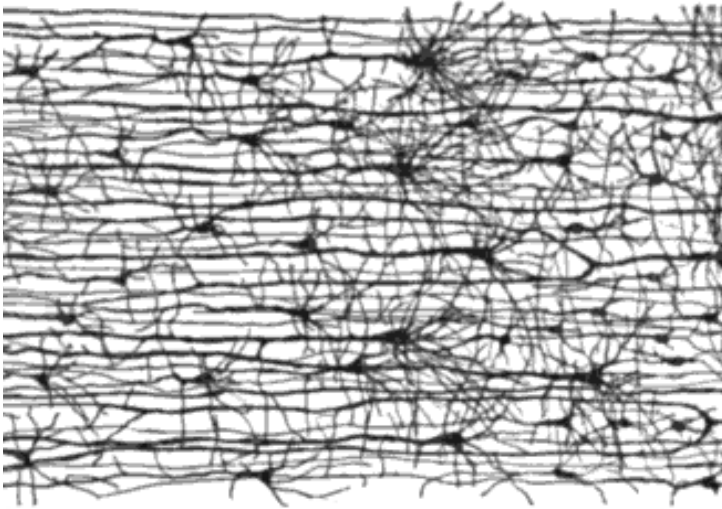


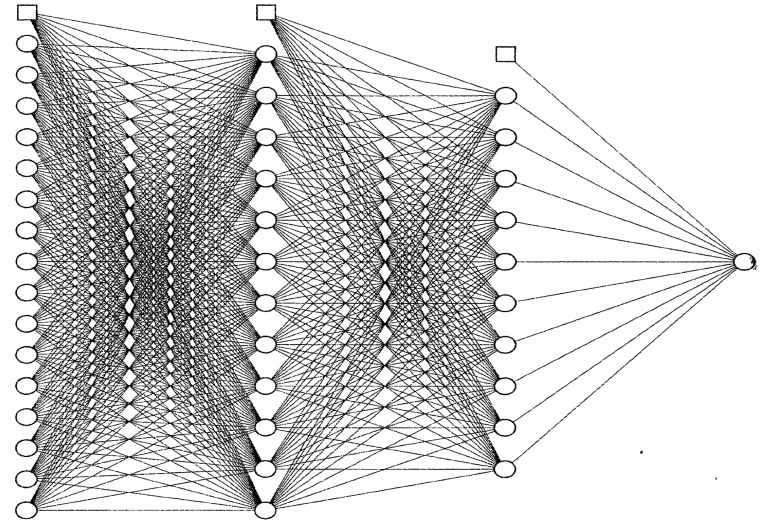
Section 10 Artificial Neural Network

- Background
- Perceptron
- Neural Network
- Backpropagation
- Algorithms

□ Human Neural Networks v.s. Artificial Neural Networks



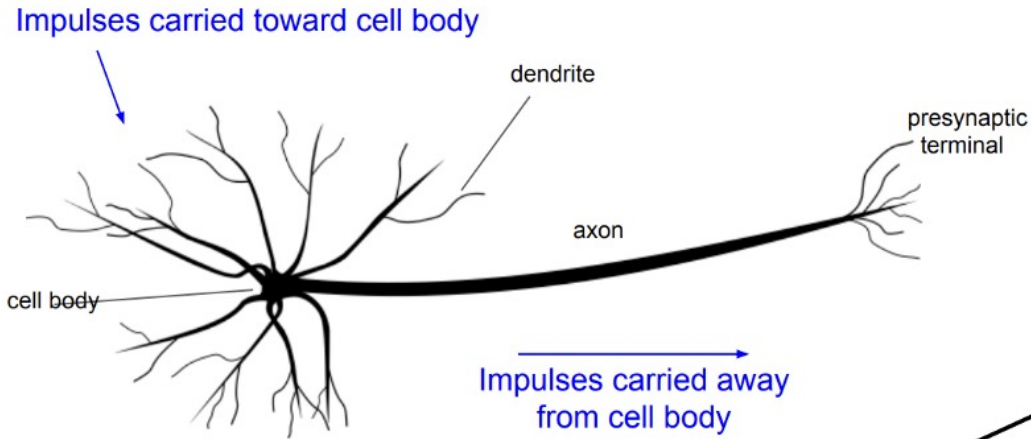
Human Neural Networks was introduced in 1943 by neurophysiologist Warren McCulloch and mathematician Walter Pitts to model neurons in the brain using electrical circuits.



Artificial Neural Networks are a series of algorithms that mimic the operations of a human brain to recognize relationships between vast amounts of data. It's a very broad term that encompasses any form of Deep Learning model.

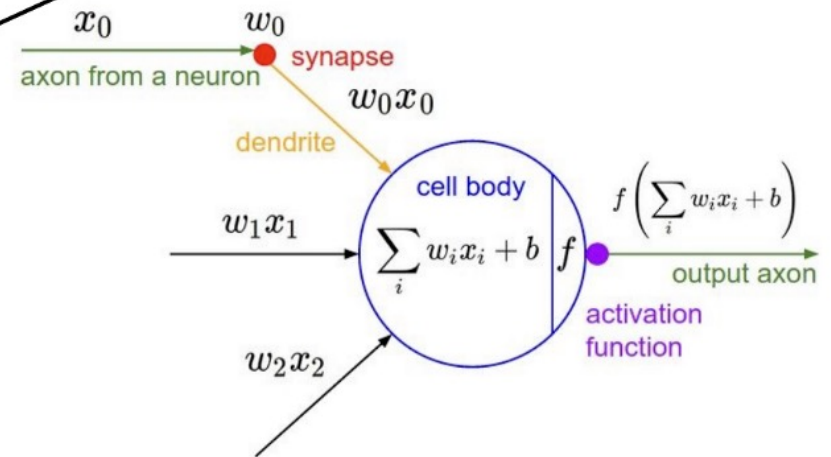
- **Neurons**

A biological Neuron

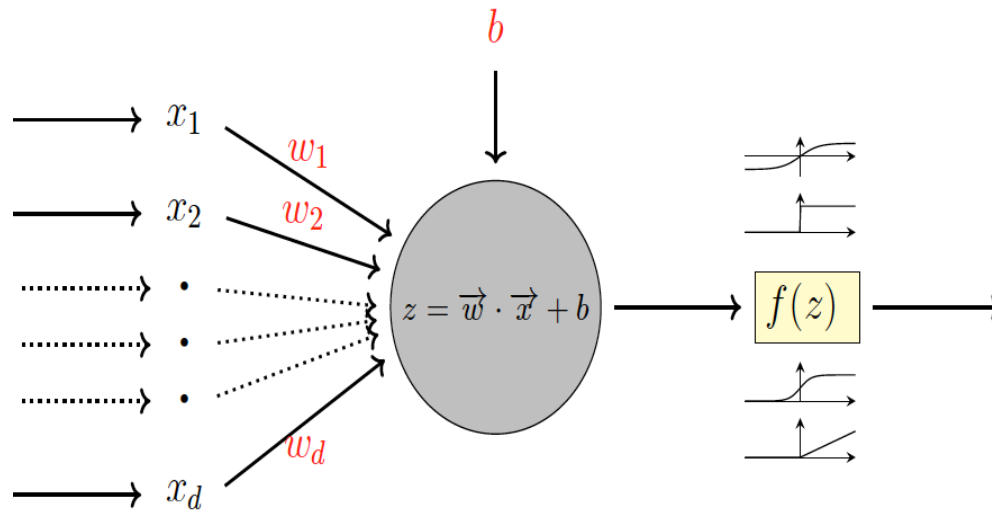
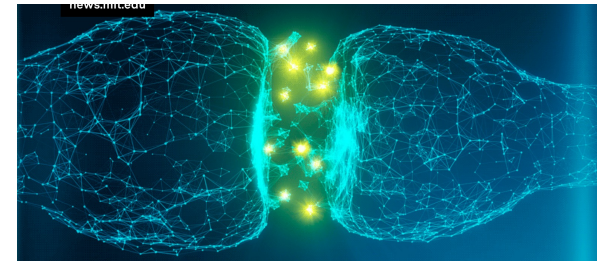


This image by Felipe Perucho is licensed under [CC-BY 3.0](https://creativecommons.org/licenses/by/3.0/)

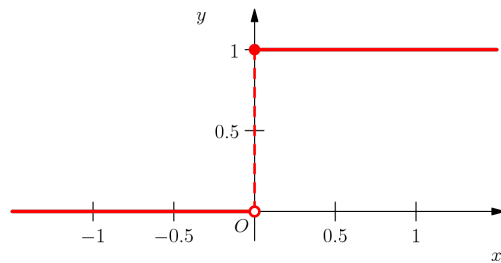
An artificial Neuron



□ Activation functions:

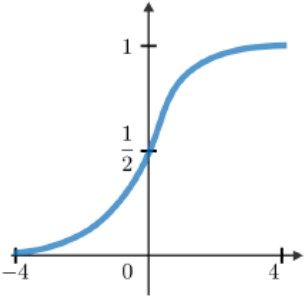
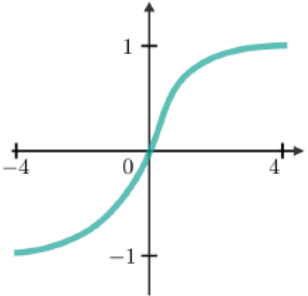
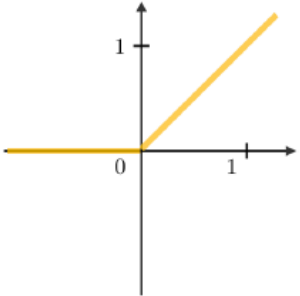
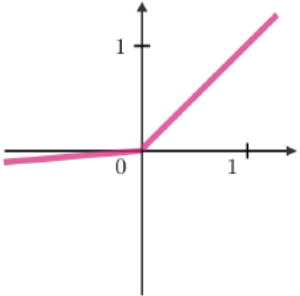


Activation function example: Heaviside step function



$$f(x) = \text{step}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

➤ More Activation Functions:

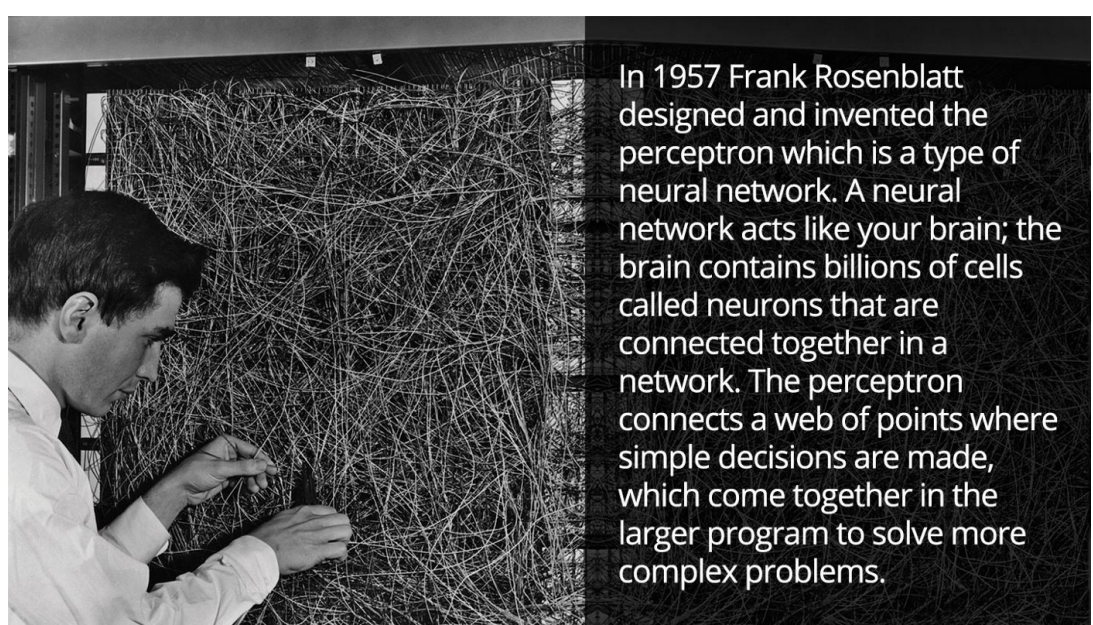
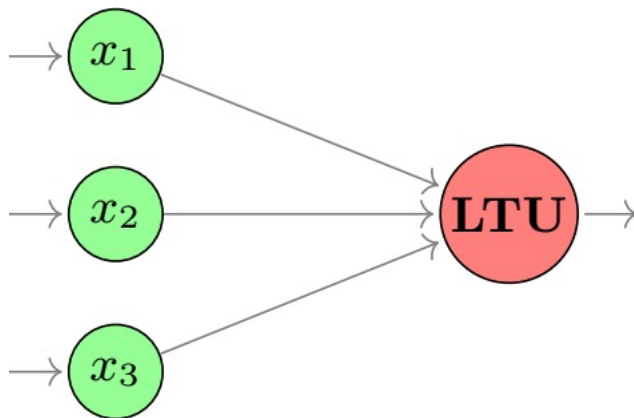
Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

Piecewise Linear, Gaussian, etc.

➤ Perceptron

- Frank Rosenblatt (psychologist).

The perceptron is based around a *linear threshold unit (LTU)*



Mark I Perceptron machine

The New Yorker, December 6, 1958 P. 44

Talk story about the perceptron, a new electronic brain which hasn't been built, but which has been successfully simulated on the I.B.M. 704. Talk with Dr. Frank Rosenblatt, of the Cornell Aeronautical Laboratory, who is one of the two men who developed the prodigy; the other man is Dr. Marshall C. Yovits, of the Office of Naval Research, in Washington. Dr. Rosenblatt defined the perceptron as the first non-biological object which will achieve an organization of its external environment in a meaningful way. It interacts with its environment, forming concepts that have not been made ready for it by a human agent. If a triangle is held up, the perceptron's eye picks up the image & conveys it along a random succession of lines to the response units, where the image is registered. It can tell the difference betw. a cat and a dog, although it wouldn't be able to tell whether the dog was to the left or right of the cat. Right now it is of no practical use, Dr. Rosenblatt conceded, but he said that one day it might be useful to send one into outer space to take in impressions for us.

<https://www.newyorker.com/magazine/1958/12/06/rival-2>

<https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon>

Remark: The 704 at that time was thus regarded as "pretty much the only computer that could handle complex math". (\$2M, 30,000lb) A current PC should be 100,000 times faster than IBM 704.

➤ Perceptron

Training Data: $\mathcal{D} = (\vec{x}^{(i)}, y^{(i)})$ for $i = 1 \dots n$.

Assumptions:

- *Binary classification* (i.e. $y^{(i)} \in \{-1, +1\}$)
- Data is *linearly separable*, i.e., there exists a hyperplane that separates all the sample points in class A from classes B.

Classifier:

$$h(\vec{x}) = f(\vec{\theta}^T \vec{x}) = \text{sign}(\vec{w} \cdot \vec{x} + b) = \begin{cases} 1, & \text{if } \vec{w} \cdot \vec{x} + b \geq 0 \\ -1, & \text{if } \vec{w} \cdot \vec{x} + b < 0 \end{cases}$$

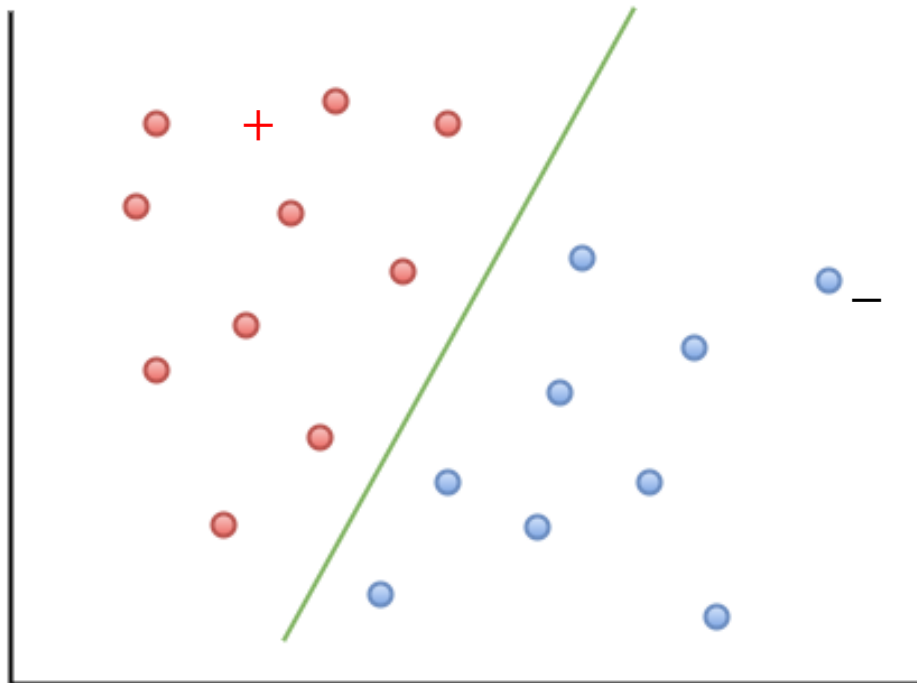
Notations: $\vec{\theta} = \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$, $\vec{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$ or $\vec{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$

Remark: If the label space is $\{0,1\}$, the classifier is the threshold function, i.e., $f(z)$ is the step function.

- **Decision Boundary: Hyperplane** $H = \{\vec{x} \in \mathbb{R}^{d+1} \mid \vec{\theta}^T \vec{x} = 0\}$

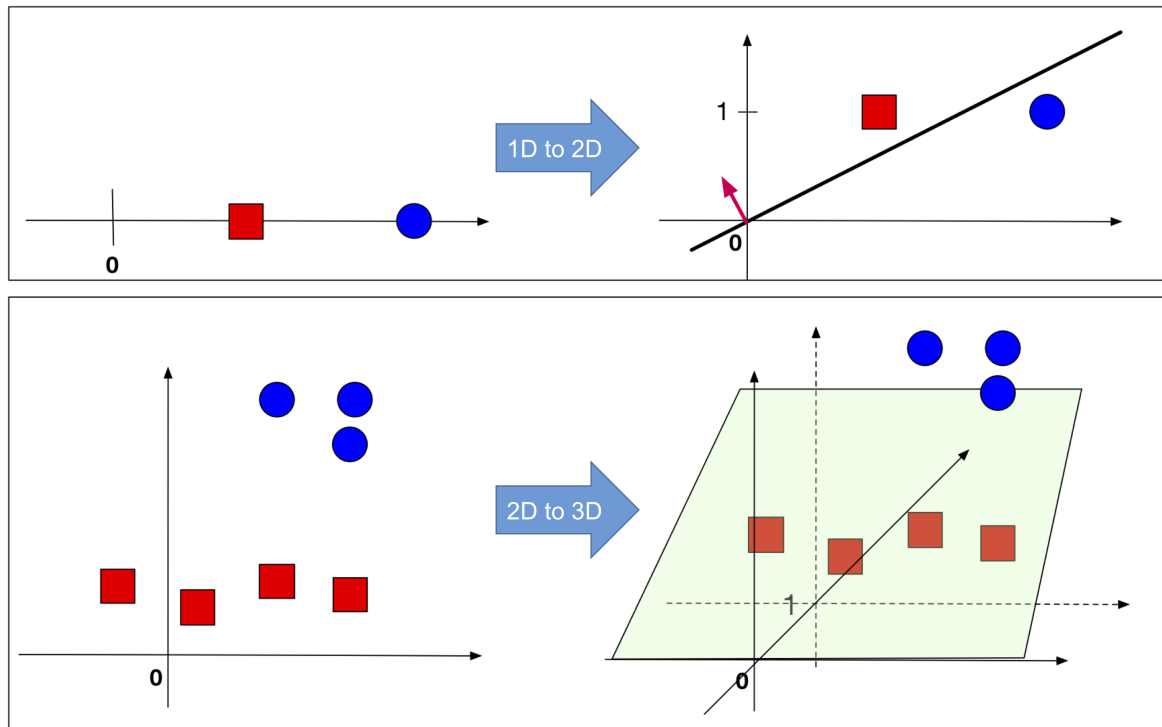
or: $H = \{\vec{x} \in \mathbb{R}^d \mid w_1 x_1 + \dots + w_d x_d + b = 0\}$

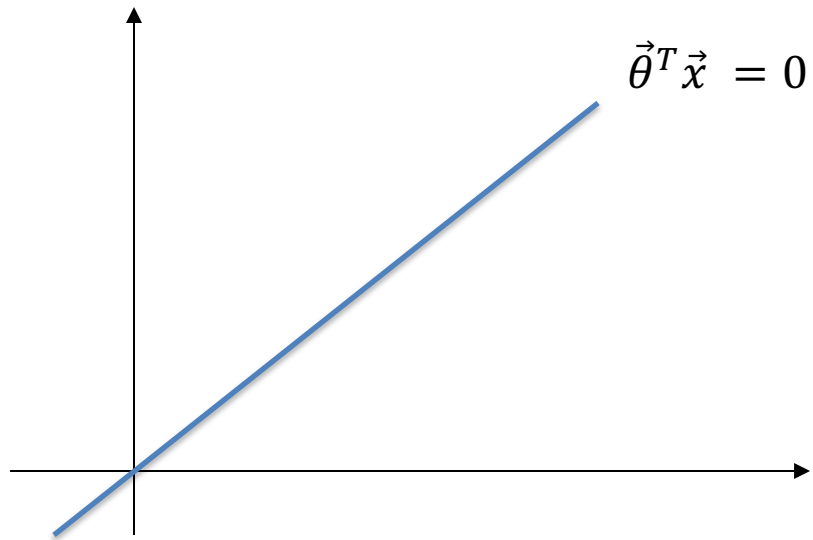
or: $H = \{\vec{x} \in \mathbb{R}^d \mid \vec{w}^T \vec{x} + b = 0\}$



Property: \vec{w} is orthogonal to the hyperplane H .

Back to notation with $x_0 = 1$





Suppose $y^{(i)} \in \{-1, +1\}$, we have

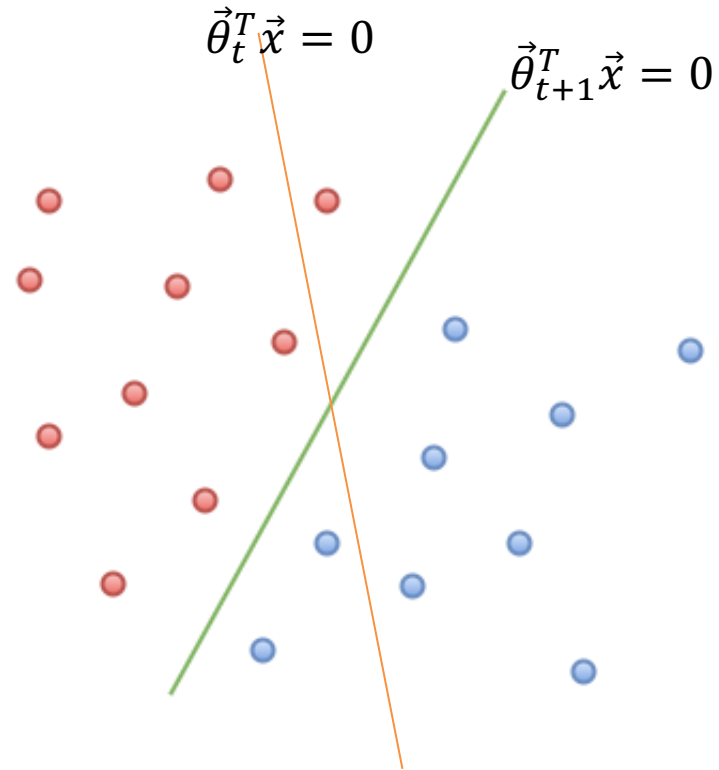
$$y^{(i)}(\vec{\theta}^T \vec{x}^{(i)}) > 0 \Leftrightarrow \vec{x}^{(i)} \text{ classified correctly.}$$

Perceptron Algorithm

Suppose $y^{(i)} \in \{-1, +1\}$,

$$h_t(\vec{x}) = \text{sign}(\vec{\theta}_t^T \vec{x})$$

Learn from step t to step $t + 1$



- If $\vec{x}^{(i)}$ is correctly classified, i.e., $y^{(i)} - h_{\theta}(\vec{x}^{(i)}) = 0$, then move on.
- If $(\vec{x}^{(i)}, y^{(i)} = -1)$ is misclassified, i.e., $y^{(i)} - h_{\theta}(\vec{x}^{(i)}) = -2$, then $\vec{\theta}_{t+1} := \vec{\theta}_t - \alpha \vec{x}^{(i)}$
- If $(\vec{x}^{(i)}, y^{(i)} = 1)$ is misclassified, i.e., $y^{(i)} - h_{\theta}(\vec{x}^{(i)}) = 2$, then $\vec{\theta}_{t+1} := \vec{\theta}_t + \alpha \vec{x}^{(i)}$

➤ Training the Perceptron

$\mathcal{D} = (\vec{x}^{(i)}, y^{(i)})$ for $i = 1 \dots n$.

Start with initial $\vec{\theta} = \vec{0}$

For $i = 1, \dots, n$

Repeat $\vec{\theta}^{next} := \vec{\theta} + \alpha (y^{(i)} - h_{\theta}(\vec{x}^{(i)})) \vec{x}^{(i)}$

The perceptron updates its weights only on misclassified points.

```

def perceptron_sgd(X, Y):
    w = np.zeros(len(X[0])) #Initialize the weight vector for the perceptron with zeros
    eta = 1 #Set the learning rate to 1
    epochs = 20 #Set the number of epochs

    for t in range(epochs):
        for i, x in enumerate(X):
            if (np.dot(X[i], w)*Y[i]) <= 0:
                w = w + eta*X[i]*Y[i]

    return w

w = perceptron_sgd(X,y)
print(w)

```

The perceptron is a form of stochastic gradient decent on the loss function

$$J(\vec{\theta}) = - \sum_{i=1}^n \left(y^{(i)} - h_{\theta}(\vec{x}^{(i)}) \right) (\vec{\theta}^T \vec{x}^{(i)})$$

Remark:

We consider the **online learning setting** for the perceptron. The algorithm has to make predictions continuously even while it's learning.

Specifically, the algorithm first sees $\vec{x}^{(1)}$ and is asked to predict what it thinks $y^{(1)}$ is. After making its prediction, the true value of $y^{(1)}$ is revealed to the algorithm and the algorithm may use this information to perform some learning. The algorithm then see $\vec{x}^{(2)}$ and keep going.

In the online learning setting, we are interested in the total number of errors made by the algorithm during this process.

It models applications in which the algorithm has to make predictions even while it's still learning.

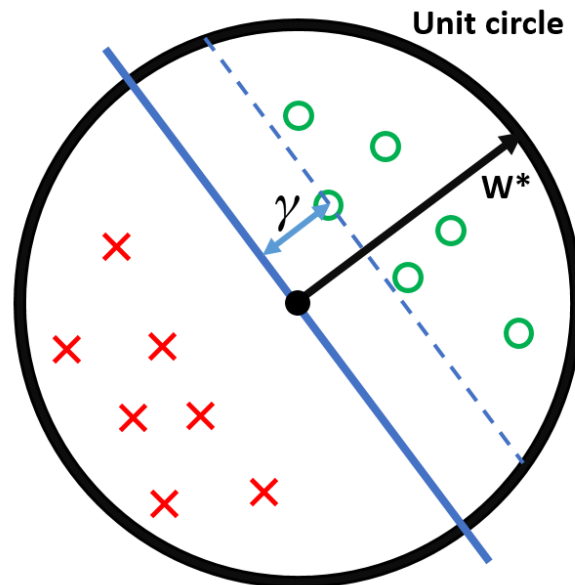
Convergence Theorem (Block, 1962, and Novikoff, 1962).

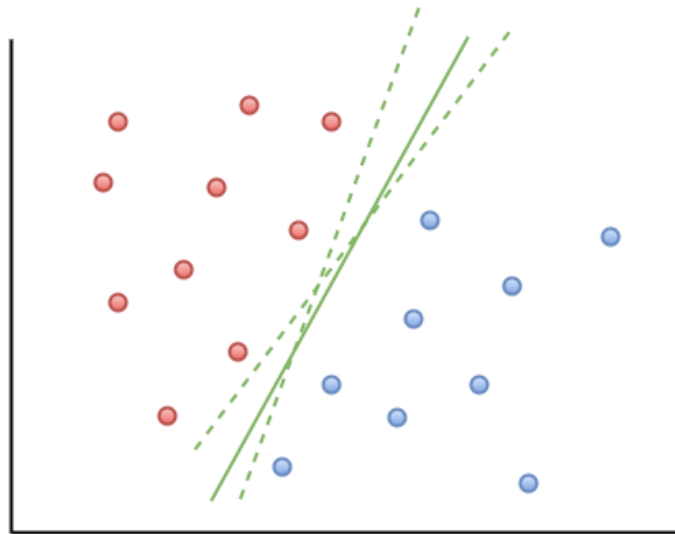
Suppose inputs are scaled to live within the **unit** sphere.

A separating hyperplane is defined by **unit** vector $\vec{\theta}$

$\gamma = \min_{\mathcal{D}} |\vec{\theta}^T \vec{x}^{(i)}|$ is the distance from hyperplane to the closed point.

Then, the Perceptron algorithm makes at most $\frac{1}{\gamma^2}$ mistakes.

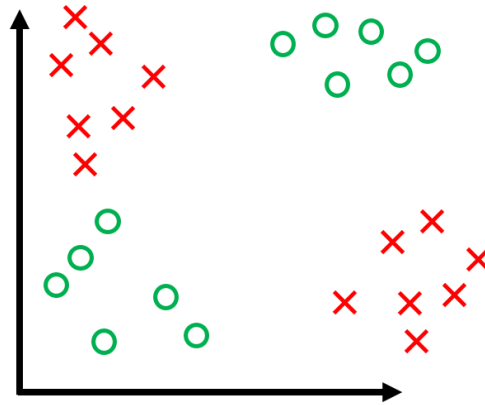




- When the data is separable, there are many solutions and which one is found depends on the starting value.
- The finite number of steps can be large, practically, if the gap is small the time to find it is large.
- When the data are not separable, the algorithm does not converge, and instead falls into a cycle.
- Video illustration for perceptron:
<https://www.youtube.com/watch?v=xpJHhHwR4DQ>
- Proof of convergence theorem.
<https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote03.html>

Perceptron's Dark Time:

Famous example of a simple non-linearly separable data set, the XOR problem



(Book *“Perceptrons: an introduction to computational geometry”* - by Marvin Minsky, founder of the MIT AI Lab, and Seymour Papert, director of the lab):

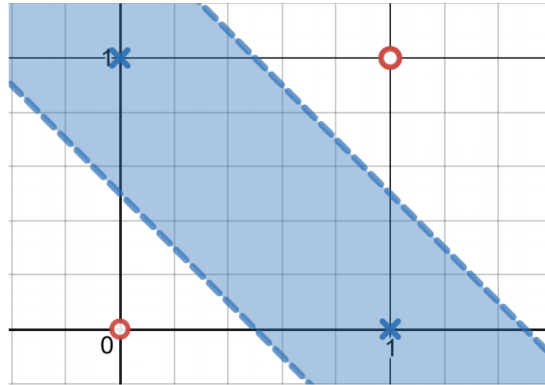
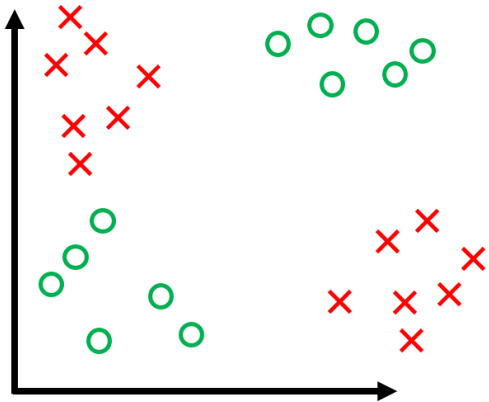
Although now unsurprising (no linear classifier can solve xor) the exceptions for the perceptron were high and when this problem was uncovered in 1969, it leads most researchers to abandon neural networks in favor of functional and logical methods.

Neural Network Time Line: 1957 -----> **1969** -----> 1980 -----> **1997** -----> 2010 -----> **2020** ----->?

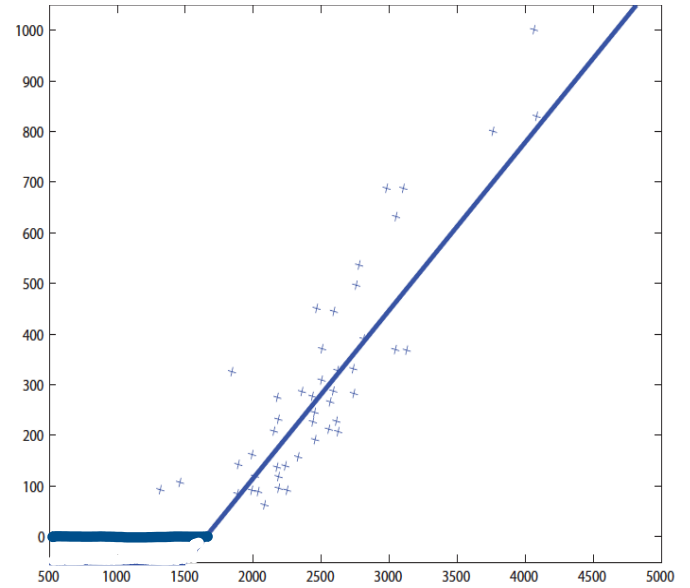
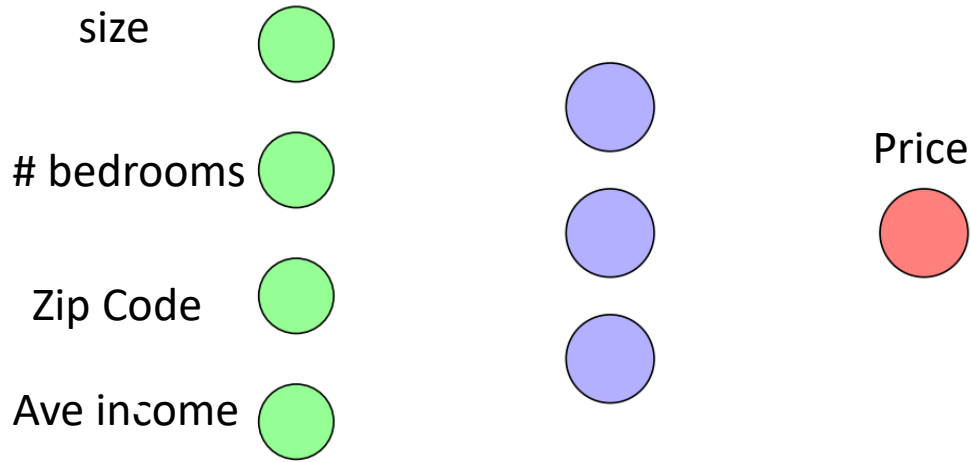
Some Interesting Documentary:

- Frank Rosenblatt https://www.youtube.com/watch?v=cNxadbrN_al
- The Thinking Machine - MIT 1961 <https://www.youtube.com/watch?v=5YBIrc-6G-0>
or a short version <https://youtu.be/aygSMgK3BEM>
- Marvin Minsky
https://openvault.wgbh.org/catalog/V_EC93438EE8A747989743A3987DD21409
- Yann LeCun: https://www.youtube.com/watch?v=FwFduRA_L6Q
- Vapnik <https://www.fi.edu/laureates/vladimir-vapnik>
- Computer for Apollo (1965) <https://www.youtube.com/watch?v=ndvmFlg1WmE>

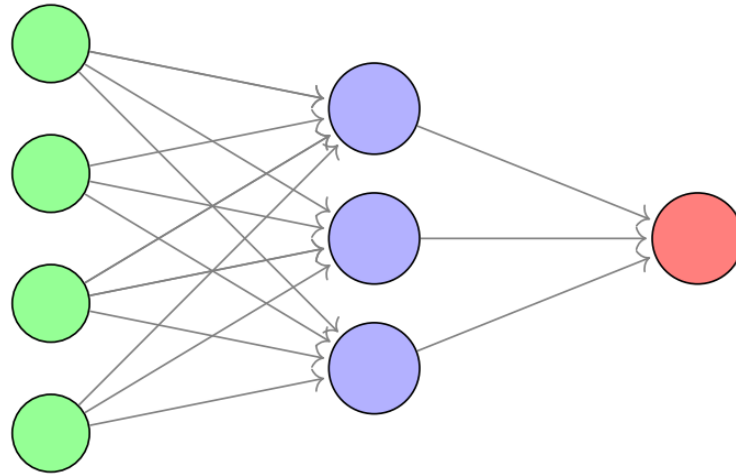
□ Two Layers Perceptron/Neural Network



■ House price example



- **Fully-connected neural networks**



Representational power

In theory, can represent any function. Assuming non-trivial non-linearity

– Bengio 2009,

<http://www.iro.umontreal.ca/~bengioy/papers/ftml.pdf>

– Bengio, Courville, Goodfellowbook

<http://www.deeplearningbook.org/contents/mlp.html>

– Simple visual proof by M. Neilsen

<http://neuralnetworksanddeeplearning.com/chap4.html>

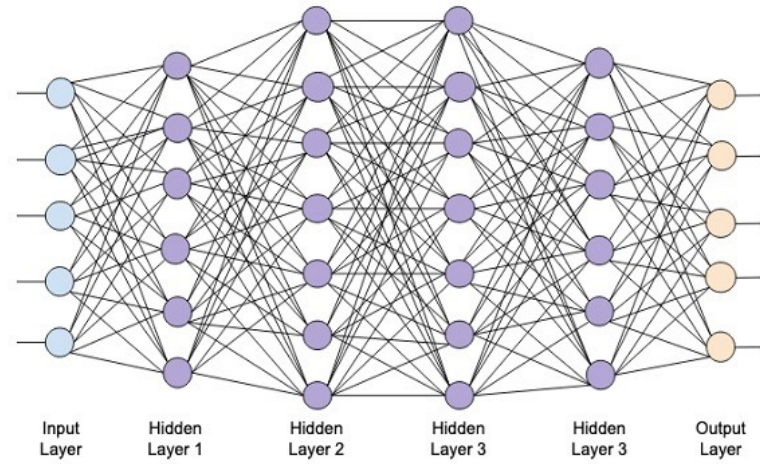
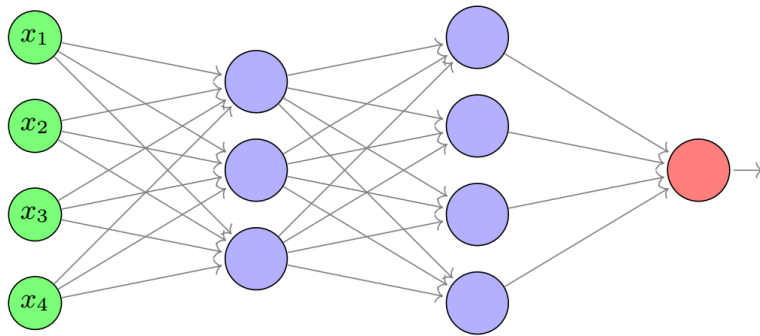
– D. Mackay book

<http://www.inference.phy.cam.ac.uk/mackay/itprnn/ps/482.491.pdf>

But issue is efficiency: very wide two layers vs narrow deep model?

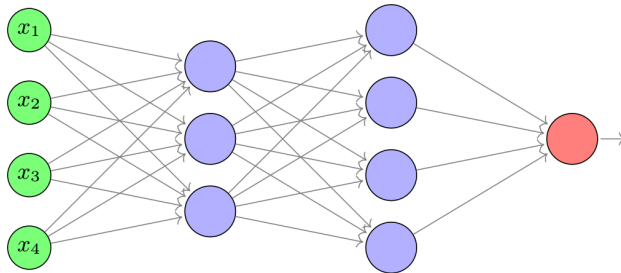
In practice, more layers helps.

- **Multi-layers Neural Networks (Deep Neural Network.)**

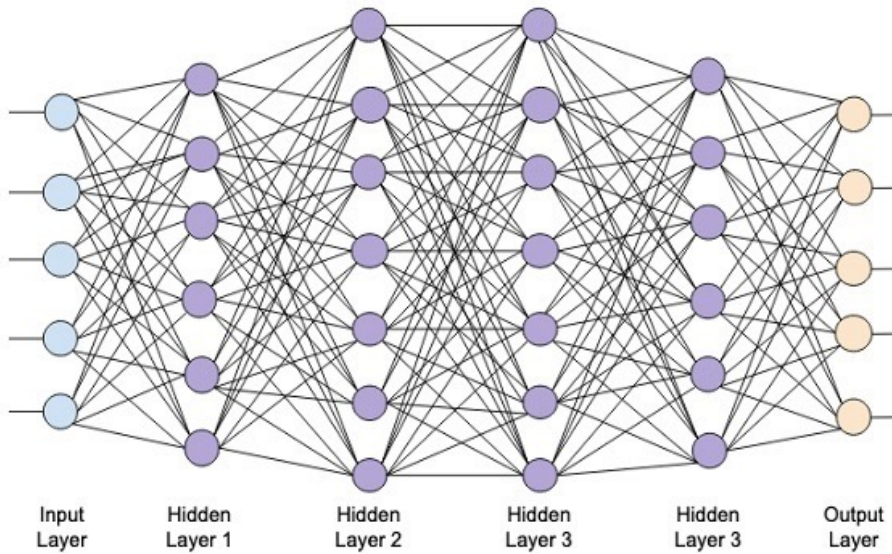


□ Classification:

- **Binary:**



- **Multi-class**



Summary:

➤ **Back-propagation** (Reverse autodifferentiation)

In 1986, (*Learning representations by back-propagating errors, Nature, 323(9): 533-536*) D. E. Rumelhart popularized the idea of **back propagation** to compute gradients. It is not a learning method, but a computational trick. It is actually a simple implementation of chain rule of derivatives.

BP algorithms as stochastic gradient descent algorithms (Robbins–Monro 1950; Kiefer- Wolfowitz 1951) with Chain rules of Gradient maps

Goal: Minimize the loss function J

Need to calculate the gradient.

➤ **The Chain Rule:**

Neural Network Coding:

1. MATLAB Neural Network (Deep Learning Toolbox).

<https://www.mathworks.com/products/deep-learning.html>

2. Python TensorFlow: <https://www.tensorflow.org/> (TensorBoard visualization)

Keras on TensorFlow: <https://keras.io/examples/>

3. Python PyTorch: <https://pytorch.org/>

4. R <https://www.r-project.org/>. (neuralnet library)

MATLAB example:

```
layers = [  
    featureInputLayer(20)  
    fullyConnectedLayer(30)  
    reluLayer  
    fullyConnectedLayer(15)  
    reluLayer  
    fullyConnectedLayer(3)  
    softmaxLayer  
    classificationLayer];
```

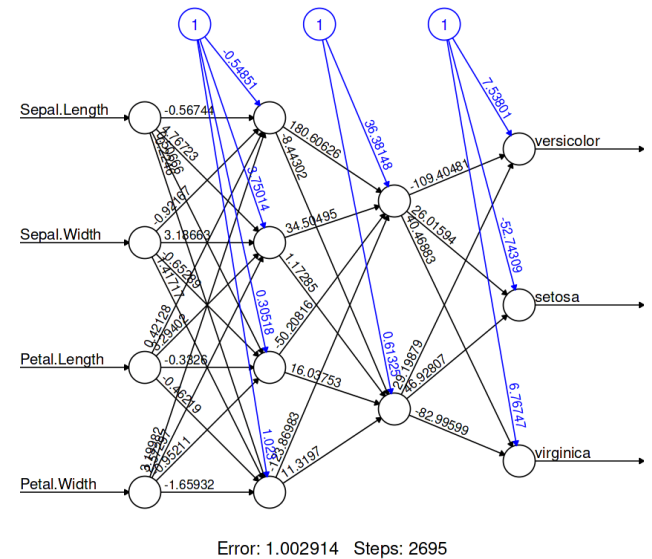
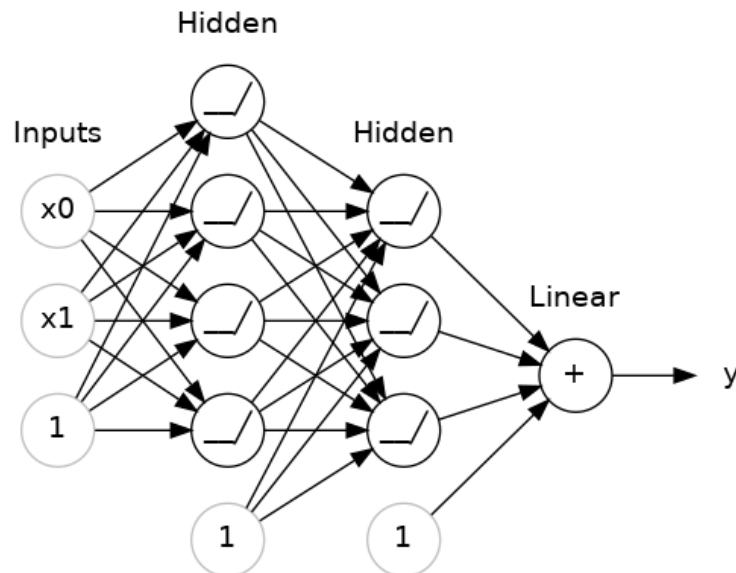
Python Example

```
from tensorflow import keras  
from tensorflow.keras import layers  
  
model = keras.Sequential([  
    # the hidden ReLU layers  
    layers.Dense(units=4, activation='relu', input_shape=[2]),  
    layers.Dense(units=3, activation='relu'),  
    # the linear output layer  
    layers.Dense(units=1),  
])
```

R Example

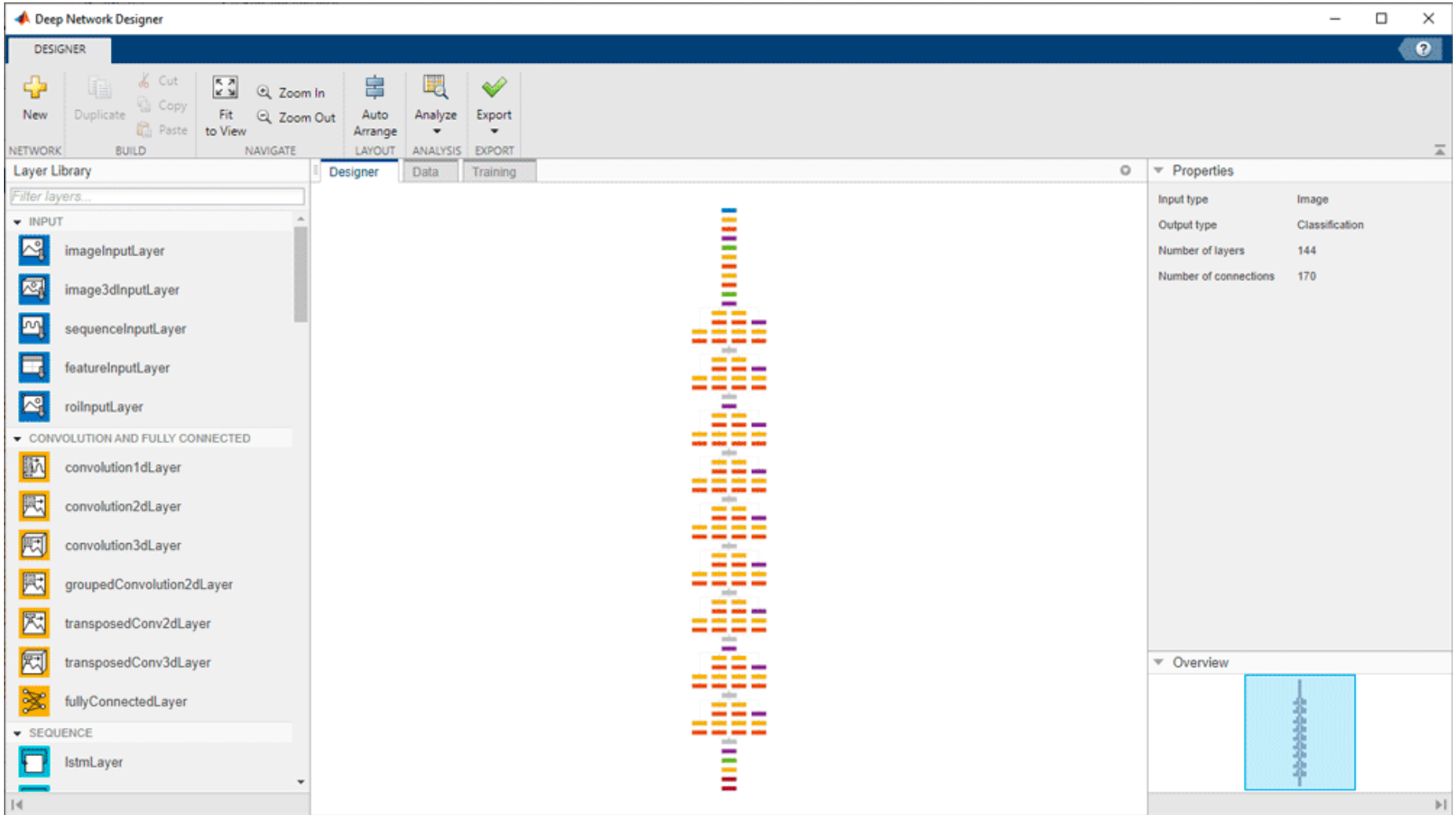
```
library(neuralnet)

model = neuralnet(
  Species~Sepal.Length+Sepal.Width+Petal.Length+Petal.Width,
  data=train_data,
  hidden=c(4,2),
  linear.output = FALSE
)
```



❑ Network Design and Model Management

MATLAB Deep Learning Toolbox



TensorBoard:

tensorboard.dev/experiment/EvNO346IT0iYbmeaWmoNCQ/#scalars&tagFilter=loss

TensorBoard.dev SCALARS

Show data download links
 Ignore outliers in chart scaling
Tooltip sorting method: default

Smoothing: 0.6

Horizontal Axis: STEP (selected), RELATIVE, WALL

Runs

Write a regex to filter runs

- cnn_dailymail_v002
- glue_v002_proportional
- pretrain
- squad_v010_allanswers
- super_glue_v102_proportional
- wmt15_enfr_v003
- wmt16_enro_v003
- wmt_t2t_ende_v003

TOGGLE ALL RUNS

experiment EvNO346IT0iYbmeaWmoNCQ

loss

Tags matching /loss/

eval

PREVIOUS PAGE

cnn_dailymail_v002/rouge1
tag: eval/cnn_dailymail_v002/rouge1

cnn_dailymail_v002/rouge2
tag: eval/cnn_dailymail_v002/rouge2

➤ Dropout and Batch Normalization

1. Dropout layer can help correct overfitting. We randomly *drop out* some fraction of a layer's input units every step of training. The weight patterns tend to be more robust.

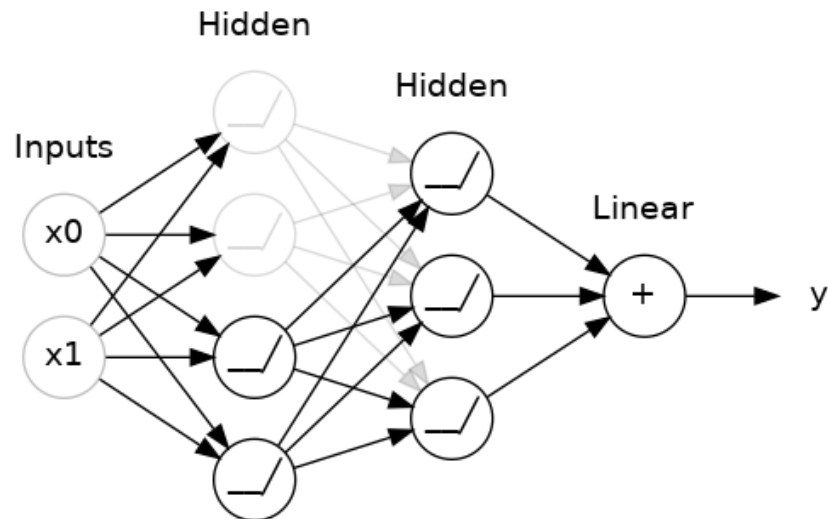
2. Batch Normalization is something like scikit-learn's [StandardScaler](#) or [MinMaxScaler](#).

Batch normalization layer looks at each batch as it comes in, first normalizing the batch with its own mean and standard deviation, and then also putting the data on a new scale with two trainable rescaling parameters.

Batch normalization, in effect, performs a kind of coordinated rescaling of its inputs.

```
from tensorflow import keras
from tensorflow.keras import layers
```

```
model = keras.Sequential([
    # the hidden ReLU layers
    layers.Dense(units=4, activation='relu', input_shape=[2]),
    layers.Dropout(0.3), # apply 30% dropout to the next layer
    layers.BatchNormalization(),
    layers.Dense(units=3, activation='relu'),
    layers.Dropout(0.3), # apply 30% dropout to the next layer
    layers.BatchNormalization(),
    # the linear output layer
    layers.Dense(units=1),
])
```



➤ Early Stopping

```
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import layers, callbacks

early_stopping = EarlyStopping(
    min_delta=0.001, # minimum amount of change to count as an improvement
    patience=20, # how many epochs to wait before stopping
    restore_best_weights=True,
)
```

These parameters say: "If there hasn't been at least an improvement of 0.001 in the validation loss over the previous 20 epochs, then stop the training and keep the best model you found."

It can sometimes be hard to tell if the validation loss is rising due to overfitting or just due to random batch variation. The parameters allow us to set some allowances around when to stop.

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

```
history = model.fit(X_train, y_train, epochs=30,
                   validation_data=(X_valid, y_valid))
```

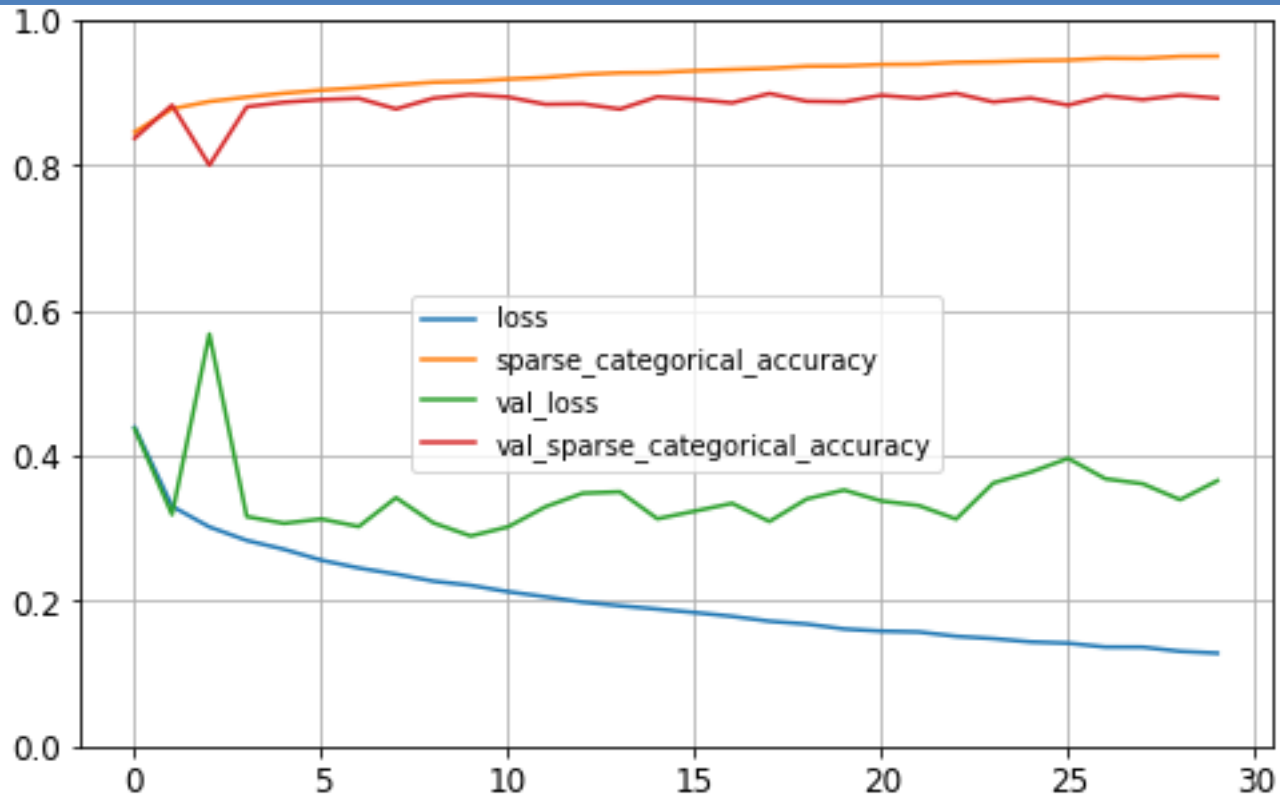
```
import pandas as pd
```

```
pd.DataFrame(history.history).plot(figsize=(8, 5))
```

```
plt.grid(True)
```

```
plt.gca().set_ylim(0, 1)
```

```
plt.show()
```



A visual proof that neural nets can compute any function

<http://neuralnetworksanddeeplearning.com/chap4.html>

Play with neural network:

<http://playground.tensorflow.org/>

<https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>

Online book about neural network:

<http://neuralnetworksanddeeplearning.com/chap3.html>

MIT Introduction to Deep Learning | 6.S191

https://www.youtube.com/watch?v=5tvmMX8r_OM

○ **MATLAB Resources:**

1. Matlab Neural Network Toolbox:

<https://www.mathworks.com/products/deep-learning.html>

2. Matlab Examples:

<https://www.mathworks.com/help/deeplearning/examples.html?category=getting-started-with-deep-learning-toolbox>

3. Get Started with Deep Learning Toolbox

<https://www.mathworks.com/help/deeplearning/getting-started-with-deep-learning-toolbox.html>

For example:

<https://www.mathworks.com/help/deeplearning/gs/create-simple-image-classification-network-using-deep-network-designer.html>