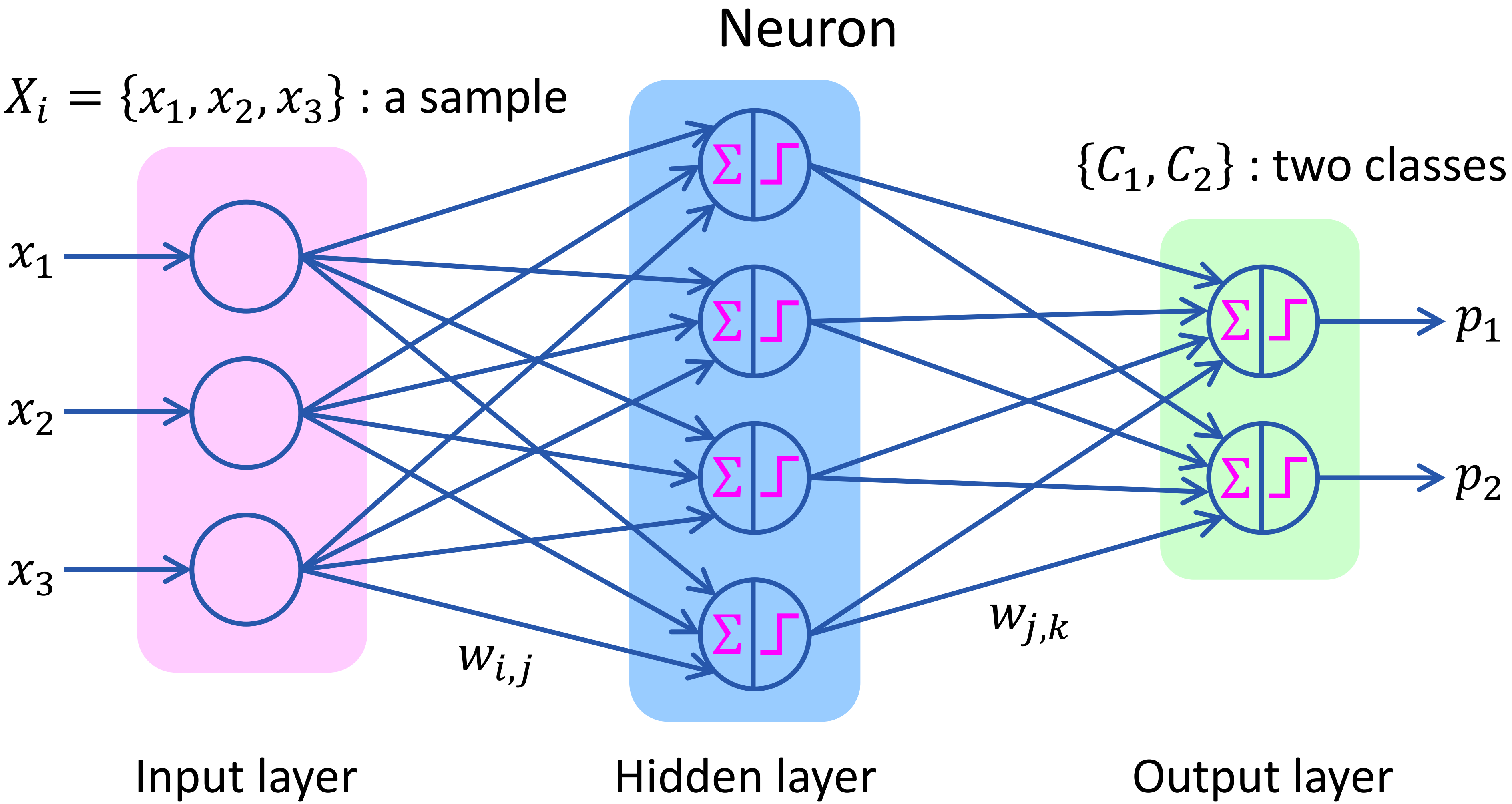
CSC 4740 / 6740 Data Mining

Spring 2024

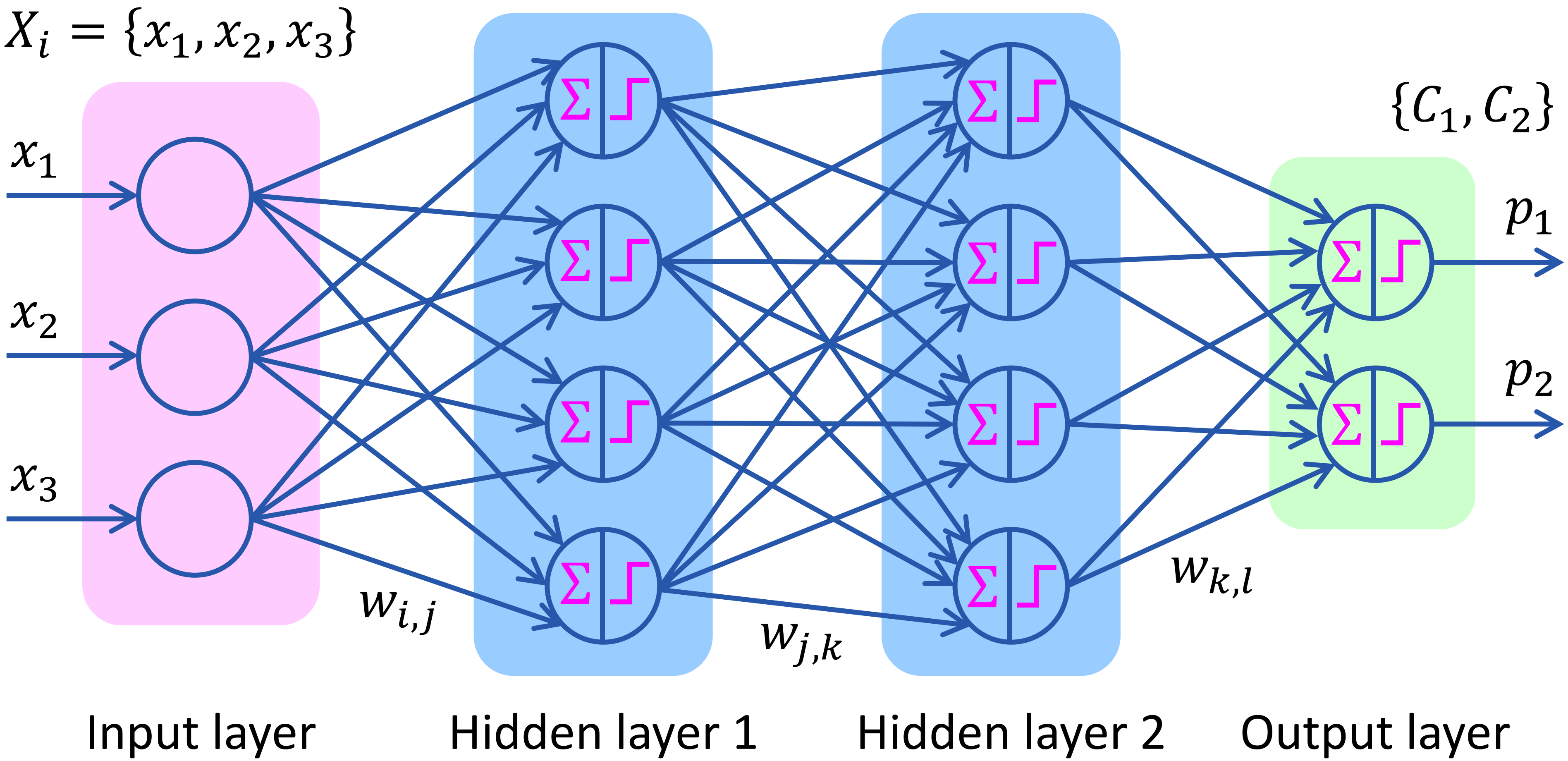
Chapter 9 Classification: Advanced Methods

GeorgiaStateUniversity

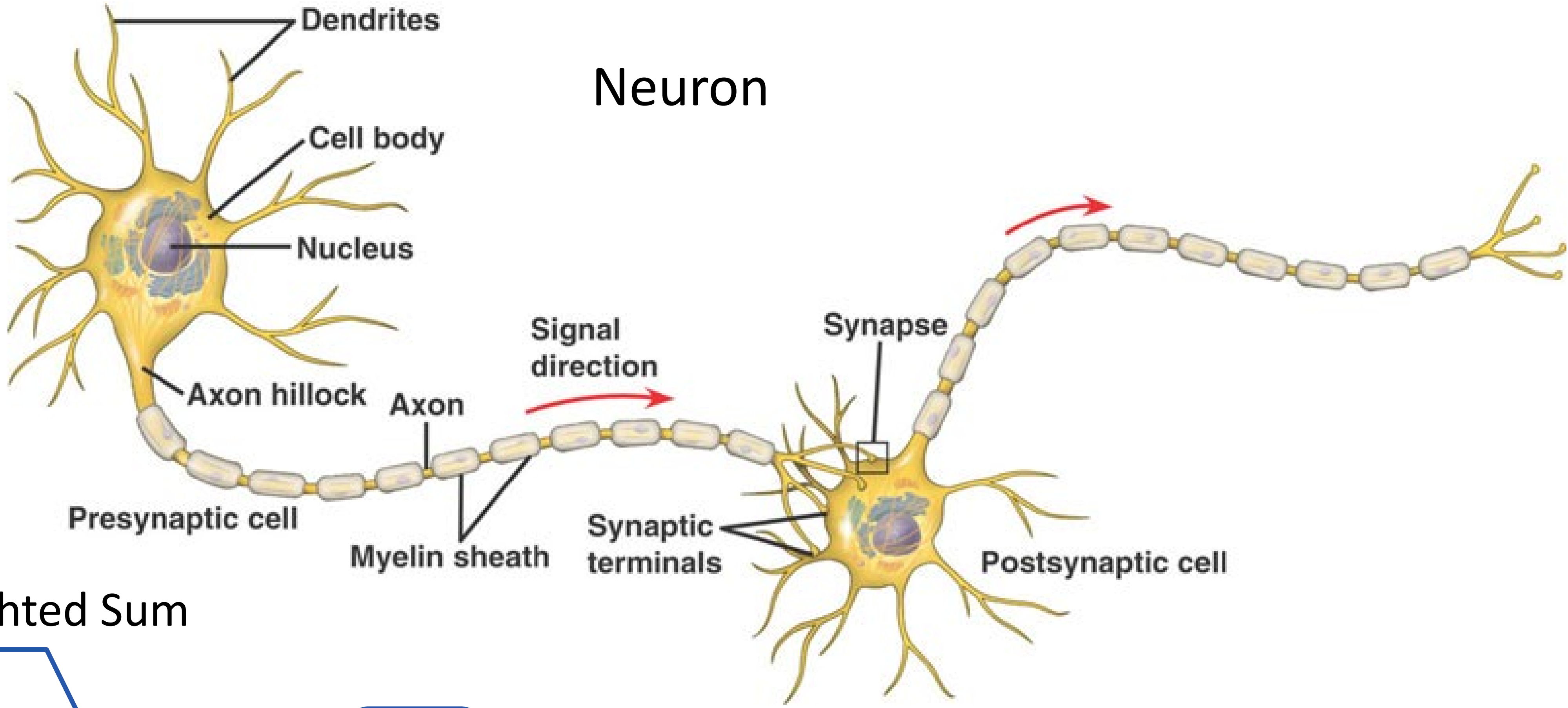
Neural Network

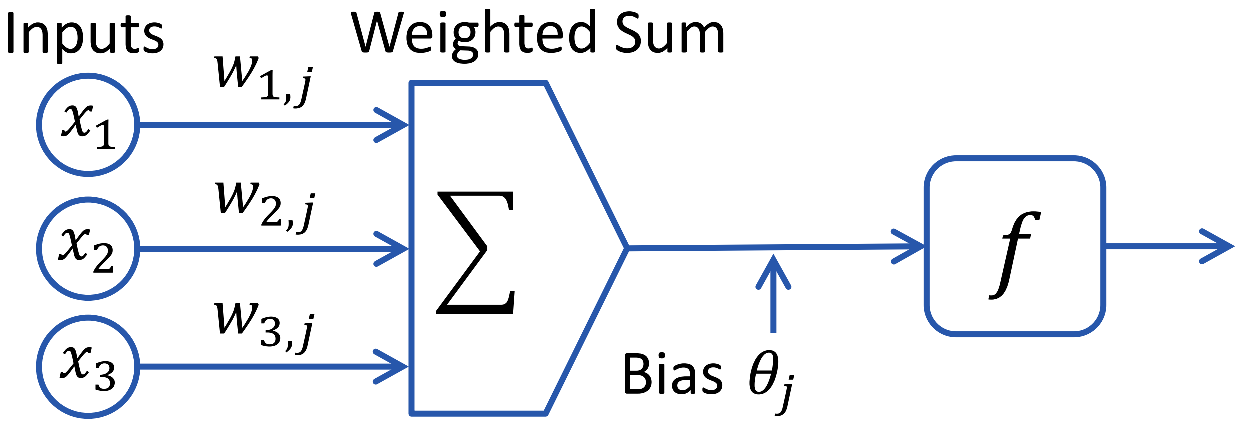


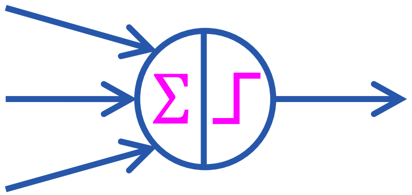
Neural Network



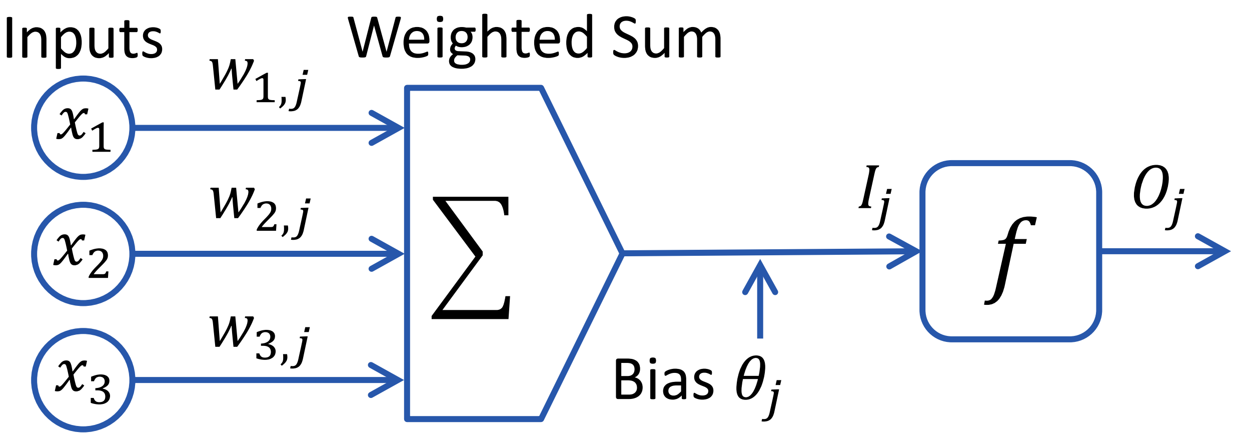
Neuron

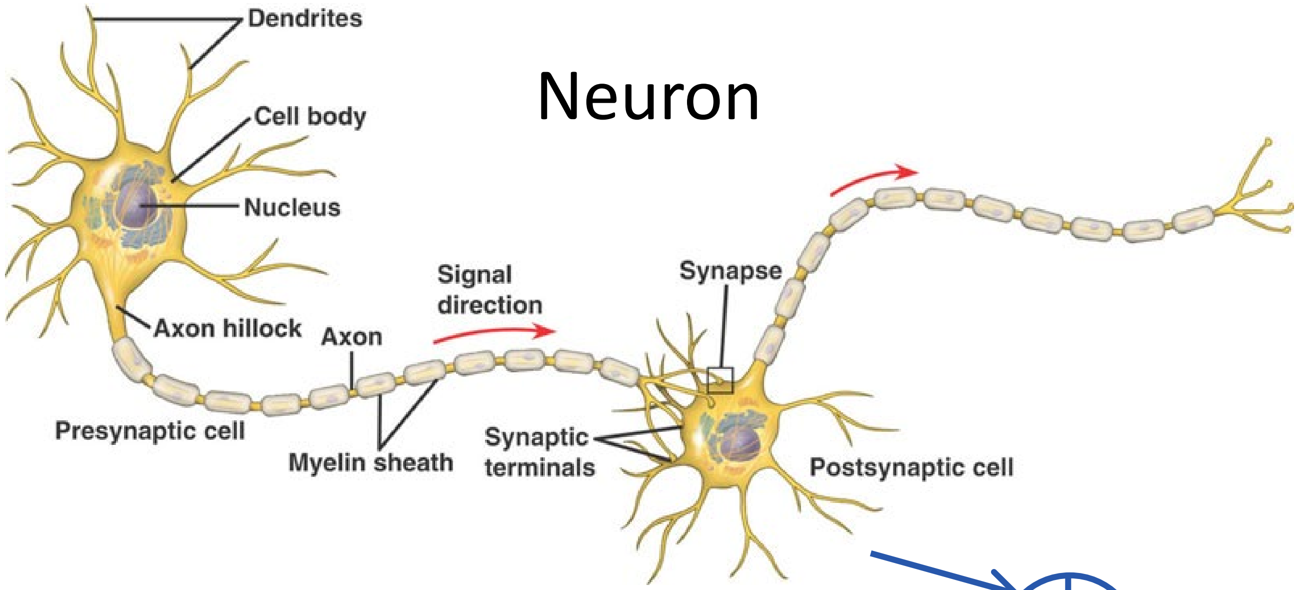


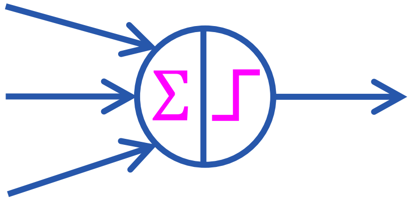




Perceptron



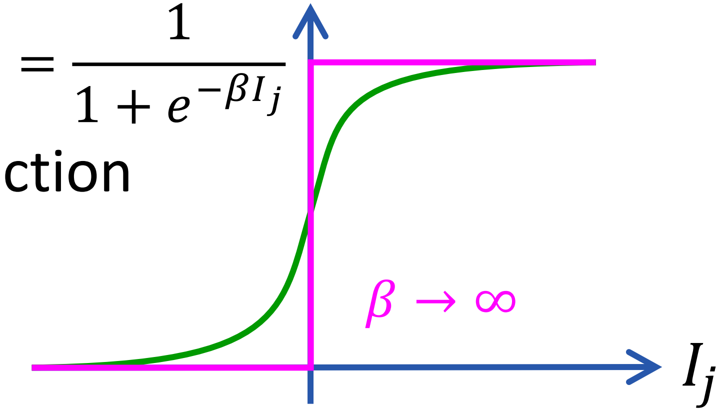




Logistic function

$$f(I_j) = \frac{1}{1 + e^{-\beta I_j}}$$

Sigmoid function



$$I_j = \sum_i w_{i,j} x_i + \theta_j$$

$$O_j = f(I_j) = \frac{1}{1 + e^{-\beta I_j}}$$

$$\beta \to \infty : \quad f(I_j) = \mathrm{sign}(I_j)$$

Neural Network

Topology:

# of units in the input layer

# of hidden layers

# of units in each hidden layer

# of units in the output layer

Parameters:

$w_{i,j}$ for each connection

β, θ_j for each hidden or output unit

Neural networks can closely approximate any function

Given enough hidden units and enough training samples

Neural Network as a Classifier

Weakness	Strength
Long learning time	High tolerance to noisy data
Some parameters are determined empirically, e.g., the network topology or “structure”	Well-suited for continuous-values inputs and outputs
Poor interpretability: Difficult to interpret the weights of the hidden units	Successful on an array of real-world data, e.g., hand-written letters
	Inherently parallel

Neural Network	SVM
Non-deterministic algorithm	Deterministic algorithm
Can easily be learned in incremental fashion	Hard to learn: learned in batch mode using quadratic programming techniques
To learn complex functions—use multilayer perceptron (nontrivial)	Using kernels can learn very complex functions

How to Train the Neural Network?

Backpropagation Algorithm: A neural network learning algorithm

- For each training data object, the weights $w_{i,j}$ are adjusted to minimize the mean squared error between prediction and ground truth

- Adjustments are made in the "**backwards**" direction: from the output layer , to the hidden layer

How to Train the Neural Network?

Backpropagation Algorithm: A neural network learning algorithm

Steps:

1. Initialize weights $w_{i,j}$ and biases θ_j to small random numbers
2. For each training data object
   1) Propagate the inputs forward
   2) Backpropagate the error by updating weights $w_{i,j}$ and biases θ_j
3. Terminating condition : when the error is very small, etc.

Backpropagation Algorithm

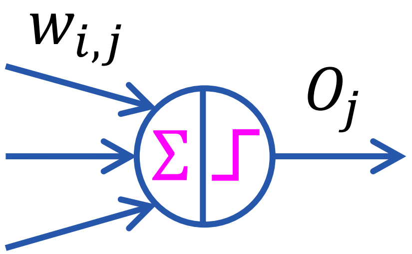
The error:

$$\mathrm{Err}_j = O_j(1 - O_j)(T_j - O_j)$$

where, O_j is the actual output

T_j is the known value

$O_j(1 - O_j)$ is the derivative of the logistic function



Unit j in the output layer

Update the weights:

$$\Delta w_{i,j} = l \cdot \mathrm{Err}_j \cdot O_i$$
$$w_{i,j} = w_{i,j} + \Delta w_{i,j}$$

Update the bias:

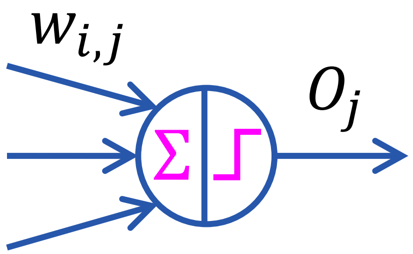
$$\Delta \theta_j = l \cdot \mathrm{Err}_j$$
$$\theta_j = \theta_j + \Delta \theta_j$$

$l \in (0,1)$: learning rate

Rule of thumb: $l = \dfrac{1}{t}$

t is # of iterations through the training set so far

Backpropagation Algorithm



Unit j in the hidden layer

The error:

$$\mathrm{Err}_j = O_j(1 - O_j) \sum_k w_{j,k} \cdot \mathrm{Err}_k$$

where, O_j is the actual output

$w_{j,k}$ is the weight of the connection from unit j to k in higher layer

Err_k is the error of unit k

Update the weights:

$$\Delta w_{i,j} = l \cdot \mathrm{Err}_j \cdot O_i$$
$$w_{i,j} = w_{i,j} + \Delta w_{i,j}$$

Update the bias:

$$\Delta \theta_j = l \cdot \mathrm{Err}_j$$
$$\theta_j = \theta_j + \Delta \theta_j$$

$l \in (0,1)$: learning rate

Rule of thumb: $l = \frac{1}{t}$

t is # of iterations through the training set so far

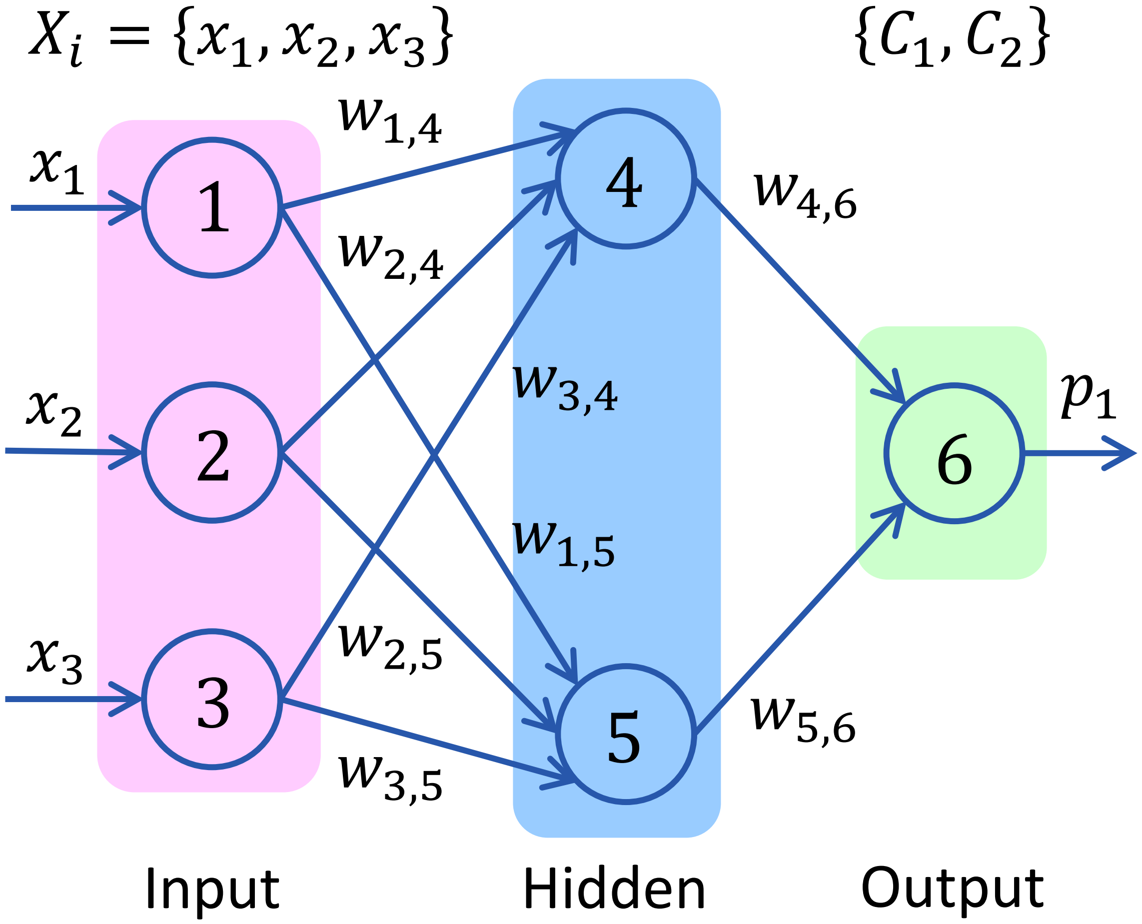
Training data object:

$X_i = \{1, 0, 1\}$

class label: 1

Initial values:

$w_{1,4}$	0.2
$w_{1,5}$	-0.3
$w_{2,4}$	0.4
$w_{2,5}$	0.1
$w_{3,4}$	-0.5
$w_{3,5}$	0.2
$w_{4,6}$	-0.3
$w_{5,6}$	-0.2
θ_4	-0.4
θ_5	0.2
θ_6	0.1



Multilayer feed-forward networks

Learning rate $l = 0.9$

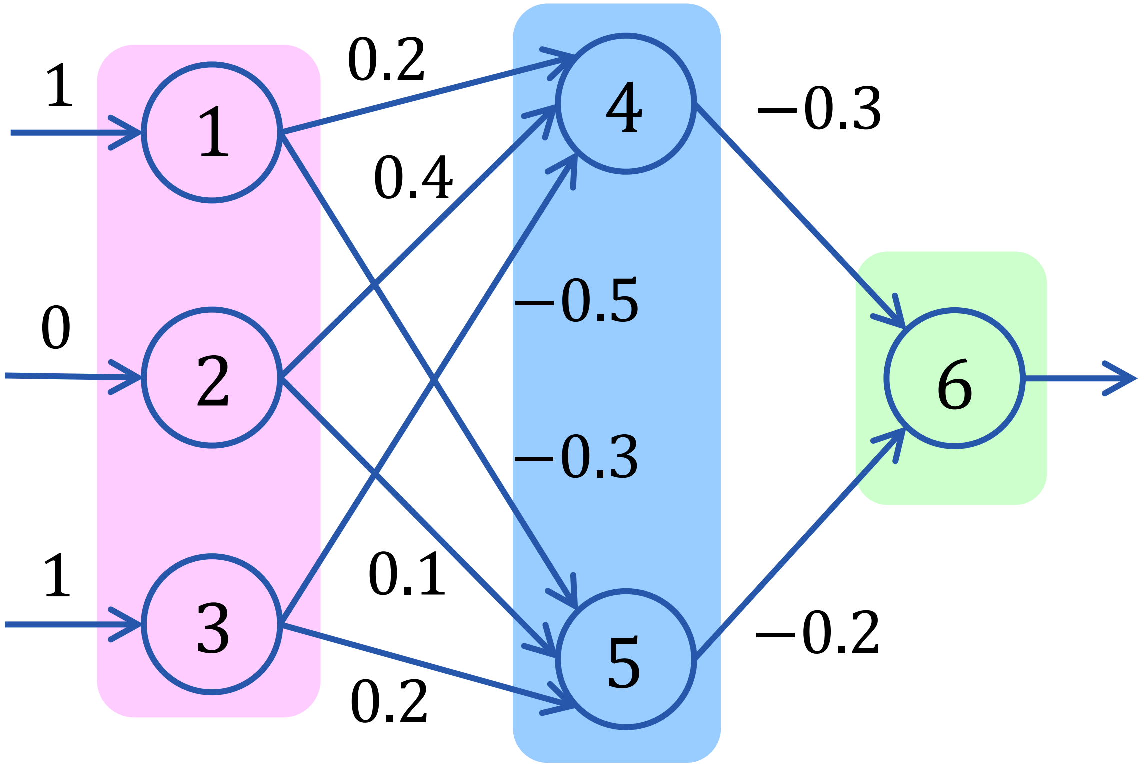
Training data object:

$X_i = \{1, 0, 1\}$

class label: 1

Initial values:

$w_{1,4}$	0.2
$w_{1,5}$	-0.3
$w_{2,4}$	0.4
$w_{2,5}$	0.1
$w_{3,4}$	-0.5
$w_{3,5}$	0.2
$w_{4,6}$	-0.3
$w_{5,6}$	-0.2
θ_4	-0.4
θ_5	0.2
θ_6	0.1



Unit j	**Net Input I_j**	**Output O_j**
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$\frac{1}{1+e^{0.7}} = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$\frac{1}{1+e^{-0.1}} = 0.525$
6	$-0.3 \times 0.332 - 0.2 \times 0.525 + 0.1 = -0.105$	$\frac{1}{1+e^{0.105}} = 0.474$

Propagate the inputs forward $I_j = \sum_i w_{i,j} x_i + \theta_j$ $O_j = f(I_j) = \frac{1}{1+e^{-\beta I_j}}$

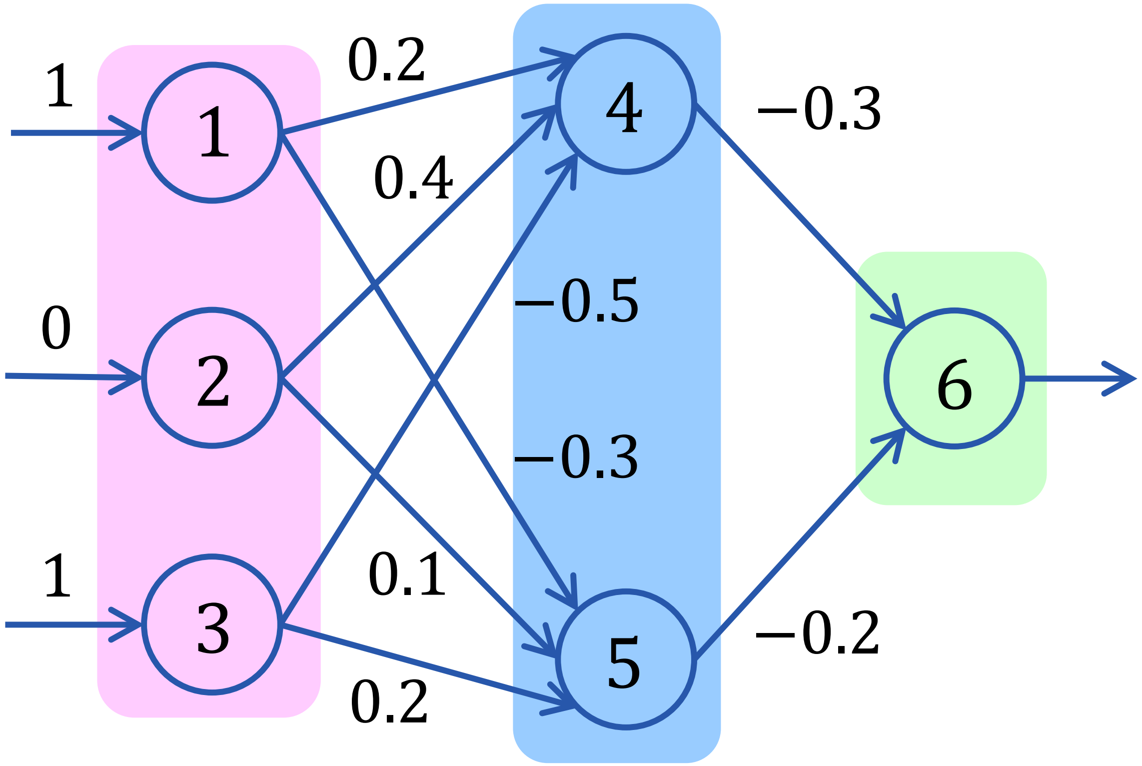
Training data object:

$X_i = \{1, 0, 1\}$

class label: 1

Initial values:

$w_{1,4}$	0.2
$w_{1,5}$	-0.3
$w_{2,4}$	0.4
$w_{2,5}$	0.1
$w_{3,4}$	-0.5
$w_{3,5}$	0.2
$w_{4,6}$	-0.3
$w_{5,6}$	-0.2
θ_4	-0.4
θ_5	0.2
θ_6	0.1



Unit j	Error Err_j
6	$0.474 \times (1 - 0.474) \times (1 - 0.474) = 0.1311$
5	$0.525 \times (1 - 0.525) \times 0.1311 \times (-0.2) = -0.0065$
4	$0.332 \times (1 - 0.332) \times 0.1311 \times (-0.3) = -0.0087$

Backpropagate the errors

Output layer: $\mathrm{Err}_j = O_j(1 - O_j)(T_j - O_j)$

Hidden layer: $\mathrm{Err}_j = O_j(1 - O_j)\sum_k w_{j,k} \cdot \mathrm{Err}_k$

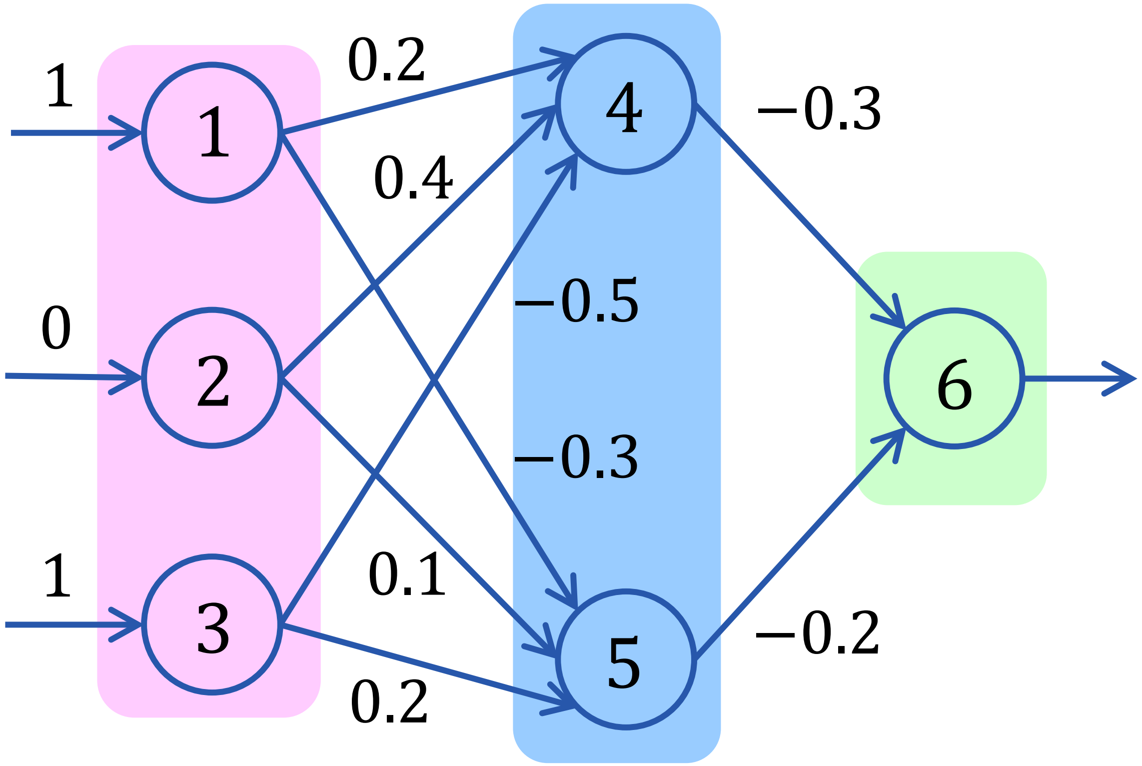
Training data object:

$X_i = \{1, 0, 1\}$

class label: 1

Initial values:

$w_{1,4}$	0.2
$w_{1,5}$	-0.3
$w_{2,4}$	0.4
$w_{2,5}$	0.1
$w_{3,4}$	-0.5
$w_{3,5}$	0.2
$w_{4,6}$	-0.3
$w_{5,6}$	-0.2
θ_4	-0.4
θ_5	0.2
θ_6	0.1



weights	**Error** Err_j
$w_{4,6}$	$-0.3 + 0.9 \times 0.1311 \times 0.332 = -0.261$
$w_{5,6}$	$-0.2 + 0.9 \times 0.1311 \times 0.525 = -0.138$

Backpropagate the errors

Learning rate $l = 0.9$

Update the weights: $\Delta w_{i,j} = l \cdot \mathrm{Err}_j \cdot O_i$

$w_{i,j} = w_{i,j} + \Delta w_{i,j}$

Ge

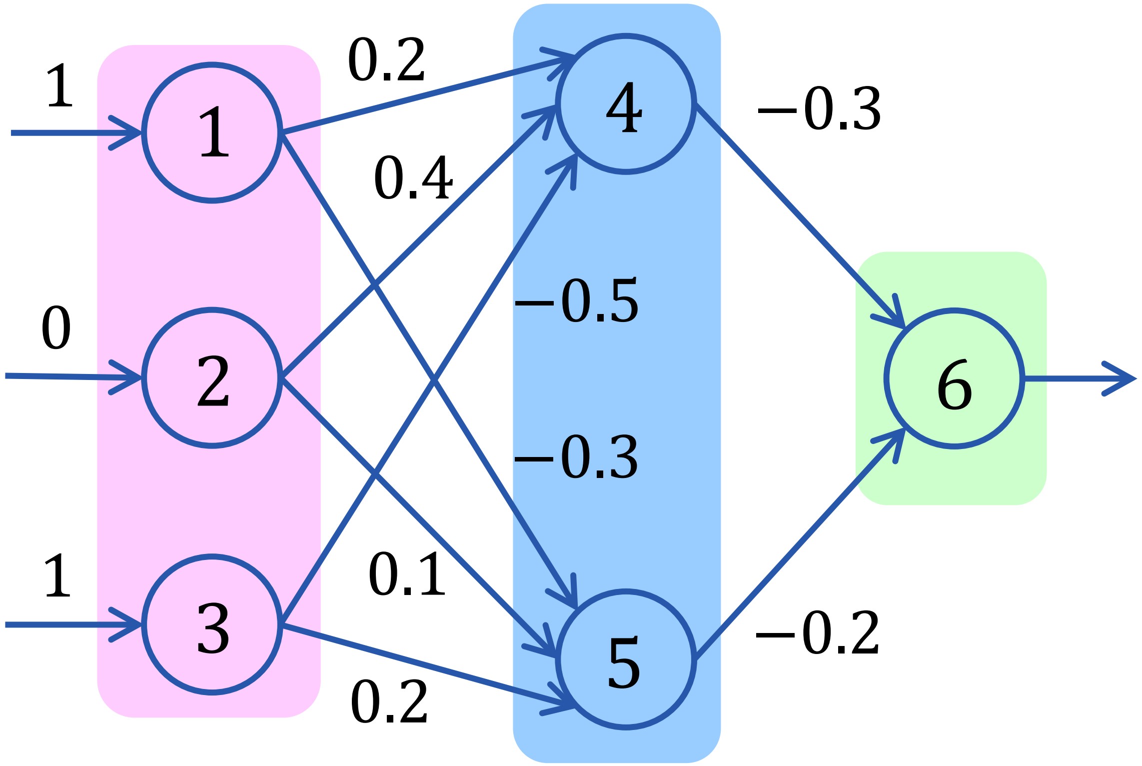
Training data object:

$X_i = \{1, 0, 1\}$

class label: 1

Initial values:

$w_{1,4}$	0.2
$w_{1,5}$	-0.3
$w_{2,4}$	0.4
$w_{2,5}$	0.1
$w_{3,4}$	-0.5
$w_{3,5}$	0.2
$w_{4,6}$	-0.3
$w_{5,6}$	-0.2
θ_4	-0.4
θ_5	0.2
θ_6	0.1



weights	Error Err_j
$w_{1,4}$	$0.2 + 0.9 \times (-0.0087) \times 1 = 0.192$
$w_{2,4}$	$0.4 + 0.9 \times (-0.0087) \times 0 = 0.4$
$w_{3,4}$	$-0.5 + 0.9 \times (-0.0087) \times 1 = -0.508$

Backpropagate the errors

Learning rate $l = 0.9$

Update the weights: $\Delta w_{i,j} = l \cdot \mathrm{Err}_j \cdot O_i$

$w_{i,j} = w_{i,j} + \Delta w_{i,j}$

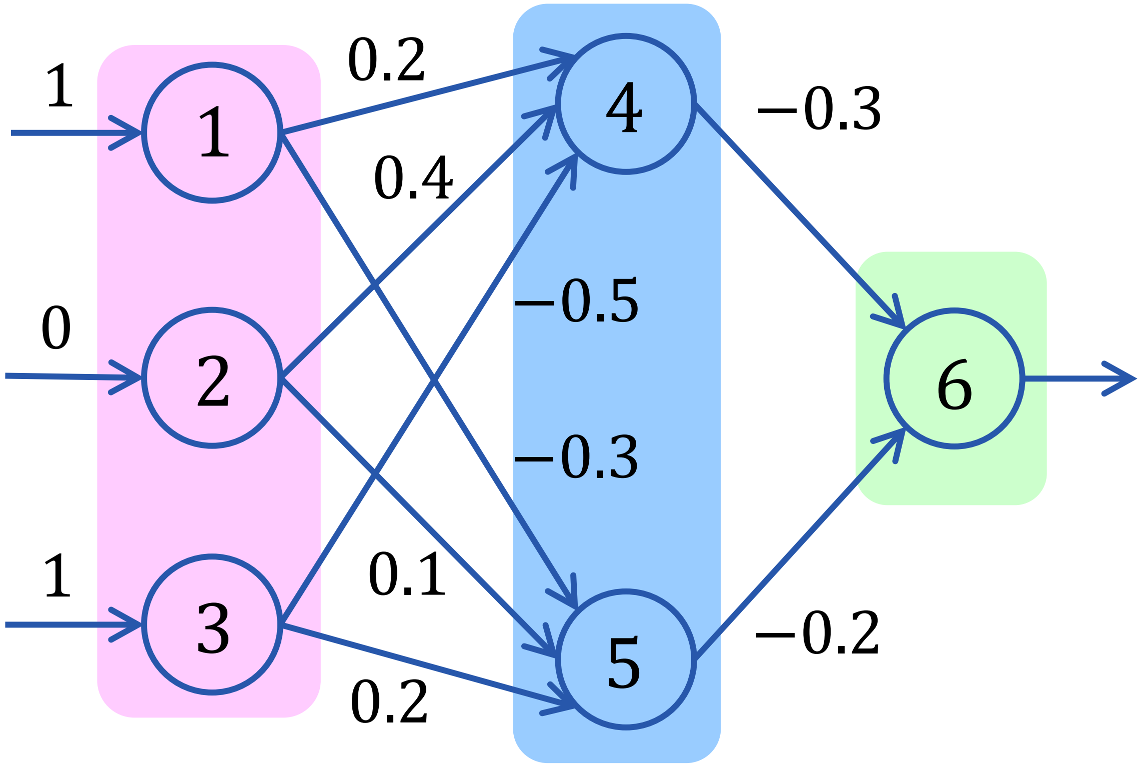
Training data object:

$X_i = \{1, 0, 1\}$

class label: 1

Initial values:

$w_{1,4}$	0.2
$w_{1,5}$	-0.3
$w_{2,4}$	0.4
$w_{2,5}$	0.1
$w_{3,4}$	-0.5
$w_{3,5}$	0.2
$w_{4,6}$	-0.3
$w_{5,6}$	-0.2
θ_4	-0.4
θ_5	0.2
θ_6	0.1



weights	Error Err_j
$w_{1,5}$	$-0.3 + 0.9 \times (-0.0065) \times 1 = -0.306$
$w_{2,5}$	$0.1 + 0.9 \times (-0.0065) \times 0 = 0.1$
$w_{3,5}$	$0.2 + 0.9 \times (-0.0065) \times 1 = 0.194$

Backpropagate the errors

Learning rate $l = 0.9$

Update the weights: $\Delta w_{i,j} = l \cdot \mathrm{Err}_j \cdot O_i$

$w_{i,j} = w_{i,j} + \Delta w_{i,j}$

Ge

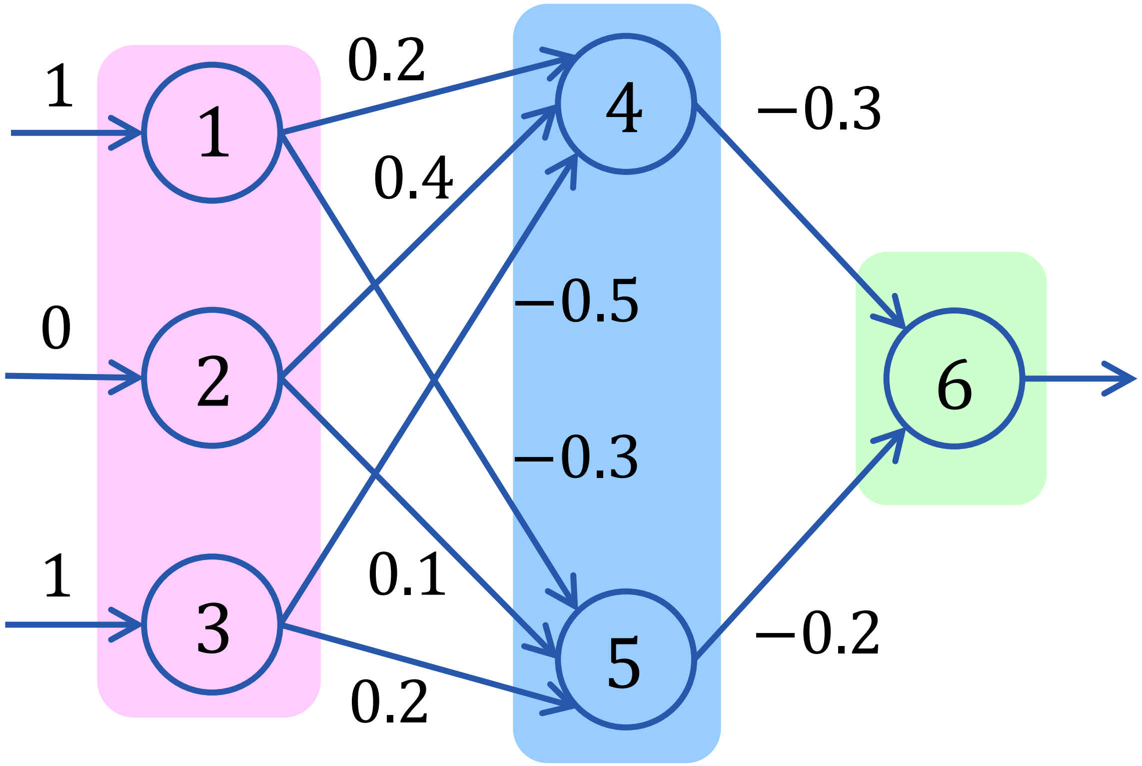
Training data object:

$X_i = \{1, 0, 1\}$

class label: 1

Initial values:

$w_{1,4}$	0.2
$w_{1,5}$	-0.3
$w_{2,4}$	0.4
$w_{2,5}$	0.1
$w_{3,4}$	-0.5
$w_{3,5}$	0.2
$w_{4,6}$	-0.3
$w_{5,6}$	-0.2
θ_4	-0.4
θ_5	0.2
θ_6	0.1



weights	Error Err_j
θ_6	$0.1 + 0.9 \times 0.1311 = 0.218$
θ_5	$0.2 + 0.9 \times (-0.0065) = 0.194$
θ_4	$-0.4 + 0.9 \times (-0.0087) = -0.408$

Backpropagate the errors

Learning rate $l = 0.9$

Update the biases: $\Delta\theta_j = l \cdot \mathrm{Err}_j$

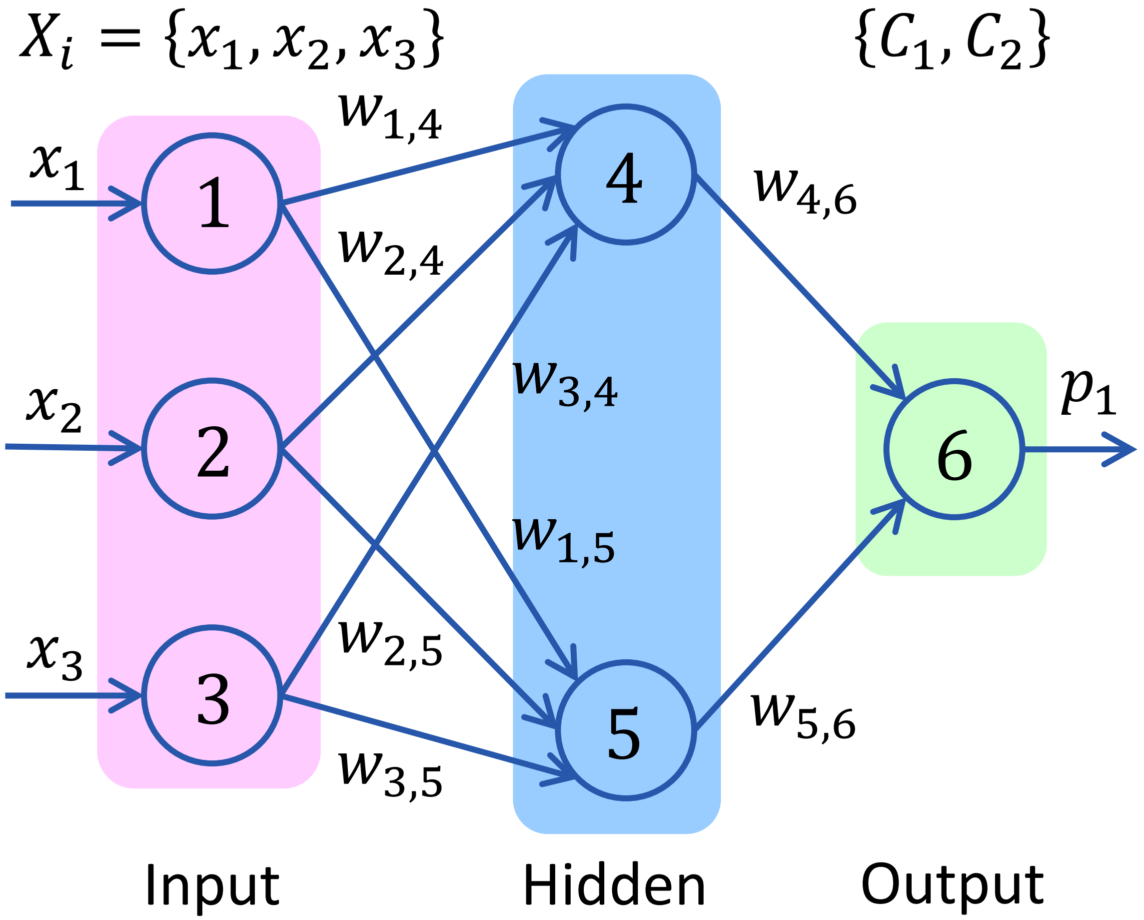
$\theta_j = \theta_j + \Delta\theta_j$

Training data object:

$X_i = \{1, 0, 1\}$ label: 1

Initial values:

weights	old	new
$w_{1,4}$	0.2	0.192
$w_{1,5}$	-0.3	-0.306
$w_{2,4}$	0.4	0.4
$w_{2,5}$	0.1	0.1
$w_{3,4}$	-0.5	-0.508
$w_{3,5}$	0.2	0.194
$w_{4,6}$	-0.3	-0.261
$w_{5,6}$	-0.2	-0.138
θ_4	-0.4	-0.408
θ_5	0.2	0.194
θ_6	0.1	0.218

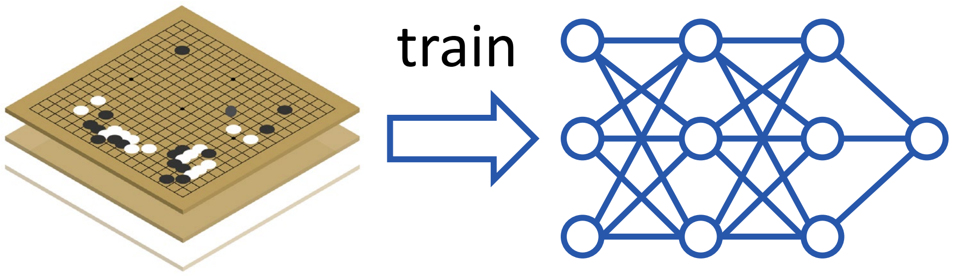


Multilayer feed-forward networks

Learning rate $l = 0.9$

Lee Sedol 9-dan vs AlphaGo

AlphaGo 4 – Lee Sedol 1



- Human expert positions
- Self-play positions

Mastering the game of Go with deep neural networks and tree search
David Silver, et al
Nature, 2016

Deep Learning Package Zoo

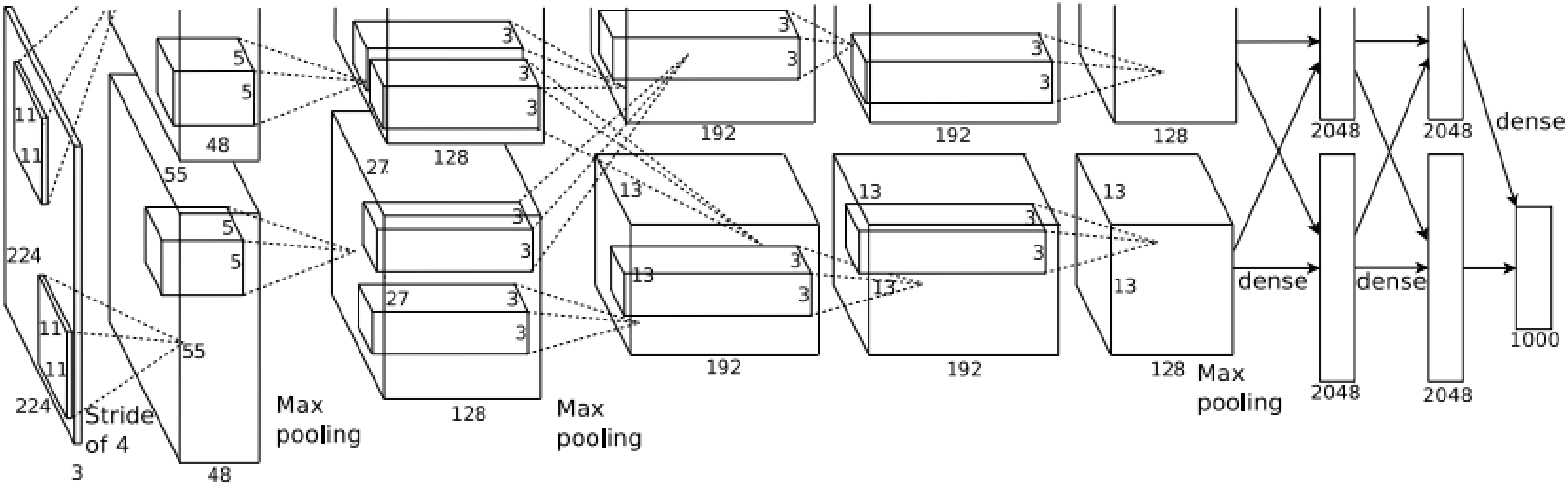
- Torch => PyTorch
- Caffe
- Theano (Keras, Lasagne)
- nVidia CuDNN
- Google TensorFlow
- Mxnet
- etc.

https://www.tensorflow.org/tutorials
https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf
https://cbmm.mit.edu/sites/default/files/documents/Tensorflow%202_0%20slides.pdf
https://www.slideshare.net/databricks/introduction-to-tensorflow-20

Pre-Trained Models

- AlexNet
- VGG-16 (Visual Geometry Group - University of Oxford)
- BERT (Bidirectional Encoder Representations from Transformers)
- GPT (Generative Pre-trained Transformer) / GPT2 / GPT3

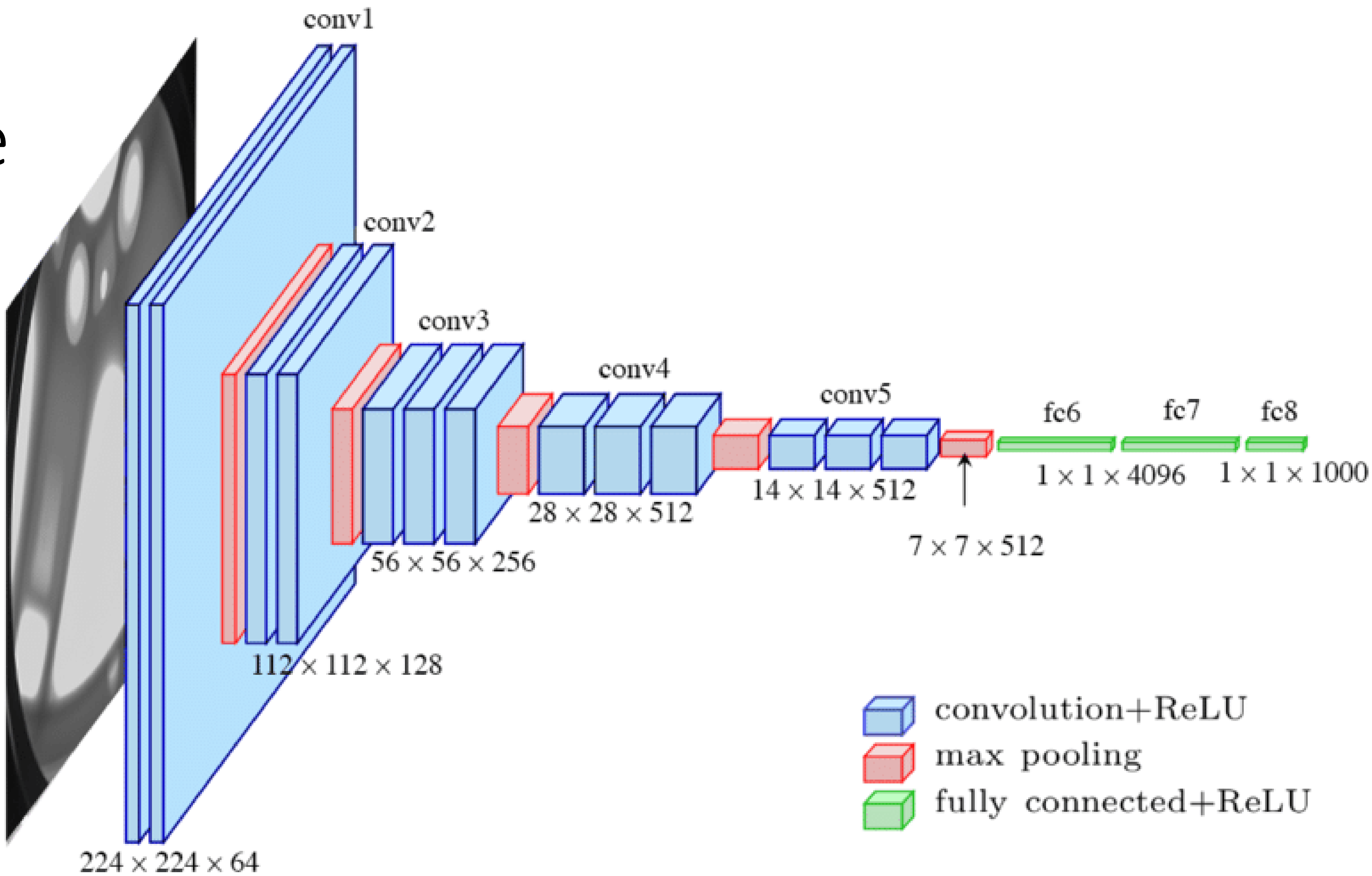
Architecture of AlexNet CNN



Georgia State University

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012): 1097-1105.

VGG-16 Architecture

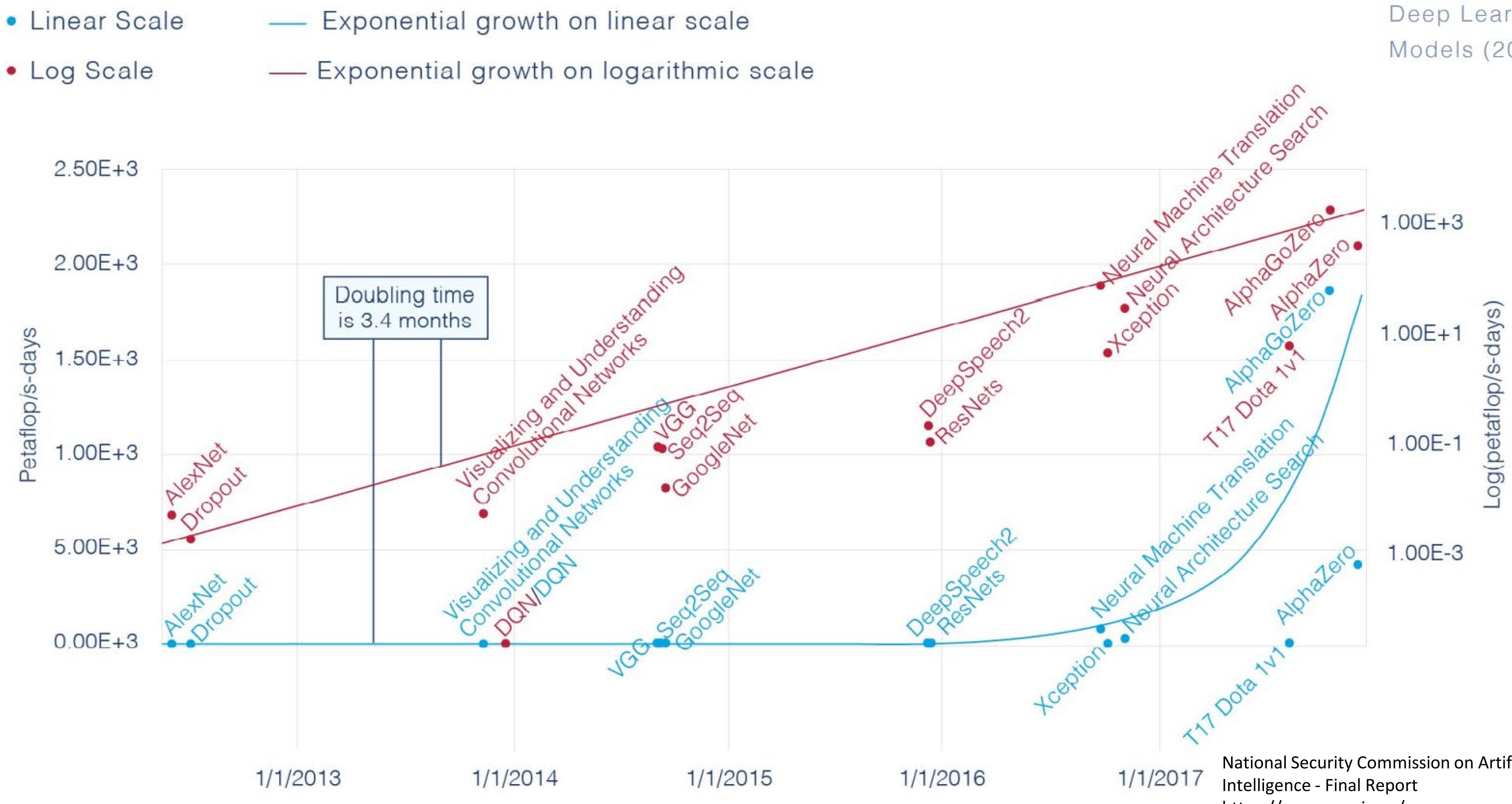


Georgia State University

Ferguson, Max, Ronay Ak, Yung-Tsun Tina Lee, and Kincho H. Law. "Automatic localization of casting defects with convolutional neural networks." In 2017 IEEE international conference on big data (big data), pp. 1726-1735. IEEE, 2017.

Compute Required to Train Largest Deep Learning Models (2012-2017)

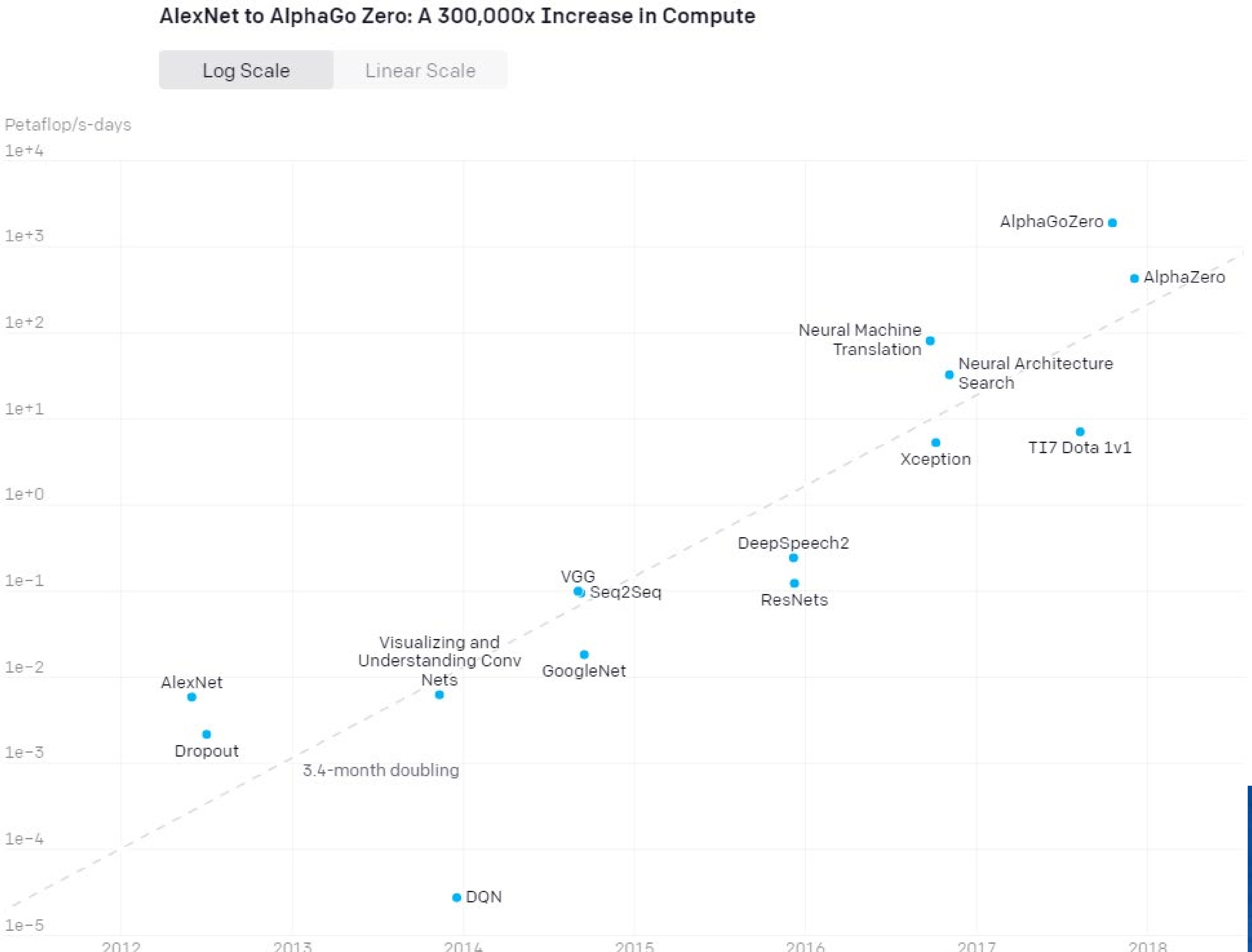
Compute Required to Train Largest Deep Learning Models (2012-2017)



Source: OpenAI, AI and Compute (May 16, 2018), https://openai.com/blog/ai-and-compute/

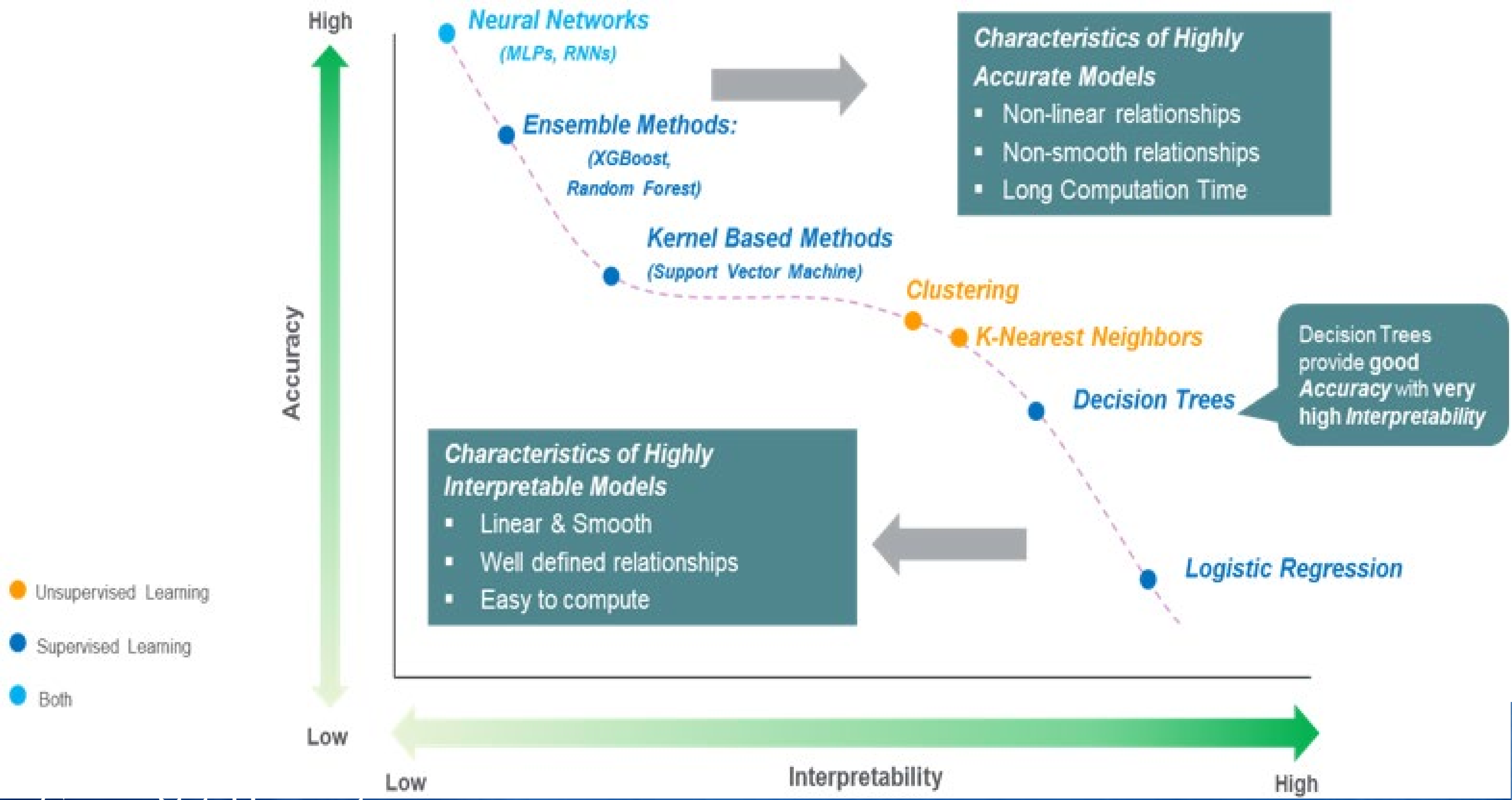
National Security Commission on Artificial Intelligence - Final Report
https://www.nscai.gov/wp-content/uploads/2021/03/Full-Report-Digital-1.pdf

AI & Compute



AI and Compute - OpenAI https://openai.com/blog/ai-and-compute/

Georgia State University


High
Accuracy
Low
Neural Networks
(MLPs, RNNs)
Ensemble Methods:
(XGBoost,
Random Forest)
Kernel Based Methods
(Support Vector Machine)
Clustering
K-Nearest Neighbors
Decision Trees
Logistic Regression
Characteristics of Highly Accurate Models
Non-linear relationships
Non-smooth relationships
Long Computation Time
Decision Trees provide good Accuracy with very high Interpretability
Characteristics of Highly Interpretable Models
Linear & Smooth
Well defined relationships
Easy to compute
Unsupervised Learning
Supervised Learning
Both
Low
Interpretability
High

k-Nearest Neighbor (k-NN) Classifier

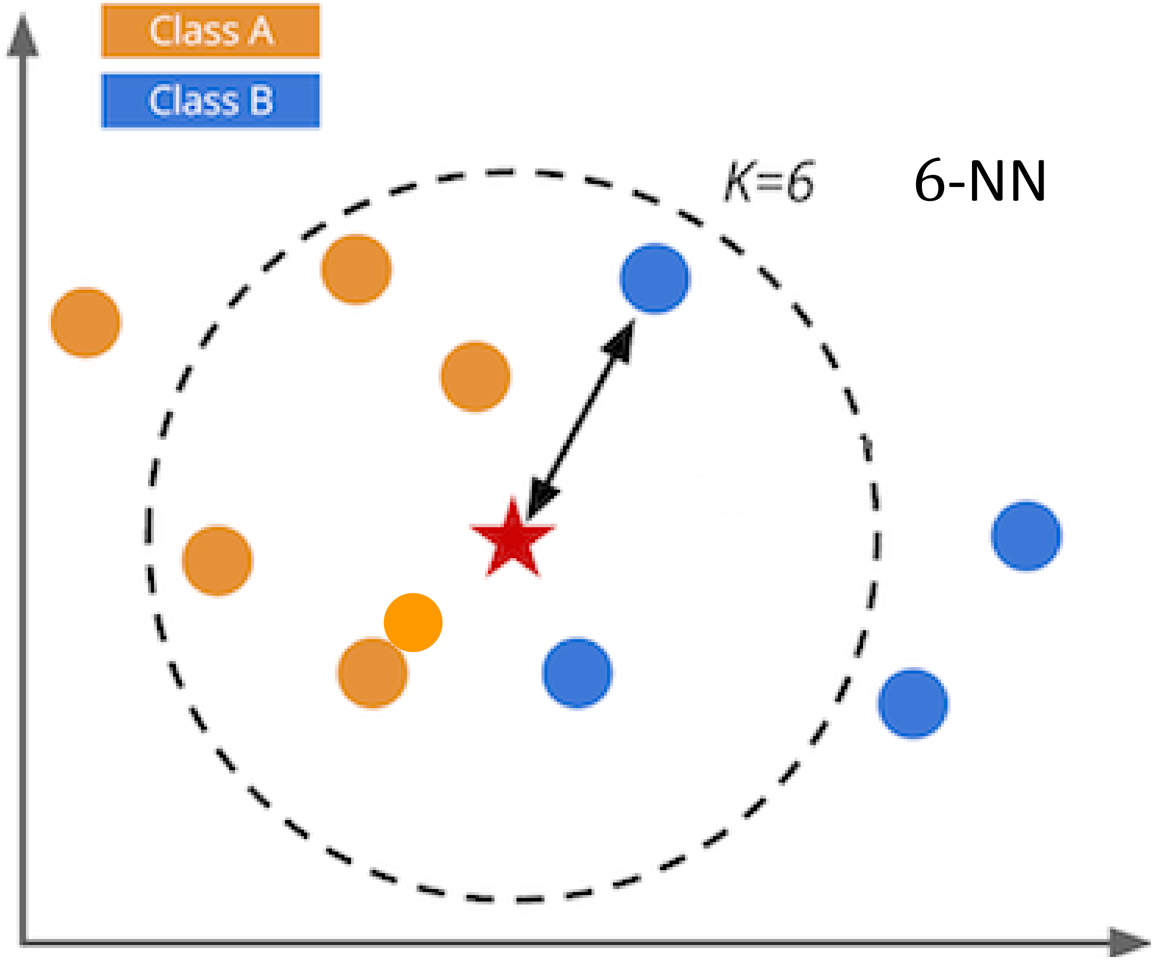
1. Compute the Distance/Similarity

$$d_{i,j} = \sqrt{\left(x_{i,1} - x_{j,1}\right)^2 + \left(x_{i,2} - x_{j,2}\right)^2}$$

2. Rank the remaining nodes

3. Select the Top-k nodes

4. Majority Vote

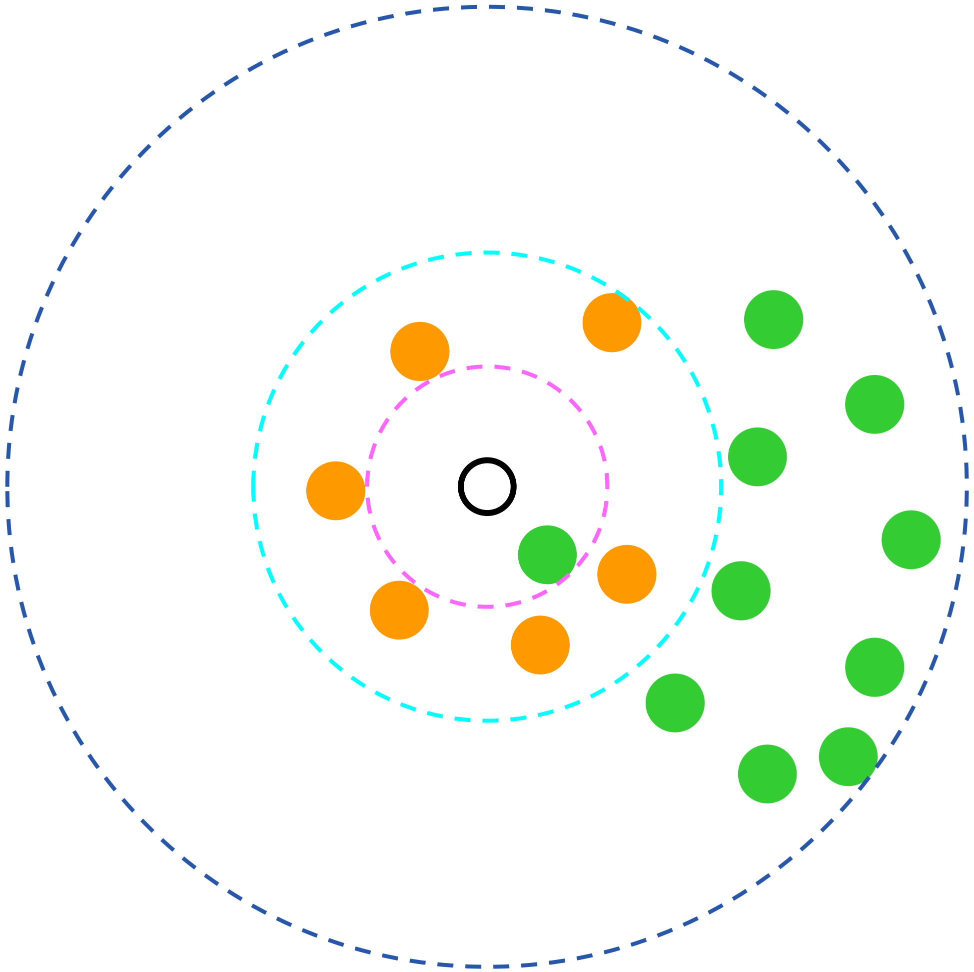


How to choose parameter k?

"too small" k: sensitive to outliers

"too large" k: many objects from other classes

medium k: highest accuracy



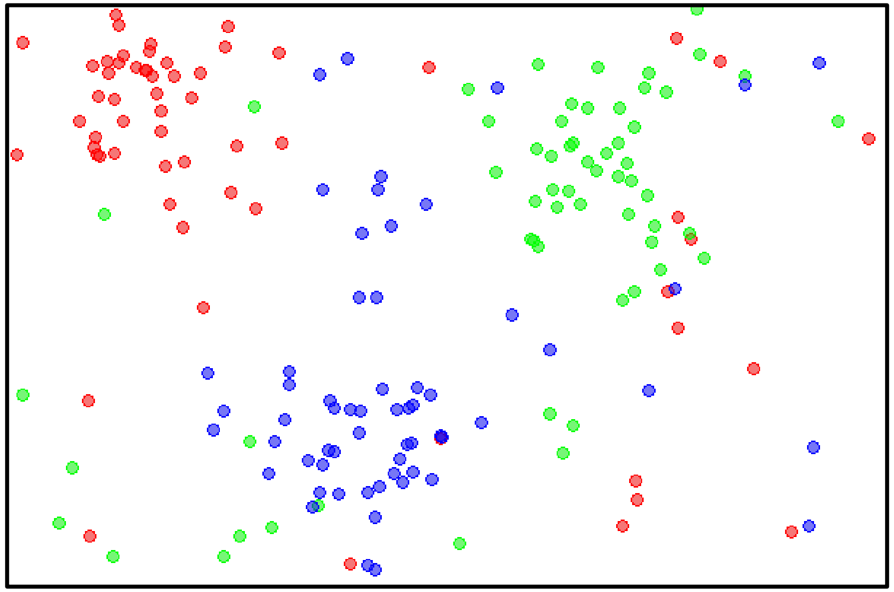
Decision set for $k = 1$

Decision set for $k = 7$

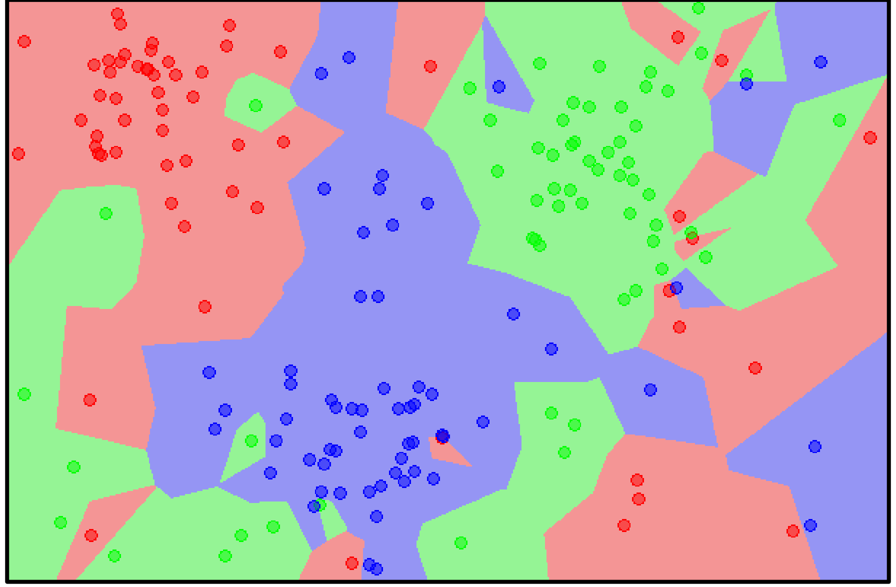
Decision set for $k = 16$

Difference Between 1-NN and 5-NN

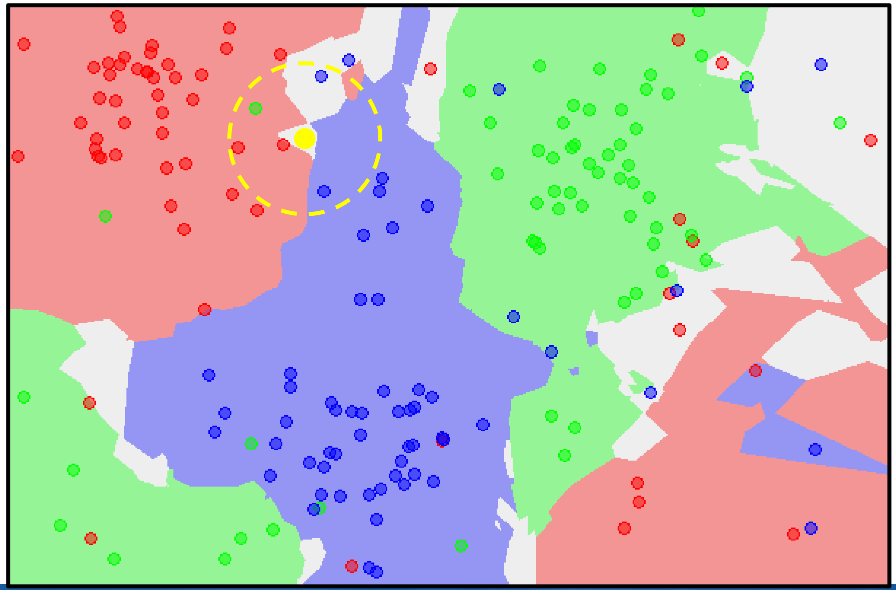
Original dataset



1-NN



5-NN



White Region: class votes are tied

{red, red, blue, blue, green}