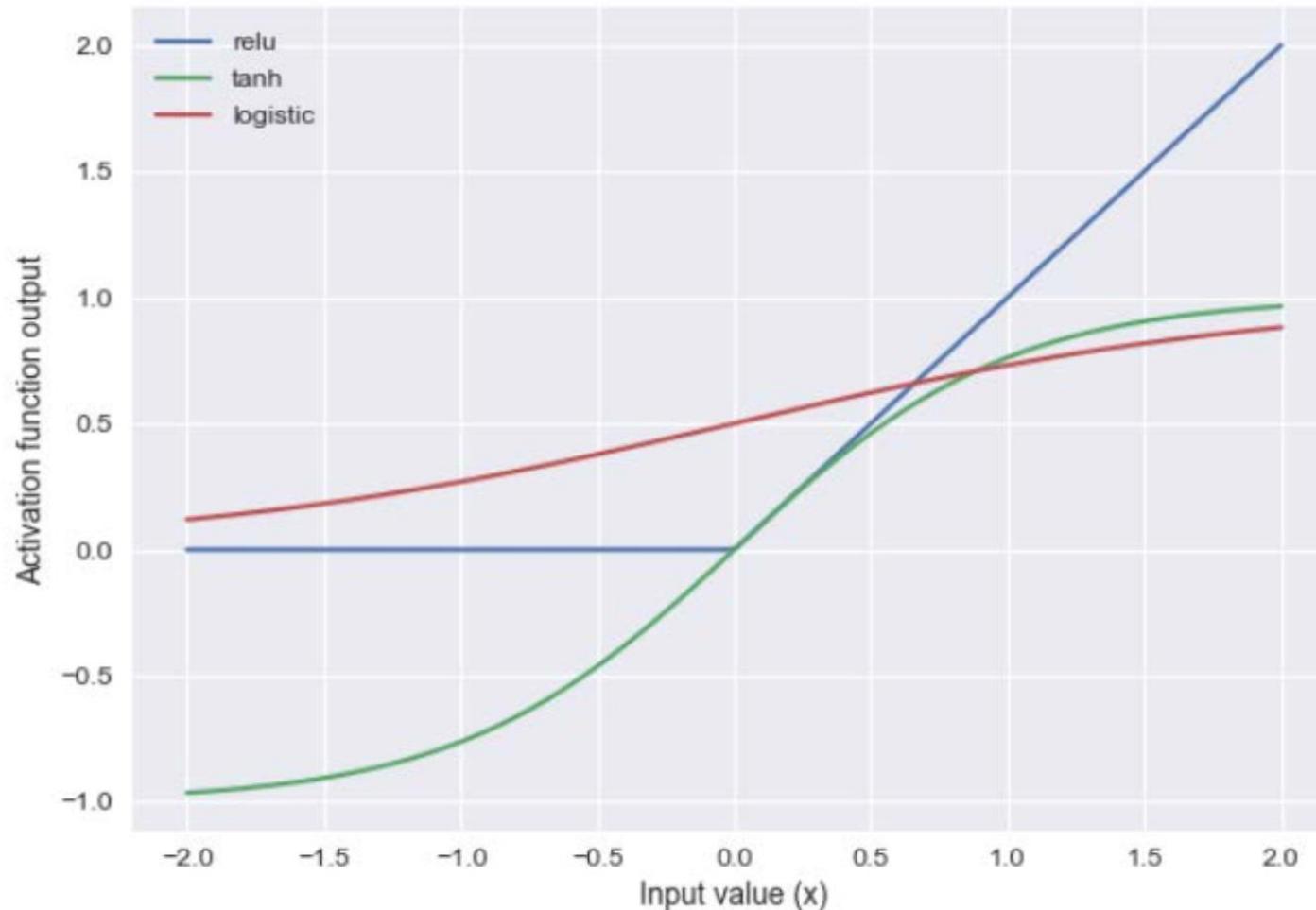
Multi-layer Perceptron with One Hidden Layer (and tanh activation function)



Multi-layer Perceptron with One Hidden Layer (and tanh activation function)



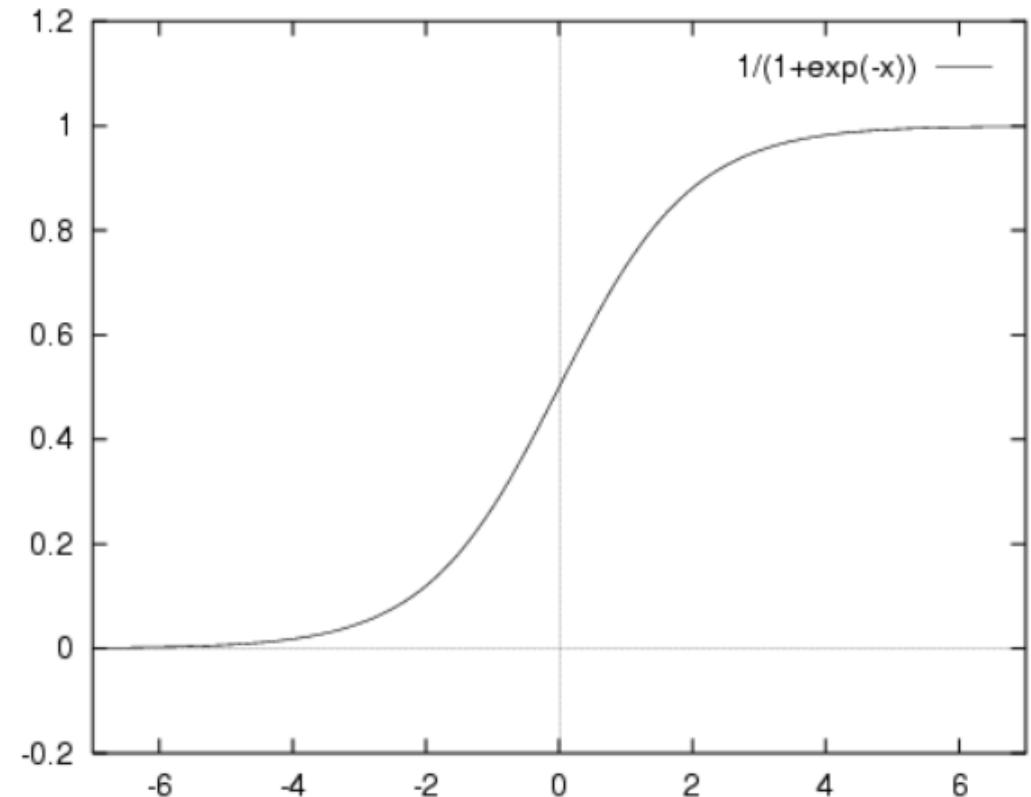
Activation Functions



$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

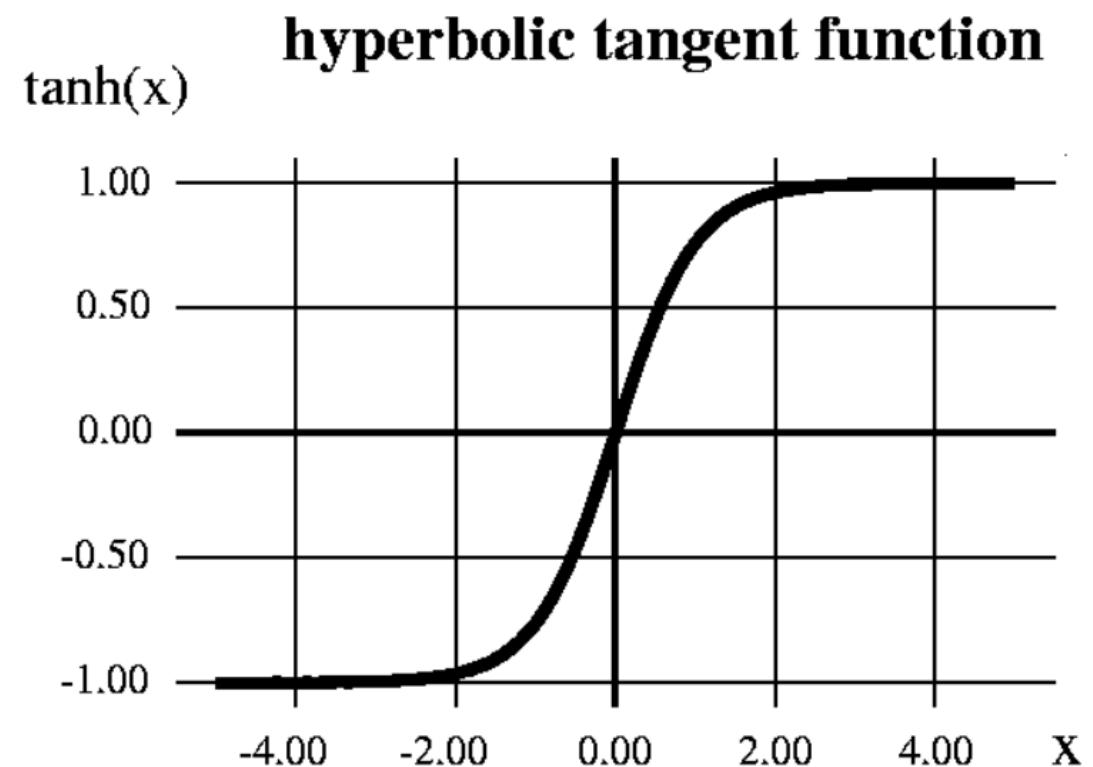
Activation Functions: Logistic/Sigmoid

- Sigmoid Activation function: It is a activation function of form $f(x) = 1 / (1 + \exp(-x))$. Its Range is between 0 and 1. It is a S—shaped curve. It is easy to understand and apply but it has major reasons which have made it fall out of popularity -
 - Vanishing gradient problem
 - Secondly , its output isn't zero centered. It makes the gradient updates go too far in different directions. $0 < \text{output} < 1$, and it makes optimization harder.
 - Sigmoids saturate and kill gradients.
 - Sigmoids have slow convergence.



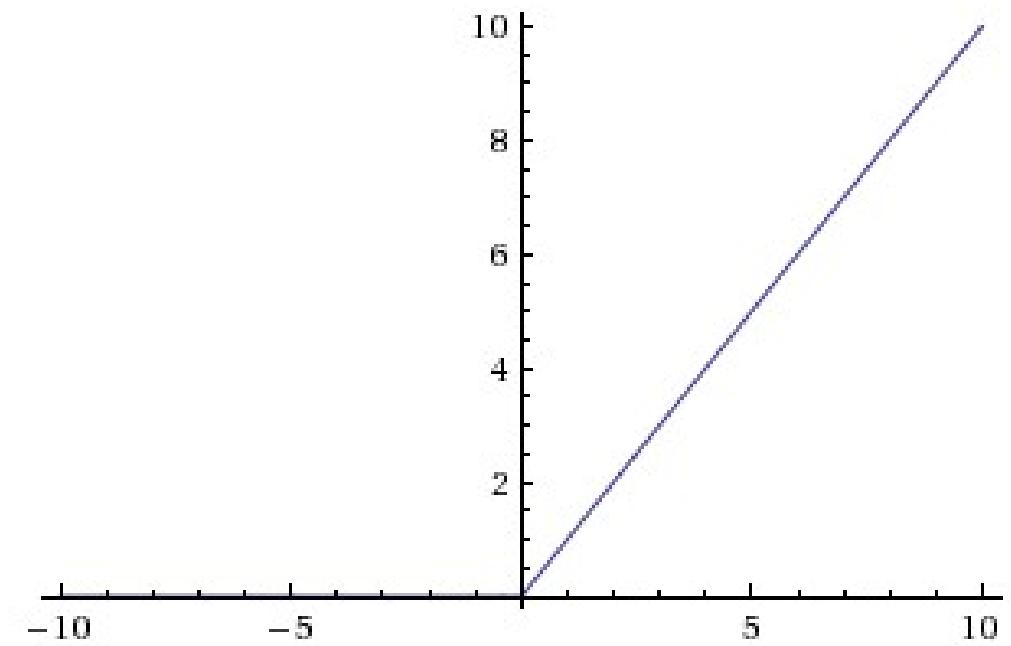
Activation Functions: tanh

- ***Hyperbolic Tangent function-***
Tanh : It's mathematical formula is $f(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$.
- Now it's output is zero centered because its range is between -1 to 1 i.e $-1 < \text{output} < 1$.
- Hence optimization is *easier* in this method hence in practice it is always preferred over Sigmoid function .
- *But still it suffers from Vanishing gradient problem.*

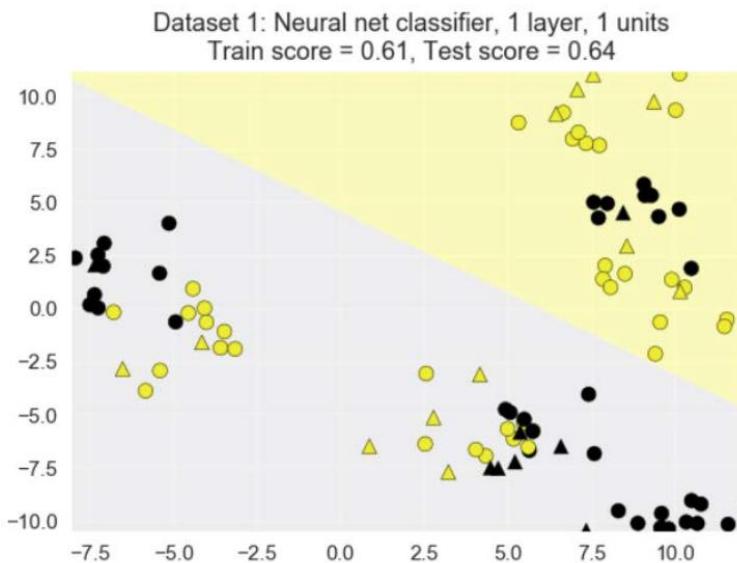


Activation Functions: ReLu

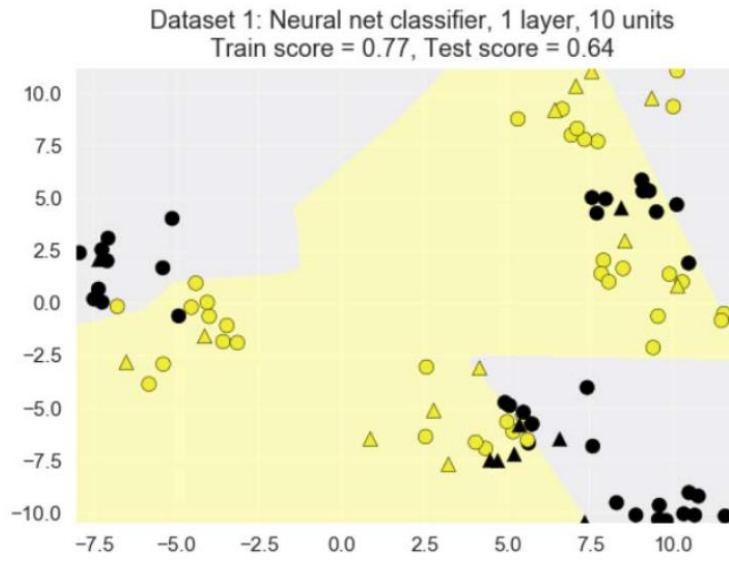
- ***ReLU- Rectified Linear units*** : It has become very popular in the past couple of years.
- It was recently proved that it had *6 times improvement in convergence* from Tanh function.
- It's just $R(x) = \max(0, x)$
- *i.e if $x < 0$, $R(x) = 0$*
- *and if $x \geq 0$, $R(x) = x$.*
- **But its limitation is that it should only be used within Hidden layers of a Neural Network Model.**



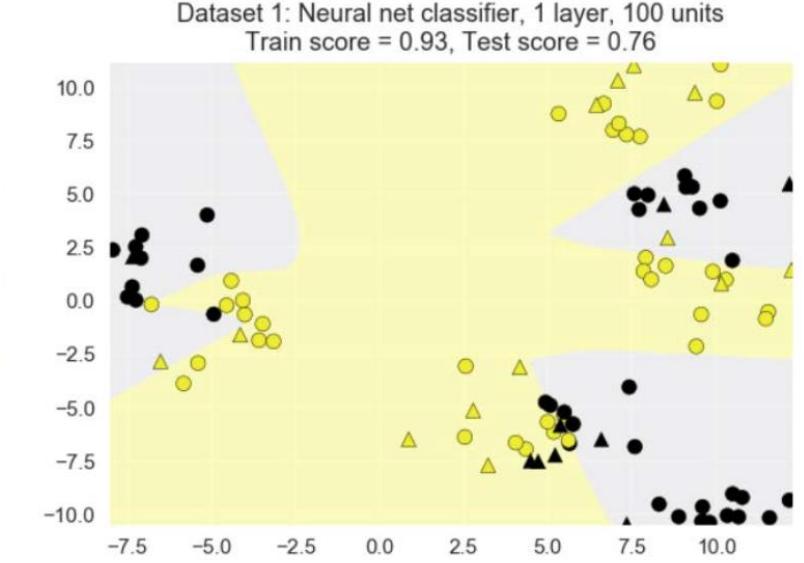
A single hidden layer network using 1, 10, or 100 units



1 layer, 1 unit

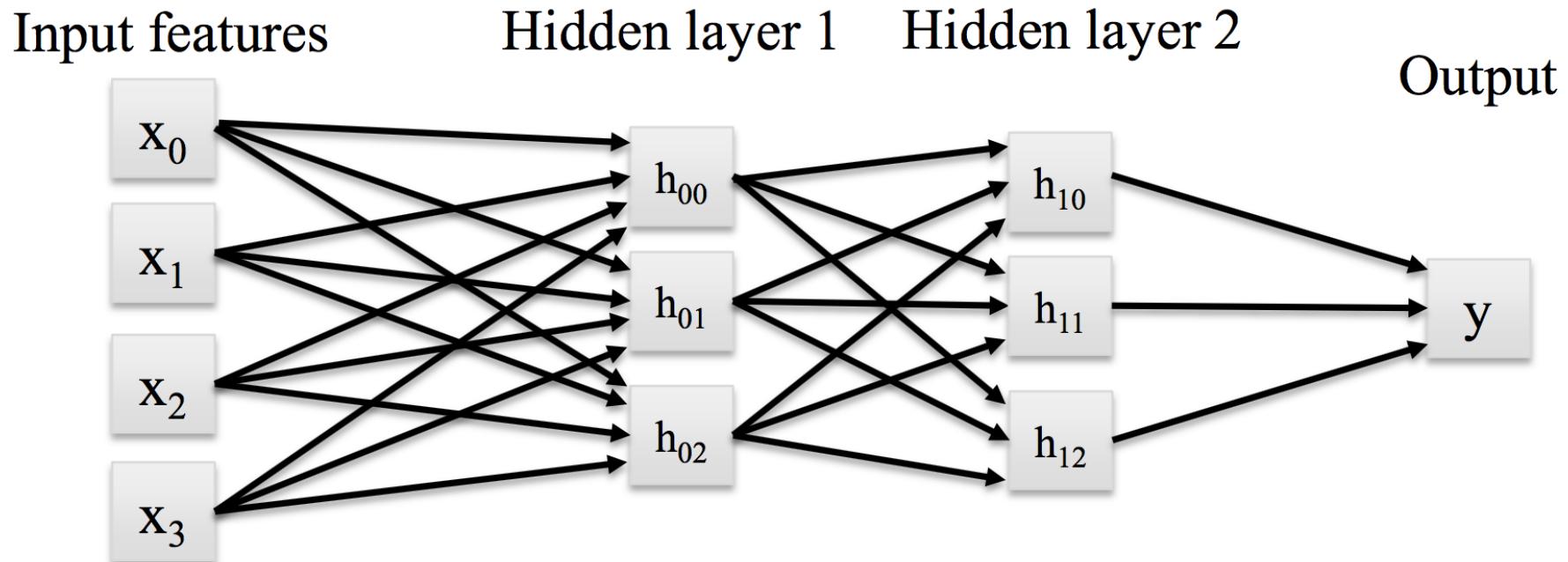


1 layer, 10 units

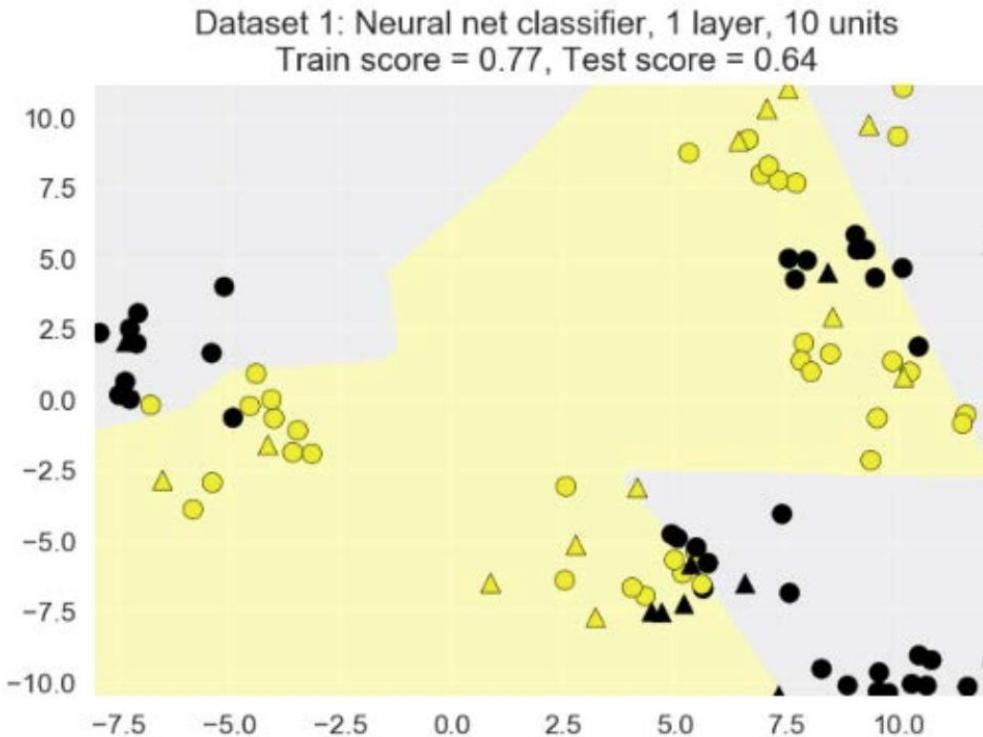


1 layer, 100 units

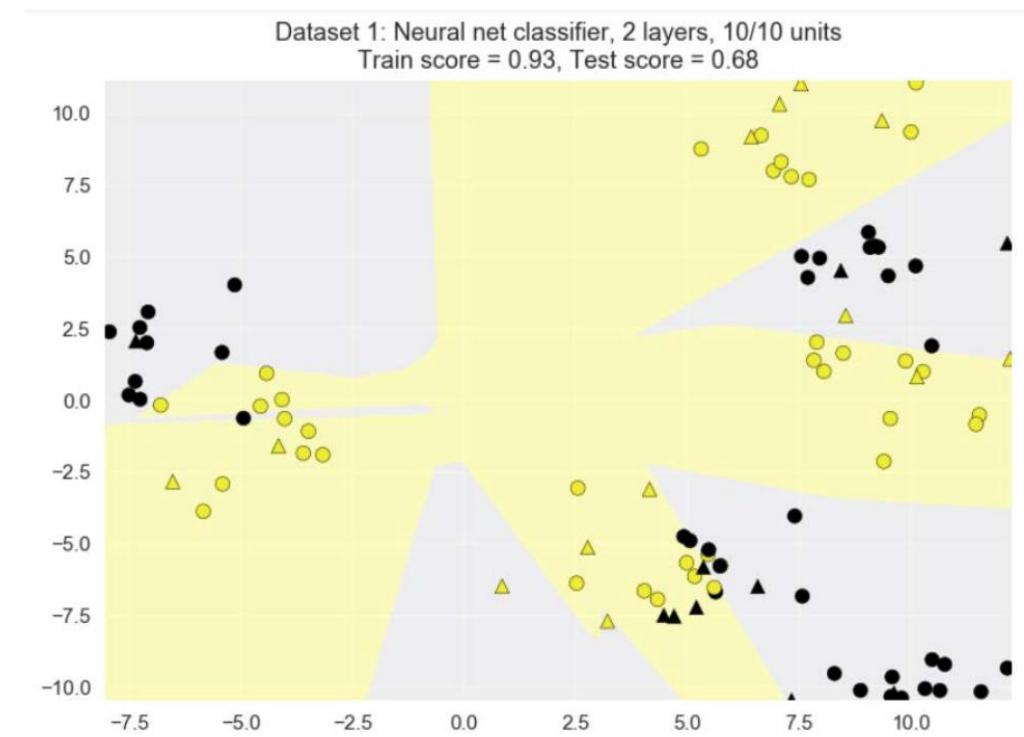
Multi-layer Perceptron with Two Hidden Layers



One vs Two Hidden Layers

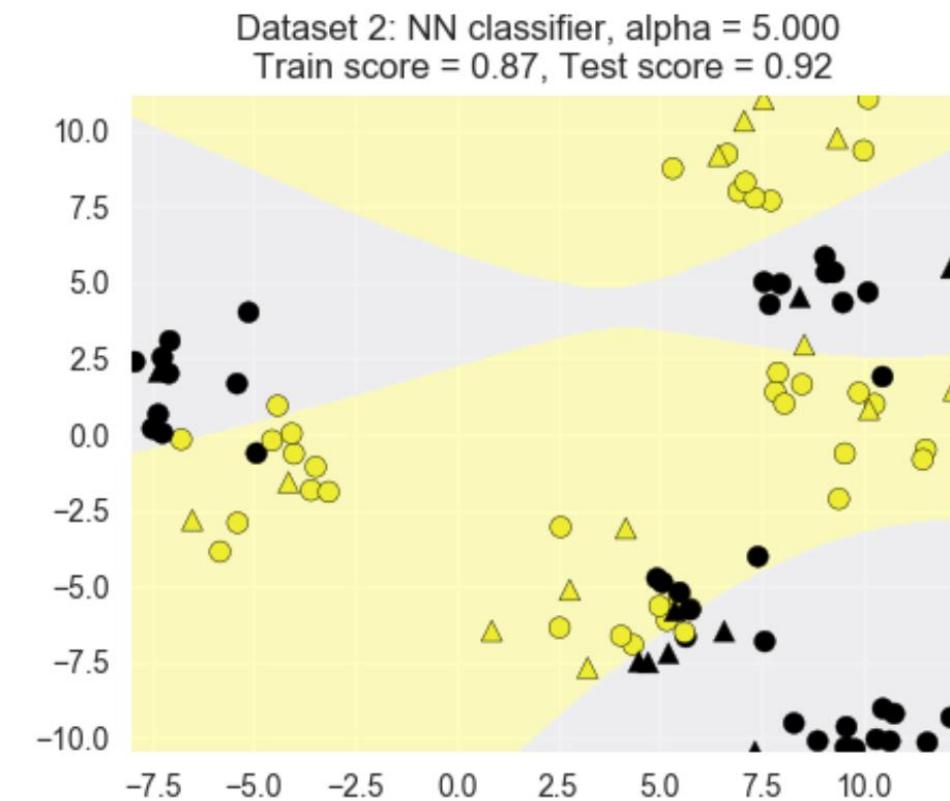
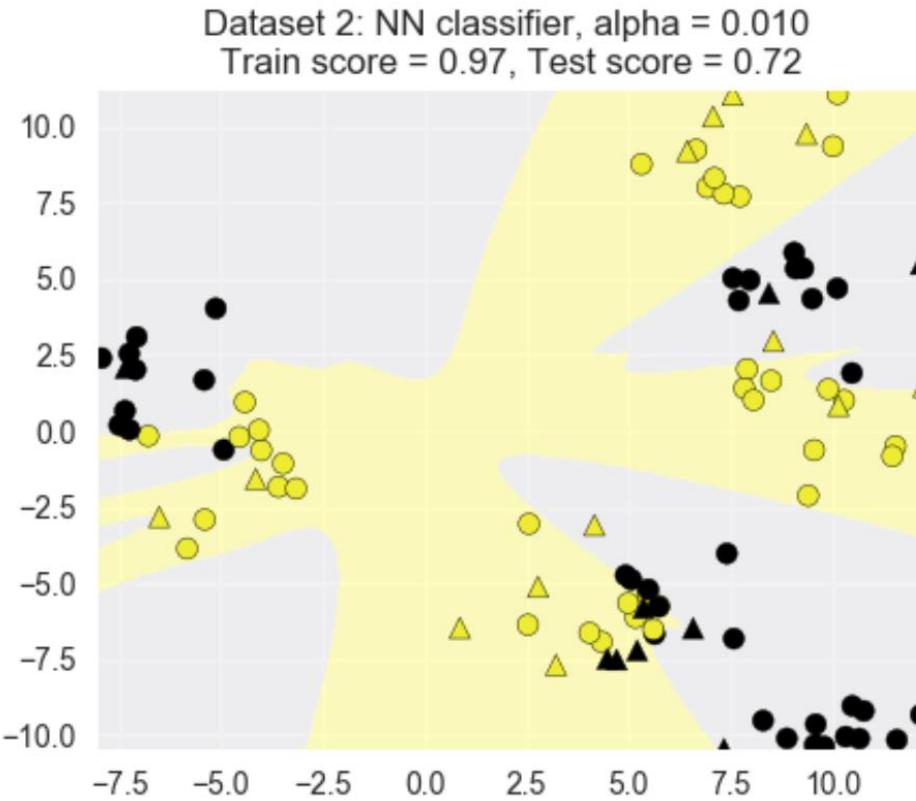


1 layer, 10 units



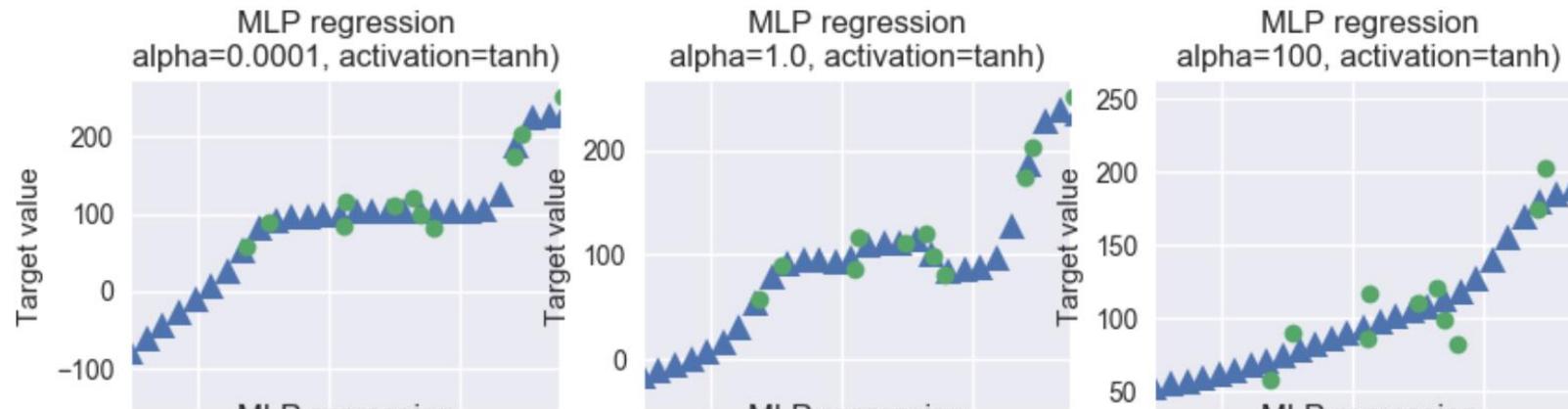
2 layers, (10, 10) units

L2 Regularization with the Alpha Parameter

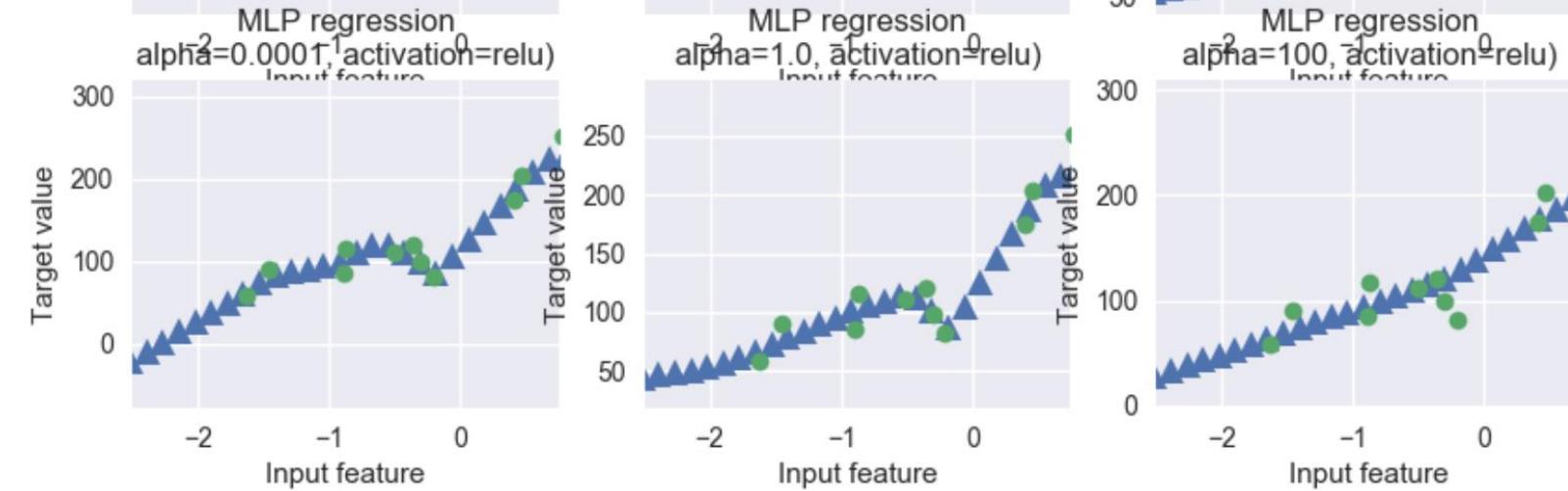


Neural Network Regression with MLPRegressor

activation = 'tanh'



activation = 'relu'



alpha = 0.0001

alpha = 1.0

alpha = 100.0

Increasing regularization

Epoch
000,562Learning rate
0.03Activation
ReLURegularization
L2Regularization rate
0.03Problem type
Classification

DATA

Which dataset do you want to use?



Ratio of training to test data: 60%

Noise: 5

Batch size: 20

REGENERATE

FEATURES

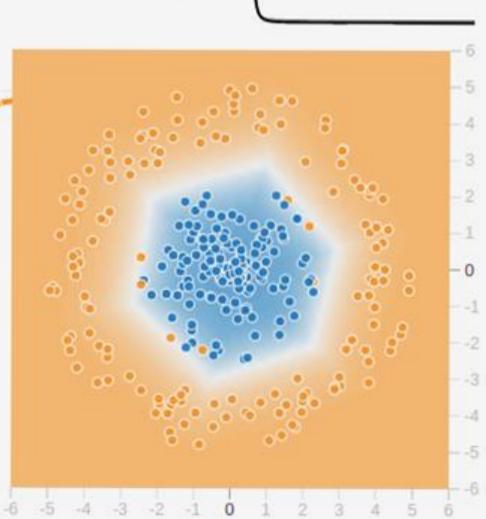
Which properties do you want to feed in?

 x_1  x_2  x_1^2  x_2^2  $x_1 x_2$  $\sin(x_1)$  $\sin(x_2)$

3 HIDDEN LAYERS



OUTPUT

Test loss 0.035
Training loss 0.044

Colors shows data, neuron and weight values.
 Show test data Discretize output

Neural Networks: Pros and Cons

Pros:

- They form the basis of state-of-the-art models and can be formed into advanced architectures that effectively capture complex features given enough data and computation.

Cons:

- Larger, more complex models require significant training time, data, and customization.
- Careful preprocessing of the data is needed.
- A good choice when the features are of similar types, but less so when features of very different types.

Neural Nets: MLPClassifier and MLPRegressor

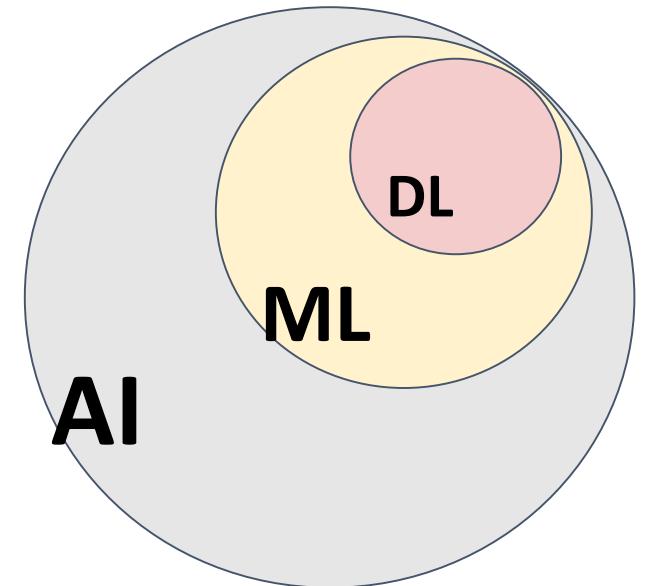
Important Parameters

- **hidden_layer_sizes**: sets the number of hidden layers (number of elements in list), and number of hidden units per layer (each list element). *Default: (100).*
- **alpha**: controls weight on the regularization penalty that shrinks weights to zero. *Default: alpha = 0.0001.*
- **activation**: controls the nonlinear function used for the activation function, including: '**relu**' (default), '**logistic**', '**tanh**'.

Deep Learning

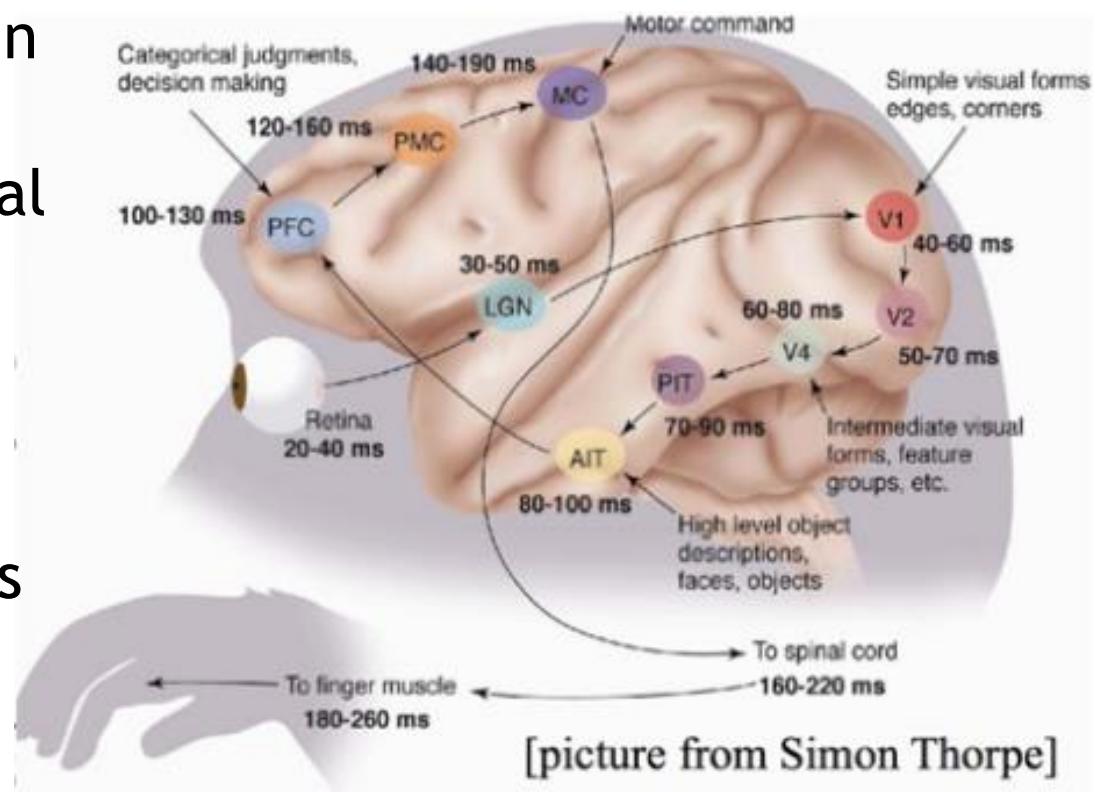
Deep Learning

- In the past 10 years, Artificial Intelligence (AI) particularly ML have shown tremendous progress.
- **Deep Learning:** part of Machine Learning (ML) field of **learning representations of data**
 - Exceptional effectiveness
 - Hierarchy of multiple “learning” layers that mimic the neurons in our brain
 - Thrives on big amounts of data



Why learn “deeply”?

- Inspired by architectural depth of brain
- Neuroscience findings: Information becomes more complex down the visual pipeline
- No successful attempts were reported before 2006
 - SVM: Vapnik and co. developed Support Vector Machine which was pretty good, but shallow learning

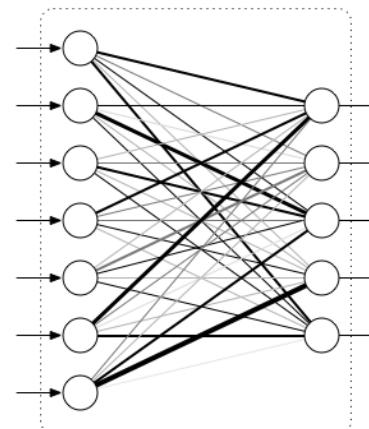


2006 Breakthrough: What made Deep Learning possible?

- Explosion of data
- Faster and cheaper computing -- multi-core CPUs and GPUs
- Improvement of ML models -- more complex, more scalable



Stacked Restricted Boltzmann Machines (RBM) or Deep Belief Nets (DBN), Hinton, 2006
Stacked Autoencoders (AE), Bengio, 2007



"I've worked all my life in Machine Learning, and I've never seen one algorithm knock over benchmarks like Deep Learning"

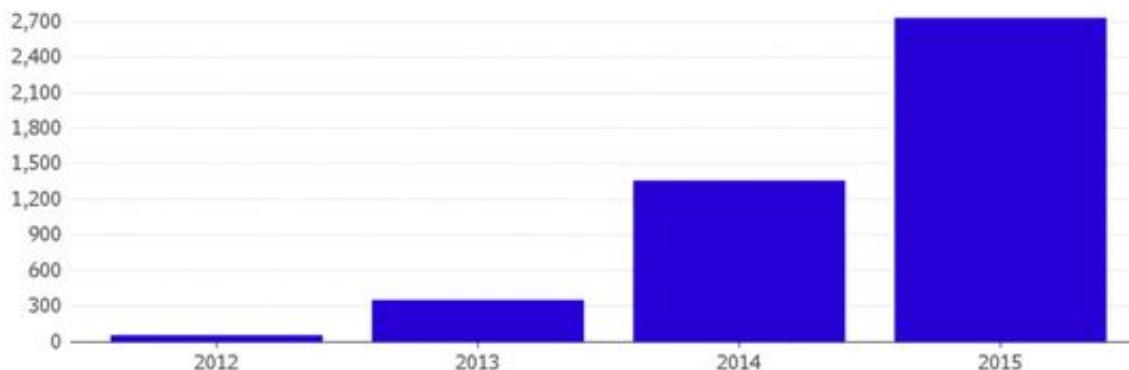
Andrew Ng
Stanford Univ, Baidu (prev.)
Co-Founder of Coursera



Deep Learning: Hype or Reality?

Artificial Intelligence Takes Off at Google

Number of software projects within Google that uses a key AI technology, called Deep Learning.



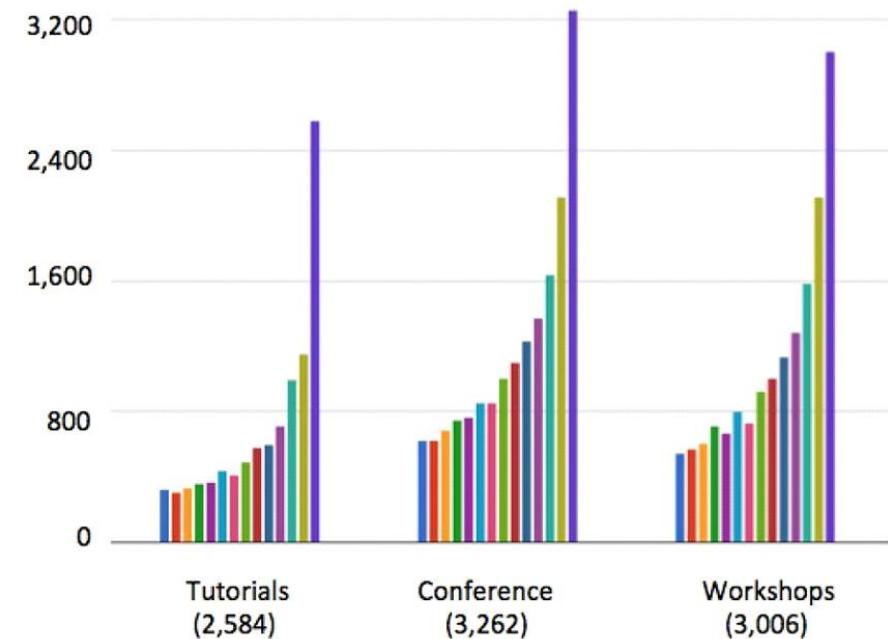
Source: Google

Note: 2015 data does not incorporate data from Q4

Bloomberg

NIPS Growth

Total Registrations 3755



A very brief history...



1958 Perceptron

1974 Backpropagation



Convolution Neural Networks for
Handwritten Recognition



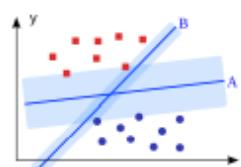
2012



1998



2006
Restricted
Boltzmann
Machine



1995
SVM reigns

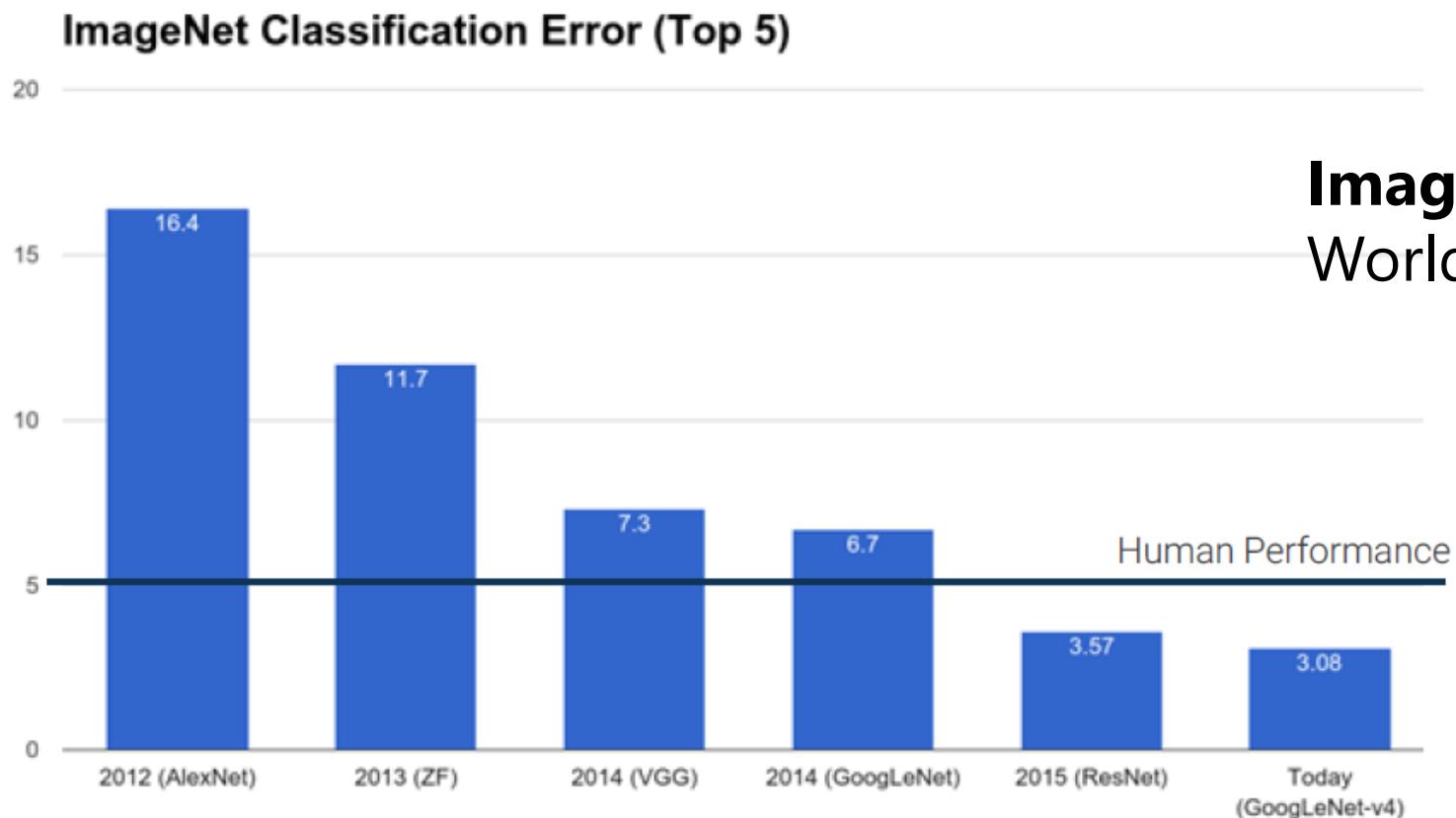


1969
Perceptron criticized

2012
AlexNet wins
ImageNet
IMAGENET

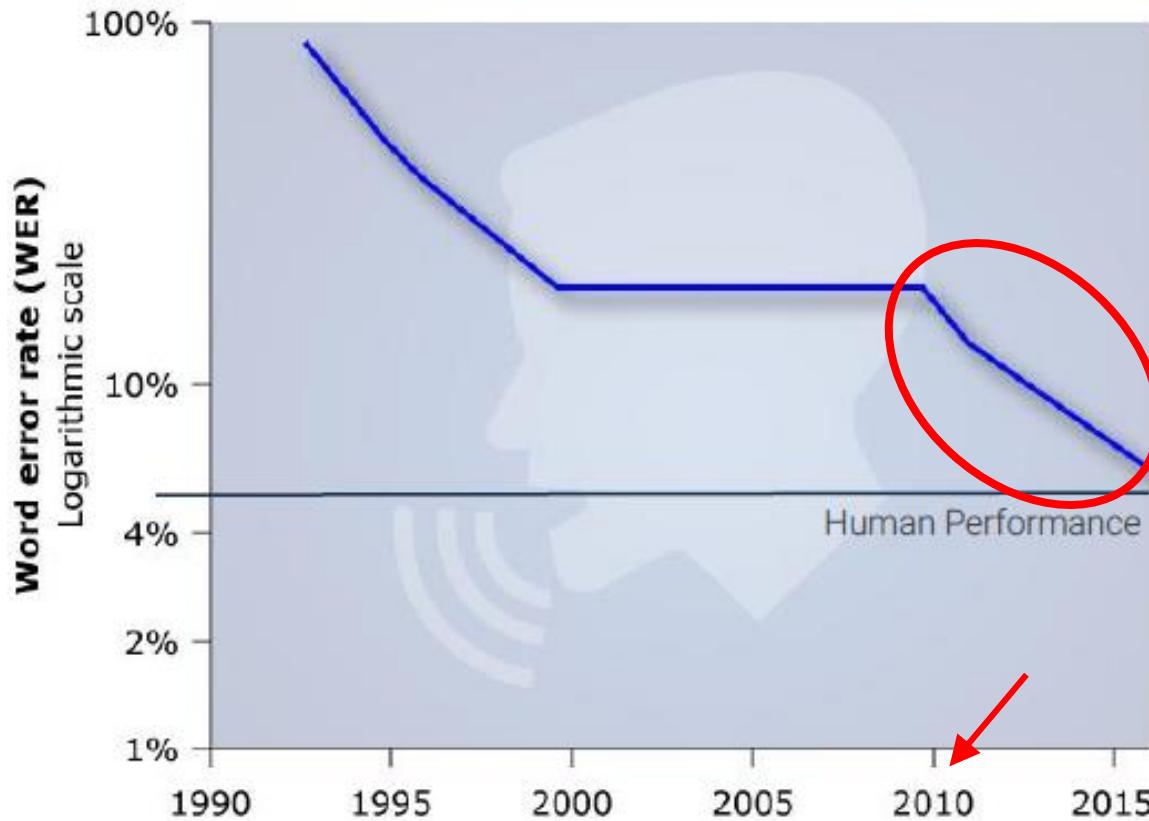
awkward silence (AI Winter)

“One algorithm to rule them all”



ImageNet: The “computer vision
World Cup”

“One algorithm to rule them all”

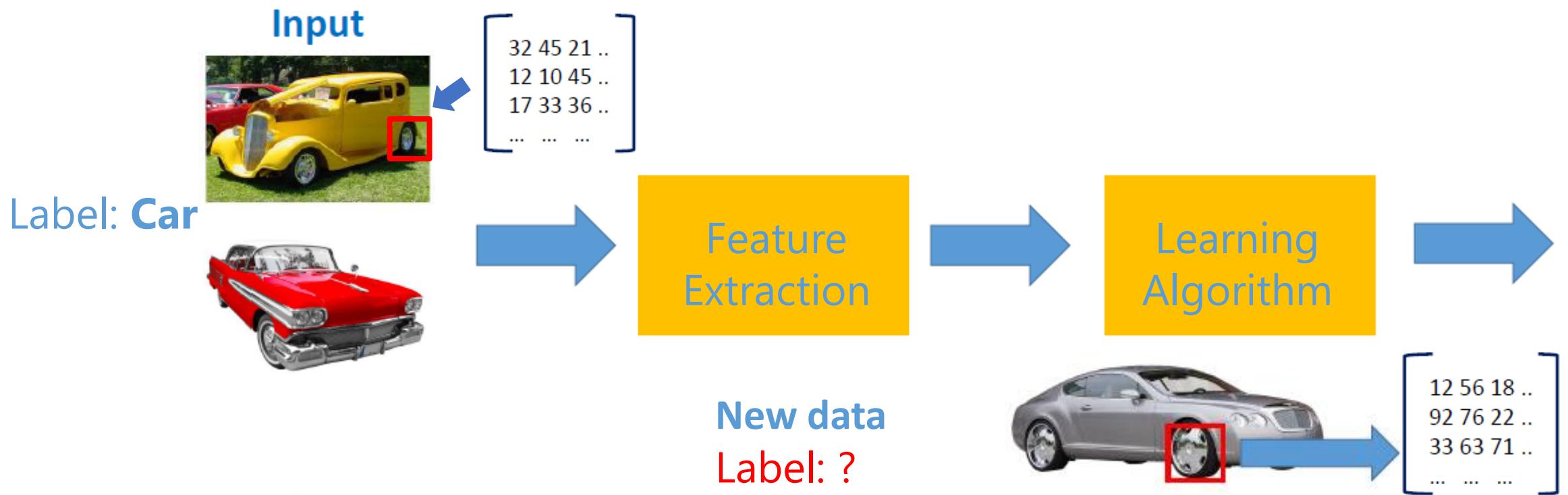


State-of-the-art Performance
for Speech Recognition

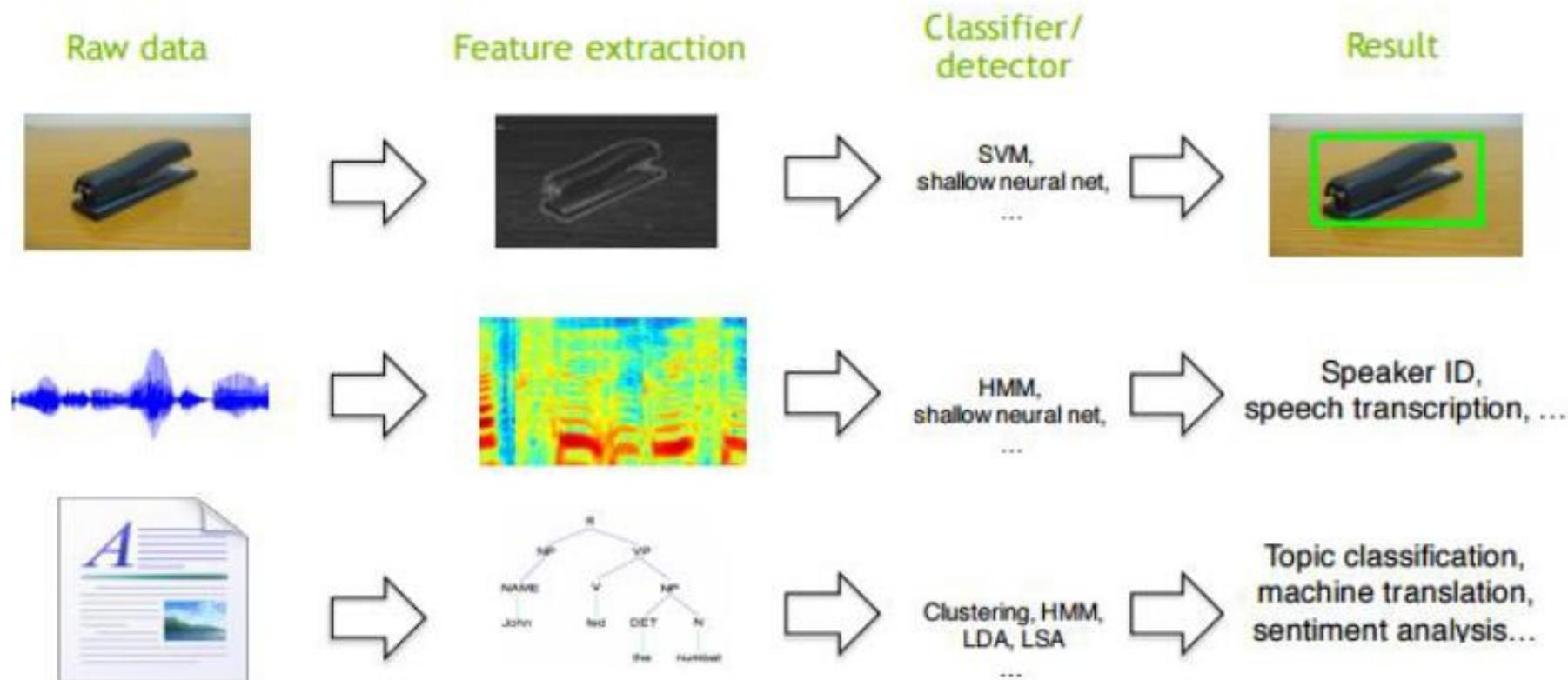
Thanks
to DL!

Revisiting Traditional Machine Learning

- **“Feature Engineering”:** Relates to extracting features by “hand-crafted” means. A simple classifier can be trained for recognition.

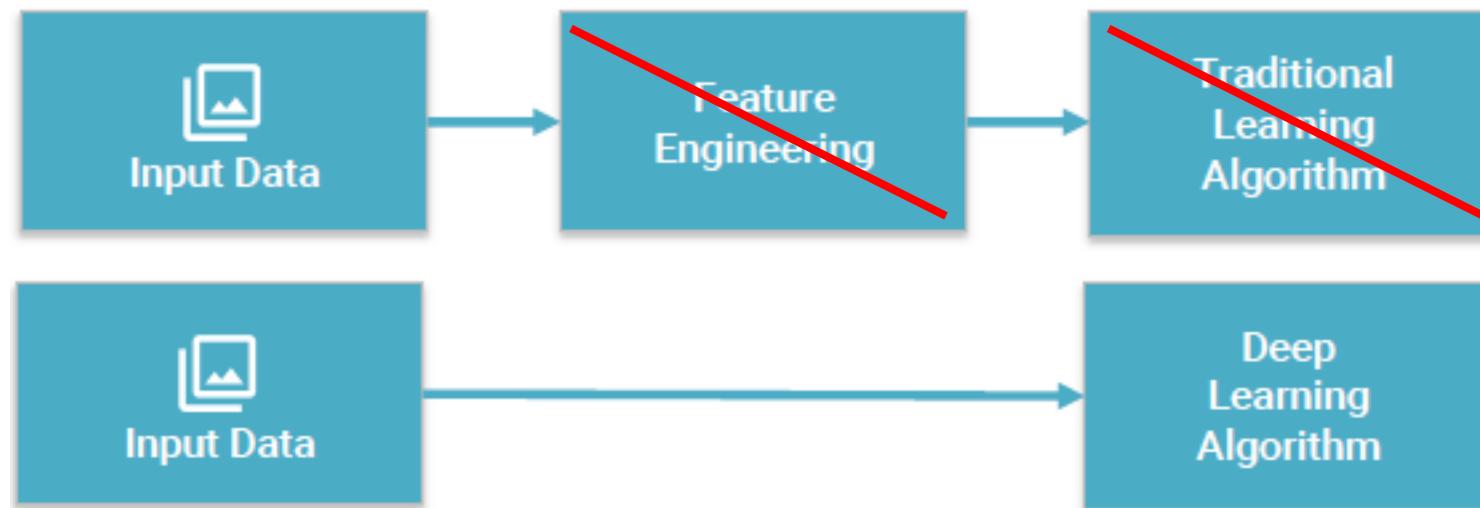


Revisiting Traditional Machine Learning

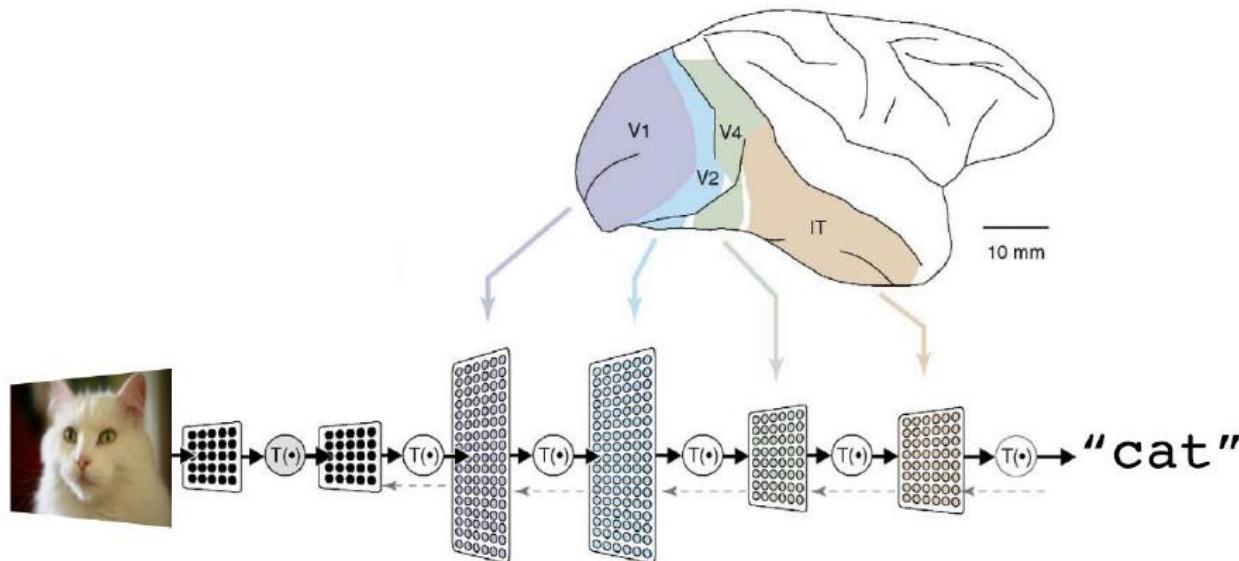


Deep Learning - All-in-one

- Removes the need to **manually craft** the feature (DL “**finds**” them instead)
- Removes the need to **perform separate learning** (DL “**learns**” them directly)



Why Go Deeper in Neural Networks?



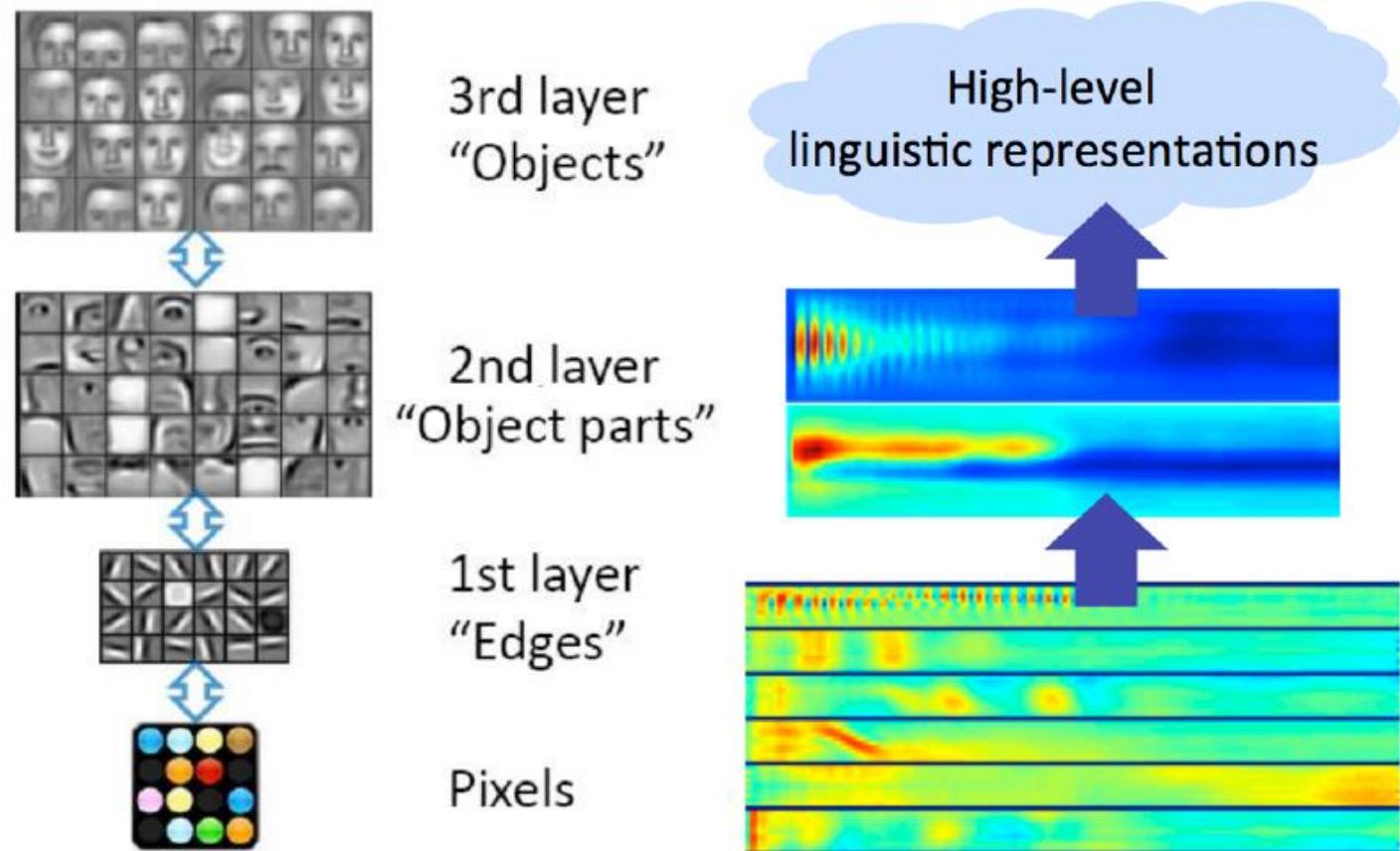
1. Biological Justification -- Hierarchical learning

- A deep neural network consists of a hierarchy of layers, whereby each layer transforms input data into more abstract representations. The output layer combines these features to make predictions

Why Go Deeper in Neural Networks?

2. Different levels of abstraction

- Natural progression from low to high level structure as seen in natural complexity

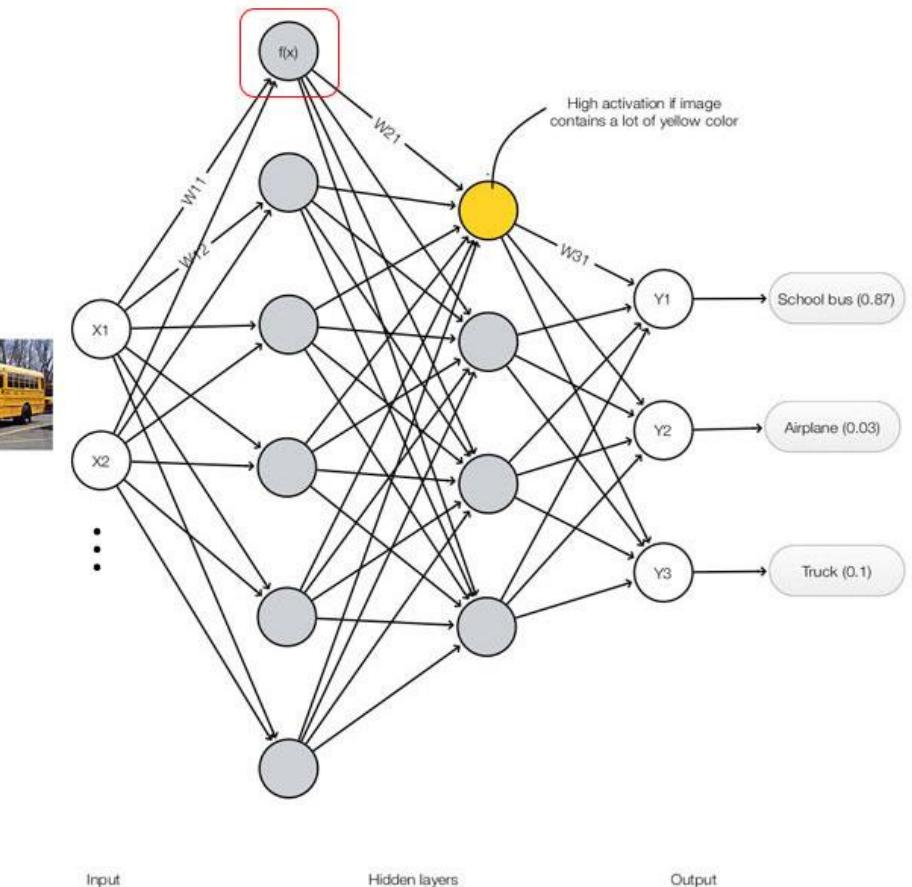


How “deep” is a deep neural network?

- Credit Assignment Path (CAP)
or (number of hidden layers +
1) more than 2
- CAP > 10 : Very deep



“There is no universally agreed upon threshold of depth dividing shallow learning from deep learning, but most researchers in the field agree that deep learning has multiple nonlinear layers (CAP > 2) and Schmidhuber considers CAP > 10 to be very deep learning”



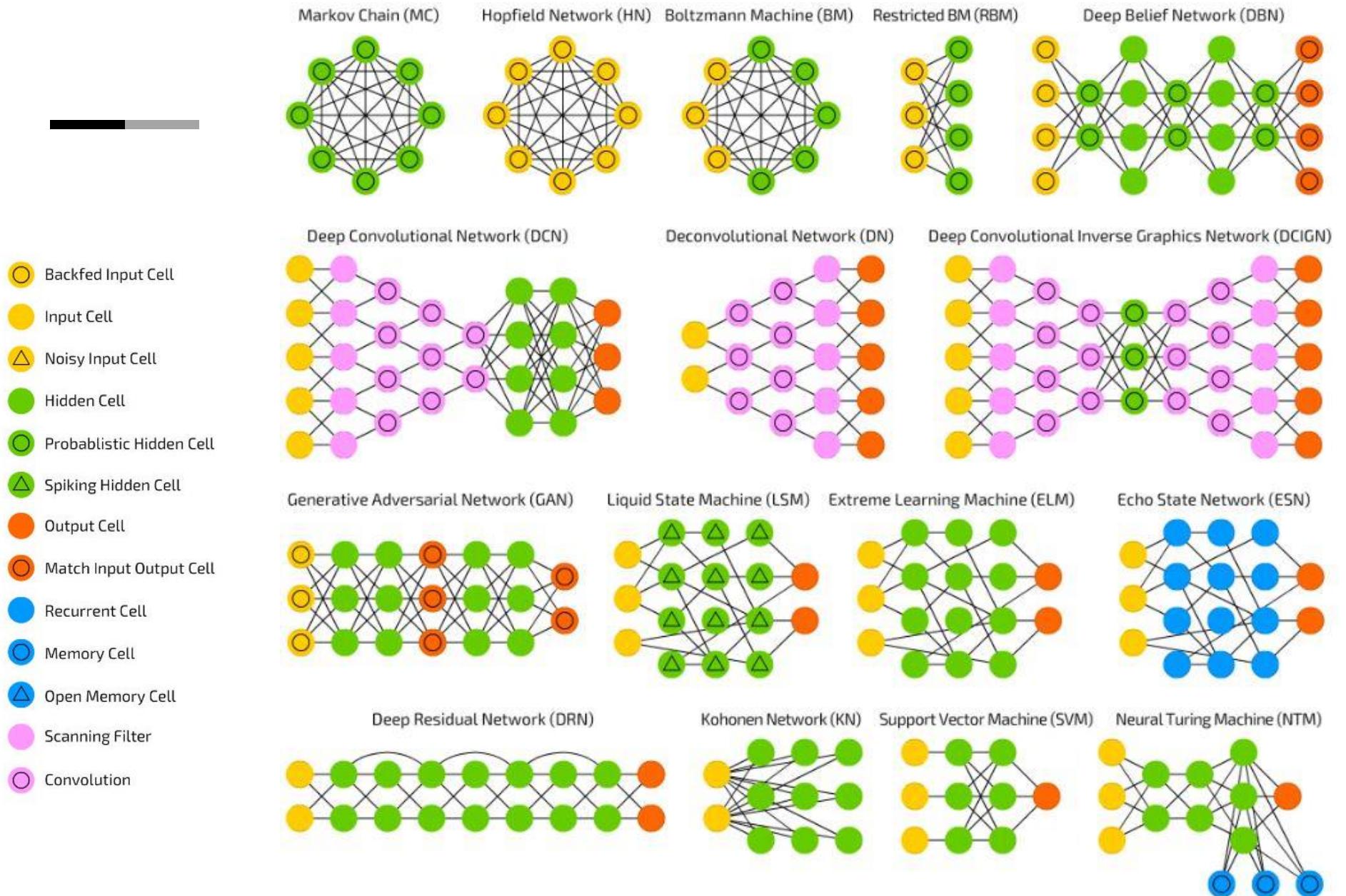
Deep Learning: Strengths

- **Robust**
 - Features are automatically learned to be optimal for the task at hand
- **Generalizable**
 - The same DL approach/architecture can be used for many different applications and data types
- **Scalable**
 - Performance improves with more data, method is massively parallelizable

Deep Learning: Weaknesses

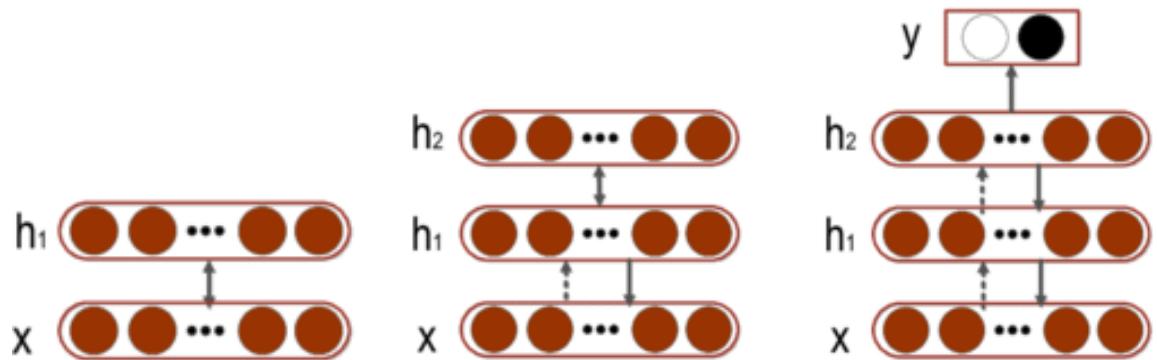
- **Requires large datasets (typically)**, also means long training periods.
- **Not necessarily > traditional ML methods** like SVMs, decision trees
- Learned features are often **difficult to understand or explain**.
 - Many vision features are also not really human-understandable (e.g, concatenations/combinations of different features).
- Requires **a good understanding of how to design new models** for new tasks, or using new forms of data, etc.

Deep Learning Flavours



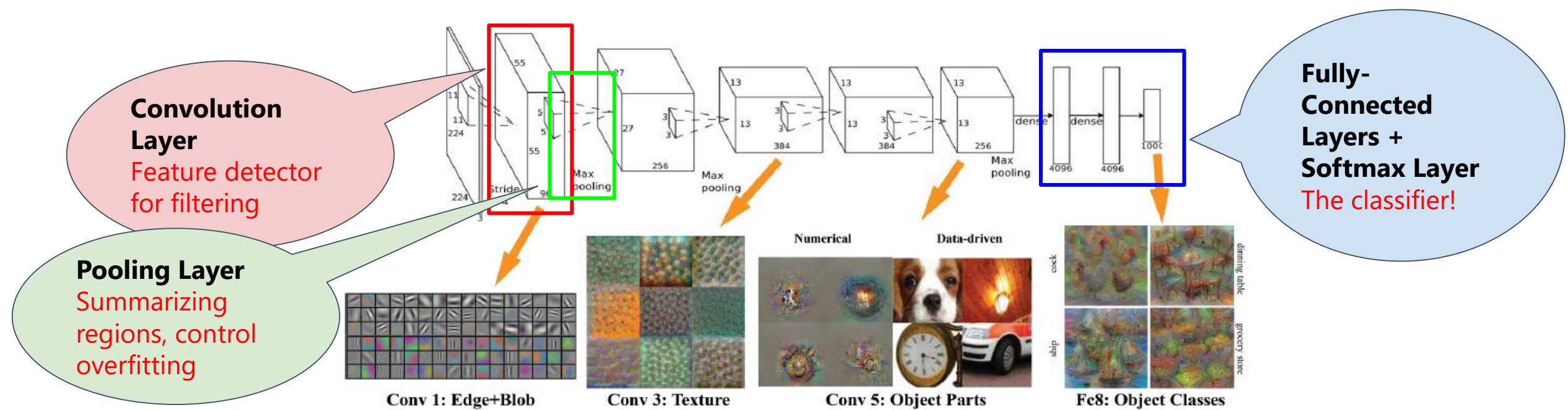
Stacking Neural Networks

- Early attempts, pioneered by Geoffrey Hinton:
 - Deep Belief Networks (DBN) or Stacked Restricted Boltzmann Machines (RBM),
 - Stacked AutoEncoders (SAE)
- Idea:
 - Layer-wise Unsupervised Pre-training
 - Train last layer in supervised mode



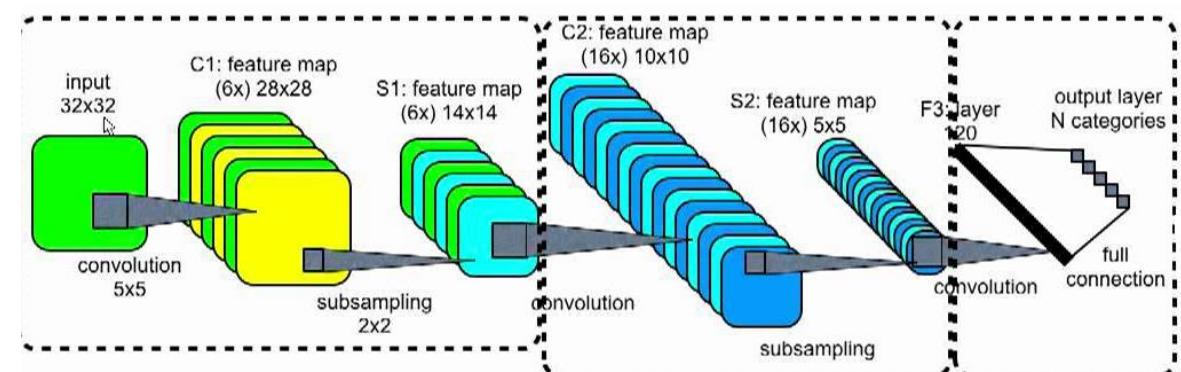
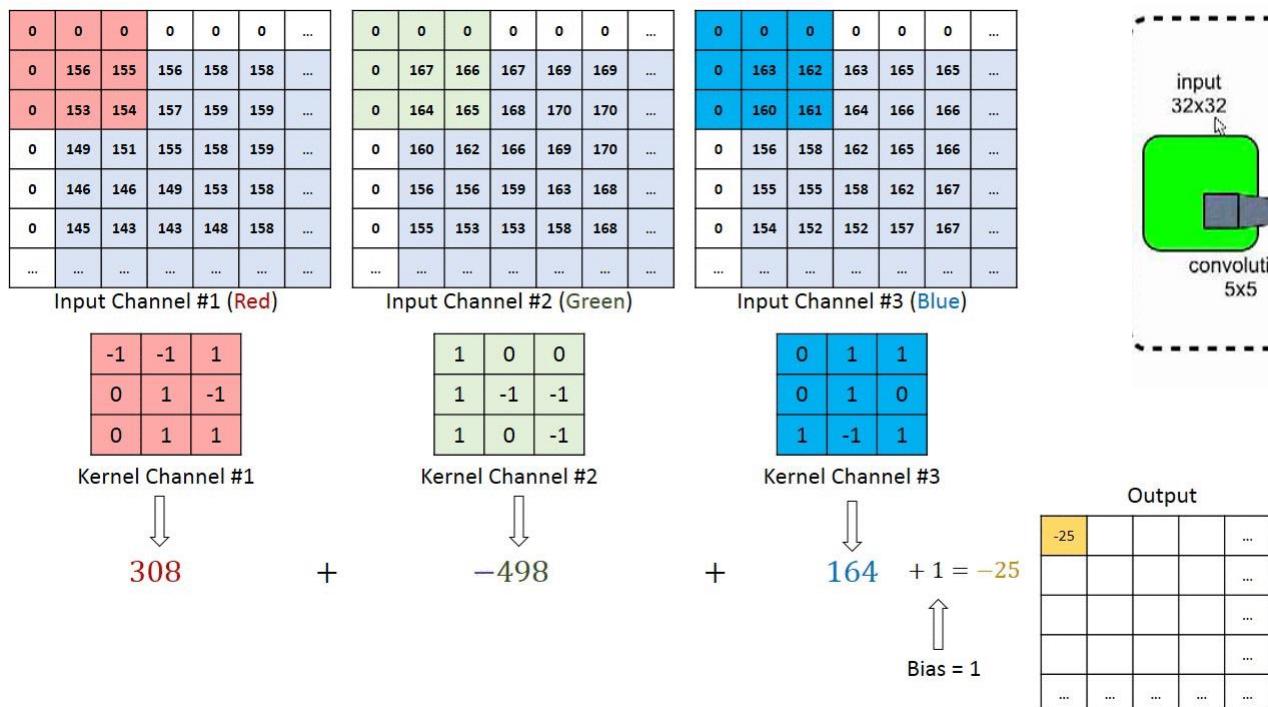
Convolutional Neural Networks (CNN)

- Learn multiple layers of transformations, stacked to extract a progressively more sophisticated representation of the input



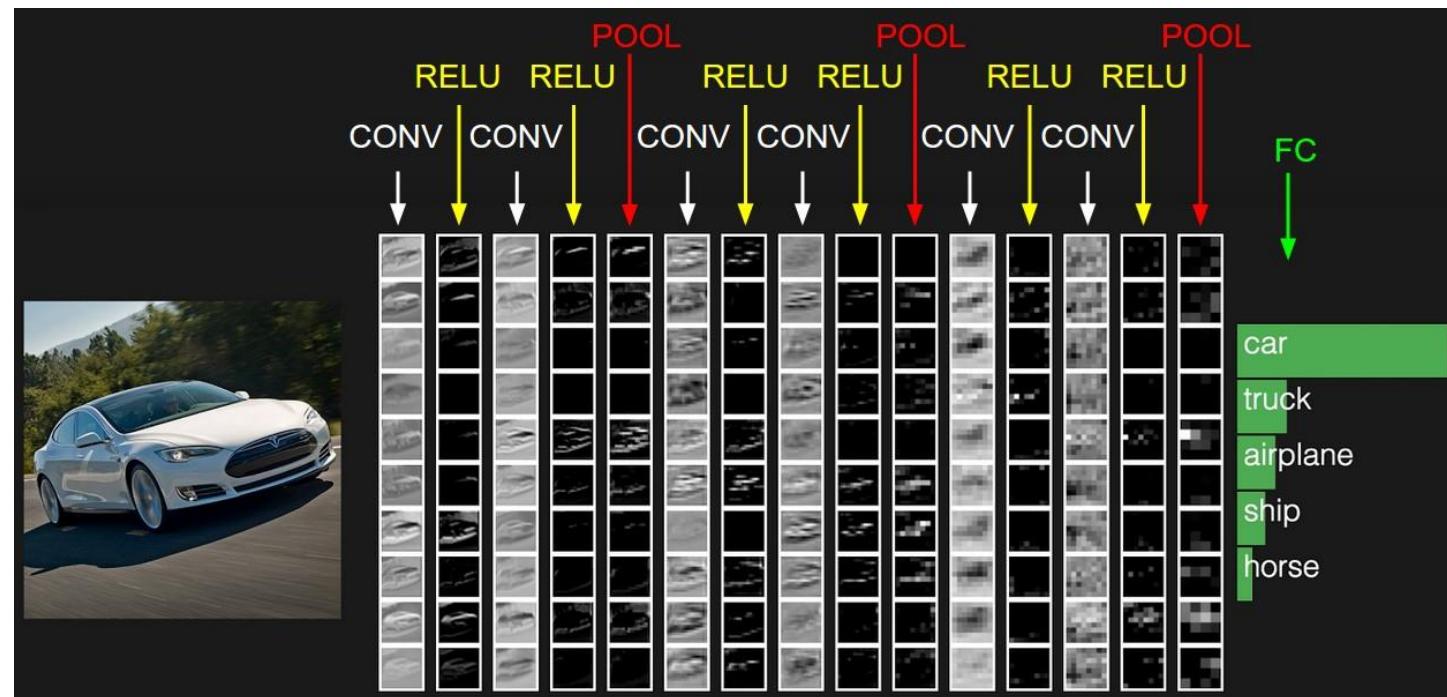
Convolutional Neural Networks (CNN)

- Each neuron in the Convolutional layer is mapped to a receptive field

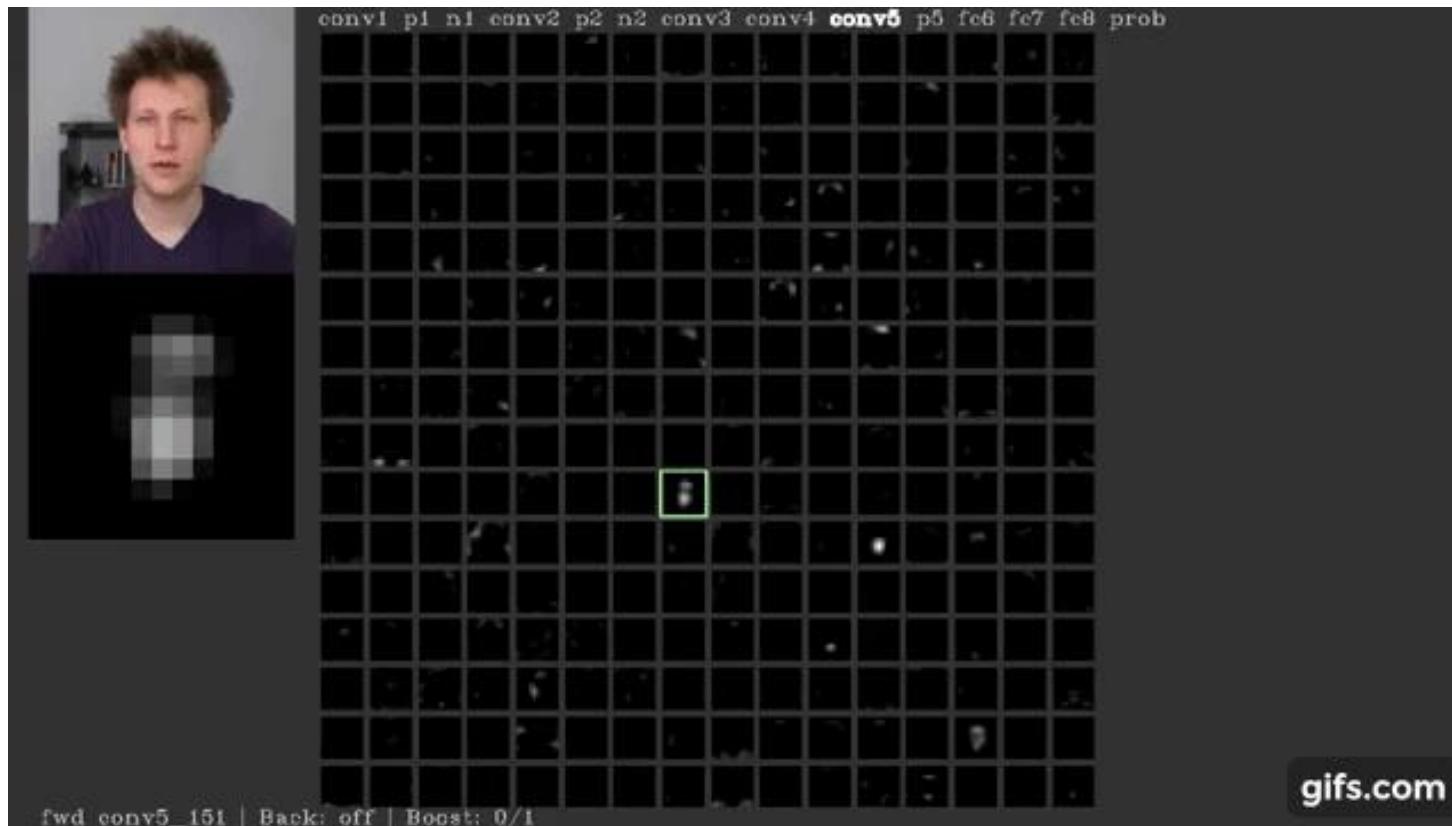


Convolutional Neural Networks (CNN)

- Pioneered by Yann LeCun back in 1989, why didn't it take off then?



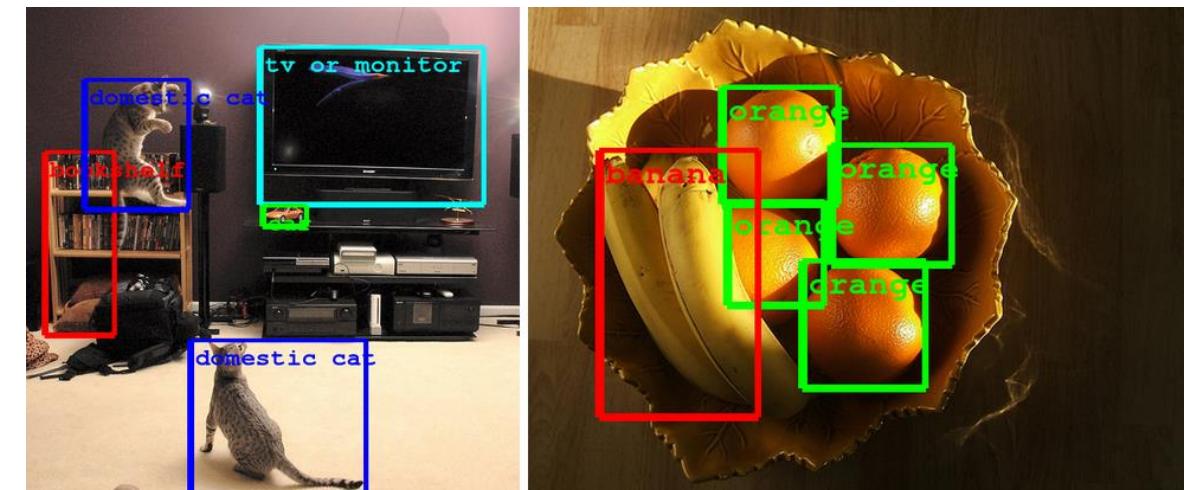
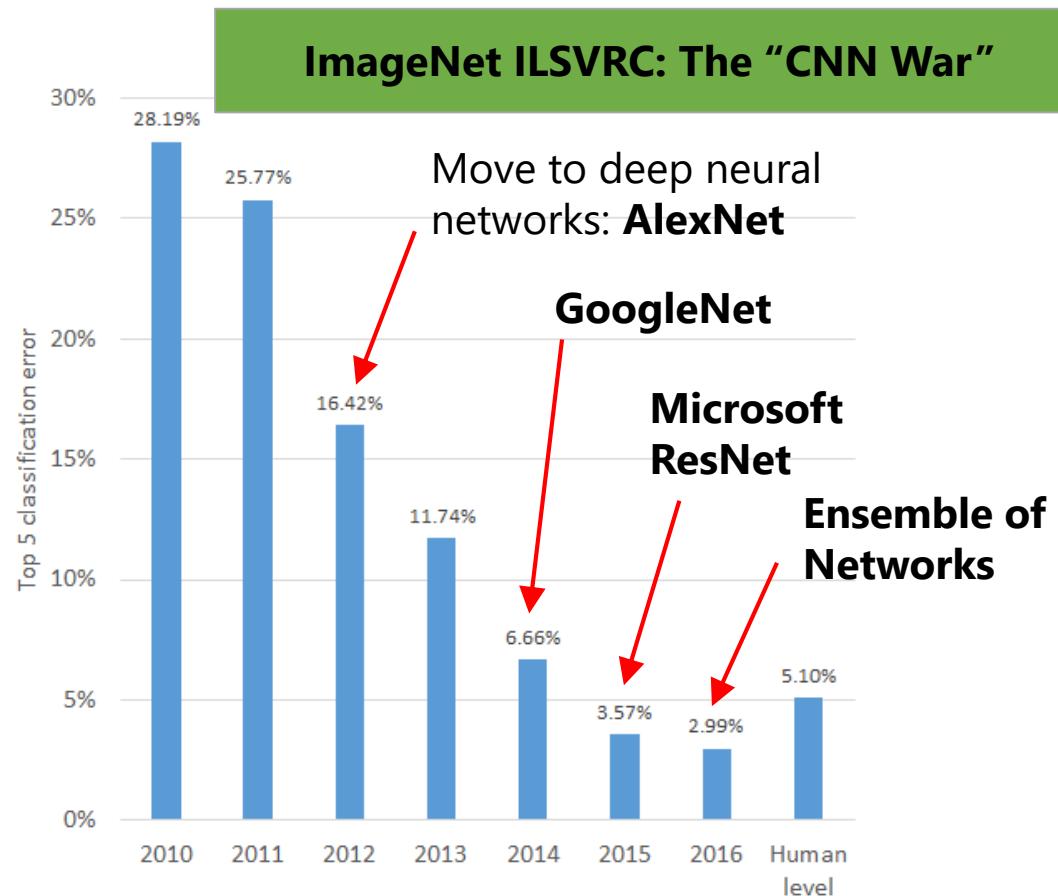
Convolutional Neural Networks (CNN)



- CNNs can be visualized to gain better understanding of what the network is learning!

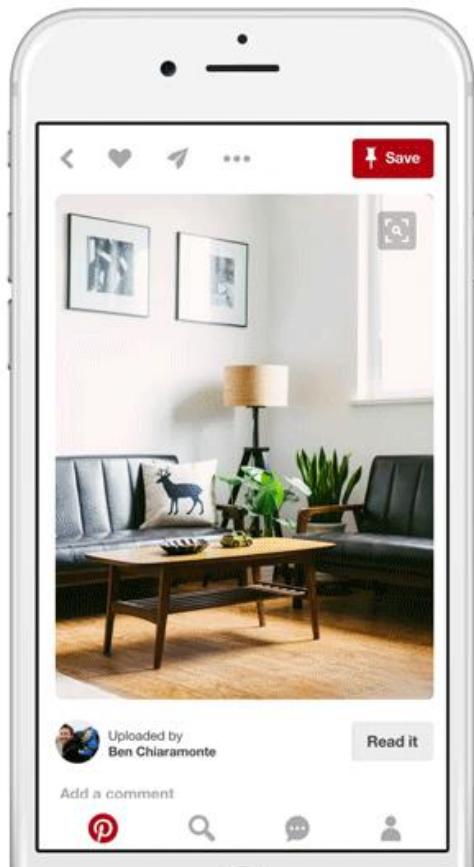
<https://github.com/yosinski/deep-visualization-toolbox>

Object Recognition and Detection/Localization



For Detecting objects:
R-CNN, Fast R-CNN, YOLO

Use Case: Visual Search @ Pinterest



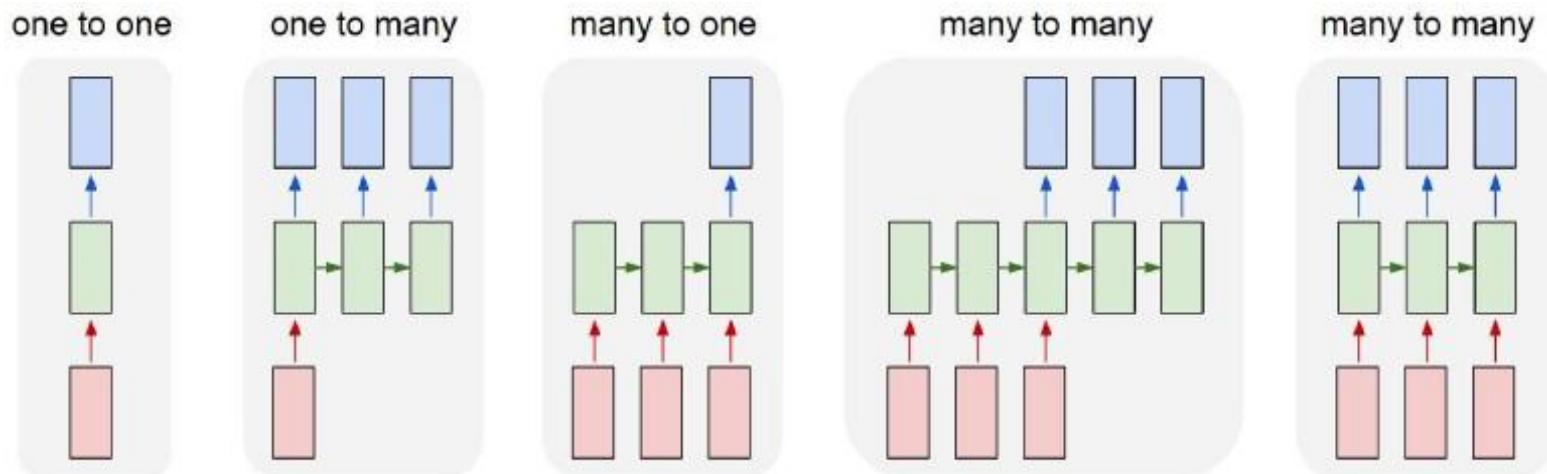
Powered by CNNs!



What was used?
Object detection
Object classification
Recommendation engine:
“Related Pins”

Recurrent Neural Networks (RNN)

- Pioneered by Juergen Schmidhuber and Sepp Hochreiter
- Great choice if your **data changes over time**, or **patterns are sequential**
- RNNs can map input sequences to output sequences!



Recurrent Neural Networks (RNN)

- Suitable for: Time series prediction, handwriting recognition, speech recognition, stock market forecasting, text translation, sentiment analysis

Single input, Sequential output

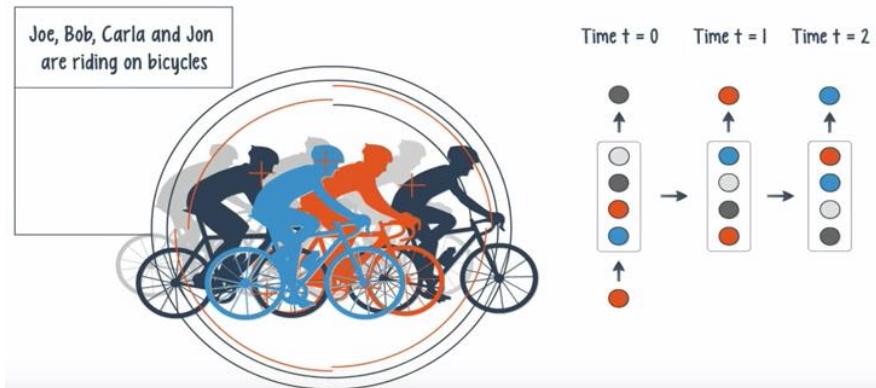
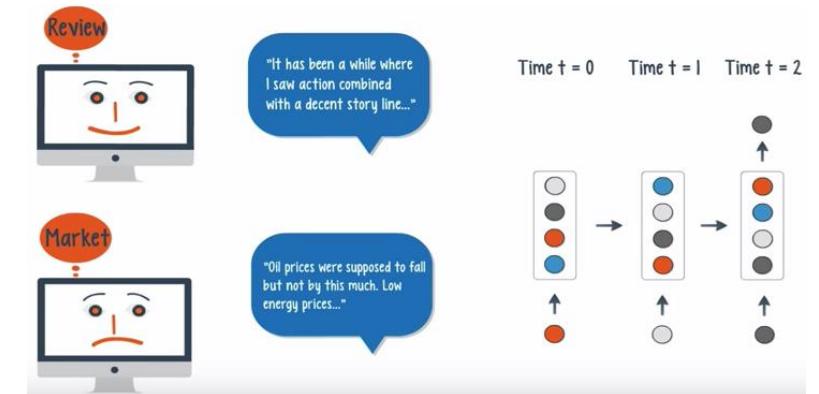


Image captioning

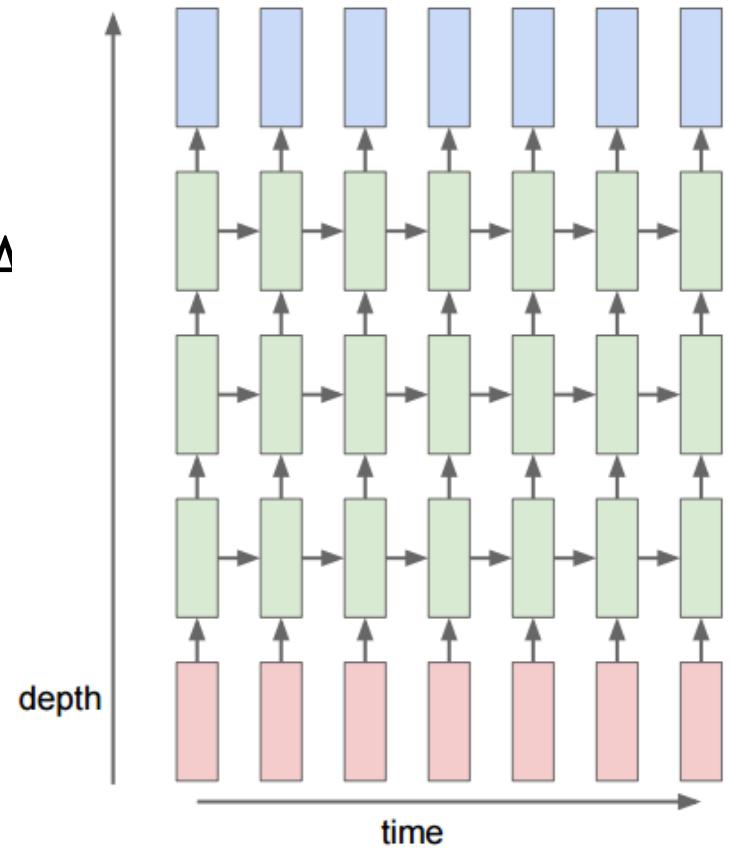
Sequential input, Single output



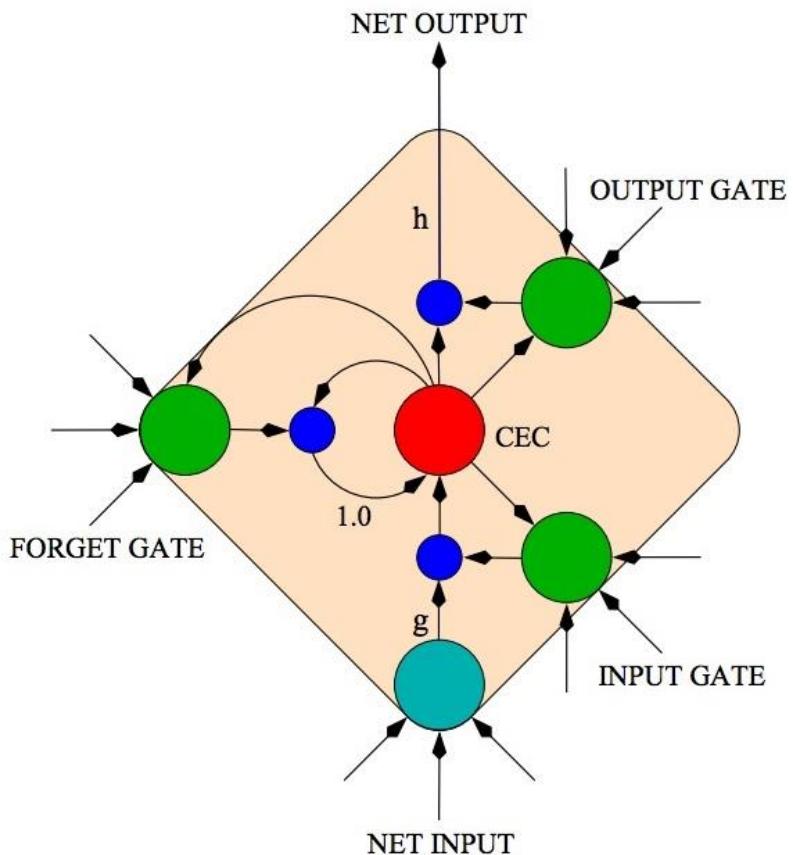
Sentiment analysis

Recurrent Neural Networks (RNN)

- Challenges:
 - Difficult to train, sensitive to parameters
 - Vanishing gradient problem just got worse! (A 100 time step RNN = 100 layer MLP!)
- RNNs can also be stacked into multiple layers to create more complex representations



Long Short-Term Memory (LSTM) RNN



- **LSTM** is a variant of RNN that provides better back-propagation dynamics,
- LSTM units give the network **memory cells** to read, write and reset operations. During training, the network can learn when it should “remember” and when it should “forget”

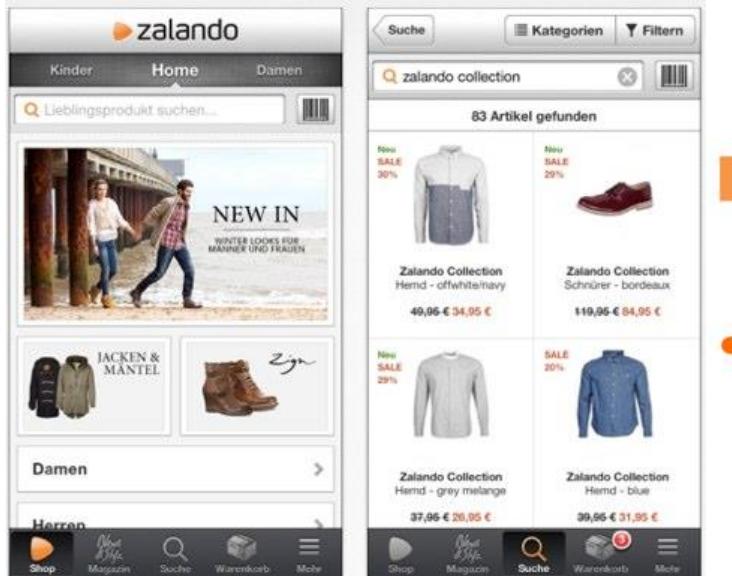
LSTM: Modeling Text Data

This is one of Crichton's best books. The characters of Karen Ross, Peter Elliot, Munro, and Amy are beautifully developed and their interactions are exciting, complex, and fast-paced throughout this impressive novel. And about 99.8 percent of that got lost in the film. Seriously, the screenplay AND the directing were horrendous and clearly done by people who could not fathom what was good about the novel. I can't fault the actors because frankly, they never had a chance to make this turkey live up to Crichton's original work. I know good novels, especially those with a science fiction edge, are hard to bring to the screen in a way that lives up to the original. But this may be the absolute worst disparity in quality between novel and screen adaptation ever. The book is really, really good. The movie is just dreadful.

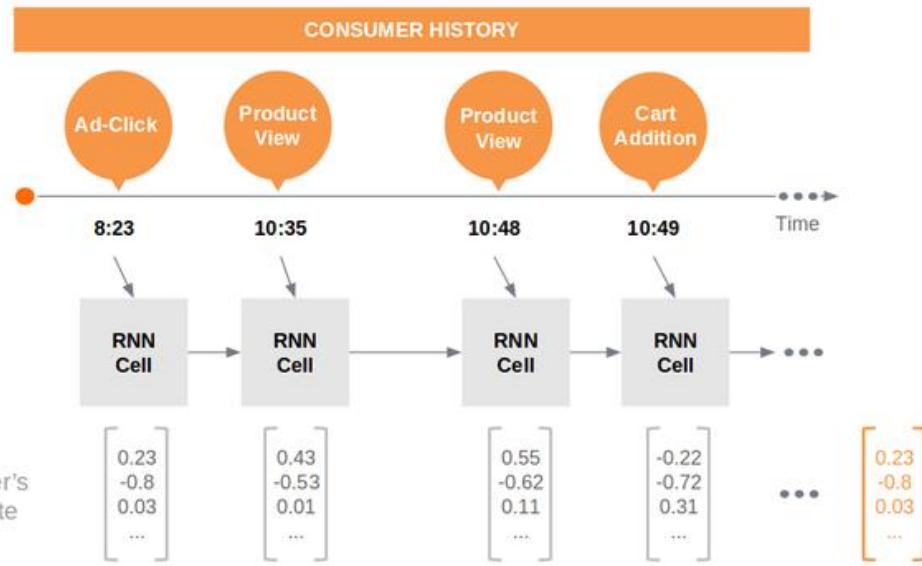
- **Text is sequential:** Insert characters into LSTM to predict probabilities of the next letter
- Text models can be turned into **sentiment classifiers:** Cater to longer bunch of sentences, like product/book/movie reviews

Use Case: Forecasting Consumer Preferences @Zalando

Powered by RNNs!

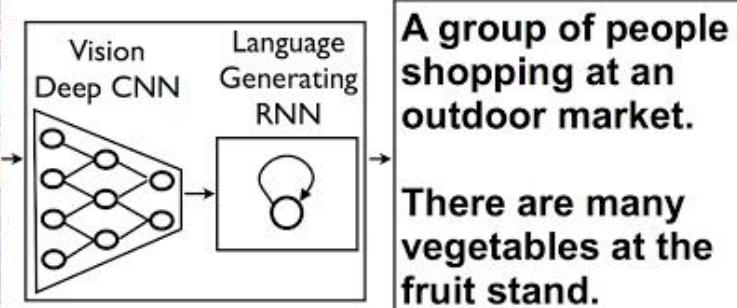


Consumer's latent state



What was achieved?
Track consumer sessions
“Memorize” patterns in consumer behaviour to predict probability of orders

Hybrid Networks



A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.



A close up of a child holding a stuffed animal



Two pizzas sitting on top of a stove top oven

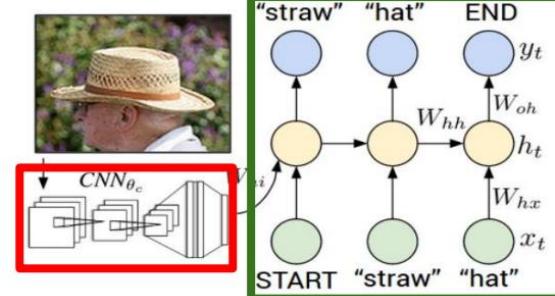


A man flying through the air while riding a skateboard

- **Image Captioning:** Combines CNN and RNN

- Generates fitting natural language captions only based on the pixel data

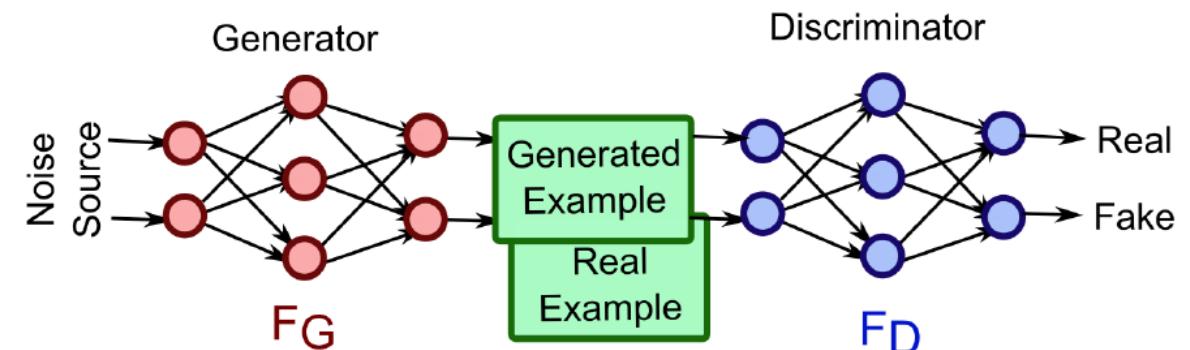
Recurrent Neural Network



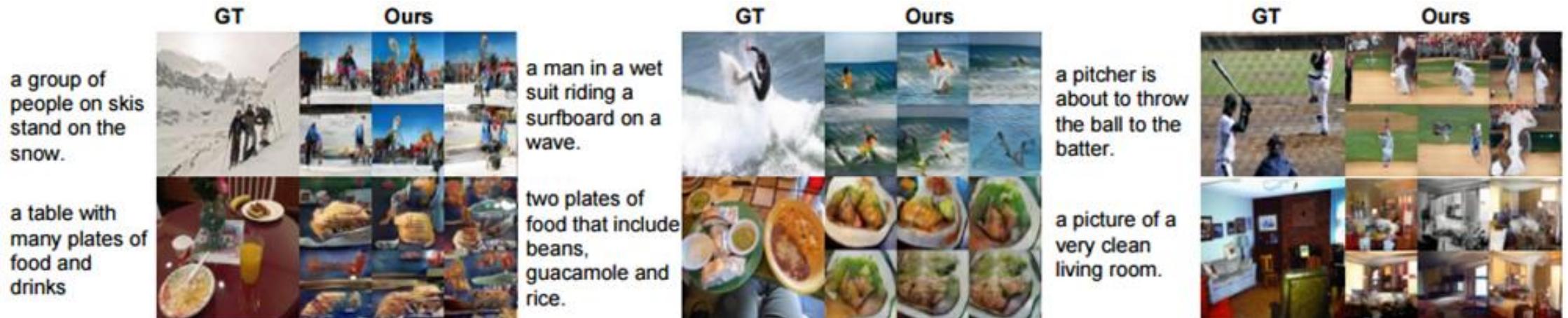
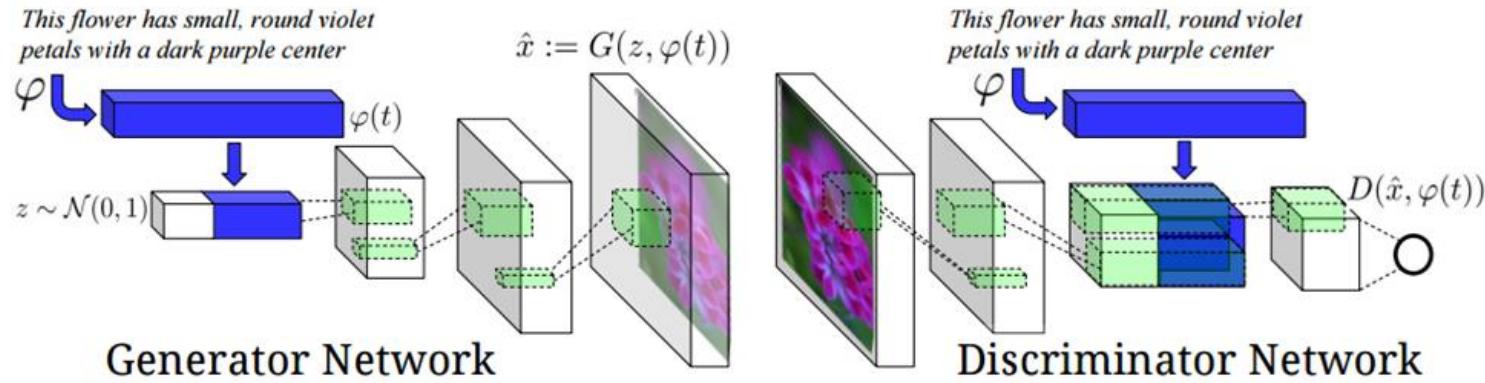
Convolutional Neural Network

Generative Adversarial Networks (GAN)

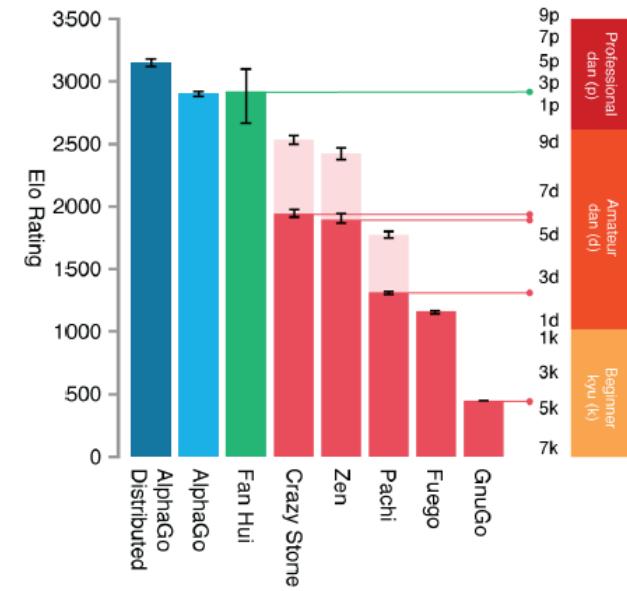
- Inspired by adversarial competition in **Game Theory**
 - **Two networks:** One tasked to generate content, the other to judge content



GAN: Text-to-Image Synthesis



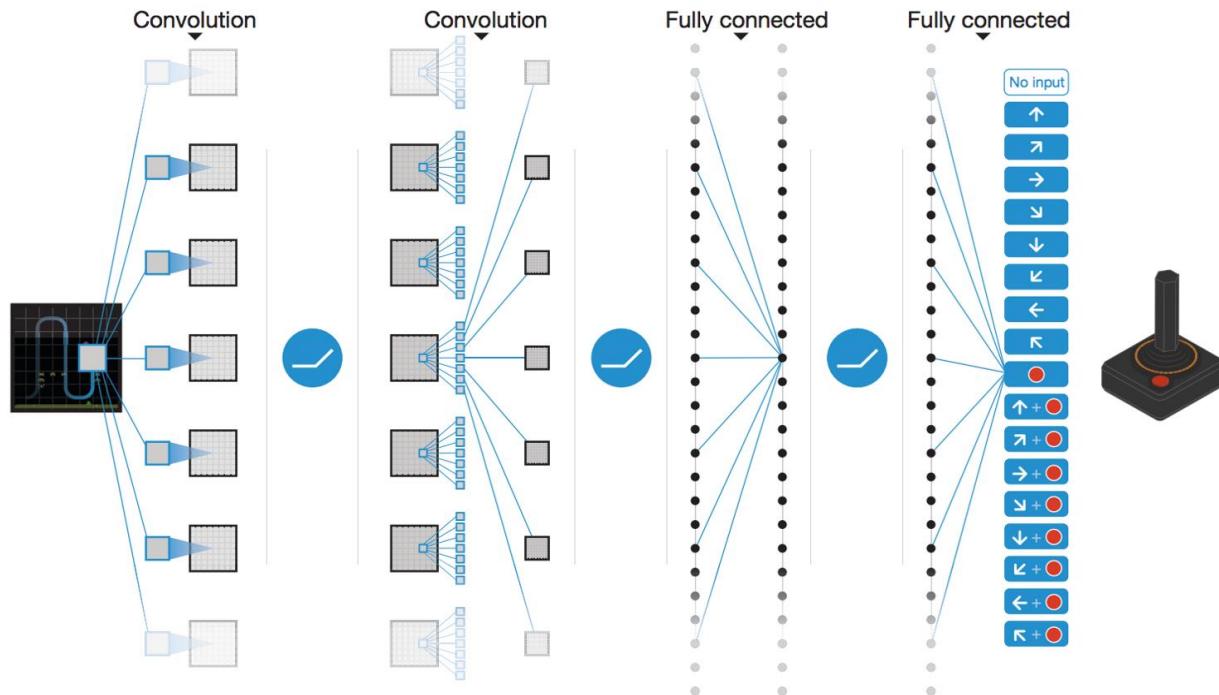
Deep Learning for Games



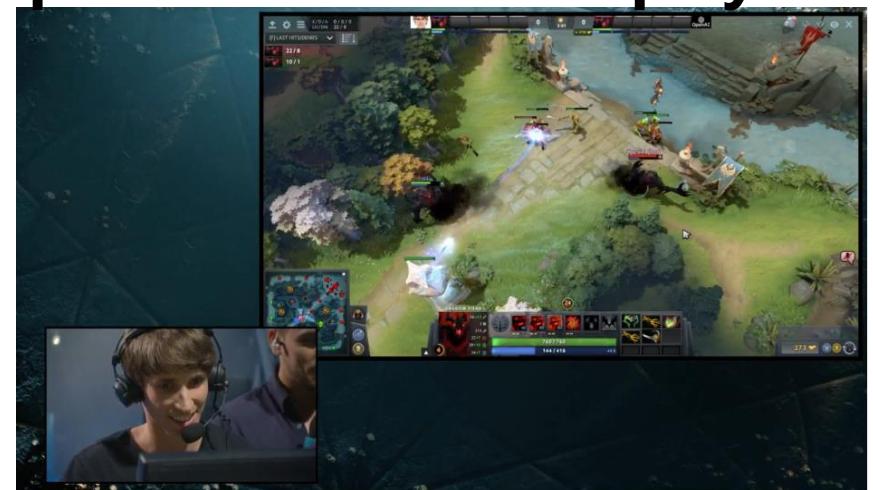
Examines thousands of human GO moves, and then masters it by playing with itself over and over again.

History was made: Google DeepMind's **AlphaGo** beats Go champion Lee Sedol

Deep Learning for Games



Just In: OpenAI's bot beats professional Dota 2 players



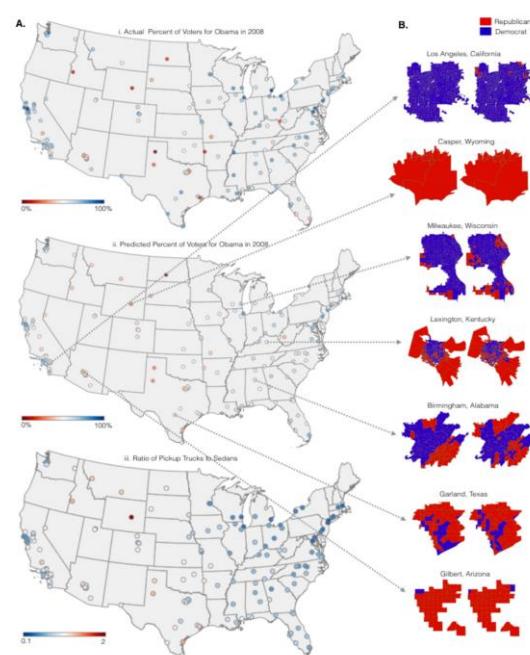
- **Deep Q-Network (DQN)** is a model-free approach to reinforcement learning using deep networks in environments with discrete action choices

What Else Can Deep Learning Generate?

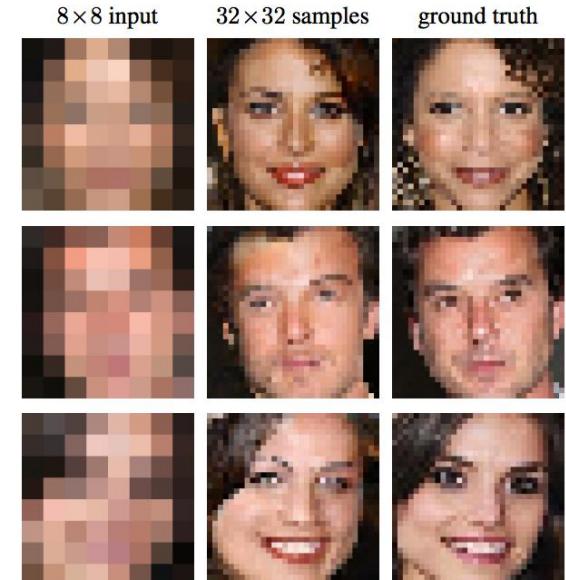


— Generate Expressive Music

Learning Demographics and
Predicting Election Results
based on Street View
Images

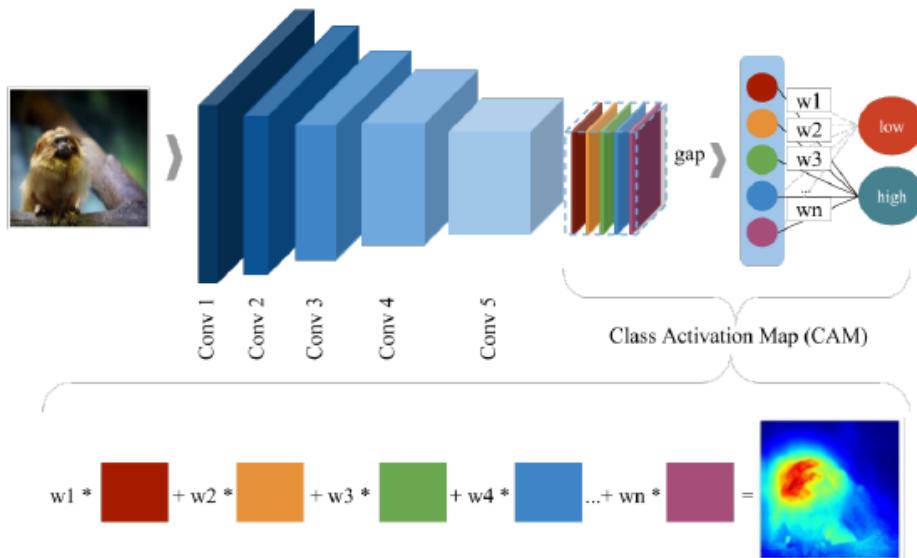


Pixel Restoration “CSI-style”

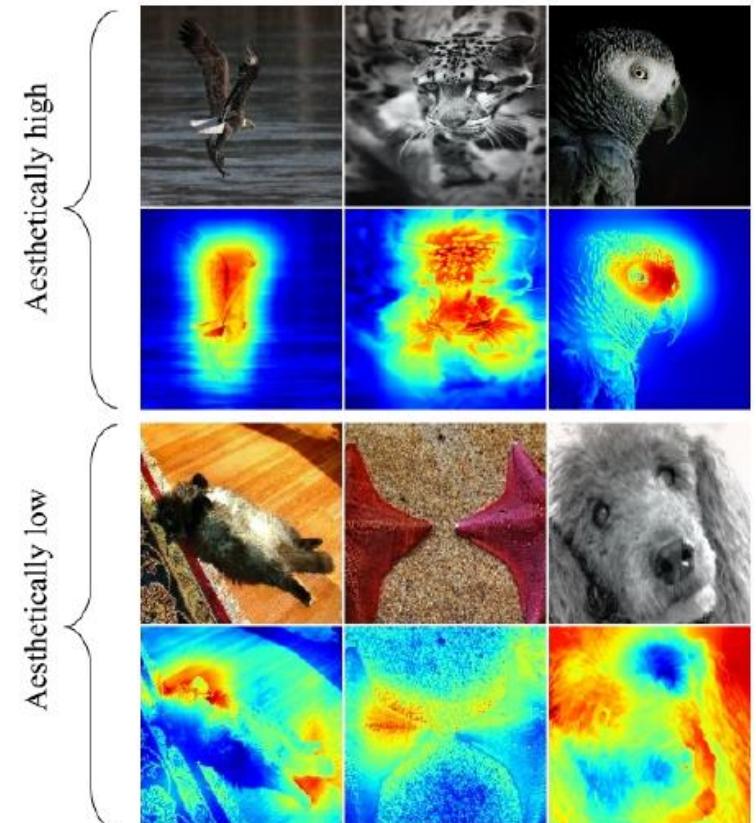


Some Homegrown Research

Predicting and Localizing Aesthetics in Images



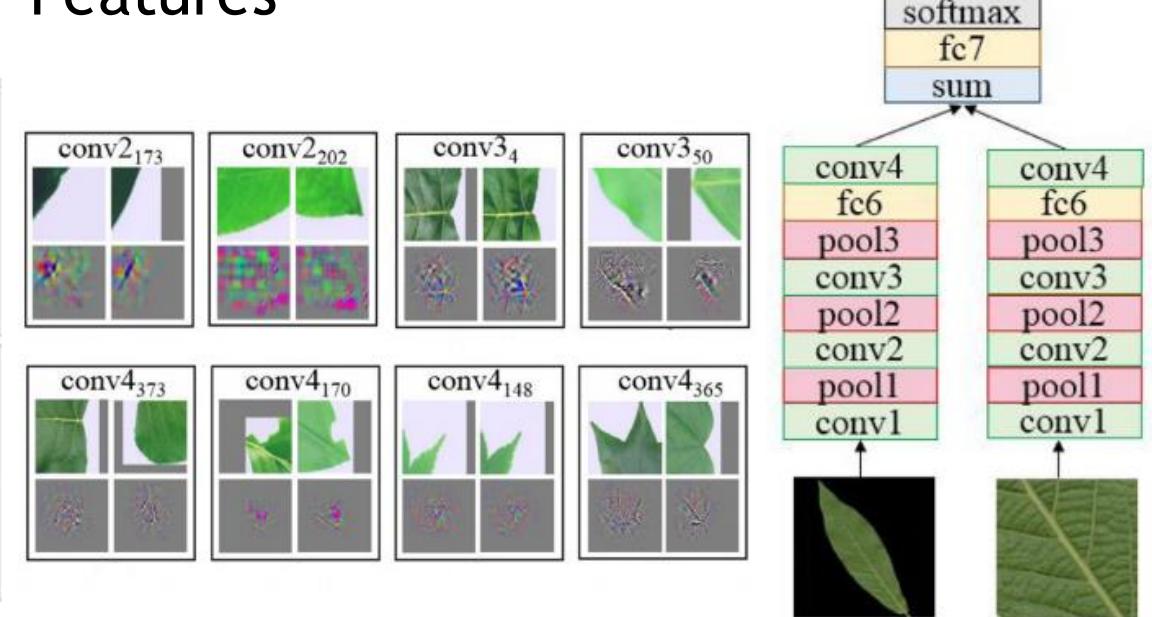
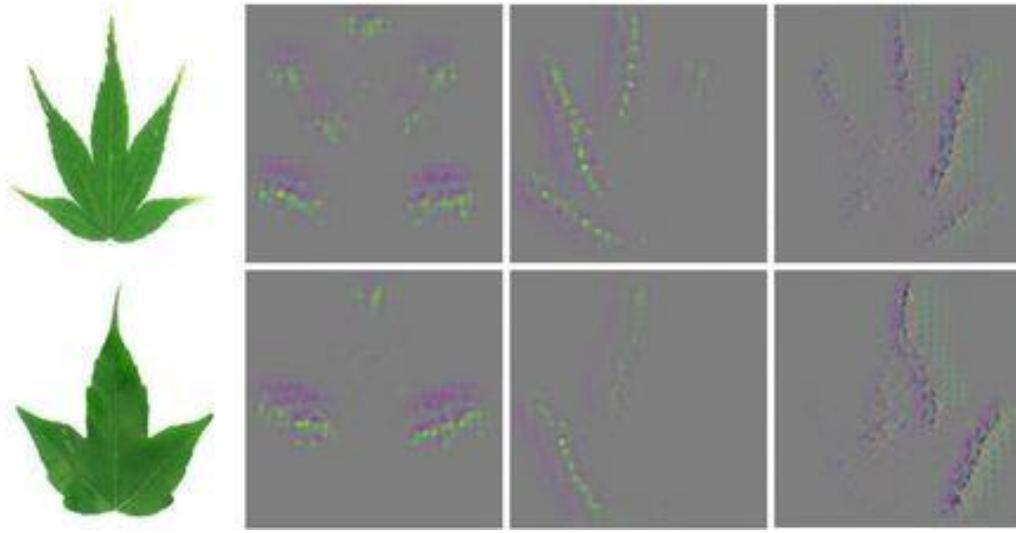
State-of-the-art:
82.27% (Text-augmented MultGAP Network, ICIP 2017)



Some Homegrown Research



Plant Identification & Learning Leaf Features



MalayaKew and Flavia datasets

Top-1 accuracy: 96.3% (Early Fusion conv-sum, PR 2017)

Deep Learning Tools - Open Source!



theano



Caffe



DL4J Deep Learning for Java



Microsoft
CNTK



dmlc
mxnet

NVIDIA DIGITS

Lasagne



K Keras

Is Deep Learning over-hyped?

Is Deep Learning a black box that is unexplainable?

What should I do if I want to get started?

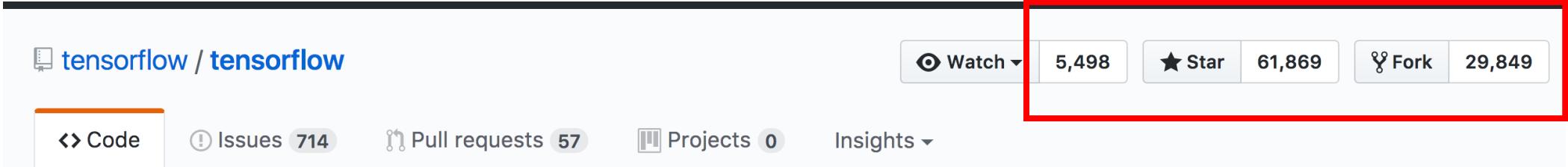
TensorFlow

What is TensorFlow?

- URL: <https://www.tensorflow.org/>
- Released under the open source license on November 9, 2015
- Current version 1.2
- Open source software library for numerical computation using data flow graphs
- Originally developed by Google Brain Team to conduct machine learning and deep neural networks research
- General enough to be applicable in a wide variety of other domains as well
- TensorFlow provides an extensive suite of functions and classes that allow users to build various models from scratch.



Most popular on Github



tensorflow / tensorflow

Watch 5,498 Star 61,869 Fork 29,849

Code Issues 714 Pull requests 57 Projects 0 Insights

Computation using data flow graphs for scalable machine learning <http://tensorflow.org>

tensorflow machine-learning python deep-learning deep-neural-networks neural-network ml distributed

19,049 commits 16 branches 33 releases 917 contributors Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

AnishShah committed with drpngx [issue #10835] Negative axis support for gradient of reduce_prod (#11019) ... Latest commit ea79ba4 3 hours ago

tensorflow [issue #10835] Negative axis support for gradient of reduce_prod (#11019) 3 hours ago

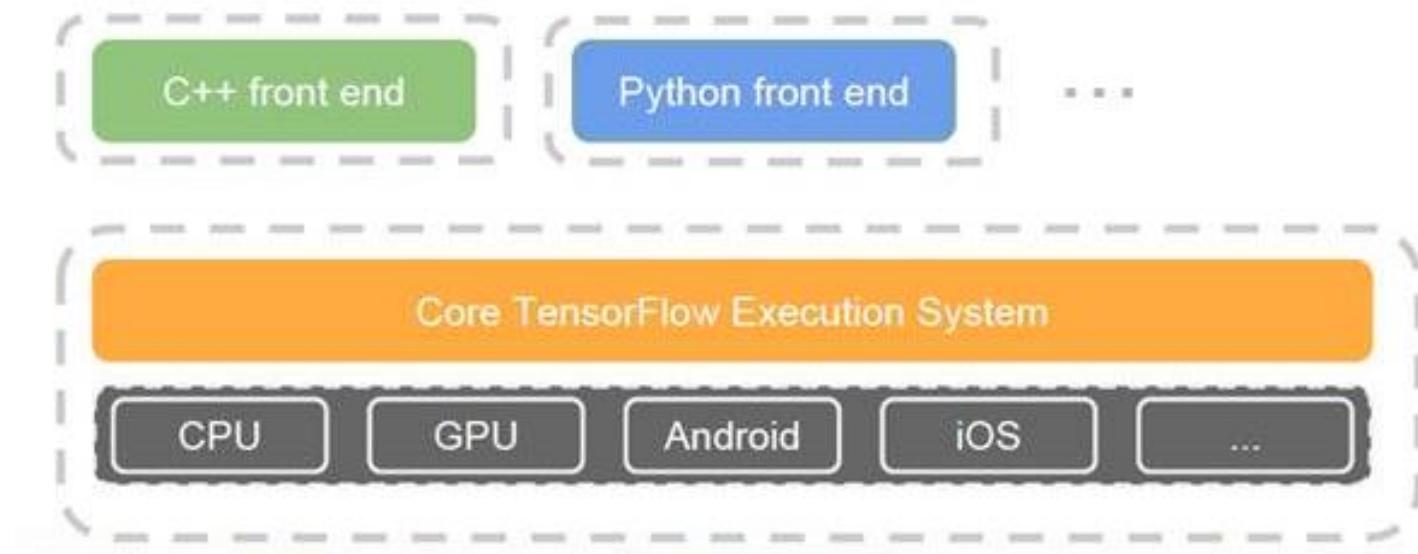
third_party Add python import library on Windows (#10980) a day ago

tools Create tf_env_collect.sh 13 days ago

<https://github.com/tensorflow/tensorflow>

TensorFlow architecture

- Core in C++
 - Low overhead
- Different front ends for specifying/driving the computation
 - Python, C++, R and many more



CPU - GPU

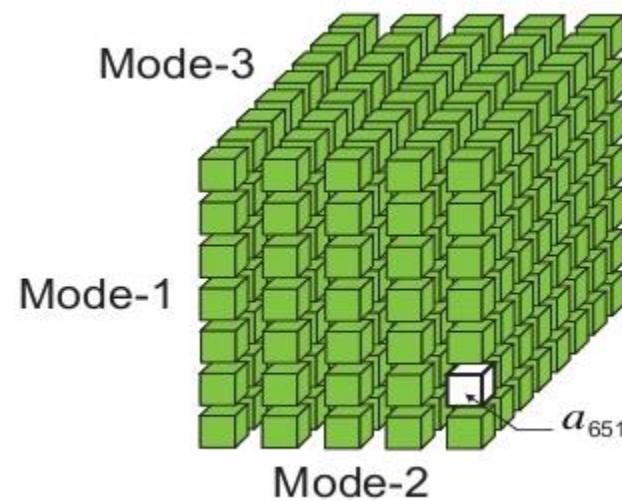
- In TensorFlow, the supported device types are CPU and GPU. They are represented as strings. For example:

- "/cpu:0": The CPU of your machine.
- "/gpu:0": The GPU of your machine, if you have one.
- "/gpu:1": The second GPU of your machine, etc.

```
# Creates a graph.
with tf.device('/gpu:2'):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
    c = tf.matmul(a, b)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(c))
```

Why it is called TensorFlow?

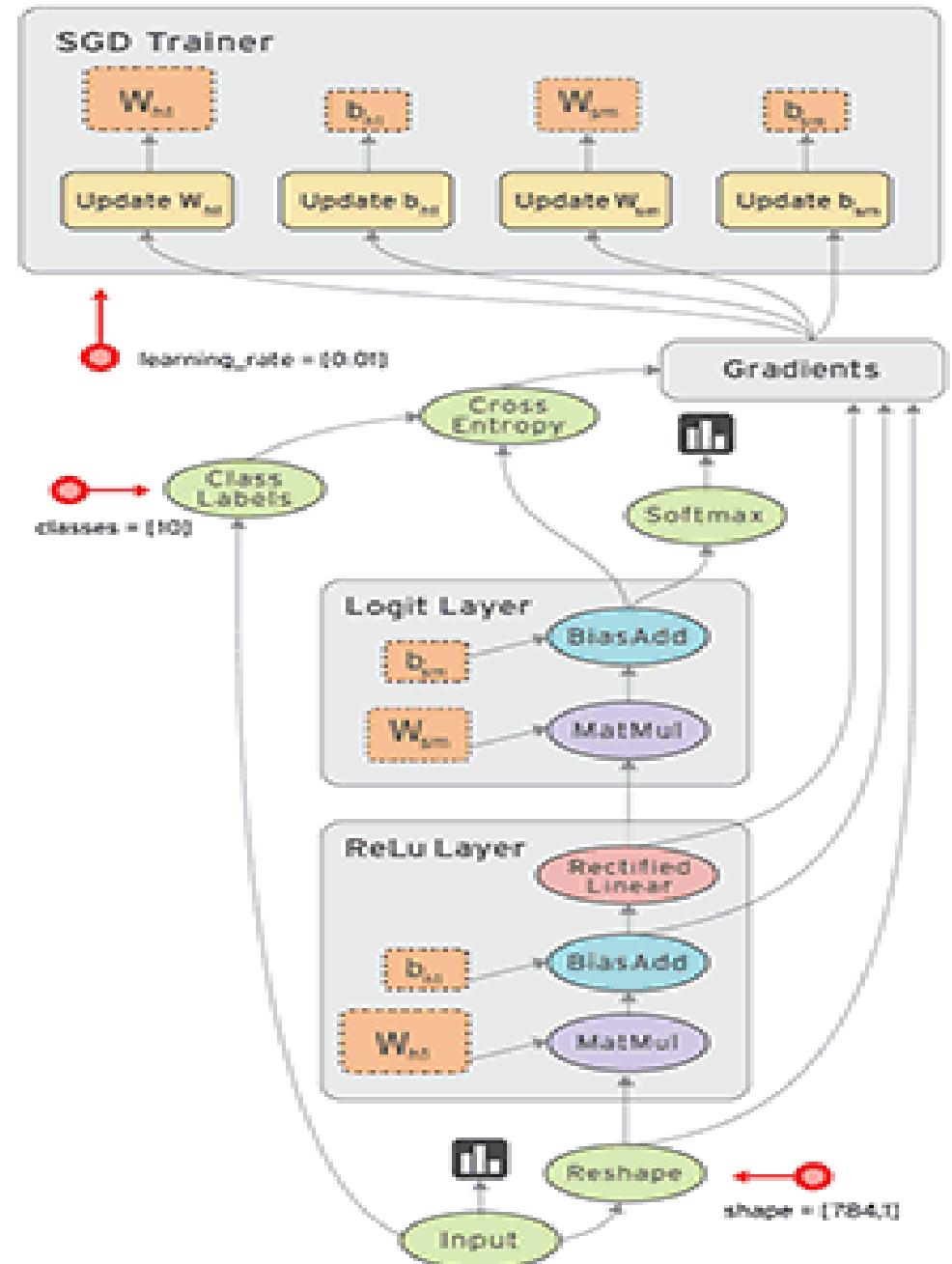
- What is a Tensor?



Dimensions	Example	Terminology																					
1	<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr></table>	0	1	2	Vector																		
0	1	2																					
2	<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	Matrix												
0	1	2																					
3	4	5																					
6	7	8																					
3	<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	3D Array (3 rd order Tensor)												
0	1	2																					
3	4	5																					
6	7	8																					
N	<table border="1"><tr><td><table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table></td><td>...</td><td><table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table></td></tr></table>	<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	...	<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	ND Array
<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	...	<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8			
0	1	2																					
3	4	5																					
6	7	8																					
0	1	2																					
3	4	5																					
6	7	8																					

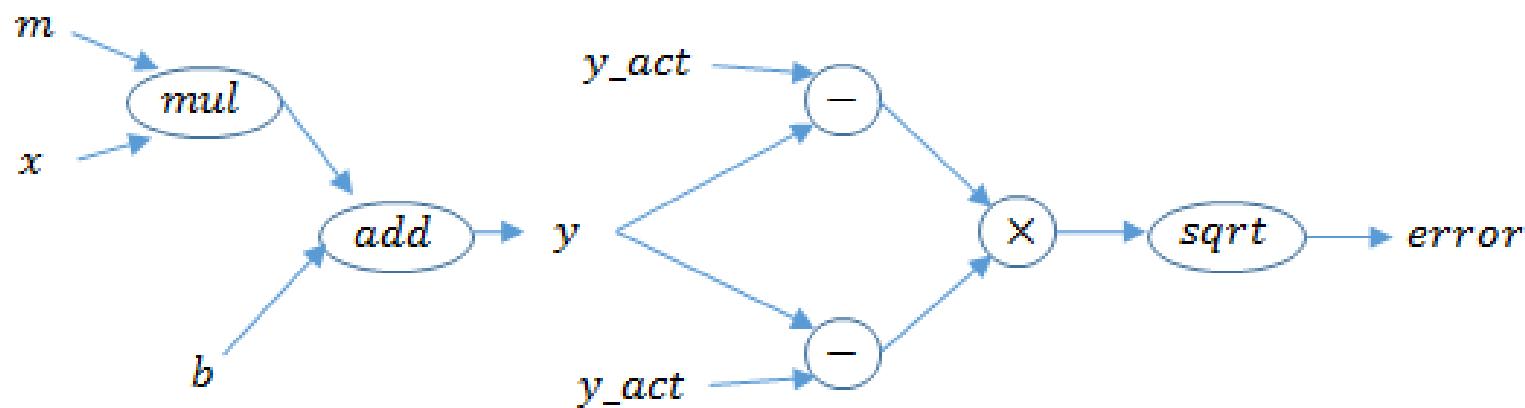
Why it is called TensorFlow?

- TensorFlow is based on computation data flow graph
- TensorFlow separates definition of computations from their execution



Steps to start with TensorFlow

- **Step 1:** Assemble a computational graph
- **Step 2:** Use a session to execute operations in the graph



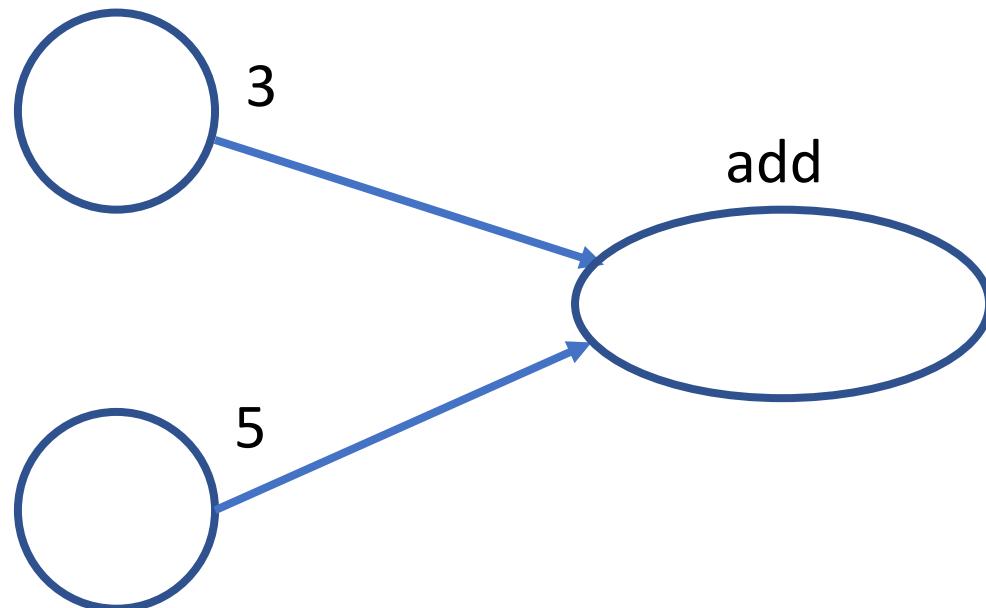
TensorFlow Graph

```
import tensorflow as tf  
a = tf.add(3,5)  
print(a)
```

Tensor("Add:0", shape=(), dtype=int32)

```
import numpy as np  
a = np.add(3,5)  
print(a)
```

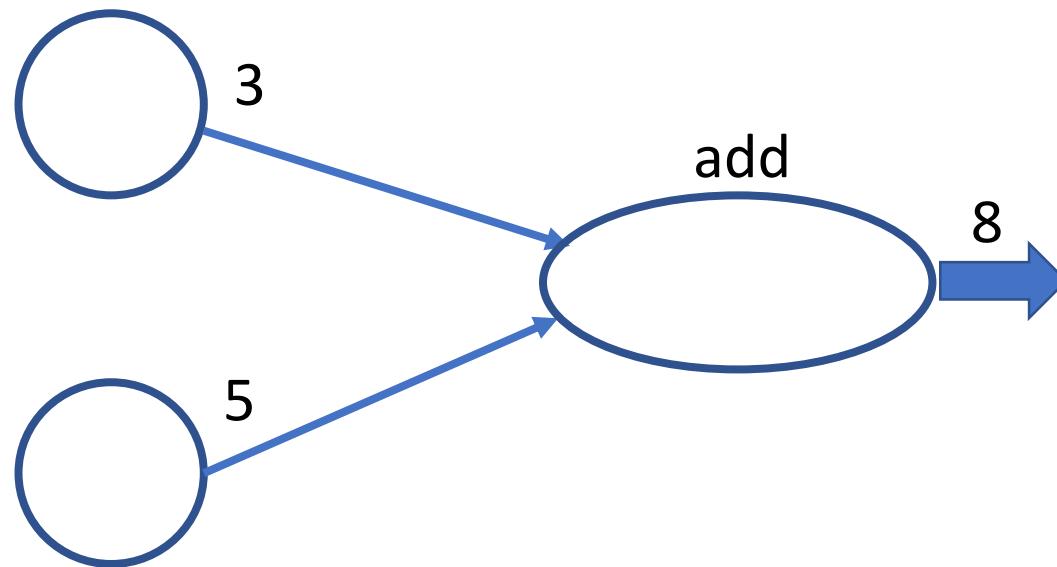
8



TensorFlow Graph

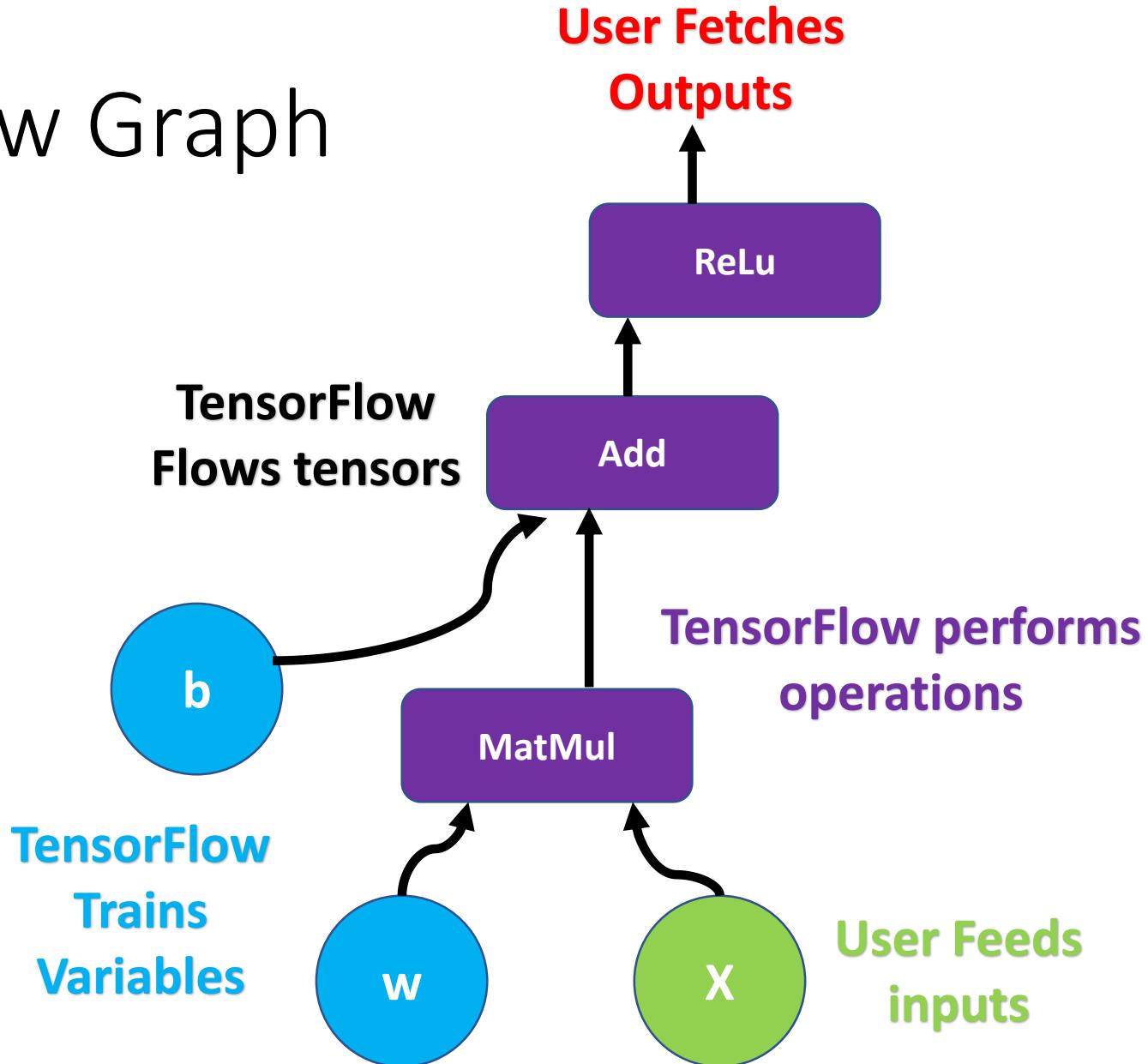
- To get the value of a
- Create a **session**, assign it to variable 'sess' so we can call it later
- Within the session, evaluate the graph to fetch the value of *a*

```
import tensorflow as tf  
a = tf.add(3, 5)  
sess = tf.Session()  
print (sess.run(a))  
sess.close()
```



- A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.

TensorFlow Graph



Variables

- Variables to hold and update parameters.
- Variables are in-memory buffers containing tensors.
- Must be explicitly initialized and can be saved to disk during and after training

```
# create variable a with scalar value
a = tf.Variable(2, name="scalar")

# create variable b as a vector
b = tf.Variable([2, 3], name="vector")

# create variable c as a 2x2 matrix
c = tf.Variable([[0, 1], [2, 3]], name="matrix")

init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
```

Placeholder

- A placeholder is simply a variable that will be assigned data to at a later date.
- It allows operations to be created and computation graph to be built, without needing the data.

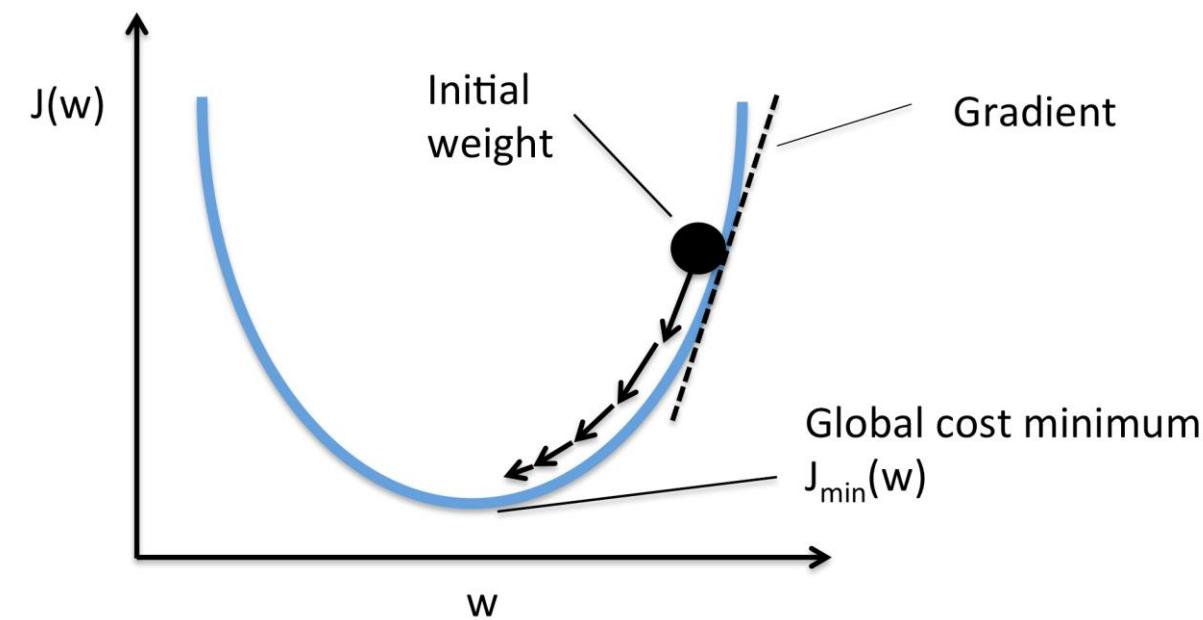
```
x = tf.placeholder("float", None)
y = x * 2

with tf.Session() as session:
    result = session.run(y, feed_dict={x: [1, 2, 3]})
print(result)
```

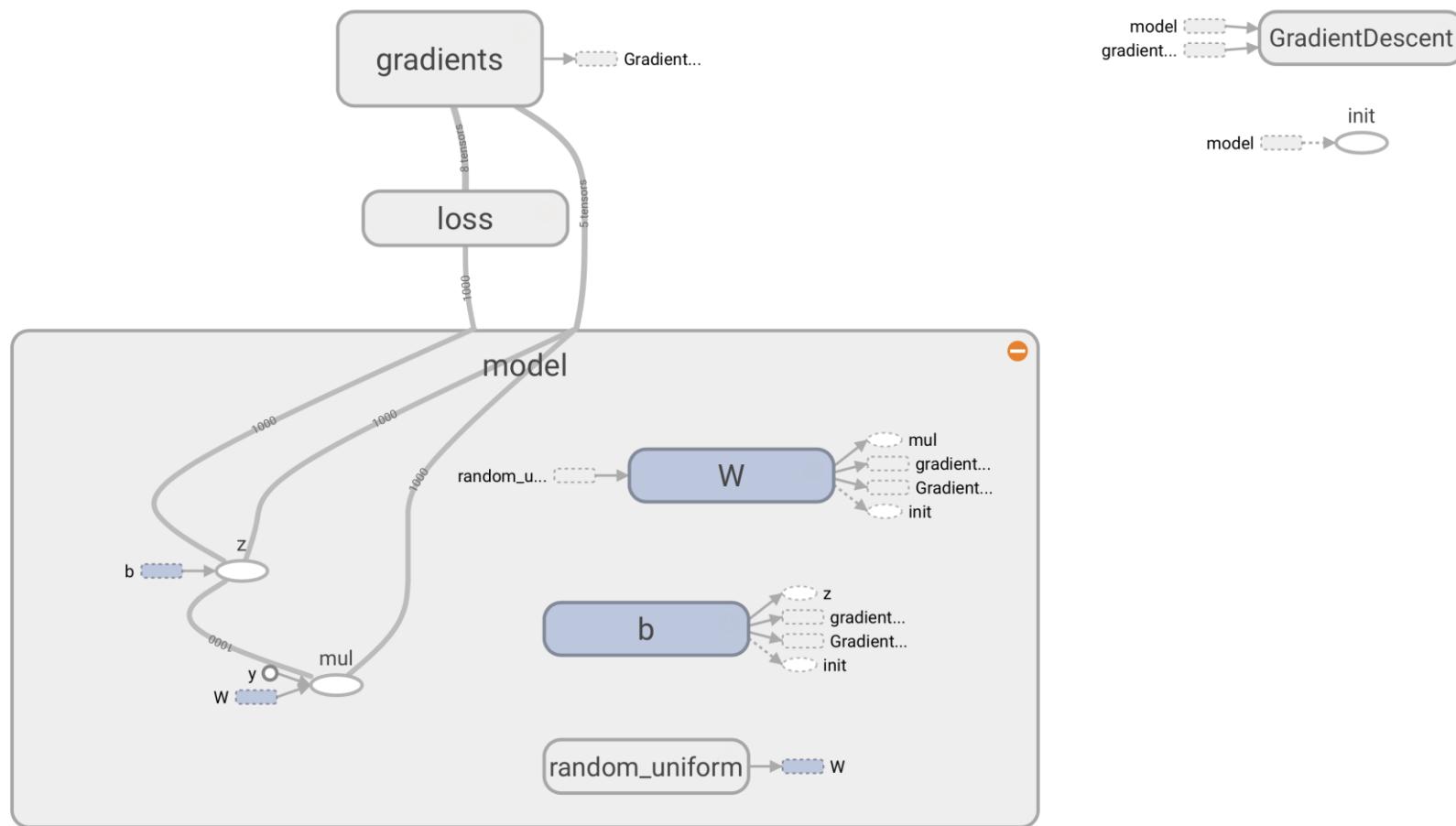
```
[ 2.  4.  6.]
```

Learn Parameters: Optimization

- The Optimizer base class provides methods to compute gradients for a loss and apply gradients to variables.
- A collection of subclasses implement classic optimization algorithms such as GradientDescent and Adagrad.
- TensorFlow provides functions to compute the derivatives for a given TensorFlow computation graph, adding operations to the graph.

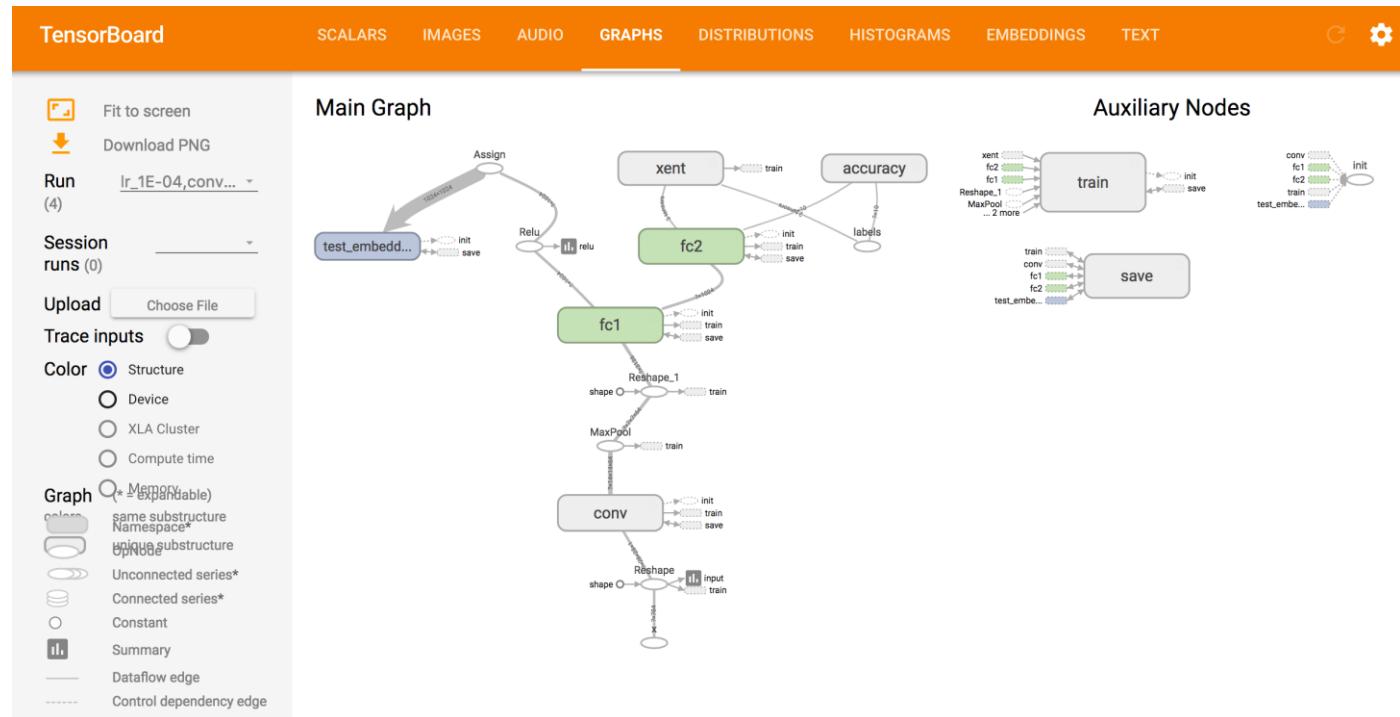


Learn parameters: Optimization



TensorBoard

- Visualize your TensorFlow graph
- Plot quantitative metrics about the execution of your graph
- Show additional data like images that pass through it



TensorFlow Models

<https://github.com/tensorflow/models>

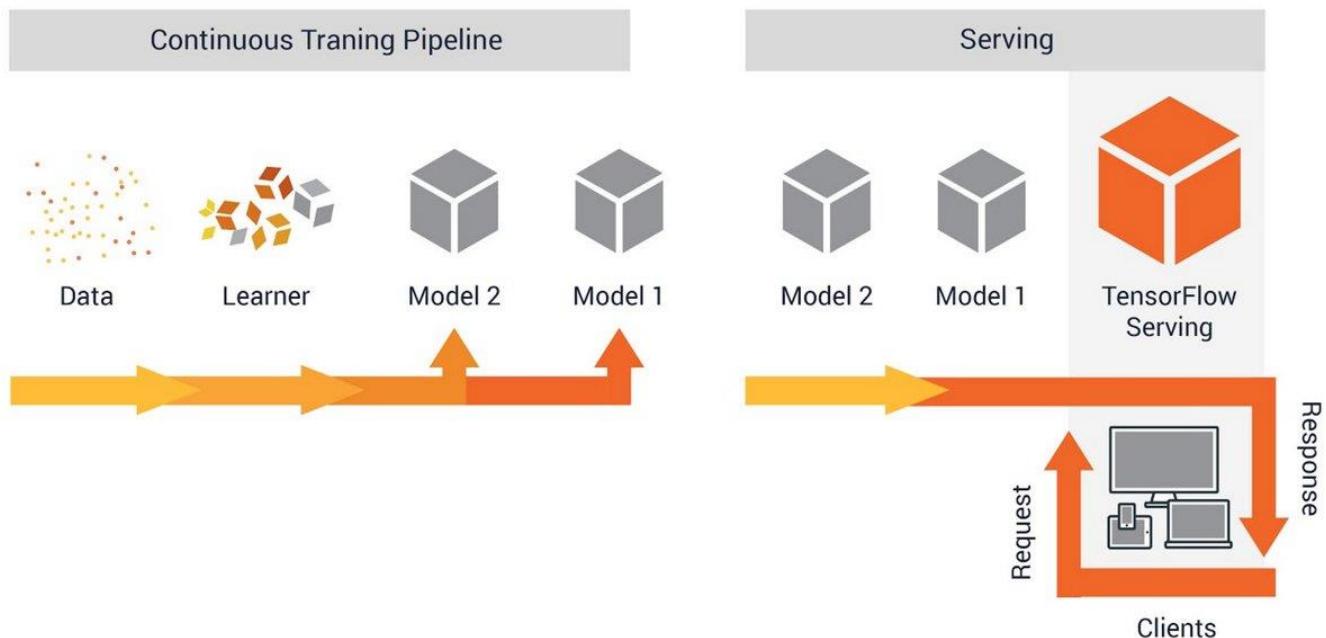
Models

- [adversarial crypto](#): protecting communications with adversarial neural cryptography.
- [adversarial text](#): semi-supervised sequence learning with adversarial training.
- [attention ocr](#): a model for real-world image text extraction.
- [autoencoder](#): various autoencoders.
- [cognitive mapping and planning](#): implementation of a spatial memory based mapping and planning architecture for visual navigation.
- [compression](#): compressing and decompressing images using a pre-trained Residual GRU network.
- [differential privacy](#): privacy-preserving student models from multiple teachers.
- [domain adaptation](#): domain separation networks.
- [im2txt](#): image-to-text neural network for image captioning.
- [inception](#): deep convolutional networks for computer vision.

TensorFlow Serving

- Flexible, high-performance serving system for machine learning models, designed for production environments.
- Easy to deploy new algorithms and experiments, while keeping the same server architecture and APIs

Serve models in production with TensorFlow Serving



Evaluation

Please fill up the online evaluation form for today's session

<https://www.research.net/r/F3PQK5H>

