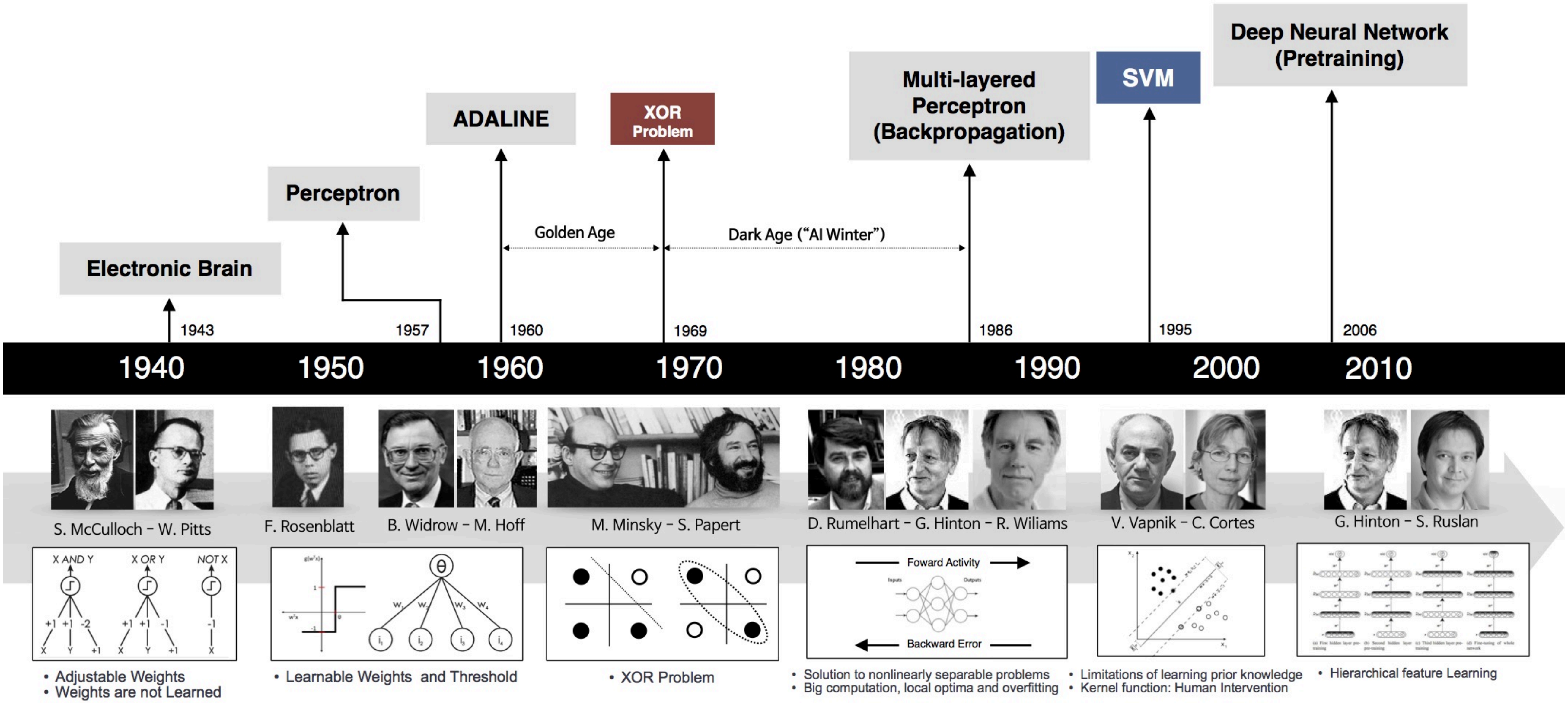
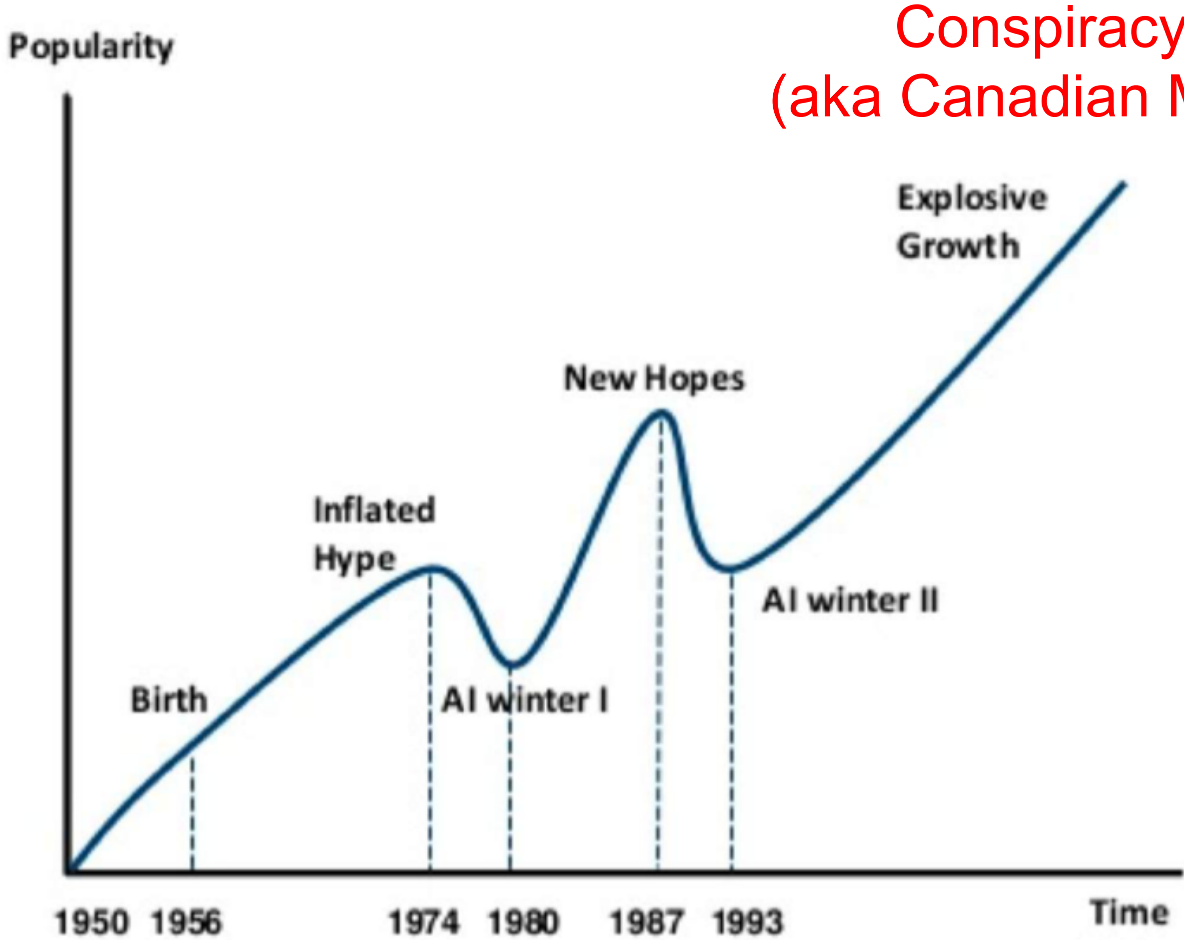
PMF Machine Learning
Introduction to Deep Learning

dr. sc. Tomislav Lipić
Rudjer Boskovic Institute
Laboratory for Machine Learning and Knowledge Representation

History of Artificial Intelligence (AI)

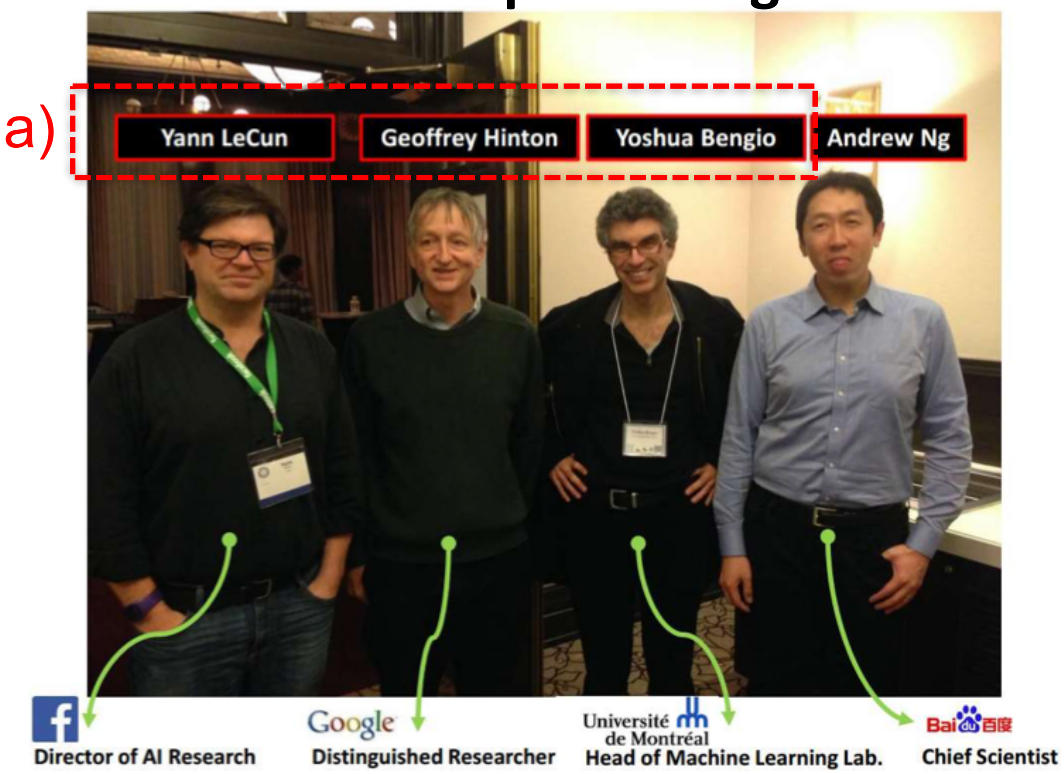


Artificial Intelligence (AI) Hypes
=> We are now in ‘Deep Learning (DL) Hype’



“Deep Learning Conspiracy” (aka Canadian Mafia)

‘Heroes of Deep Learning‘:



Ian Goodfellow, Andrej Karpathy, to name a few…

(http://cs231n.github.io/)

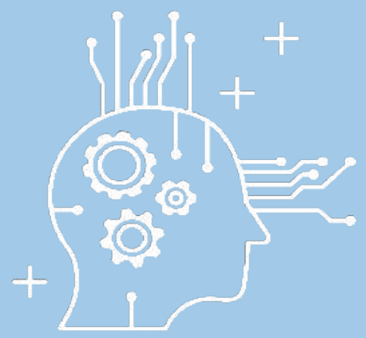
Recommended readings:

- **LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton**. Deep learning. Nature 521.7553 (2015): 436.
- **J. Schmidhuber**. Deep Learning in Neural Networks: An Overview. Neural Networks, Volume 61, January 2015, Pages 85-117 (DOI: 10.1016/j.neunet.2014.09.003
  - detailed preprint: https://arxiv.org/abs/1404.7828 (88 pages, 888 references)

What is Deep Learning (DL)?

Artificial Intelligence

Any technique that enables computers to mimic human behavior

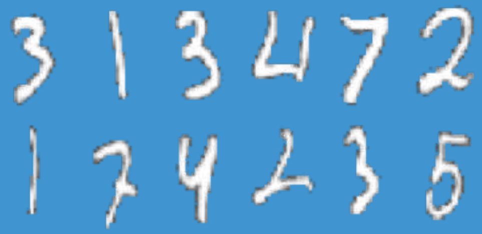


Machine Learning

Ability to learn without explicitly being programmed

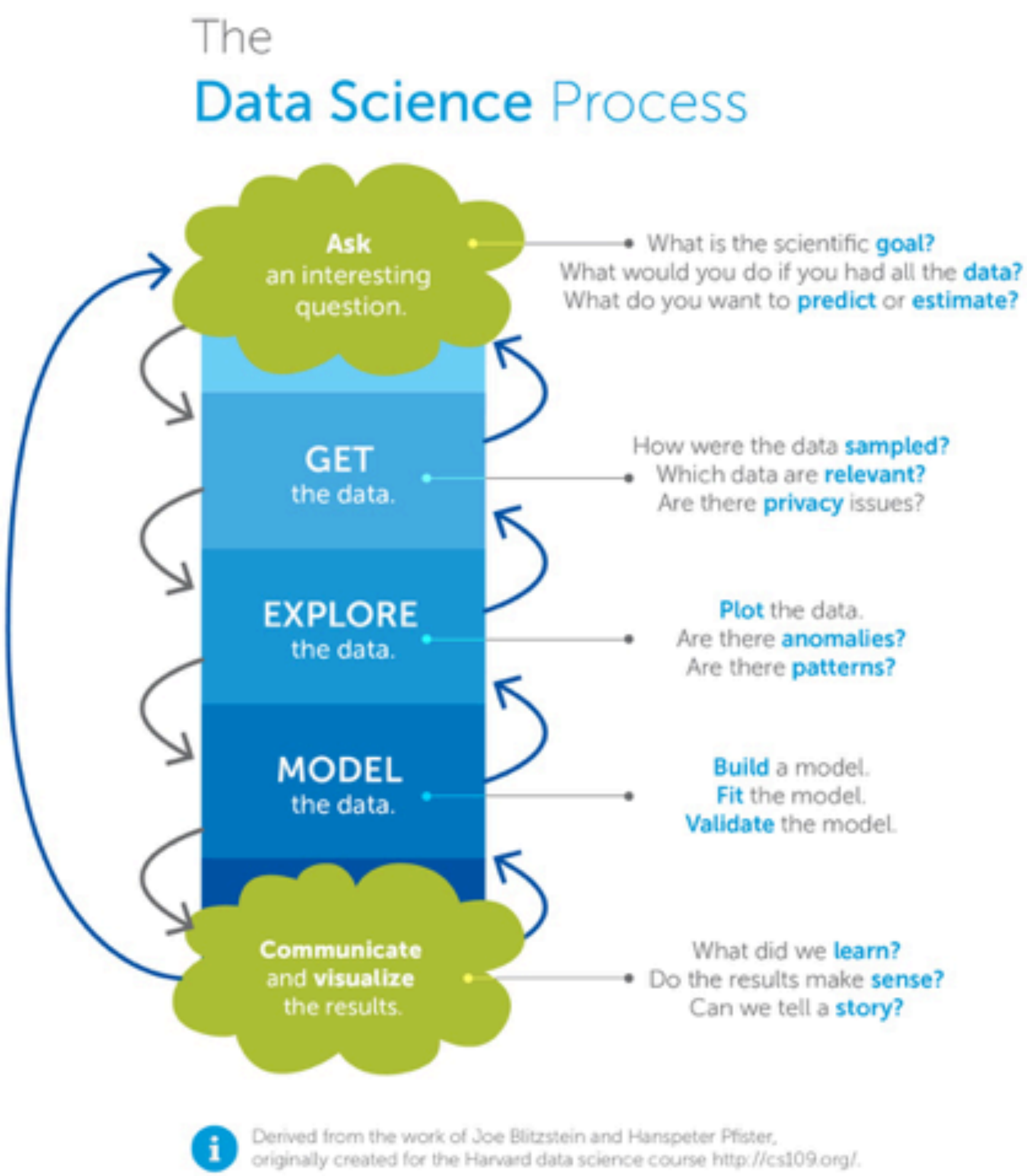
Deep Learning

Extract patterns from data using neural networks

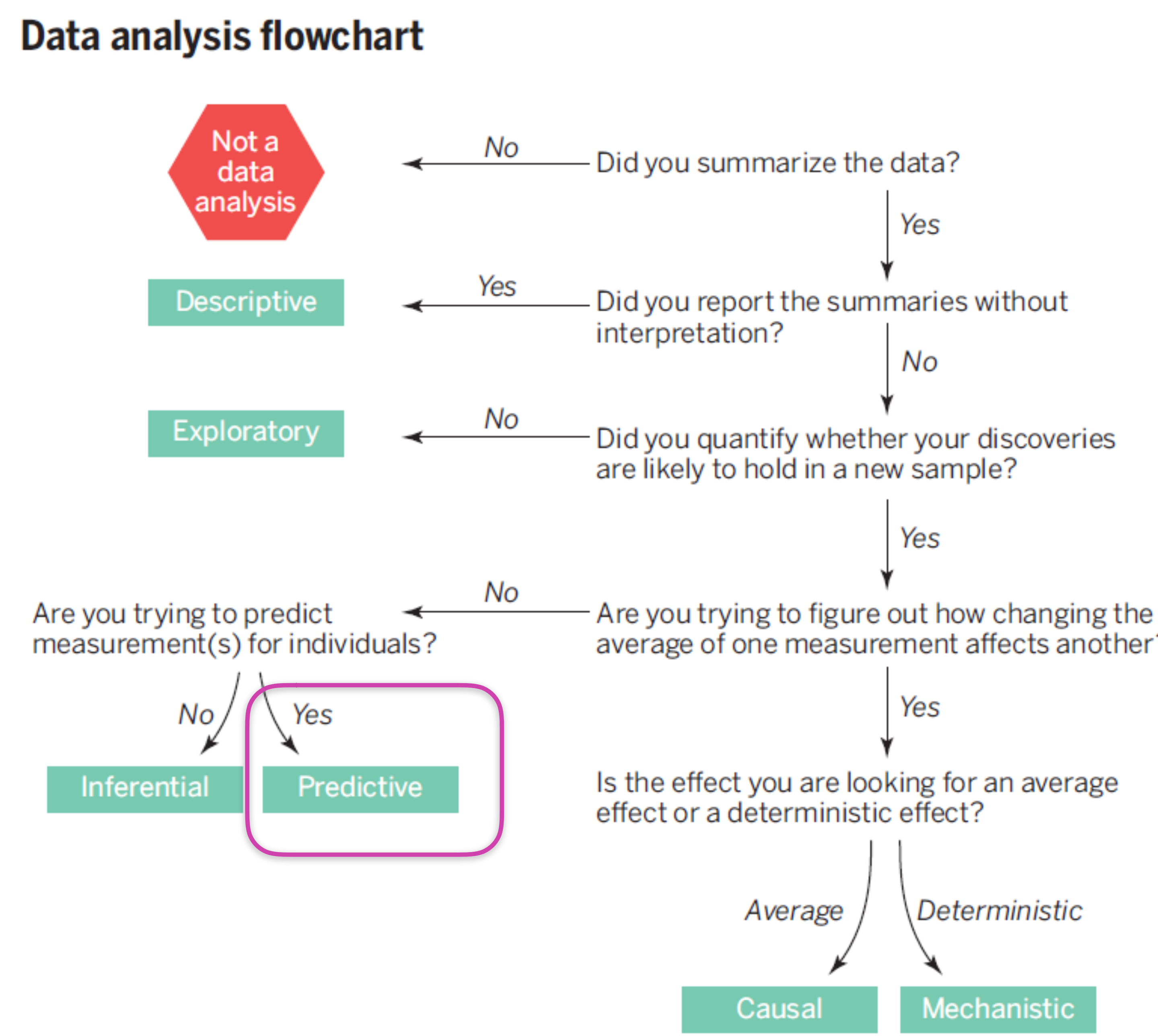


Data Science (DS)
vs
Machine Learning(ML)
vs
Deep Learning (DL)

Data Science Process



Jeffery T. Leek *and* **Roger D. Peng,**

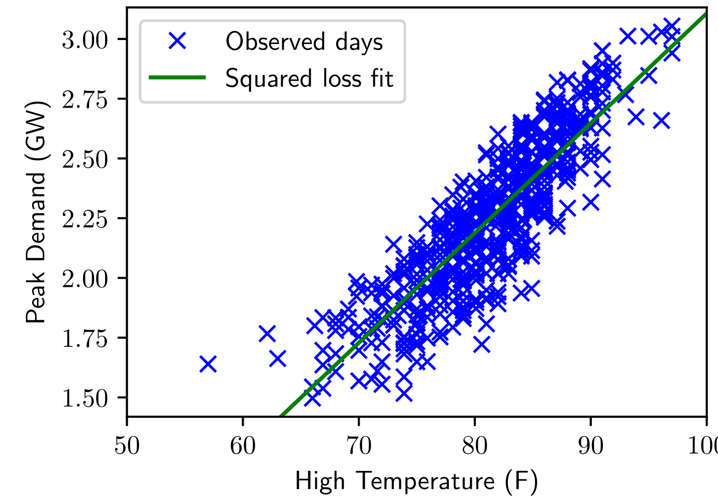


Predictive data analysis uses a subset of measurements (the features) to predict another measurement (the outcome) on a single person or unit.

Machine Learning

Basic idea: in many domains, it is difficult to hand-build a predictive model, but easy to collect lots of data; machine learning provides a way to **automatically infer the predictive model from data**

Date	High Temperature (F)	Peak Demand (GW)
2011-06-01	84.0	2.651
2011-06-02	73.0	2.081
2011-06-03	75.2	1.844
2011-06-04	84.9	1.959
…	…	…



The basic process [supervised learning]:

Training Data	Machine learning algorithm	Predictions
$(x^{(1)}, y^{(1)})$ $(x^{(2)}, y^{(2)})$ $(x^{(3)}, y^{(3)})$ $\vdots$	→ Hypothesis function $y^{(i)} \approx h(x^{(i)})$ →	New example x $\hat{y} = h(x)$

Input features: $x^{(i)} \in \mathbb{R}^n, i = 1, \ldots, m$

$$\mathrm{E.g.:}\ x^{(i)} = \begin{bmatrix} \mathrm{High\_Temperature}^{(i)} \\ \mathrm{Is\_Weekday}^{(i)} \\ 1 \end{bmatrix}$$

Outputs: $y^{(i)} \in \mathcal{Y}, i = 1, \ldots, m$

$$\mathrm{E.g.:}\ y^{(i)} \in \mathbb{R} = \ \mathrm{Peak\_Demand}^{(i)}$$

Hypothesis function: $h_\theta : \mathbb{R}^n \to \mathcal{Y}$, predicts output given input

$$\mathrm{E.g.:}\ h_\theta(x) = \sum_{j=1}^{n} \theta_j \cdot x_j$$

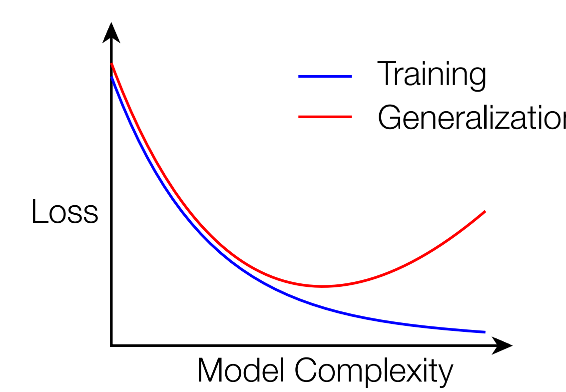
Loss function: $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$, measures the difference between a prediction and an actual output

$$\mathrm{E.g.:}\ \ell(\hat{y}, y) = (\hat{y} - y)^2$$

The canonical machine learning optimization problem:

$$\underset{\theta}{\mathrm{minimize}} \sum_{i=1}^{m} \ell(h_\theta(x^{(i)}), y^{(i)})$$

We really care about is how well our function (**model**) will **generalize** to *new examples*

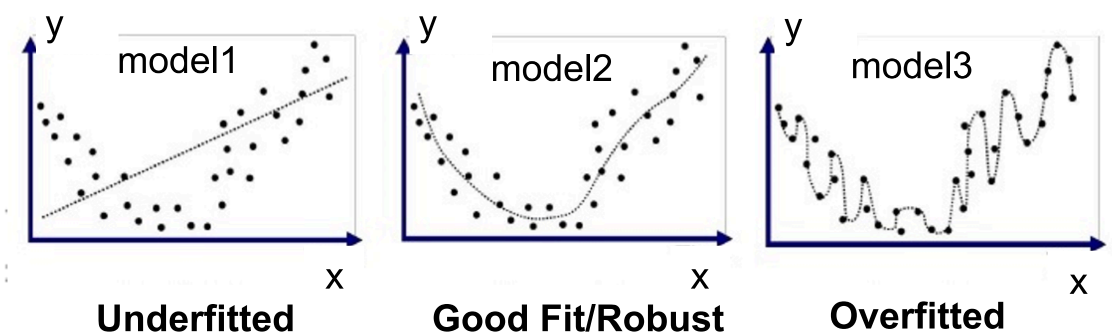


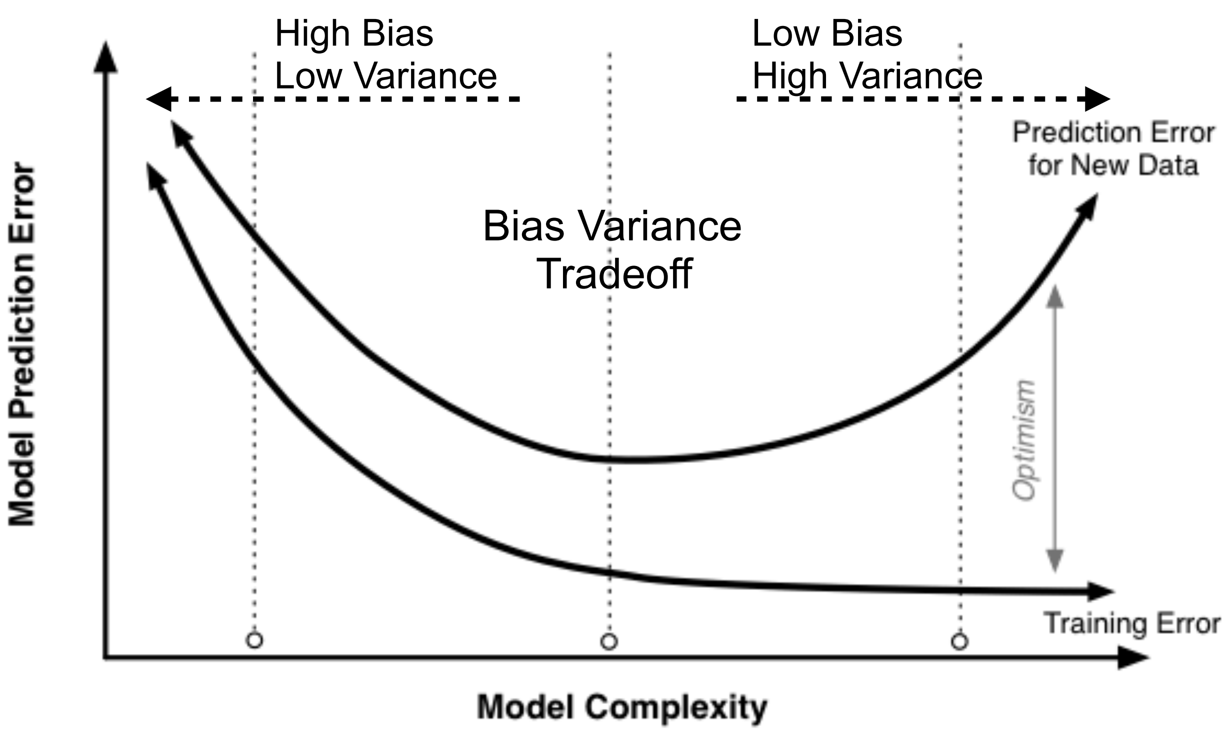
Machine Learning

The canonical machine learning problem is that we don't really care about minimizing this objective on the given data set (training data), we really care about how our learned model will **generalize to new (unseen) examples**.

Model: $y \approx \hat{f}(x)$

Good Generalization - Low Prediction Error on New Data:

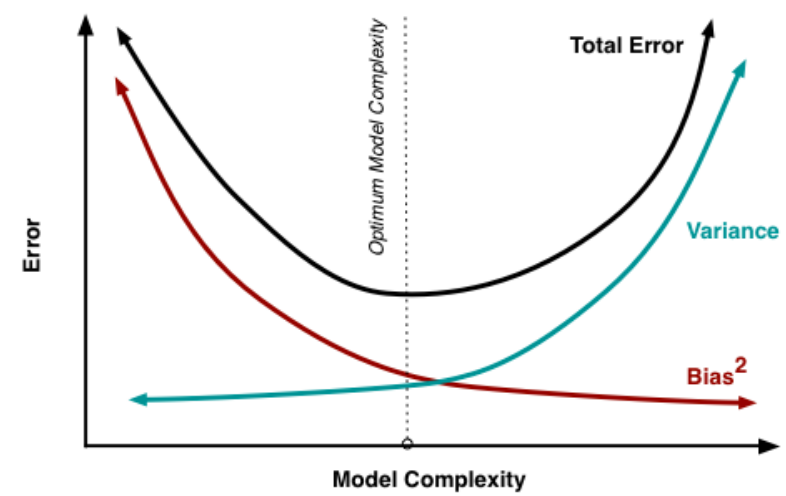


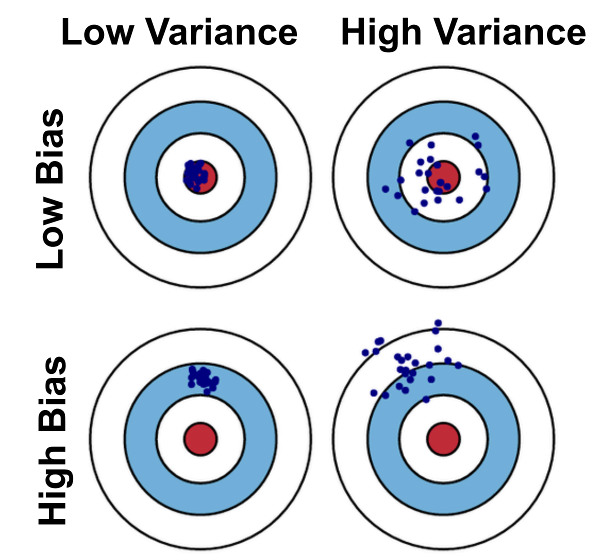


As model becomes more complex, training loss always decreases; **generalization loss decreases to a point**, then starts to increase.

Bias vs Variance Tradeoff:

http://scott.fortmann-roe.com/docs/BiasVariance.html





Total Error = Bias² + Variance + Irriducible Error

$$\mathrm{E}\left[\left(y-\hat{f}(x)\right)^2\right] = \left(\mathrm{Bias}\left[\hat{f}(x)\right]\right)^2 + \mathrm{Var}\left[\hat{f}(x)\right] + \sigma^2$$

Error due to Bias: The error due to bias is taken as the difference between the expected (or average) prediction of our model and the correct value which we are trying to predict.

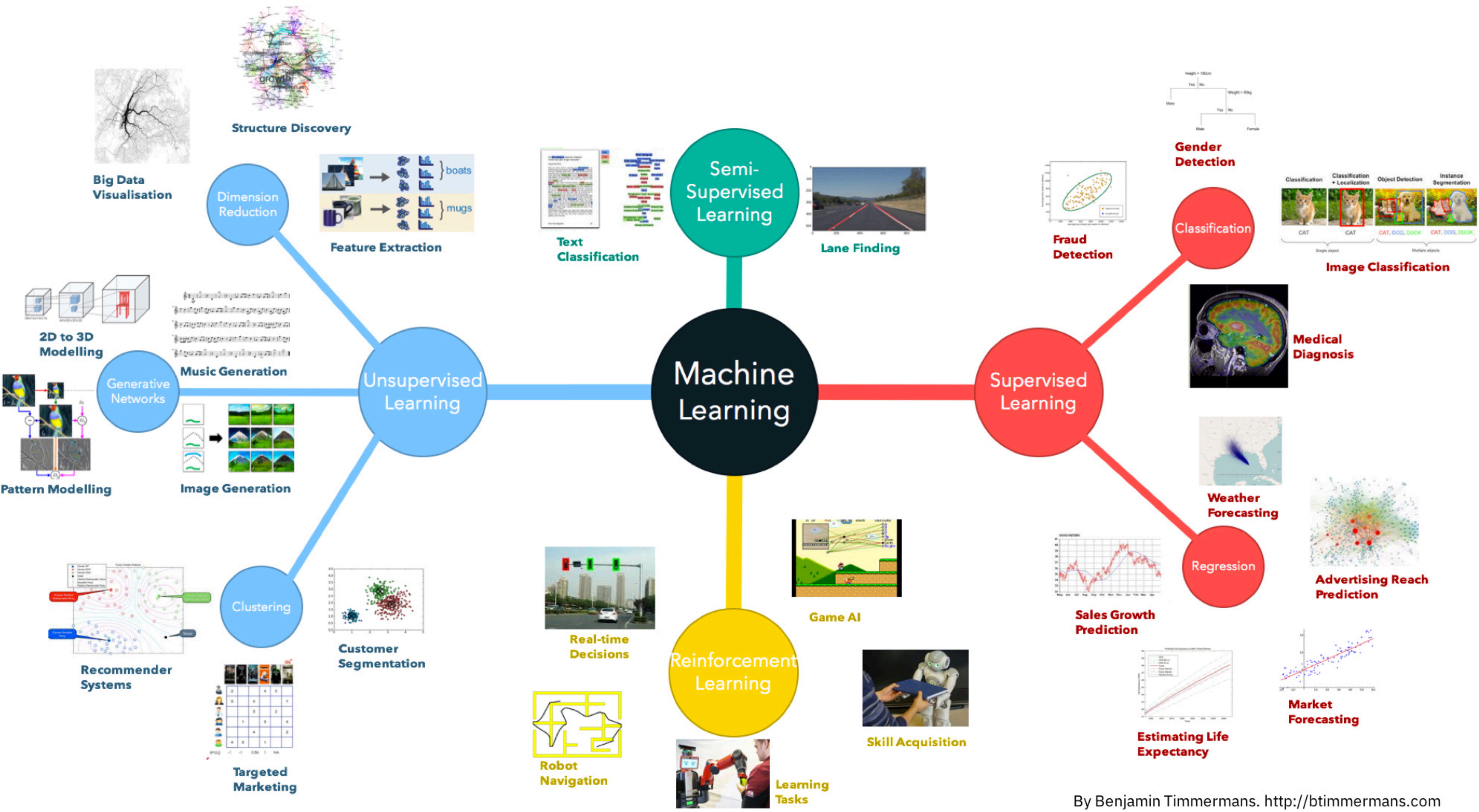
$$\mathrm{Bias}\left[\hat{f}(x)\right] = \mathrm{E}\left[\hat{f}(x)\right] - f(x)$$

Error due to Variance: The error due to variance is taken as the variability of a model prediction for a given data point

$$\mathrm{Var}\left[\hat{f}(x)\right] = \mathrm{E}[\hat{f}(x)^2] - \mathrm{E}[\hat{f}(x)]^2$$

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: springer.
Hastie, T., Tibhshirani, R. and M. Wainwright, (2015), Statistical Learning with Sparsity: The Lasso and Generalizations, Chapman and Hall.
Hastie, T., R. Tibshirani, and J. Friedman, (2011), The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition, Springer.

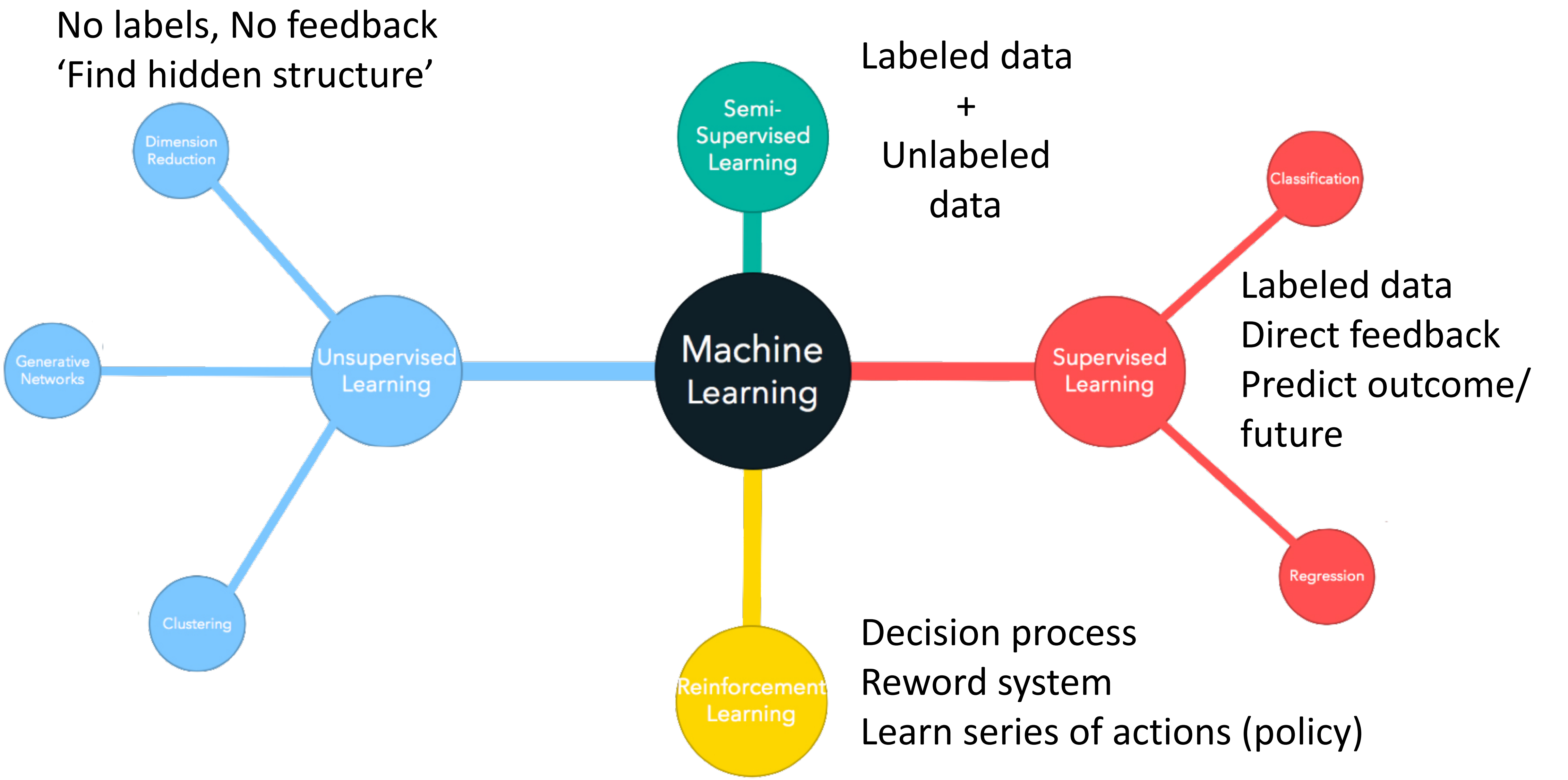
Machine Learning (ML) Overview: Examples of Tasks



By Benjamin Timmermans. http://btimmermans.com

Machine Learning (ML) Overview: Examples of Tasks

- can we do all these tasks with same learning algorithm (hint: NN?)

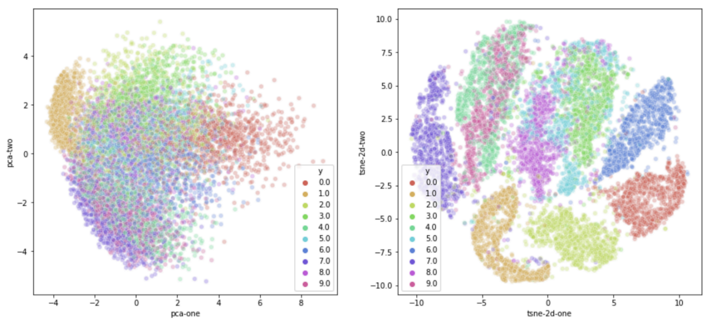


Recommended readings for refreshing ML:

- Notes from CS229: http://cs229.stanford.edu/syllabus.html
- Cheat-sheets for CS229: https://github.com/afshinea/stanford-cs-229-machine-learning

Machine Learning (ML) Overview: Examples of Tasks

MINST data: PCA and t-SNE 2d

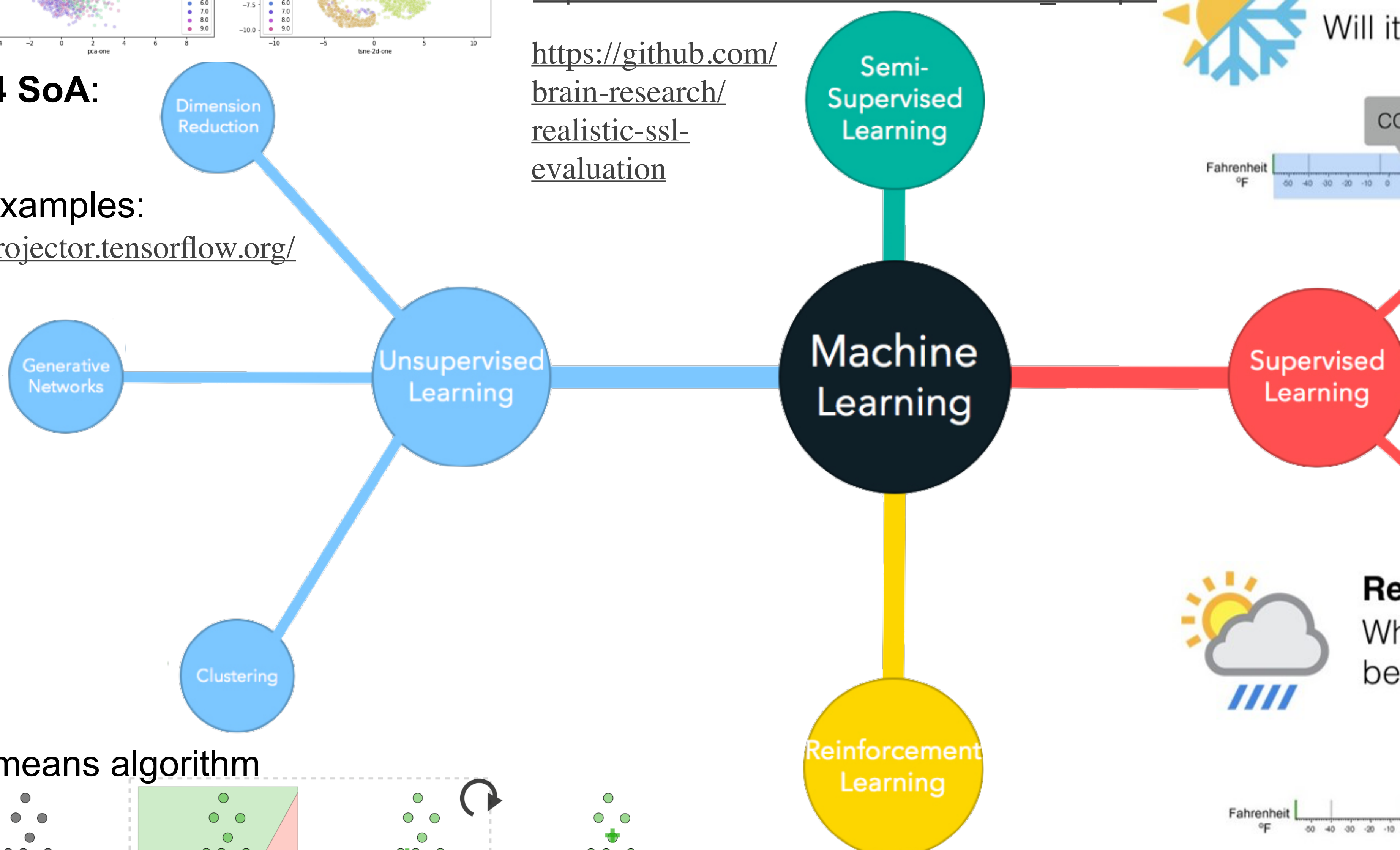


Look 4 SoA: UMAP

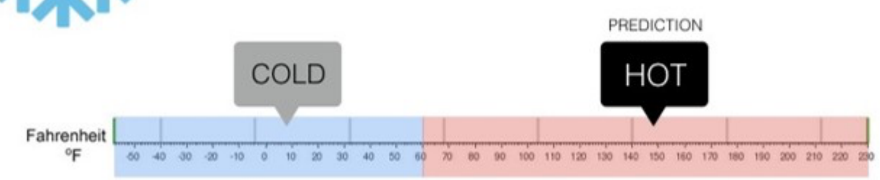
'Live' Examples:
https://projector.tensorflow.org/

http://www.cs.cmu.edu/~10701/slides/17_SSL.pdf

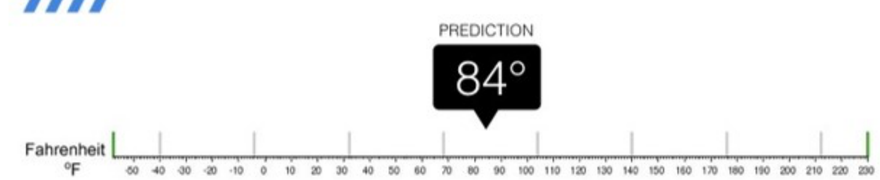
https://github.com/brain-research/realistic-ssl-evaluation



Classification
Will it be Cold or Hot tomorrow?

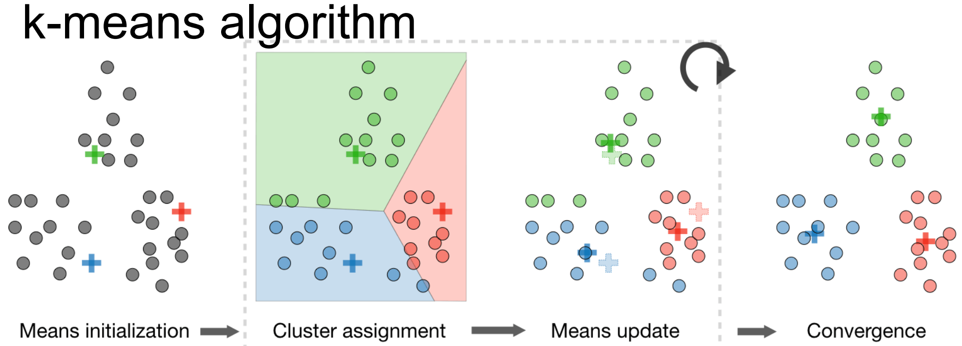


Regression
What is the temperature going to be tomorrow?



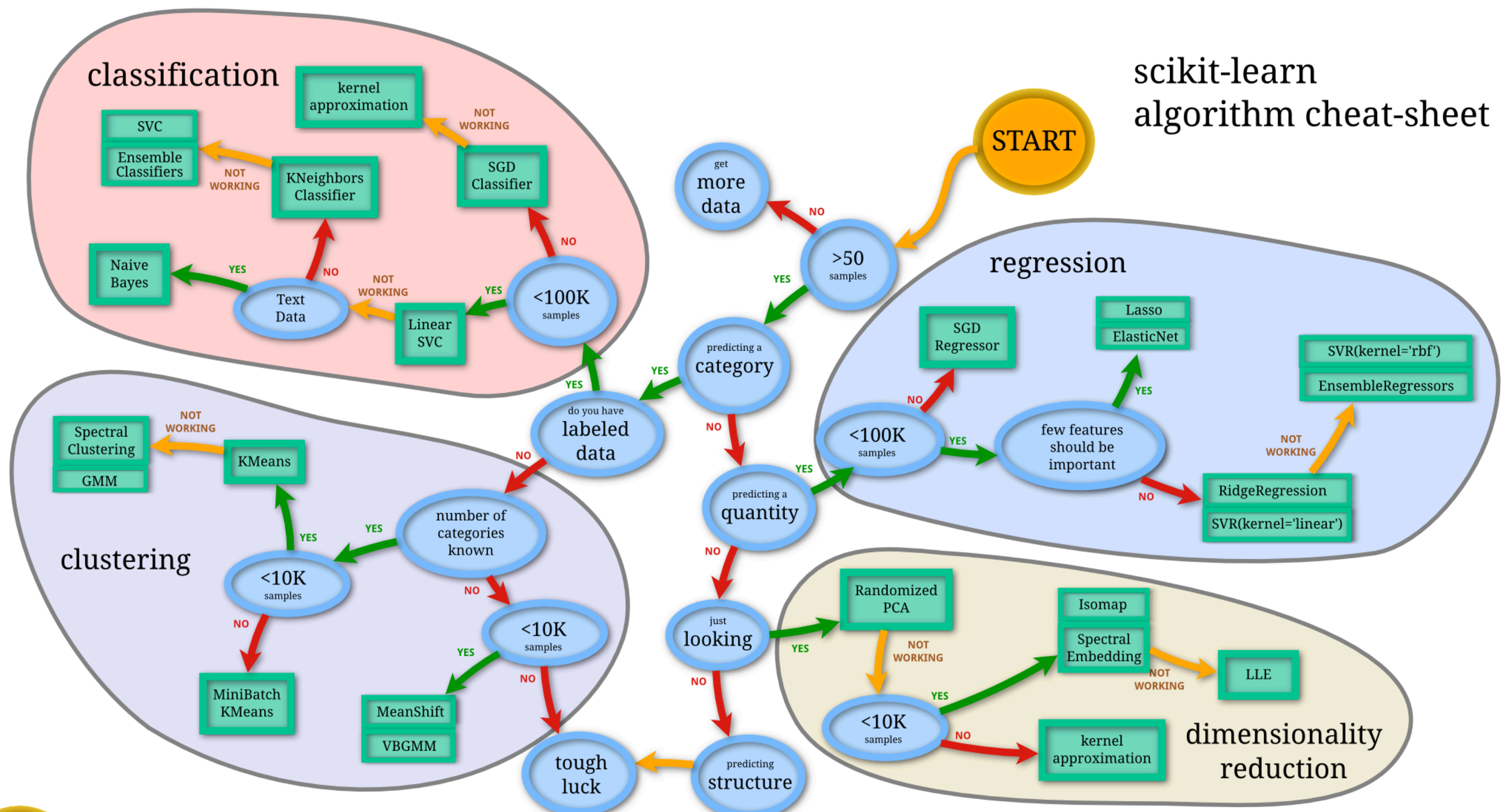
By Benjamin Timmermans. http://btimmermans.com

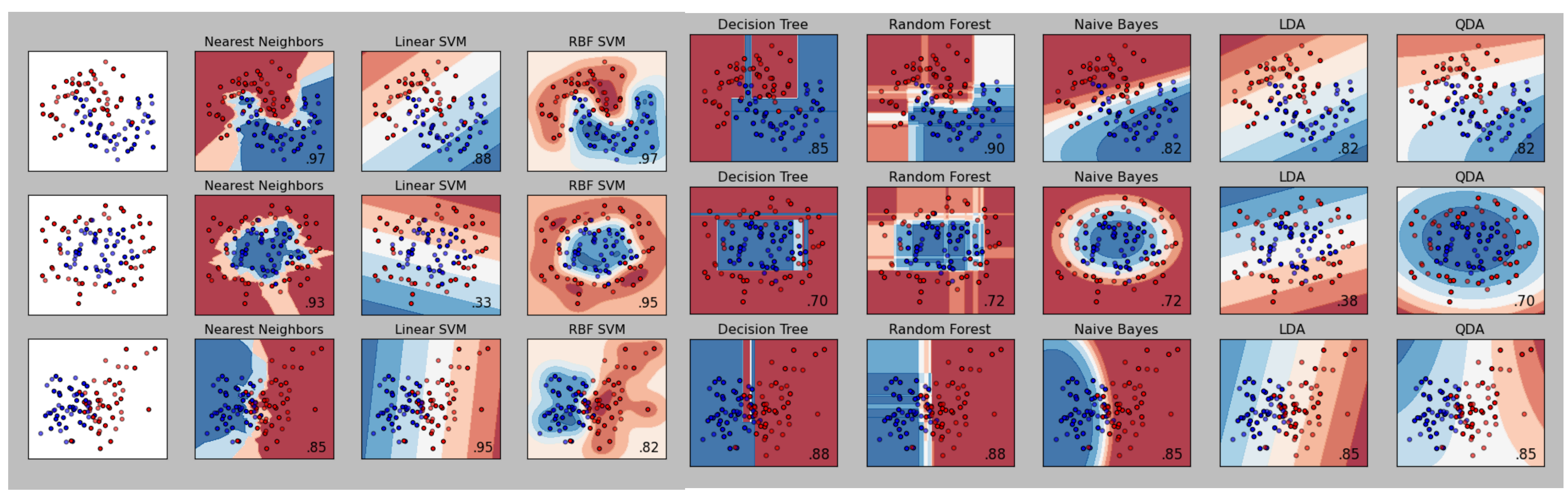
k-means algorithm



https://developers.google.com/machine-learning/clustering/clustering-algorithms

Machine Learning (ML) Overview: Algorithms





DataCamp SciKit Learn Cheetsheet:
https://www.datacamp.com/community/blog/scikit-learn-cheat-sheet

Machine Learning (ML) Overview: Algorithm components

=> (1) **Model** representation, (2) Evaluation and (3) Optimization

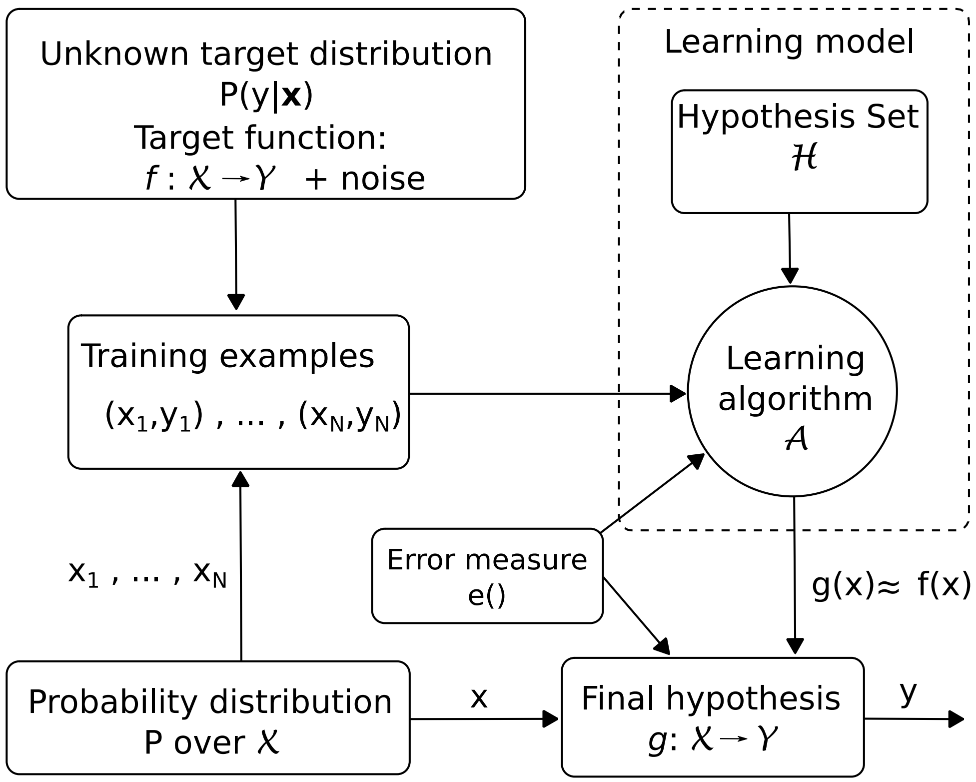


Table 1: The three components of learning algorithms.

Representation	Evaluation	Optimization
Instances	Accuracy/Error rate	Combinatorial optimization
K-nearest neighbor	Precision and recall	Greedy search
Support vector machines	Squared error	Beam search
Hyperplanes	Likelihood	Branch-and-bound
Naive Bayes	Posterior probability	Continuous optimization
Logistic regression	Information gain	Unconstrained
Decision trees	K-L divergence	Gradient descent
Sets of rules	Cost/Utility	Conjugate gradient
Propositional rules	Margin	Quasi-Newton methods
Logic programs		Constrained
Neural networks		Linear programming
Graphical models		Quadratic programming
Bayesian networks		
Conditional random fields		

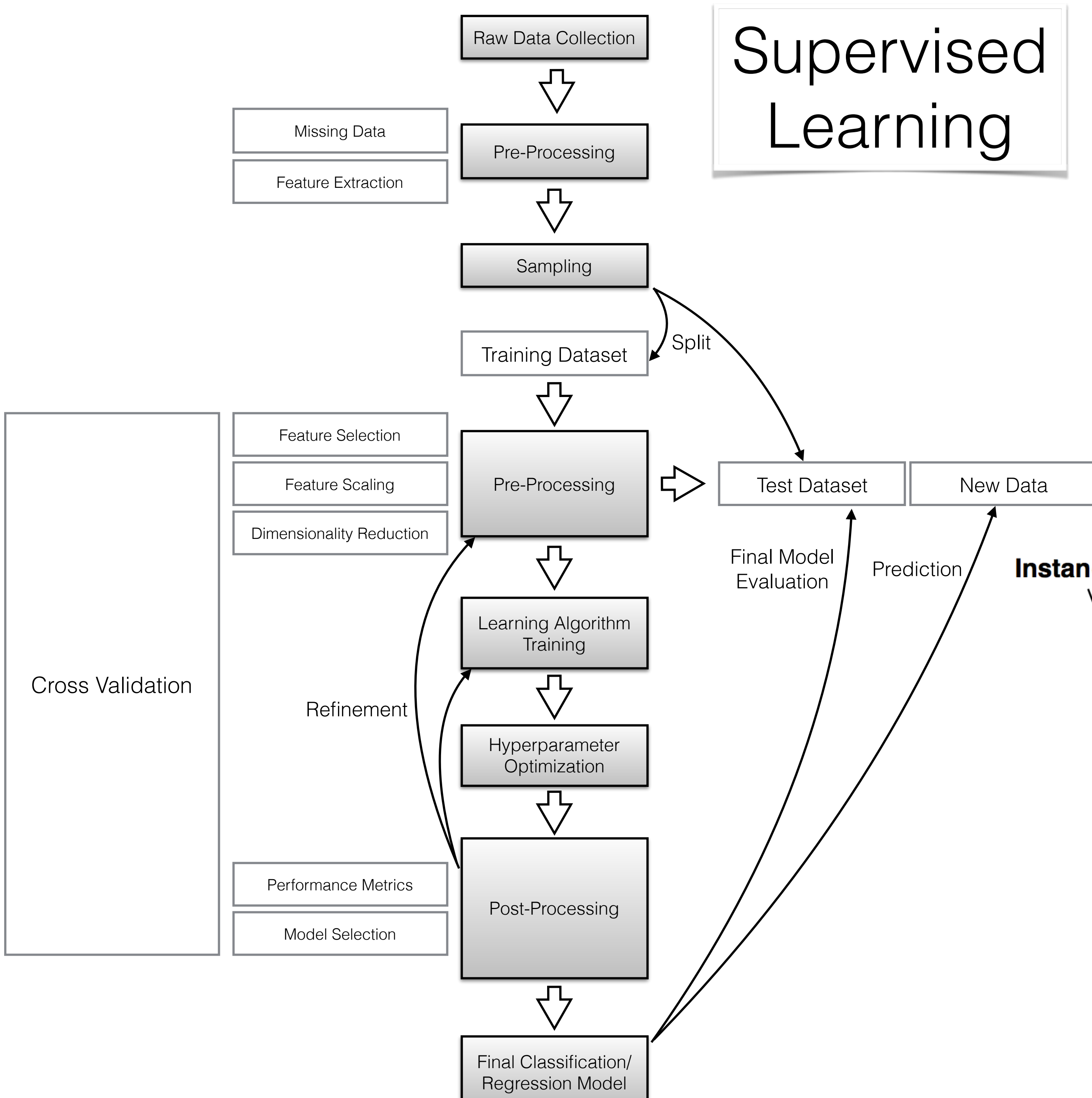
- Representation: algorithm, implementation
- Evaluation: metric selection, results based on real data
- Optimization: from off-the-shelf optimizers to custom designed ones

Recommended readings:

- **Domingos, Pedro M**. "*A few useful things to know about machine learning.*" Commun. acm 55.10 (2012): 78-87.
  - https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf
- Bennett, Kristin P., and Emilio Parrado-Hernández. "*The interplay of optimization and machine learning research*." Journal of Machine Learning Research 7.Jul (2006): 1265-1281
  - http://www.jmlr.org/papers/volume7/MLOPT-intro06a/MLOPT-intro06a.pdf

Machine Learning (ML) Overview: Input for algorithm

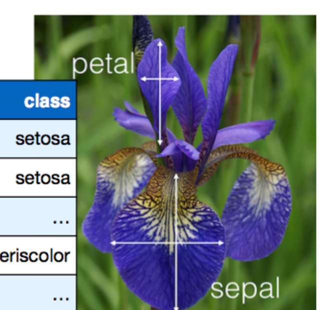
=> an example of usual input for common supervised learning setting



https://archive.ics.uci.edu/ml/datasets/Iris

Instances (samples, observations)

	sepal_length	sepal_width	petal_length	petal_width	class
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
...	...	...	...	...	...
50	6.4	3.2	4.5	1.5	veriscolor
...	...	...	...	...	...
150	5.9	3.0	5.1	1.8	virginica



Features (attributes, dimensions)

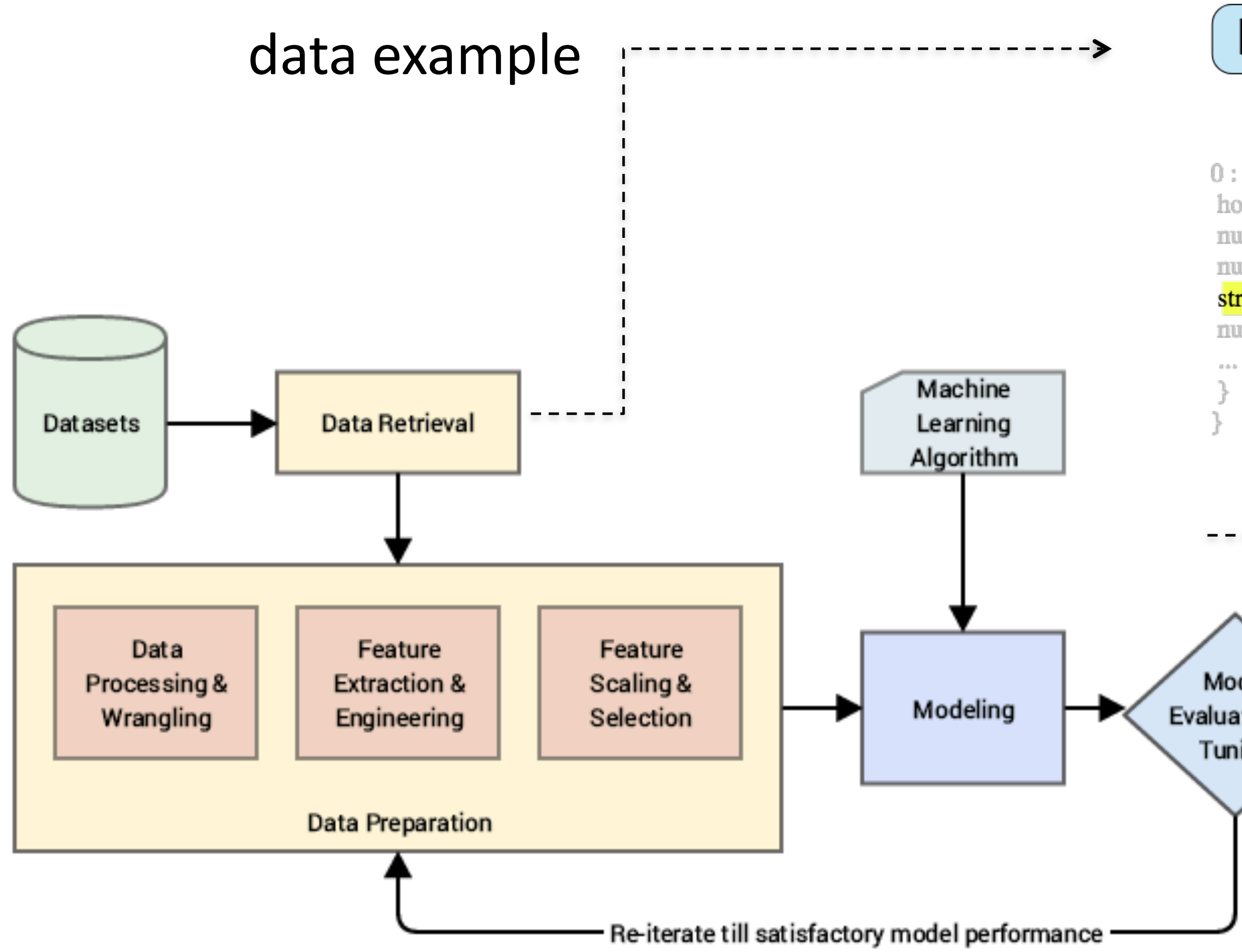
Classes (targets)

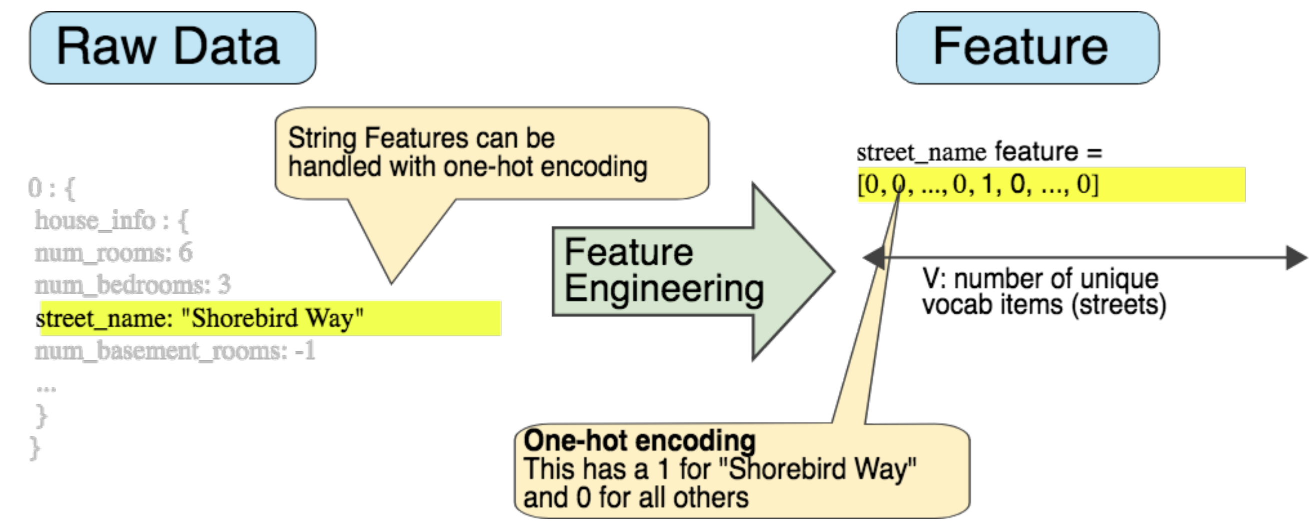
$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ x_{31} & x_{32} & \cdots & x_{3D} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix}$$

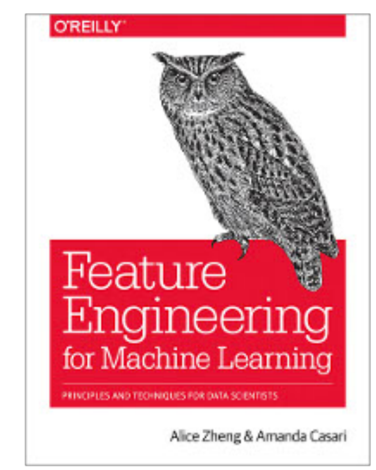
$$\mathbf{y} = [y_1, y_2, y_3, \cdots y_N]$$

Key factor in ‘Traditional ML’ ≈ Feature engineering

=> correct use of inputs is key for a successful ML application





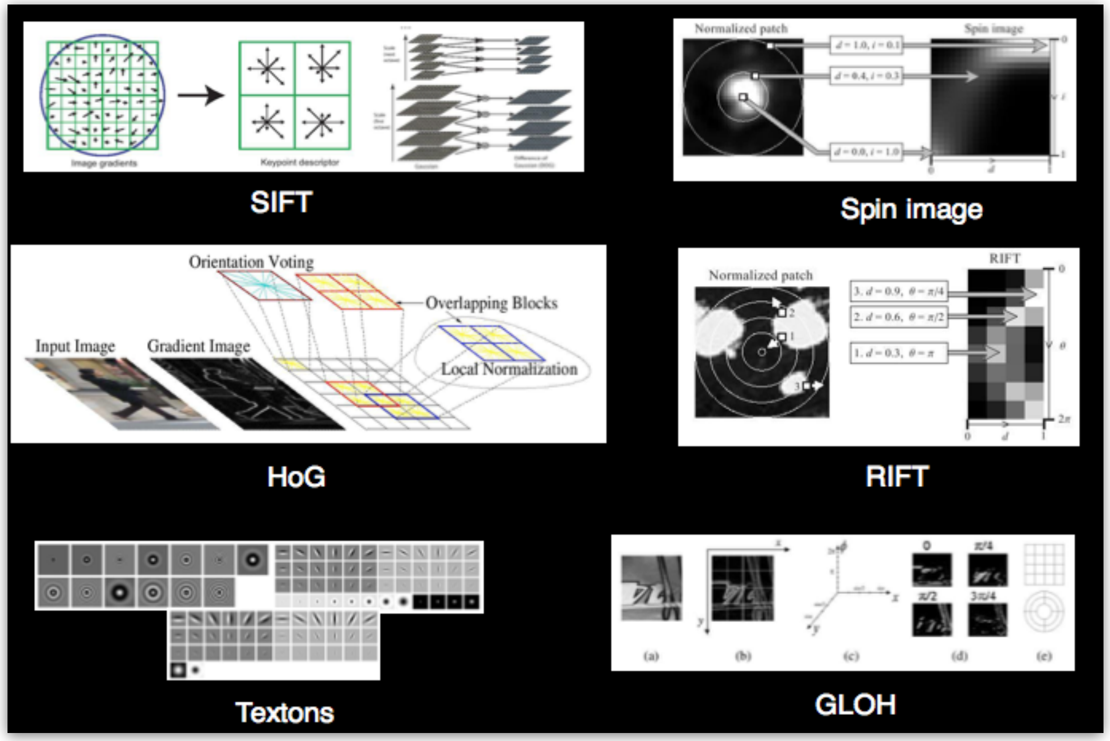


Prof. Pedro Domingos from the University of Washington, in his paper titled, “*A Few Useful Things to Know about Machine Learning*” tells us the following. “At the end of the day, some machine learning projects succeed and some fail. What makes the difference? **Easily the most important factor is the features used**.”

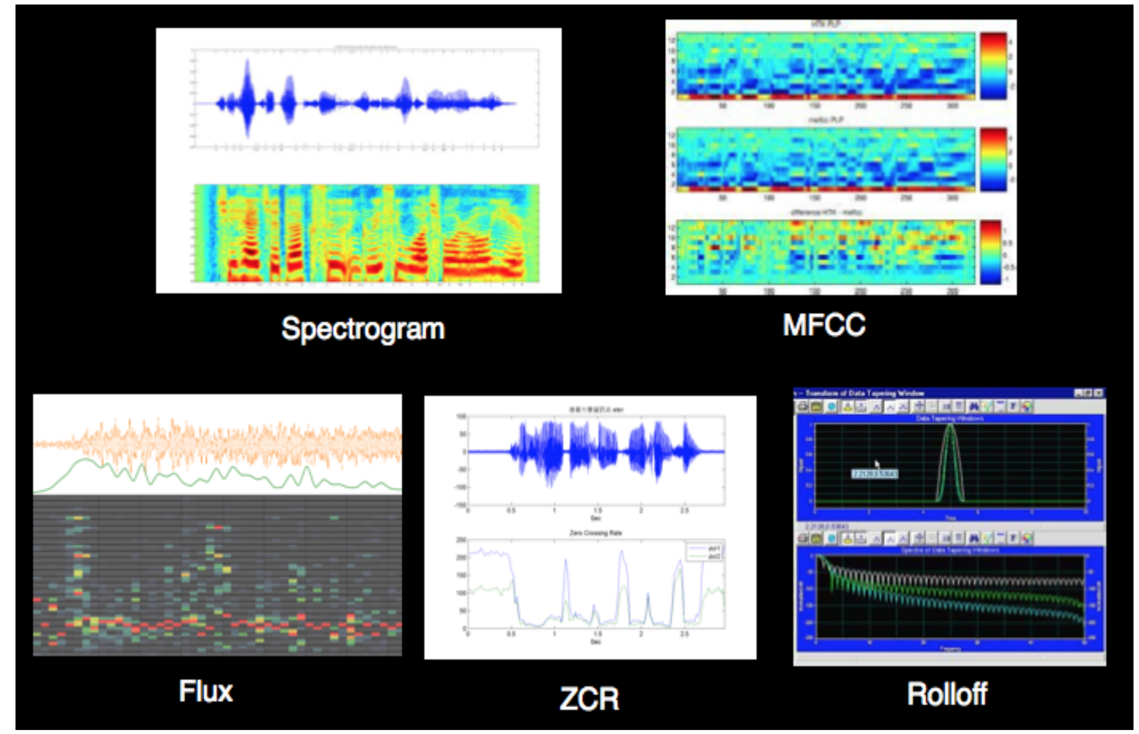
Machine Learning (ML) Overview: Input for algorithm

=> an example of usual input for common supervised learning setting

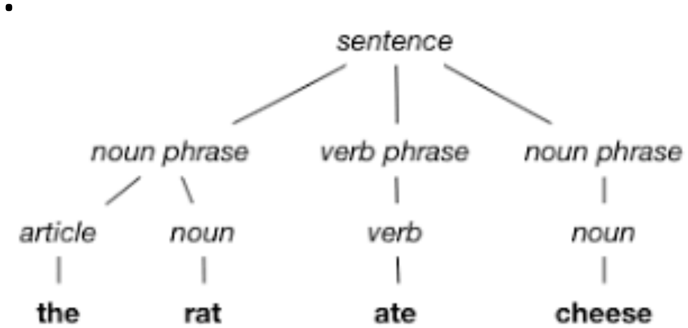
Image processing:



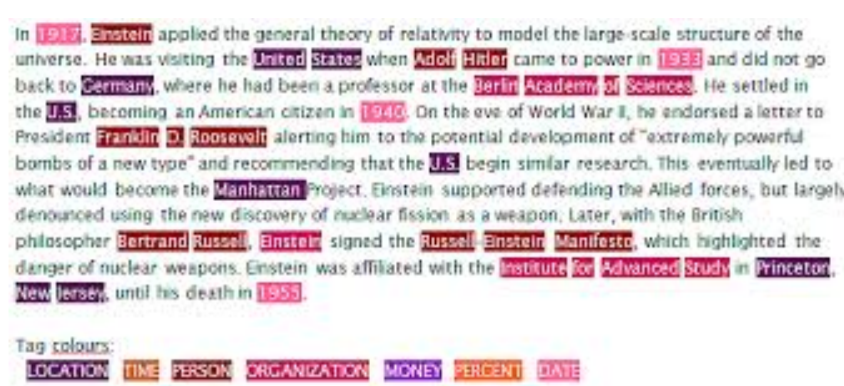
Audio signal processing:



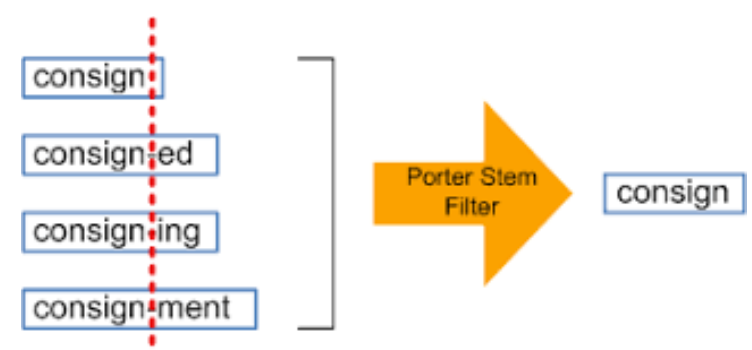
NLP:



PoS tagging



NER

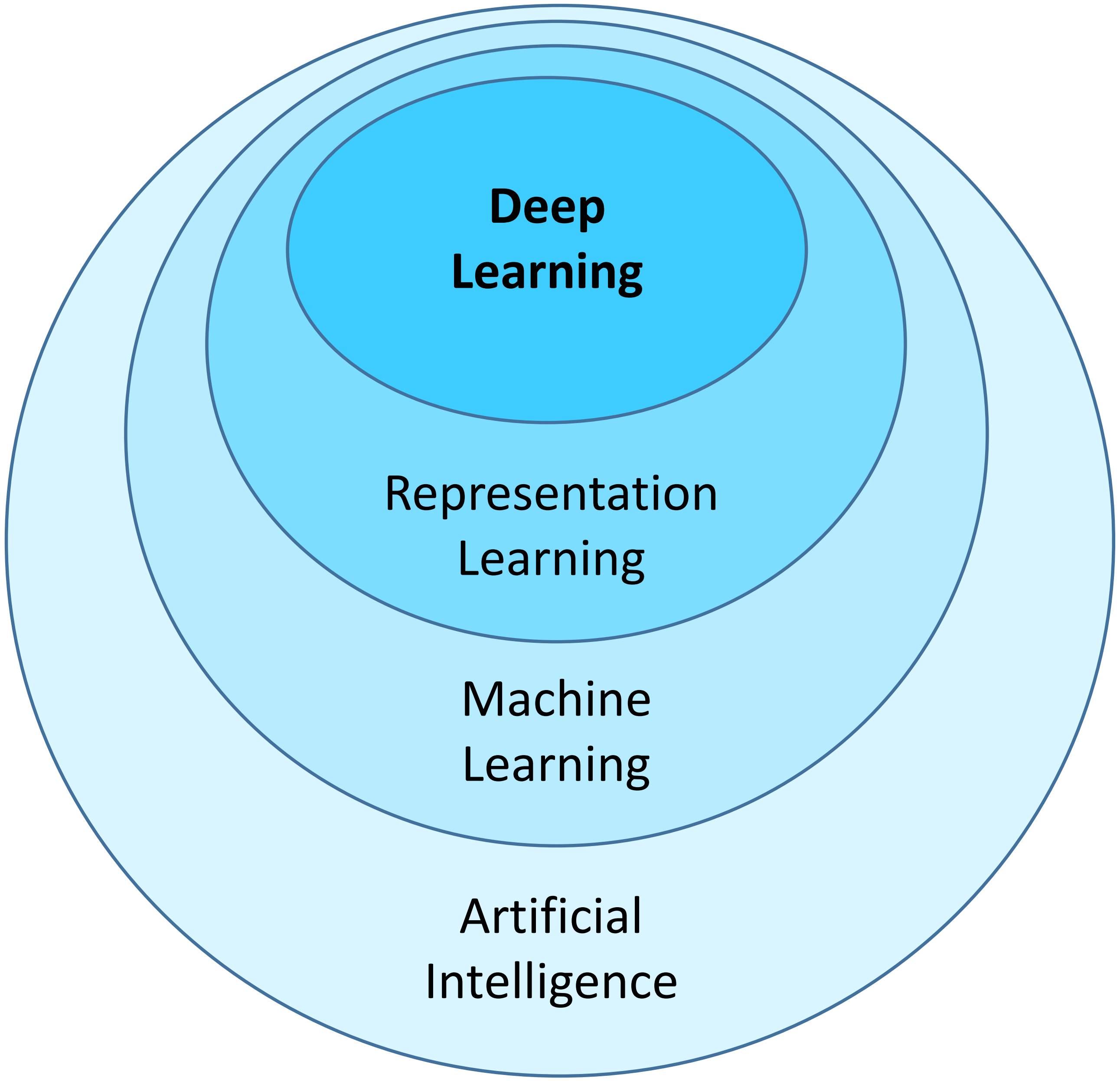


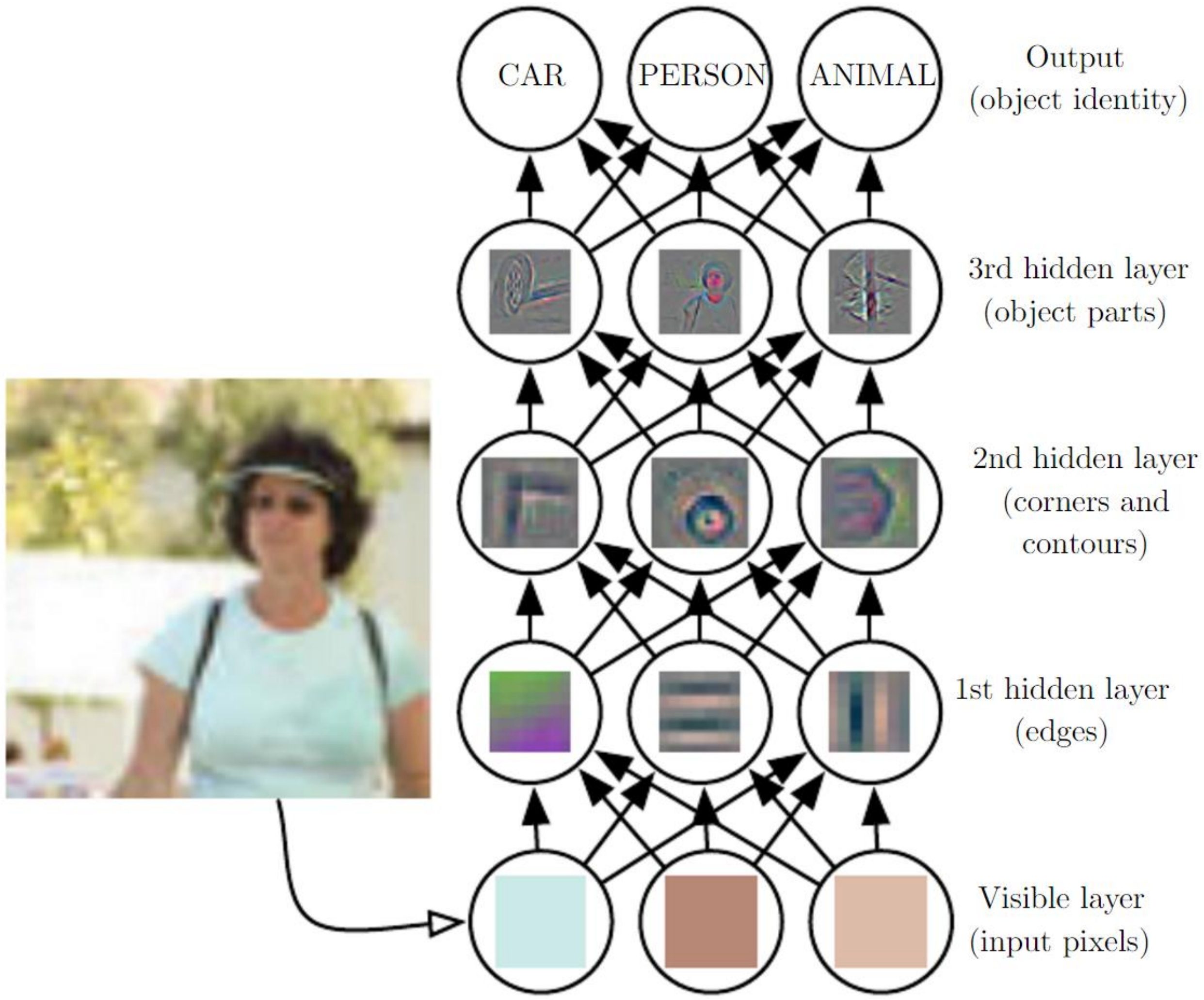
Stammer

Deep learning: Basics

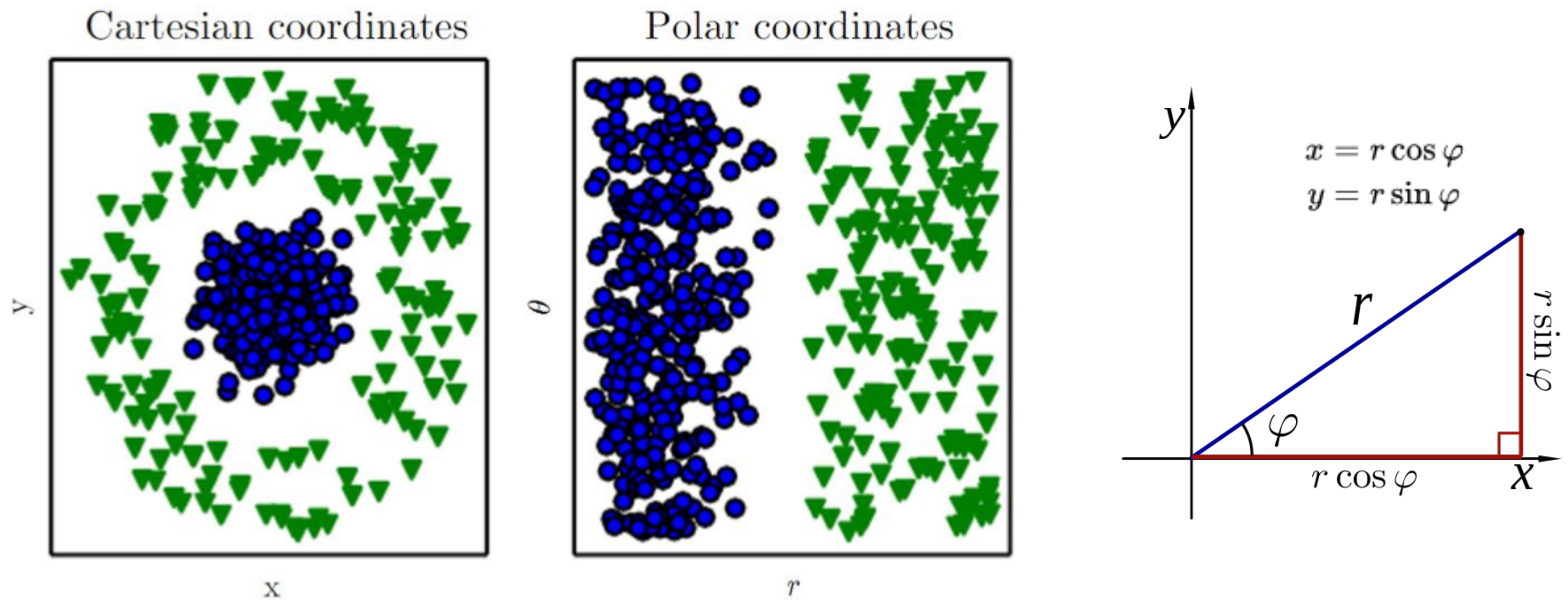
Deep learning is **Representation Learning (RL)**

- learning a hierarchy of features directly from the data instead of hand engineering





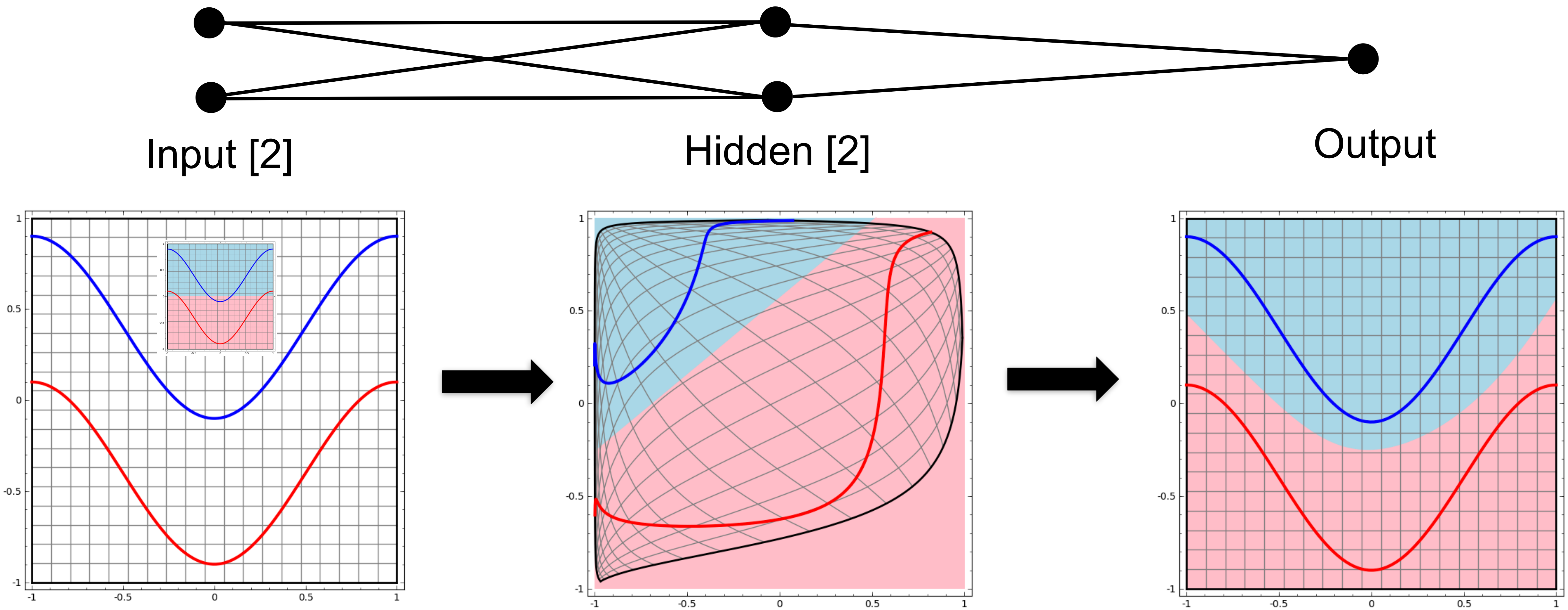
Example: Representation Matters



Task: draw a line to separate the green triangles and blue circles

Example: DL = **RL** (aka Feature Learning)

- hidden layer in NN learns a representation so that the data is linearly separable

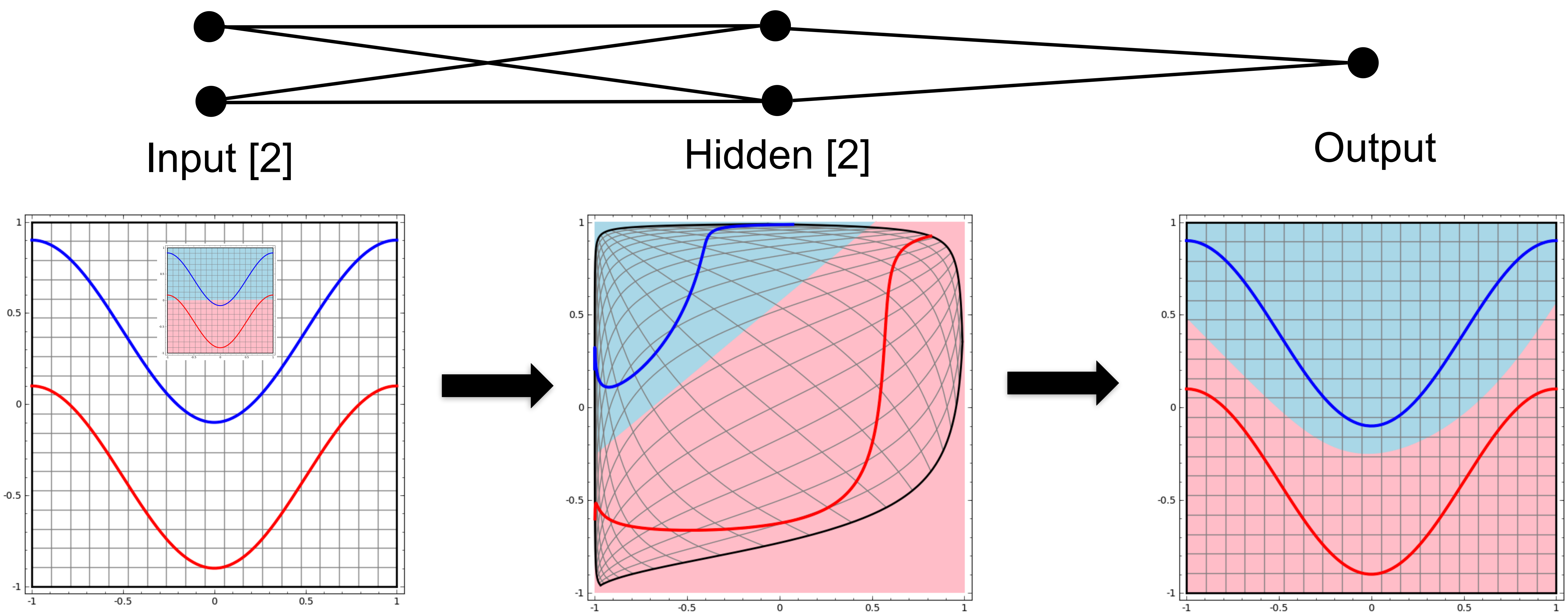


Task: draw a line to separate the blue curve and red curve

Source: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Example: DL = **RL** (aka Feature Learning)

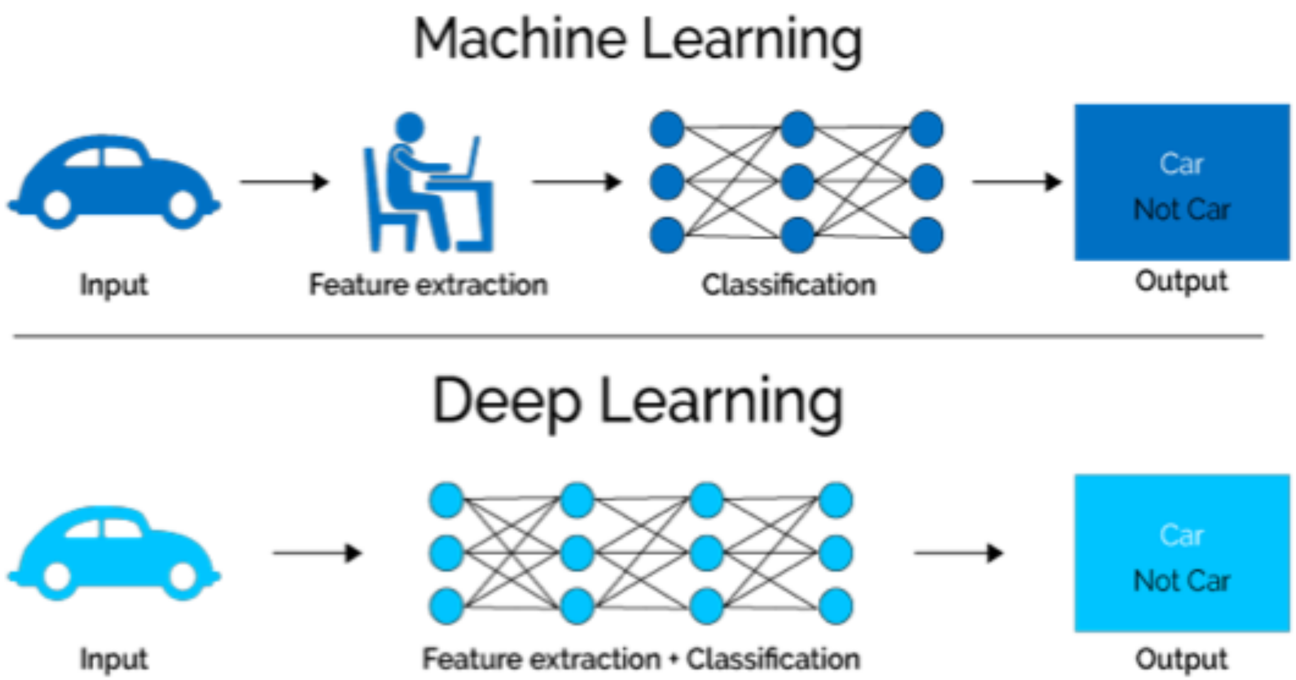
- hidden layer in NN learns a representation so that the data is linearly separable

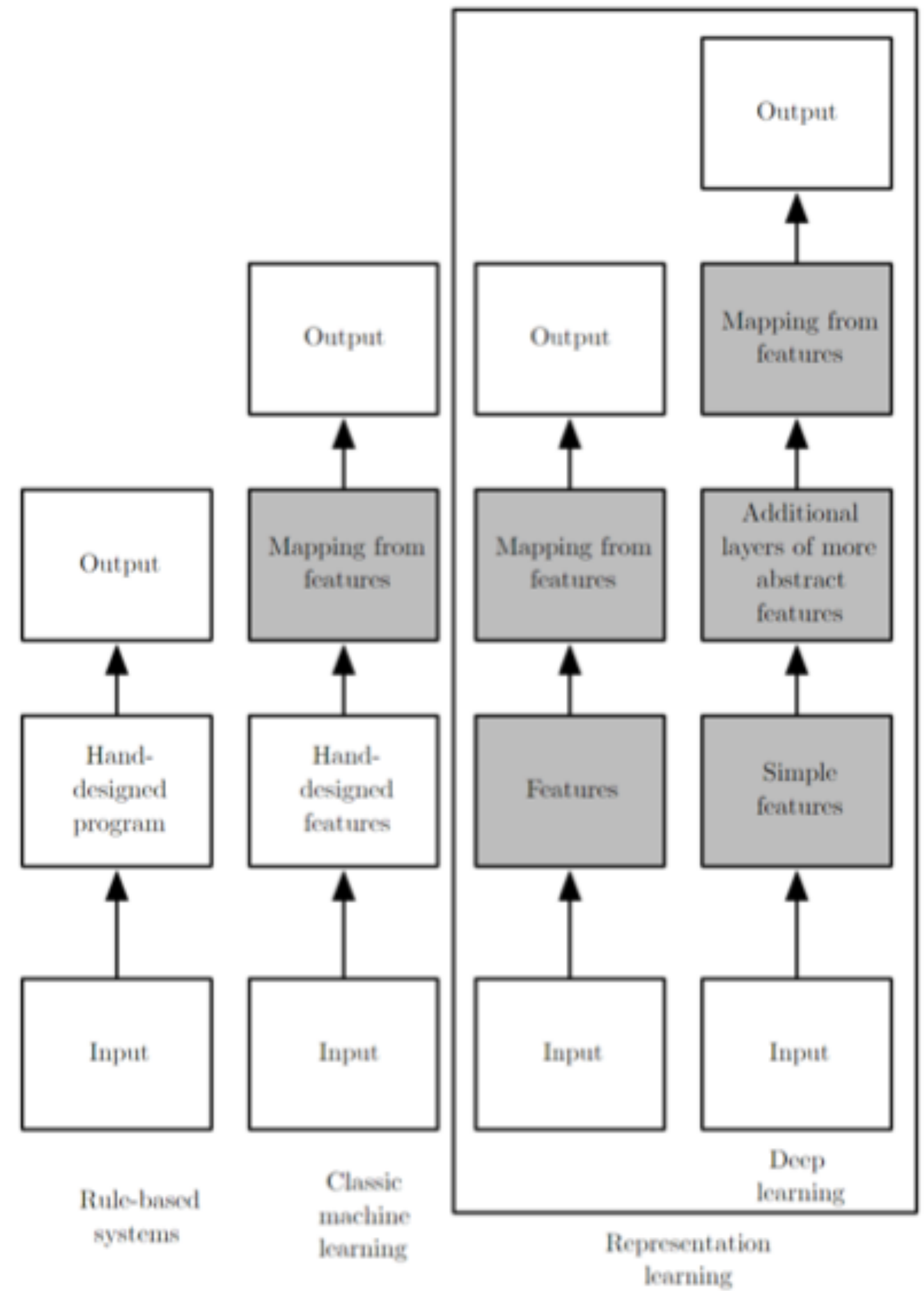


Task: draw a line to separate the blue curve and red curve

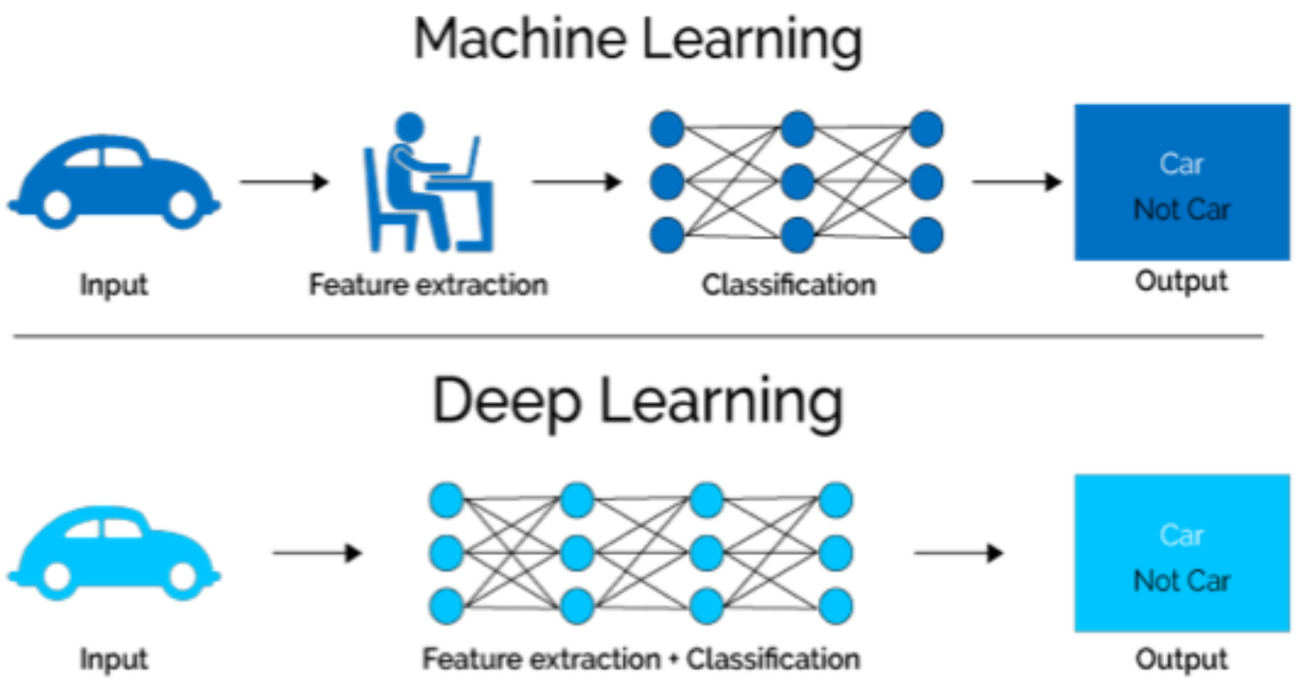
Source: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

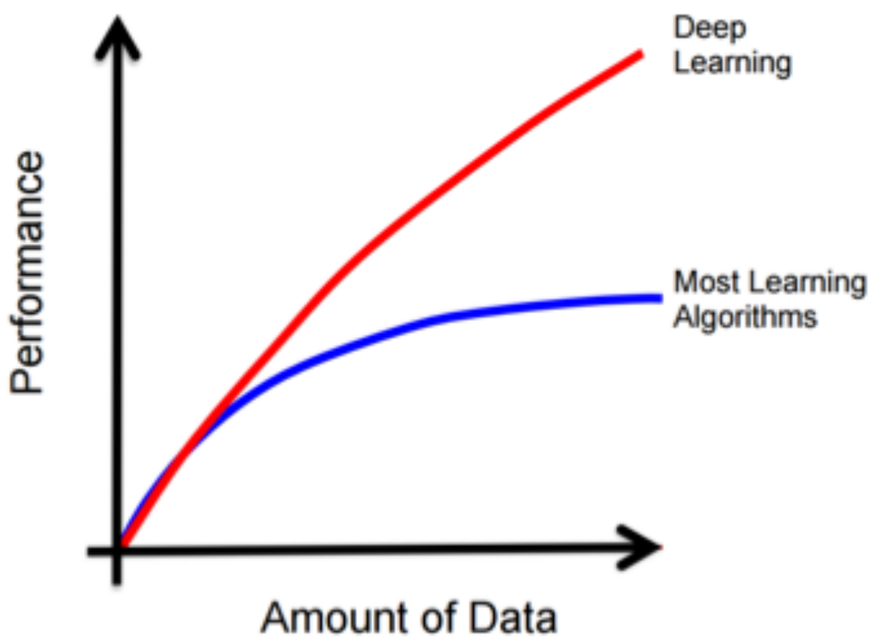
Q: Why DL?
A: Scalable ML learning (end to end learning)





Q: Why DL?
A: Scalable ML learning (end to end learning)





Why Now?

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

⋮

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

⋮

Neural Networks date back decades, so why the resurgence?

1. Big Data
- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

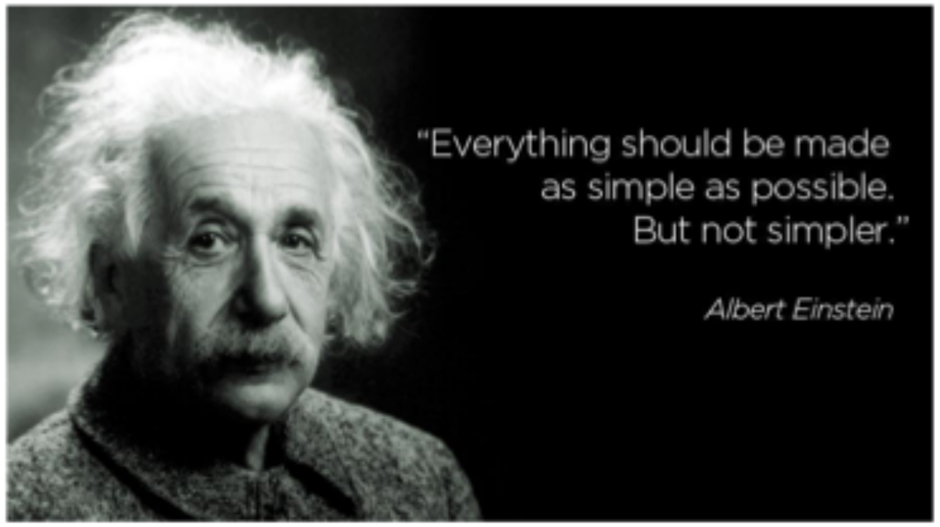
2. Hardware
- Graphics Processing Units (GPUs)
- Massively Parallelizable

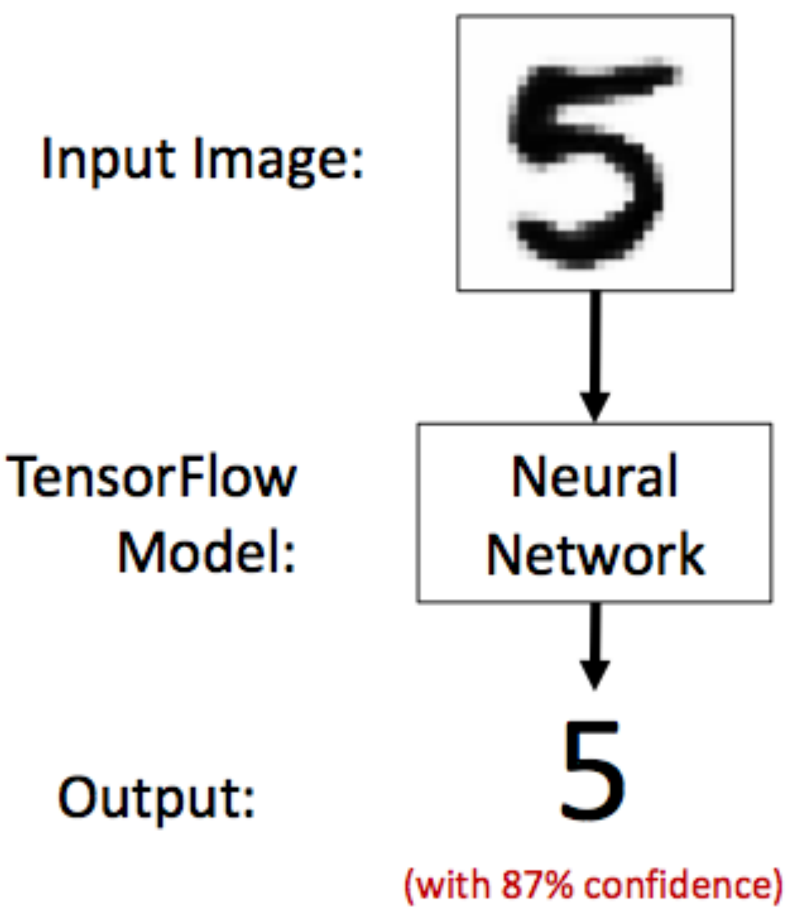
3. Software
- Improved Techniques
- New Models
- Toolboxes

TensorFlow

Source: https://introdeeplearning.com/

How to do DL: Simple example





1

```python
# import tensorflow and keras (tf.keras not "vanilla" Keras)
import tensorflow as tf
from tensorflow import keras
```

2

```python
# get data
(train_images, train_labels), (test_images, test_labels) = \
keras.datasets.mnist.load_data()
```

3

```python
# setup model
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

4

```python
# train model
model.fit(train_images, train_labels, epochs=5)
```

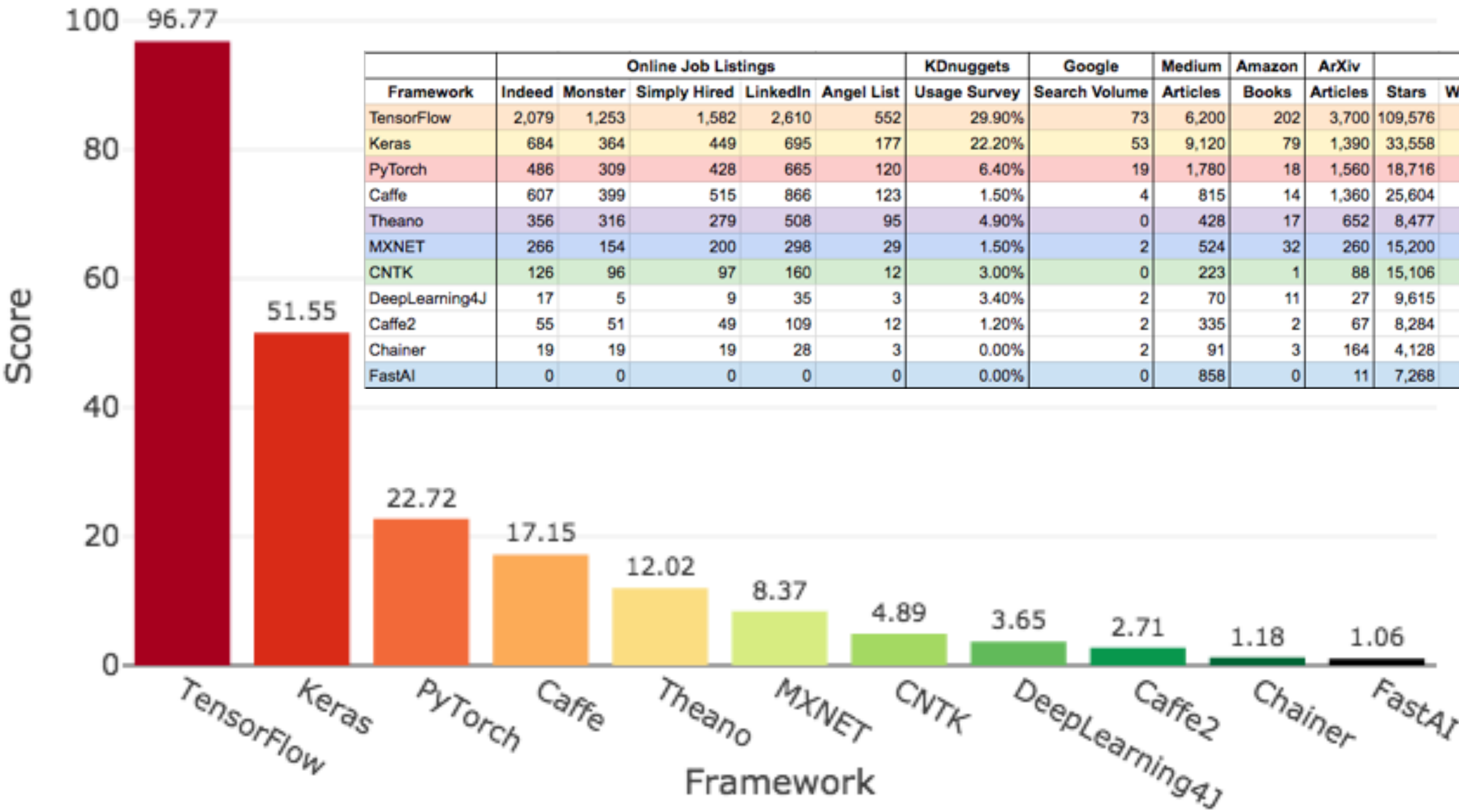
5

```python
# evaluate
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test accuracy:', test_acc)
```

6

```python
# make predictions
predictions = model.predict(test_images)
```

Deep Learning Framework Power Scores 2018



	Online Job Listings					KDnuggets	Google	Medium	Amazon	ArXiv	GitHub Activity			
Framework	Indeed	Monster	Simply Hired	LinkedIn	Angel List	Usage Survey	Search Volume	Articles	Books	Articles	Stars	Watchers	Forks	Contributors
TensorFlow	2,079	1,253	1,582	2,610	552	29.90%	73	6,200	202	3,700	109,576	8,334	67,551	1,642
Keras	684	364	449	695	177	22.20%	53	9,120	79	1,390	33,558	1,847	12,658	719
PyTorch	486	309	428	665	120	6.40%	19	1,780	18	1,560	18,716	952	4,474	760
Caffe	607	399	515	866	123	1.50%	4	815	14	1,360	25,604	2,218	15,633	270
Theano	356	316	279	508	95	4.90%	0	428	17	652	8,477	585	2,447	328
MXNET	266	154	200	298	29	1.50%	2	524	32	260	15,200	1,170	5,498	587
CNTK	126	96	97	160	12	3.00%	0	223	1	88	15,106	1,368	4,029	189
DeepLearning4J	17	5	9	35	3	3.40%	2	70	11	27	9,615	829	4,441	232
Caffe2	55	51	49	109	12	1.20%	2	335	2	67	8,284	577	2,102	193
Chainer	19	19	19	28	3	0.00%	2	91	3	164	4,128	325	1,095	182
FastAI	0	0	0	0	0	0.00%	0	858	0	11	7,268	432	2,647	195

https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Deep Learning in **One** Slide

- **What is it:**
  Extract useful patterns from data.
- **How:**
  Neural network + optimization
- **How (Practical):**
  Python + TensorFlow & friends
- **Hard Part:**
  Good Questions + Good Data
- **Why now:**
  Data, hardware, community, tools, investment
- **Where do we stand?**
  Most big questions of intelligence have not been answered nor properly formulated

Exciting progress:

- Face recognition
- Image classification
- Speech recognition
- Text-to-speech generation
- Handwriting transcription
- Machine translation
- Medical diagnosis
- Cars: drivable area, lane keeping
- Digital assistants
- Ads, search, social recommendations
- Game playing with deep RL

Source: https://deeplearning.mit.edu/

TensorFlow in One Slide

- **What is it:** Deep Learning Library *(and more)*
  - **Facts:** Open Source, Python, Google
- **Community:**
  - 117,000+ GitHub stars
  - TensorFlow.org: Blogs, Documentation, DevSummit, YouTube talks
- **Ecosystem:**
  - Keras: high-level API
  - TensorFlow.js: in the browser
  - TensorFlow Lite: on the phone
  - Colaboratory: in the cloud
  - TPU: optimized hardware
  - TensorBoard: visualization
  - TensorFlow Hub: graph modules
- **Alternatives**: PyTorch, MXNet, CNTK

Extras:

- Swift for TensorFlow
- TensorFlow Serving
- TensorFlow Extended (TFX)
- TensorFlow Probability
- Tensor2Tensor

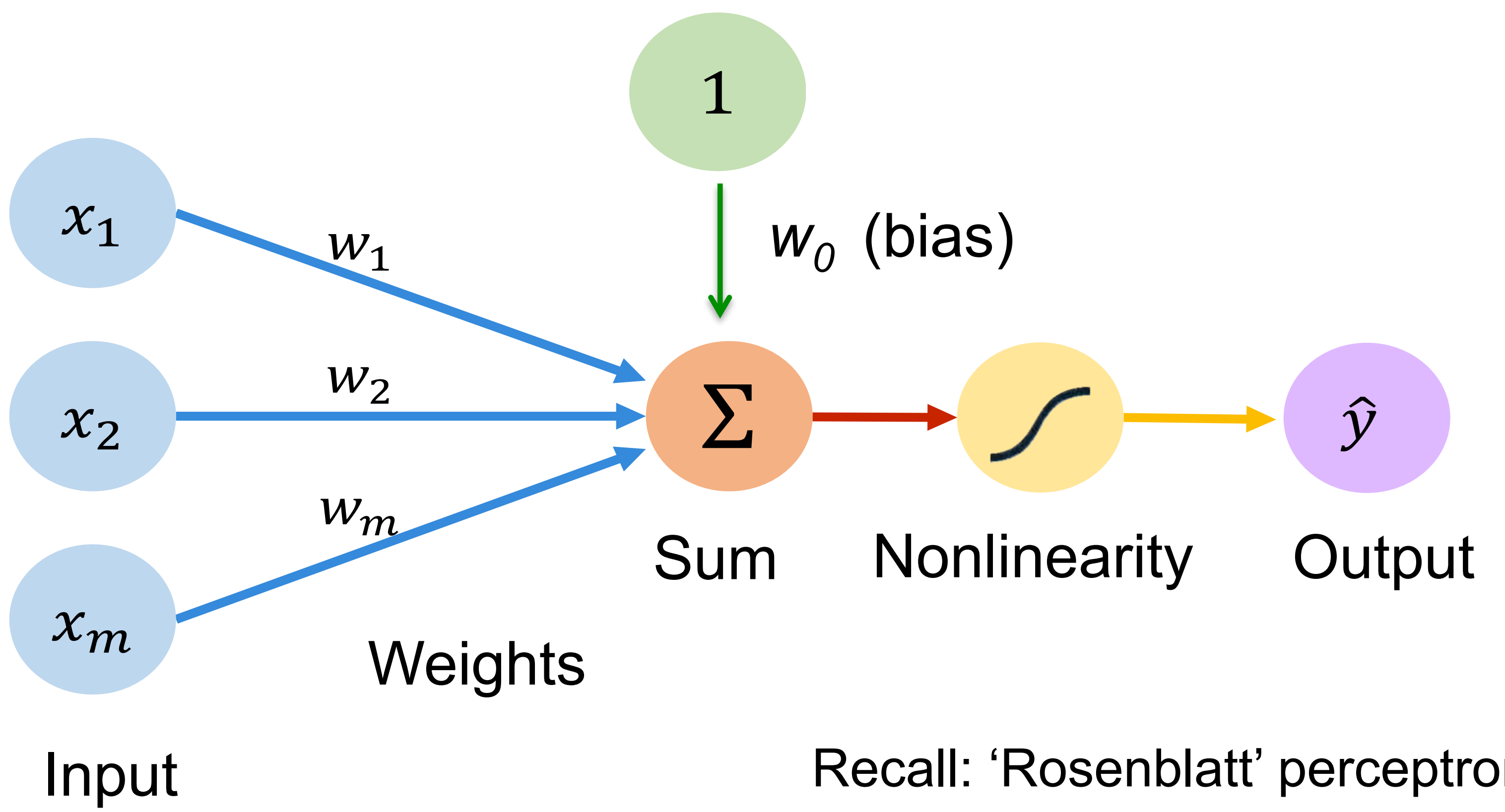
Recommended course and materials:

- CS 20: Tensorflow for Deep Learning Research:
  - ◇ http://web.stanford.edu/class/cs20si/
- Tensorflow Tutorials & Guides:
  - ◇ https://www.tensorflow.org/tutorials
  - ◇ https://www.tensorflow.org/guide

Perceptron:
Structural Building Block of DL

Source: MIT 6.S191: http://introtodeeplearning.com

Perceptron (Neuron): Forward propagation



Output

Linear combination of inputs

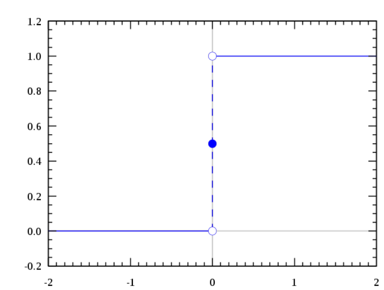
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \; w_i \right)$$

Non-linear activation function

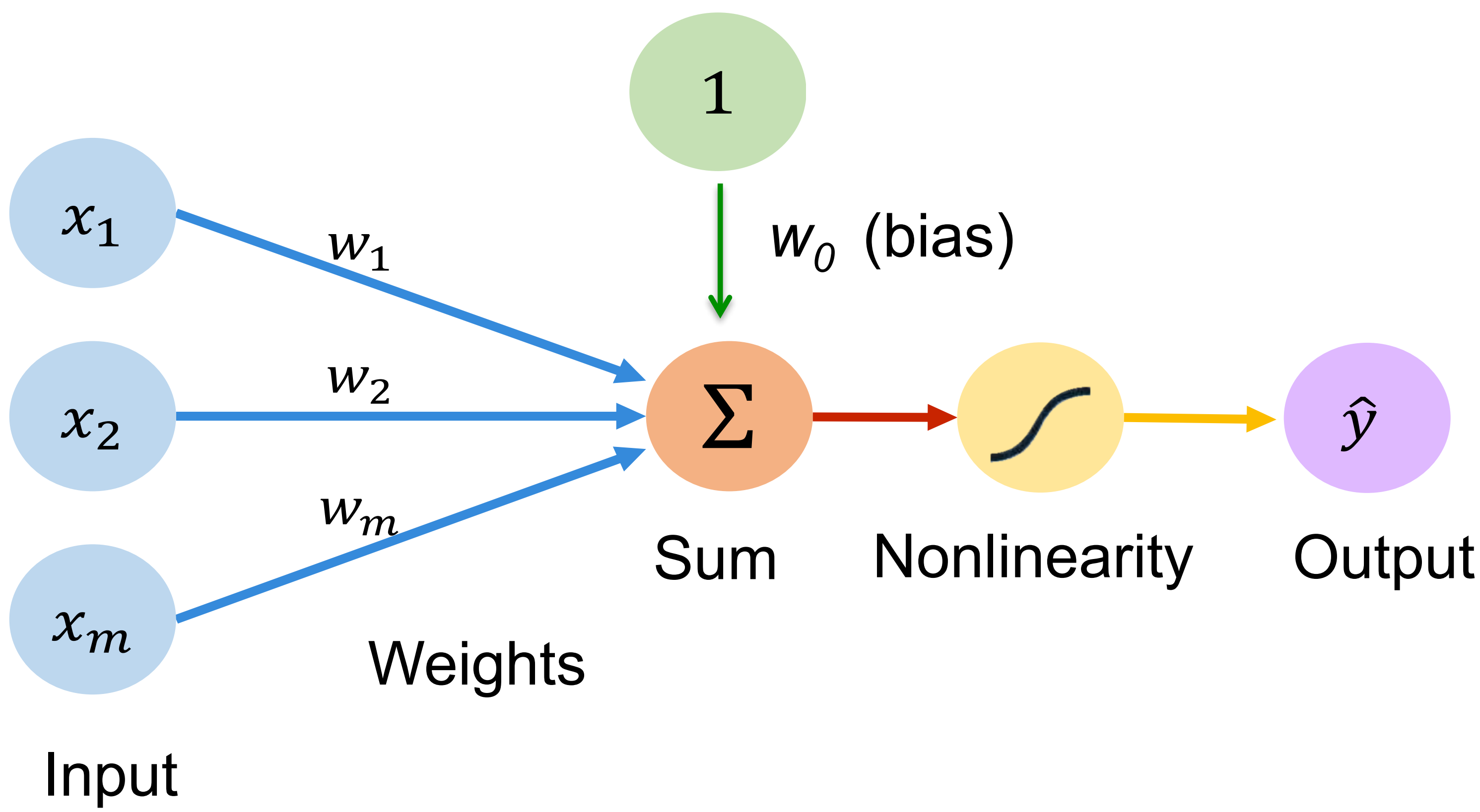
Bias

Recall: 'Rosenblatt' perceptron

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$



Neuron: Forward propagation (vectorized notation)



Output

Linear combination of inputs

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

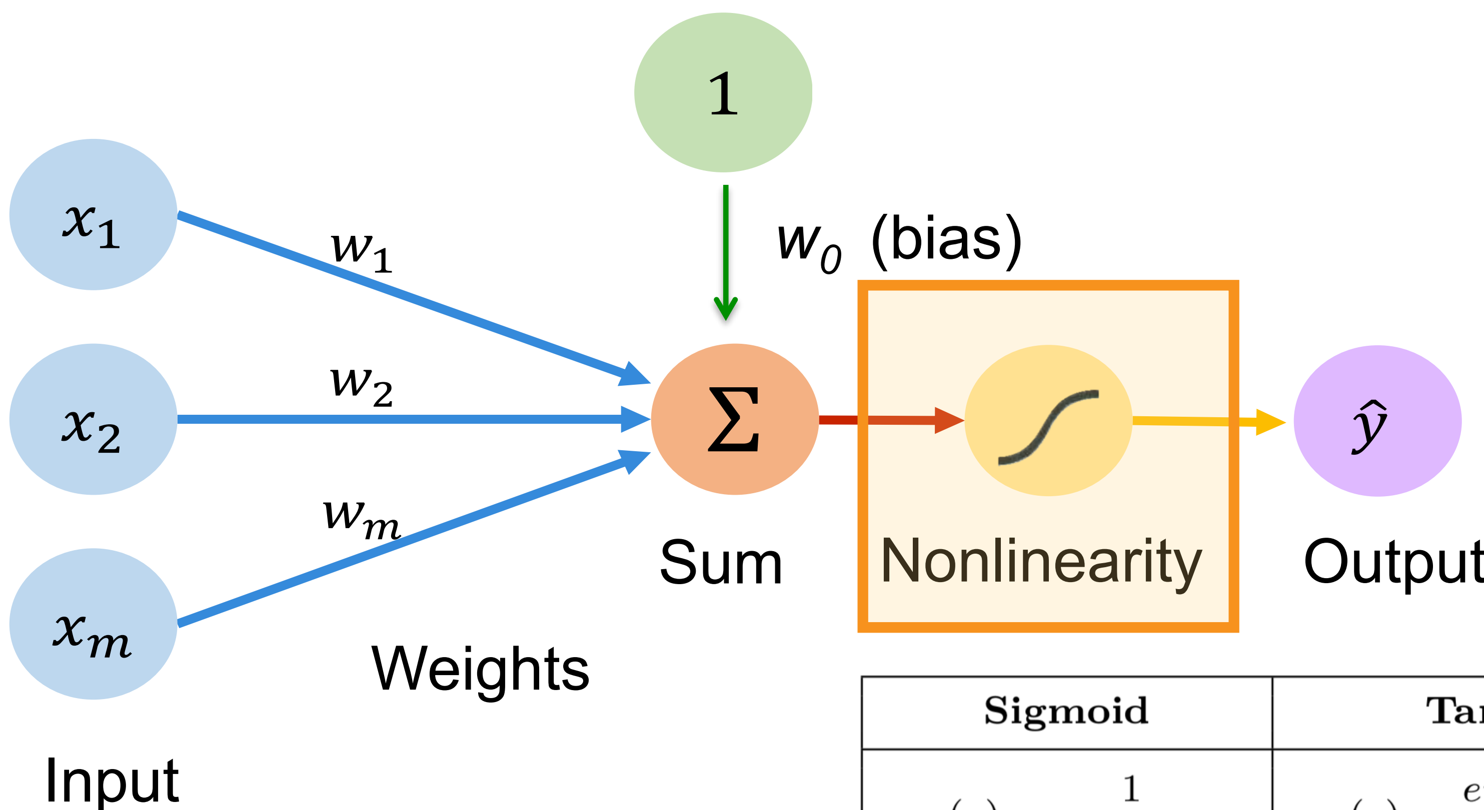
Non-linear activation function

Bias

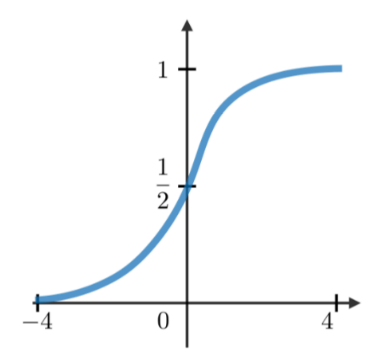
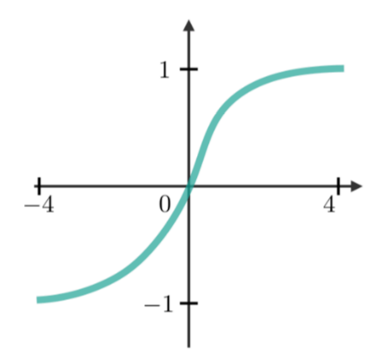
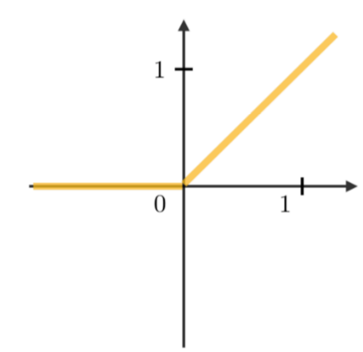
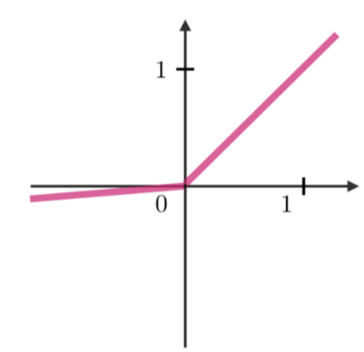
$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

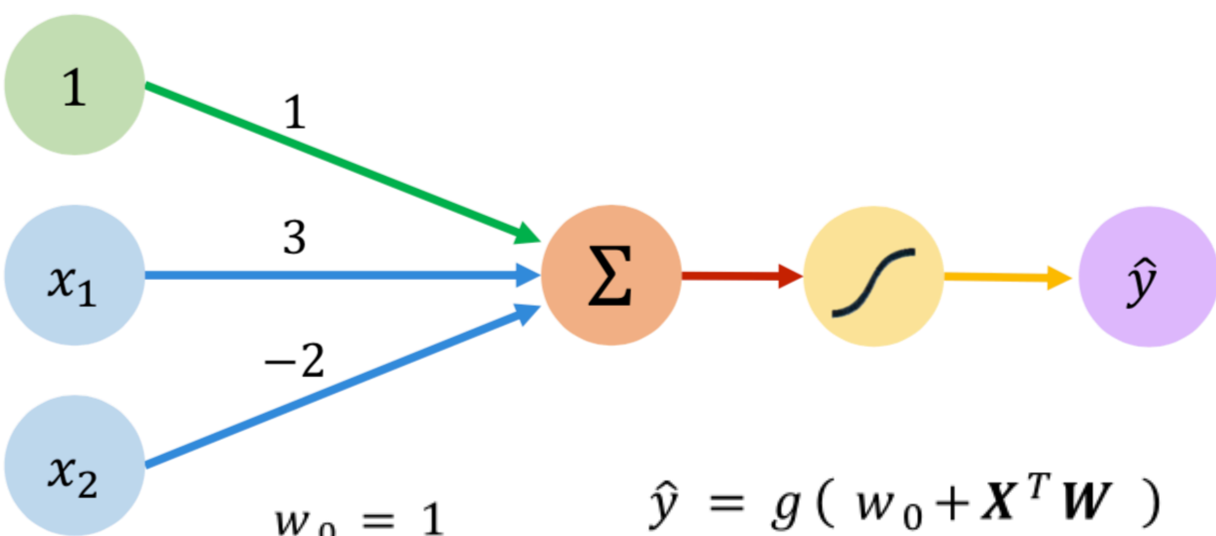
Neuron: Forward propagation (activation function)



$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

Sigmoid	**Tanh**	**ReLU**	**Leaky ReLU**
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0,z)$	$g(z) = \max(\epsilon z,z)$ with $\epsilon \ll 1$
			

Neuron: forward propagation example



$$w_0 = 1$$

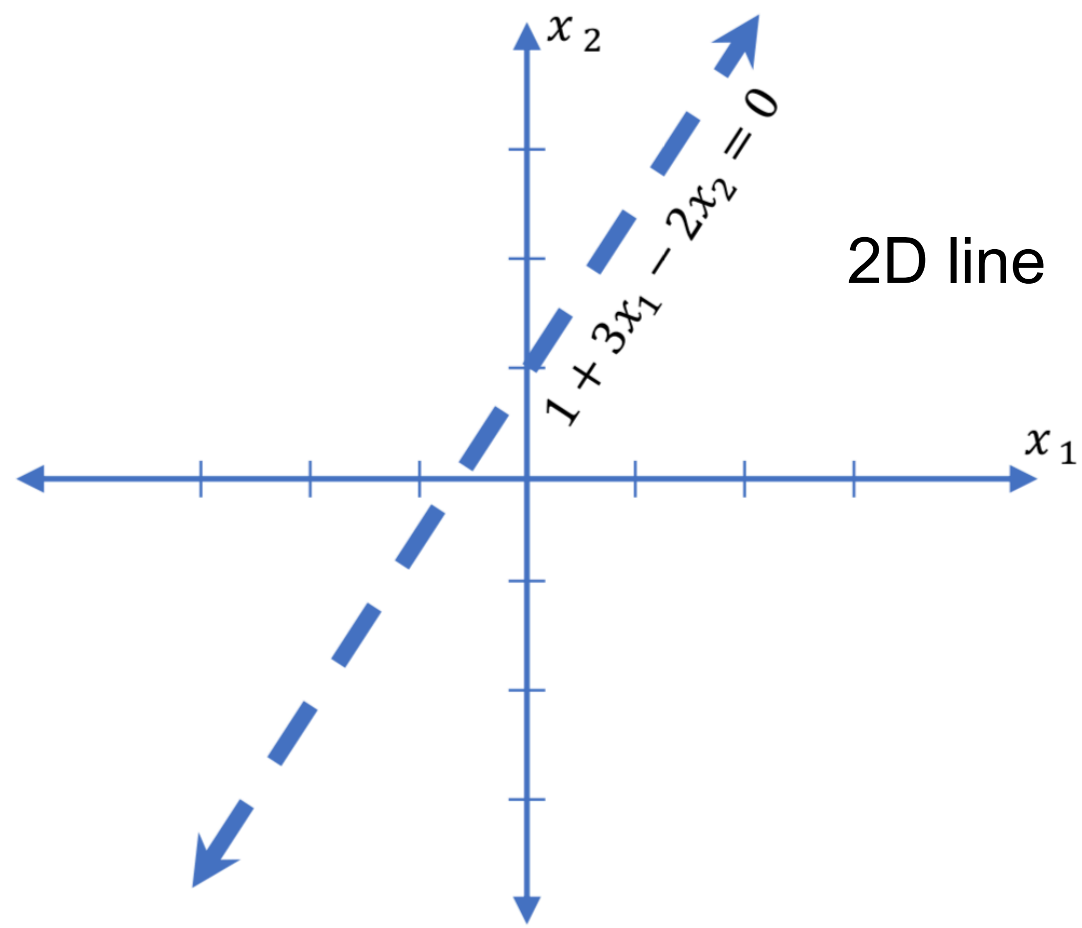
$$\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$$

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

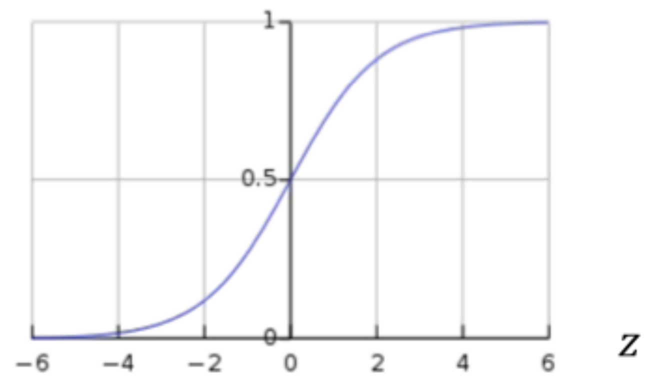
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$

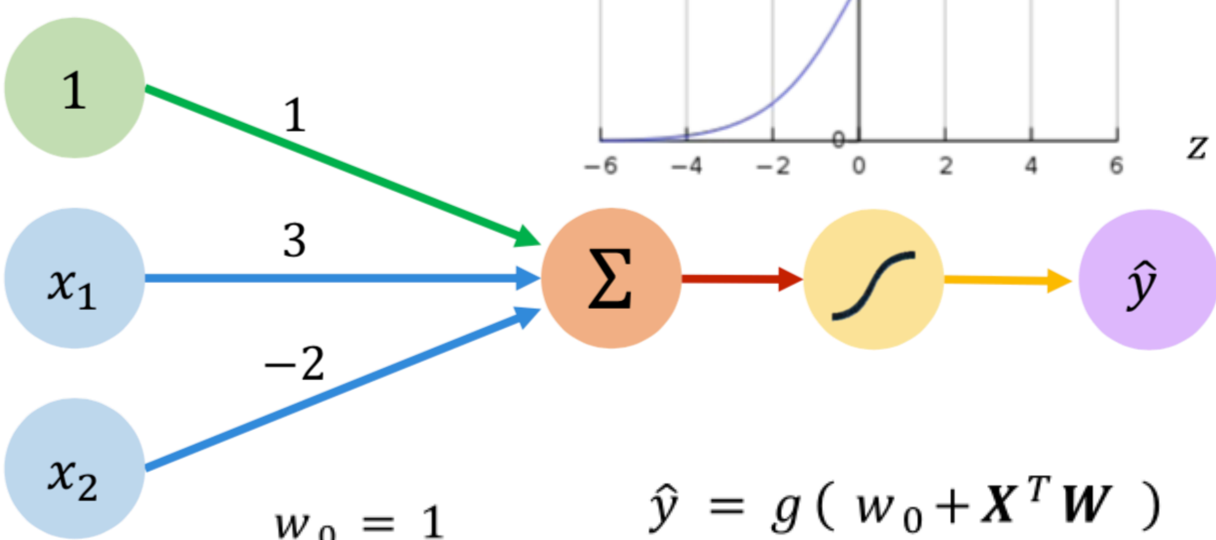
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



Neuron: forward propagation example

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$





$$w_0 = 1$$

$$\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$$

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

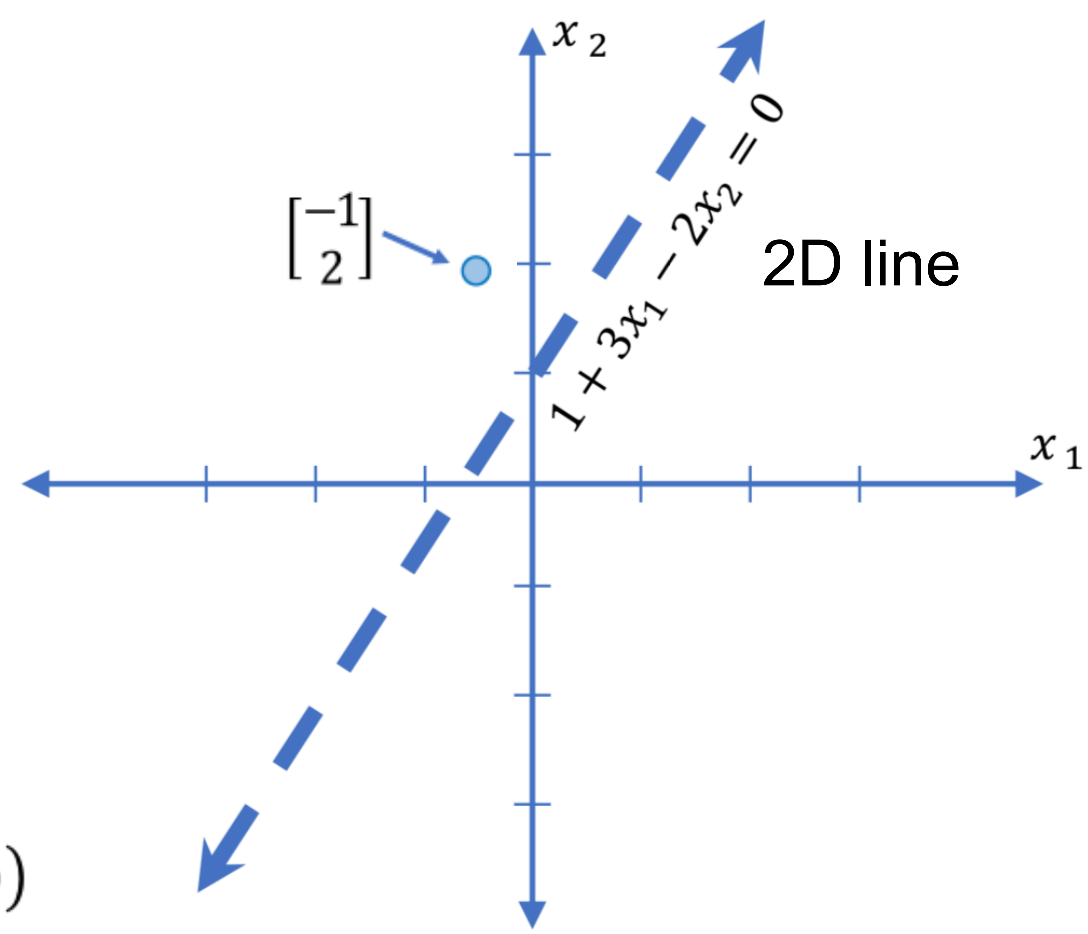
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

$$\hat{y} = g(1 + 3x_1 - 2x_2)$$

$$\hat{y} = g\big(1 + (3 * -1) - (2 * 2)\big)$$

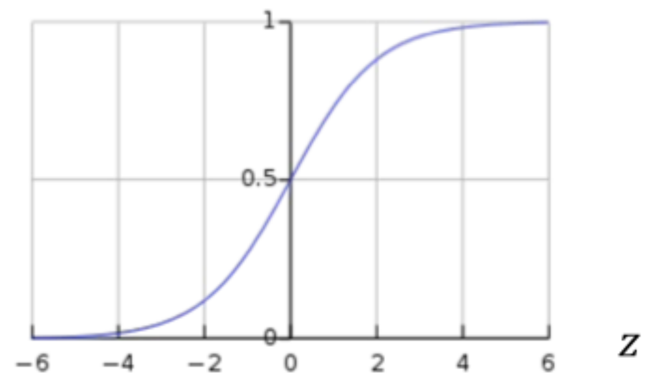
$$= g(-6) \approx 0.002$$

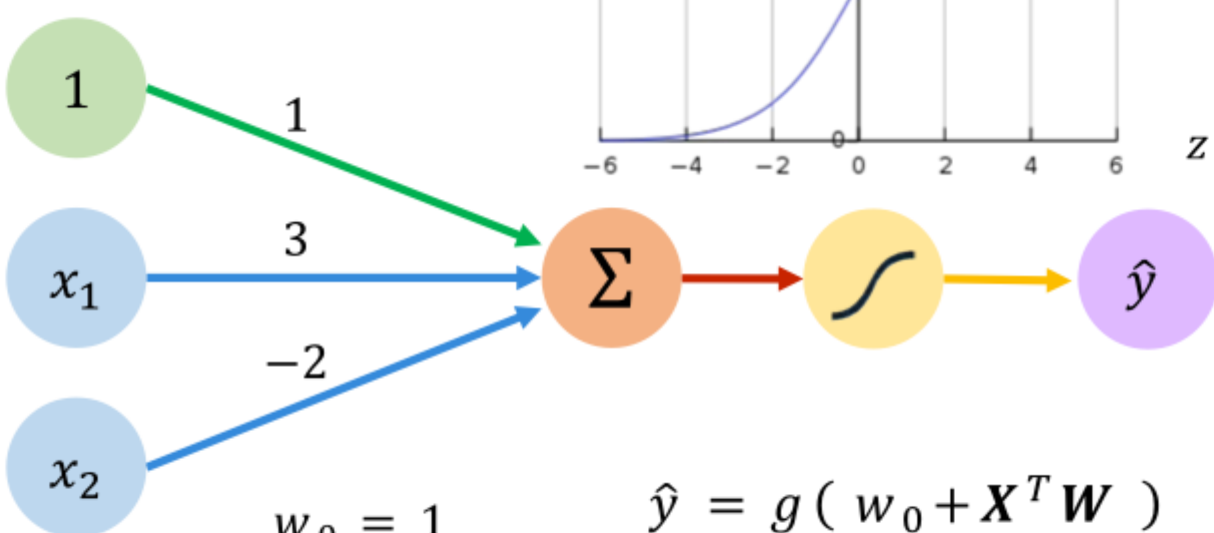
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



Neuron example (logistic regression)

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$





$$w_0 = 1$$

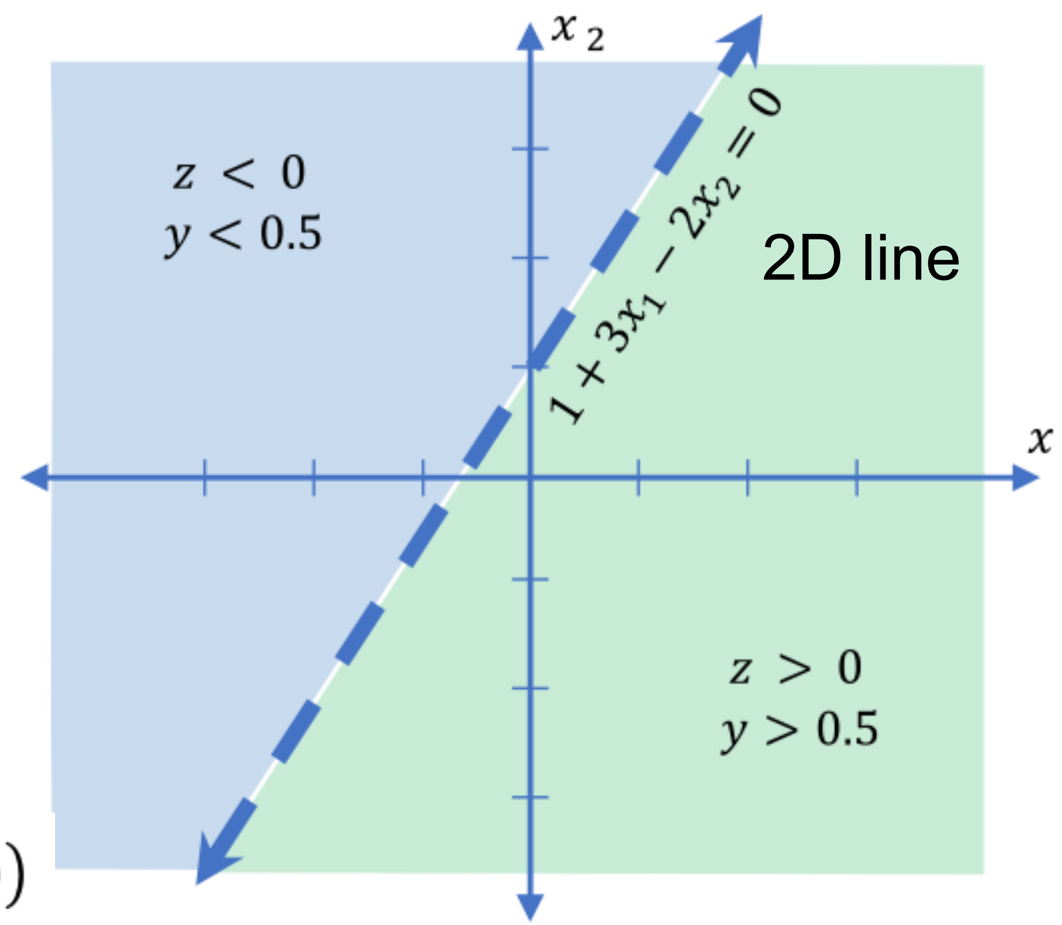
$$\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$$

$$\begin{aligned} \hat{y} &= g(w_0 + \boldsymbol{X}^T \boldsymbol{W}) \\ &= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \end{aligned}$$

$$\hat{y} = g(1 + 3x_1 - 2x_2)$$

$$\begin{aligned} \hat{y} &= g\big(1 + (3 * -1) - (2 * 2)\big) \\ &= g(-6) \approx 0.002 \end{aligned}$$

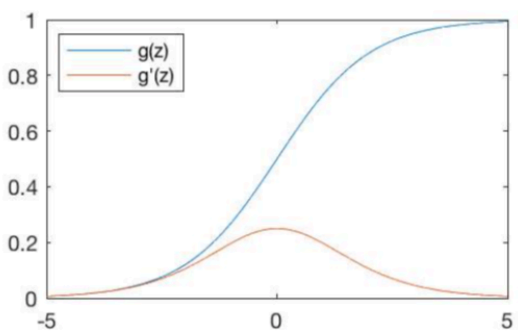
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



Common activation functions

Traditional (logistic)

Sigmoid Function



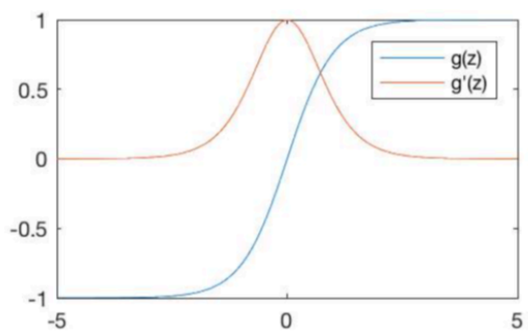
$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

`tf.nn.sigmoid(z)`

Commonly used now

Hyperbolic Tangent

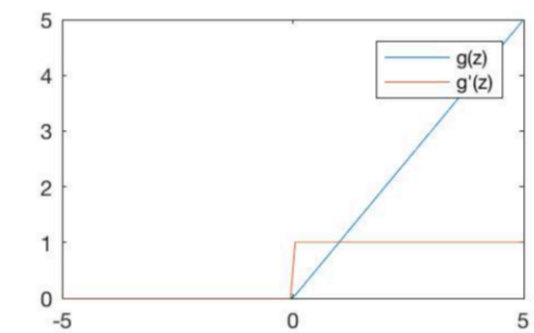


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

`tf.nn.tanh(z)`

Rectified Linear Unit (ReLU)



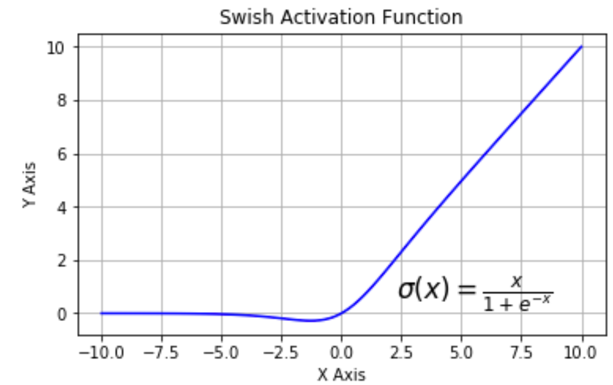
$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

`tf.nn.relu(z)`

New trend

- **Swish**

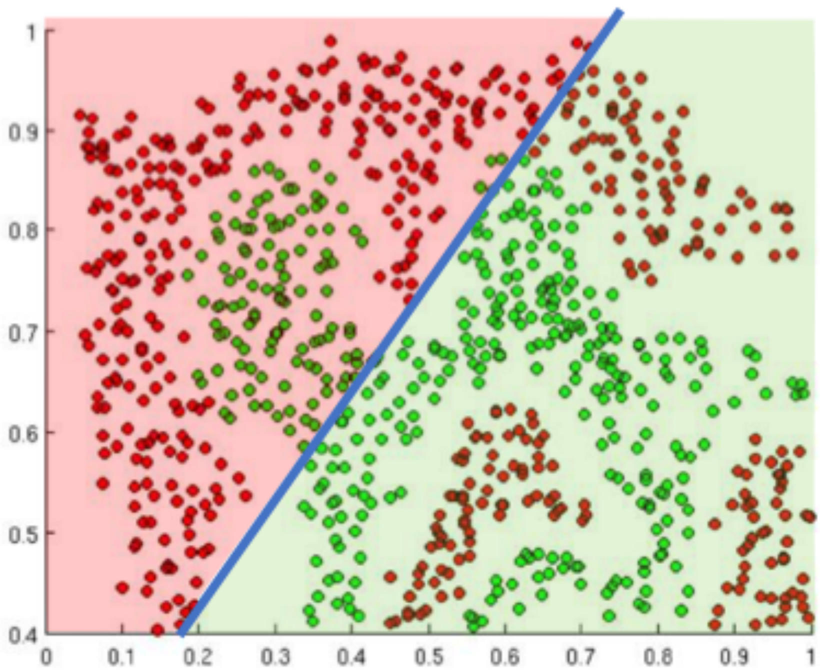


- Searching for activation functions

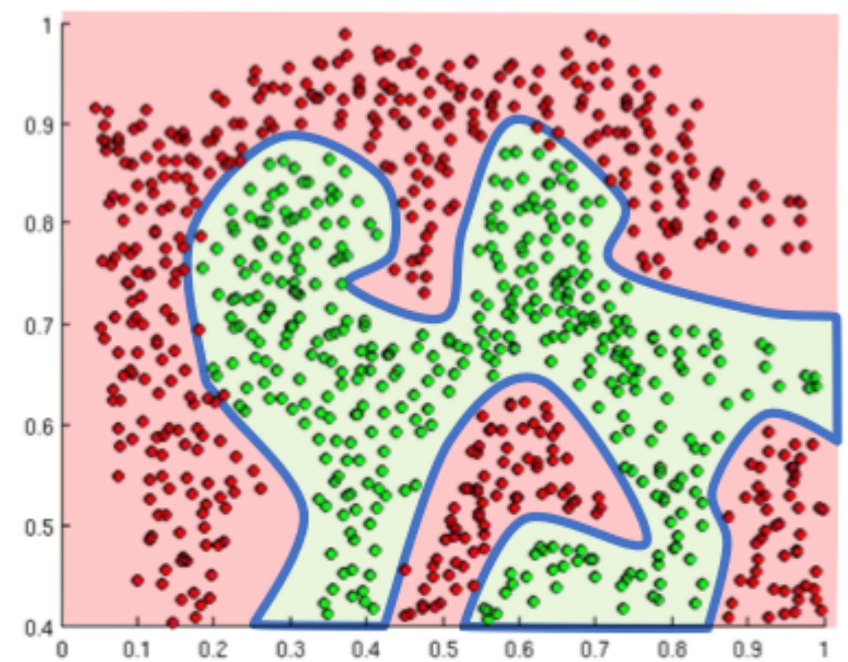
NOTE: all activation functions are non-linear, why?, important hyperparameter

Importance of activation functions

- The purpose of activation functions is to introduce **non-linearities** into the **network**



Linear Activation functions produce linear decisions no matter the network size

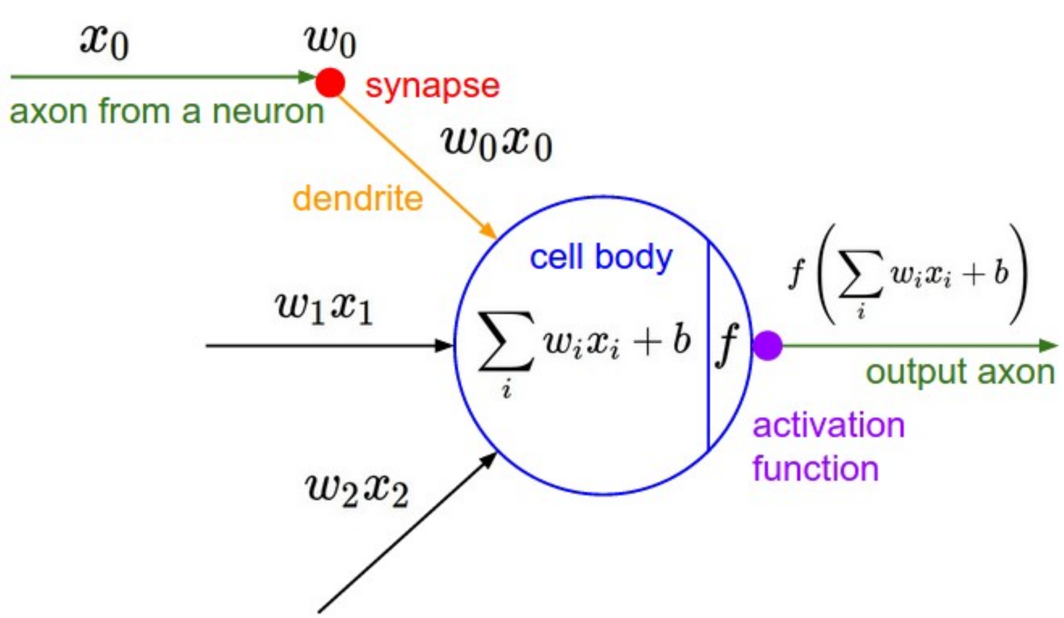


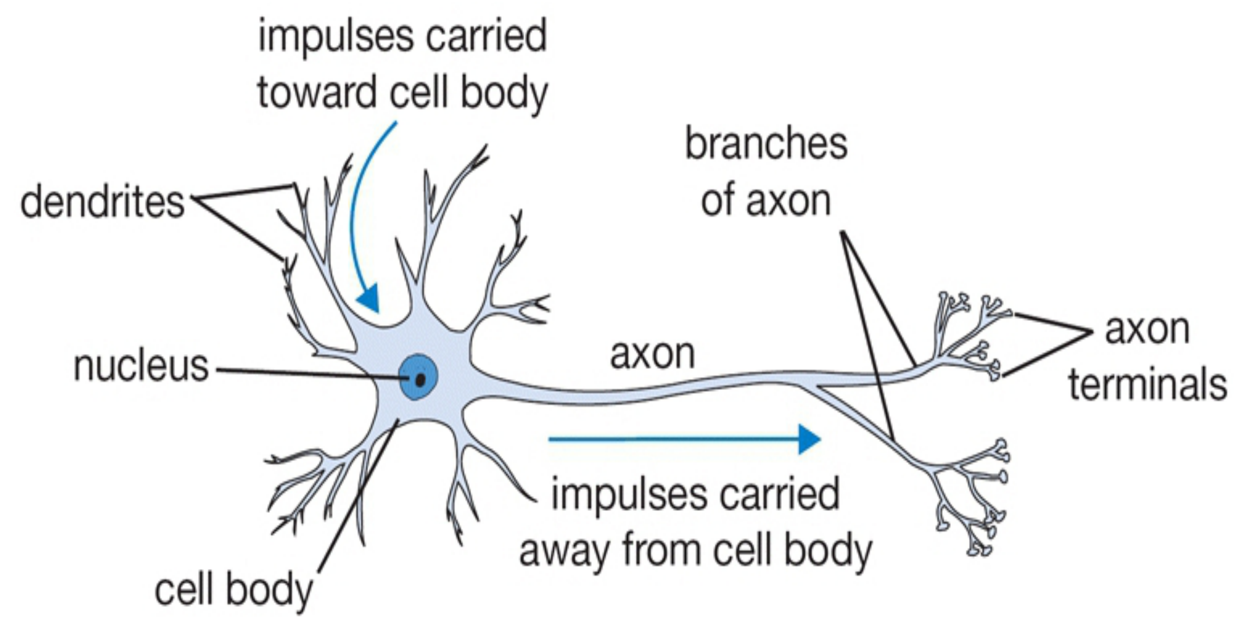
Non-linearities allow us to approximate arbitrarily complex functions

Building Artificial Neural Networks

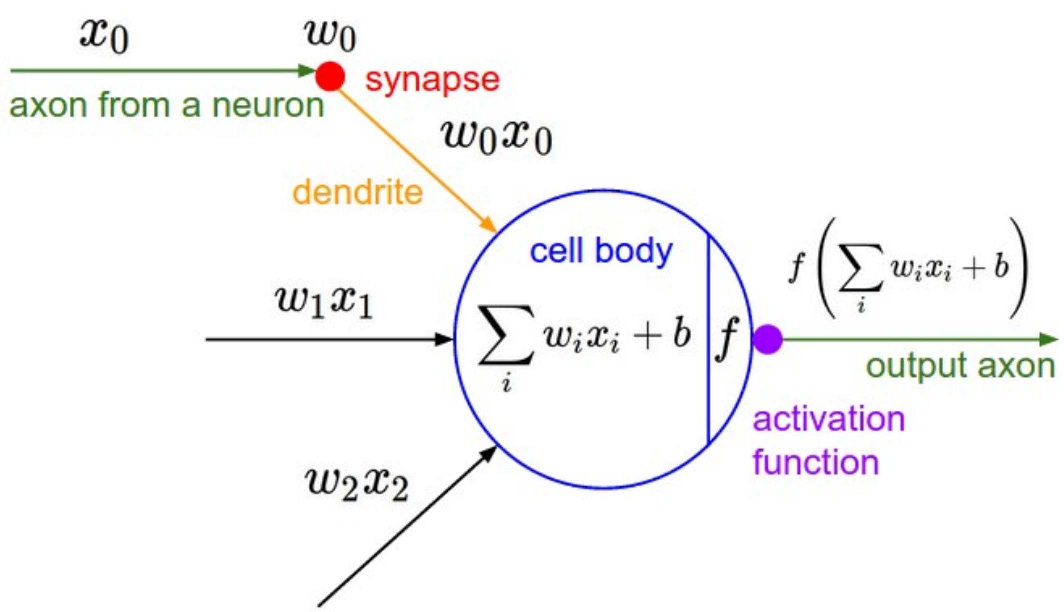
Source: MIT 6.S191: http://introtodeeplearning.com

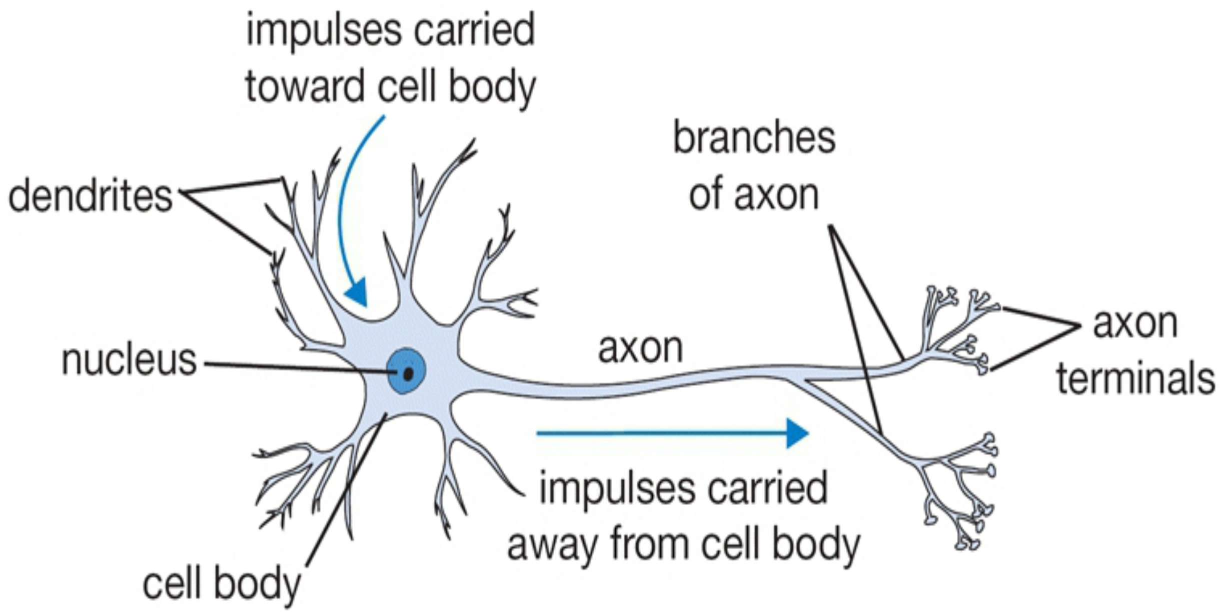
Artificial Neuron: Simplified Display





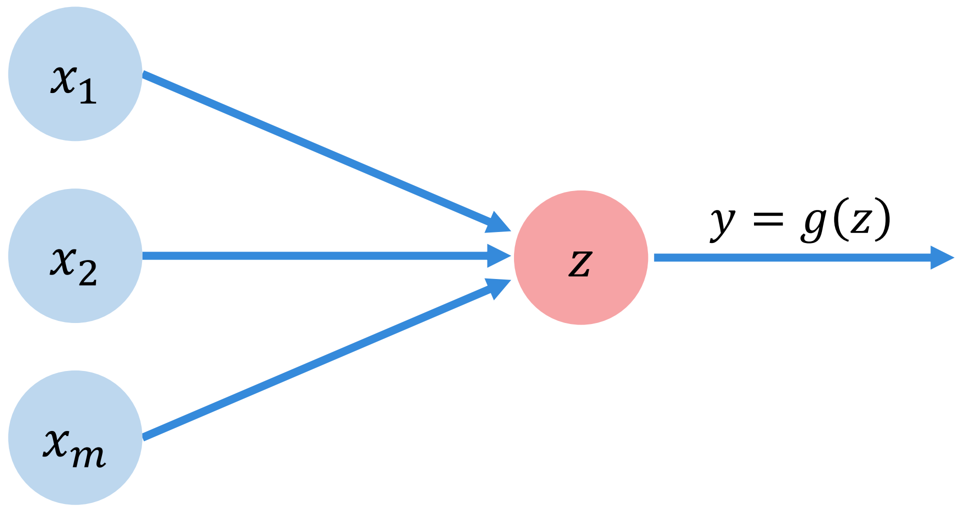
Artificial Neuron: Simplified Display





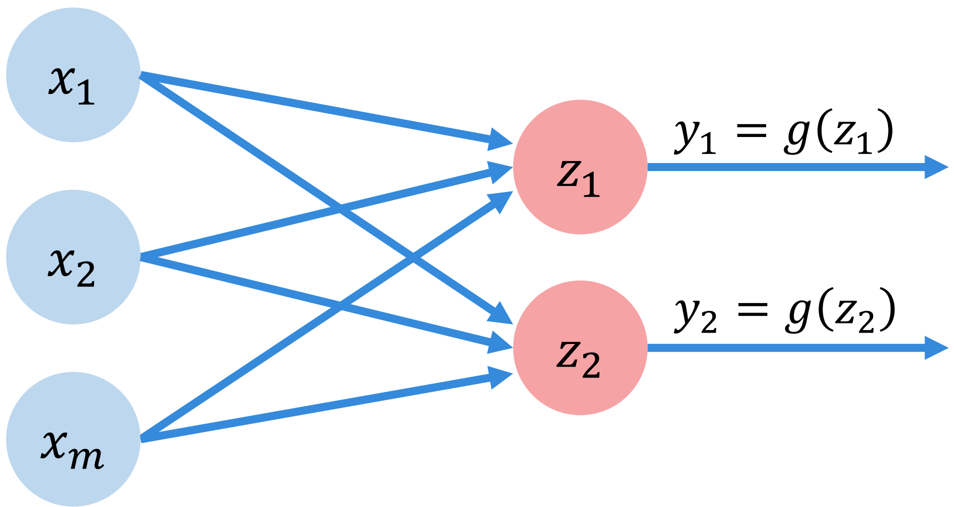

```
class Neuron(object):
  # ...
  def forward(self, inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

Artificial Neuron: Simplified Display



$$z = w_0 + \sum_{j=1}^{m} x_j w_j$$

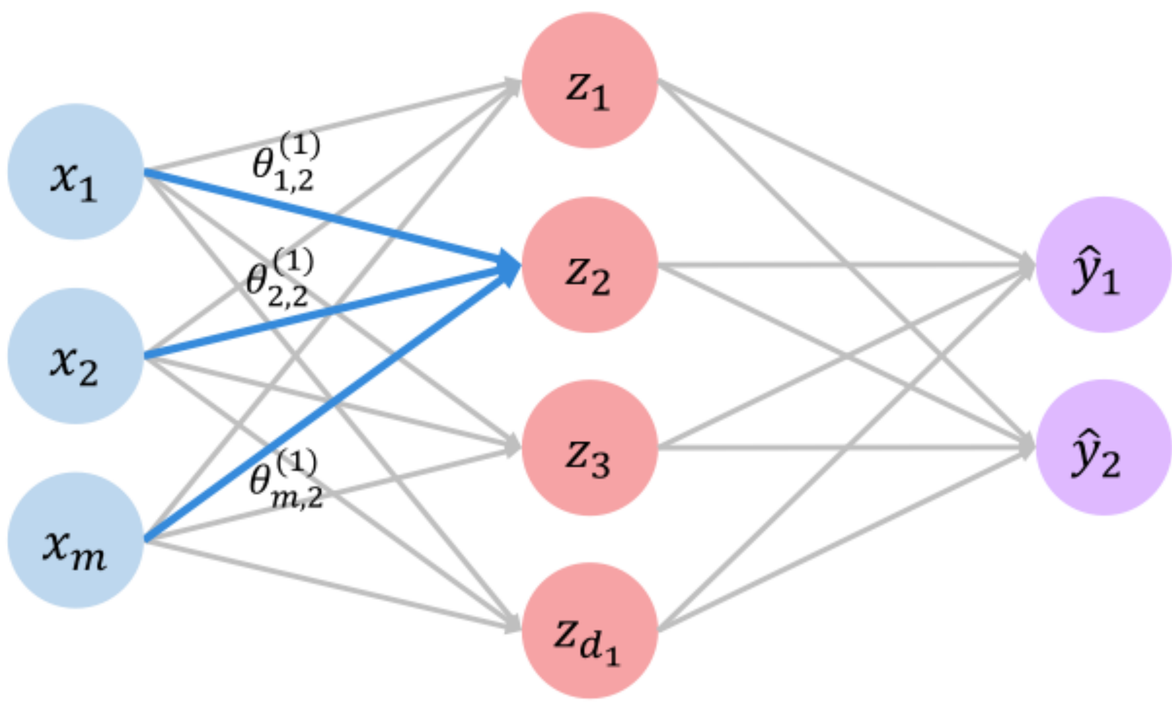
Artificial Neuron: Multi Output Perceptron



$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

Output Type	Output Distribution	Output Layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary cross-entropy
Discrete	Multinoulli	Softmax	Discrete cross-entropy
Continuous	Gaussian	Linear	Gaussian cross-entropy (MSE)
Continuous	Mixture of Gaussian	Mixture Density	Cross-entropy
Continuous	Arbitrary	See part III: GAN, VAE, FVBN	Various

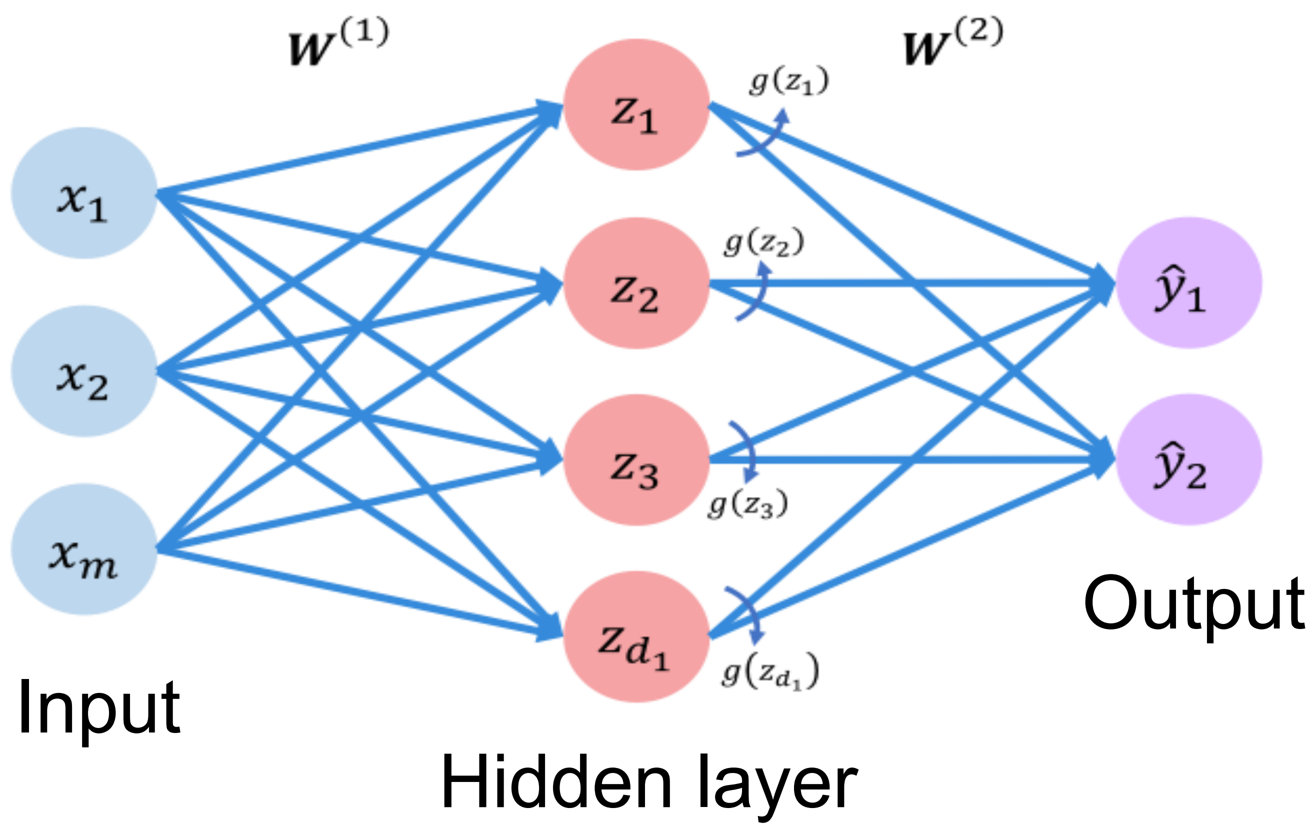
Artificial Neuron Network (with one hidden layer)



$$
\begin{aligned}
z_2 &= w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)} \\
&= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 \, w_{2,2}^{(1)} + x_m \, w_{m,2}^{(1)}
\end{aligned}
$$

Artificial Neuron Network (with one hidden layer)

- Number of neurons: 4 + 2 (input layer is not counted)
- Number of parameters: 3*4 + 4*2 + bias (4 + 2) = 26



linear/nonlinearity before output?

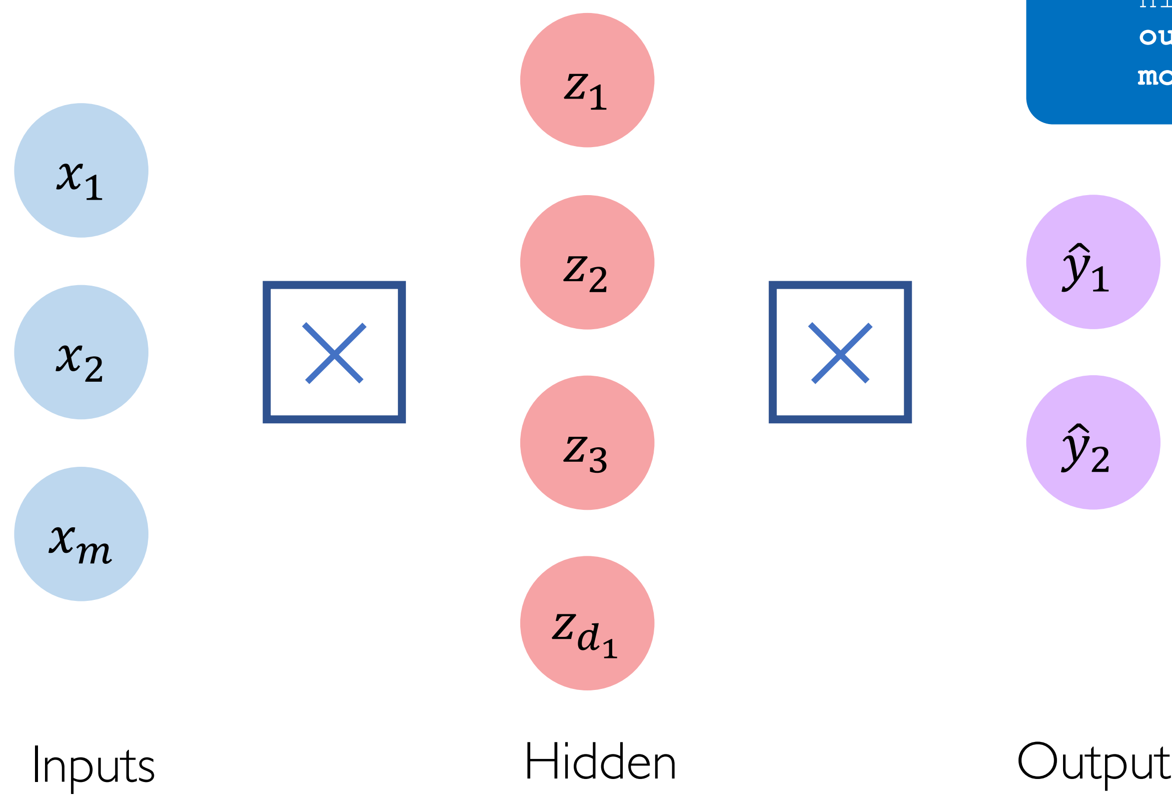
$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j) \, w_{j,i}^{(2)} \right)$$

Artificial Neuron Network (with one hidden layer)

- Simplified display for fully connected (Dense) layers

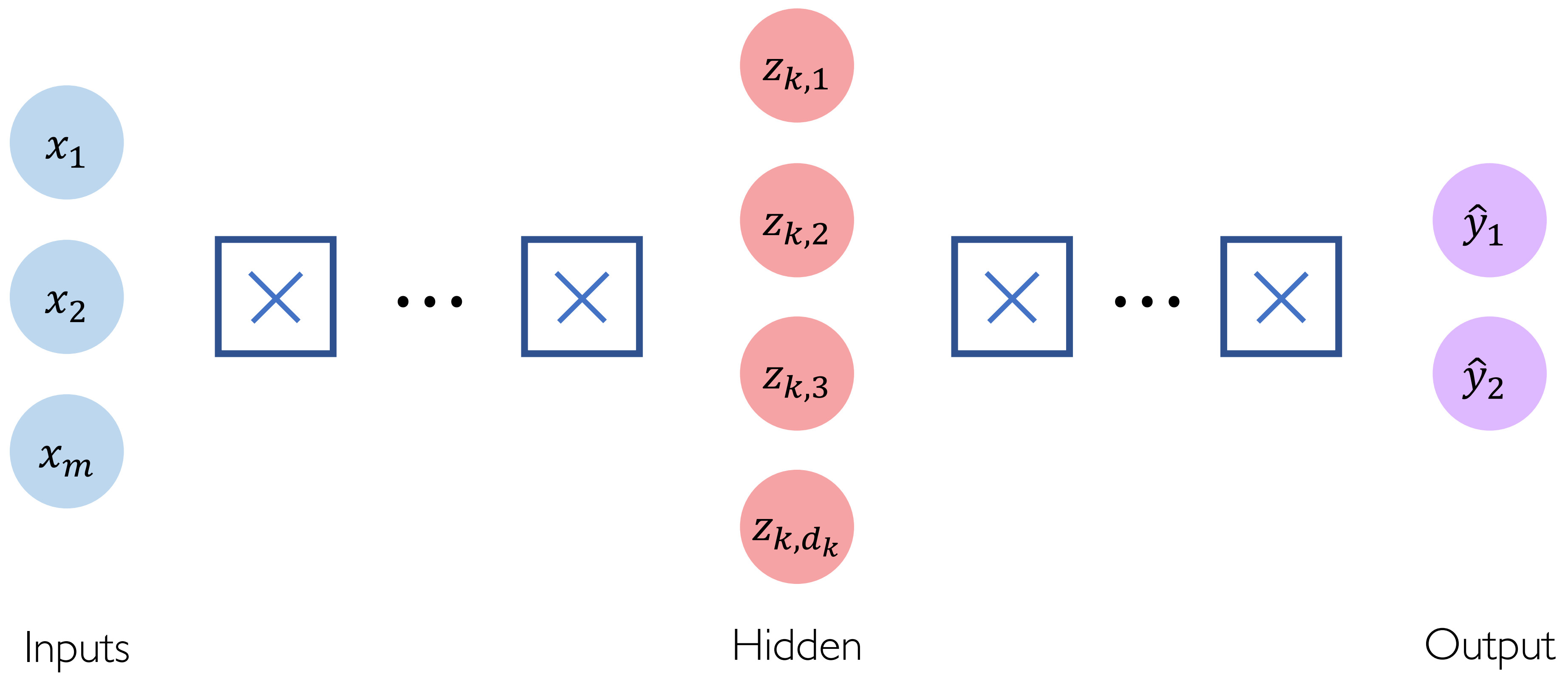
```
from tf.keras.layers import *

inputs = Inputs(m)
hidden = Dense(d1)(inputs)
outputs = Dense(2)(hidden)
model = Model(inputs, outputs)
```



Deep (Feed-Forward) Neuron Network

- Simplified display for fully connected (Dense) layers

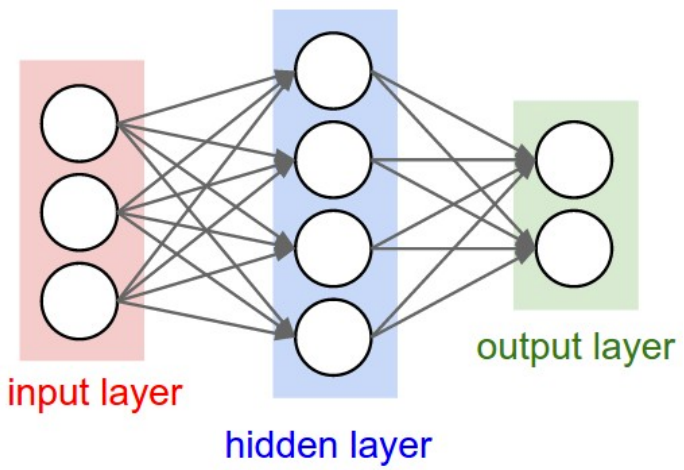


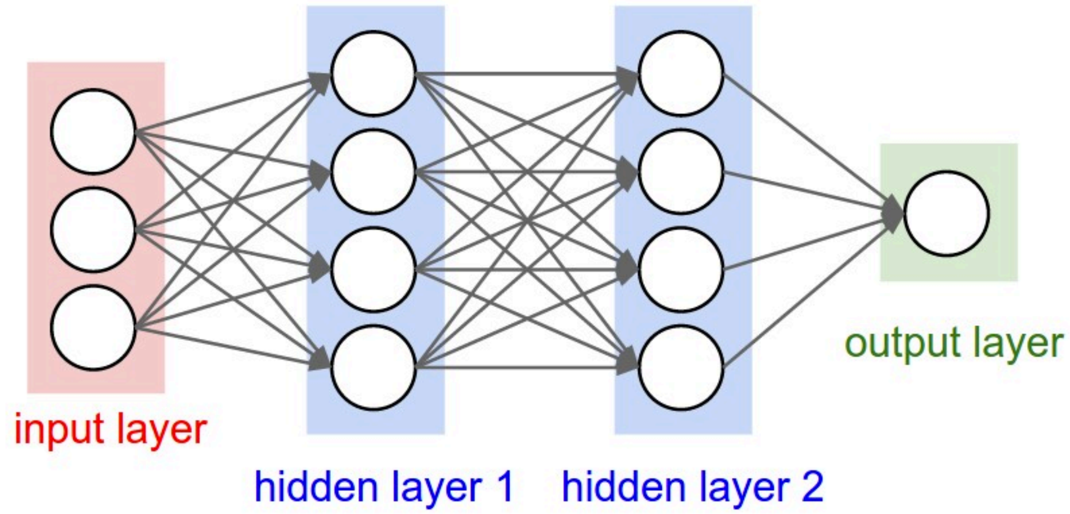
$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g_{\mathbf{k-1}}(z_{k-1,j})\, w_{j,i}^{(k)}$$

- here **z** value is a layer output before nonlinearity (depends on convention)
- **in general** each **layer k** can have different nonlinearity
- this depends on architecture choice

Deep (Feed-Forward) Naming convention

Layer type: fully-connected (Danse) layer





2-layer Neural Network with:
- three inputs &
- one hidden layer of 4 neurons (or units) &
- one output layer with 2 neurons.
- Total number neurons: 4 + 2 = 6
- Total of learnable parameters:
    - [3x4] + [4x2] + 4 + 2 (biases) = 26

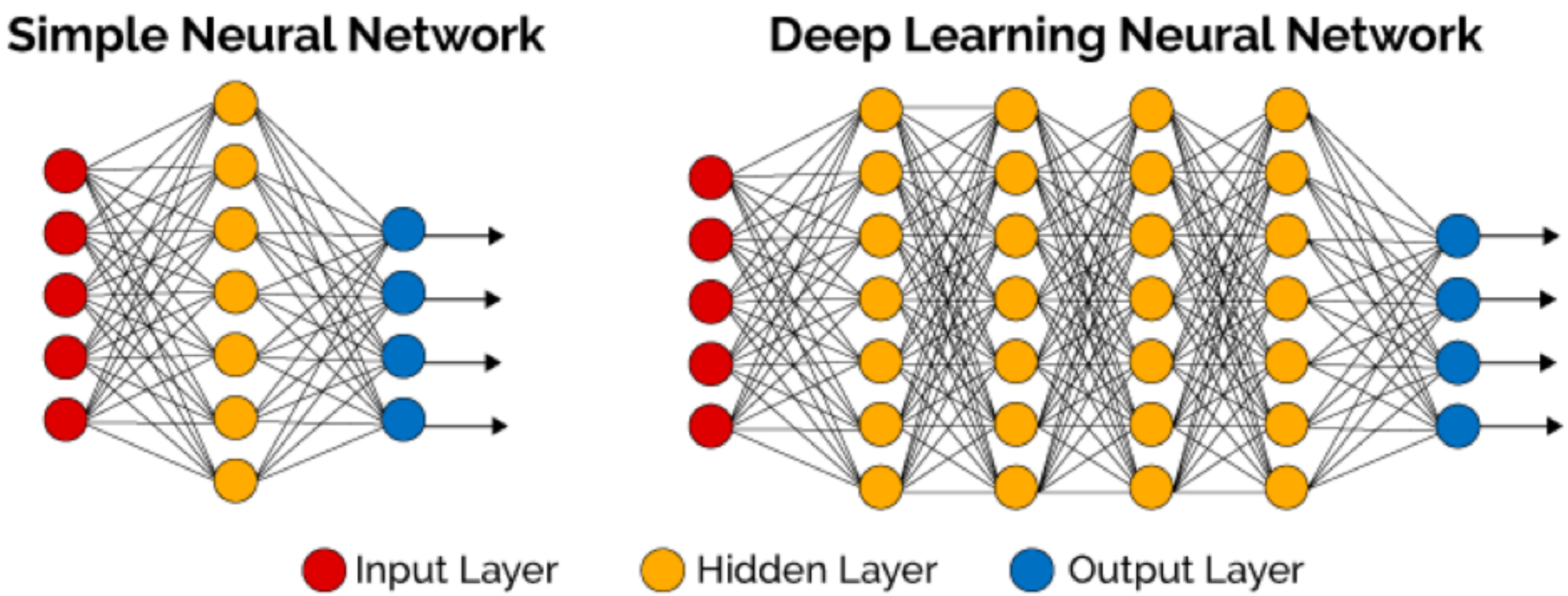
3-layer Neural Network with:
- three inputs &
- two hidden layers of 4 neurons each &
- one output layer
- Total number neurons: 4 + 4 + 1 = 9
- Total of learnable parameters:
    - [3x4] + [4*4] + [4x1] + 4 + 4 +1 (biases) = 41

Power of NN: Universal Approximation Theorem

A feed-forward network with a single layer is sufficient to *represent* (**not learn**)

an approximation of any function to an arbitrary degree of accuracy

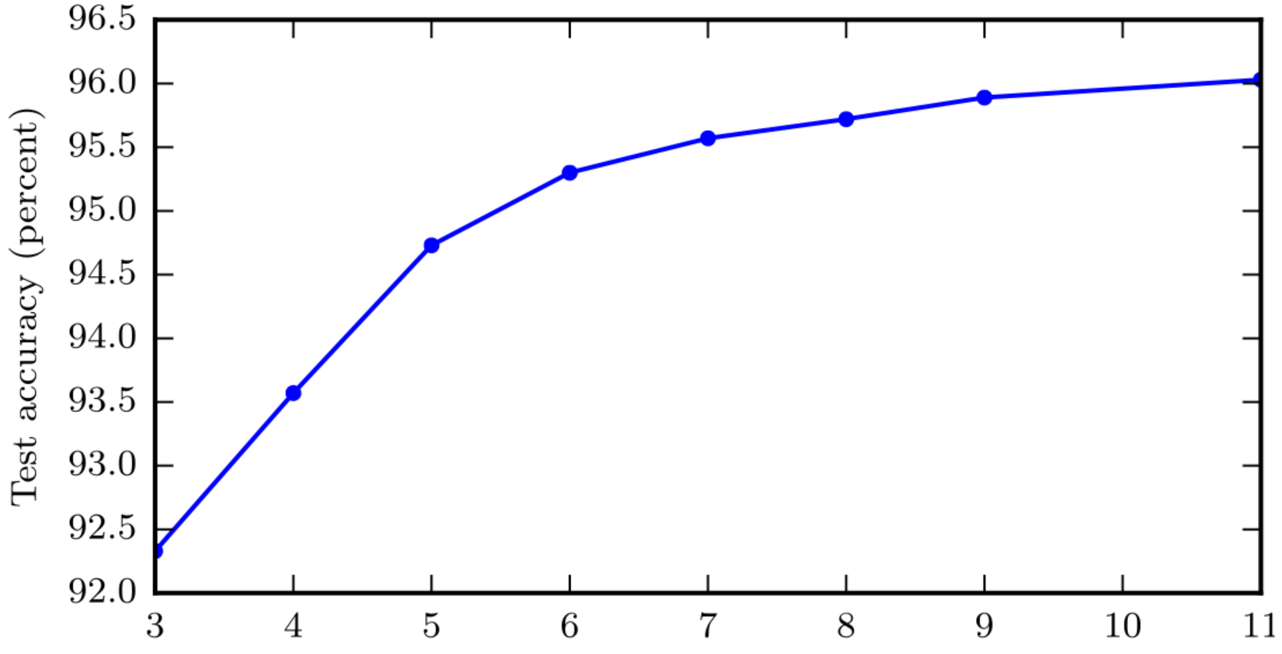
(for Intuitive explanation read: http://neuralnetworksanddeeplearning.com/chap4.html)



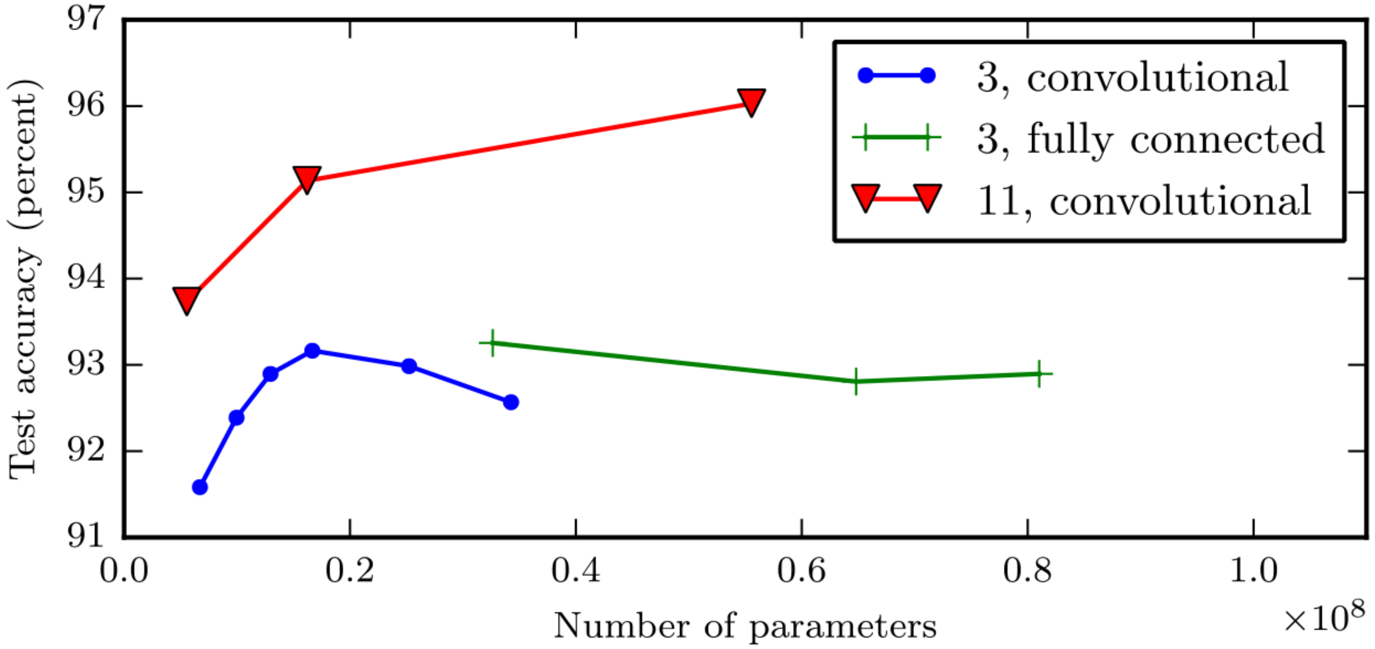
So why deep NN?
- Shallow net may need (exponentially) more width
- Shallow net may overfit more (may not generalize)

Example: Better Generalization with Greater Depth

Effect of depth.



Effect of number of parameters.

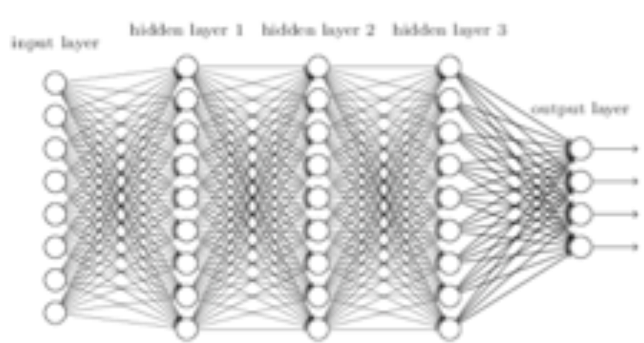


Goodfellow et. al., Deep Learning, 2017, http://www.deeplearningbook.org/ (Chapter 6)

Deep Neural Networks

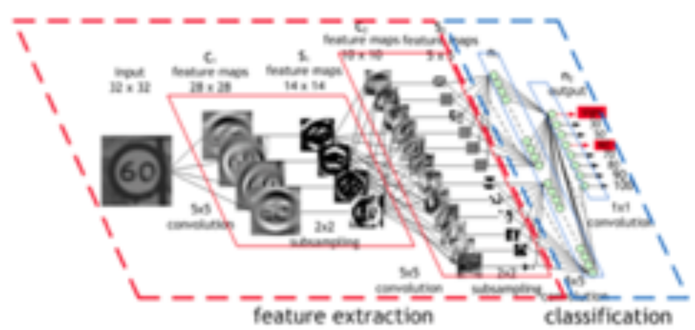
providing lift for classification and forecasting models

***Deep* Neural Networks**



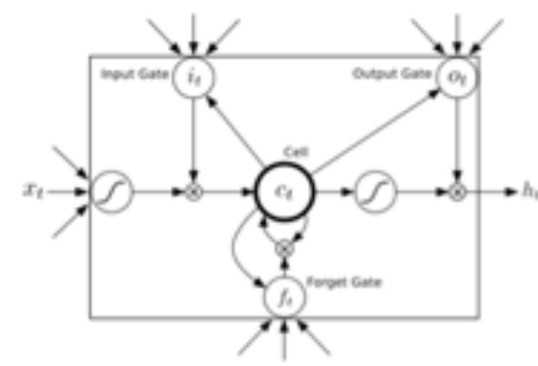
feature extraction and classification of images

***Convolutional* Neural Networks**



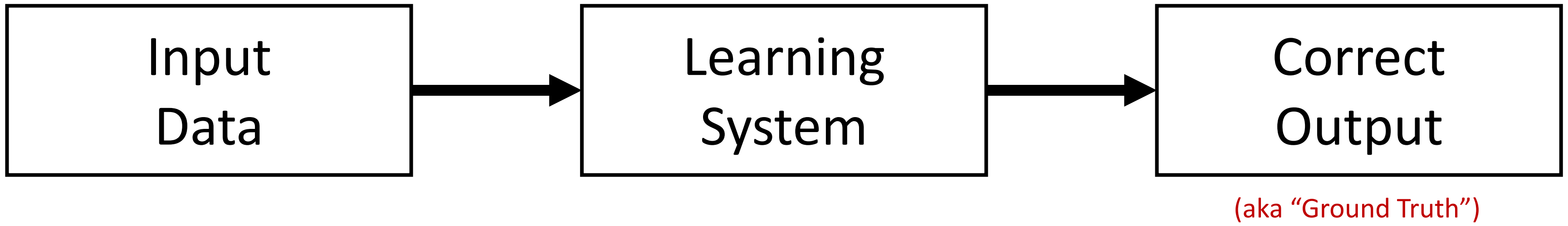
for sequence of events, language models, time series, etc.

***Recurrent* Neural Networks**

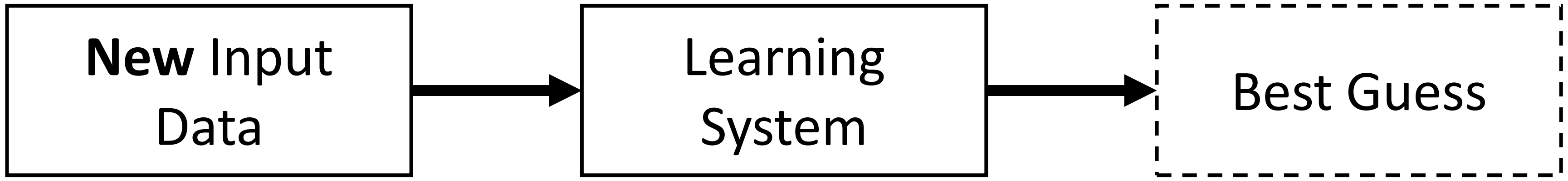


DL: Training and Testing

Training Stage:



Testing Stage:

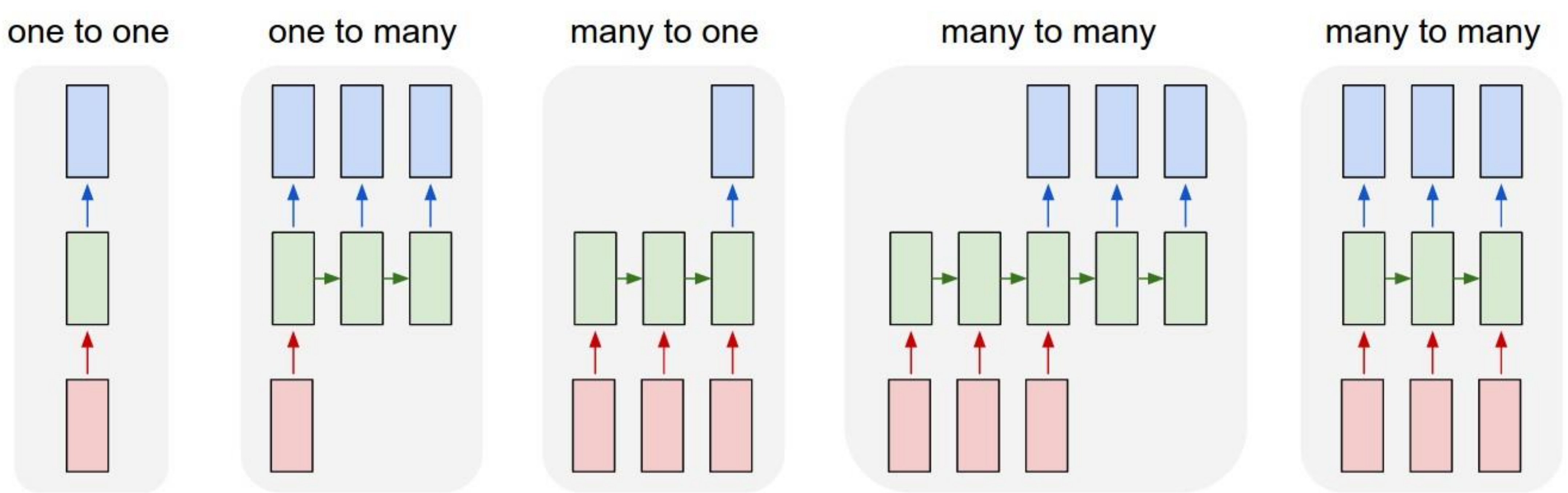


What we can do with deep learning?

Input Data → Learning System → Correct Output

- Number
- Vector of numbers
- Sequence of numbers
- Sequence of vectors of numbers

- Number
- Vector of numbers
- Sequence of numbers
- Sequence of vectors of numbers



Q: How NN Learns?

Forward Pass:

Input Data → Neural Network → Prediction

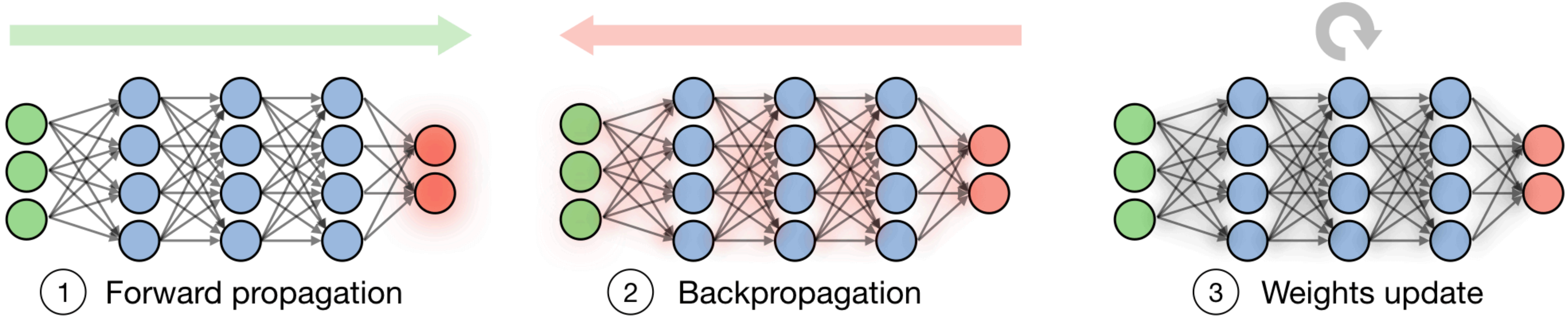
Backward Pass (aka Backpropagation):

Neural Network ← Measure of Error

Adjust to Reduce Error

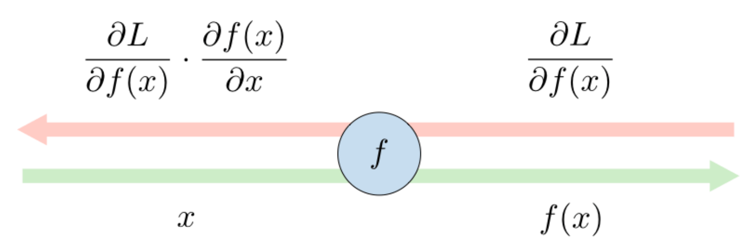
Q: How NN Learns?
A: Backpropagation + Gradient Descent

Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. **The derivative with respect to each weight is computed using the chain rule**.



Forward pass to compute network output and “error”

Backward pass to compute gradients



A fraction (learning rate) of the weight’s gradient is subtracted from the weight

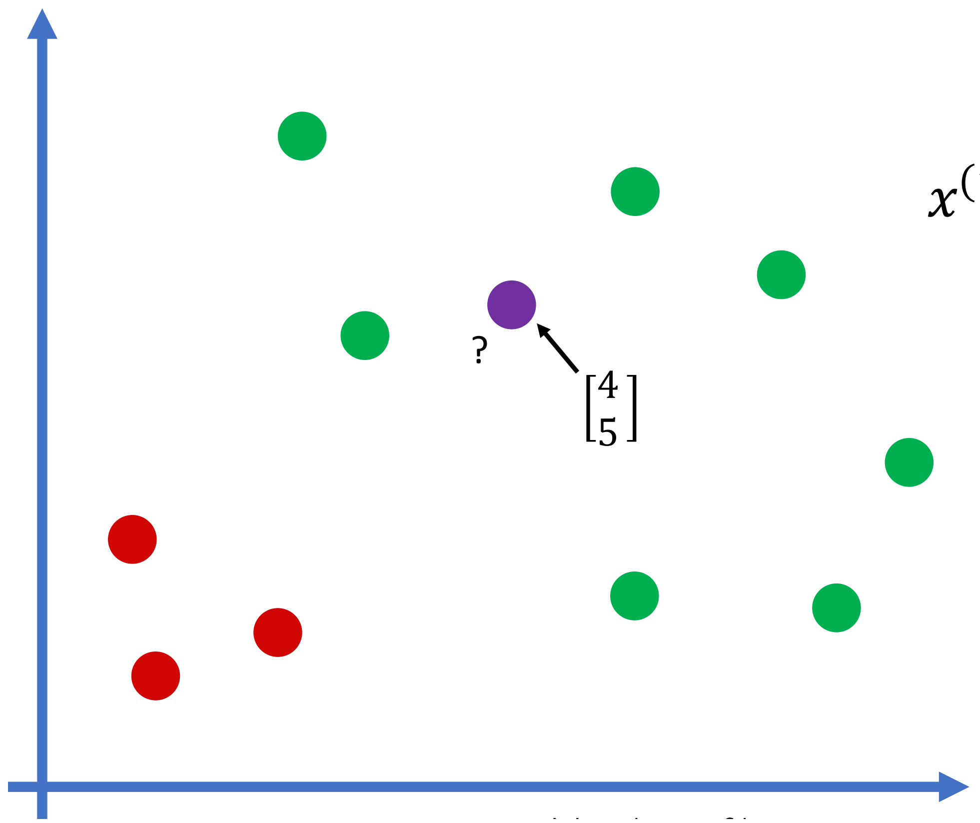
$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

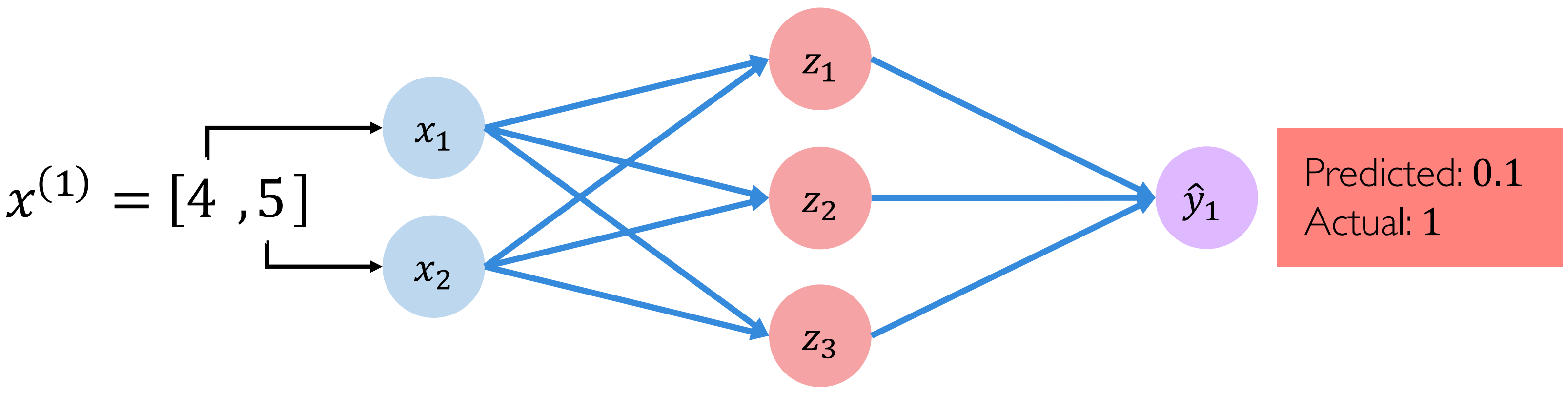
Recommended reading:

- *Backprop is very simple* (‘by hand explanation’):
  - ◇ URL: goo.gl/tYVG6J
- *Automatic differentiation in machine learning: a survey*
  - ◇ https://arxiv.org/abs/1804.07612

Basic concepts of NN Training: Quantifying Loss

=> The **loss** of our network measures the cost incurred from incorrect predictions

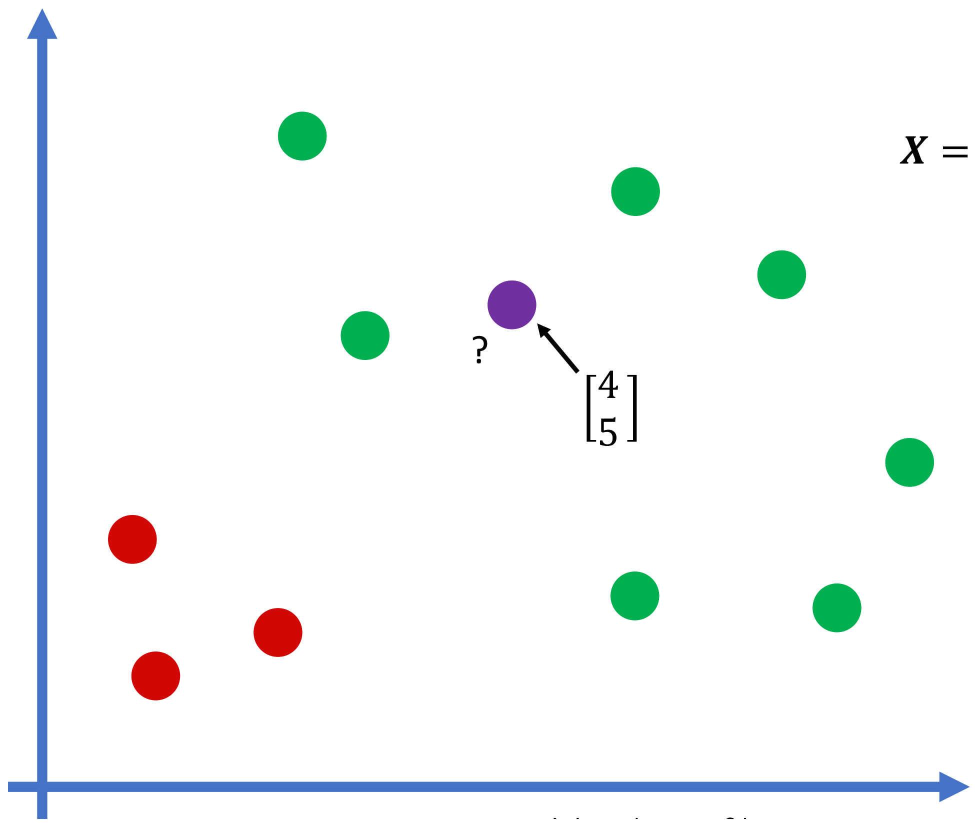




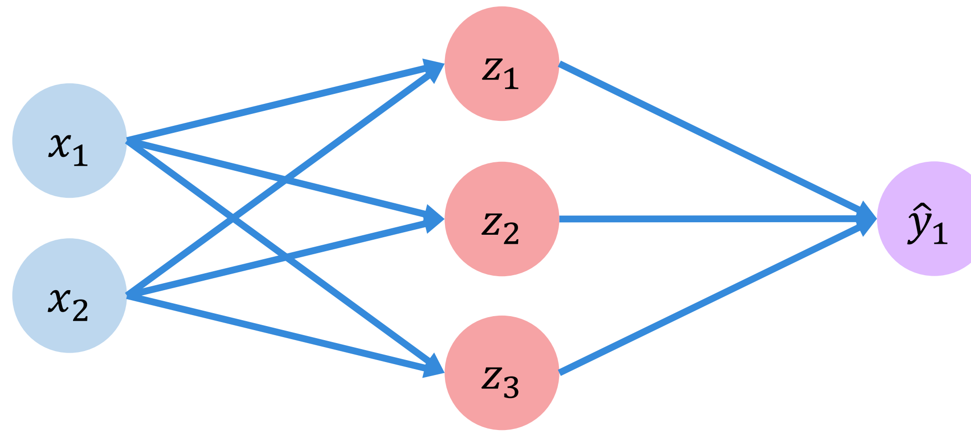
$$\mathcal{L}\Big(\underbrace{f\big(x^{(i)}; \boldsymbol{W}\big)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}\Big)$$

Basic concepts of NN Training: Empirical Loss

=> The **empirical loss** measures the total loss over our entire dataset



$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



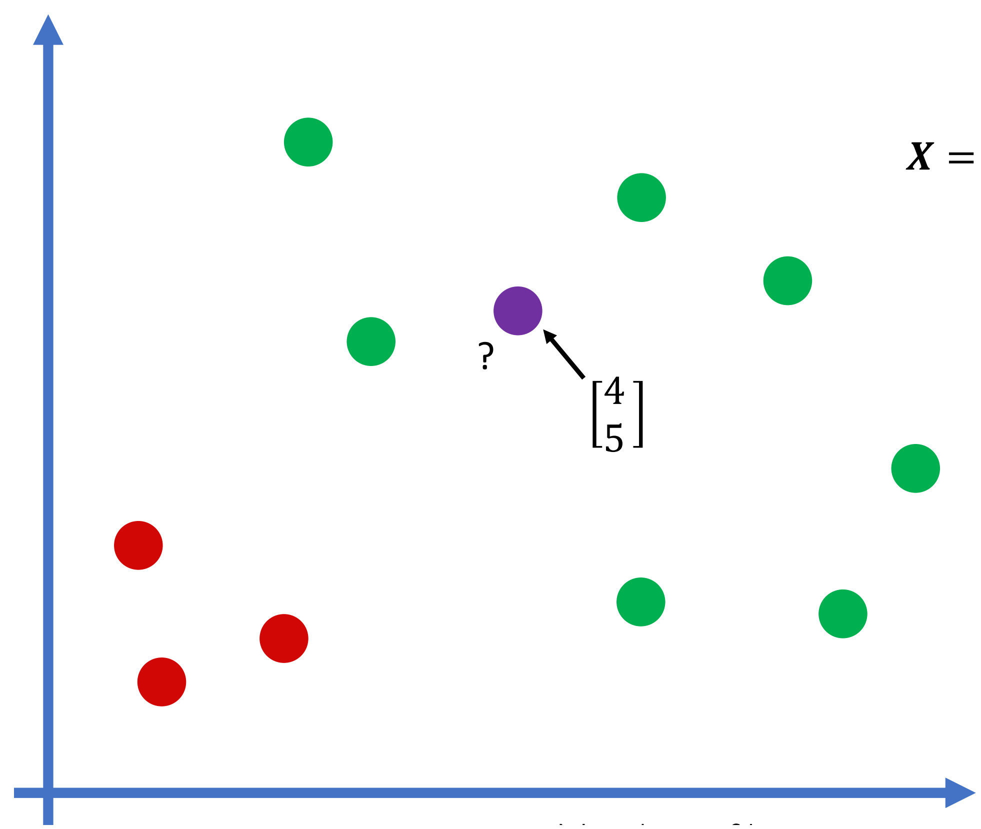
$$\begin{array}{cc} f(x) & y \\ \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \end{array}$$

Also known as:
- Objective function
- Cost function
- Empirical Risk

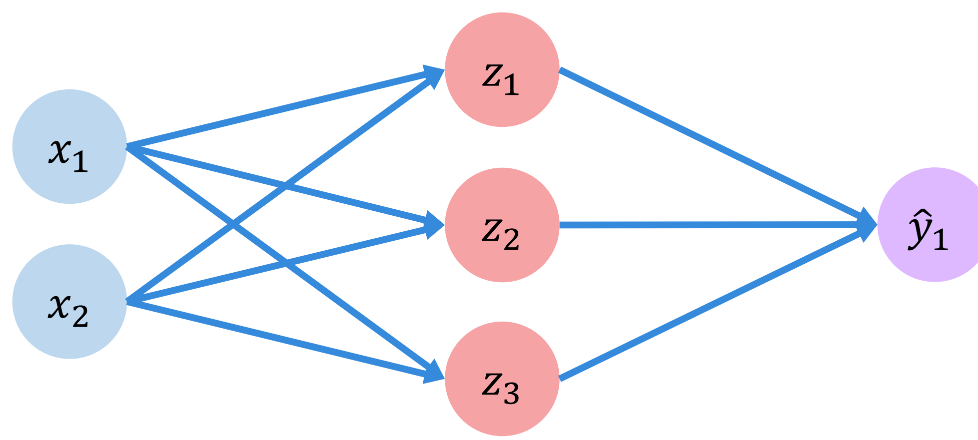
$$\boldsymbol{J}(\boldsymbol{W}) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\Big(\underbrace{f\big(x^{(i)}; \boldsymbol{W}\big)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}\Big)$$

Basic concepts of NN Training: Cross Entropy Loss

=> **Cross entropy loss** can be used with models that output a probability between 0 and 1



$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\begin{array}{cc} f(x) & y \\ \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \end{array}$$

Also known as:
- Objective function
- Cost function
- Empirical Risk

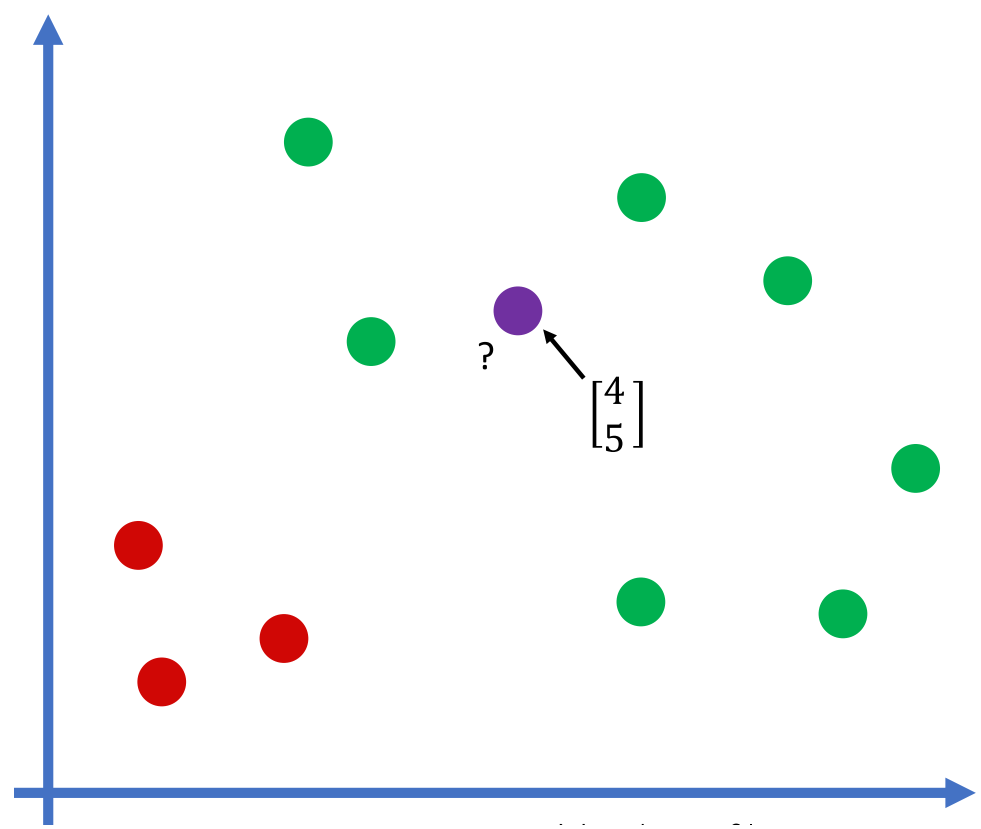
$$\boldsymbol{J}(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\Big(\underbrace{f\big(x^{(i)}; \boldsymbol{W}\big)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}\Big)$$

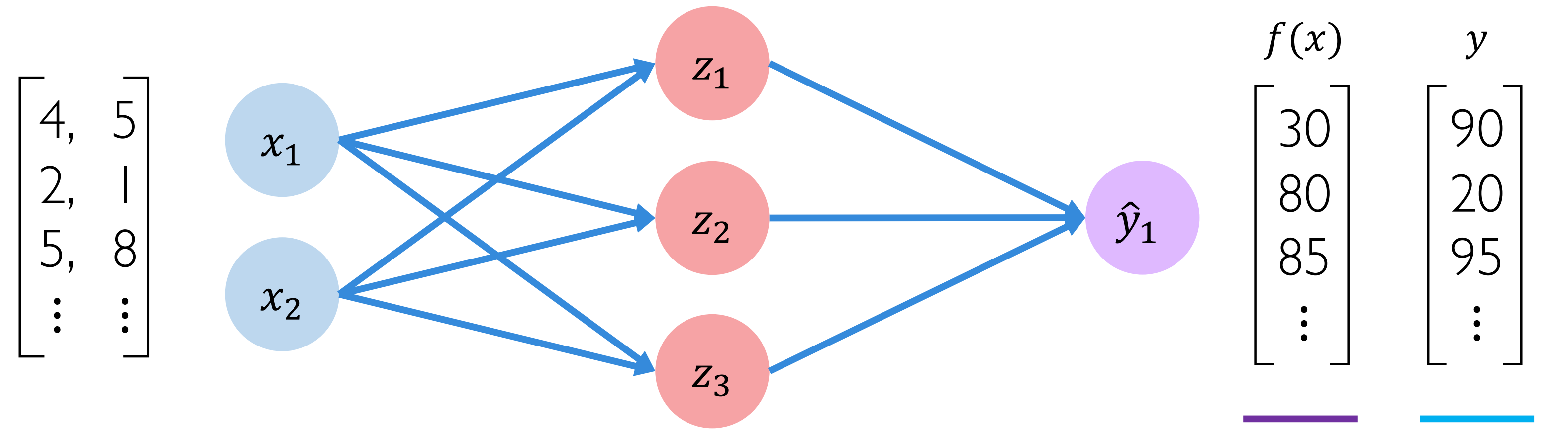
$$\boldsymbol{J}(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log\Big(\underbrace{f\big(x^{(i)}; \boldsymbol{W}\big)}_{\text{Predicted}}\Big) + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log\Big(1 - \underbrace{f\big(x^{(i)}; \boldsymbol{W}\big)}_{\text{Predicted}}\Big)$$

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(model.y, model.pred) )
```

Basic concepts of NN Training: Mean Squared Error

=> **Mean squared error loss** can be used with regression models that output continuous real numbers





Also known as:
- Objective function
- Cost function
- Empirical Risk

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\Big(\underbrace{f\big(x^{(i)}; \boldsymbol{W}\big)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}\Big)$$

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \Big(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\big(x^{(i)}; \boldsymbol{W}\big)}_{\text{Predicted}}\Big)^2$$

```
loss = tf.reduce_mean( tf.square(tf.subtract(model.y, model.pred) )
```

Basic concepts of NN Training: Learning is optimization problem

We want to find the network weights that achieve the lowest loss

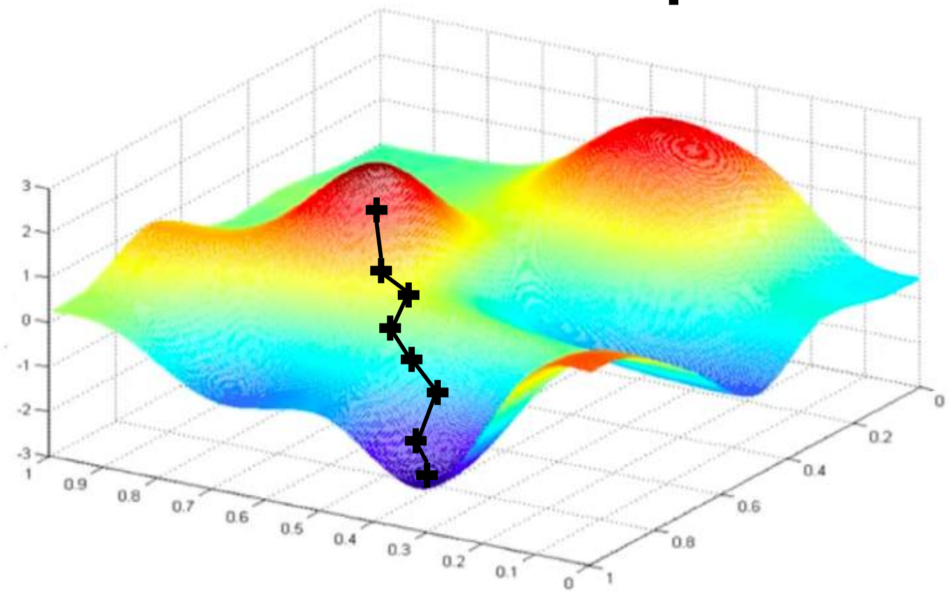
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

$$\boldsymbol{W} = \left\{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \cdots\right\}$$

NN parameters

Basic concepts of NN Training: Gradient Descent



```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
(we could initialize NN in different ways; spoiler – transfer learning)
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$ (computationaly heavy to compute for large datasets!)
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \boxed{\eta} \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$ learning rate
5. Return weights

```
weights = tf.random_normal(shape, stddev=sigma)
```

```
grads = tf.gradients(ys=loss, xs=weights)
```

```
weights_new = weights.assign(weights - lr * grads)
```

Basic concepts of NN Training: Stochastic Gradient Descent

Stochastic Gradient Descent:

→ easy to compute, but very noisy

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0,\sigma^2)$
2. Loop until convergence:
3. Pick single data point $\boldsymbol{i}$
4. $\boxed{\text{Compute gradient, } \frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights

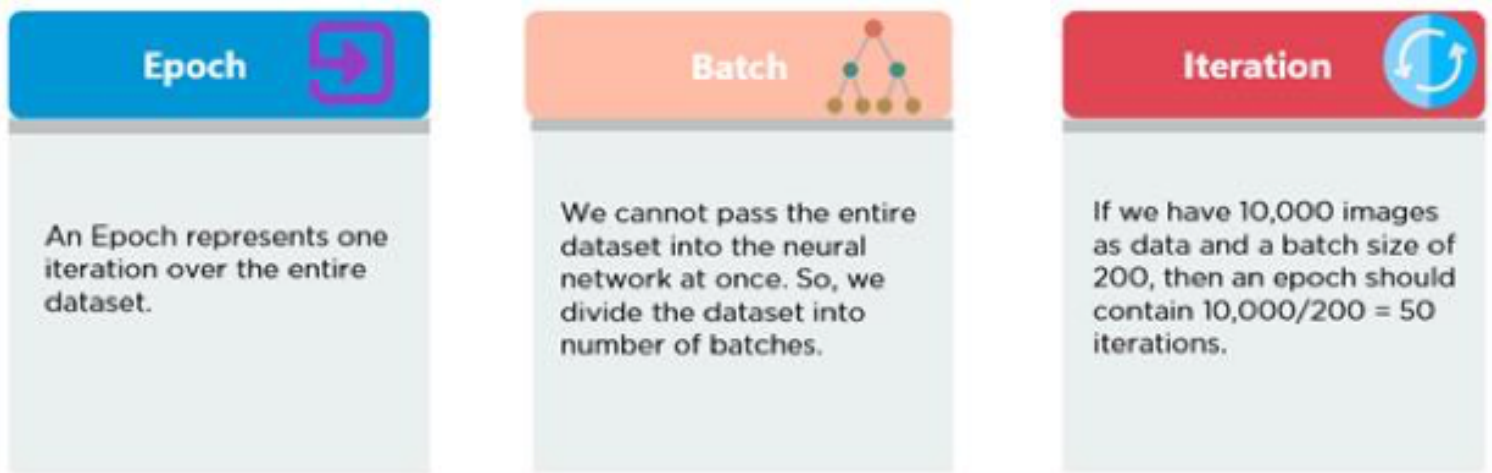
Mini-Batch Gradient Descent

→ fast to compute, much better at estimating 'true' gradient

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0,\sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. $\boxed{\text{Compute gradient, } \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B}\frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights

Basic concepts of NN Training: **Mini-batches** while training



Mini-Batch size: Number of training instances, which the network evaluates per weight update step.

- Larger batch size = more computational speed
- Smaller batch size = (empirically) better generalization

Recommendations:

- "Training with large minibatches is bad for your health. More importantly, it's bad for your test error. Friends don't let friends use minibatches larger than 32." - Yann LeCun:
  - ◇ *Revisiting Small Batch Training for Deep Neural Networks* (2018)
    - https://arxiv.org/abs/1804.07612
- It is hyperparametar usually based on memory constraints (if any, not commonly cross-validated), or set to some value, e.g. 32, 64 or 128. We use powers of 2 in practice because many vectorized operation implementations work faster when their inputs are sized in powers of 2. – A. Karpathy (cs231n Notes)

Basic concepts of NN Training: **Adaptive Learning Rates**

- Momentum — `tf.train.MomentumOptimizer` — Qian et al. "On the momentum term in gradient descent learning algorithms." 1999.
- Adagrad — `tf.train.AdagradOptimizer` — Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.
- Adadelta — `tf.train.AdadeltaOptimizer` — Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.
- Adam — `tf.train.AdamOptimizer` — Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.
- RMSProp — `tf.train.RMSPropOptimizer`

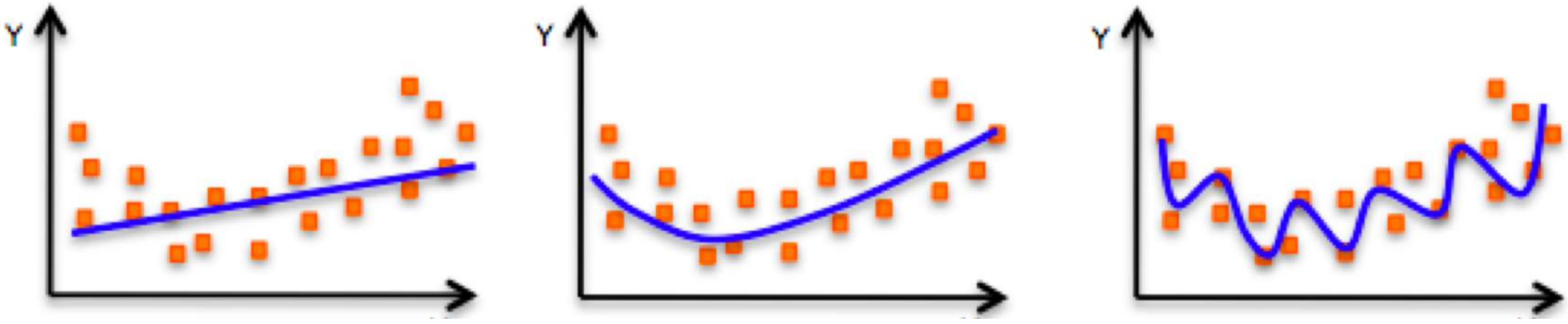
Recommended reading (for details):

- http://ruder.io/optimizing-gradient-descent/

Basic concepts of NN Training: **Regularization**

=> **Technique** that constrains our optimization problem to discourage complex models

Improve generalization of our model on unseen data



Underfitting
Model does not have capacity to fully learn the data

Ideal fit

Overfitting
Too complex, extra parameters, does not generalize well

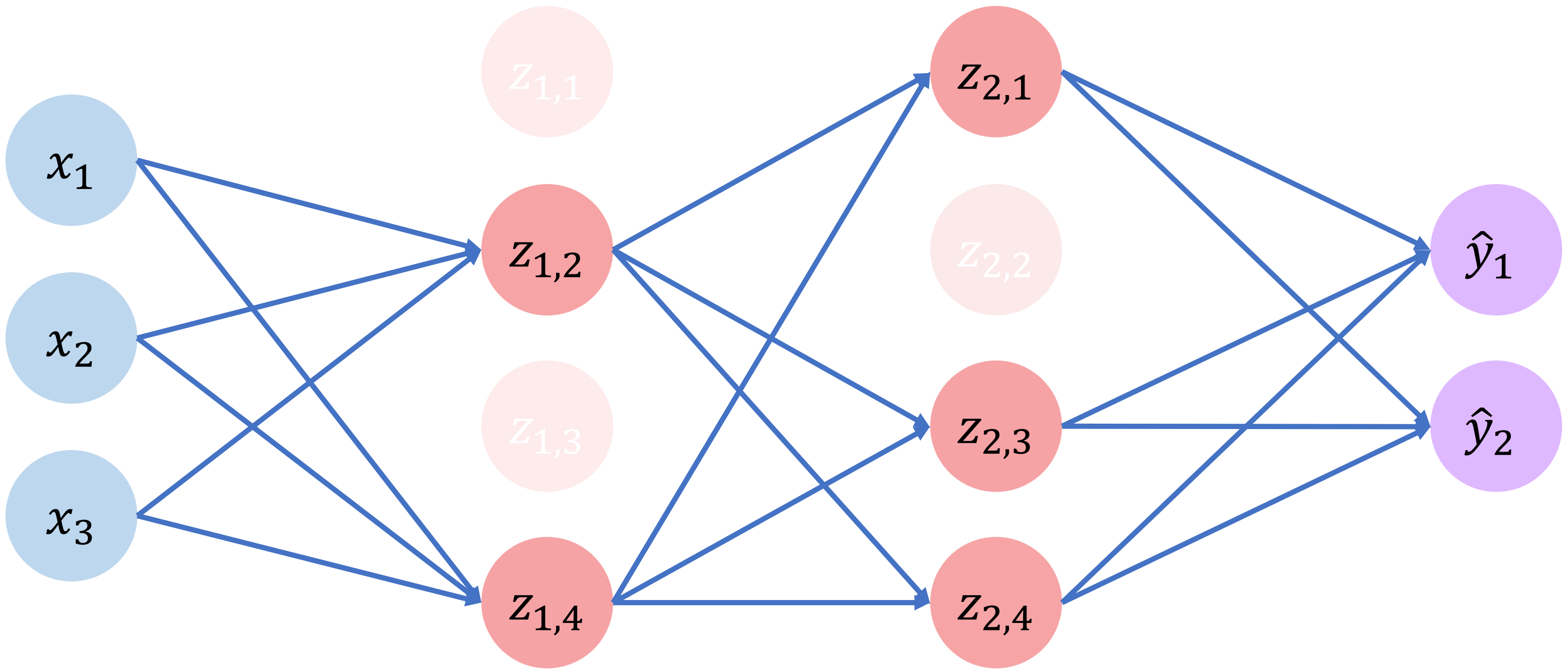
Regularization: Dropout

=> During training, randomly set some activations to 0

Typically ‘drop’ 50% of activations in layer

Forces network to not rely on any 1 node

```
tf.keras.layers.Dropout(p=0.5)
```



Regularization: Dropout implementation example

```python
"""
Inverted Dropout: Recommended implementation example.
We drop and scale at train time and don't do anything at test time.
"""

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

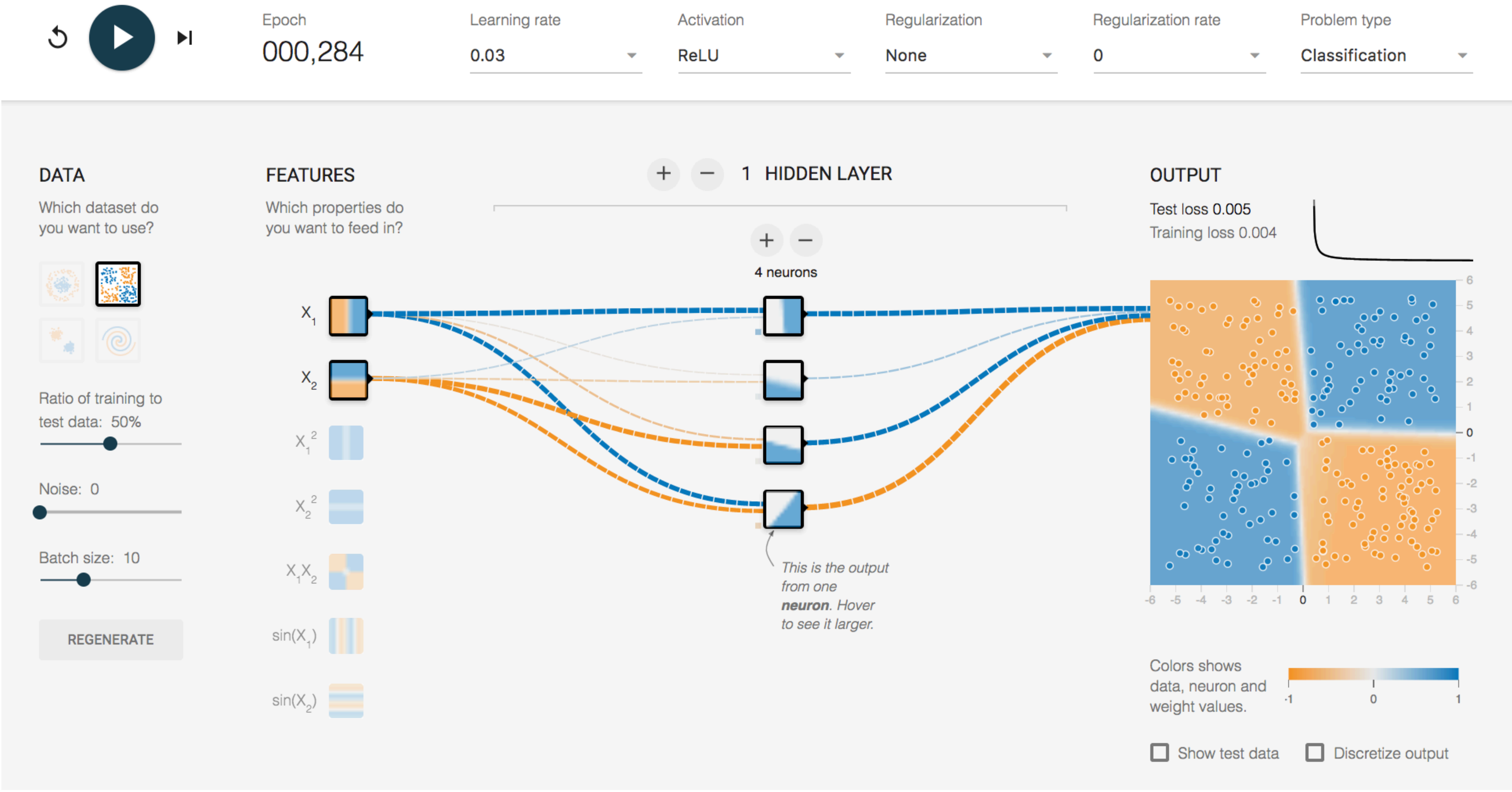
(http://cs231n.github.io/)

Regularization: Early stopping

=> Stop training before we have chance to overfit

Demo: Neural Network Playground

https://playground.tensorflow.org



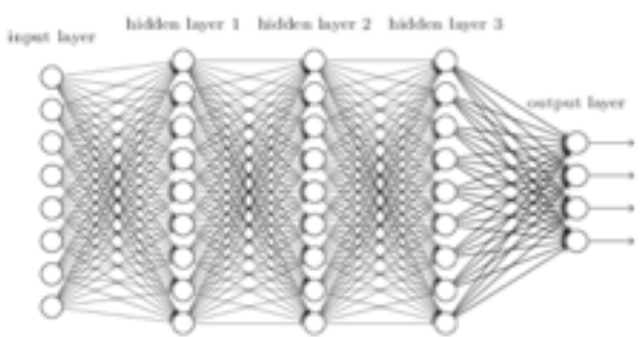
Hands-on Materials

https://tinyurl.com/y5jb7d7b

Deep Neural Networks

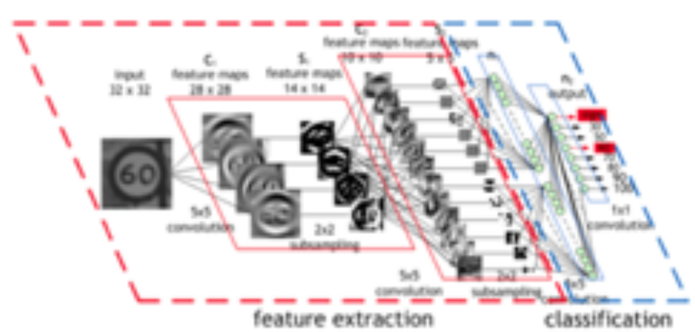
providing lift for classification and forecasting models

Deep
Neural Networks



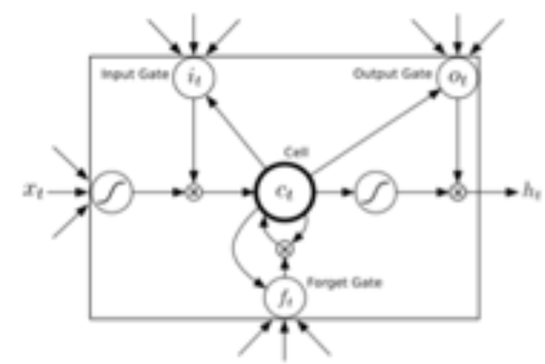
feature extraction and classification of images

Convolutional
Neural Networks



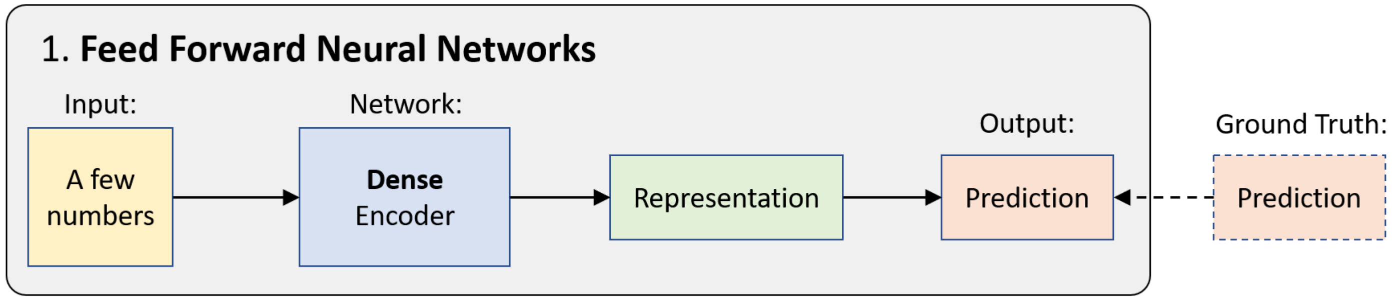
for sequence of events, language models, time series, etc.

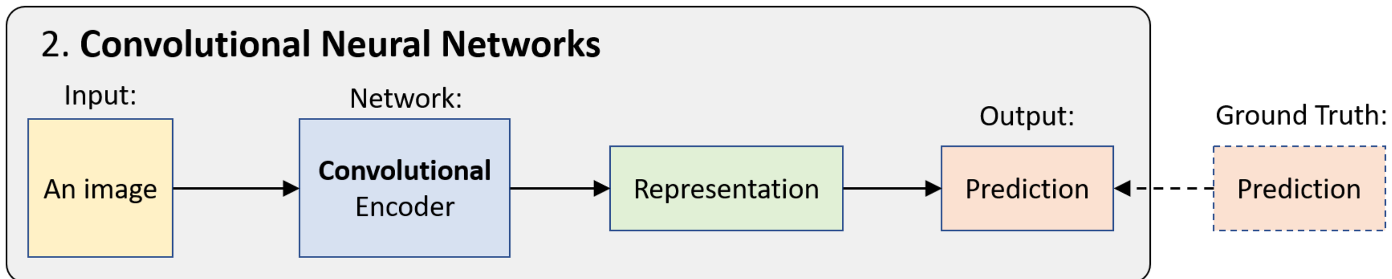
Recurrent
Neural Networks



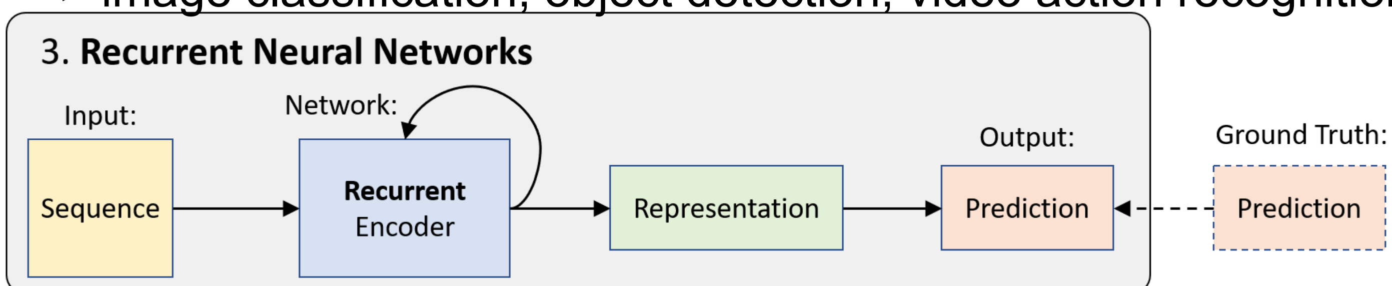
Deep Neural Networks

Supervised Learning

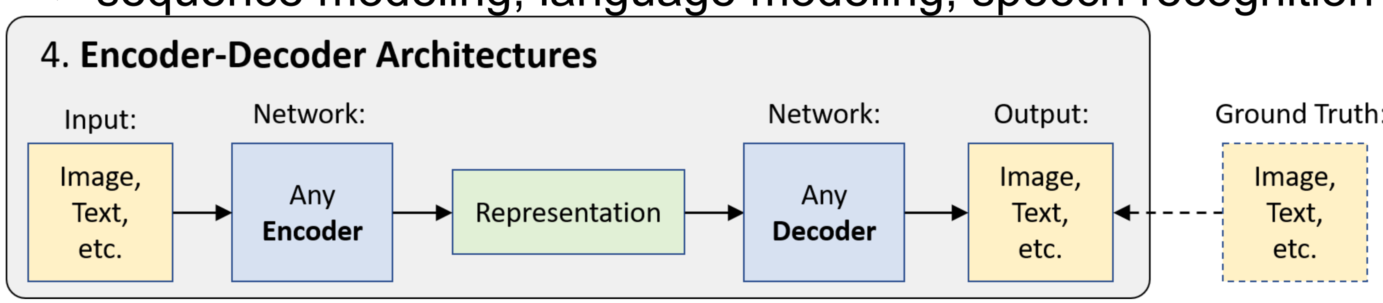




=> image classification, object detection, video action recognition

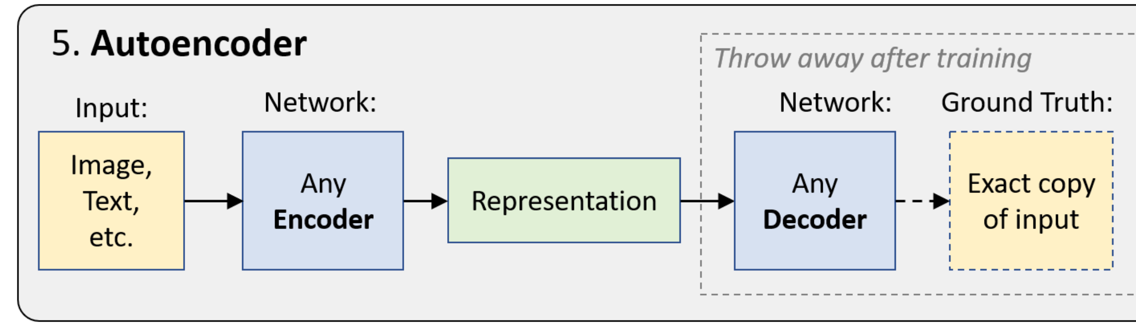


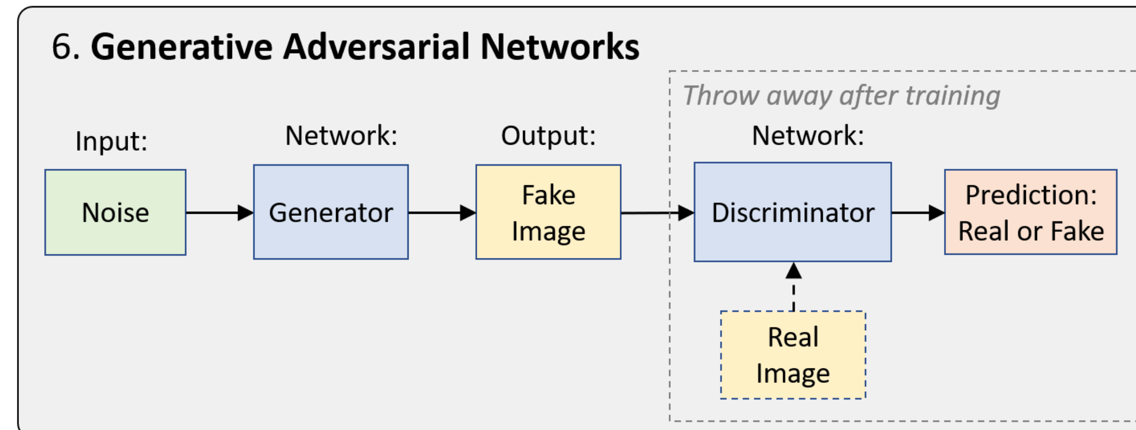
=> sequence modeling, language modeling, speech recognition



=> semantic segmentation, machine translation

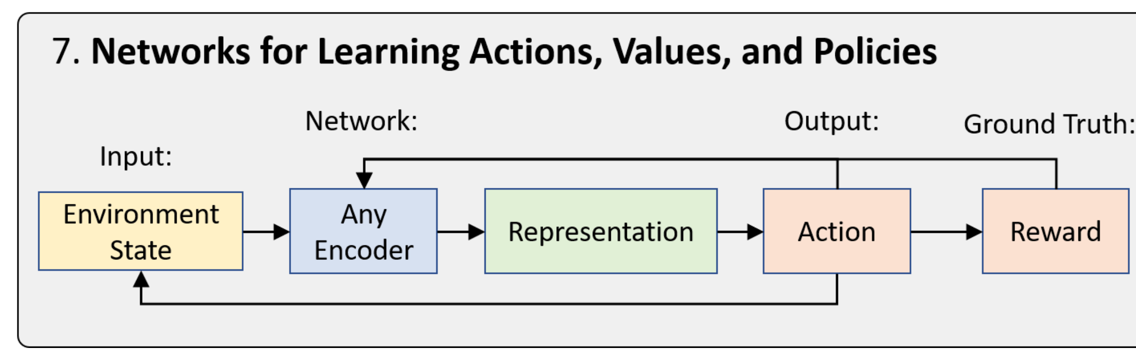
Unsupervised Learning





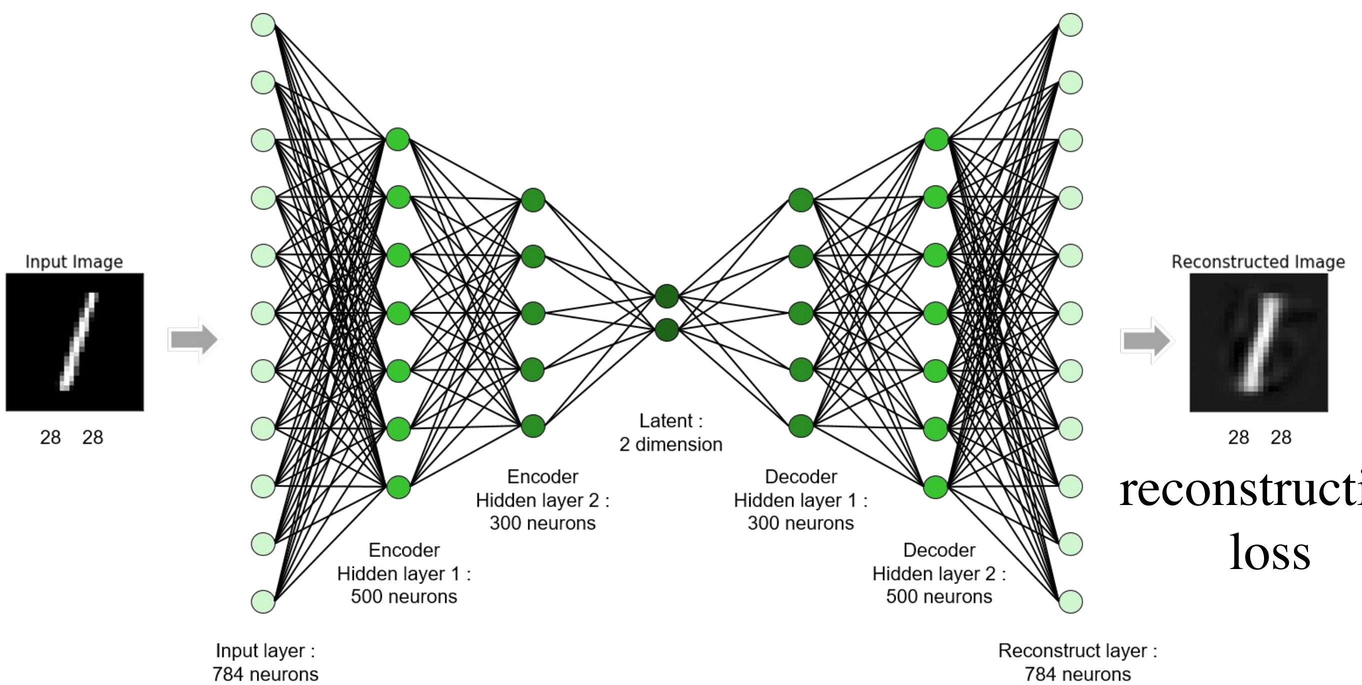
=> unsupervised generation of realistic images

Reinforcement Learning



Autoencoder

=> neural networks in **unsupervised learning setting**



reconstruction loss

dimensionality reduction?

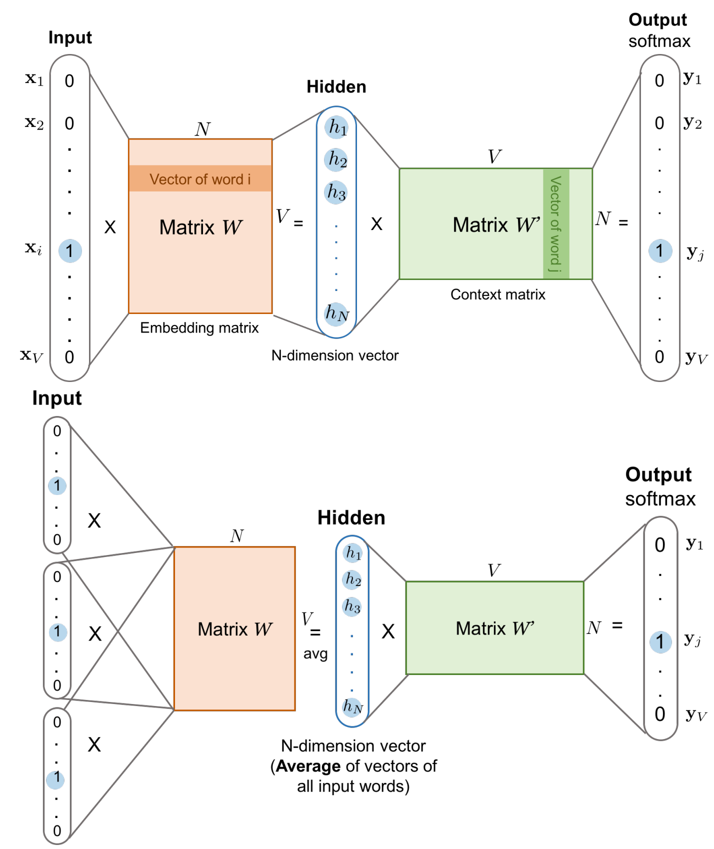
data denoising

=> Word2Vec http://jalammar.github.io/illustrated-word2vec/

> "The man who passes the sentence should swing the sword." – Ned Stark

Sliding window (size = 5)	Target word	Context
[The man who]	the	man, who
[The man who passes]	man	the, who, passes
[The man who passes the]	who	the, man, passes, the
[man who passes the sentence]	passes	man, who, the, sentence
…	…	…
[sentence should swing the sword]	swing	sentence, should, the, sword
[should swing the sword]	the	should, swing, sword
[swing the sword]	sword	swing, the

https://github.com/lesley2958/word2vec/blob/master/word2vec.ipynb

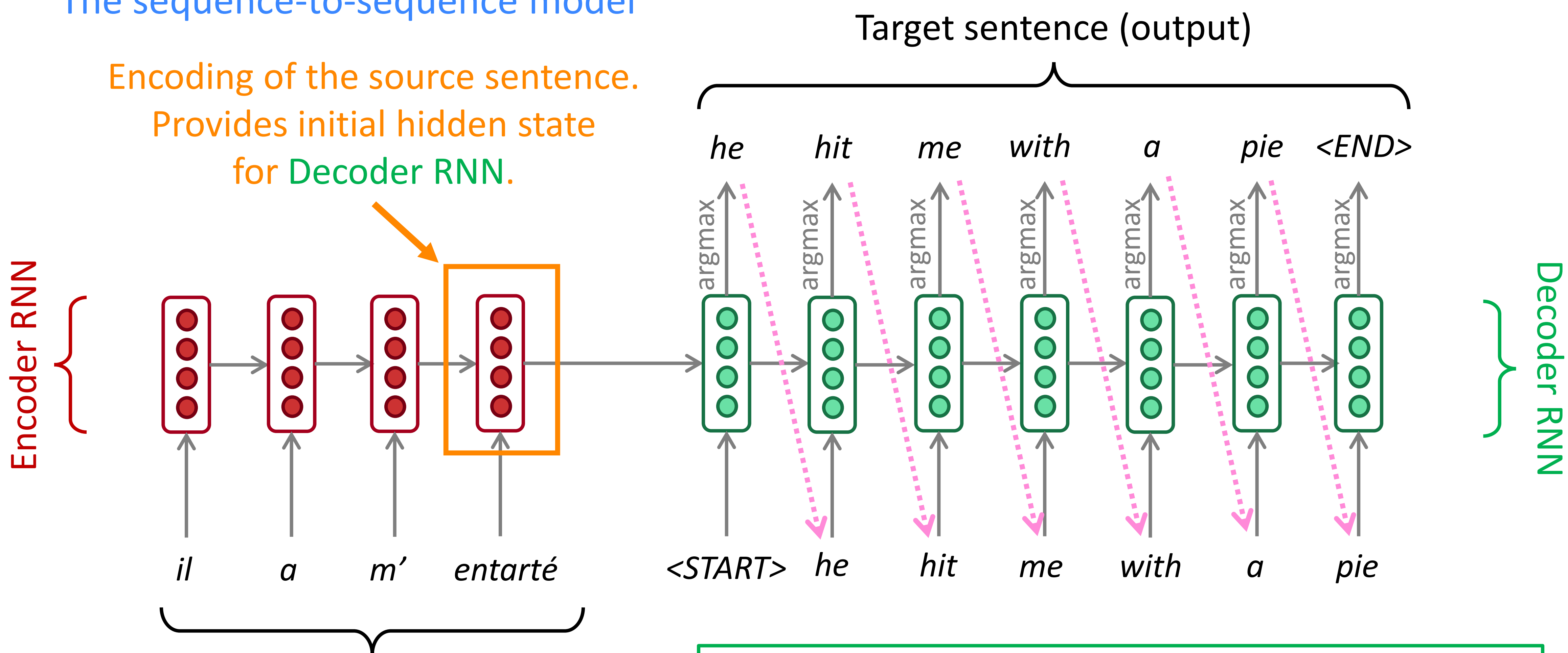


negative sampling,
hierarchical softmax

back-propagating the gradient from the soft-max classifier to the dense word vectors such that the cross entropy loss of the classifier is minimized.

Neural Machine Translation: Encoder-Decoder architecture
- the sequence-to-sequence model

The sequence-to-sequence model



Source sentence (input)
(words are usually represented with word2vec)

Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

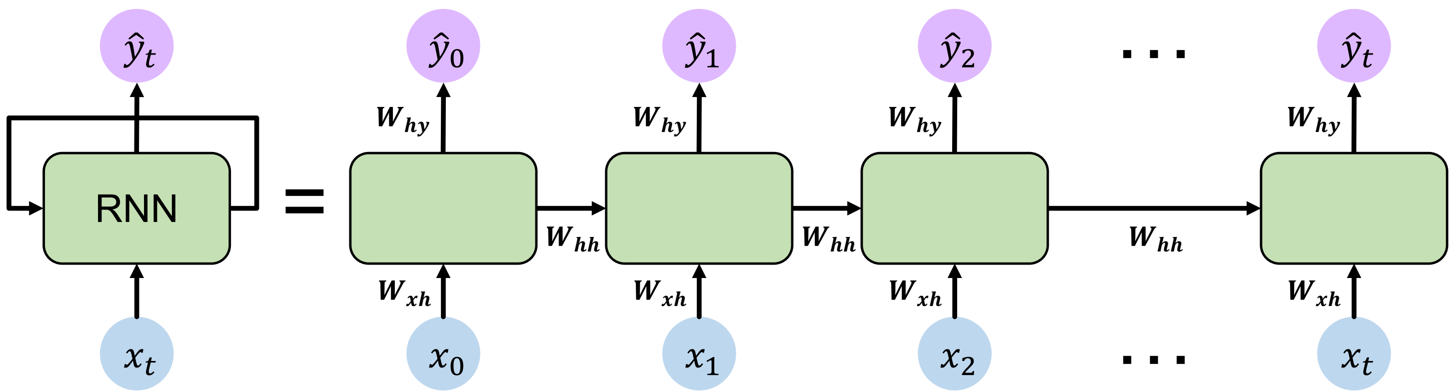
Encoder RNN produces an encoding of the source sentence.

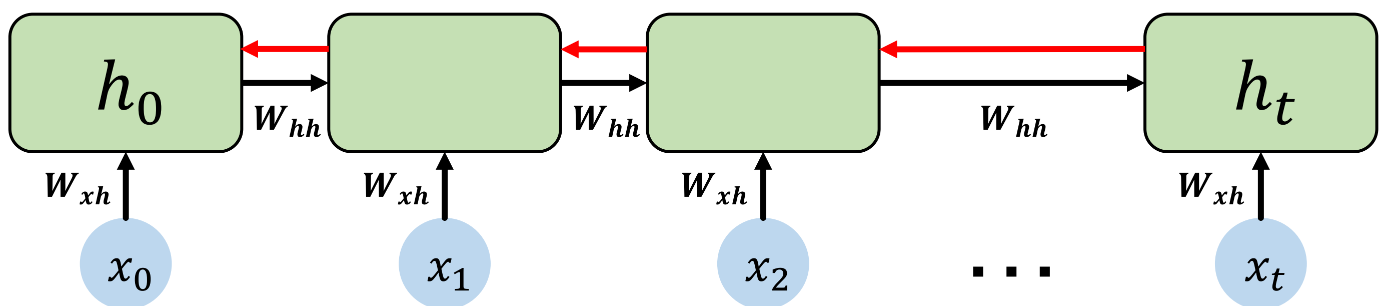
Note: This diagram shows **test time** behavior: decoder output is fed in ·····> as next step’s input

For state of the art see Transformer architecture: http://jalammar.github.io/illustrated-transformer/

Recurrent Neural Network (RNN)

Re-use the **same weight matrices** at every time step





BPTT

Output Vector

$$\hat{y}_t = \boldsymbol{W_{hy}} h_t$$

Update Hidden State

$$h_t = \tanh(\boldsymbol{W_{hh}} h_{t-1} + \boldsymbol{W_{xh}} x_t)$$

Input Vector

$$x_t$$

Computing the gradient wrt h_0 involves **many factors of W_{hh}** (and repeated f'!)

https://datascience-enthusiast.com/DL/Building_a_Recurrent_Neural_Network-Step_by_Step_v1.html

Many values > 1:
exploding gradients

Gradient clipping to scale big gradients

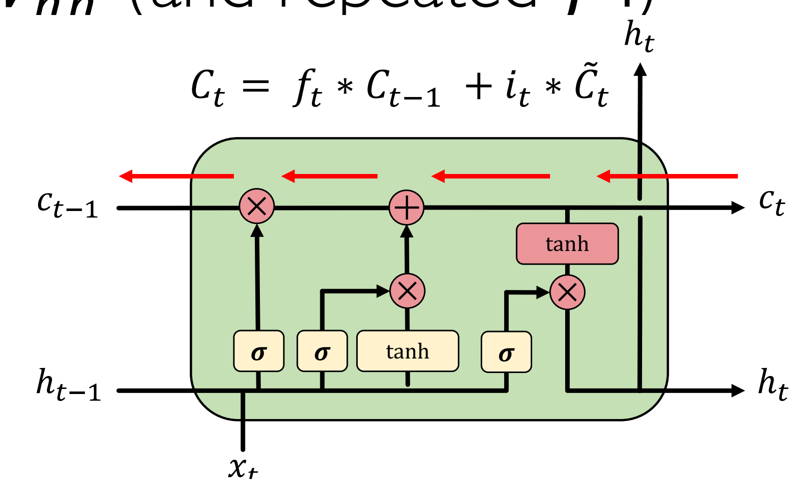
Largest singular value < 1:
vanishing gradients

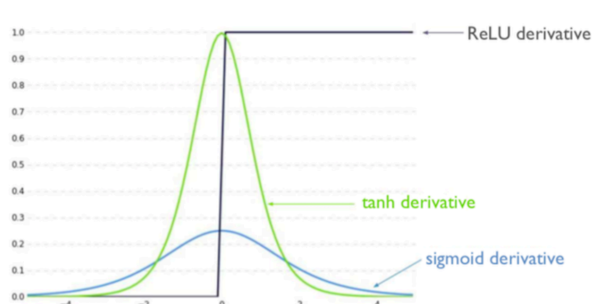
1. **Activation function**
2. **Weight initialization**
3. **Network architecture**

gated cell

LSTM, GRU, etc.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$





Backpropagation from C_t to C_{t-1} requires only elementwise multiplication!
No matrix multiplication → avoid vanishing gradient problem.

Sequence-to-sequence models: Encoder-Network architecture
- many NLP tasks can be modeled as sequence to sequence

Machine Translation: text → translated text

Summarization: long text → short text

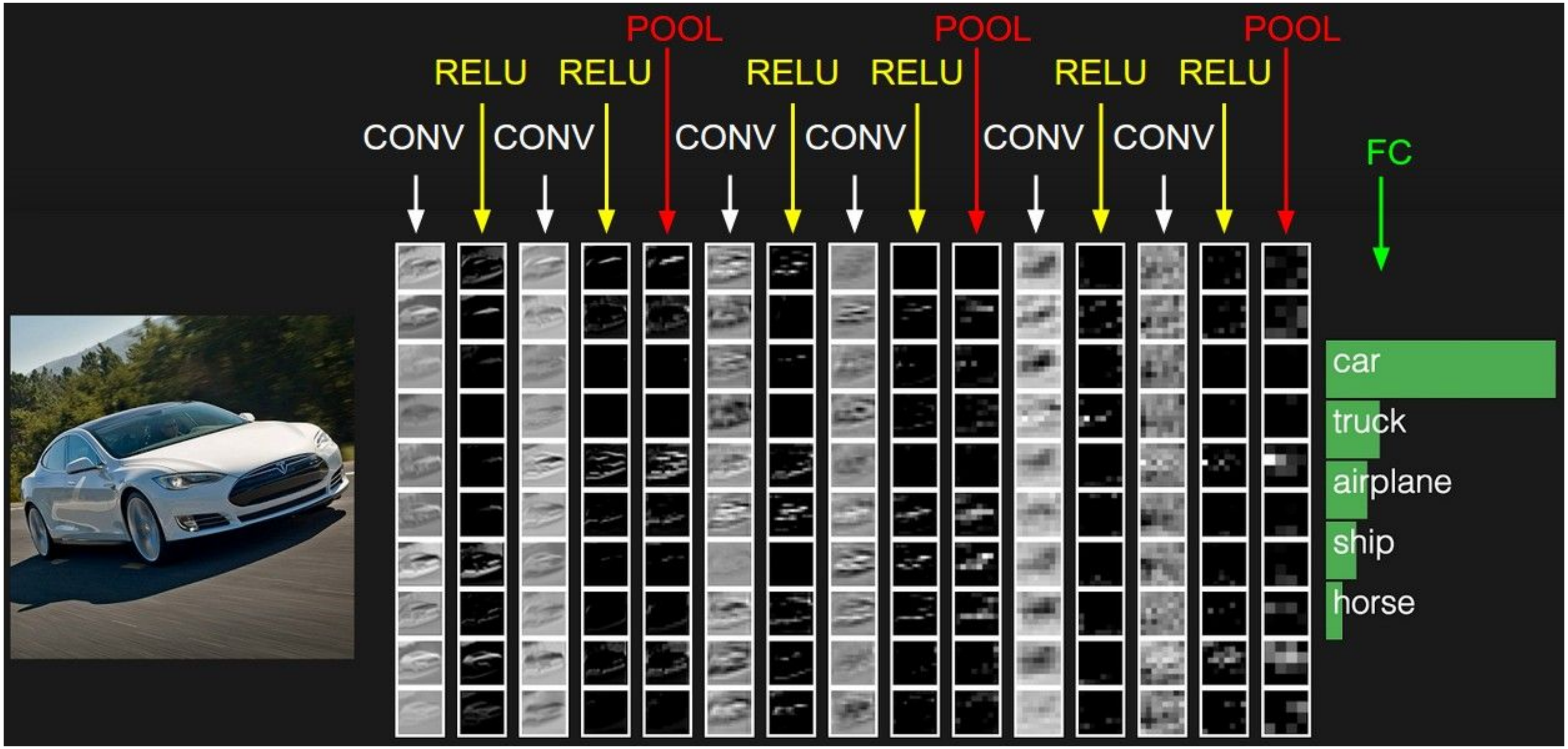
Dialogue (Chatbot): previous utterances → next utterances

Code generation: text in natural language → Python code

DL Library:
- https://github.com/tensorflow/tensor2tensor/#summarization

Convolutional Neural Networks (CNN)

= recall CNN: http://cs231n.github.io/convolutional-networks/



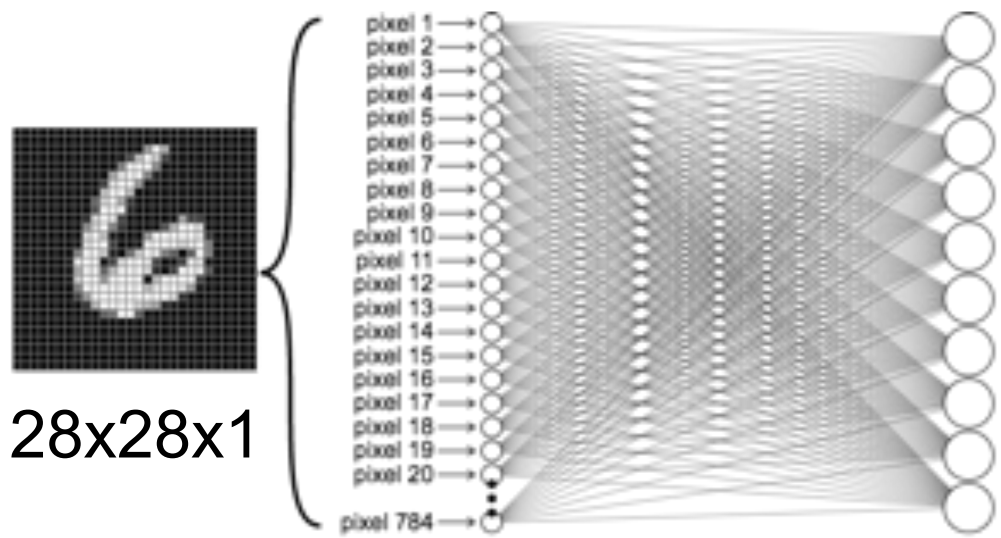
https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

1. CONV: Convolution: Apply filters with learned weights to generate feature maps.
2. RELU: Non-linearity: Often ReLU.
3. POOL: Pooling: Downsampling operation on each feature map
4. FC: Fully connected layer

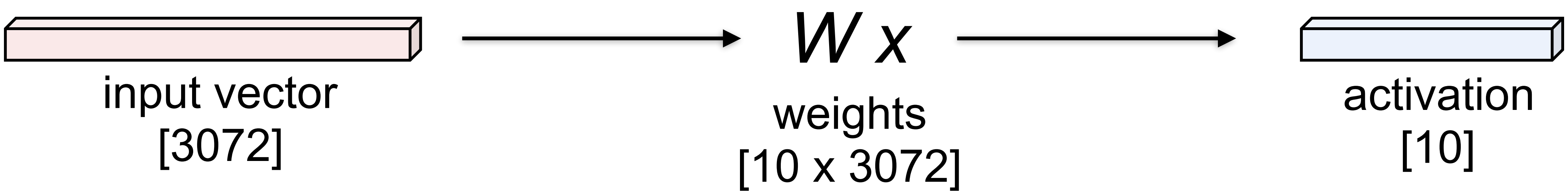
\+
Dropout, Batch/Layer normalization

Fully Connected (FC) Layer

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- Many, many, too many parameters



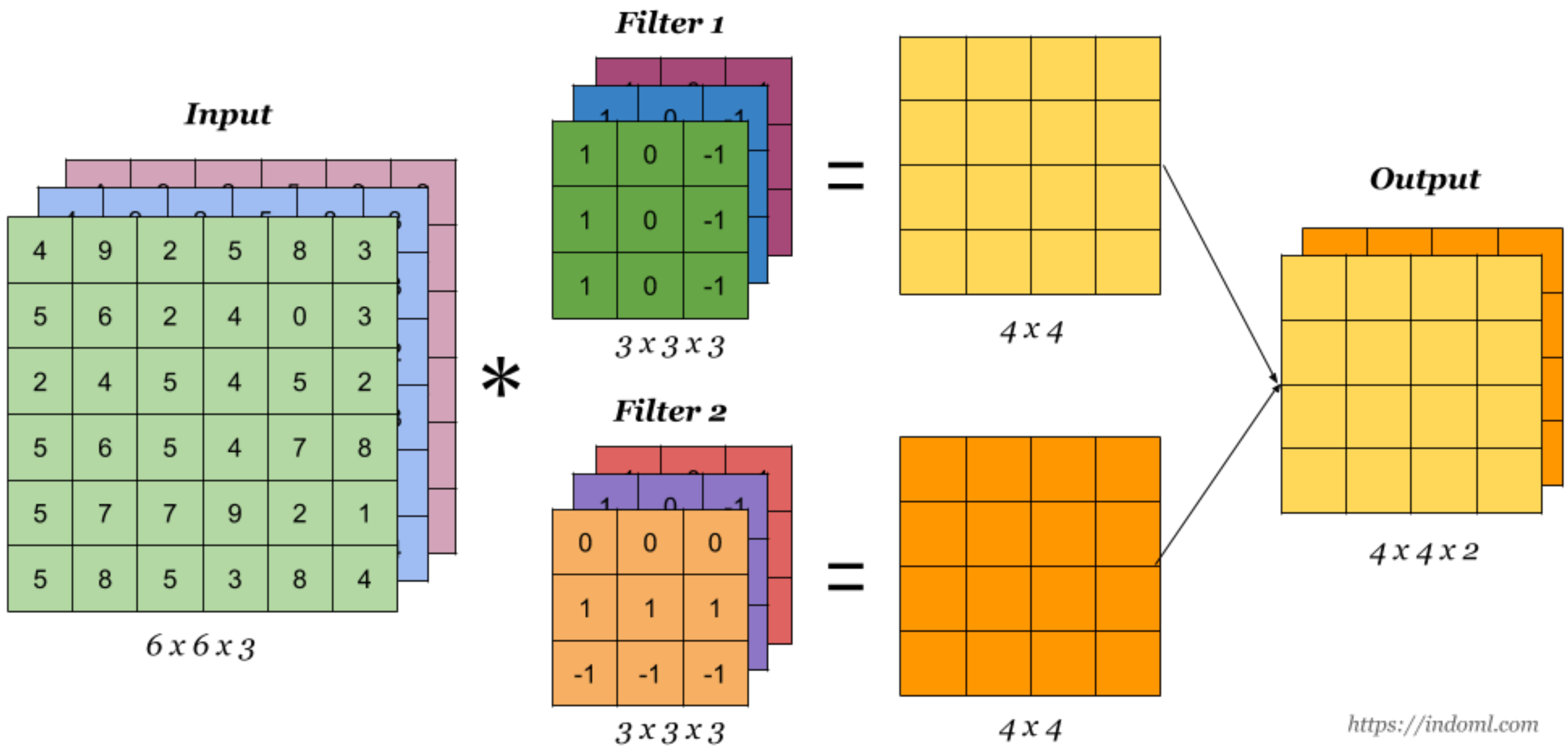
32x32x3 image -> vectorize in 1D array : 3072 x 1



Convolution (CONV) Layer

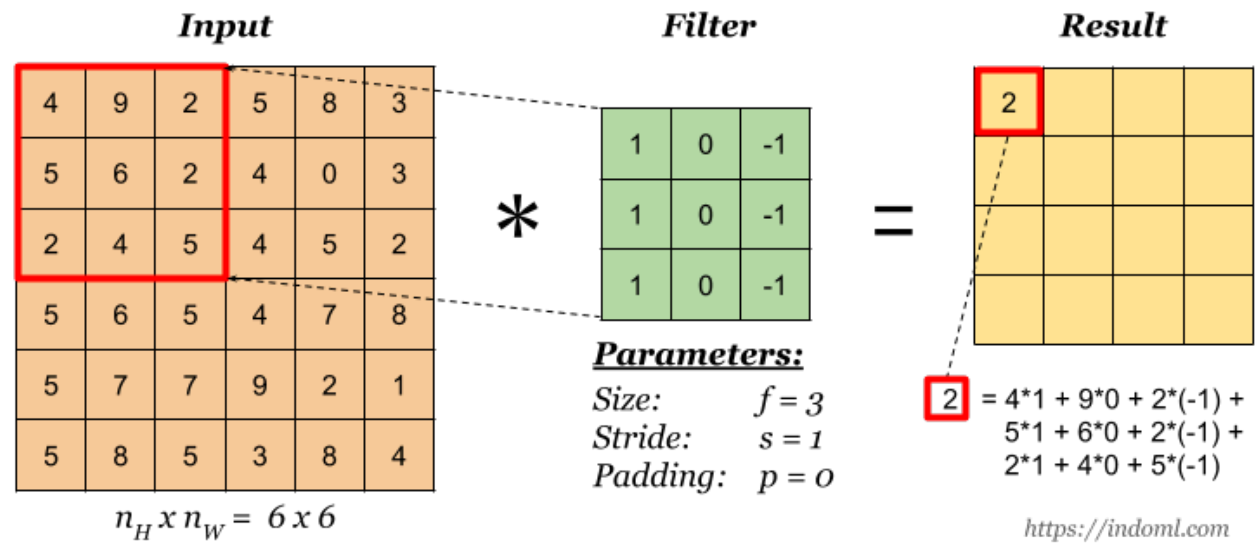
- Use spatial structure of the input connecting patches of the input to neurons in hidden layer
- Applying filters to extract features:

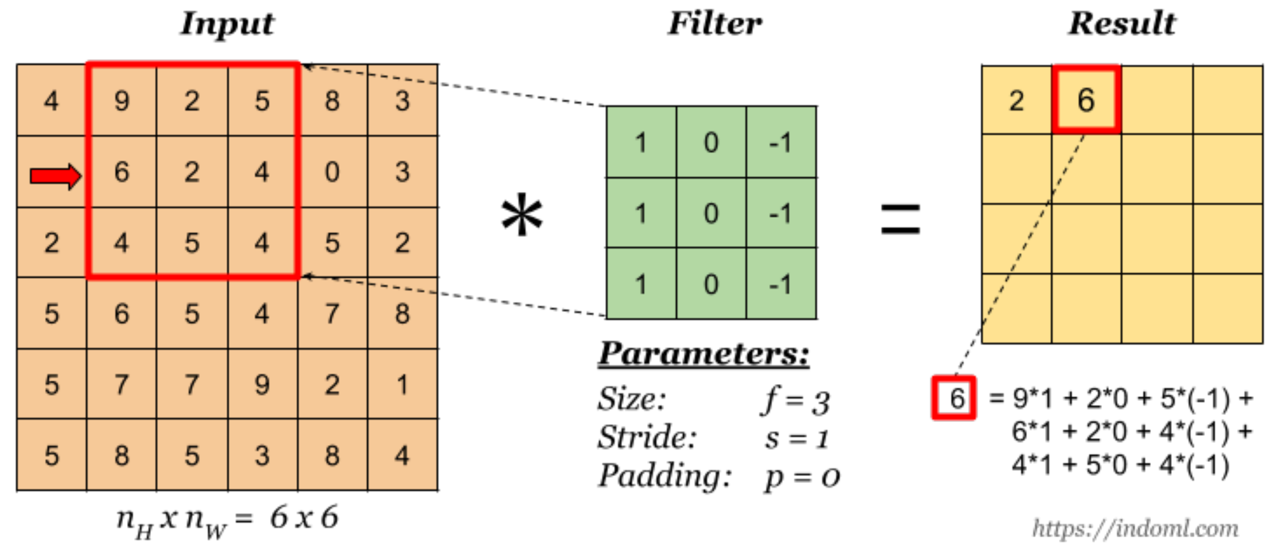
  1. Applying set of weights (filter) to extract **local features**
  2. Use **multiple filters** to extract different features
  3. **Spatially share** parameters of each filter (feature from one part matter elsewhere)



Convolution operation

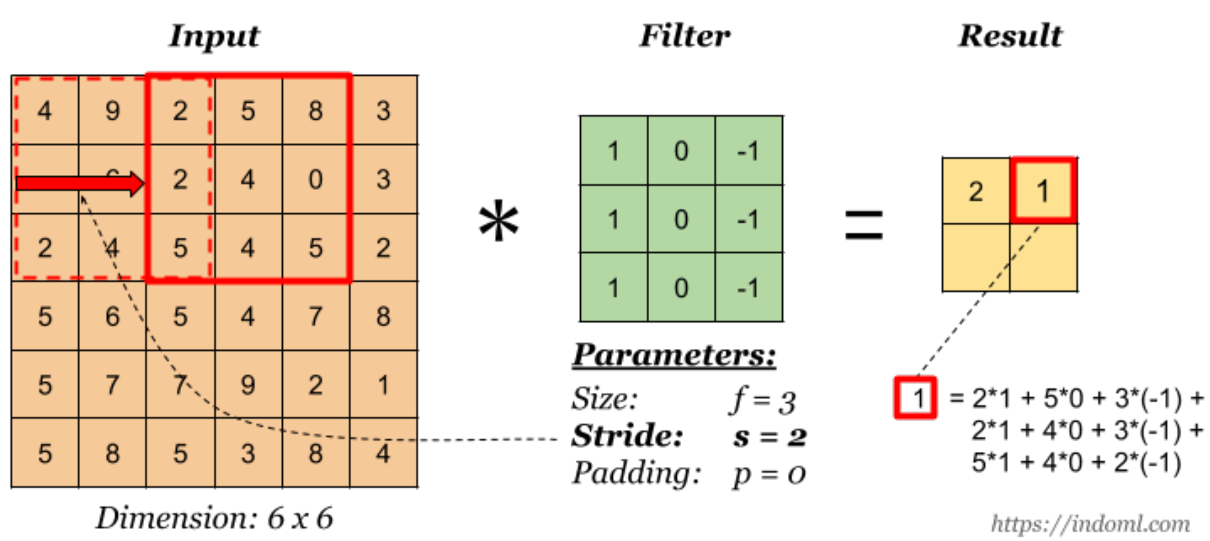
1. Overlay the filter to the input, perform element wise multiplication and add the result
2. Move the filter to the right one position (according to the stride setting)



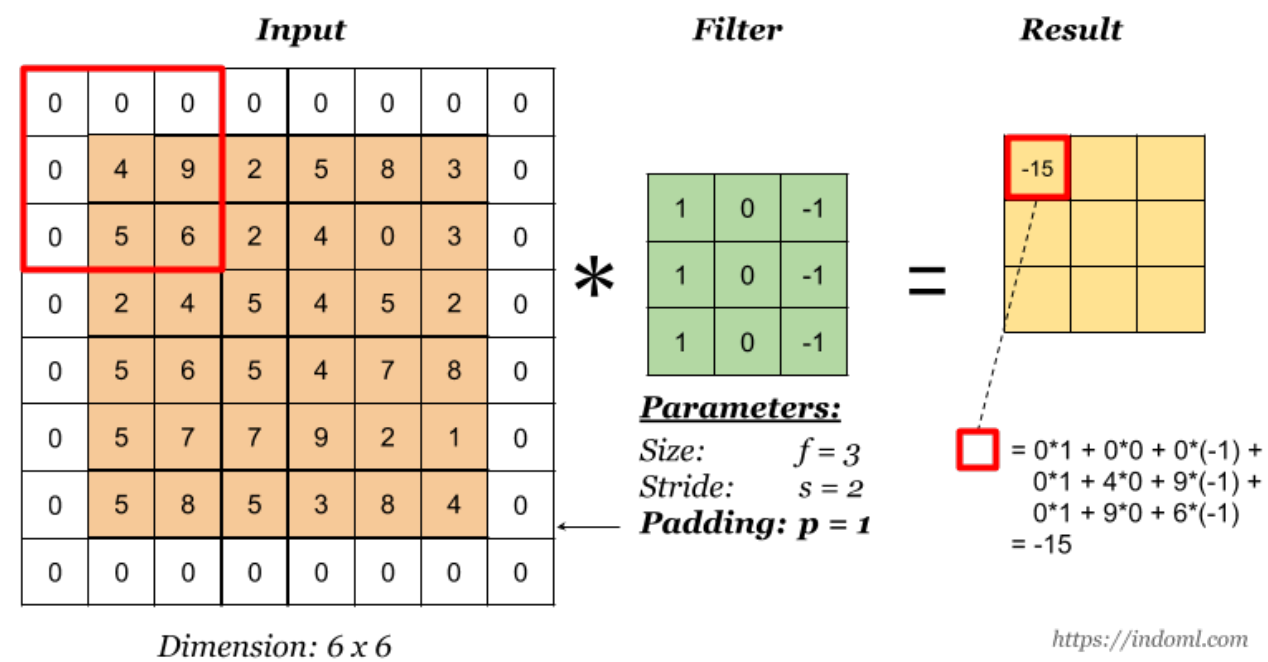


Today number of calculations: (4 x 4) x (3 x 3) = 144.

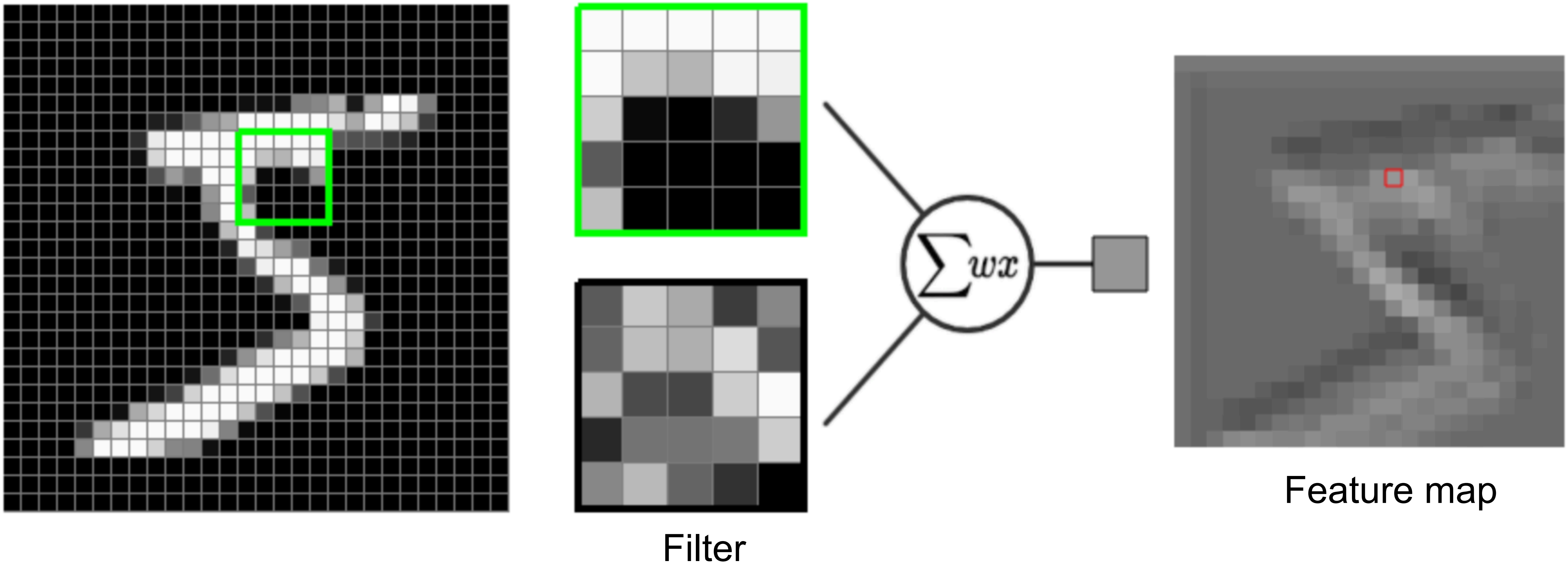
Stride



Padding



DEMO: Convolution operation



CONV operation
=> producing Feature Maps

Original Image

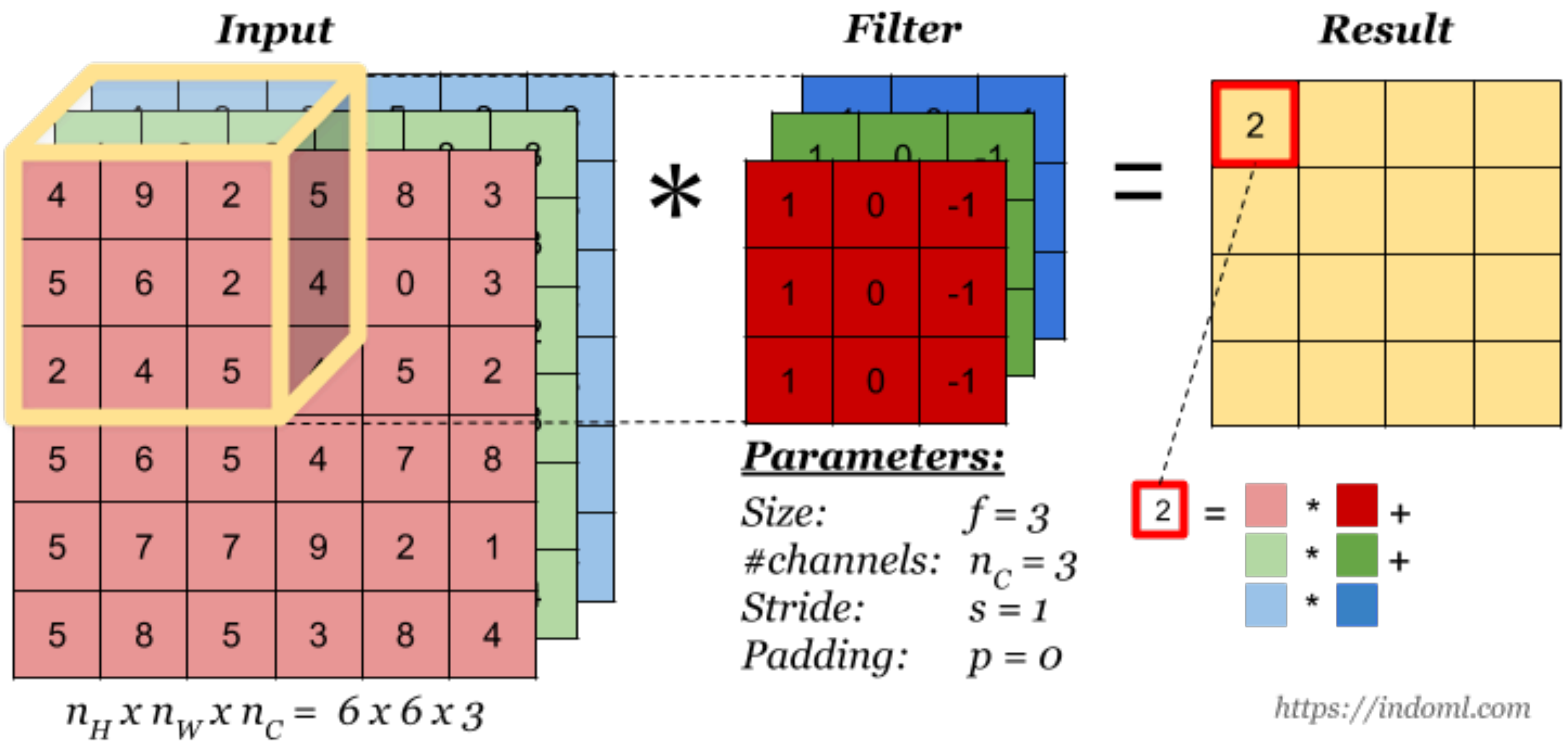
Sharpen

Edge Detect

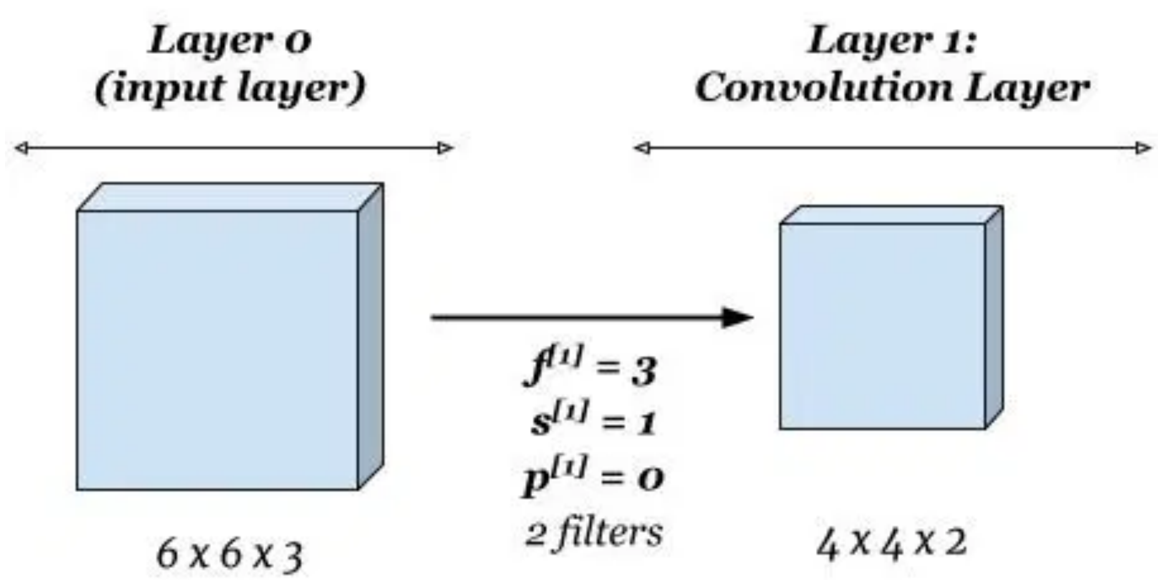
Strong Edge Detect

http://graphicsminer.com/kernel

Convolution operation Volume



The total number of multiplications to calculate the result is (4 x 4) x (3 x 3 x 3) = 432.

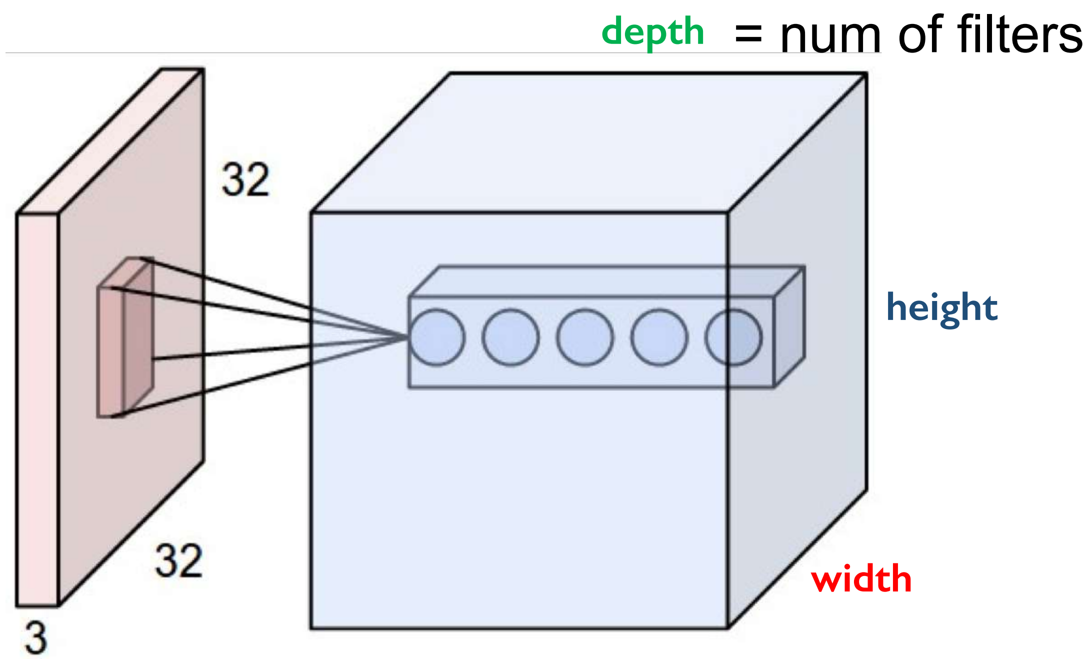


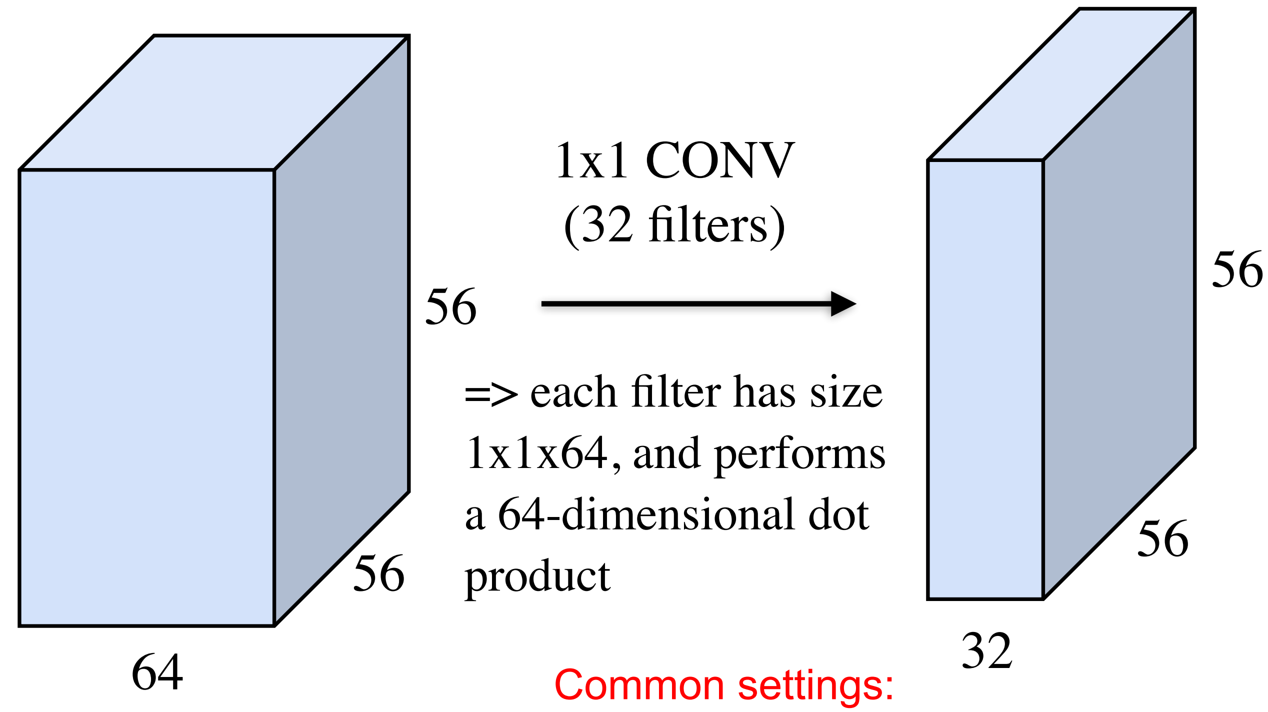
Example of visualization of obtained Feature Maps

Visualization of the Feature Maps Extracted From the First Convolutional Layer in the VGG16 Model

CONV Layer - Example

= recall CNN: http://cs231n.github.io/convolutional-networks/





Input volume: 32x32x3
CONV: 10 5x5 filters with stride 1, pad 2

Output volume size?
(32+2*2-5)/1+1 = 32 spatially, so 32x32x10

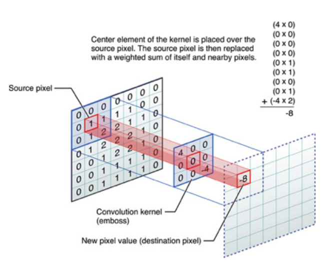
Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params
(+1 for bias) => 76*10 = 760

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0



CONV operation is translation equivariant (not invariant)

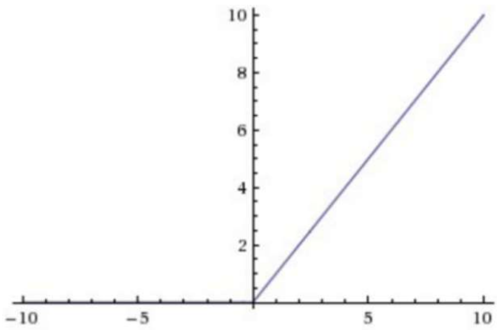
$$T_{\Delta x}(C_k(f)) = C_{T_{\Delta x}k}(f) = C_k(T_{\Delta x}(f))$$

RELU Layer - Example

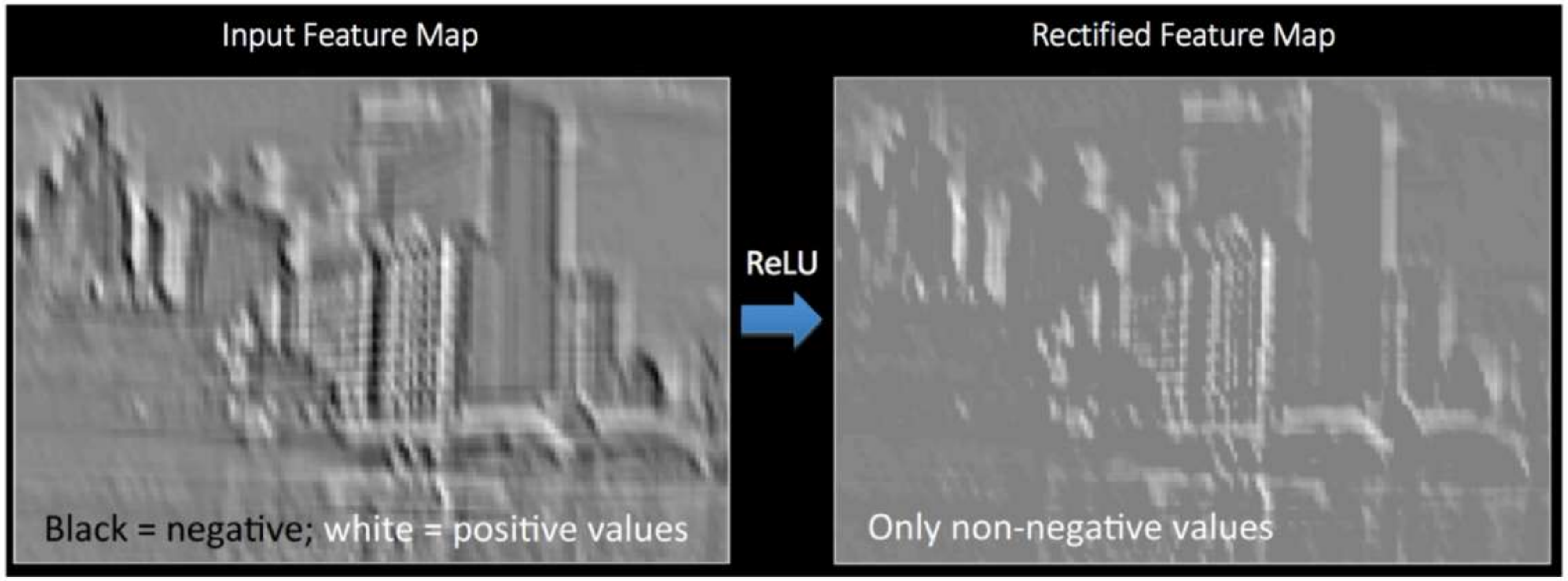
= recall CNN: http://cs231n.github.io/convolutional-networks/

- Apply after every convolution operation (i.e.,after CONV layers)
- Operates over each activation map independently
- It is pixel-by-pixel operation => replaces all negative values to 0

Rectified Linear Unit (ReLU)



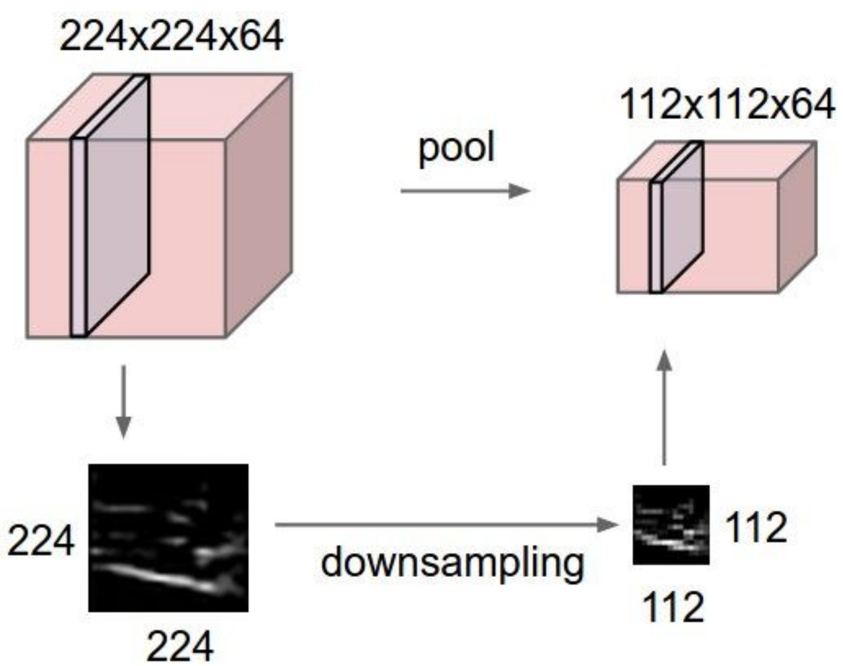
$g(z) = \max(0, z)$



POOL Layer - Example MAX POOLING

= recall CNN: http://cs231n.github.io/convolutional-networks/

- Makes the representations smaller and more manageable
- Operates over each activation map independently



- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

F = 2, S = 2
F = 3, S = 2

Single depth slice

x

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

y

max pool with 2x2 filters and stride 2

6	8
3	4

=> Are CNN translation invariant? What about scale and rotation invariance?

CNN Architectures

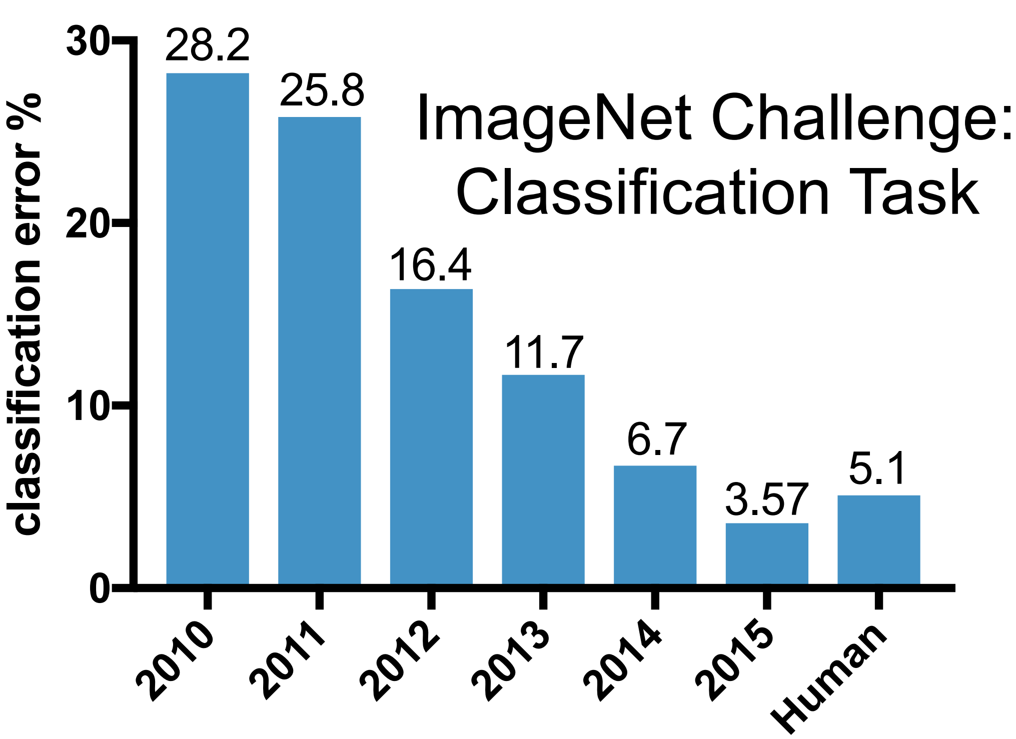
2012: AlexNet. First CNN to win.
- 8 layers, 61 million parameters

2013: ZFNet
- 8 layers, more filters

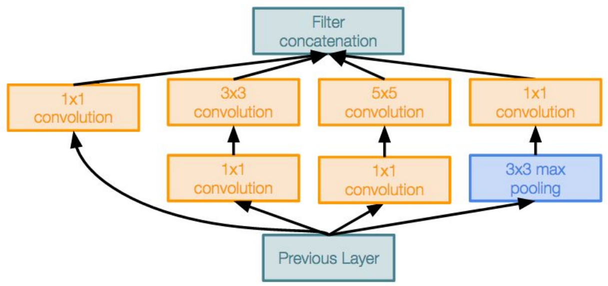
2014: VGG
- 19 layers

2014: GoogLeNet
- "Inception" modules
- 22 layers, 5million parameters

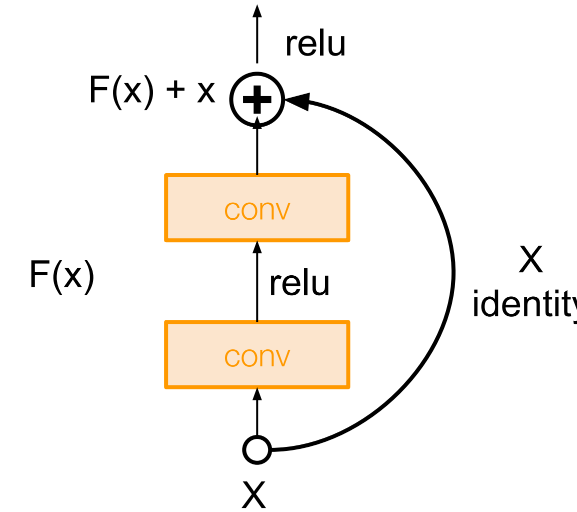
2015: ResNet
- 152 layers



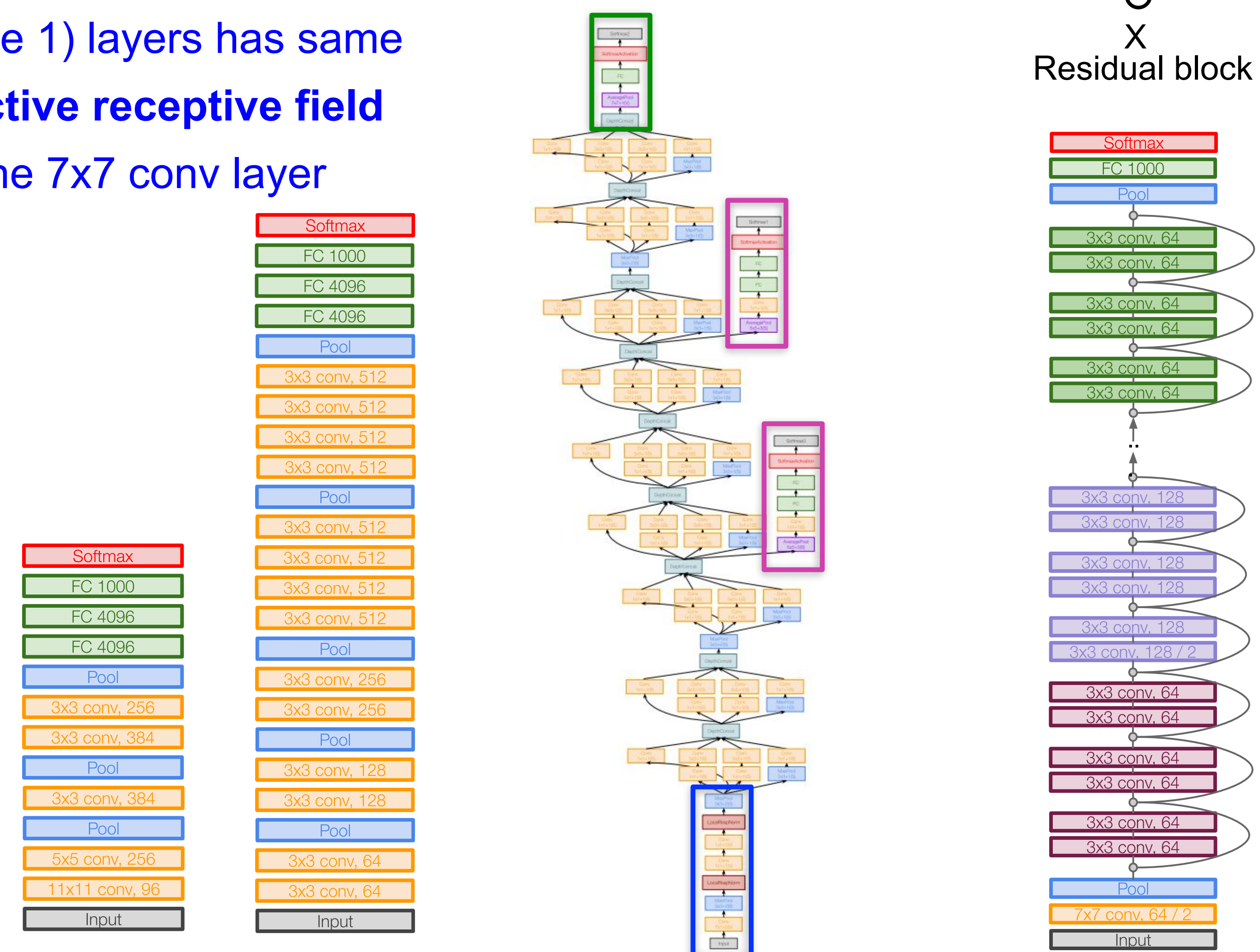
Stack of three 3x3 cor..
(stride 1) layers has same
effective receptive field
as one 7x7 conv layer



Inception module



Residual block



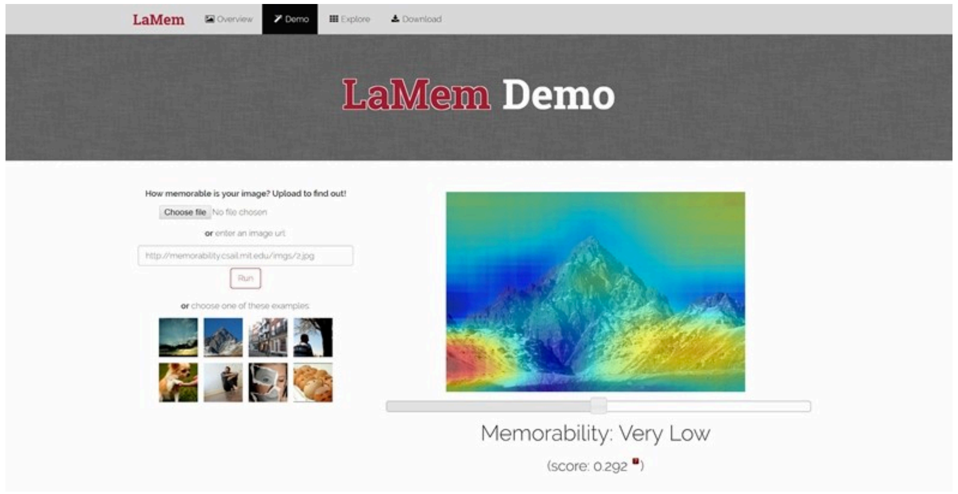
AlexNet VGG19 GoogLeNet ResNet

http://cs231n.stanford.edu

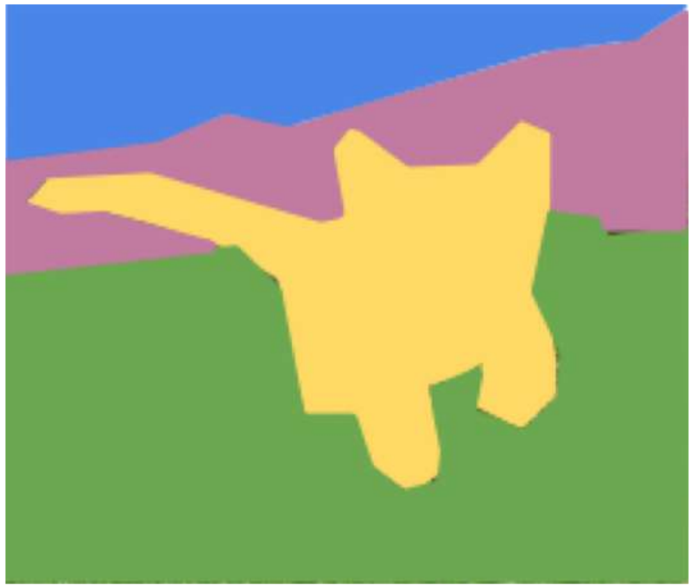
CNN are ubiquitous:
=> workhorse for Computer Vision applications

ImageNet Challenge: Classification Task

Image memorability score



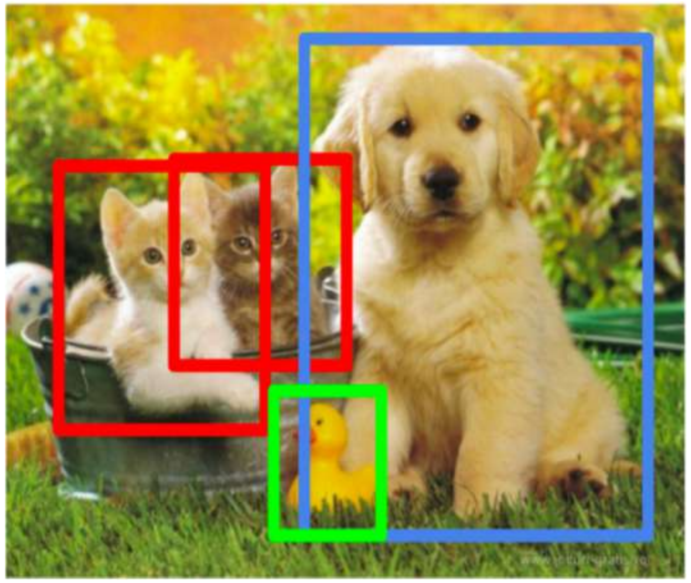
Semantic Segmentation



CAT

Fully Convolutional Networks (FCN)

Object Detection



CAT, DOG, DUCK

[Fast, Faster] R-CNN + YOLO v1,2,3

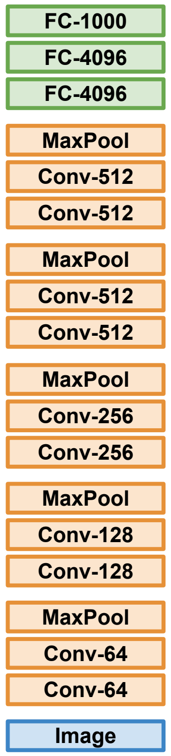
Image Captioning

The cat is in the grass.

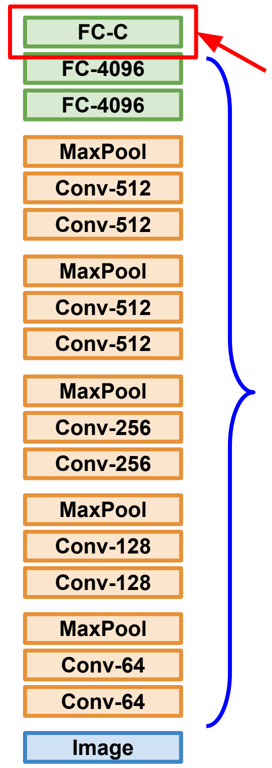
CNN + RNN

Transfer learning with CNN

1. Train on Imagenet



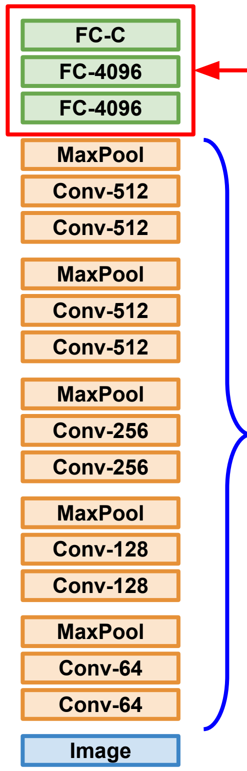
2. Small Dataset (C classes)



Reinitialize this and train

Freeze these

3. Bigger dataset



Train these

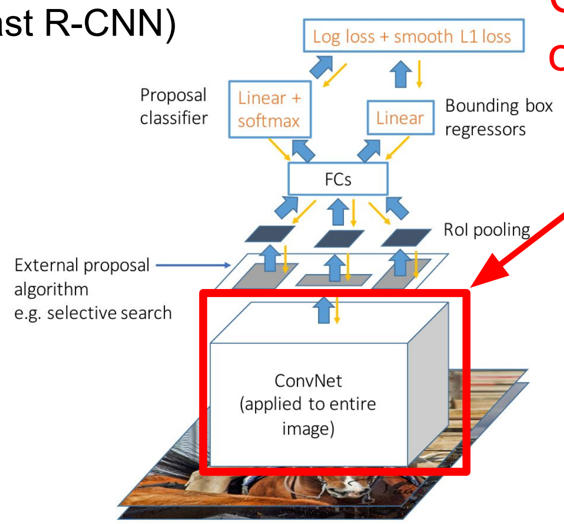
With bigger dataset, train more layers

Freeze these

Lower learning rate when finetuning; 1/10 of original LR is good starting point

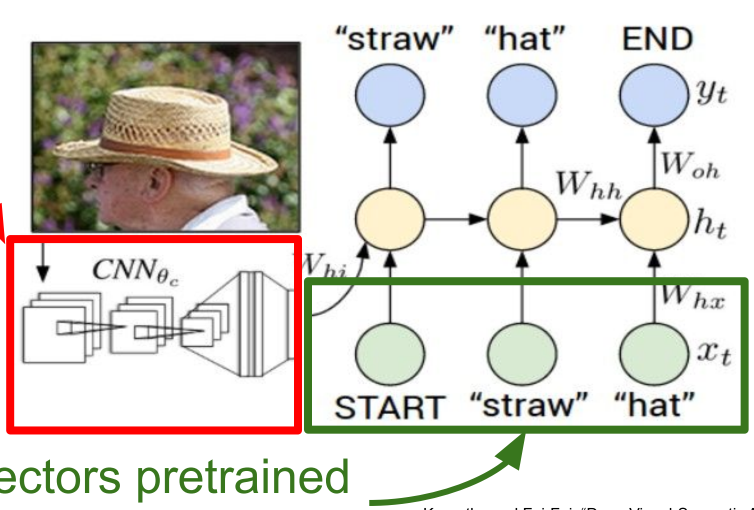
	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Object Detection (Fast R-CNN)



CNN pretrained on ImageNet

Image Captioning: CNN + RNN

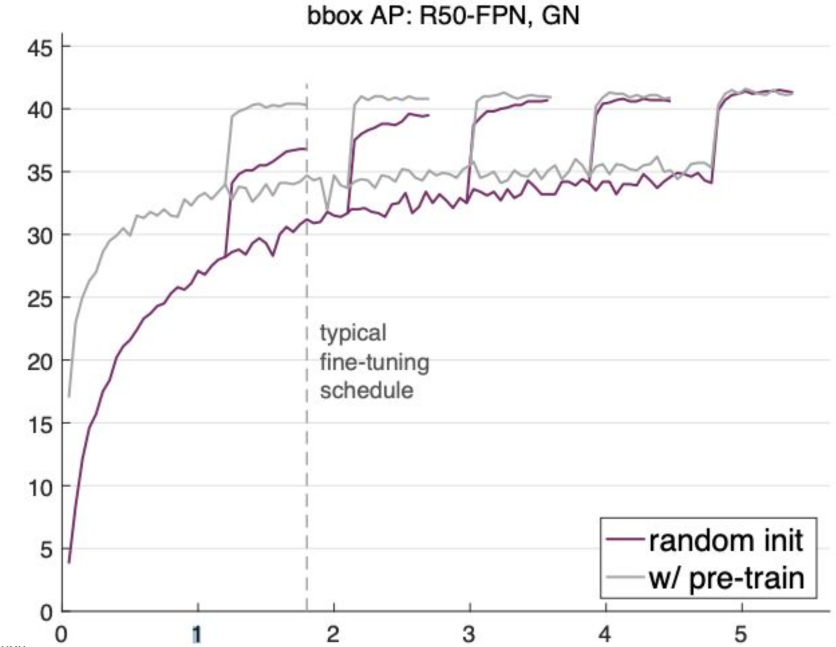


Word vectors pretrained with word2vec

:, "Fast R-CNN", ICCV 2015
:opyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational pur…

however, not always necessary



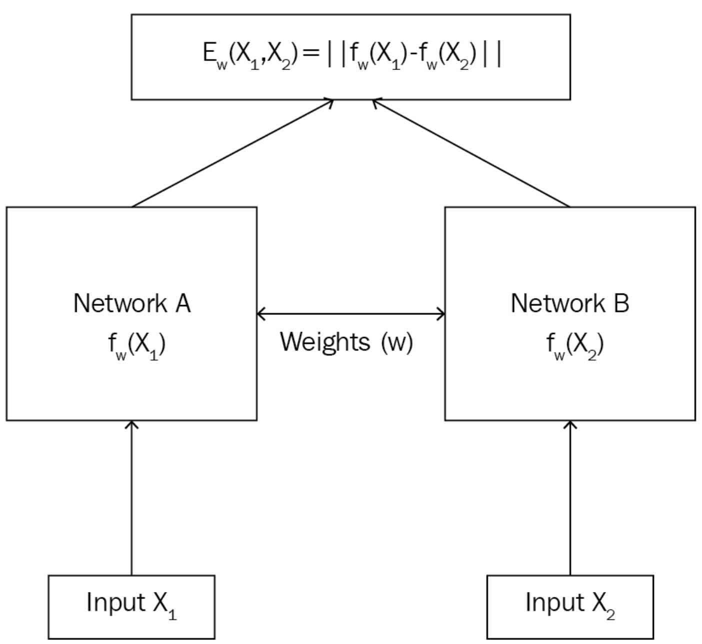
He et al, "Rethinking ImageNet Pre-training", 2018

Face recognition - CNN Siamese network + One-shot learning

- Learn image representations with siamese NN (learning ‘similarity’ function)
- Reuse features from the network for one-shot learning

Deep Face Recognition: A Survey, 2018.

Siamese network



One-shot Learning

https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf

In the case of one-shot learning, a single exemplar of an object class is presented to the algorithm.

*This should be distinguished from **zero-shot learning**, in which the model cannot look at any examples from the target classes.*

"Dimensionality Reduction by Learning an Invariant Mapping", 2006.

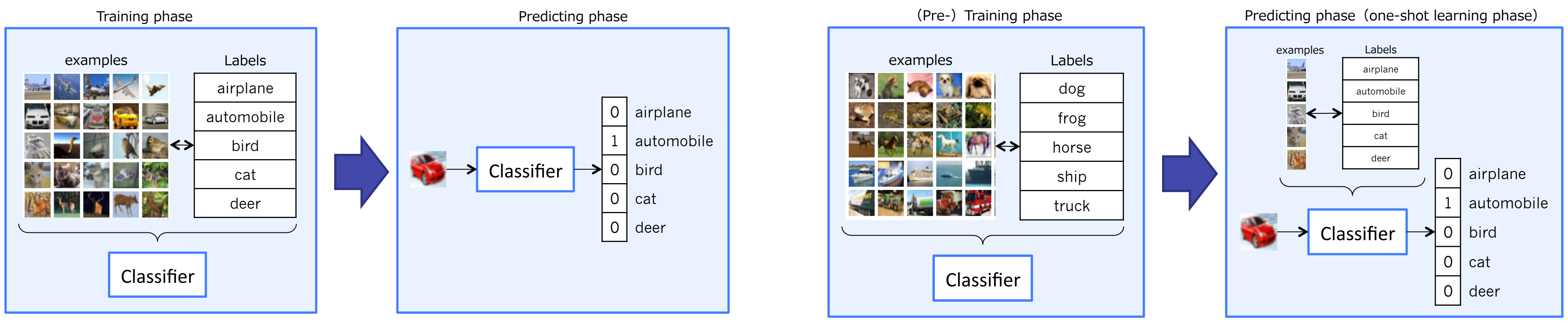
*The **contrastive loss** requires face image pairs and then pulls together positive pairs and pushes apart negative pairs.*

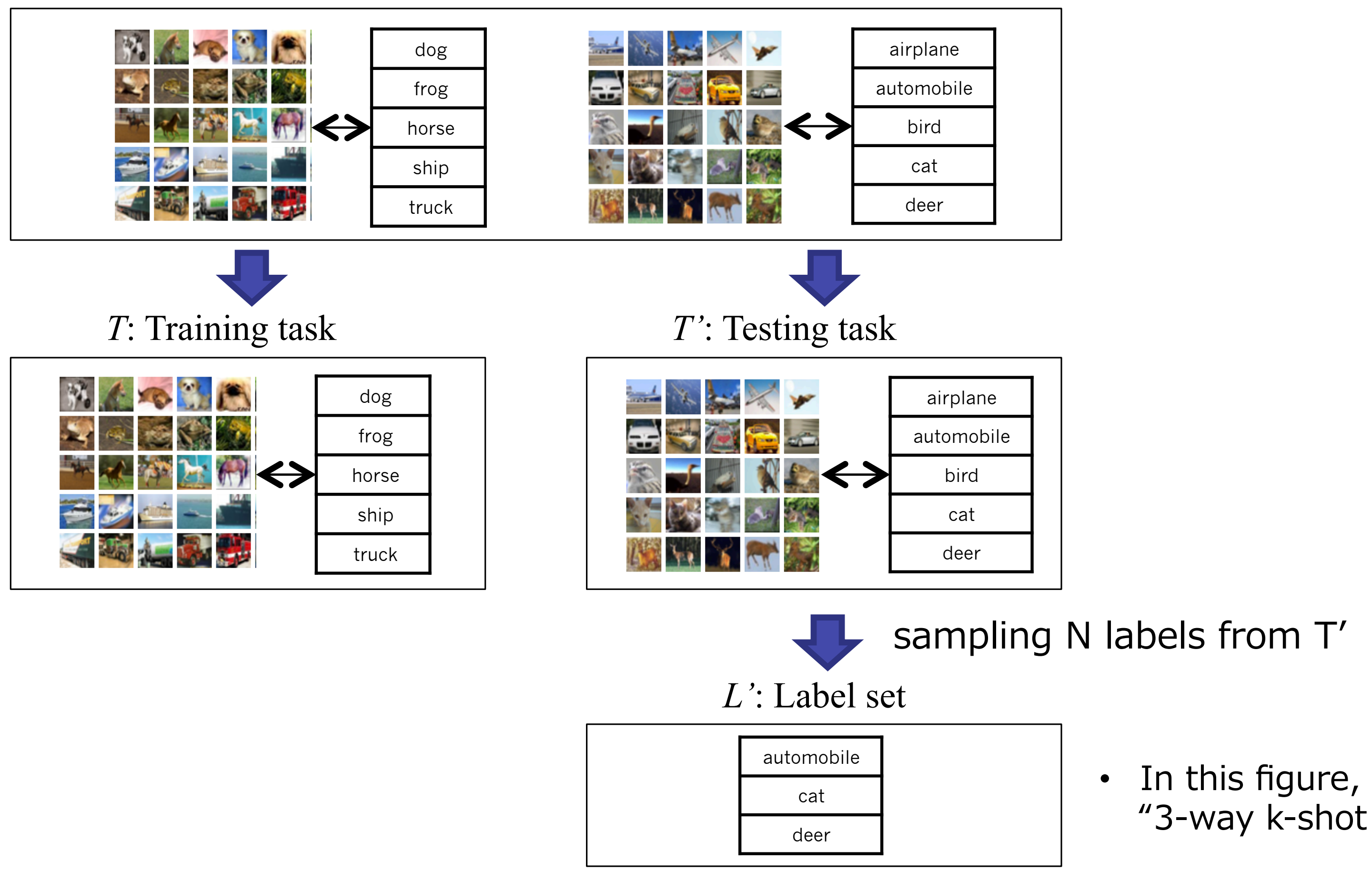
"FaceNet: A Unified Embedding for Face Recognition and Clustering.", 2015

The **Triplet loss** involves an anchor example and one positive or matching example (same class) and one negative or non-matching example (differing class). The loss function penalizes the model such that the distance between the matching examples is reduced and the distance between the non-matching examples is increased.

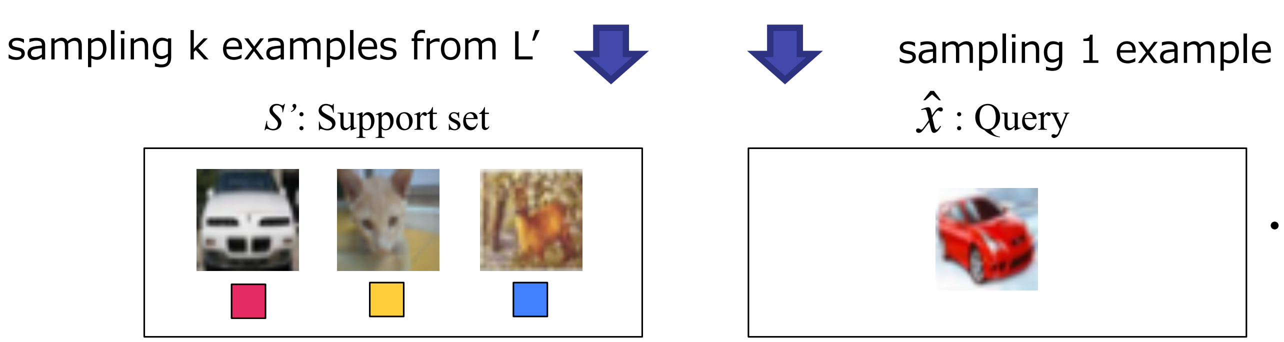
It is crucial to select hard triplets, that are active and can therefore contribute to improving the model. (inspired by curriculum learning)

Task: N-way k-shot learning



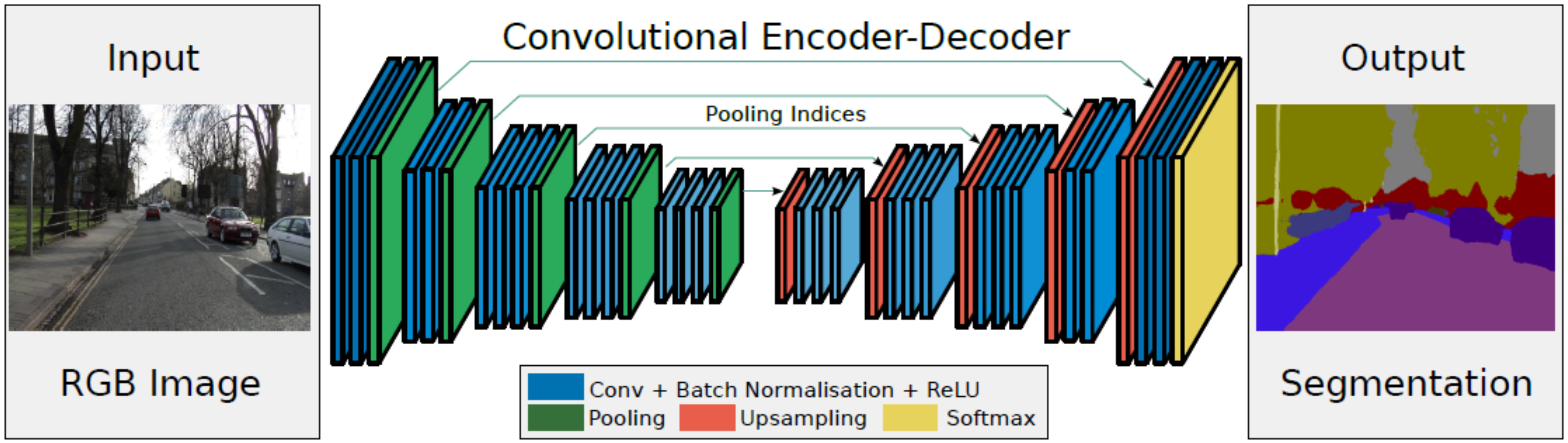


- In this figure, L' has 3 classes, thus "3-way k-shot learning"



- Task: classify $\hat{x}$ into 3 classes, {automobile, cat, deer}, using support set

Semantic segmentation: Encoder-Decoder Network architecture
- SegNet, DeconvNet, U-Net

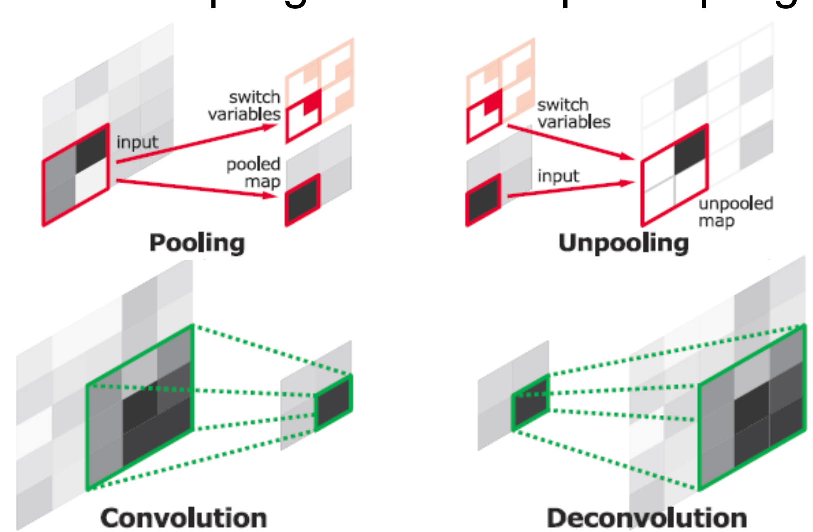


Encoder:
- takes input image and generates feature vector
- aggregate features at multiple levels

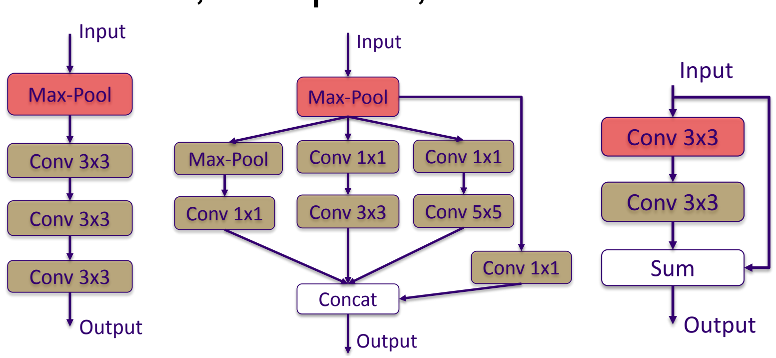
Decoder:
- takes feature vector and generates segmentation map
- decode features aggregated by encoder

Basic Building Blocks:

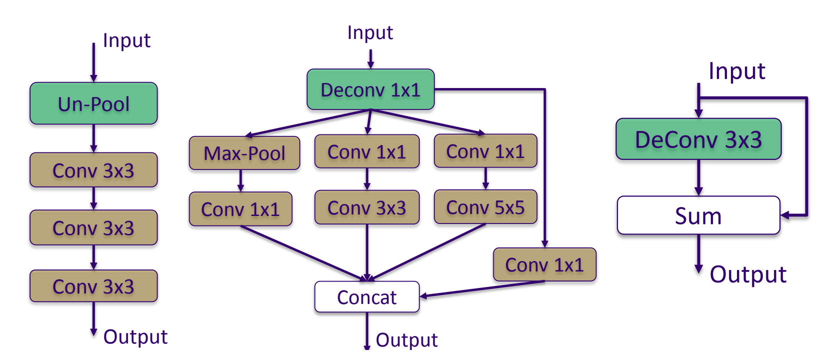
Down Sampling   Up Sampling



Encoding Block Types:
- VGG, Inception, ResNet



Decoding Block Types:
- VGG, Inception, ResNet



U-Net Example: https://github.com/hspitzer/histo-seg/blob/master/tutorial-segmentation.ipynb

Acknowledgement: List of basic materials

Ian Goodfellow, Yoshua Bengio and Aaron Courville, MIT Deep Learning Book:

- https://github.com/janishar/mit-deep-learning-book-pdf

Courses:

- Introduction to Deep Learning, MIT S191
  - Good materials for basic (easy/soft) introduction to DL models
  - We did not cover:
    - Deep Generative Models (**Variational Autoencoders** + **Generative Adversarial Networks**)
    - **Deep Reinforcement Learning**

- Stanford CS231n, http://cs231n.stanford.edu/
  - Good introduction to deep learning for Computer Vision application
  - Must read lecture notes: http://cs231n.github.io/ (Neural networks, Convolutional Neural Networks)

- Stanford CS224n, https://web.stanford.edu/class/cs224n/
  - Good introduction to deep learning for NLP applications
  - Notes for word2vec, seq2seq models

- Deep Learning, MIT, https://deeplearning.mit.edu/
  - Deep Learning - State of the Art, 2018
  - List of some more advanced DL topics

- Representation Learning, Mila IFT 6135
  - https://sites.google.com/mila.quebec/ift6135/lectures?authuser=0
  - Attention, Self-Attention and Transformers

DL models: Limitations

- Very **data hungry** (e.g. often minion of examples)
- **Computationally intensive** to train and deeply (requires GPU)

- Easily fooled by **adversarial examples**
- Can be subject to **algorithmic bias**

- Poor representing uncertainty (how do you know what the model knows?)
- Uninterpretable black boxes, difficult to trust

- Finicky to optimize: non-convex, choice of architecture, learning parameters
- Often require expert knowledge to design, fine tune architectures

Additional DL Topics: Materials

https://tinyurl.com/y5h9oojn