

<http://www.digicortex.net/node/17>

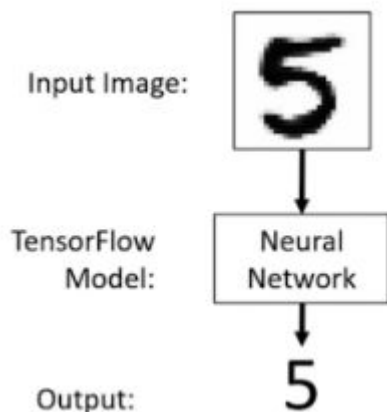
# Deep Learning Intro

---

Jay Urbain, PhD

Professor, Department of Electrical Engineering and Computer Science  
Milwaukee School of Engineering

# Tools tf.Keras



1

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

2

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

3

```
# 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

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

5

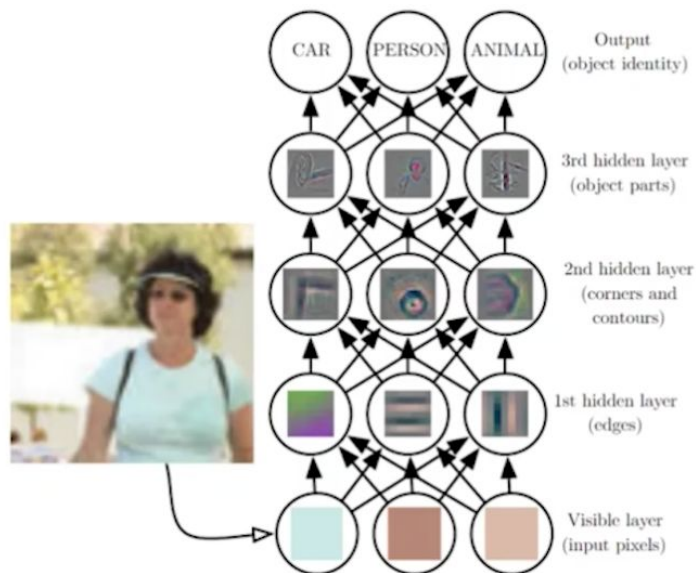
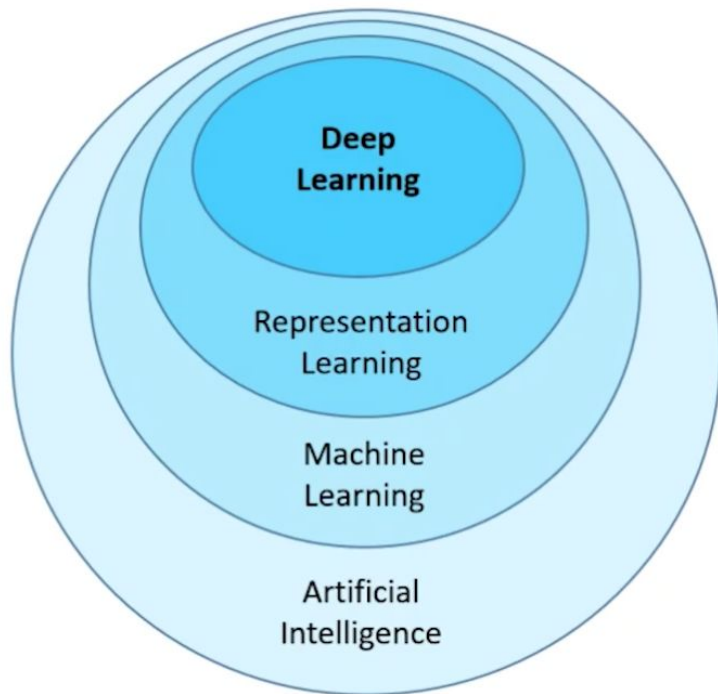
```
# evaluate
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test accuracy:', test_acc)
```

6

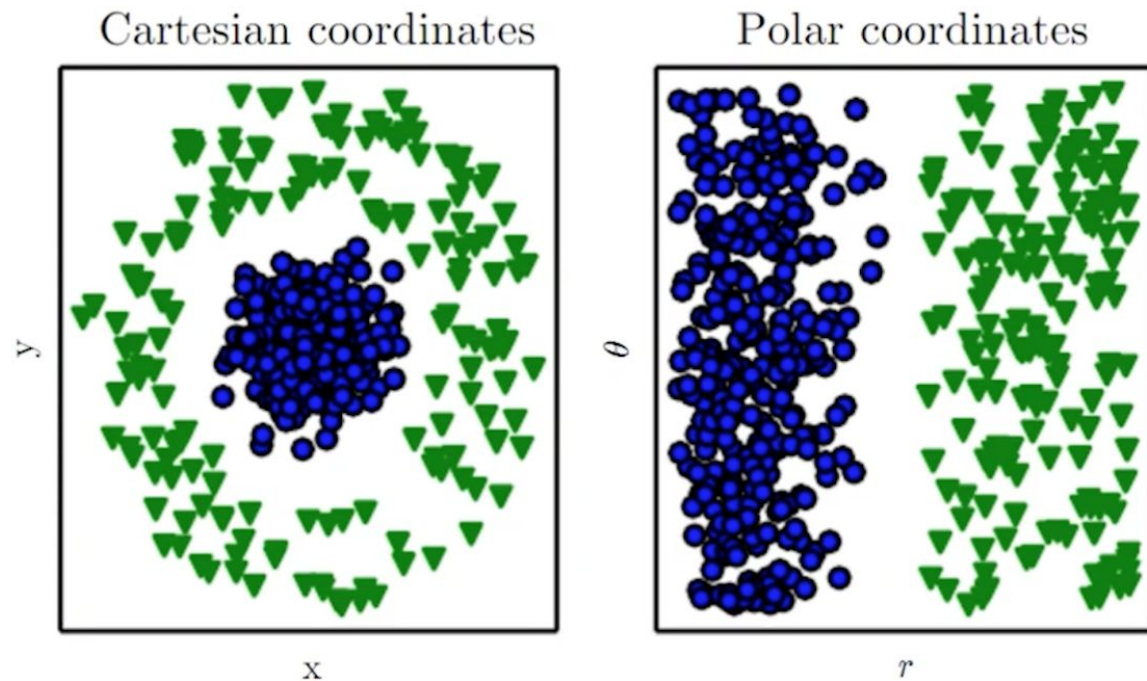
```
# make predictions
predictions = model.predict(test_images)
```

# Deep Learning is **Representation Learning**

(aka Feature Learning)



# Representation Matters



**Task:** Draw a line to separate the **green triangles** and **blue circles**.

# Deep learning = Learning representations/features

- The traditional model of pattern recognition (since the late 50's)
  - Fixed/engineered features (or fixed kernel) + trainable classifier



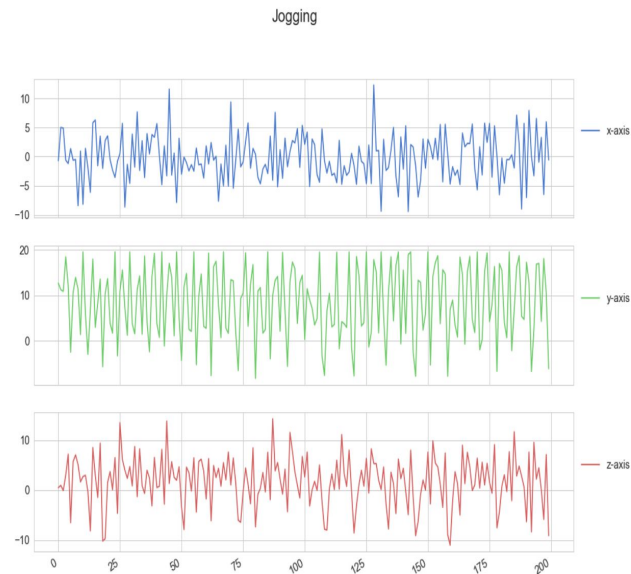
- End-to-end learning / Feature learning / Deep learning
  - Trainable features (or kernel) + trainable classifier



# Sequence classification: activity recognition

## Temporal sensor analysis and activity tracking

	user	activity	timestamp	x-axis	y-axis	z-axis
0	33	Jogging	49105962326000	-0.694638	12.680544	0.503953
1	33	Jogging	49106062271000	5.012288	11.264028	0.953424
2	33	Jogging	49106112167000	4.903325	10.882658	-0.081722
3	33	Jogging	49106222305000	-0.612916	18.496431	3.023717
4	33	Jogging	49106332290000	-1.184970	12.108489	7.205164





# Traditional models: activity recognition

[http://www.cis.fordham.edu/wisdm/public\\_files/sensorKDD-2010.pdf](http://www.cis.fordham.edu/wisdm/public_files/sensorKDD-2010.pdf)

Average[3]: Average acceleration (for each axis)

Standard Deviation[3]: Standard deviation (for each axis)

Average Absolute Difference[3]: Average absolute difference between the value of each of the 200 readings within the ED and the mean value over those 200 values (for each axis)

Average Resultant Acceleration[1]: Average of the square roots of the sum of the values of each axis squared  $\sqrt{(x_i^2 + y_i^2 + z_i^2)}$  over the ED

Time Between Peaks[3]: Time in milliseconds between peaks in the sinusoidal waves associated with most activities (for each axis)

Binned Distribution[30]: We determine the range of values for each axis (maximum – minimum), divide this range into 10 equal sized bins, and then record what fraction of the 200 values fell within each of the bins.

Table 2: Accuracies of Activity Recognition

	% of Records Correctly Predicted			
	J48	Logistic Regression	Multilayer Perceptron	Straw Man
Walking	89.9	<b>93.6</b>	91.7	37.2
Jogging	96.5	98.0	<b>98.3</b>	29.2
Upstairs	59.3	27.5	<b>61.5</b>	12.2
Downstairs	<b>55.5</b>	12.3	44.3	10.0
Sitting	<b>95.7</b>	92.2	95.0	6.4
Standing	<b>93.3</b>	87.0	91.9	5.0
Overall	85.1	78.1	<b>91.7</b>	37.2

	user	activity	timestamp	x-axis	y-axis	z-axis
0	33	Jogging	49105962326000	-0.694638	12.680544	0.503953
1	33	Jogging	49106062271000	5.012288	11.264028	0.953424
2	33	Jogging	49106112167000	4.903325	10.882658	-0.081722
3	33	Jogging	49106222305000	-0.612916	18.496431	3.023717
4	33	Jogging	49106332290000	-1.184970	12.108489	7.205164

# Traditional models: activity recognition

Note: Improved features using traditional machine learning models (SVM, RF, LR) can yield an accuracy in the ~92% range (Quihao Jin, Jay Urbain):

- 3-sec mean
- 5-sec mean
- Median filtering
- Acceleration axis deltas
- FFT - Fast Fourier Transform



# DL Experiment - WISDM Activity Dataset

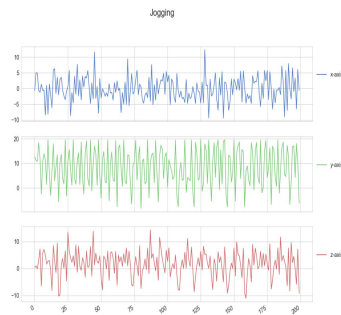
## Temporal sensor analysis and activity

	user	activity	timestamp	x-axis	y-axis	z-axis
0	33	Jogging	49105962326000	-0.694638	12.680544	0.503953
1	33	Jogging	49106062271000	5.012288	11.264028	0.953424
2	33	Jogging	49106112167000	4.903325	10.882658	-0.081722
3	33	Jogging	49106222305000	-0.612916	18.496431	3.023717
4	33	Jogging	49106332290000	-1.184970	12.108489	7.205164

Model	Accuracy
1) 1-layer LSTM	94.2
2) 2-layer LSTM	95.5
3) 1-layer Convnet	94.5
<b>4) 2-layer Convnet</b>	<b>98.7</b>
5) 2-layer Convnet + 1-layer LSTM	98.5

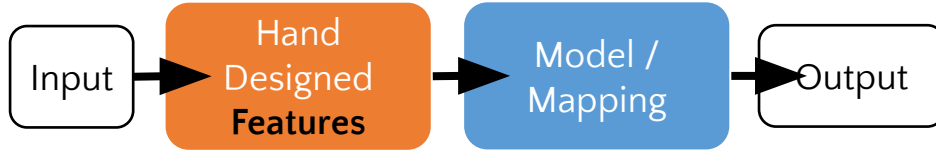
Layer (type)	Output Shape	Param #
conv1d_9 (Conv1D)	(None, 100, 64)	1024
max_pooling1d_9 (MaxPooling1	(None, 50, 64)	0
dropout_9 (Dropout)	(None, 50, 64)	0
conv1d_10 (Conv1D)	(None, 50, 128)	41088
max_pooling1d_10 (MaxPooling	(None, 25, 128)	0
dropout_10 (Dropout)	(None, 25, 128)	0
gru_5 (GRU)	(None, 25, 50)	26850
flatten_5 (Flatten)	(None, 1250)	0
dense_8 (Dense)	(None, 6)	7506
Total params: 76,468		

**Test accuracy: 0.987**

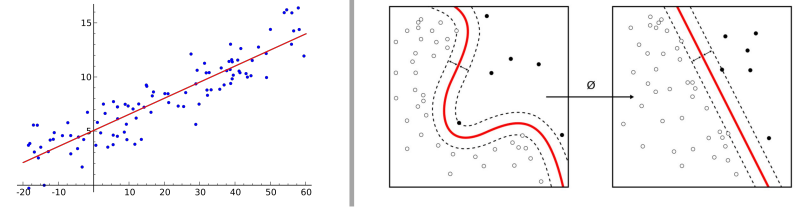


# Difference in Workflow

Classic Machine Learning [ 1990 : now ]



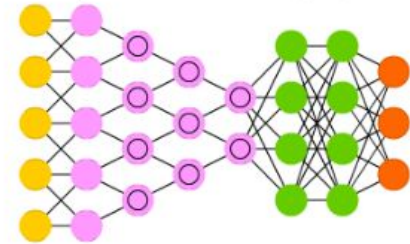
Examples [ Regression and SVMs ]



Deep/End-to-End Learning [ 2012 : now ]



Example [ Conv Net ]



Machine learning workflow shifts from engineering features for “shallow” models to architecting deep learning models with the ability to learn hierarchical representations of features

# Trainable feature hierarchy

Hierarchy of representations with increasing level of abstraction. Each stage is a kind of trainable feature transform

- Image recognition

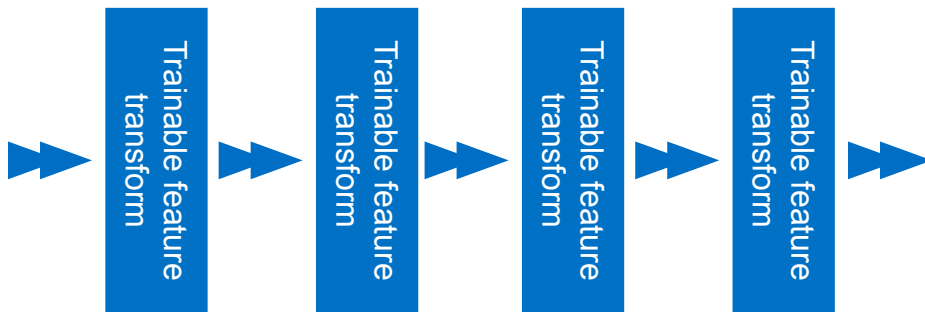
- Pixel → edge → motif → part → object

- Text

- Character → word → word group → clause → sentence → story/semantic understanding

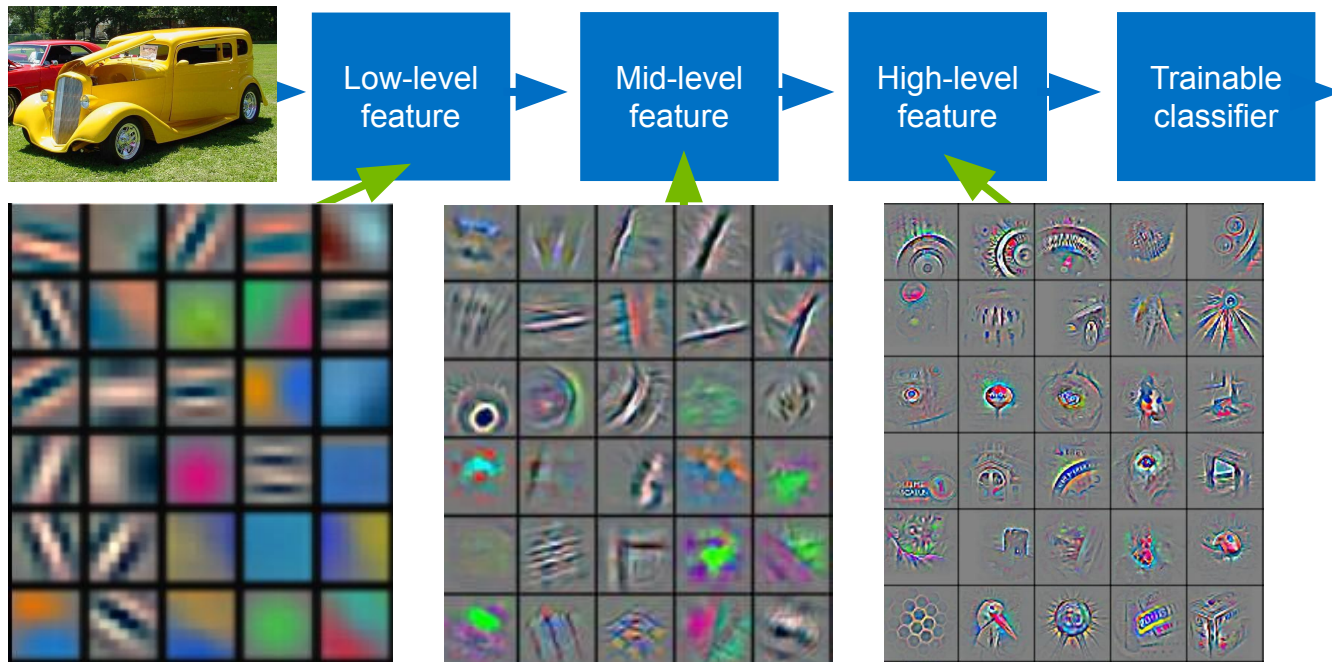
- Speech

- Sample → spectral band → sound → ... → phone → phoneme → word



# Deep learning = learning hierarchical representations

It's **deep** if it has **more than one stage** of non-linear feature transformation

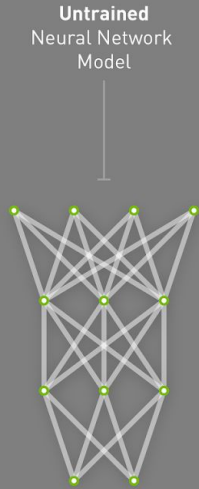


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

# DEEP LEARNING

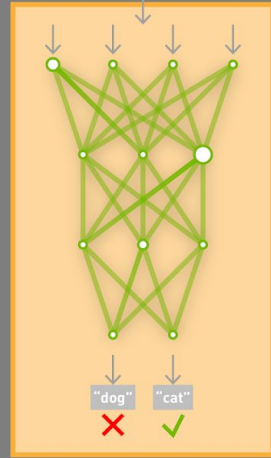
## TRAINING

Learning a new capability  
from existing data



Deep Learning  
Framework

TRAINING  
DATASET



Trained Model  
New Capability



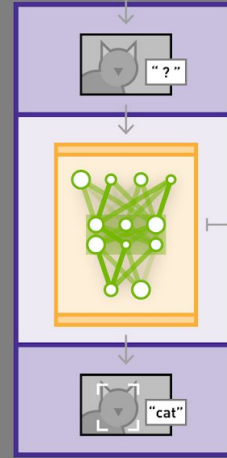
## INFERENCE

Applying this capability  
to new data

NEW  
DATA

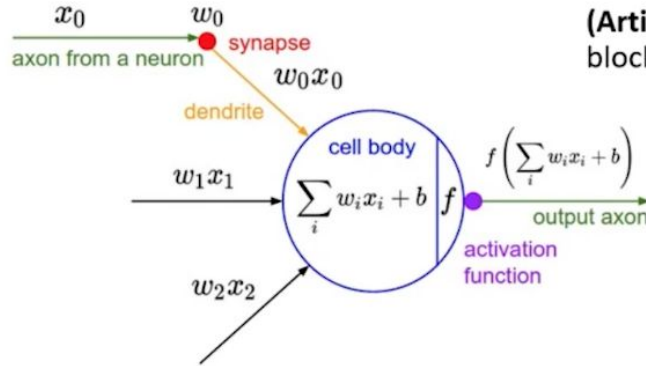


App or Service  
Featuring Capability

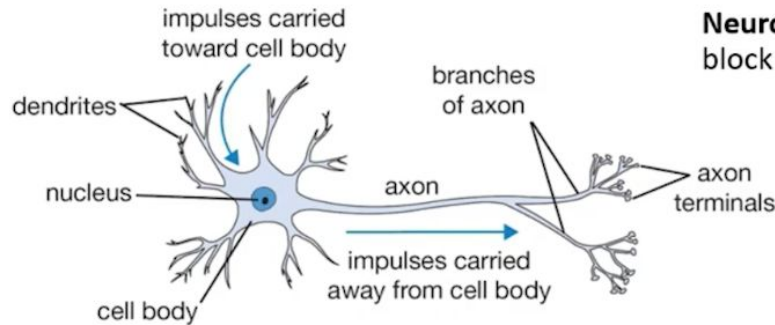


Trained Model  
Optimized for  
Performance

# Neuron: Biological Inspiration for Computation

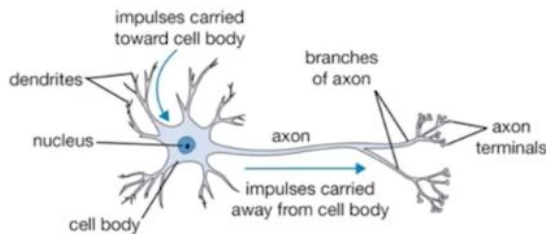


**(Artificial) Neuron:** computational building block for the "neural network"

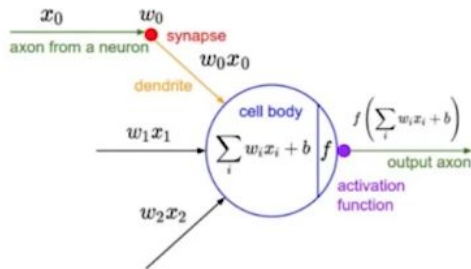


**Neuron:** computational building block for the brain

# Neuron: Biological Inspiration for Computation



- **Neuron:** computational building block for the brain



- **(Artificial) Neuron:** computational building block for the “neural network”

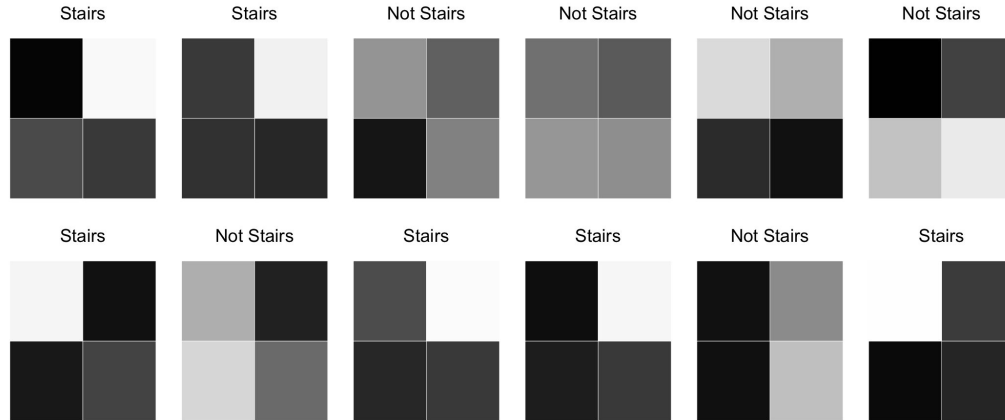
## Key Difference:

- **Parameters:** Human brains have  $\sim 10,000,000$  times synapses than artificial neural networks.
- **Topology:** Human brains have no “layers”. **Async:** The human brain works asynchronously, ANNs work synchronously.
- **Learning algorithm:** ANNs use gradient descent for learning. We don't know what human brains use
- **Power consumption:** Biological neural networks use very little power compared to artificial networks
- **Stages:** Biological networks usually never stop learning. ANNs first train then test.



# Non-linear problem: predict Stairs/Not Stairs

- Collection of grayscale images, each a 2×2 grid of pixels where each pixel has an intensity value between 0 (white) and 255 (black).
- The goal is to build a model that identifies images with a “stairs” pattern.



# Single Layer Perceptron

$$f(x) = \begin{cases} 1 & \text{if } w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

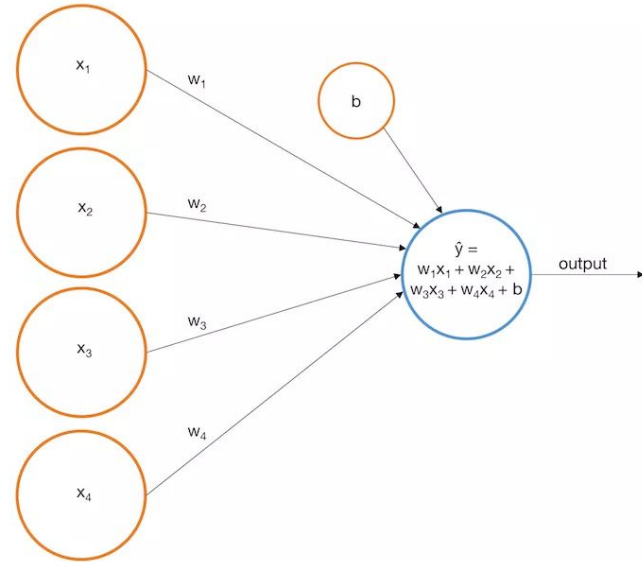
Let's re-express this as follows

$$\hat{y} = \mathbf{w} \cdot \mathbf{x} + b$$

$$f(x) = \begin{cases} 1 & \text{if } \hat{y} > 0 \\ 0 & \text{otherwise} \end{cases}$$

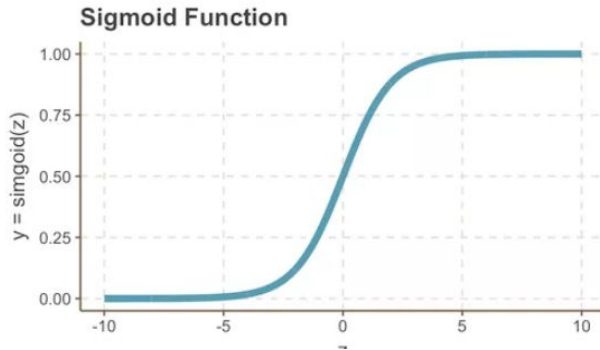
Here  $\hat{y}$  is our *prediction score*.

$$\hat{y} = -0.0019x_1 + -0.0016x_2 + 0.0020x_3 + 0.0023x_4 + 0.0003$$



# Single Layer Perceptron with Sigmoid Activation: use non-linear activation function

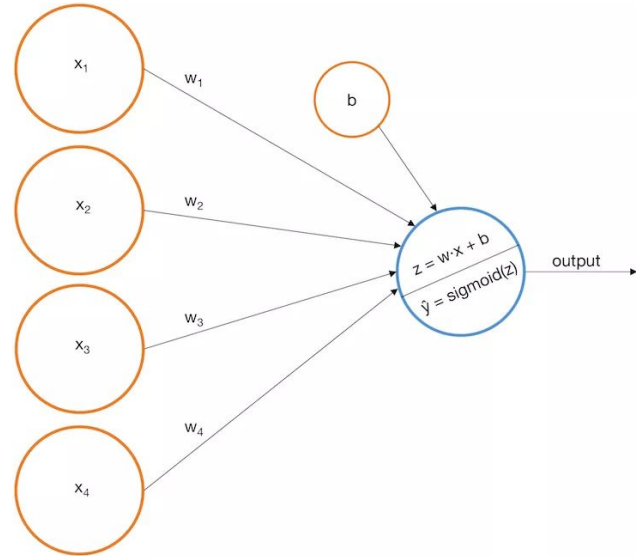
$$\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$$



$$z = w \cdot x = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$$

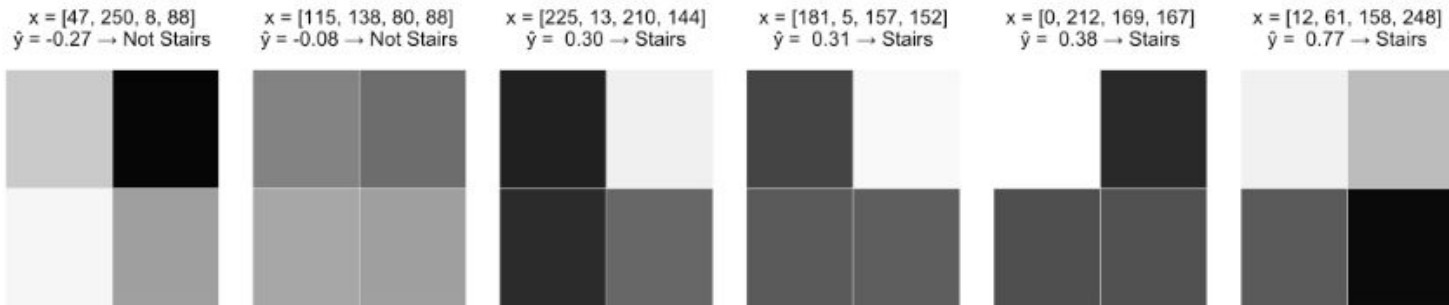
$$\hat{y} = \text{sigmoid}(z) = \frac{1}{1+e^{-z}}$$

$$f(x) = \begin{cases} 1 & \text{if } \hat{y} > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

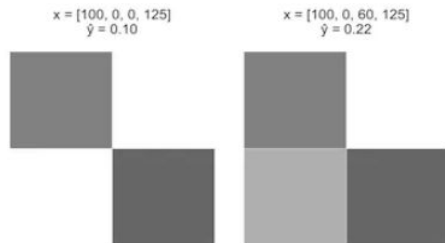


Logistic  
Regression

# Single Layer Perceptron – problems



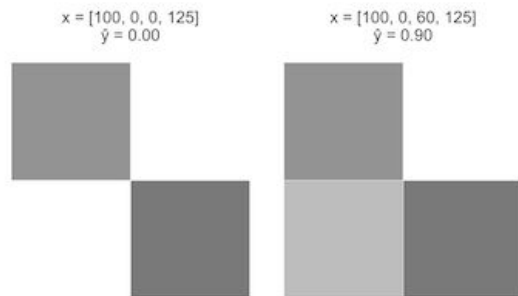
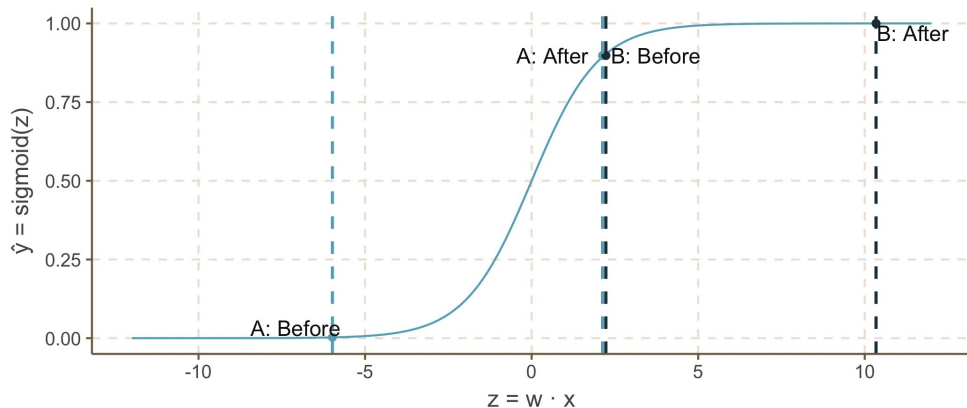
- The model outputs a real number whose value correlates with the concept of likelihood, but there's no basis to interpret the values as probabilities, they can be outside the range  $[0, 1]$ .
- The model can't capture the non-linear relationship (interactions) between the variables and the target:



Start with an image,  $x = [100, 0, 0, 125]$ . Increase  $x_3$  from 0 to 60  
Variables are independent.

# Sigmoid

- The sigmoid function causes the increase of 60 (right image) to “fire” (increase rapidly) as  $z = w \cdot x$  increases, but the pace slows down as  $z$  continues to increase.
- This aligns with our intuition that the right image should reflect a greater increase in the likelihood of stairs versus the left image.



# Still have problems

- $\hat{y}$  has a monotonic relationship with each variable. What if we want to identify lightly shaded stairs?
- The model does not account for variable interaction.
  - Increasing or decreasing a pixel by  $x_3$  should potentially increase or decrease  $\hat{y}$  depending on the values of the other variables. The current model has no way of achieving this.

# Multi-Layer Perceptron with Sigmoid activation function

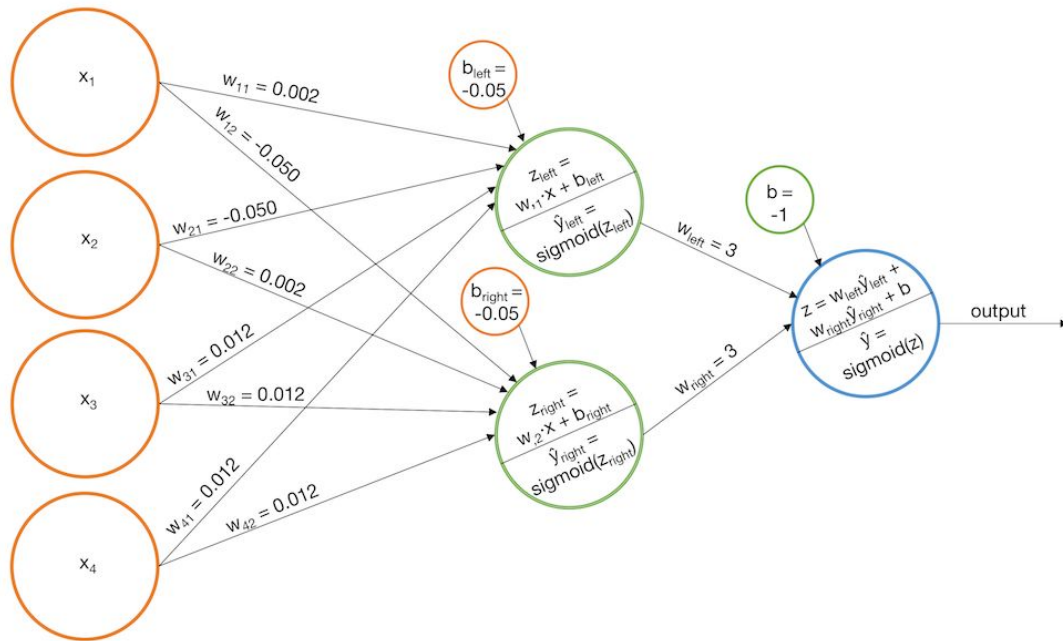
- Can solve both of the above issues by adding an extra *layer* to the perceptron model.



# Multi-Layer Perceptron with Sigmoid activation function

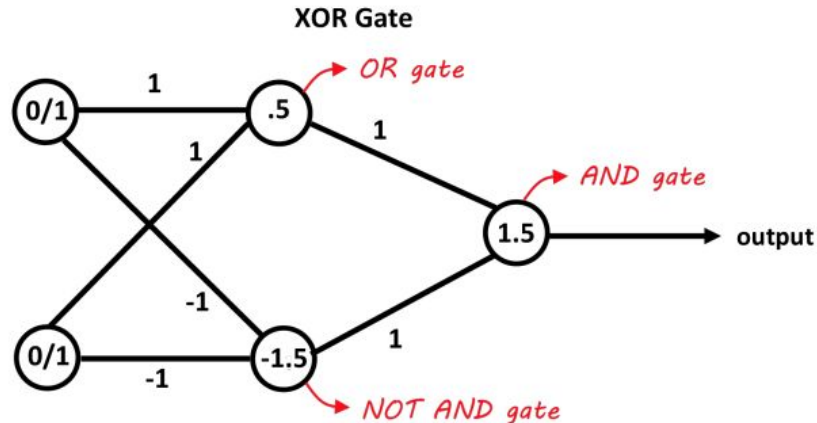
## Example: Identify the stairs pattern

- Build a model that fires when “left stairs” are identified,  $\hat{y}_{\text{left}}$
- Build a model that fires when “right stairs” are identified,  $\hat{y}_{\text{right}}$
- Add the score of the base models so that the final sigmoid function only fires if **both**  $\hat{y}_{\text{left}}$  and  $\hat{y}_{\text{right}}$  are large



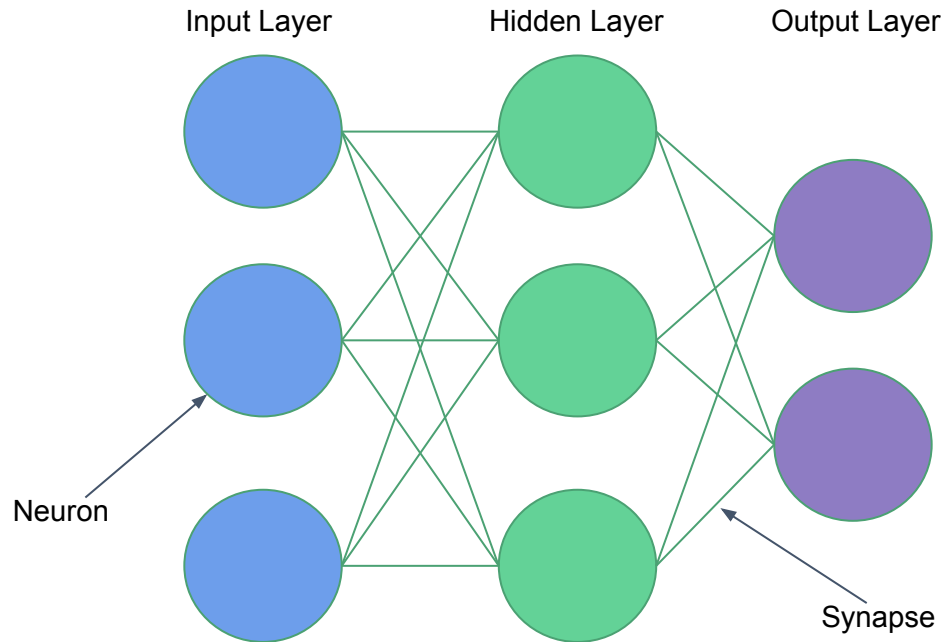
# XOR Problem – Minsky paper

- Can't solve with a single layer
- Exercise: create single node networks for OR and AND. Try to create single node network for XOR. Create and verify solution with 2 layers.



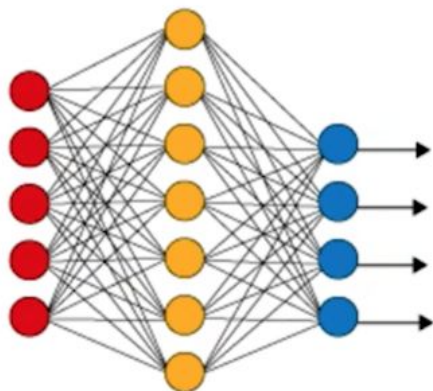
# Neural Network Architecture

## Multi-layer Perceptron

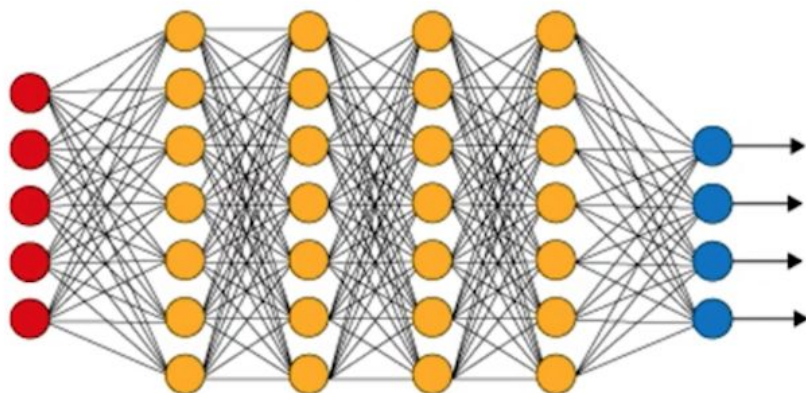


# Combing Neurons in Hidden Layers: The “Emergent” Power to Approximate

Simple Neural Network



Deep Learning Neural Network



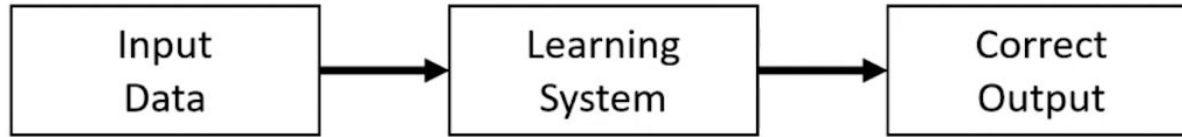
● Input Layer

● Hidden Layer

● Output Layer

**Universality:** For any arbitrary function  $f(x)$ , there exists a neural network that closely approximate it for any input  $x$

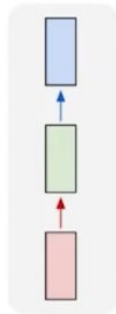
# What can we do with Deep Learning?



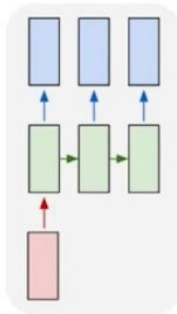
- Number
- Vector of numbers
- Sequence of numbers
- Sequence of vectors of numbers

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

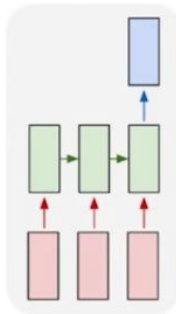
one to one



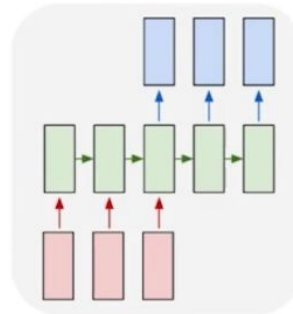
one to many



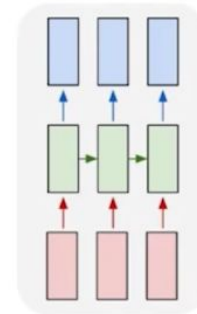
many to one



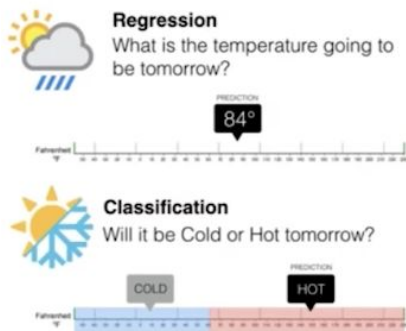
many to many



many to many



# Loss Functions



- Loss function quantifies gap between prediction and ground truth
- **For regression:**
  - Mean Squared Error (MSE)
- **For classification:**
  - Cross Entropy Loss

## Mean Squared Error

$$MSE = \frac{1}{N} \sum (t_i - s_i)^2$$

Prediction  $s_i$

Ground Truth  $t_i$

## Cross Entropy Loss

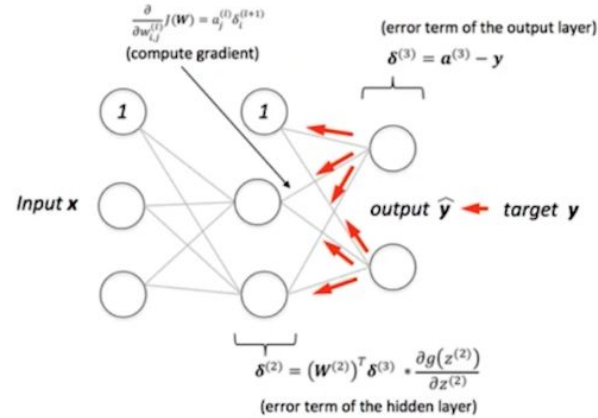
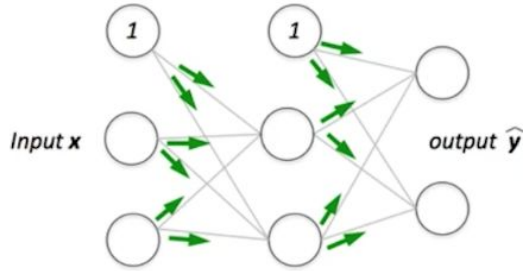
$$CE = - \sum_i^C t_i \log(s_i)$$

Classes  $C$

Prediction  $s_i$

Ground Truth  $\{0,1\}$   $t_i$

# Backpropagation



**Task:** Update the **weights** and **biases** to decrease **loss function**

**Subtasks:**

1. Forward pass to compute network output and “error”
2. Backward pass to compute gradients
3. A fraction of the weight’s gradient is subtracted from the weight.



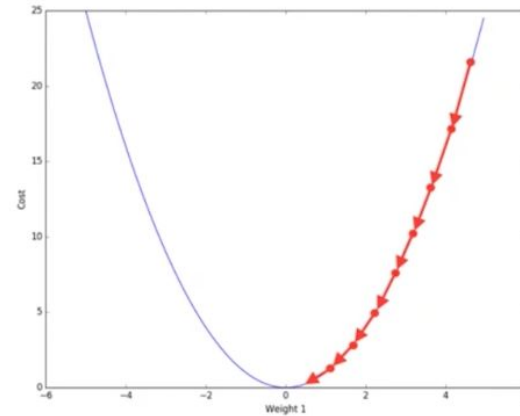
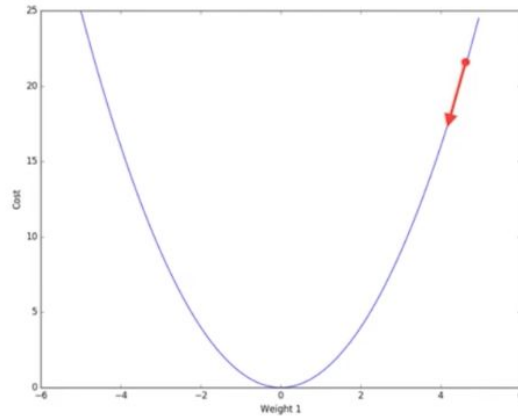
Learning Rate

Numerical Method: **Automatic Differentiation**



# Learning is an Optimization Problem

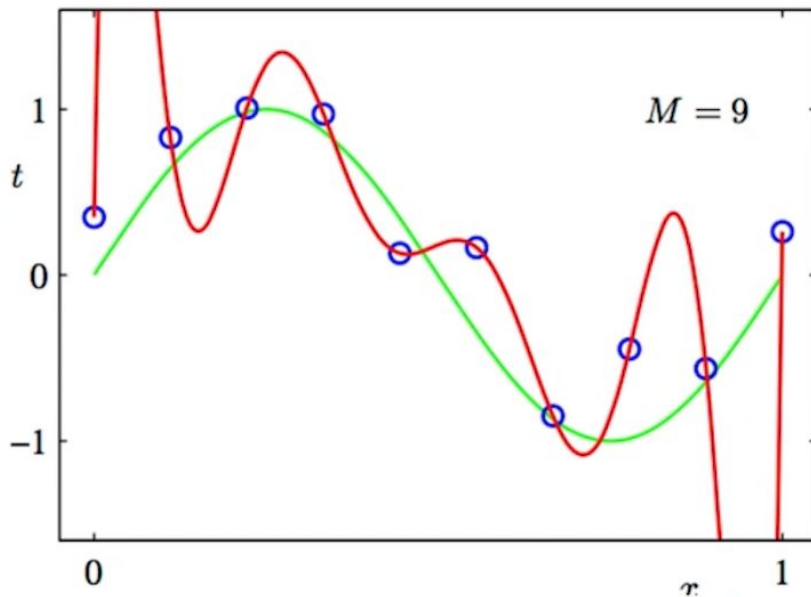
**Task:** Update the **weights** and **biases** to decrease **loss function**



SGD: Stochastic Gradient Descent

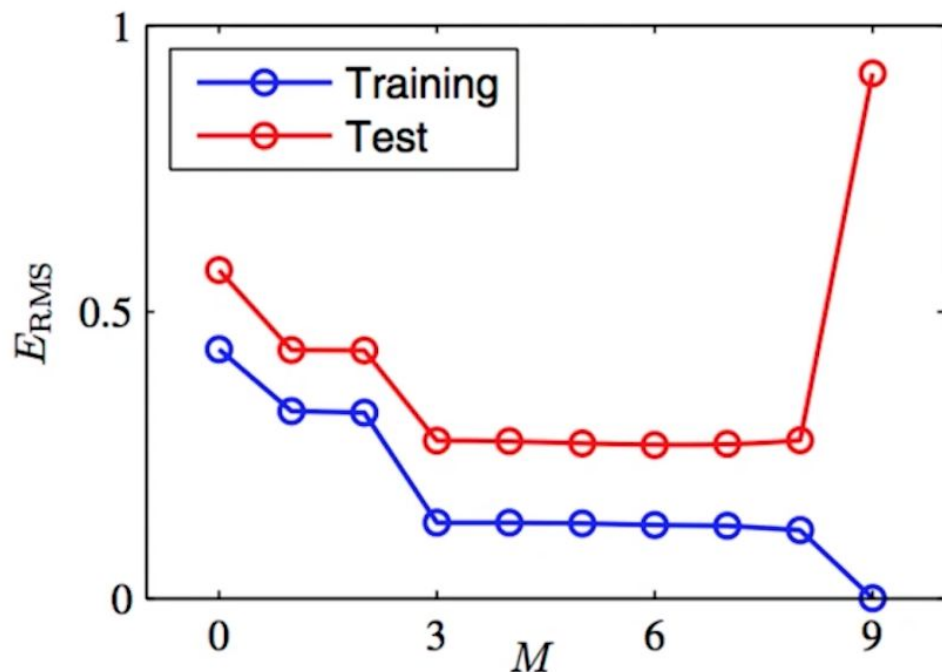
# Overfitting and Regularization

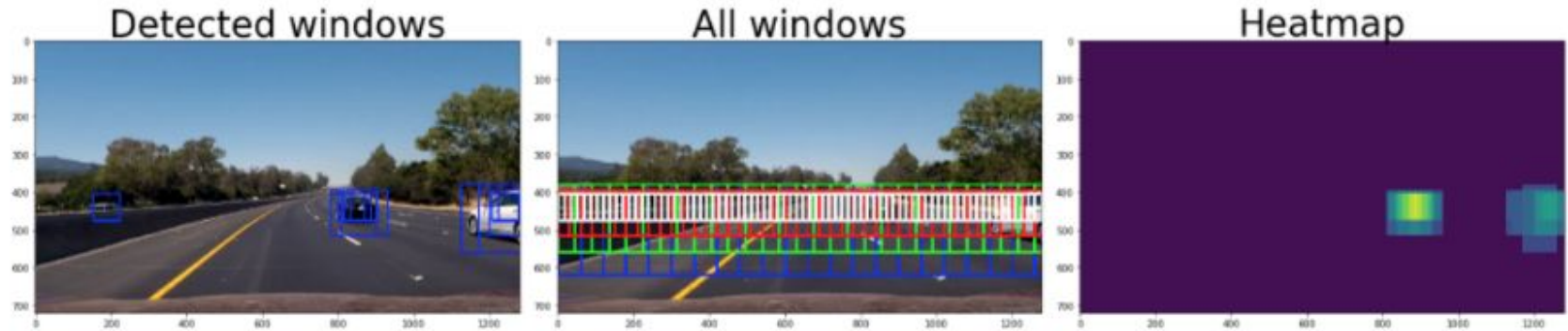
- Help the network **generalize** to data it hasn't seen.
- Big problem for **small datasets**.
- Overfitting example (a sine curve vs 9-degree polynomial):



# Overfitting and Regularization

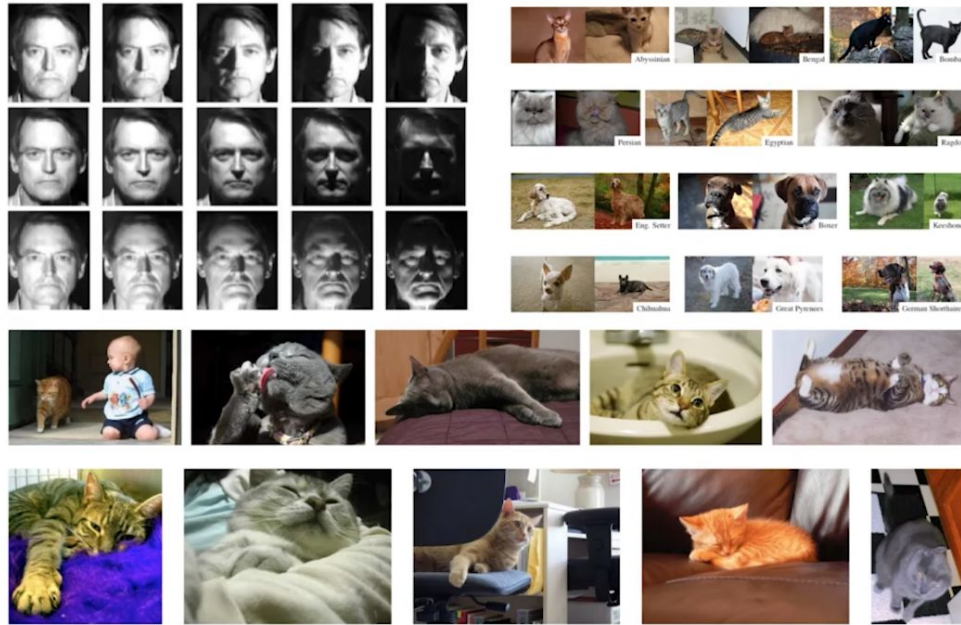
- Overfitting: The error decreases in the training set but increases in the test set.





Systems are currently good at basic prediction problems - deep learning + supervised data.

Pure perception, i.e., scene understanding is very hard.  
Identifying objects in an image is good, but understanding how they relate is hard



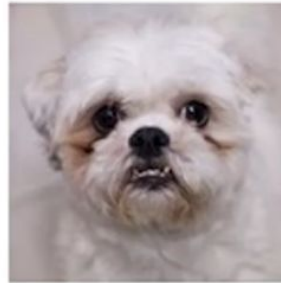
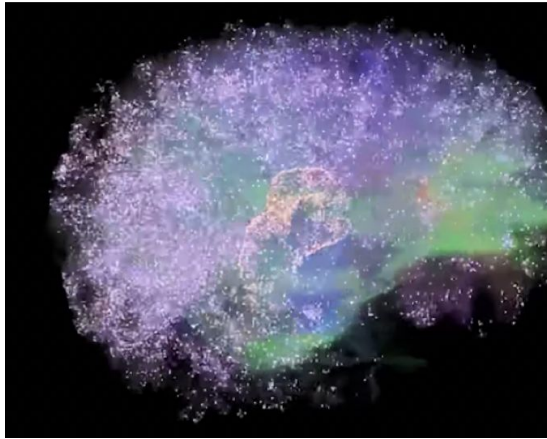
# Deep Learning:

Our intuition about what's "hard" is flawed (in complicated ways)

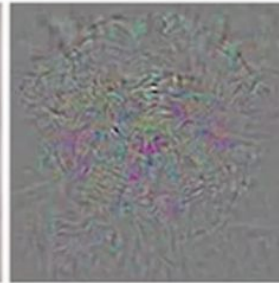
**Visual perception:** 540,000,000 years of data

**Bipedal movement:** 230,000,000 years of data

**Abstract thought:** 100,000 years of data



Prediction: **Dog**



+ Distortion



Prediction: **Ostrich**

Generative Models may  
help

"Encoded in the large, highly evolve sensory and motor portions of the human brain is a **billion years of experience** about the nature of the world and how to survive in it.... Abstract thought, though, is a new trick, perhaps less than **100 thousand years** old. We have not yet mastered it. It is not all that intrinsically difficult; it just seems so when we do it."

- Hans Moravec, *Mind Children* (1988)