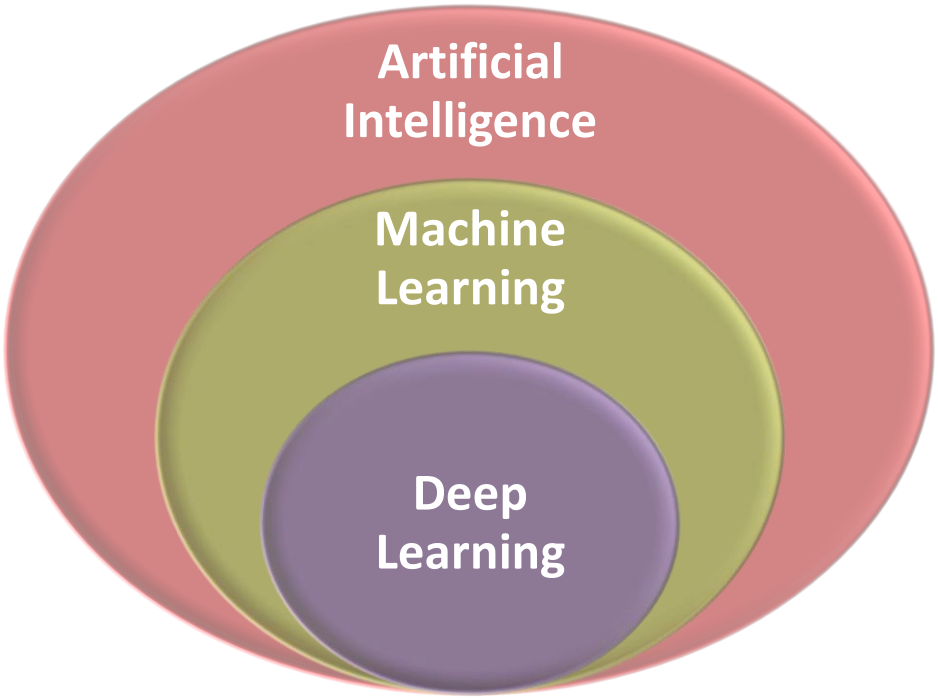
Deep Learning

Deep Learning

- Deep learning is an immensely rich subfield of machine learning, with powerful applications ranging from machine perception to natural language processing, all the way up to creative AI.



Artificial Intelligence, Machine Learning and Deep Learning

Artificial Intelligence

- A concise definition of **AI** would be:

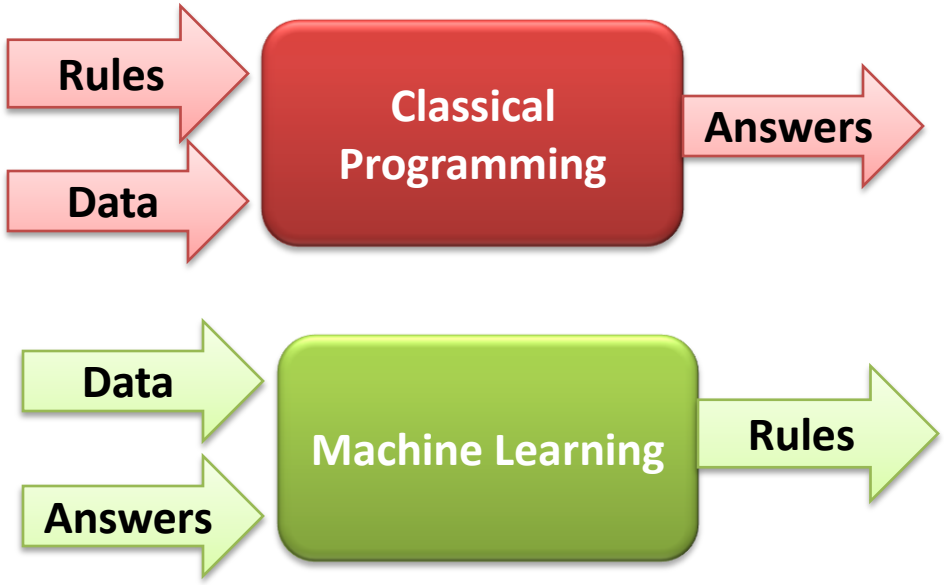
“ the effort to automate intellectual tasks normally performed by humans. “

- AI is a very general field which encompasses machine learning and deep learning, but also includes many more approaches that do not involve any learning.

Machine Learning

- With Machine Learning, humans would input data as well as the answers expected from the data, and out would come the rules.
- These rules could then be applied to new data to produce original answers.

A machine learning system is "trained" rather than explicitly programmed. It is presented with many "examples" relevant to a task, and it finds statistical structure in these examples which eventually allows the system to come up with rules for automating the task.



To do machine learning, we need three things:

- Input data points.
- Examples of the expected output.
- A way to measure if the algorithm is doing a good job, to measure the distance between its current output and its expected output.

  This is used as a feedback signal to adjust the way the algorithm works.

  This adjustment step is what we call "learning".

- **Machine learning is, technically:**

  Searching for useful representations of some input data, within a pre-defined space of possibilities, using guidance from some feedback signal. This simple idea allows for solving a remarkably broad range of intellectual tasks, from speech recognition to autonomous car driving.

Deep Learning

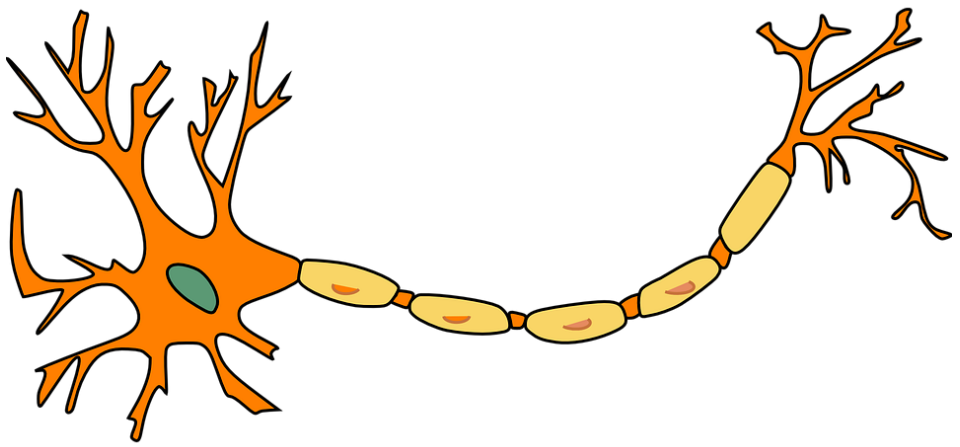
- Deep learning is a specific subfield of machine learning, a new take on learning representations from data which puts an emphasis on learning successive "layers" of increasingly meaningful representations.
- The "deep" in "deep learning" simply stands for this idea of successive layers of representations—how many layers contribute to a model of the data is called the "depth" of the model.

Deep Learning

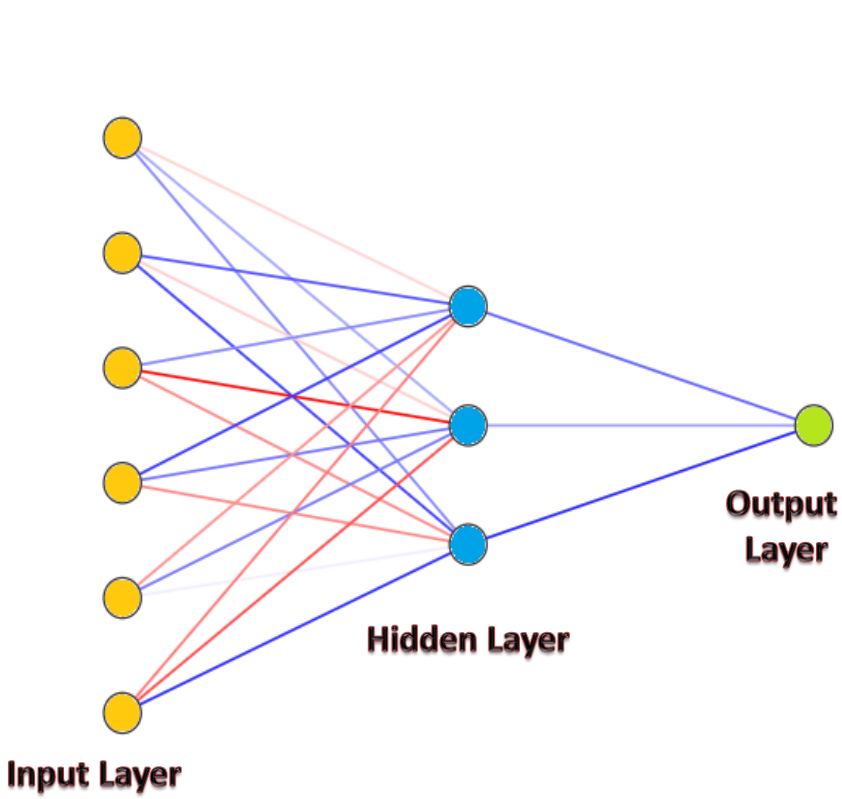
- Modern deep learning often involves tens or even hundreds of successive layers of representation and they are all learned automatically from exposure to training data.
- Other approaches to machine learning tend to focus on learning only one or two layers of representation of the data, sometimes called **"shallow learning".**

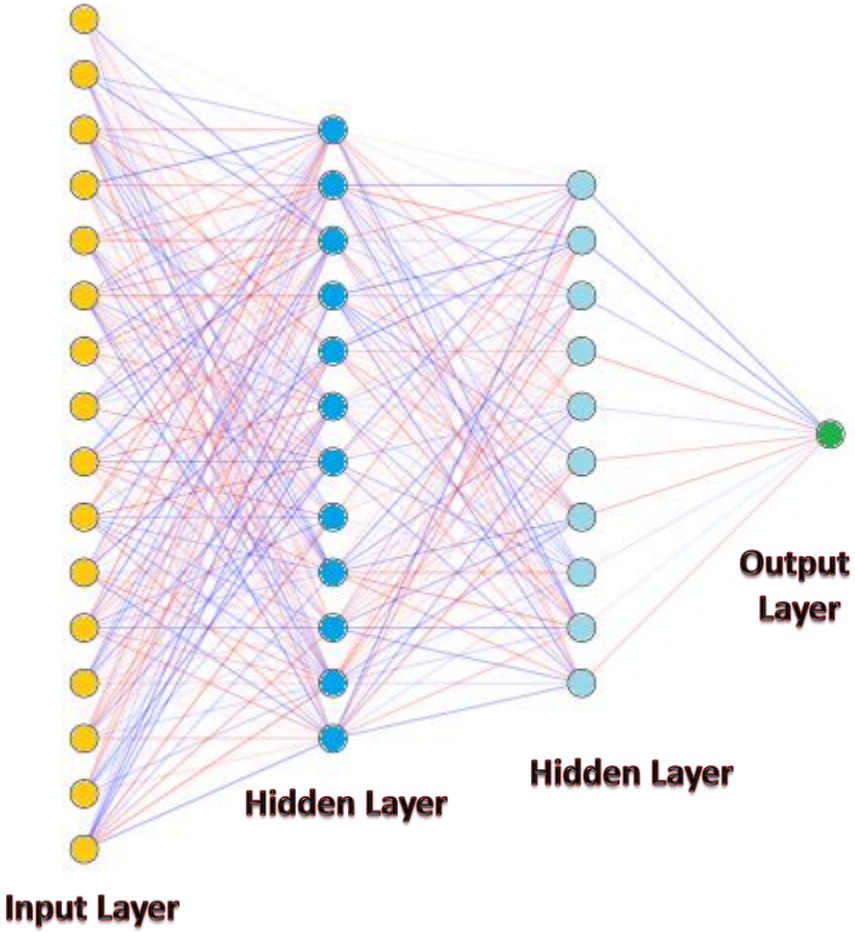
Deep Learning

- In deep learning, these layered representations are learned via models called "neural networks", structured in literal layers stacked one after the other.
- The term "neural network" is a reference to neurobiology.



Deep Neural Networks

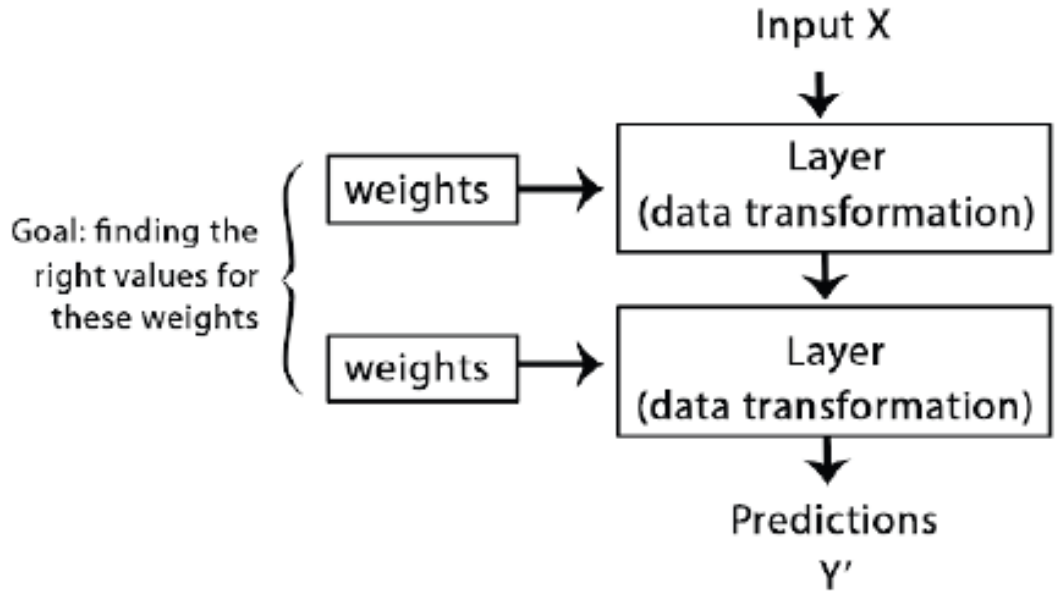




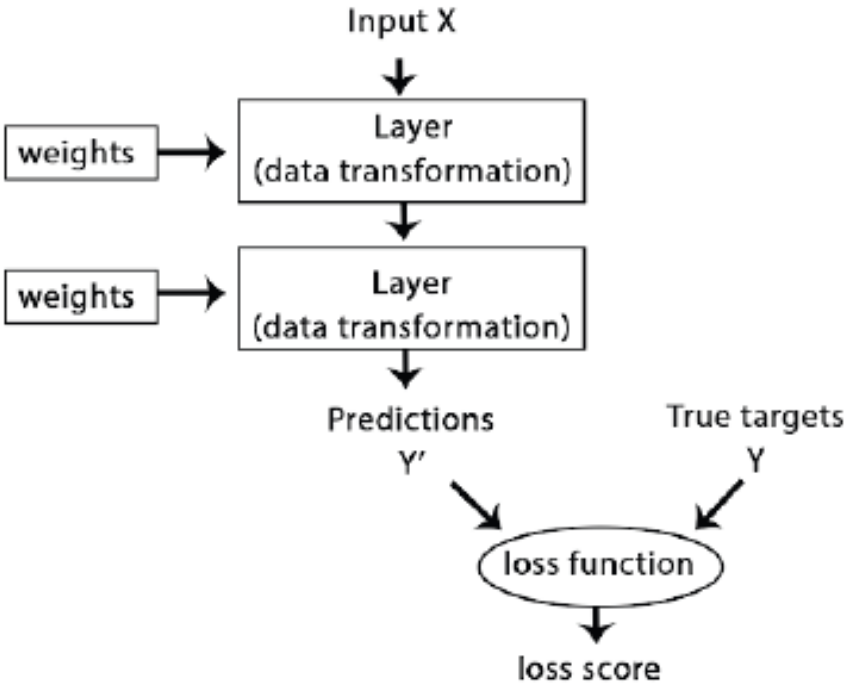
Understanding how deep learning works

- Deep neural networks do input-to-target mapping via a deep sequence of simple data transformations called "layers", and that these data transformations are learned by exposure to examples.

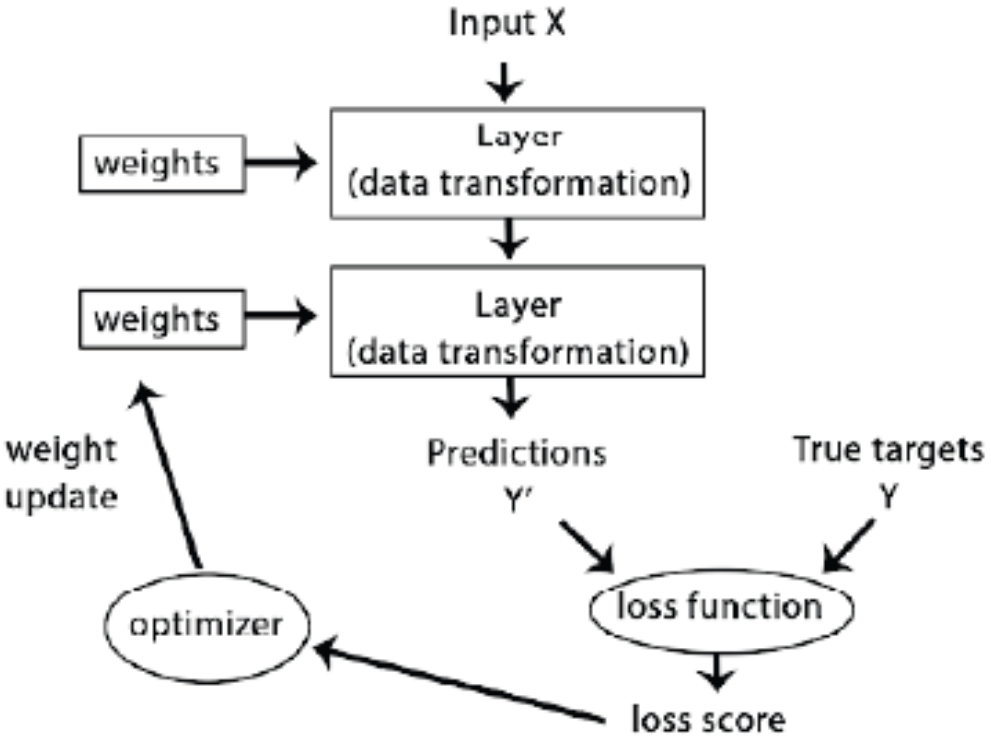
- The specification of what a layer does to its input data is stored in the layer's **"weights"**, which in essence are a bunch of numbers.
- Weights are also sometimes called the **"parameters"** of a layer.
- **"learning"** will mean finding a set of values for the weights of all layers in a network, such that the network will correctly map your example inputs to their associated targets.



- To control the output of a neural network, you need to be able to measure how far this output is from what you expected.
- This is the job of the **"loss function"** of the network, also called **"objective function".**
- The loss function takes the predictions of the network and the true target , and computes a distance score, capturing how well the network has done on this specific example.



- The fundamental trick in deep learning is to use the loss score as a feedback signal to adjust the value of the weights by a little bit, in a direction that would lower the loss score for the current example.
- This adjustment is the job of the "optimizer", which implements what is called the **"backpropagation"** algorithm, the central algorithm in deep learning.



What deep learning has achieved so far

- Deep learning has achieved the following breakthroughs, all in historically difficult areas of machine learning:

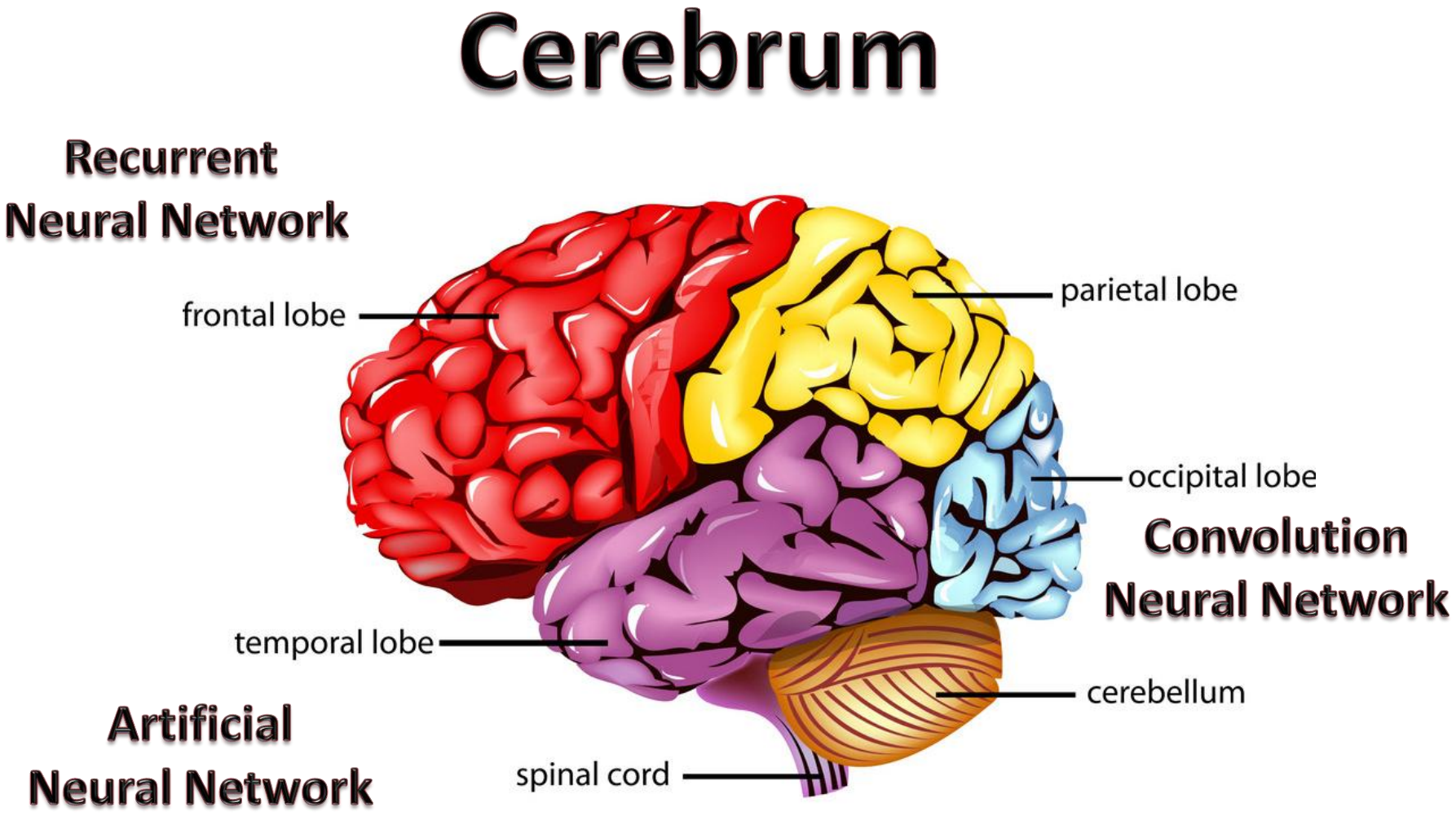
✓ *Near-human level image classification.*
✓ *Near-human level speech recognition.*
✓ *Near-human level handwriting transcription.*
✓ *Improved text-to-speech conversion.*
✓ *Digital assistants such as Amazon Alexa.*
✓ *Improved ad targeting, as used by Google, Bing.*
✓ *Improved search results on the web.*

What makes deep learning different

- Deep learning took off so quickly is primarily that it offered better performance on many problems.
- Deep learning is also making problem-solving much easier, because it completely automates what used to be the most crucial step in a machine learning workflow:

"feature engineering".

Human Brain


Cerebrum
Recurrent
Neural Network
frontal lobe
parietal lobe
occipital lobe
Convolution
Neural Network
temporal lobe
cerebellum
Artificial
Neural Network
spinal cord

Input Layer

- This layer accepts input features.
- It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information(features) to the hidden layer.

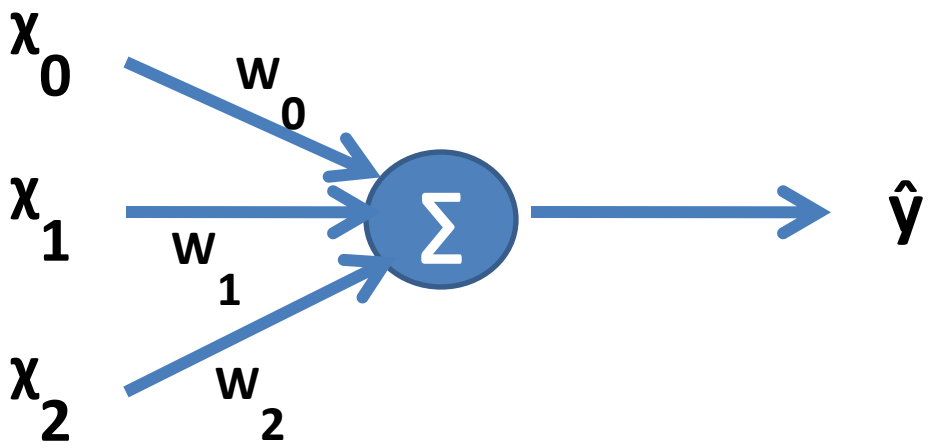
Hidden Layer

- Nodes of this layer are not exposed to the outer world, they are the part of the abstraction provided by any neural network.
- Hidden layer performs all sort of computation on the features entered through the input layer and transfer the result to the output layer.

Output Layer

- This layer returns the final output computed by the network to the application.

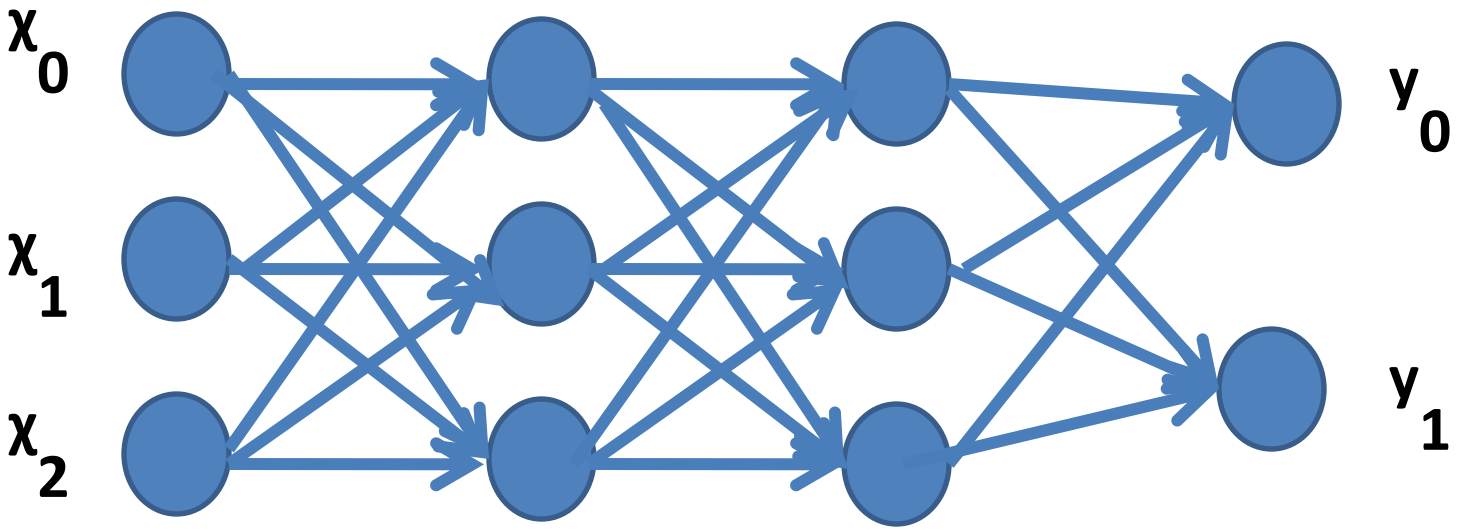
Neural Network with Single Neuron



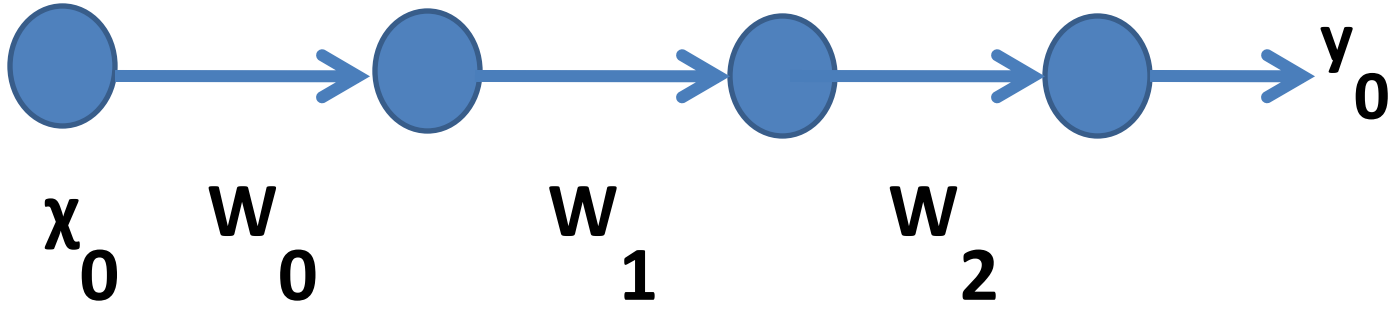
$$\hat{y} = x_0 w_0 + x_1 w_1 + x_2 w_2 = \sum_i x_i w_i$$

$$\begin{pmatrix} w_0 & w_1 & w_2 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$$

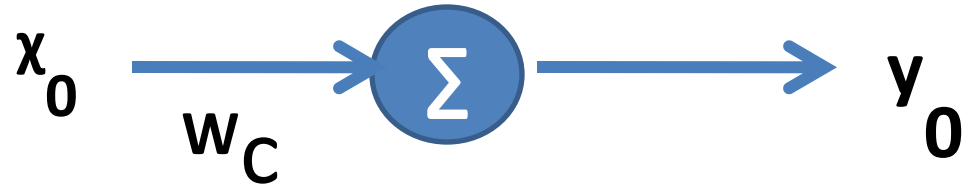
Muti-layer Perceptron



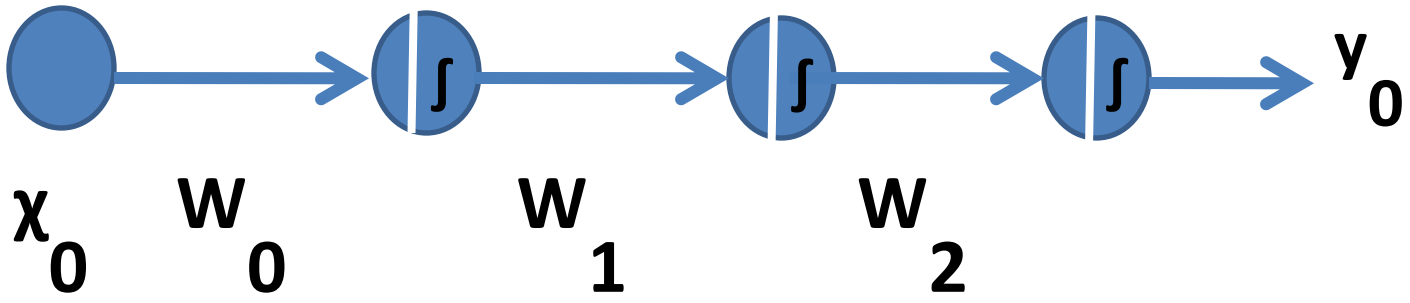
Linear Function



$$y_0 = x_0 w_0 w_1 w_2 = x_0 w_C$$

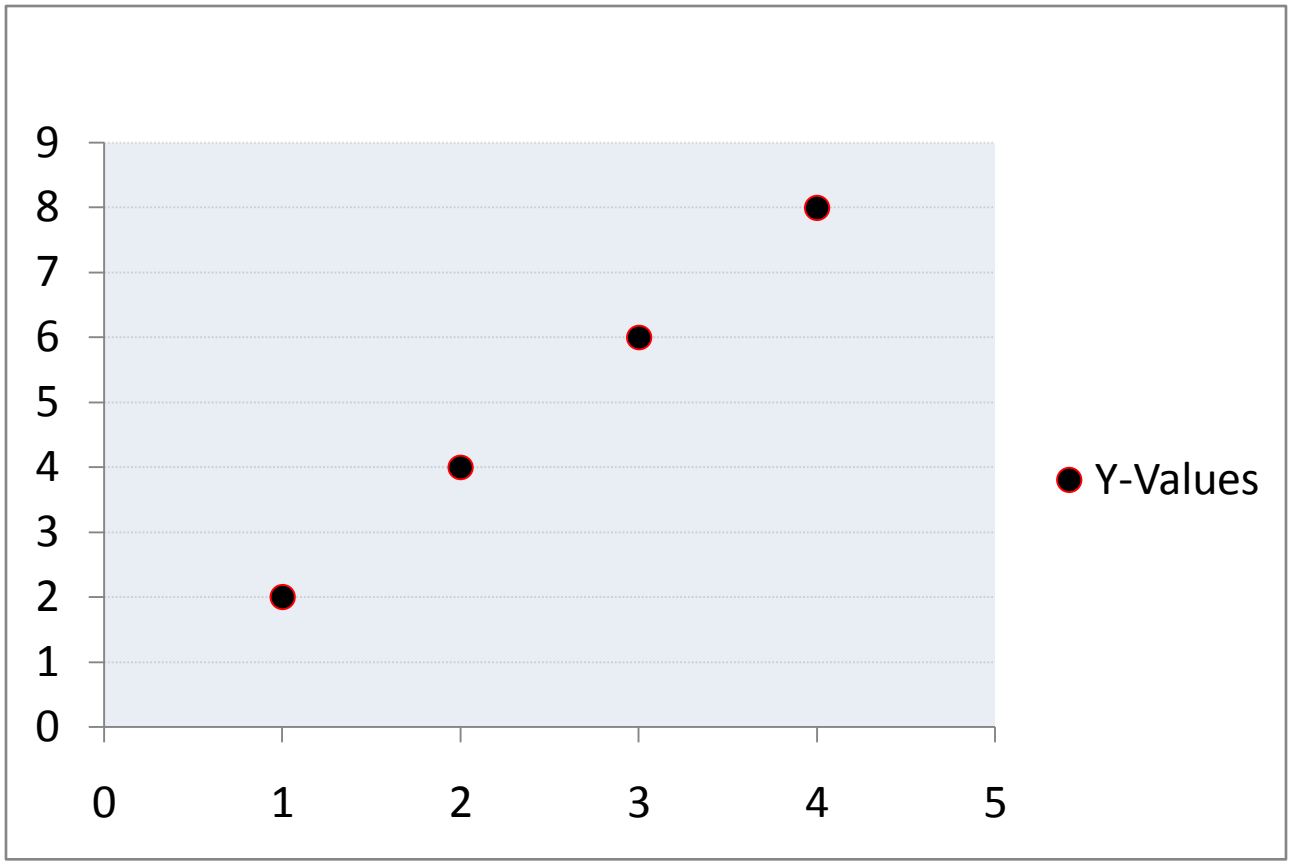


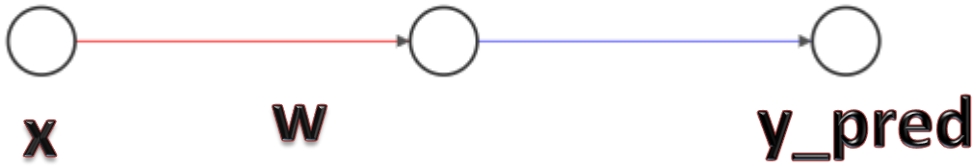
Non-Linear Function



$$y_0 = \sigma(\sigma(\sigma(x_0 W_0) W_1) W_2)$$

Simple Example





y_pred = xw

E = $(\hat{y}-y)^2 = (xw-y)^2$ *mean squared Error*

$$\frac{\partial \mathbf{E}}{\partial \mathbf{w}} = 2x(xw-y)$$ Derivative

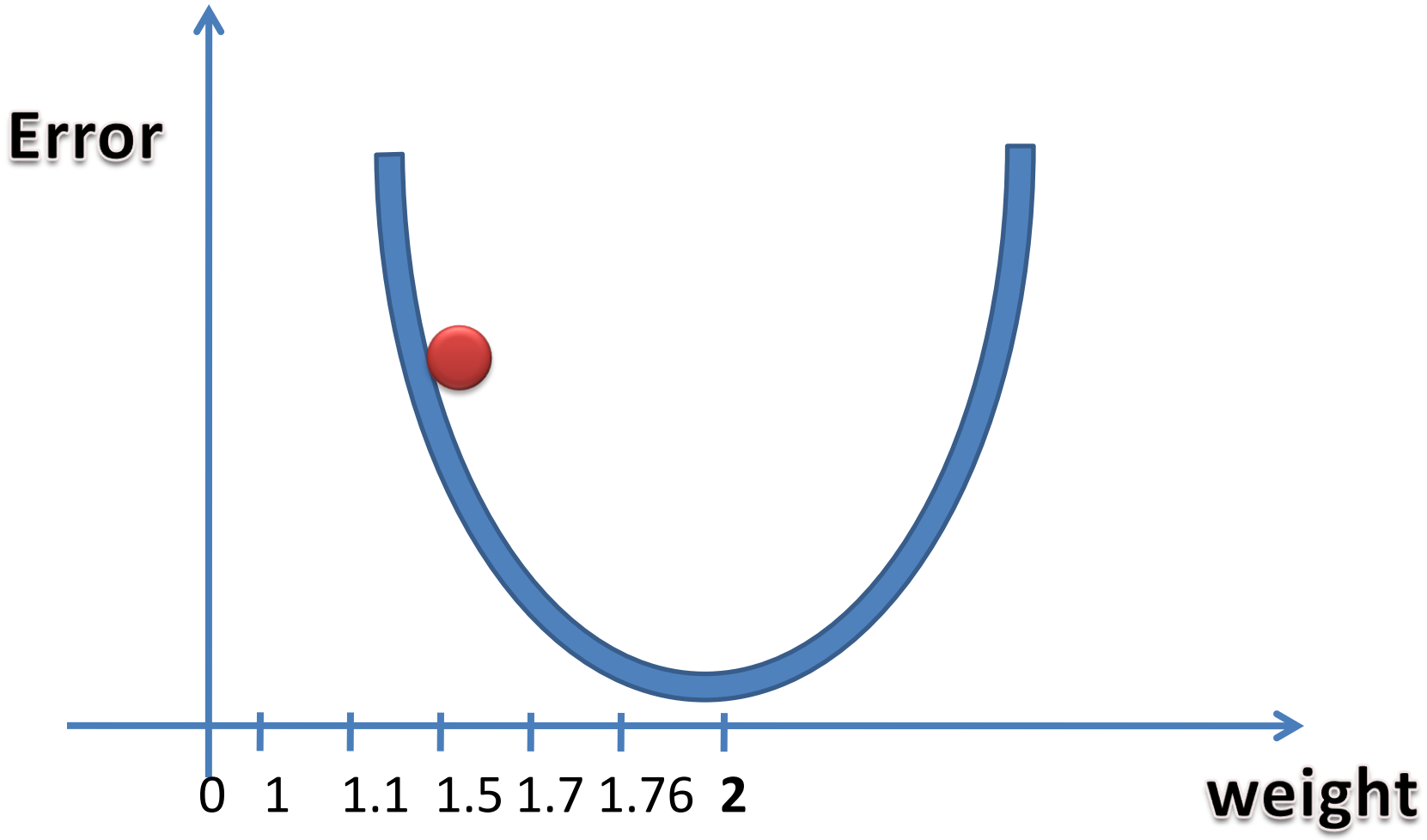
$$\mathbf{w} - \alpha \frac{\partial \mathbf{E}}{\partial \mathbf{w}} = \mathbf{w} - \alpha\, \mathbf{2x(xw-y)}$$ Update Rule

- Random weight initialization w=0.5
- learning rating $\alpha = 0.1$

- $w_{new} = w - 0.1 * 2x(xw-y)$
- (x,y)
- (2,4) w=0.5 ← 0.5 - 0.1*2*2(2*0.5-4) = 1.7
- (1,2) w =1.7 ← 1.7-0.1*2*1(1*1.7-2) = 1.76
- (3,6) w= 1.76 ← 1.76-0.1*2*3(3*1.76-6)=2.192
- w ~ 2

Oversimplified Gradient Descent:

- Calculate slope at current position
- If slope is negative, move right
- If slope is positive, move left
- (Repeat until slope == 0)



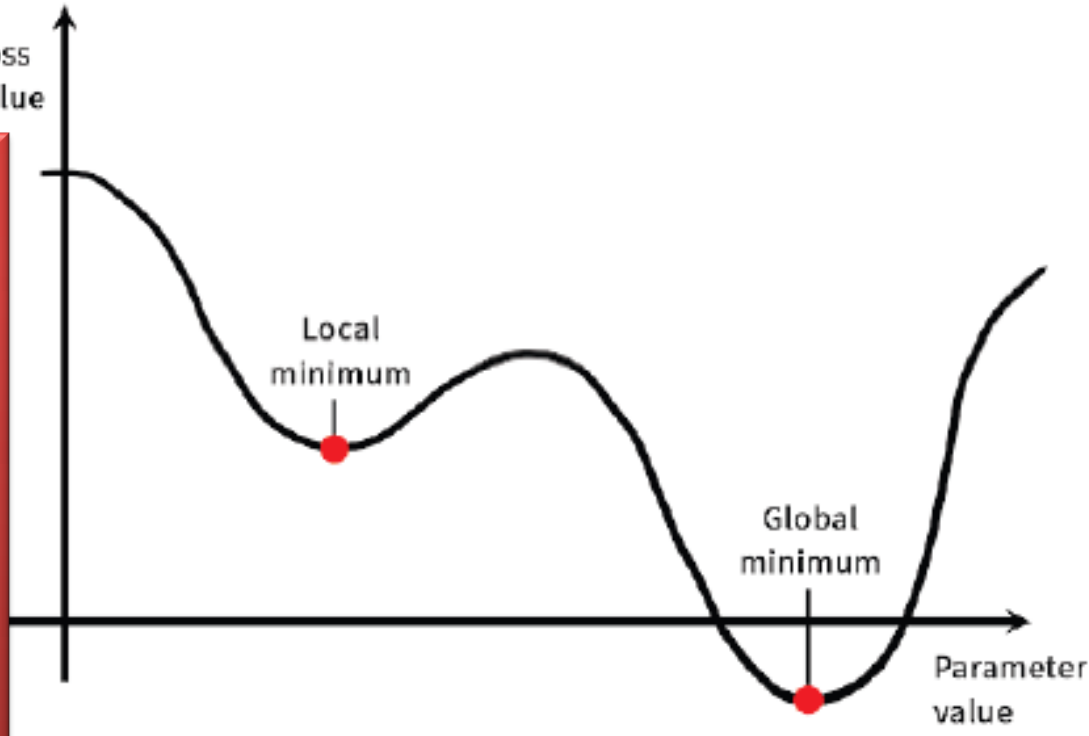
SGD Variants

- There exists multiple variants of SGD that differ by taking into account previous weight updates when computing the next weight update, rather than just looking at the current value of the gradients.
- "SGD with momentum", "Adagrad", "RMSprop“ these variants are known as **"optimization methods" or "optimizers"**.

Momentum

- Momentum addresses two issues with SGD: convergence speed, and local minima.

If the parameter considered was being optimized via SGD with a small learning rate, then the optimization process would get stuck at the local minimum, instead of making its way to the global minimum.



Momentum

- A way to avoid such issues is to use "momentum", which draws inspiration from physics.
- This means updating the parameter **w** not only on the current gradient value but also based on the previous parameter update.

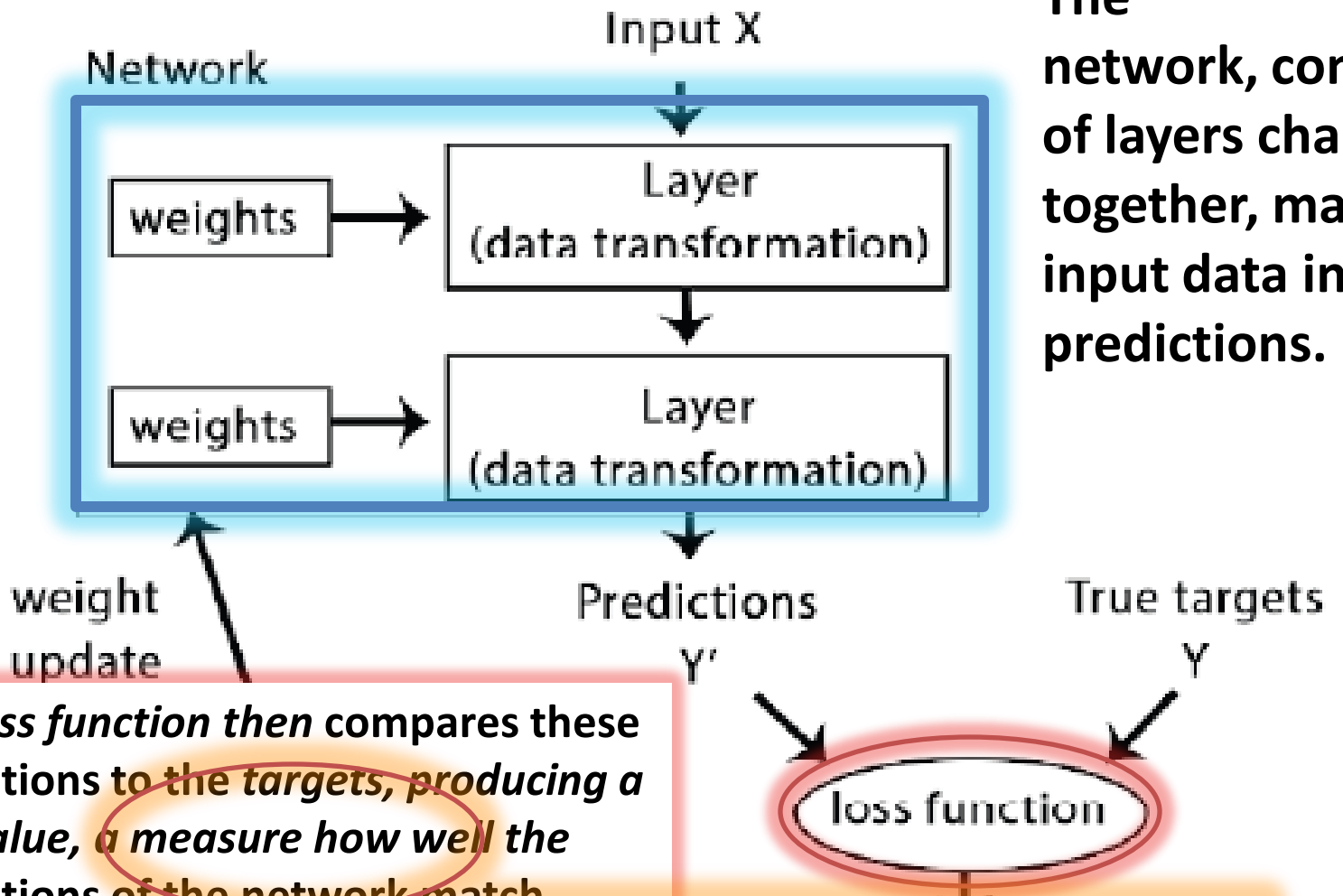
Momentum

past_velocity = 0.
momentum = 0.1 # A constant momentum factor

while loss > 0.01: # Optimization loop
    w, loss, gradient = get_current_parameters()
    velocity = **past_velocity** * **momentum** + learning_rate * gradient
    w = w + **momentum** * **velocity** - learning_rate * gradient
    past_velocity = velocity
    update_parameter(w)

Anatomy of a neural network

- Training a neural network revolves around the following objects:
- *Layers, which are combined into a network (or model).*
- The *input data and corresponding targets.*
- The *loss function, which defines the feedback signal which is used for learning.*
- The *optimizer, which determines how the learning proceeds.*



The network, composed of layers chained together, maps the input data into predictions.

The *loss function then* compares these predictions to the *targets, producing a loss value, a measure how well the* predictions of the network match what was expected.

The *optimizer uses this loss value* to update the weights of the network.

Activation Function

- Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it.

- The purpose of the activation function is to *introduce non-linearity* into the output of a neuron.

Activation Function

- Neural network has neurons that work in correspondence of *weight, bias* and their respective activation function.
- In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output.
- This process is known as *back-propagation*.
- Activation functions make the back-propagation possible

Activation Function

- **Sigmoid Function :**
- Usually used in output layer of a binary classification, where result is either 0 or 1
- As value for sigmoid function lies between 0 and 1 only
- Result can be predicted easily to be 1 if value is greater than 0.5 and 0 otherwise.

Activation Function

- **tanh() Function :**
- Hyperbolic tangent function (also known as tanh)
- The hyperbolic tangent function outputs in the range (-1, 1), thus mapping strongly negative inputs to negative values.

Activation Function

- **RELU (***Rectified linear unit***)**

- It is the most widely used activation function.

- ReLu is less computationally expensive because it involves simpler mathematical operations.

Gradient Descent

- Gradient Descent is used while training a machine learning model.
- A gradient measures how much the output of a function changes if you change the inputs a little bit.
- It simply measures the change in all weights with regard to the change in error.

Types of Gradient descents:

- **Batch Gradient Descent:** Parameters are updated after computing the gradient of error with respect to the entire training set

- **Stochastic Gradient Descent:** Parameters are updated after computing the gradient of error with respect to a single training example

- **Mini-Batch Gradient Descent:** Parameters are updated after computing the gradient of error with respect to a subset of the training set

Optimization techniques for Gradient Descent

- **Momentum method**: This method is used to accelerate the gradient descent algorithm by taking into consideration the exponentially weighted average of the gradients.

- **new weight ←(old weight)-(learning rate)(gradient)**
- **new weight ←(old weight)-(learning rate)(gradient) + past gradient**
- **(accumulator)←(old accumulator)(momentum)+gradient**
- **momentum → weighted average of past gradients**
- **new weight ←(old weight)-(learning rate)(accumulator)**

..

- **RMSprop**: Root Mean Square Propagation
- RMSprop is another adaptive method which retains the learning rate for each parameter but uses a moving average over the gradients to make optimization more suited for more non-convex optimization

- **Adam optimizer**: Another algorithm that uses adaptive method. Adam stands for Adaptive momentum estimation, it tends to combine the best part of RMSprop & momentum optimizer

Loss Functions

- **Regression Loss Functions**
- **Mean Squared Error Loss**
  - Mean squared error is calculated as the average of the squared differences between the predicted and actual values.
  - ‘*mean_squared_error*‘
- **Mean Absolute Error Loss**
  - ‘*mean_absolute_error*‘

- **Binary Classification Loss Functions**
- **Binary Cross-Entropy Loss**
  - It is intended for use with binary classification where the target values are in the set {0, 1}
  - ‘*binary_crossentropy*‘
- **Hinge Loss**
  - It is intended for use with binary classification where the target values are in the set {-1, 1}.
  - ‘hinge‘

Keras

- Keras, the Python deep learning library
- one of the most widely used deep learning frameworks
- A big part of that success is that Keras has always put ease of use and accessibility front and center.

Keras

- Keras has the following key features:

✓ It allows the same code to run on CPU or on GPU, seamlessly.

✓ It has a user-friendly API which makes it easy to quickly prototype deep learning models.

✓ It has build-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.

Keras

- There are two ways to define a model
- Using the “**Sequential class** “

  (only for linear stacks of layers, which is the most common network architecture by far)

- Using the **"functional API“**

  (for directed acyclic graphs of layers, allowing to build completely arbitrary architectures)

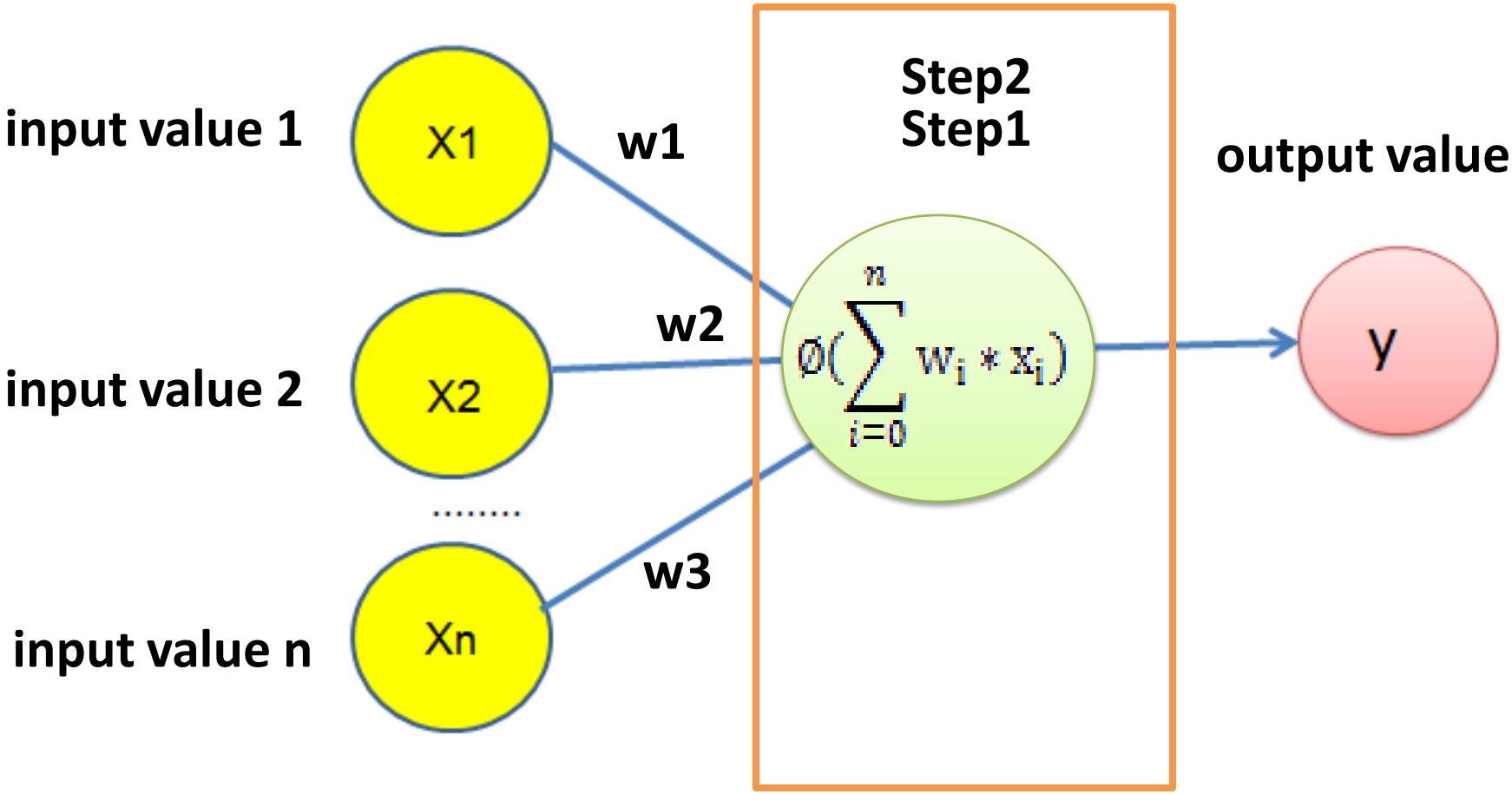
A network definition using the Sequential model

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(784,)))
model.add(layers.Dense(10, activation='softmax'))
```

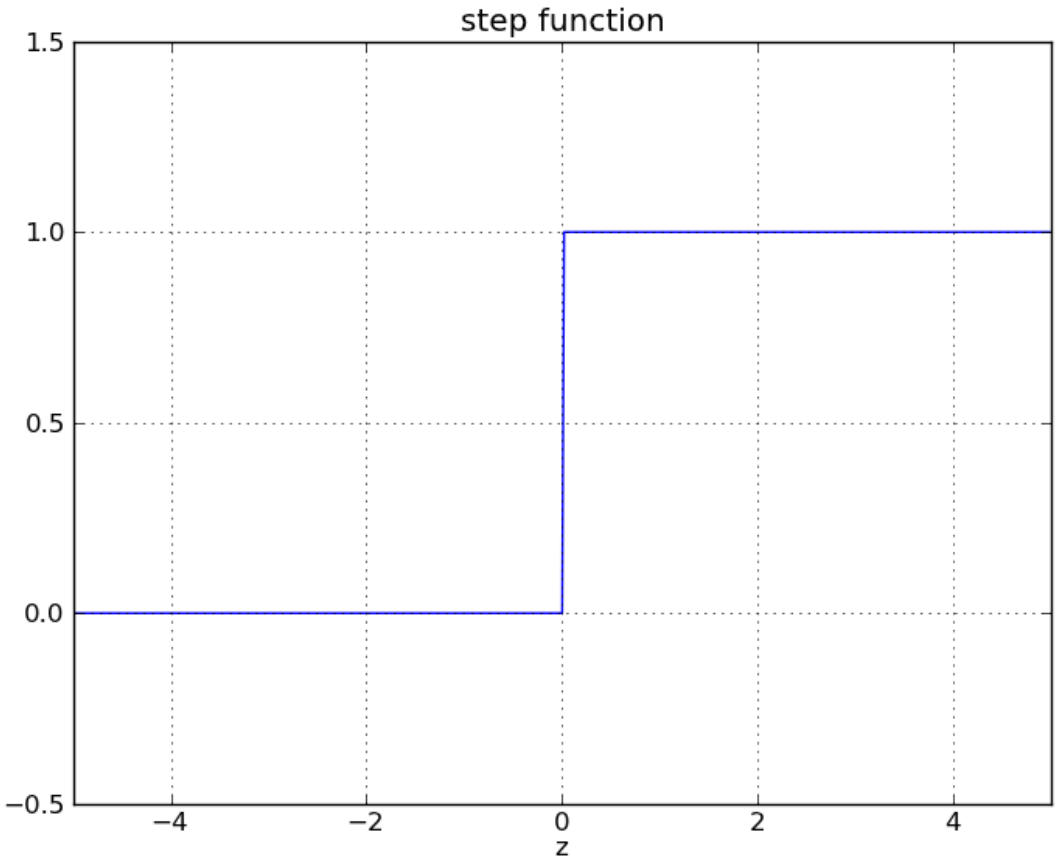
A network definition using the functional API

```
input_tensor = layers.Input(shape=(784,))
x = layers.Dense(32, activation='relu')(input_tensor)
output_tensor = layers.Dense(10, activation='softmax')(x)

model = models.Model(input=input_tensor, output=output_tensor)
```

ANN


input value 1
input value 2
input value n
X1
X2
Xn
w1
w2
w3
Step2
Step1
$\emptyset(\sum_{i=0}^{n} w_i * x_i)$
output value
y

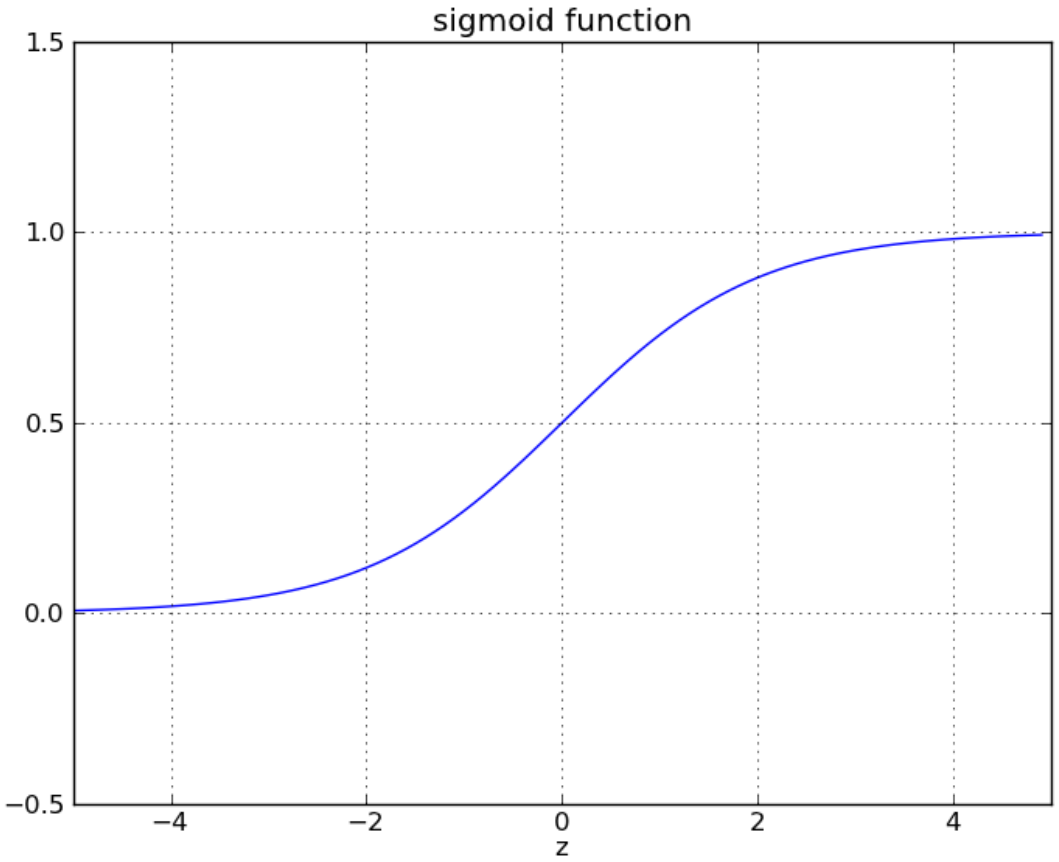
Threshold Function



$$\phi(x) = \begin{cases} 1 \text{ if } x >= 0 \\ 0 \text{ if } x < 0 \end{cases}$$

$$\sum_{i=0}^{n} w_i * x_i$$

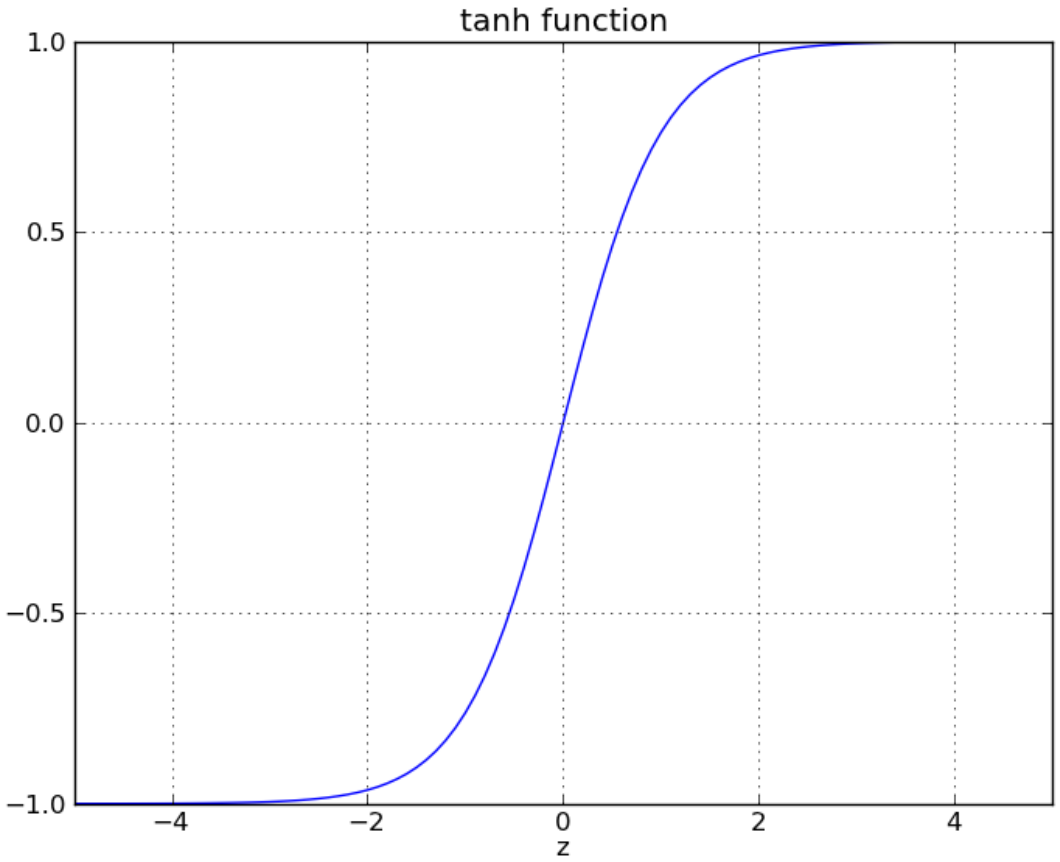
Sigmoid



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

$$\sum_{i=0}^{n} w_i * x_i$$

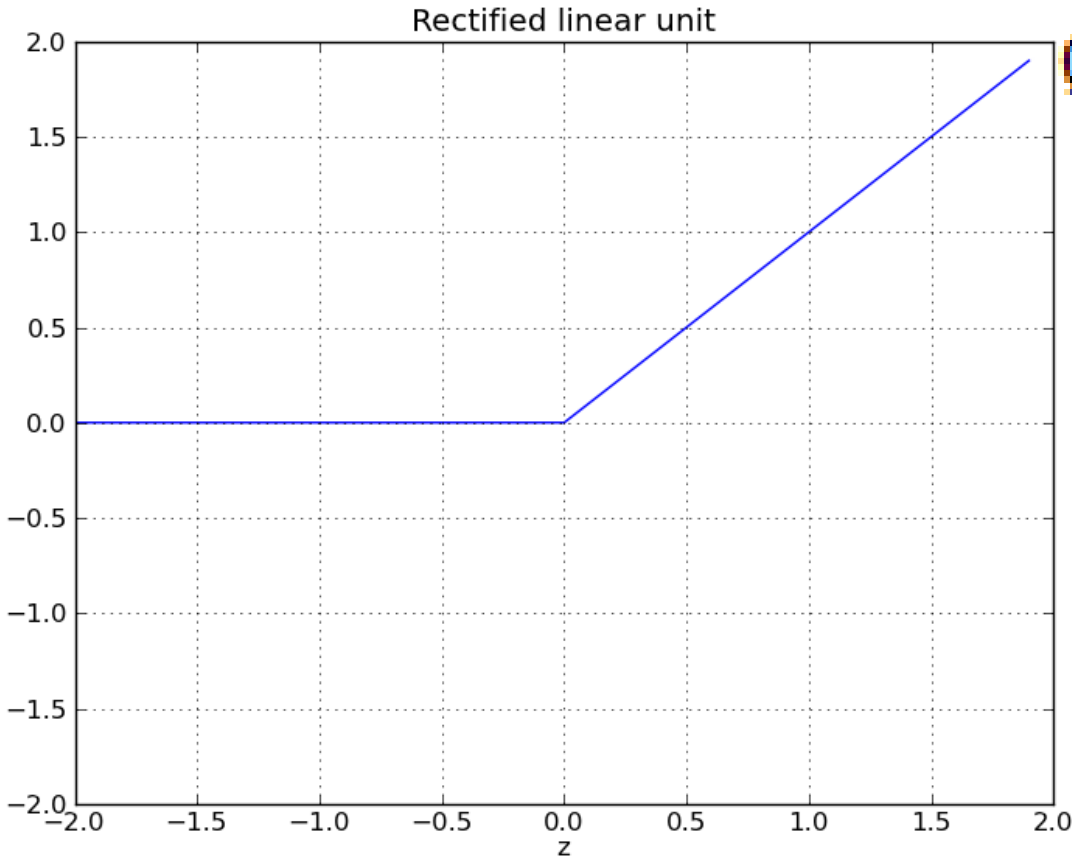
tanh



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

$$\sum_{i=0}^{n} w_i * x_i$$

ReLu



$\emptyset(x)$ = *max(x,0)*

$$\sum_{i=0}^{n} w_i * x_i$$

Overfitting

- When the accuracy of our model on the validation data would peak after training for a number of epochs, and would then start decreasing. In other words, our model would *overfit* to the training data
- It's often possible to achieve high accuracy on the *training set*, what we really want is to develop models that generalize well to a *testing set* (or data they haven't seen before).

Underfitting

- The opposite of overfitting is *underfitting*.
- If the model is not powerful enough, or has simply not been trained long enough.
- This means the network has not learned the relevant patterns in the training data.

Overfitting

- If we train for too long though, the model will start to overfit and learn patterns from the training data that don't generalize to the test data
- We need to strike a balance

Prevent overfitting

- To prevent overfitting, the best solution is to use more training data
- A model trained on more data will naturally generalize better.
- When that is no longer possible, the next best solution is to use techniques like regularization.

Regularization

- These place constraints on the quantity and type of information your model can store.
- If a network can only afford to memorize a small number of patterns, the optimization process will force it to focus on the most prominent patterns, which have a better chance of generalizing well.

Prevent overfitting

- The simplest way to prevent overfitting is to reduce the size of the model, i.e. the number of learnable parameters in the model (which is determined by the number of layers and the number of units per layer).
- Deep learning models tend to be good at fitting to the training data, but the real challenge is generalization, not fitting.

Prevent overfitting

- There is no magical formula to determine the right size or architecture of your model (in terms of the number of layers, or the right size for each layer).
- We will have to experiment using a series of different architectures.
- To find an appropriate model size, it's best to start with relatively few layers and parameters, then begin increasing the size of the layers or adding new layers until you see diminishing returns on the validation loss.

Regularization Techniques

- Two common regularization techniques—**weight regularization** and **dropout**

Weight regularization

- A common way to mitigate overfitting is to put constraints on the complexity of a network by forcing its weights only to take small values, which makes the distribution of weight values more "regular".
  - ***This is called "weight regularization“***

Weight regularization

- It is done by adding to the loss function of the network a cost associated with having large weights.
- This cost comes in two flavors:

➢ L1 regularization, where the cost added is proportional to the absolute value of the weights coefficients

➢ L2 regularization, where the cost added is proportional to the square of the value of the weights coefficients

Weight regularization

- keras.layers.Dense(16,

kernel_regularizer=keras.regularizers.l2(0.001),

activation="relu", input_shape=(10,))

l2(0.001) means that every coefficient in the weight matrix of the layer will add

0.001 * weight_coefficient_value2**

to the total loss of the network.

Dropout

- Dropout is one of the most effective and most commonly used regularization techniques for neural networks
- Dropout, applied to a layer, consists of randomly "dropping out" (i.e. set to zero) a number of output features of the layer during training.

Dropout

- The "dropout rate" is the fraction of the features that are being zeroed-out; it is usually set between 0.2 and 0.5.

```
keras.layers.Dense(16, activation=tf.nn.relu,
input_shape=(NUM_WORDS,))
    keras.layers.Dropout(0.5)
```

Summary

- The most common ways to prevent overfitting in neural networks:

- Get more training data.
- Reduce the capacity of the network.
- Add weight regularization.
- Add dropout.

Convolution Neural Network

- A convolutional neural network

  (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery.

Convolution Neural Network

- Facebook uses neural nets for their automatic tagging algorithms
- Google for their photo search
- Amazon for their product recommendations
- Pinterest for their home feed personalization
- Instagram for their search infrastructure.

Image classification

- Most popular, use case of these networks is for image processing.

How Humans identify images??

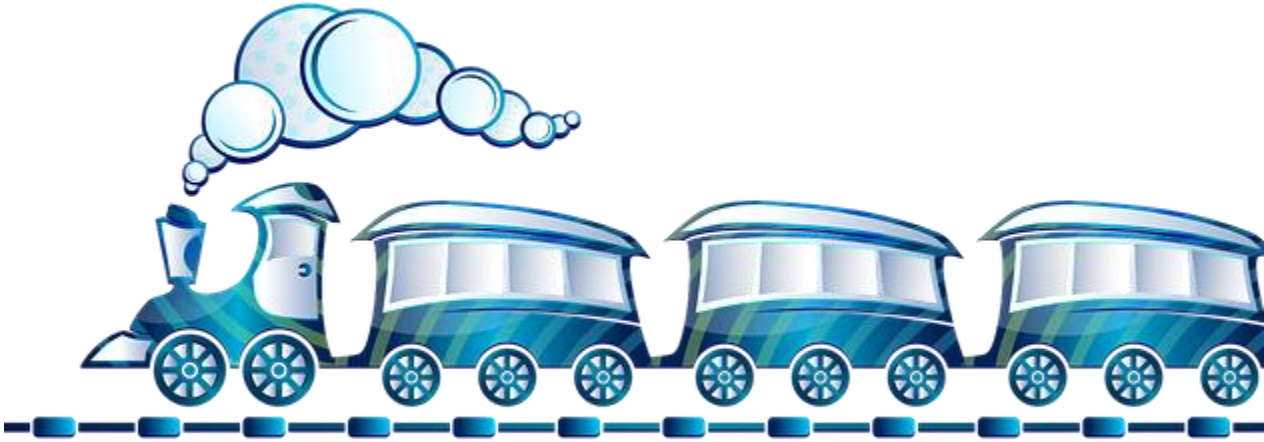
- For humans, this task of recognition is one of the first skills we learn from the moment we are born and is one that comes naturally and effortlessly as adults.
- Most of the time we are able to immediately characterize the scene and give each object a label, all without even consciously noticing.

What's that?

A Train

What's this?

A Train

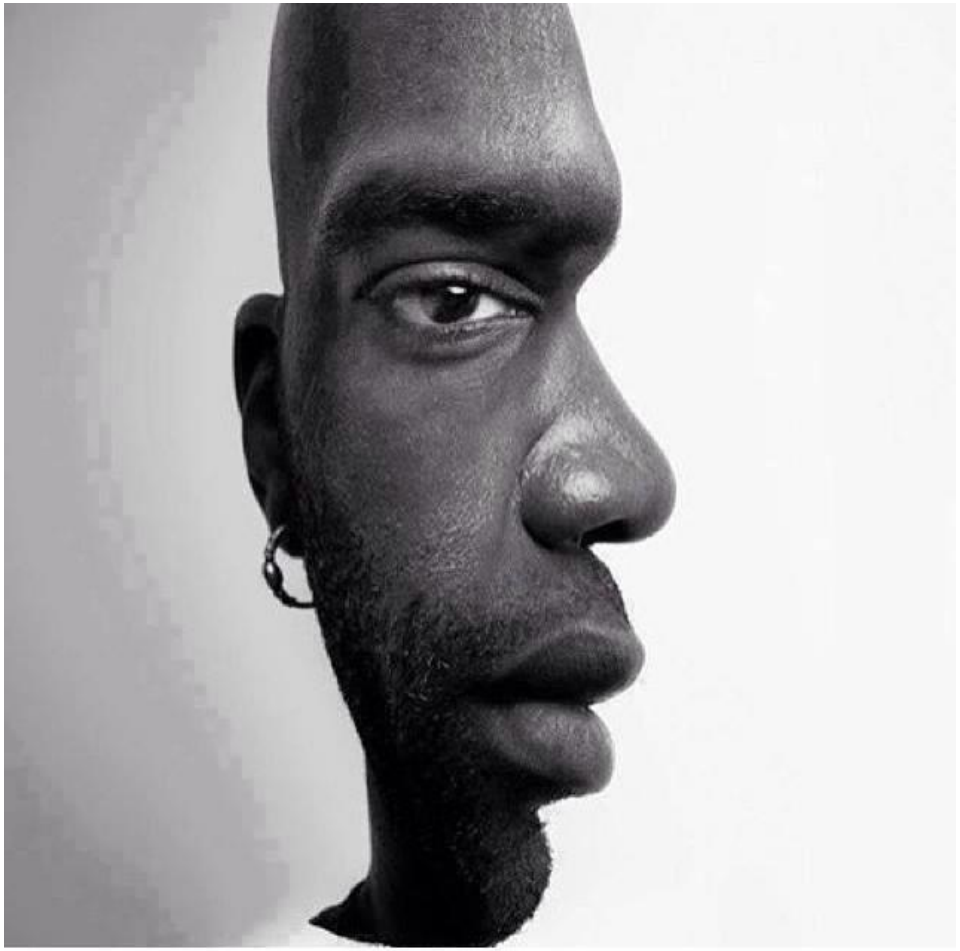


Teach Machine

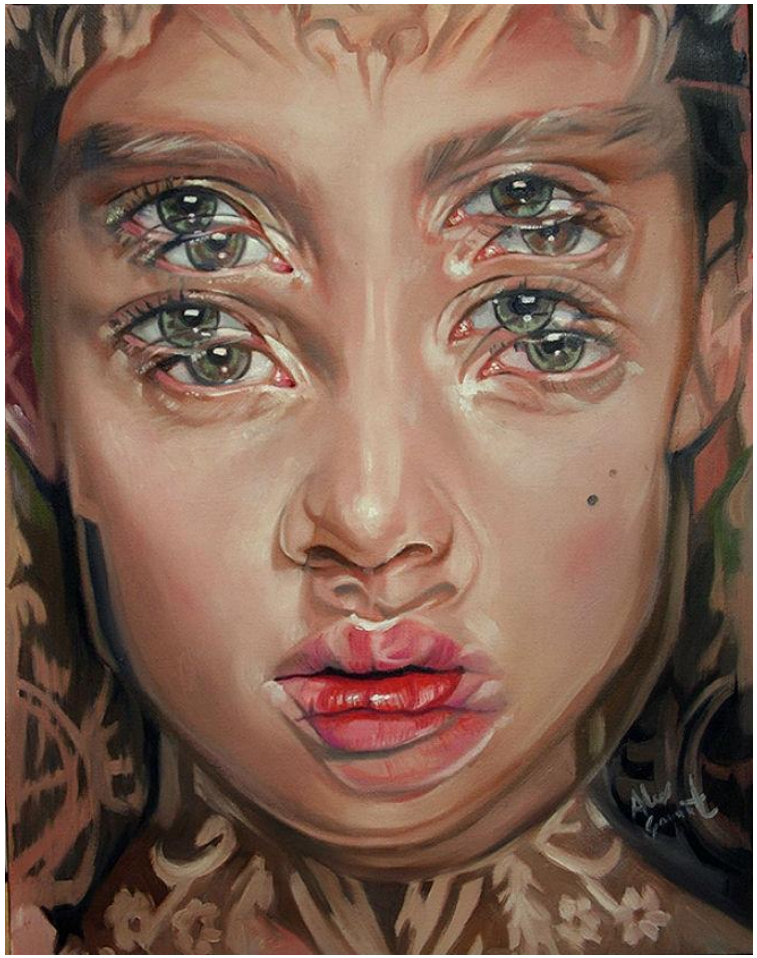
Human skills of being able to :

- Quickly recognize patterns
- Generalize from prior knowledge
- Adapt to different image environments are ones that we do not share with our fellow machines.

illusions



illusions



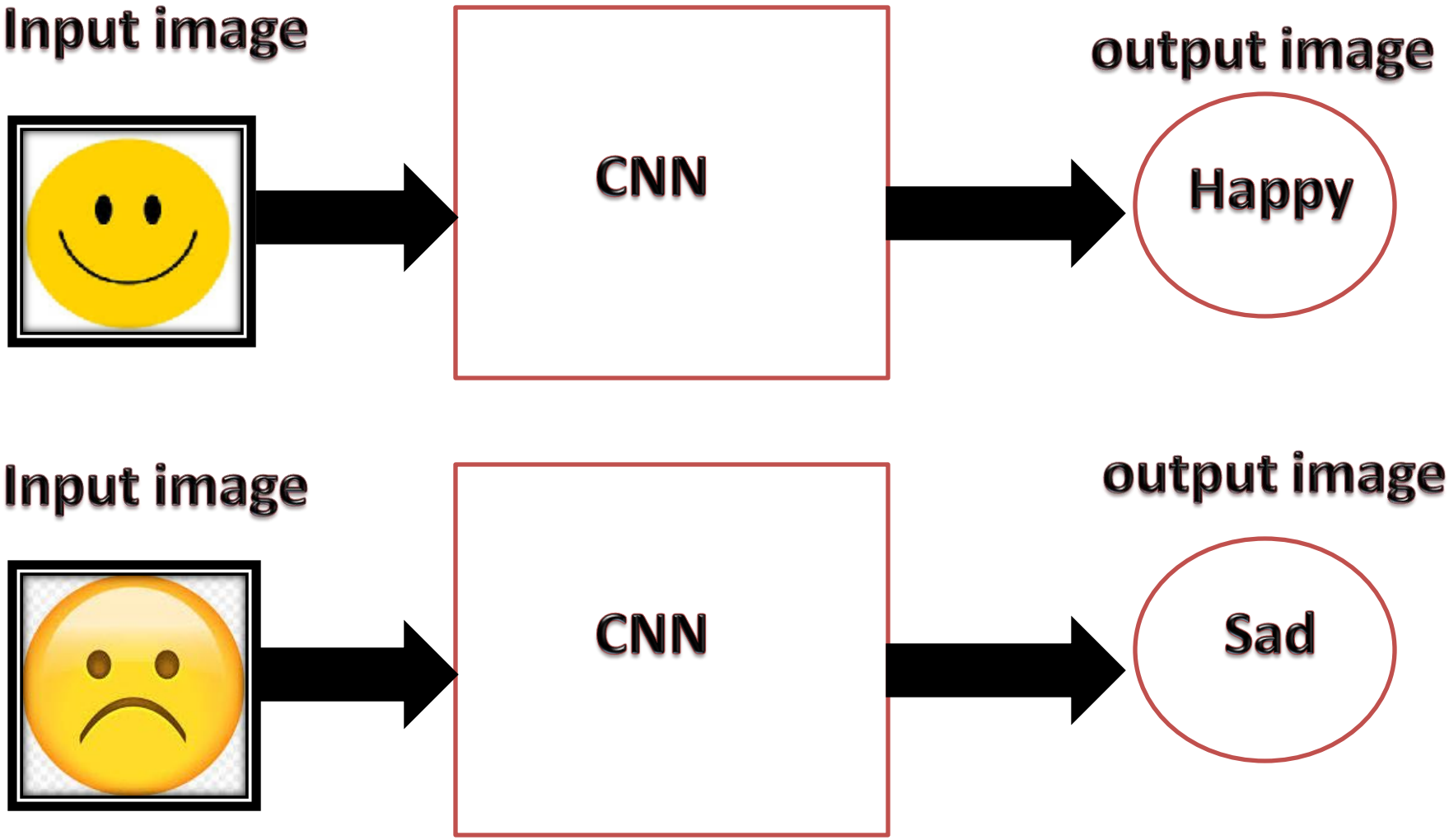
Convolution Neural Network

Input image

CNN

output label
image class

Convolution Neural Network



How Does Computer See an image??

When a computer sees an image ,it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values).

What we see

What computers see

How Does Computer See an image??

What we see

What computers see

These numbers, when we perform image classification, are the only inputs available to the computer.

The idea is that you give the computer this array of numbers and it will output numbers that describe the probability of the image being a certain class

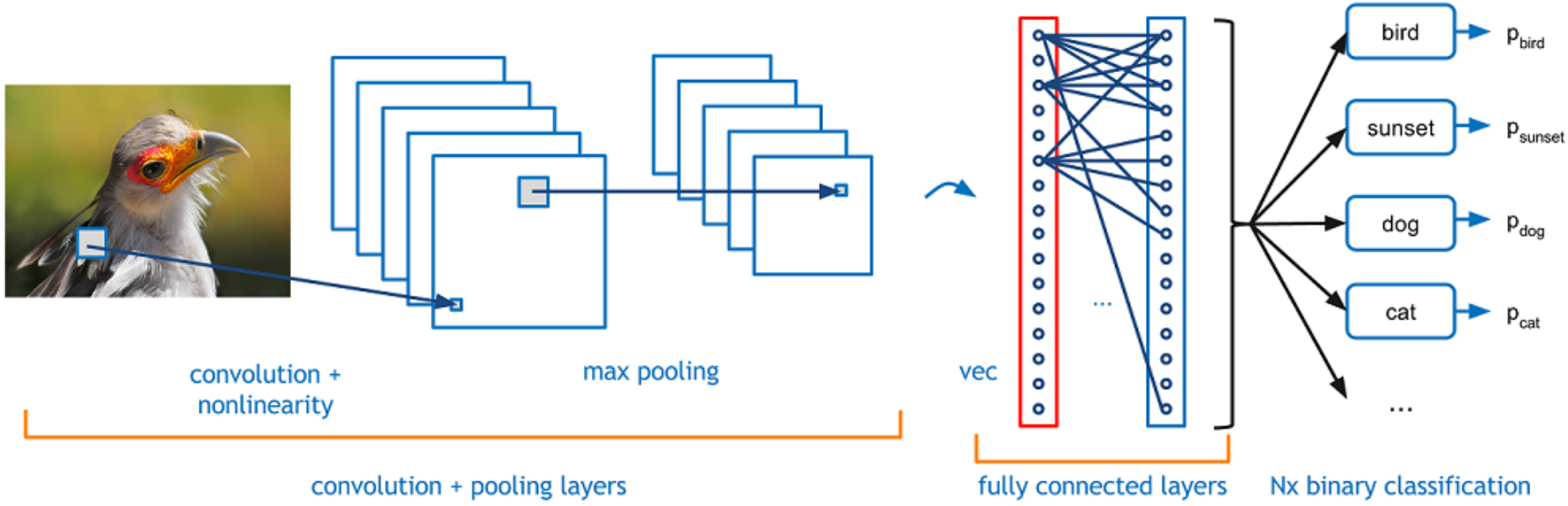
(80% for flower, 15% for sky etc).

What We Want the Computer to Do

- we want the computer to do is to be able to differentiate between all the images it's given and figure out the **unique features** that make a dog a dog , that make a cat a cat or that make a flower a flower .
- The computer is able to perform image classification by looking for low level features such as edges and curves, and then building up to more abstract concepts through a series of **convolutional layers**.

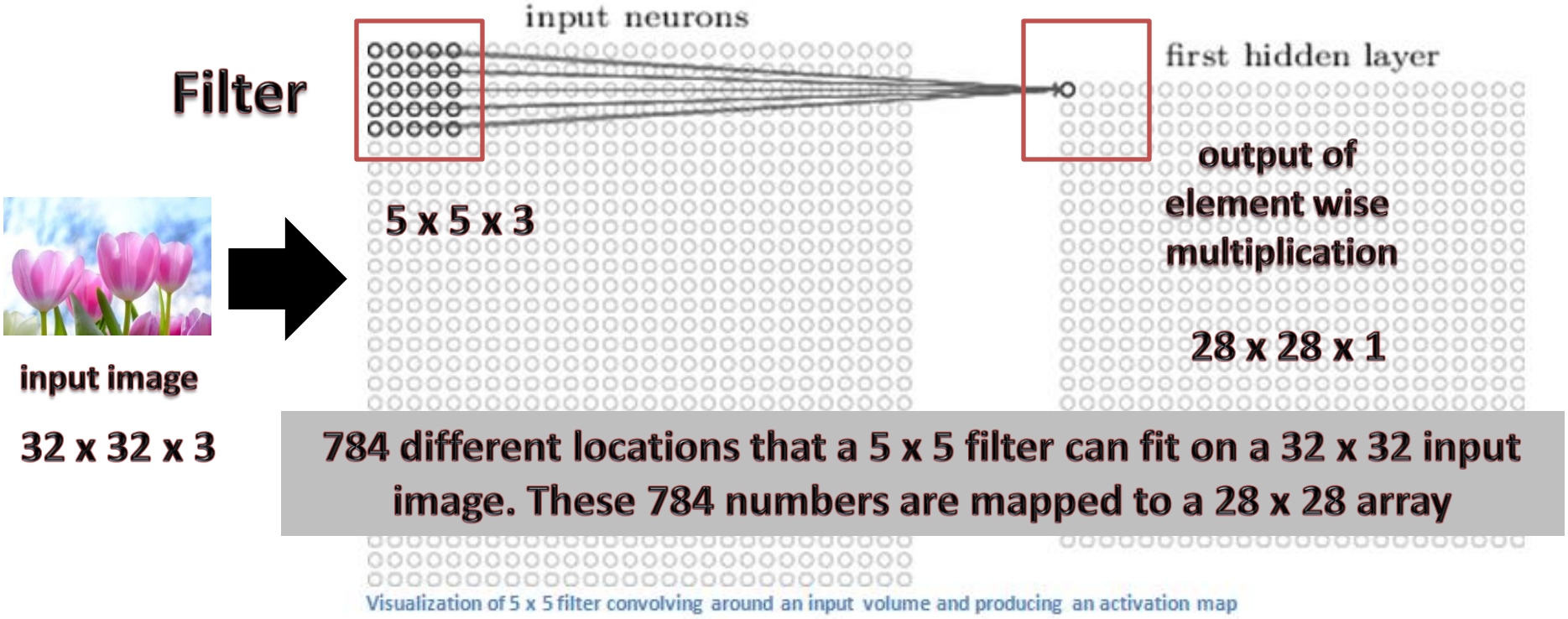
Convolution Neural Network

CNNs take the image, pass it through a series of convolutional, nonlinear, pooling (downsampling), and fully connected layers, and get an output.



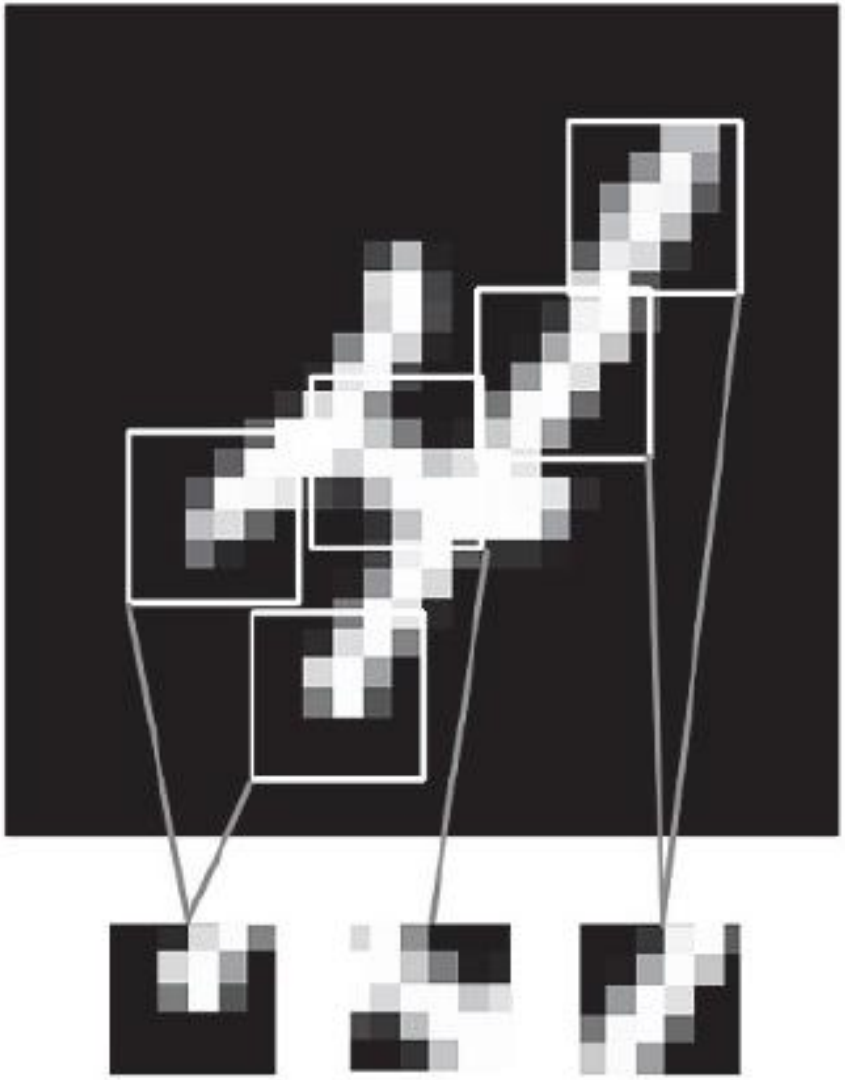
Convolution Neural Network

- **First Layer – Math Part**
  - The first layer in a CNN is always a **Convolutional Layer**.



Convolutional Layer

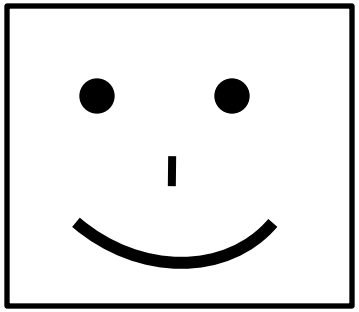
- The fundamental difference between a **densely-connected layer** and a **convolution layer** :
  - **Dense layers learn global patterns in their input feature space**
  - **Convolution layers learn local patterns i.e. in the case of images, patterns found in small 2D windows of the inputs**.

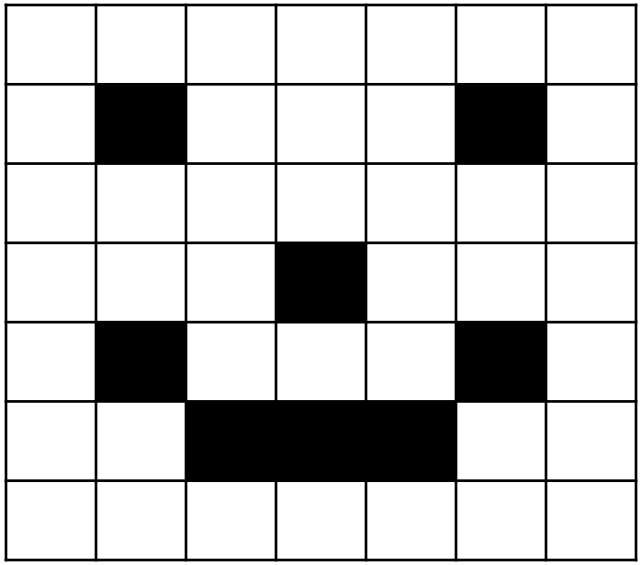


Convolutional Layer

- The patterns they learn are *translation-invariant :*
  - ***after learning a certain pattern in the bottom right corner of a picture, a convnet is able to recognize it anywhere, e.g. in the top left corner.***
  - ***A densely-connected network would have to learn the pattern anew if it appeared at a new location***
- This makes convnets very data-efficient when processing images : they need less training samples to learn representations that have generalization power

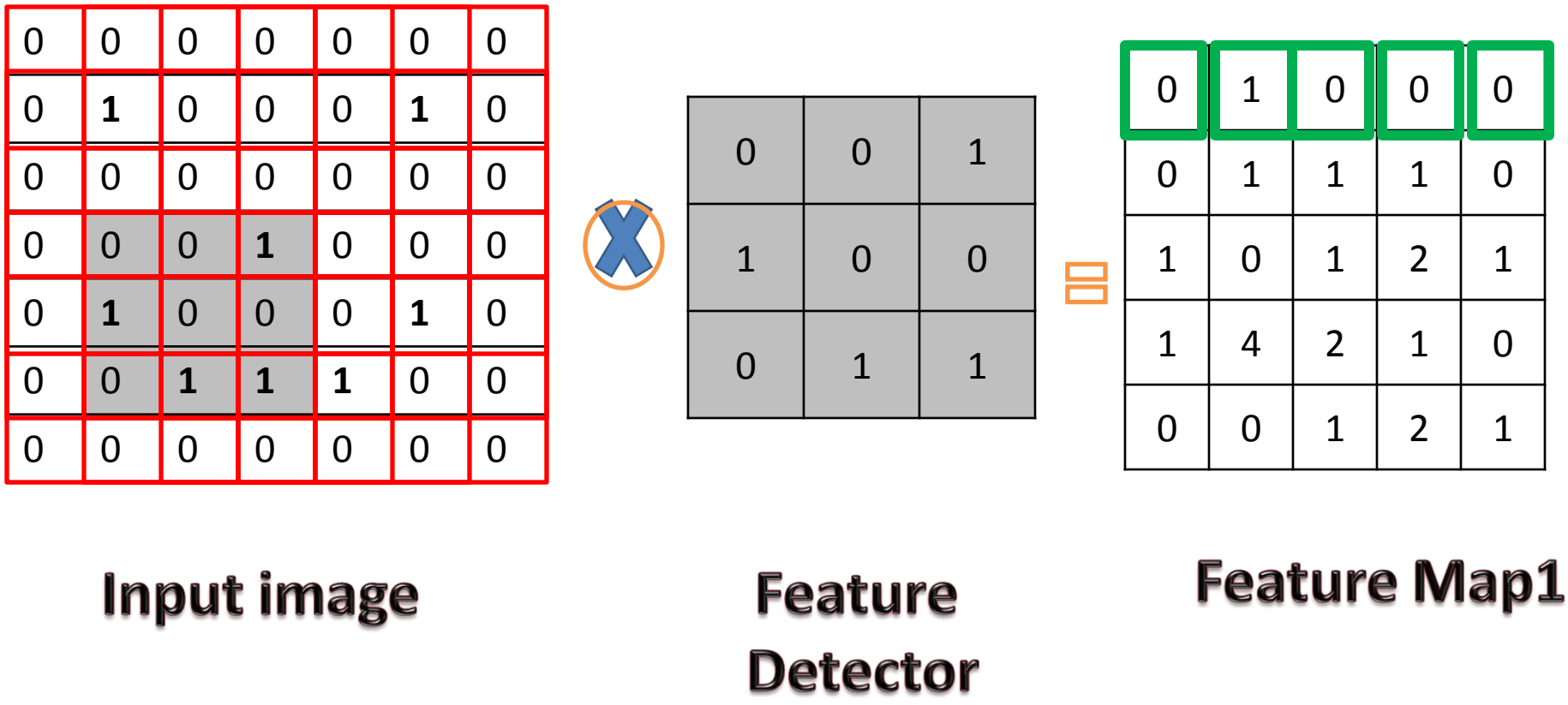
CNN



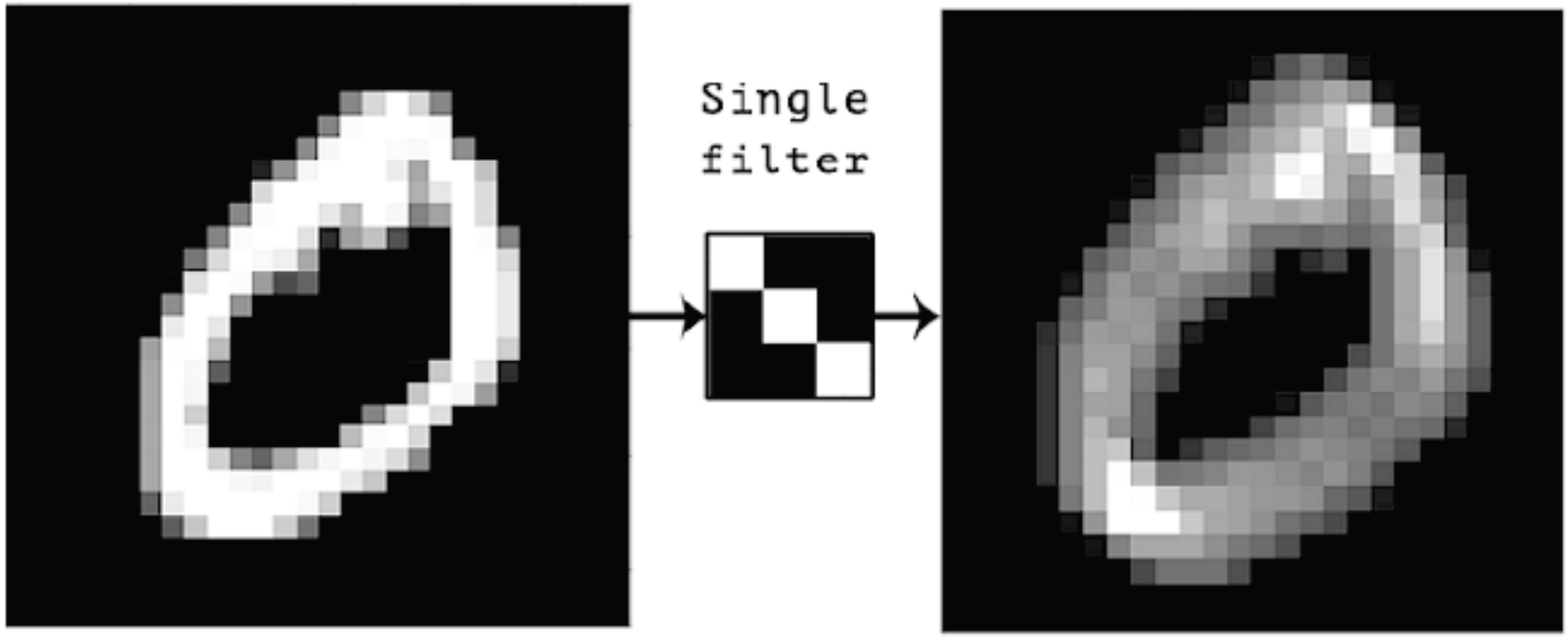


0	0	0	0	0	0	0
0	**1**	0	0	0	**1**	0
0	0	0	0	0	0	0
0	0	0	**1**	0	0	0
0	**1**	0	0	0	**1**	0
0	0	**1**	**1**	**1**	0	0
0	0	0	0	0	0	0

Step1-Convolution

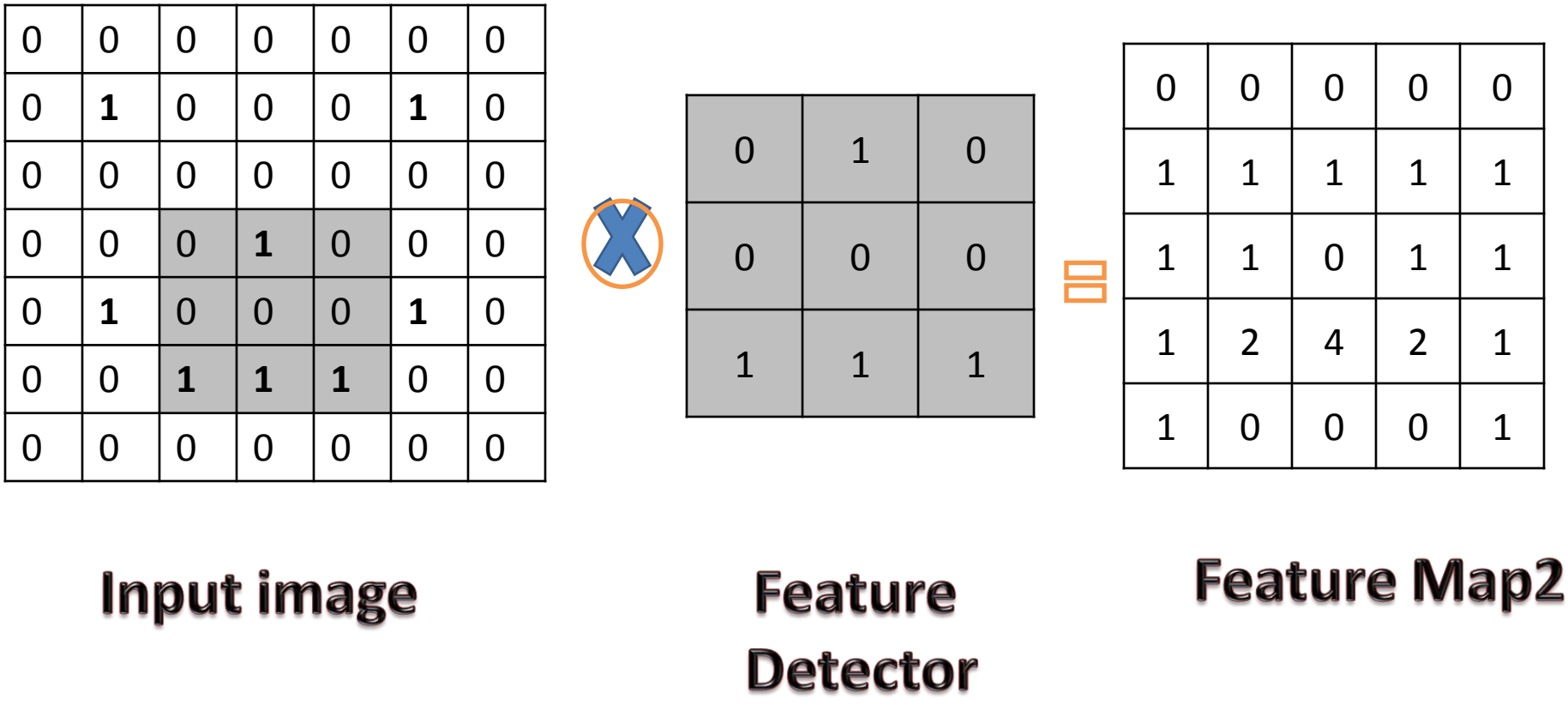


Feature Map / Response Map



- convolution is most typically done with 3x3 windows and no stride (stride 1).

Step1-Convolution



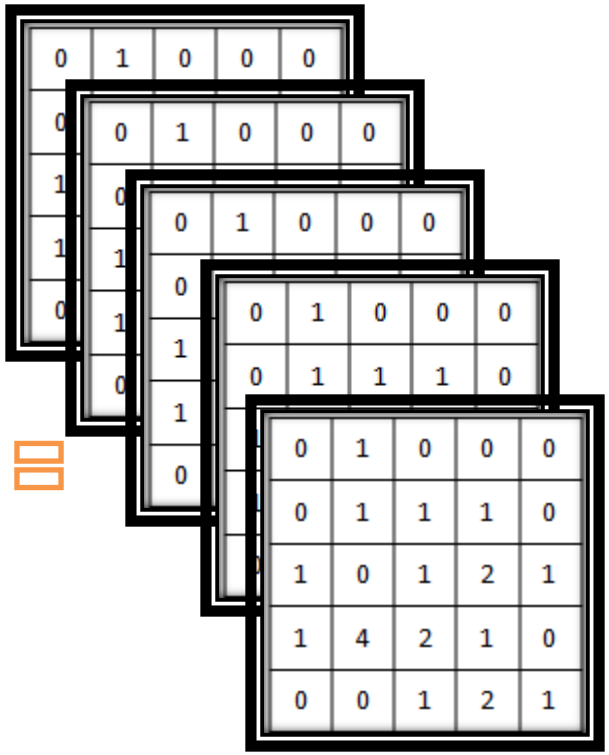
Step1-Convolution

0	0	0	0	0	0	0
0	**1**	0	0	0	**1**	0
0	0	0	0	0	0	0
0	0	0	**1**	0	0	0
0	**1**	0	0	0	**1**	0
0	0	**1**	**1**	**1**	0	0
0	0	0	0	0	0	0

Input image

0	0	1
1	0	0
0	1	1

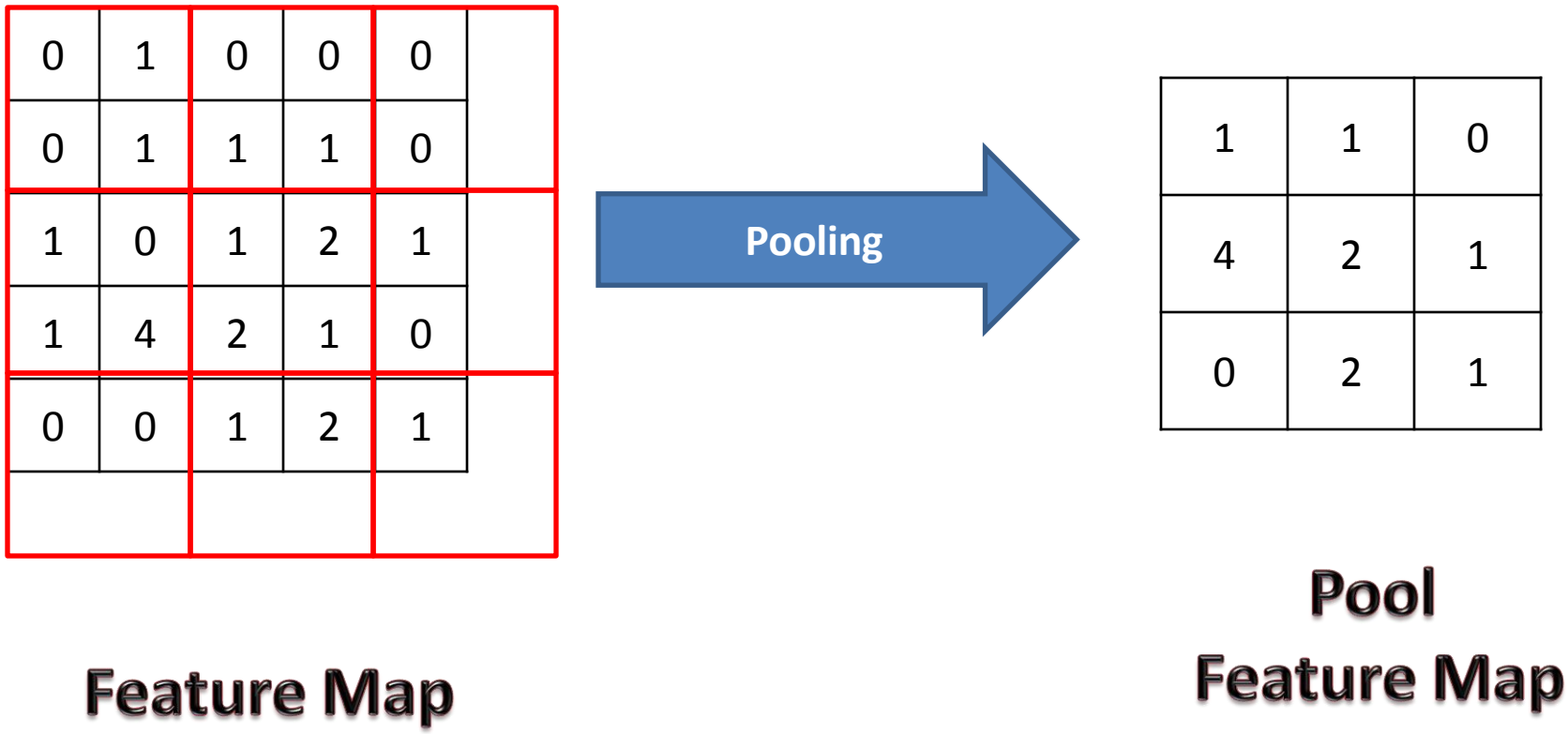
Feature Detector



Feature Map

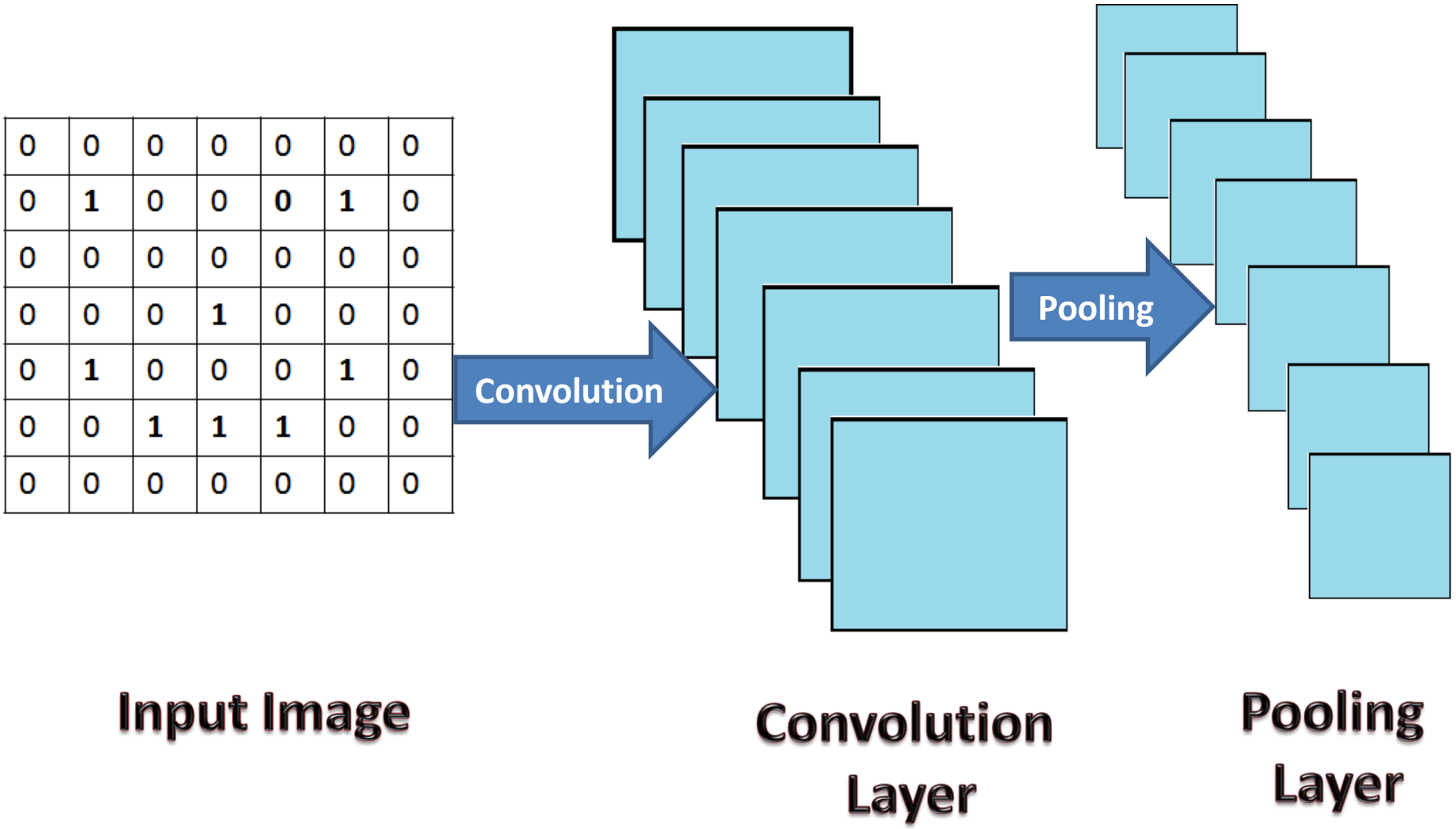
Max Pooling

Max Pooling



Max pooling is usually done with 2x2 windows and stride 2, so as to downsample the feature maps

Max Pooling


0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Convolution
Pooling
Input Image
Convolution Layer
Pooling Layer

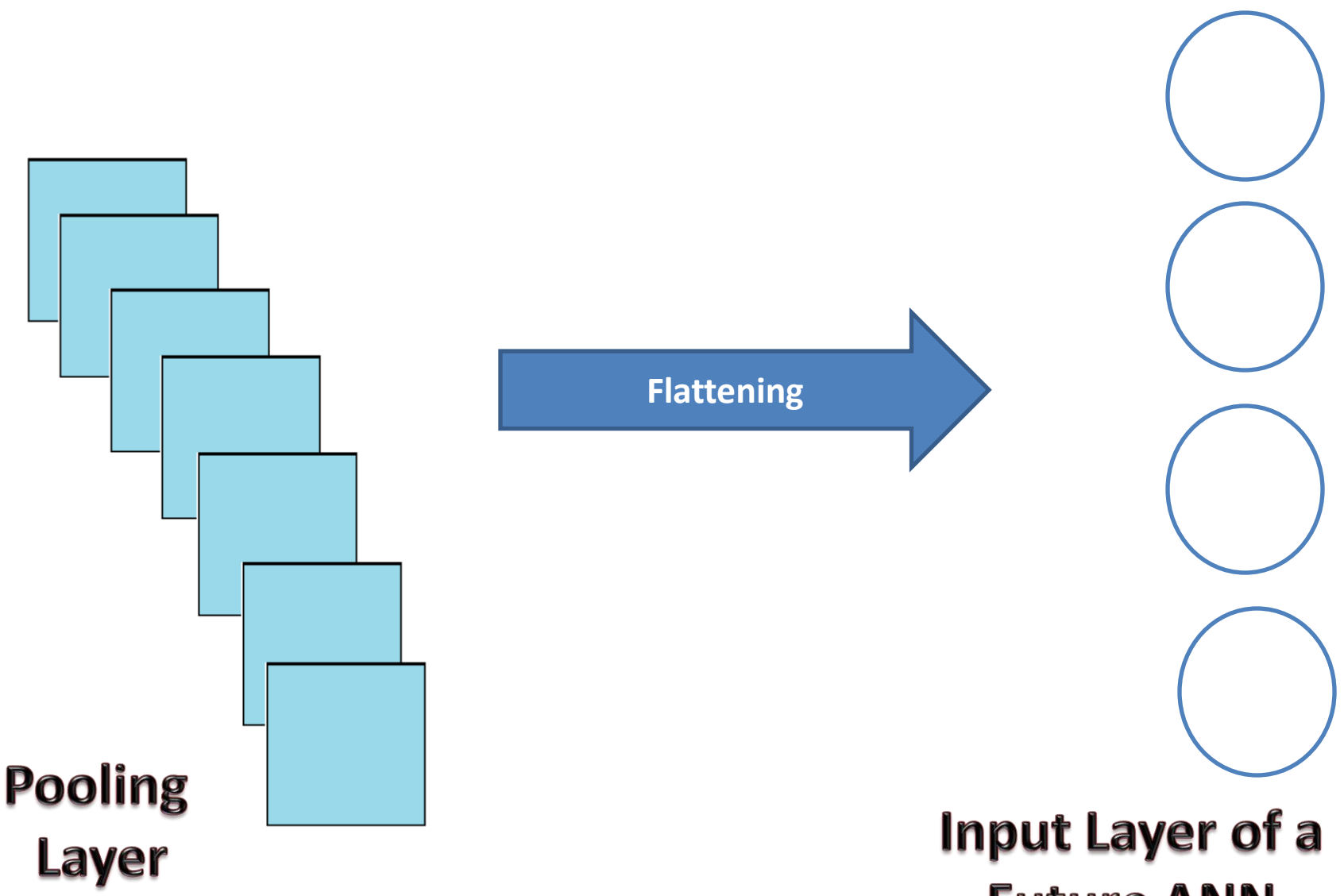
Flattening

1	1	0
4	2	1
0	2	1

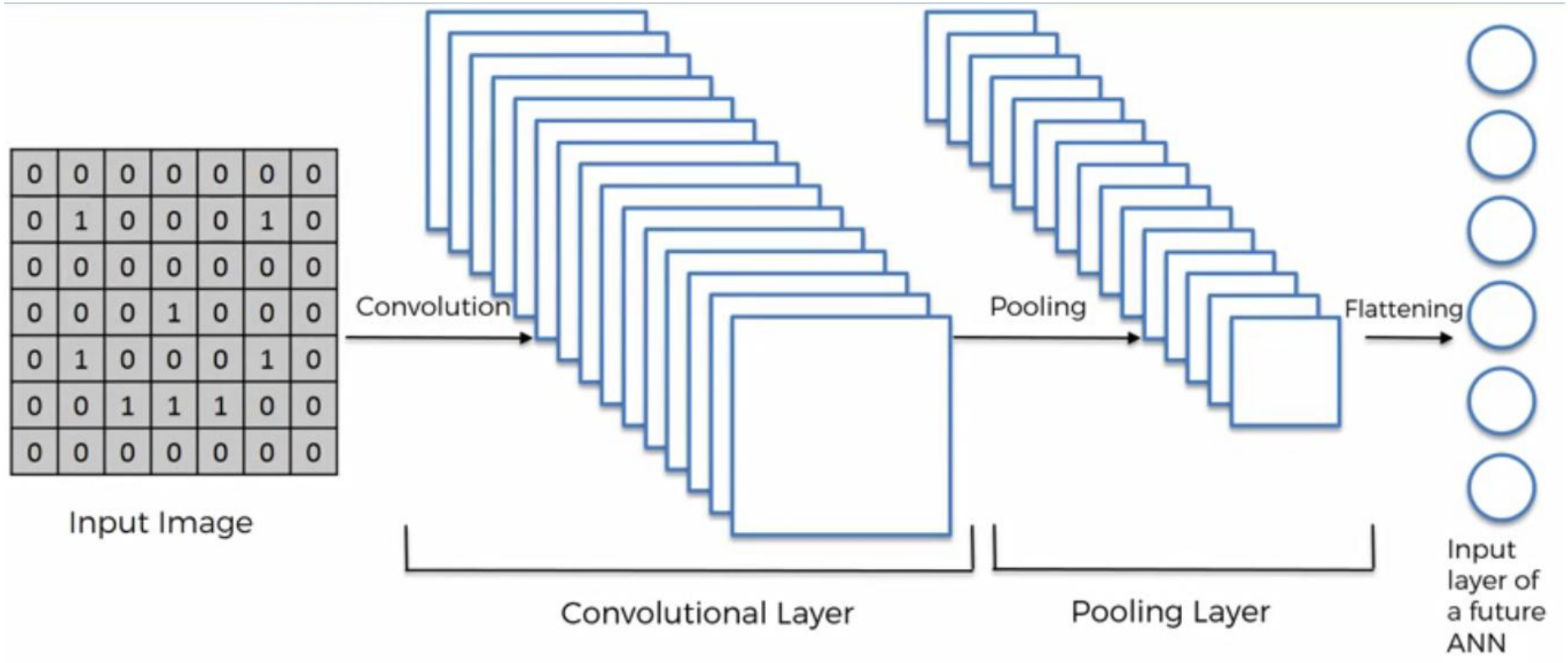
Pool
Feature Map

1
1
0
4
2
1
0
2
1

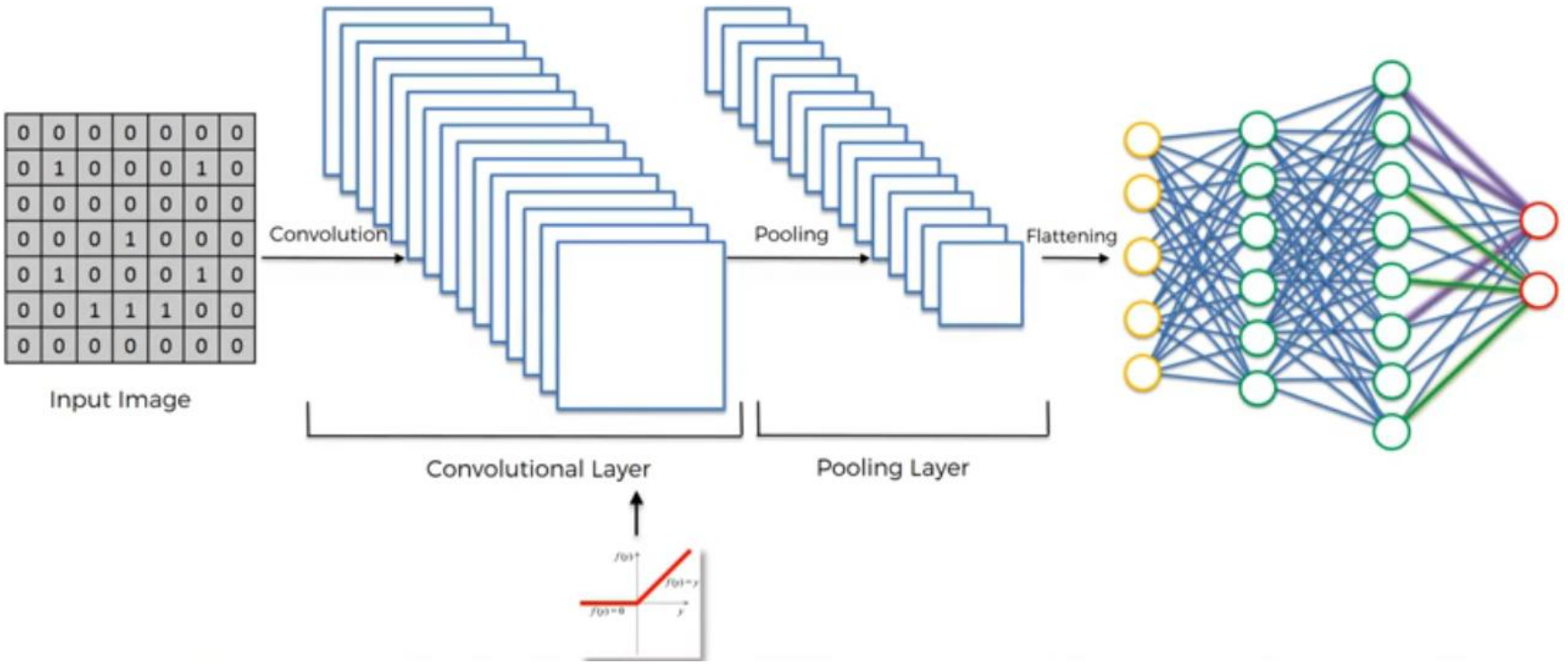
Flattening



Flattening


0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Input Image
Convolution
Pooling
Flattening
Convolutional Layer
Pooling Layer
Input layer of a future ANN

Full Connection



Data Augmentation

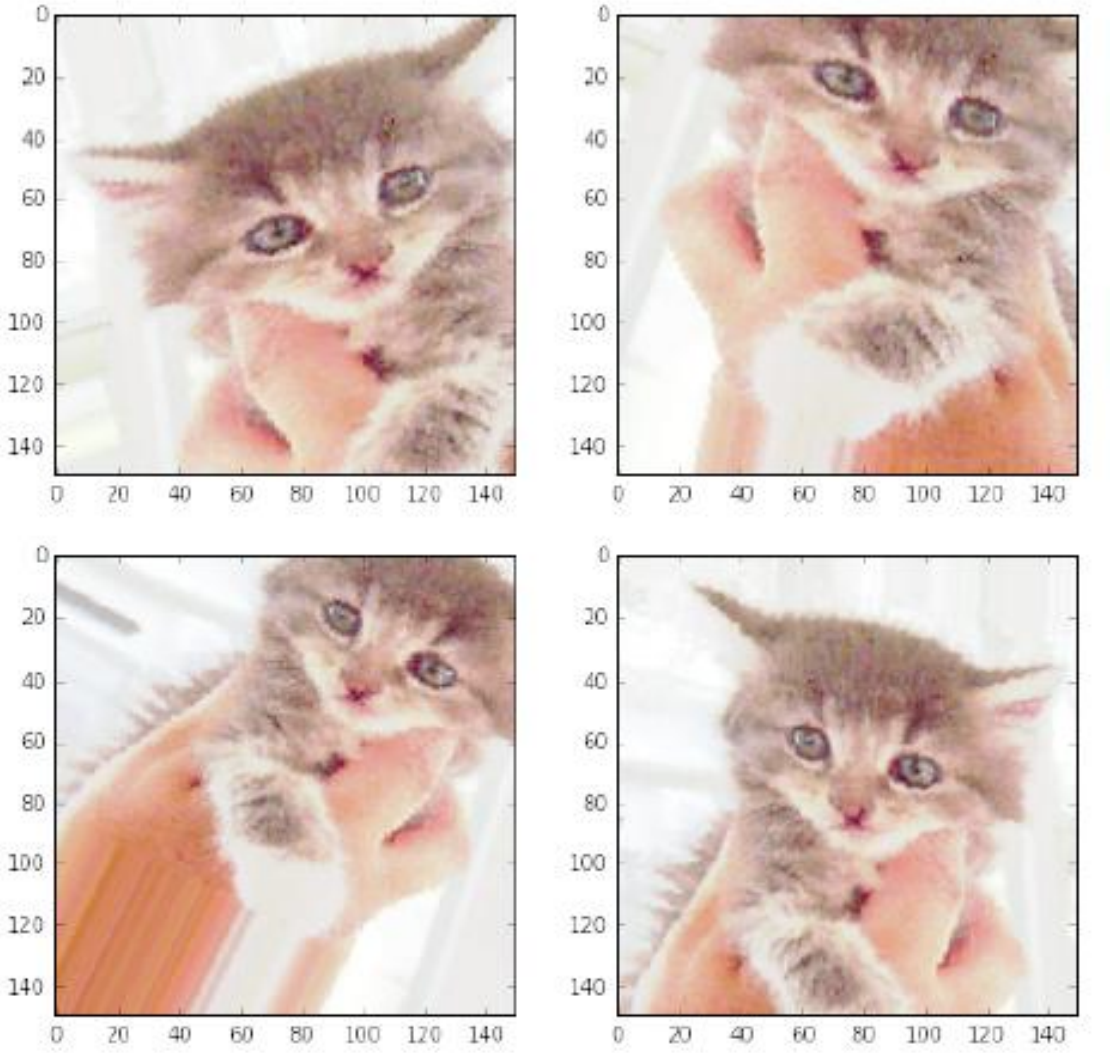
- *Having too few samples to learn from, rendering us unable to train a model able to generalize to new data.*

- *Given infinite data, our model would be exposed to every possible aspect of the data distribution at hand.*

- *Data augmentation takes the approach of generating more training data from existing training samples, by "augmenting" the samples via a number of random transformations that yield believable-looking images.*

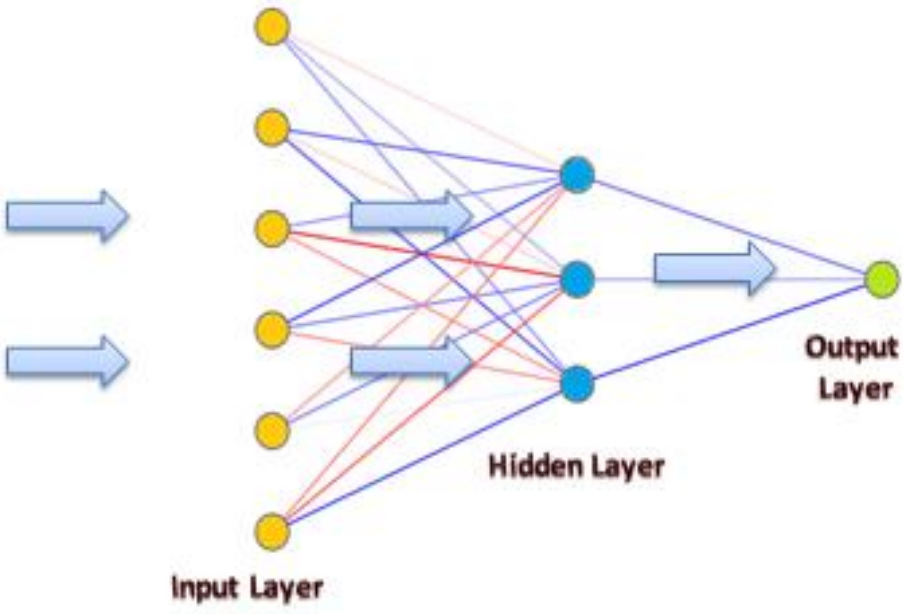
- *The goal is that at training time, our model would never see the exact same picture twice. This helps the model get exposed to more aspects of the data and generalize better.*

Generation of cat pictures via random data augmentation



Feedforward networks

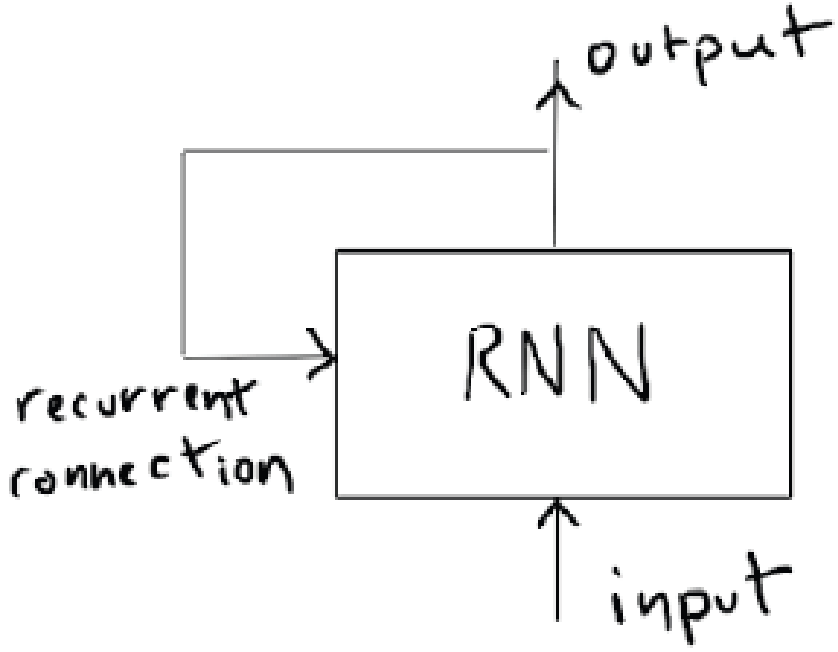
- A major characteristic of all neural networks , such as densely-connected networks and convn
- Each i sed indepe etween inputs
- With s s a seque points, we have t he network at once, i.e. turn it into a single datapoint.



RNN

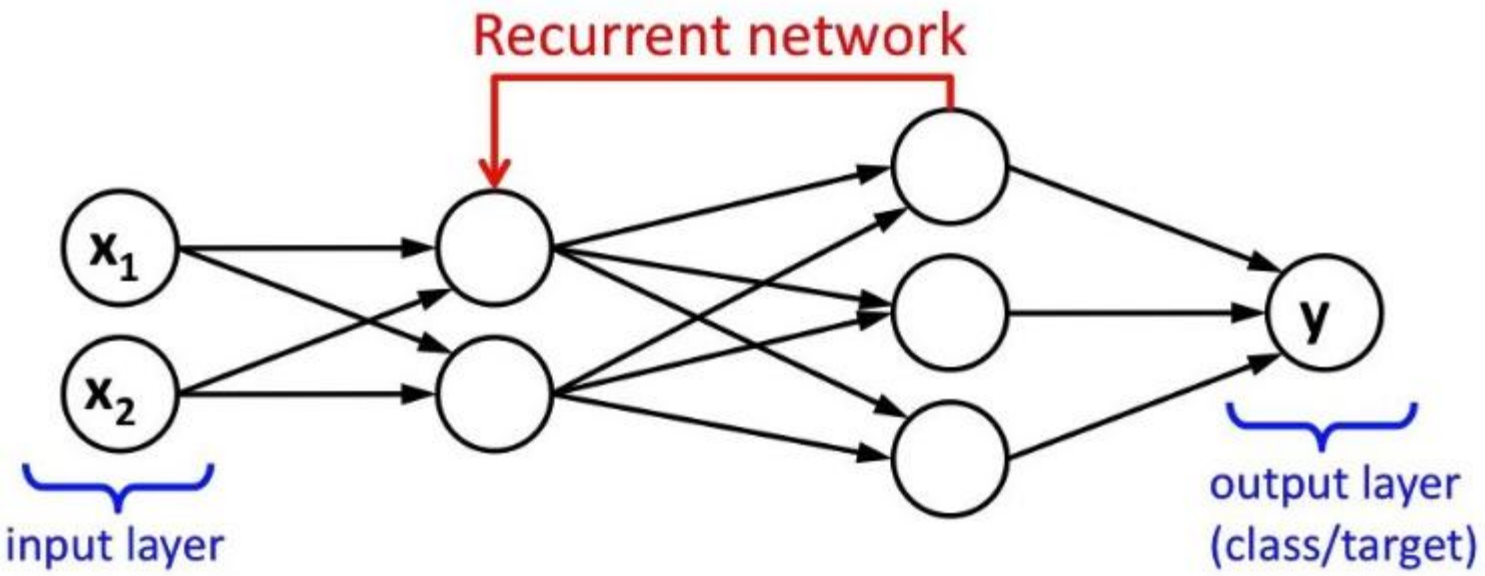
- Biological intelligence processes information incrementally while maintaining an internal model of what it is processing, built from past information and constantly getting updated as new information comes in.
- **Recurrent Neural Networks (RNNs) adopt the same principle**
- They process sequences by iterating through the sequence elements and maintaining a "state" containing information relative to what they have seen so far.

- RNNs are a type of neural network that has an internal loop



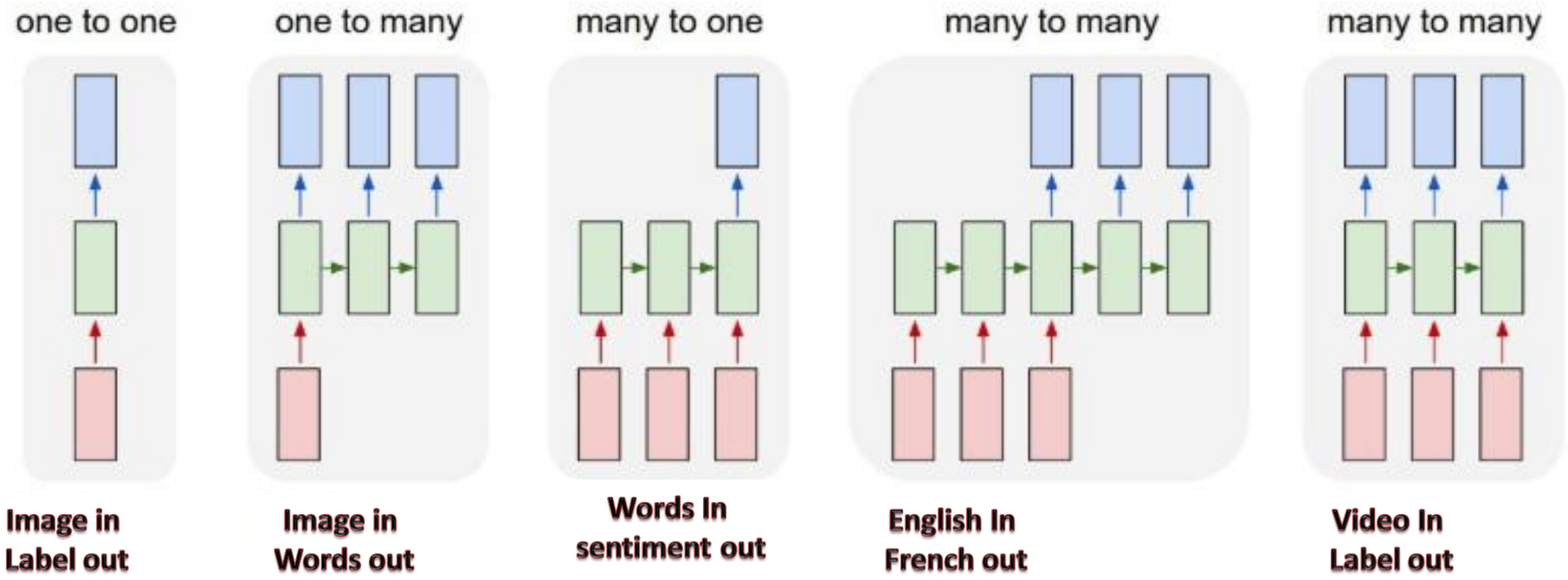
Recurrent Neural Network

On recurrent neural networks(RNN), the previous network state also influences the output, so recurrent neural networks also have a **"notion of time".**



Recurrent Neural Network

Multiple ways that you could use a recurrent neural network compared to the forward networks.

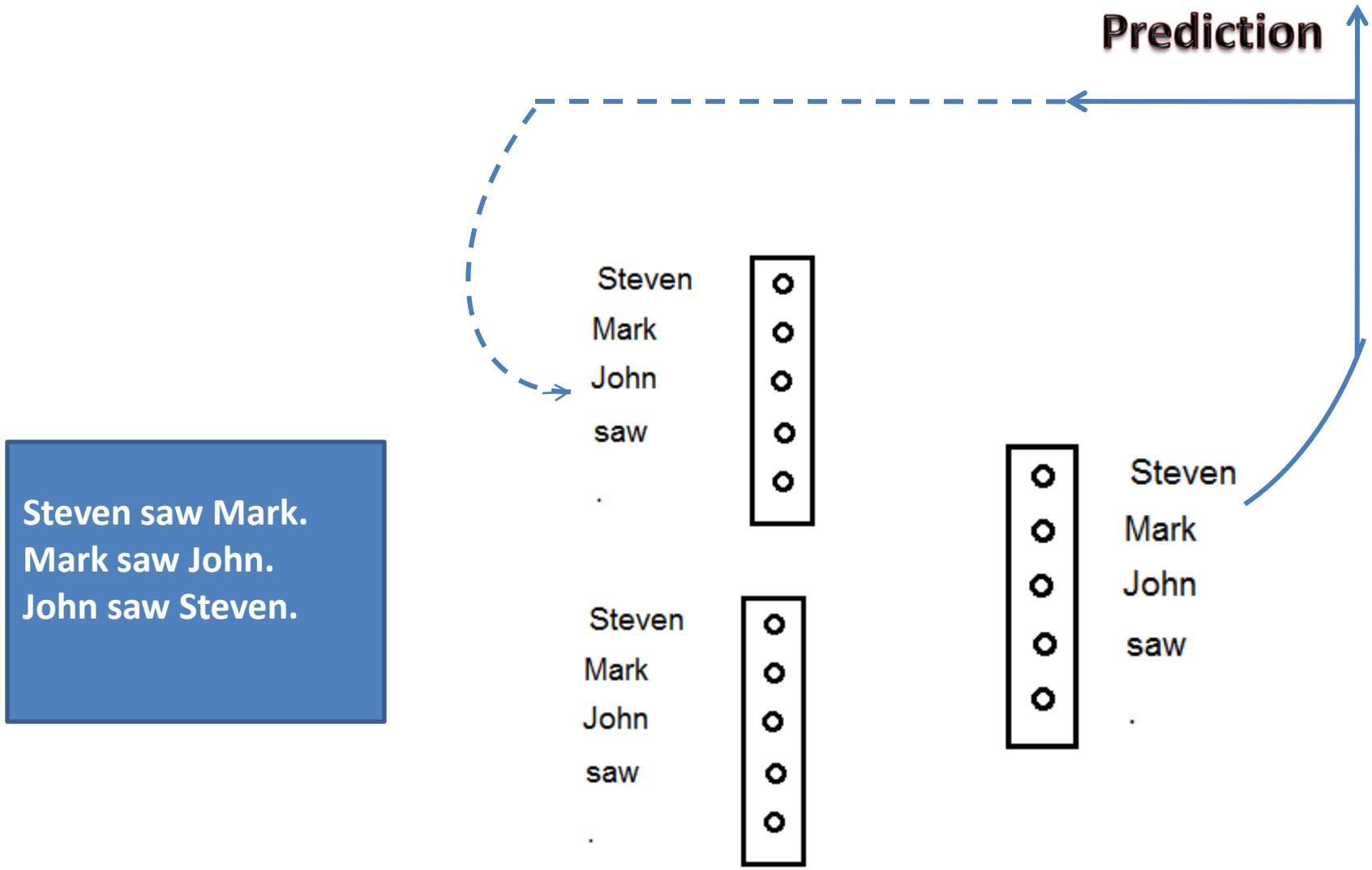


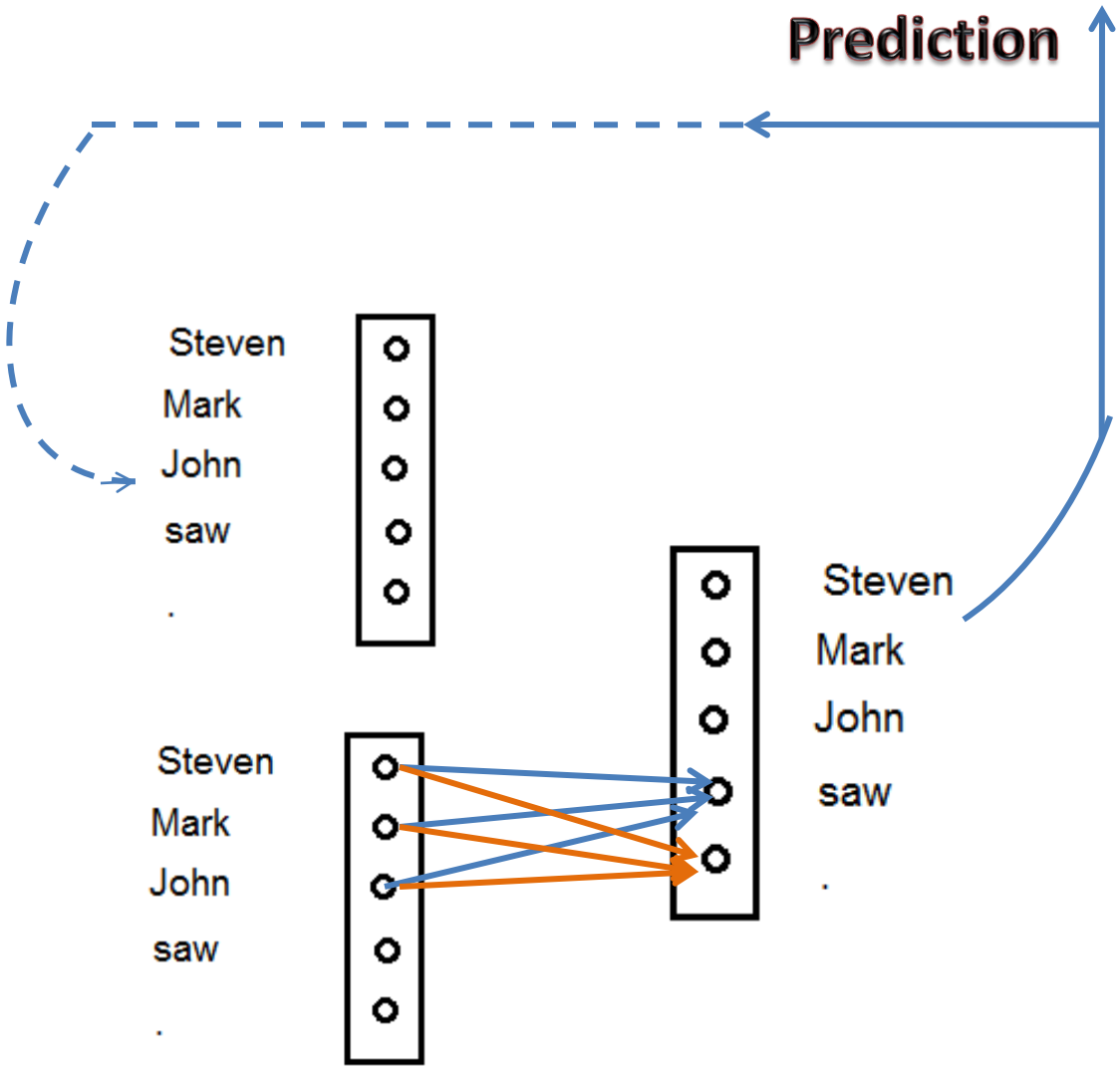
Recurrent Neural Network

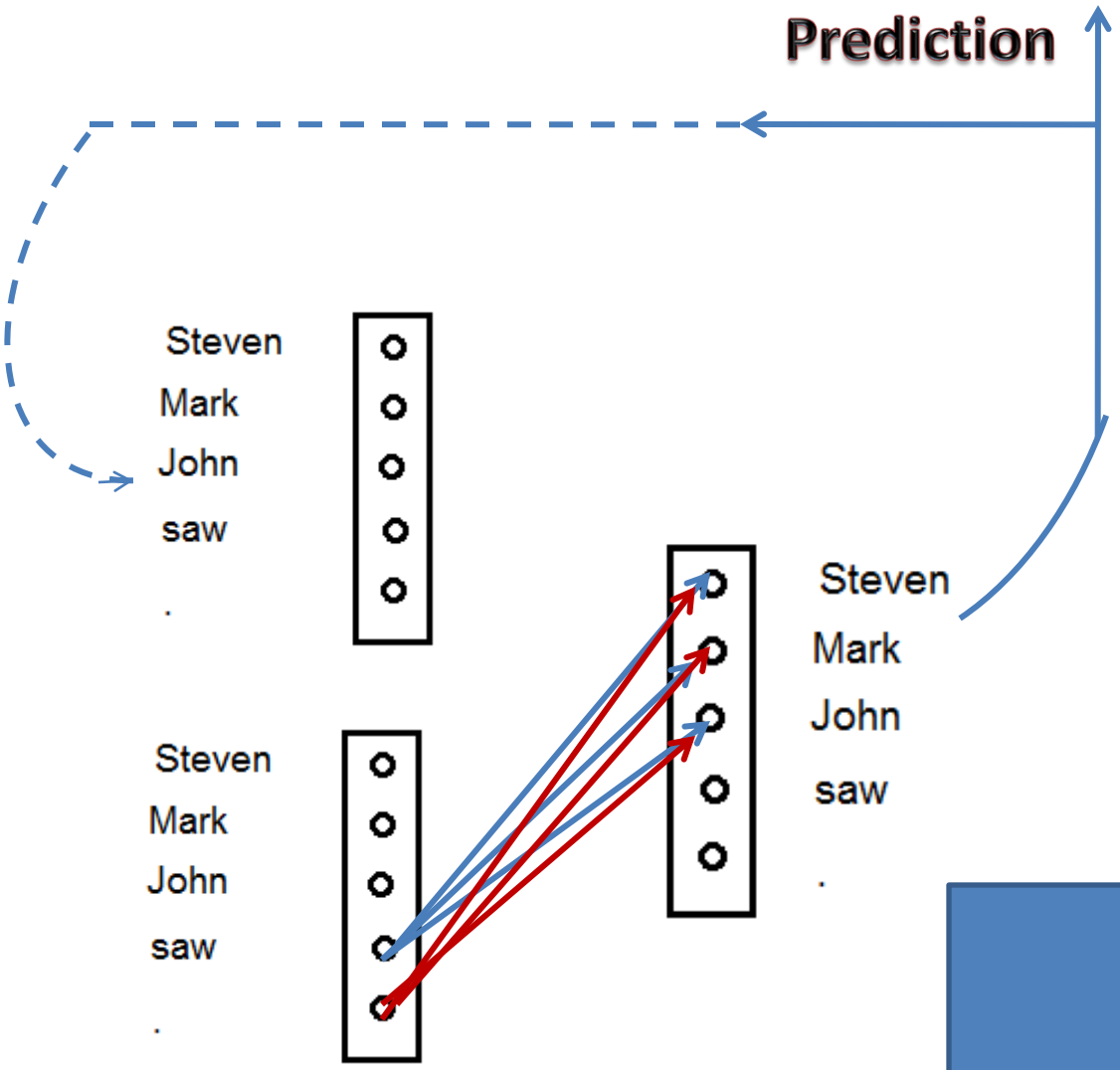
- Steven saw Mark.
- Mark saw John.
- John saw Steven.

- Small Dictionary of words ::

  {Steven, Mark, John, saw, .}


Prediction
Steven saw Mark.
Mark saw John.
John saw Steven.
Steven
Mark
John
saw
.
Steven
Mark
John
saw
.
Steven
Mark
John
saw
.


Prediction
Steven
Mark
John
saw
.
Steven
Mark
John
saw
.
Steven
Mark
John
saw
.

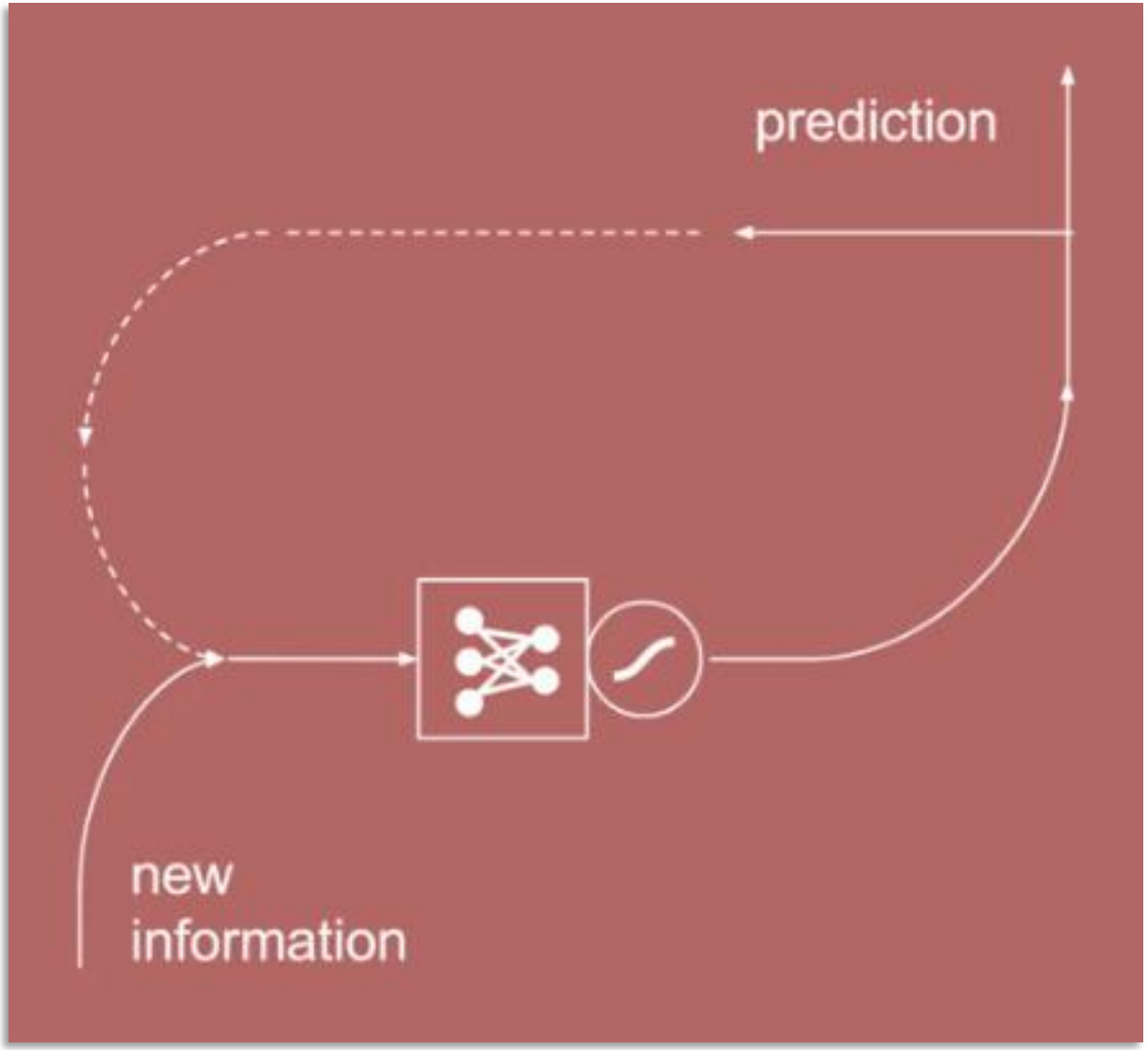


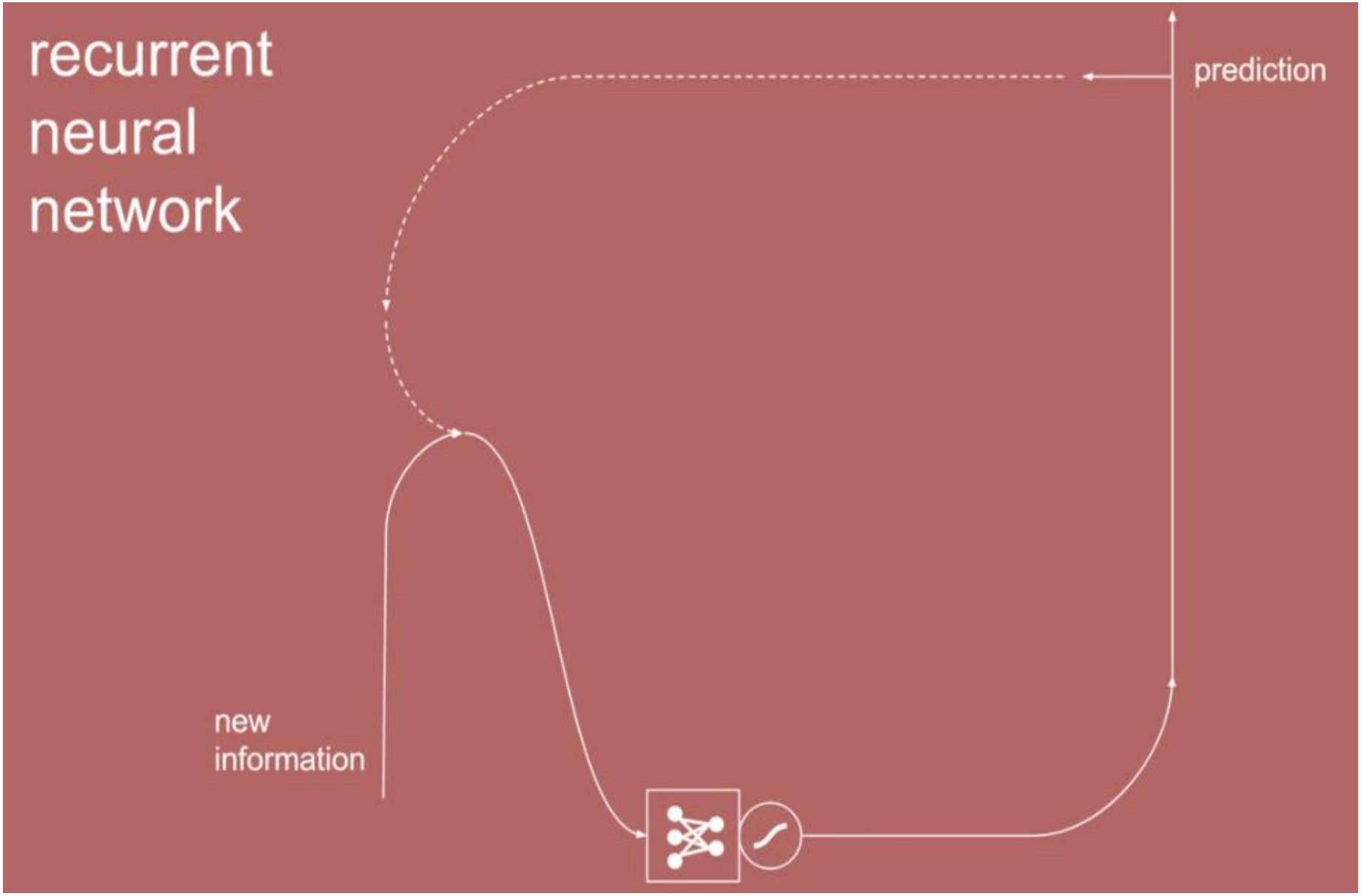
Steven saw Mark.

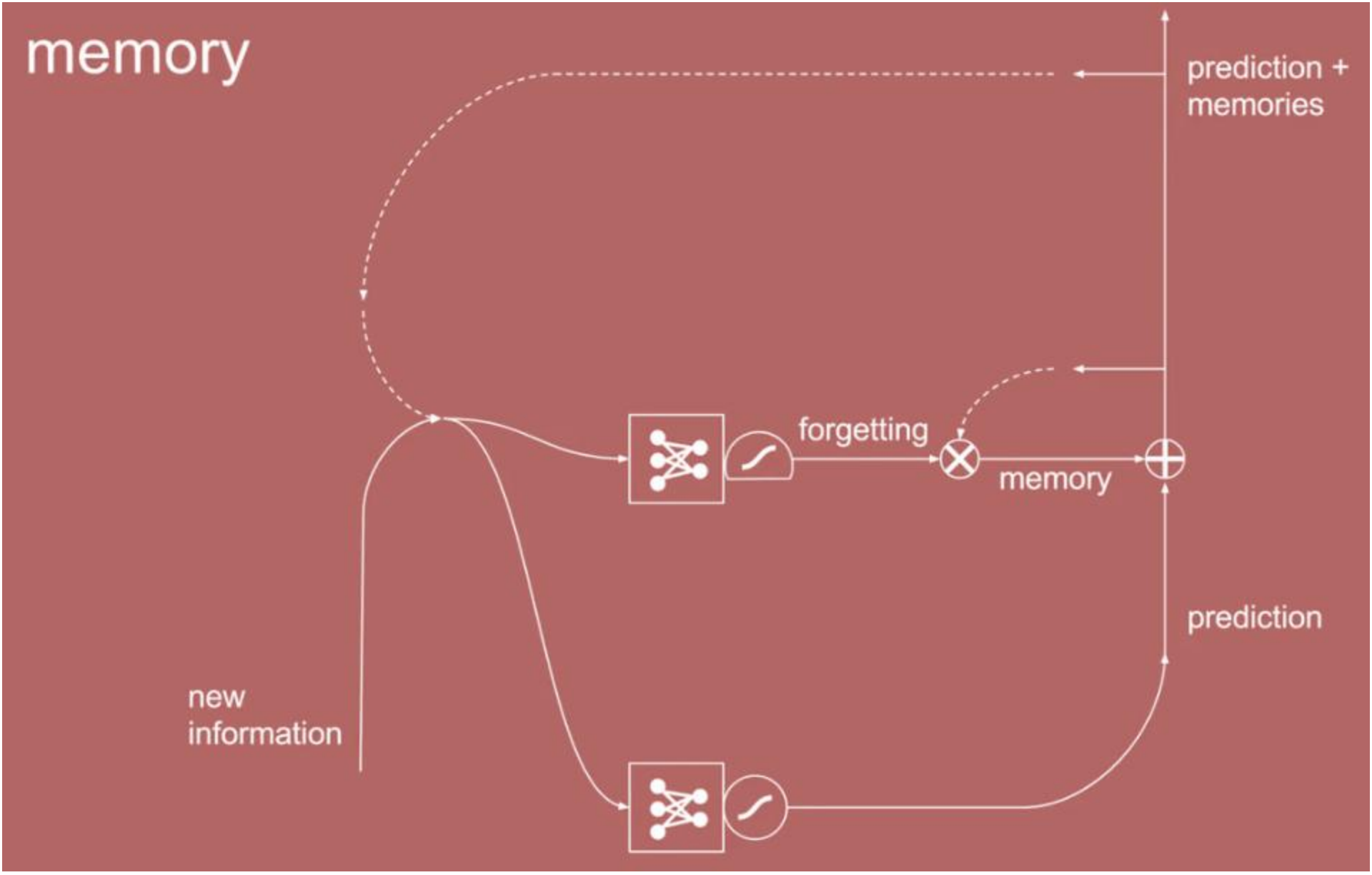
Steven saw Mark saw John saw ...

Steven.Mark.John.

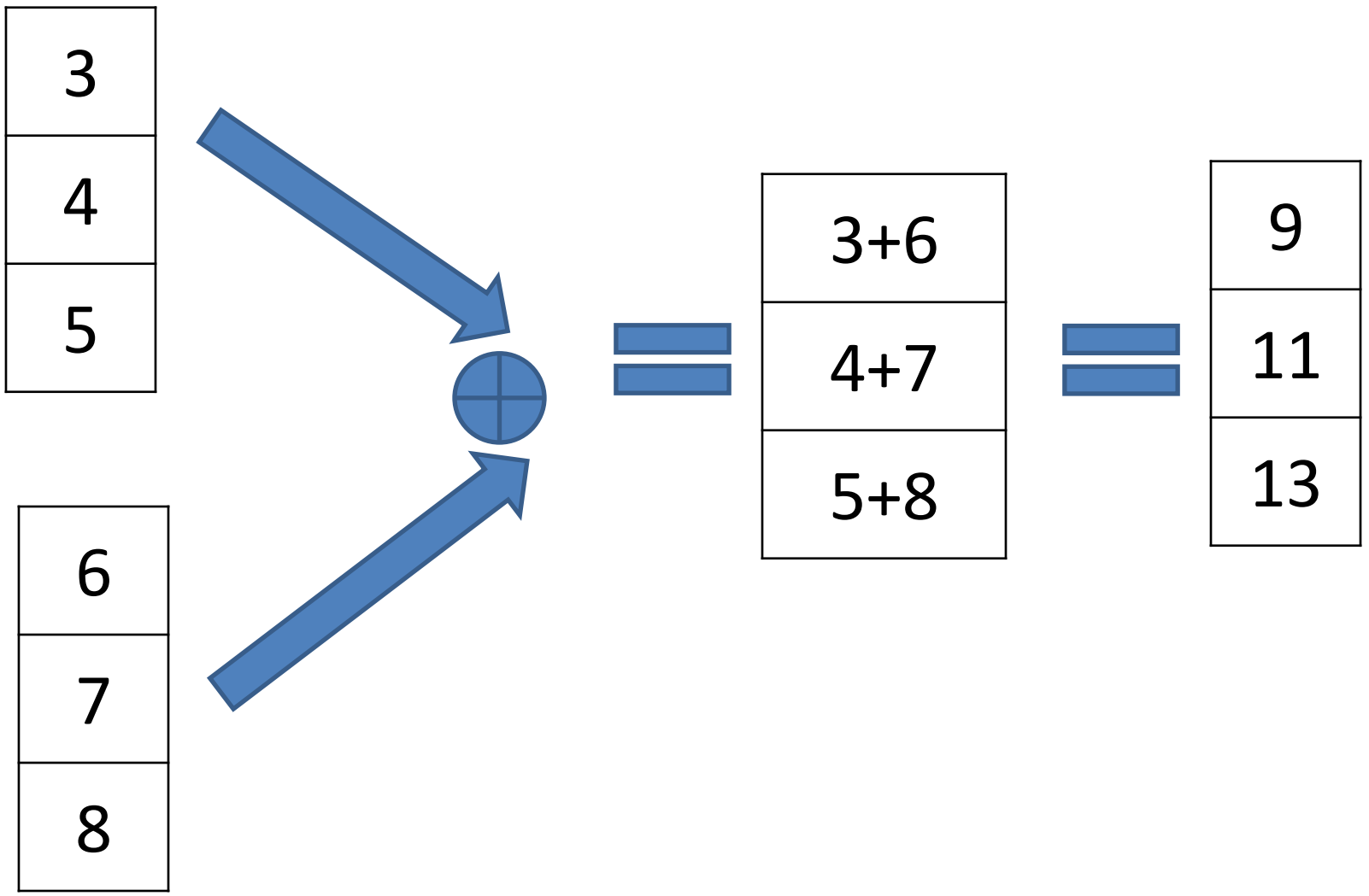
Recurrent neural network



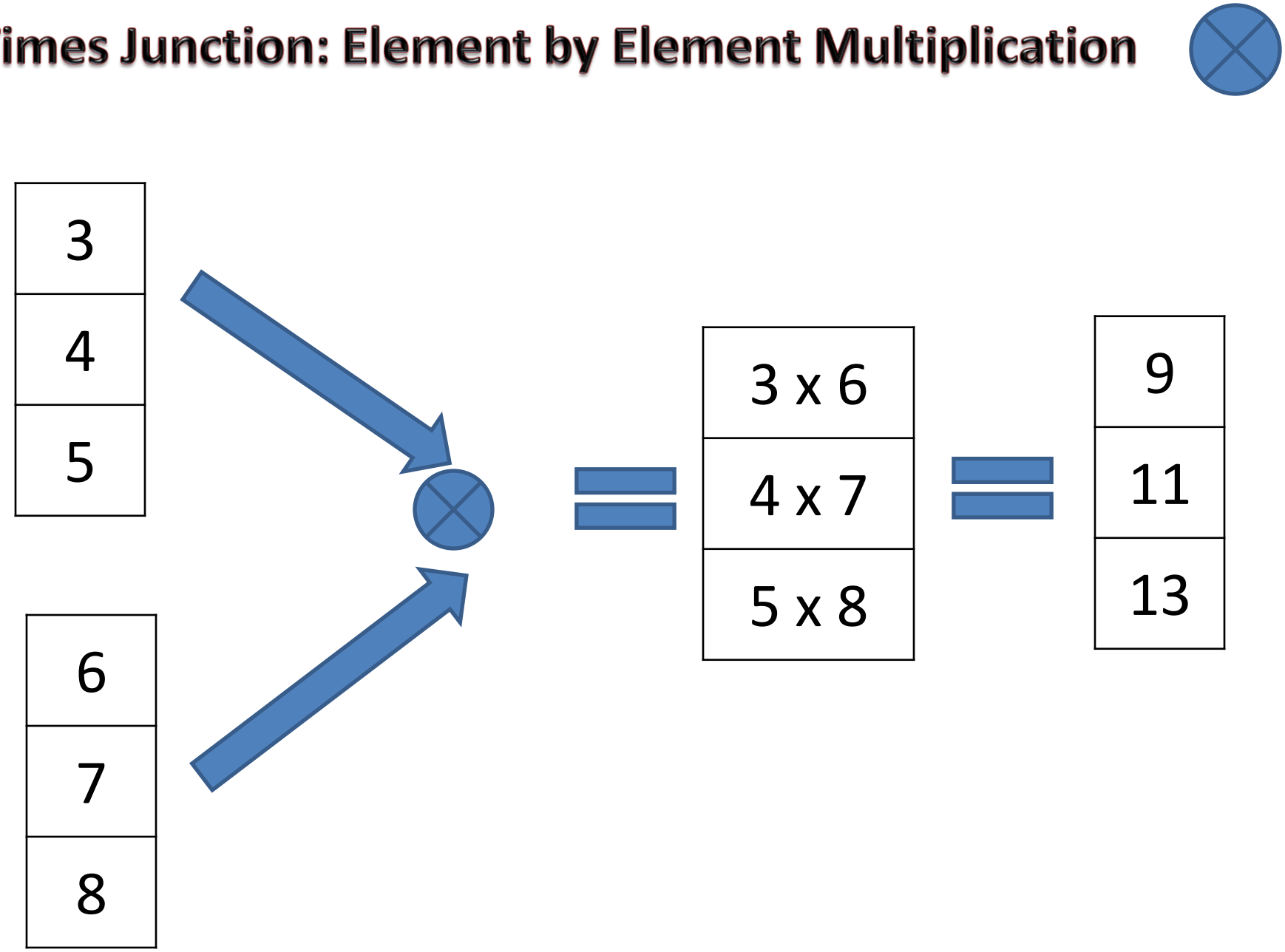

recurrent
neural
network
prediction
new
information


memory
prediction +
memories
forgetting
memory
prediction
new
information

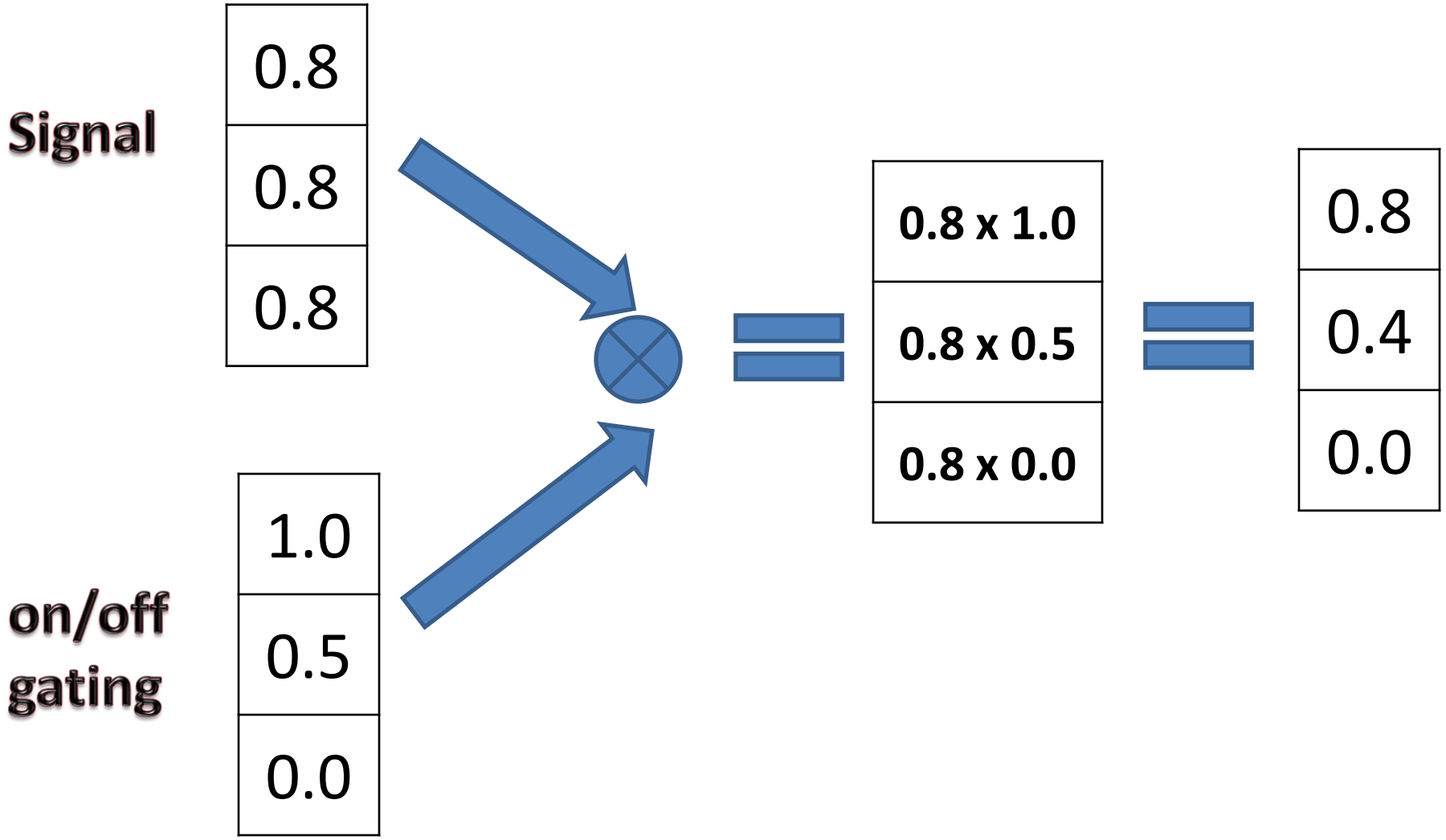
Plus Junction: Element by Element Addition



Times Junction: Element by Element Multiplication

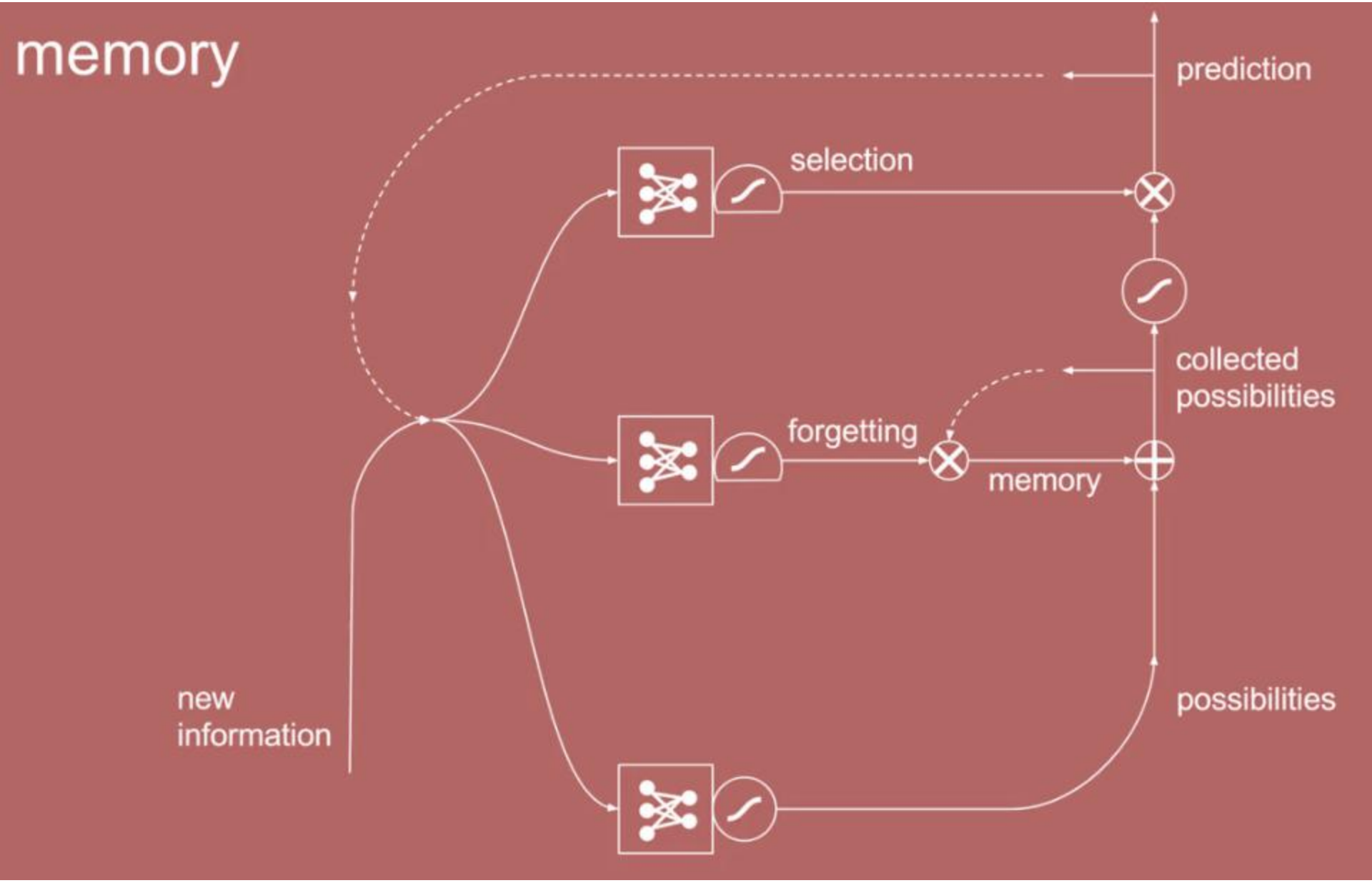


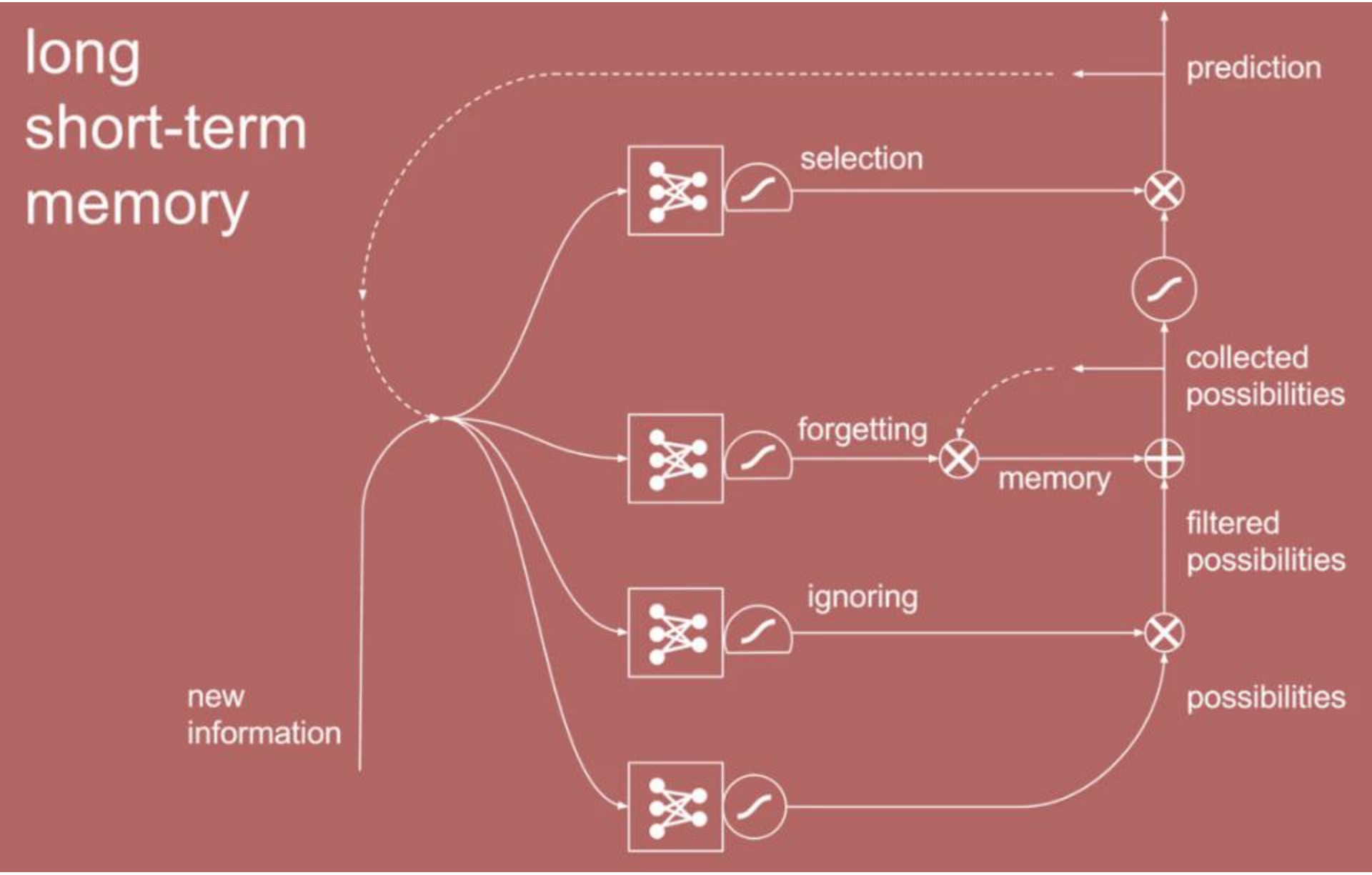
Gating

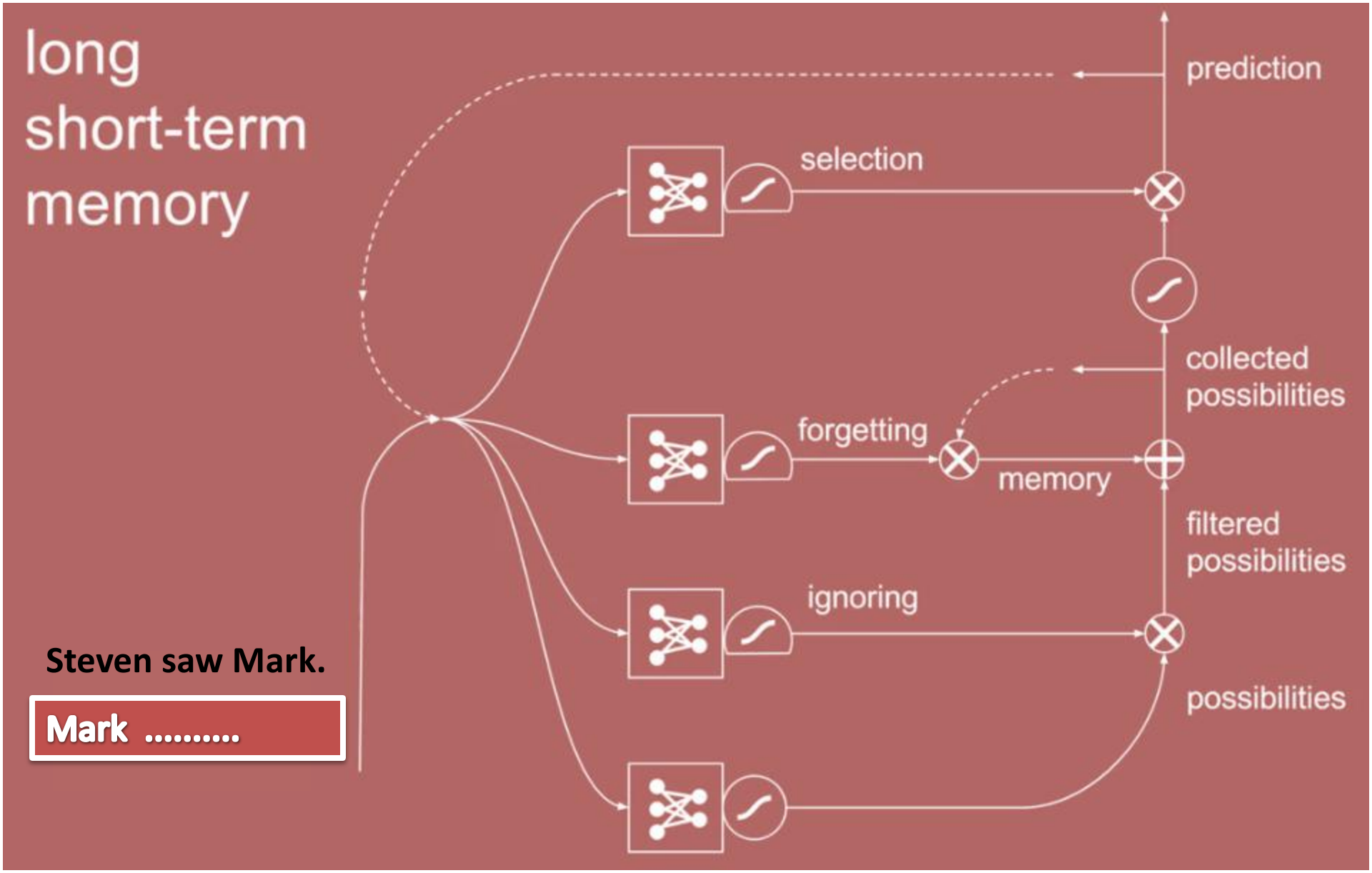

Signal
0.8
0.8
0.8
on/off gating
1.0
0.5
0.0
0.8 x 1.0
0.8 x 0.5
0.8 x 0.0
0.8
0.4
0.0

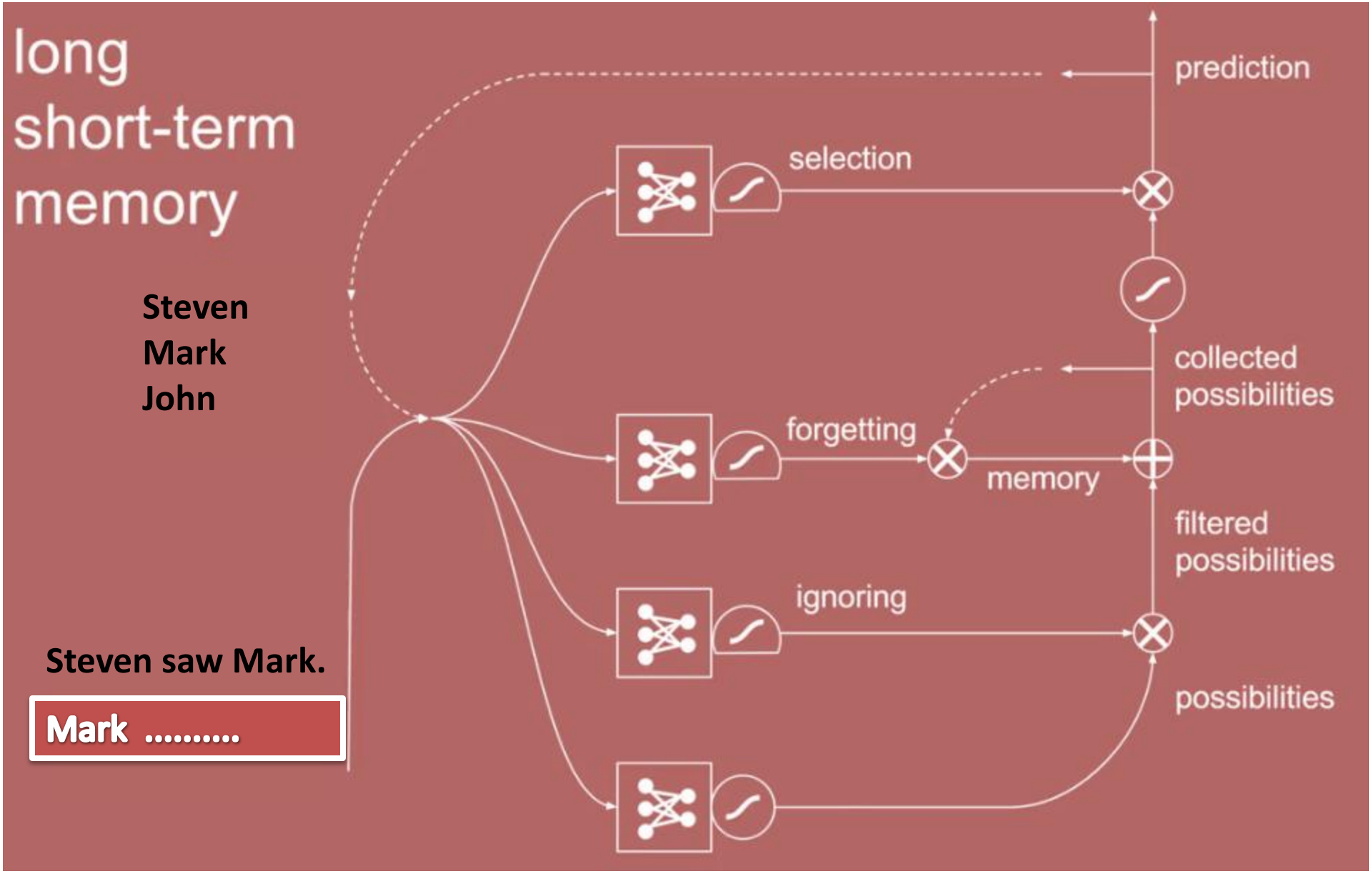
Input on which Machine was trained

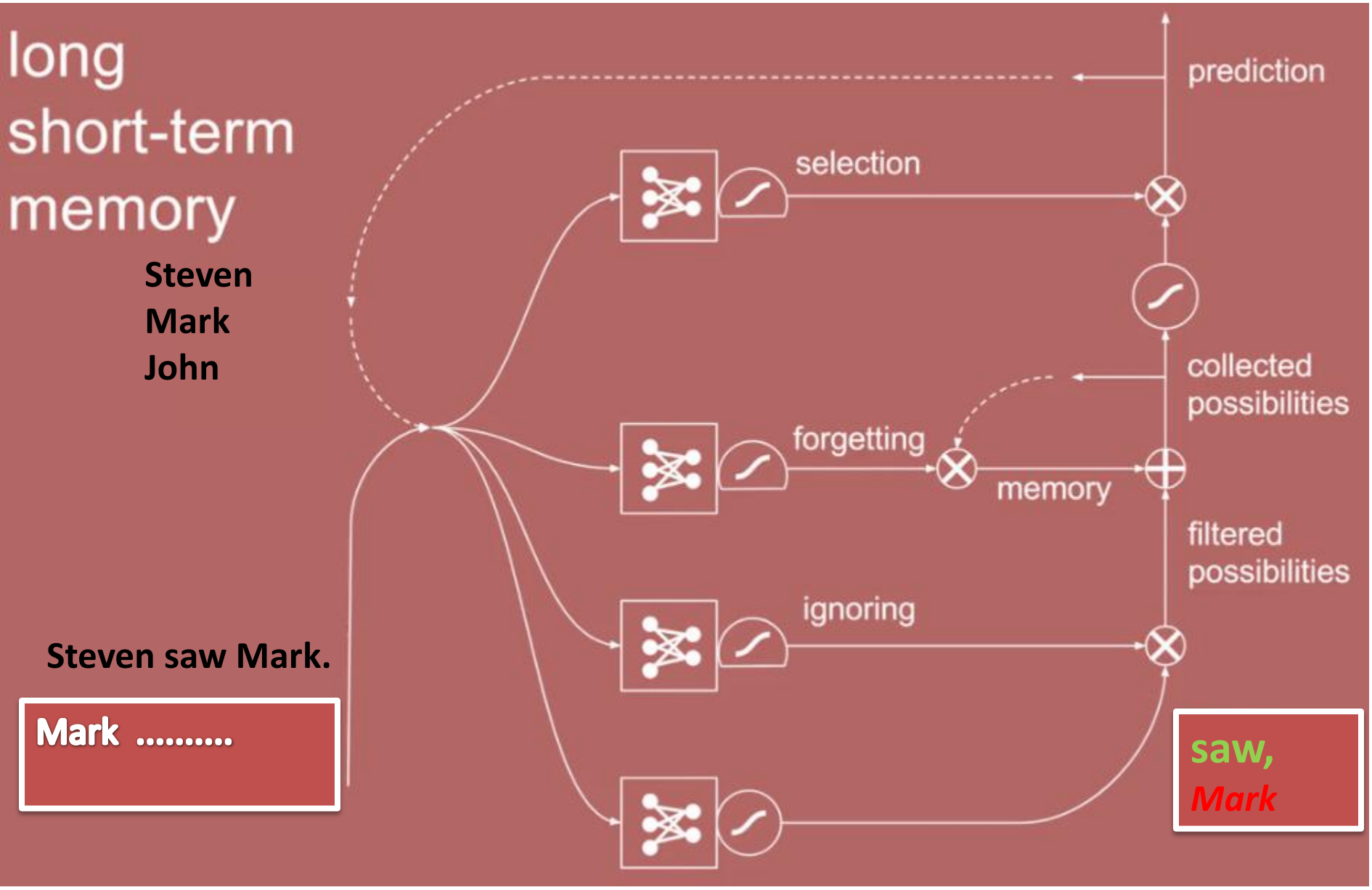
- Steven saw Mark.
- Mark saw John.
- John saw Steven.

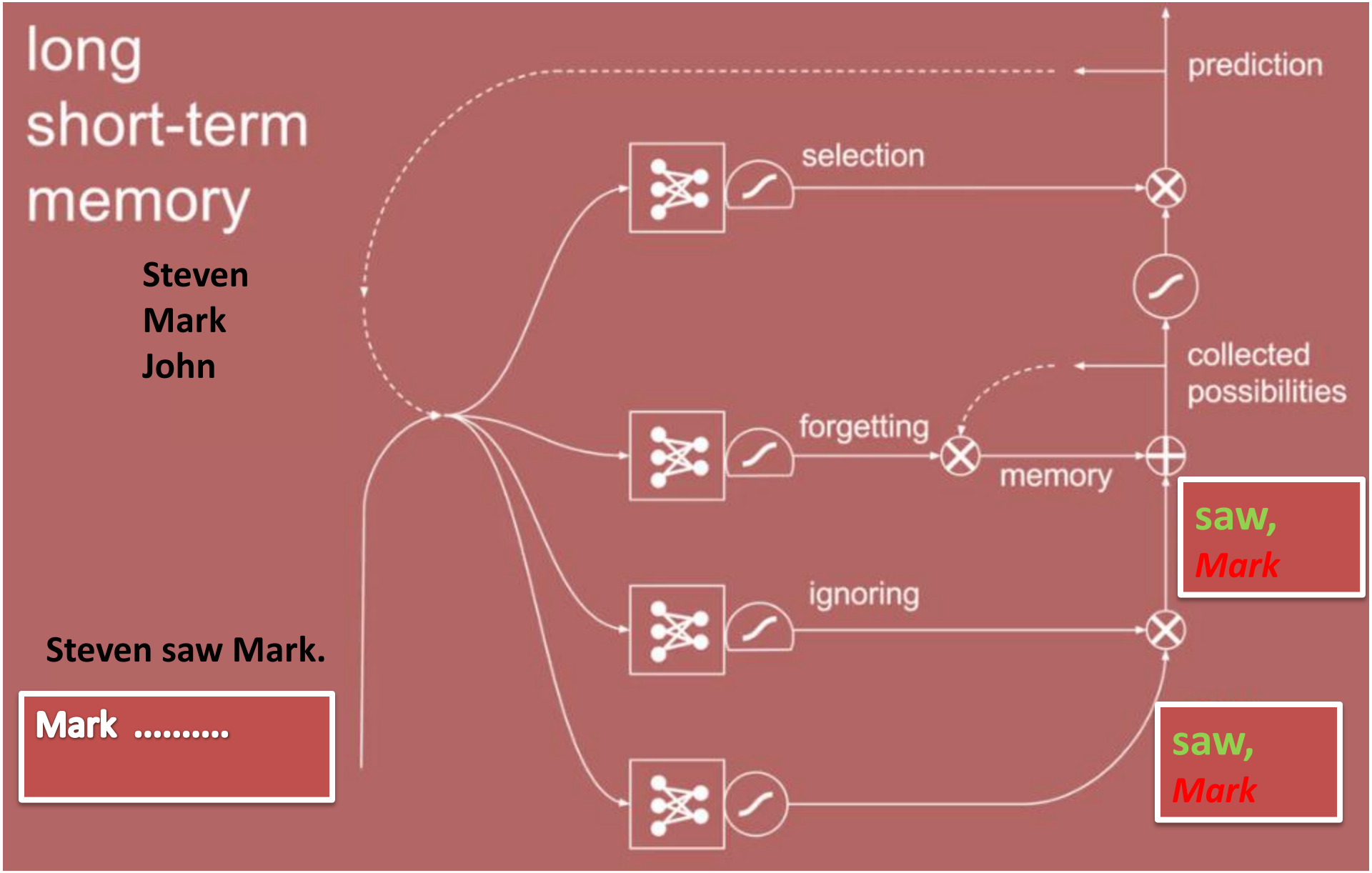

memory
prediction
selection
collected
possibilities
forgetting
memory
new
information
possibilities

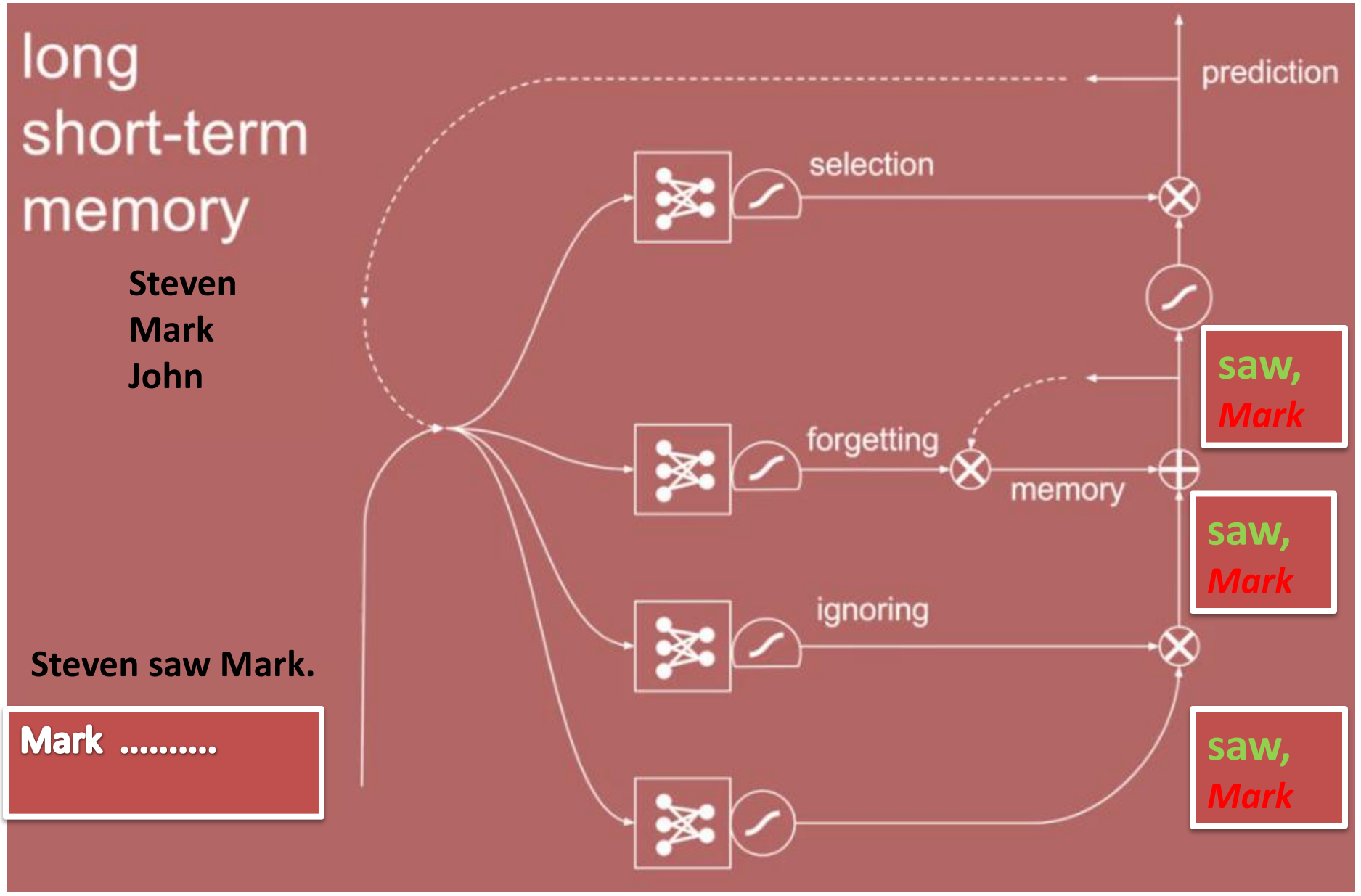

long
short-term
memory
prediction
selection
collected
possibilities
forgetting
memory
filtered
possibilities
ignoring
possibilities
new
information

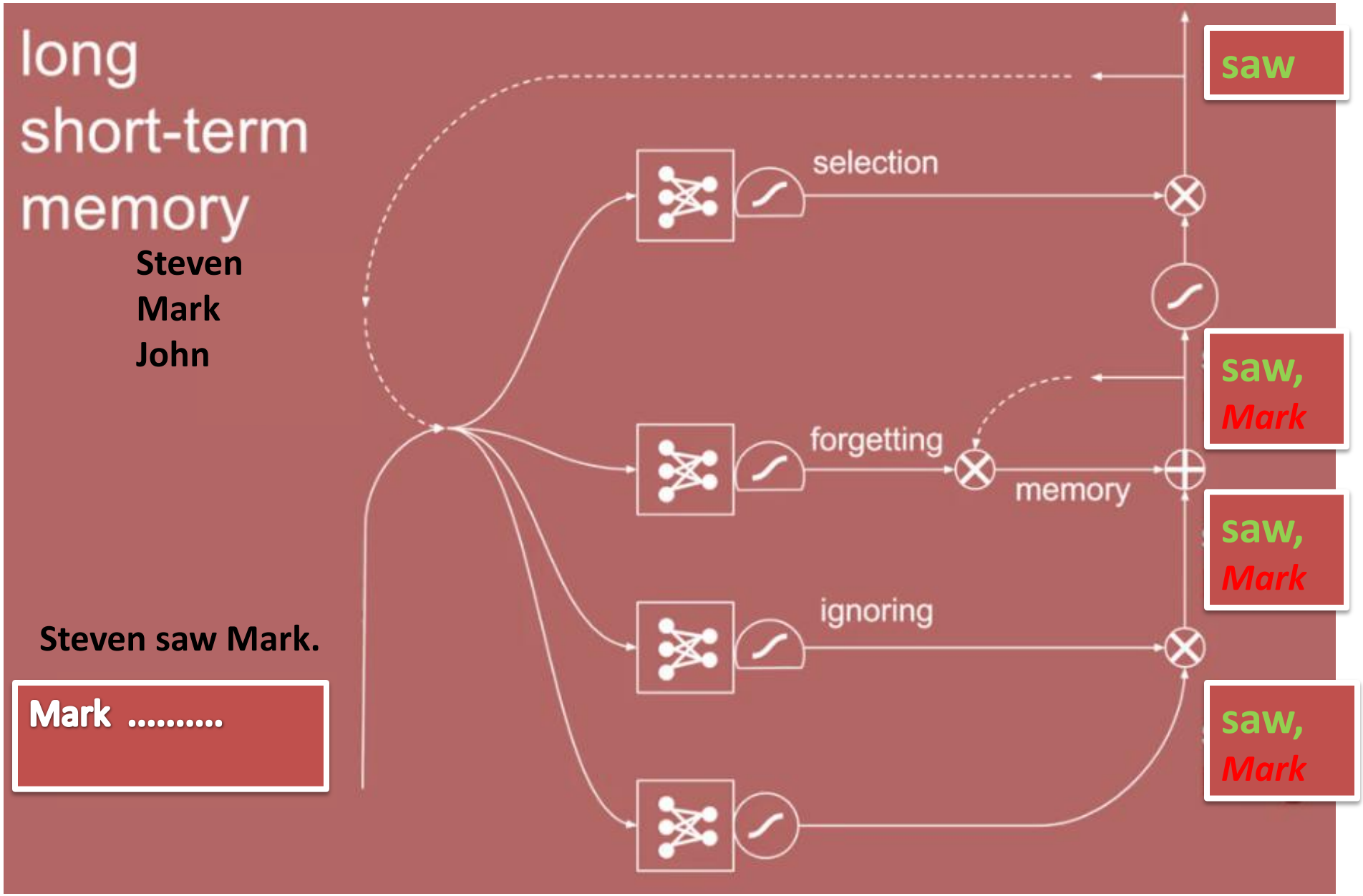

long
short-term
memory
prediction
selection
collected
possibilities
forgetting
memory
filtered
possibilities
ignoring
possibilities
Steven saw Mark.
Mark .....

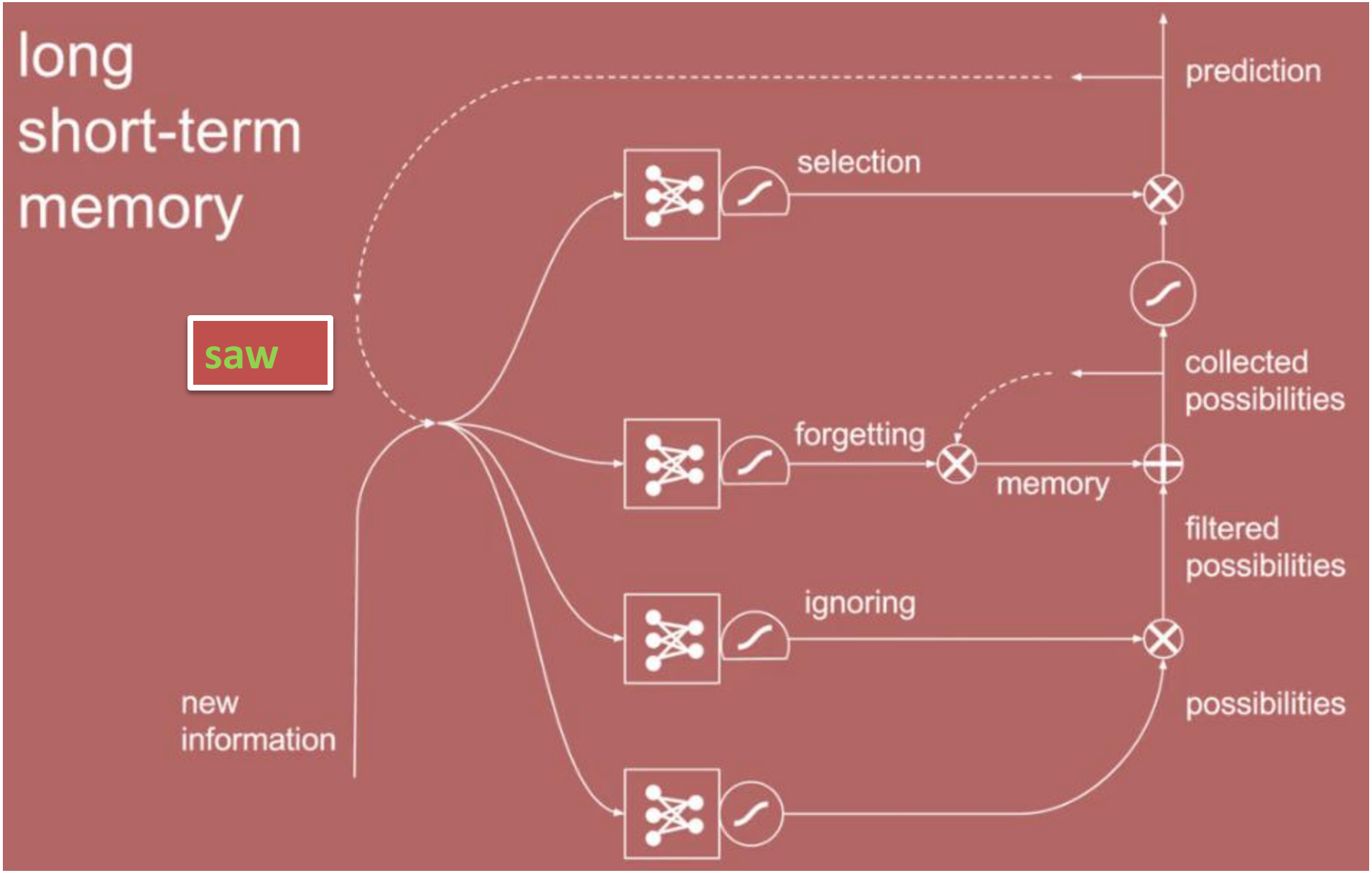

long
short-term
memory
Steven
Mark
John
Steven saw Mark.
Mark .....
prediction
selection
collected
possibilities
forgetting
memory
filtered
possibilities
ignoring
possibilities

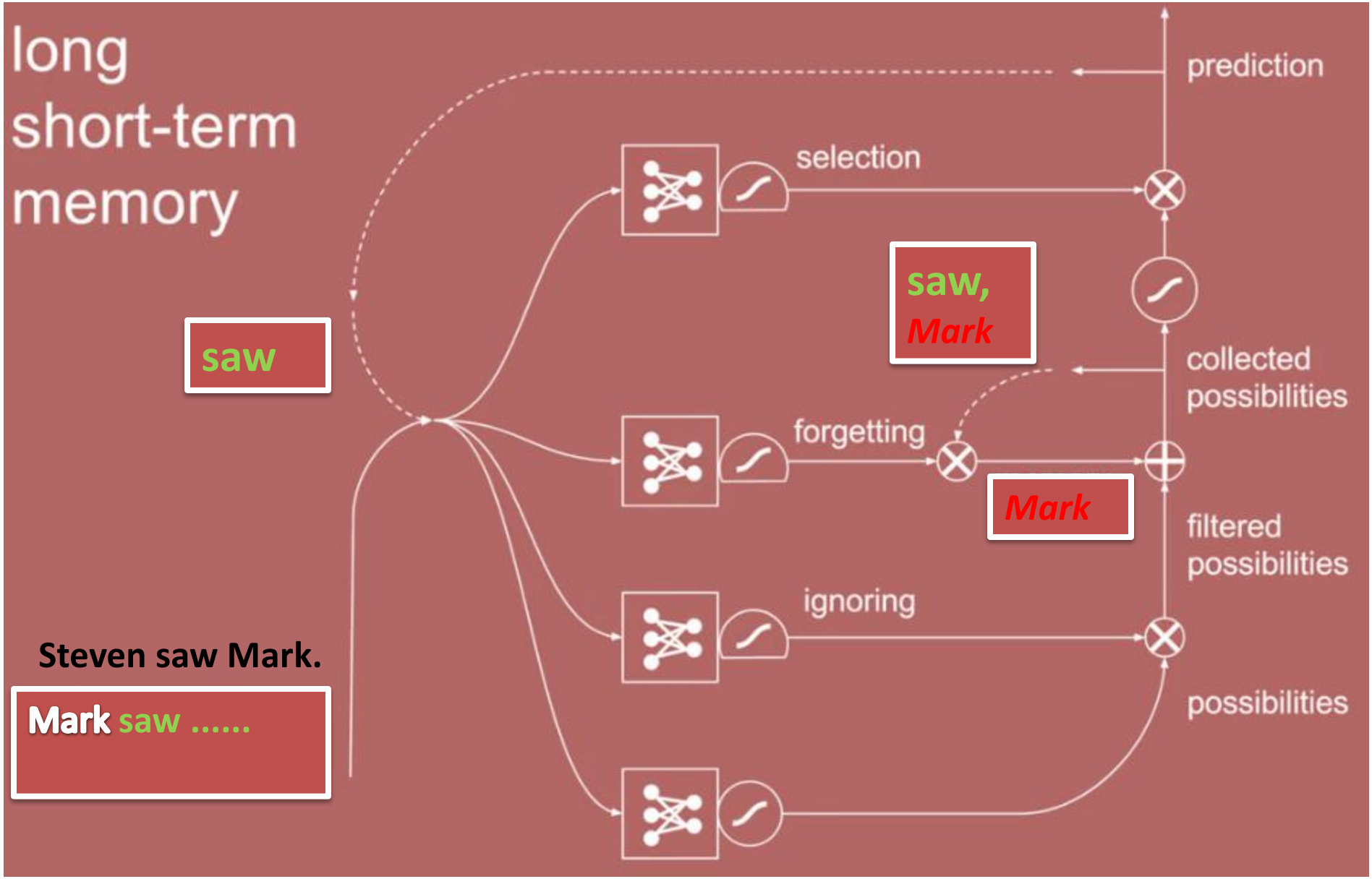

long
short-term
memory
Steven
Mark
John
Steven saw Mark.
Mark .....
prediction
selection
collected
possibilities
forgetting
memory
filtered
possibilities
ignoring
saw,
Mark

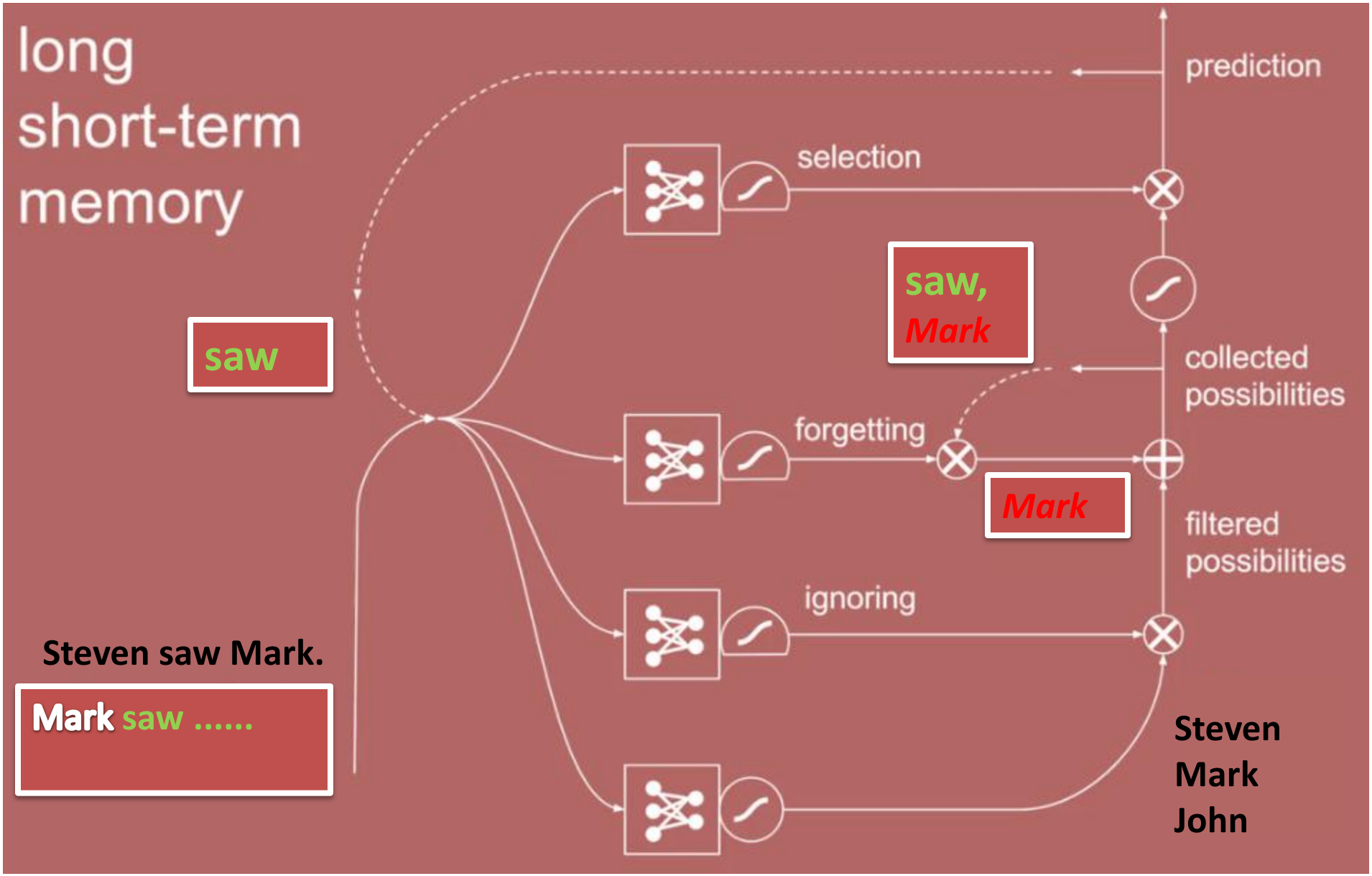

long
short-term
memory
Steven
Mark
John
prediction
selection
collected
possibilities
forgetting
memory
saw,
Mark
ignoring
Steven saw Mark.
Mark .....
saw,
Mark

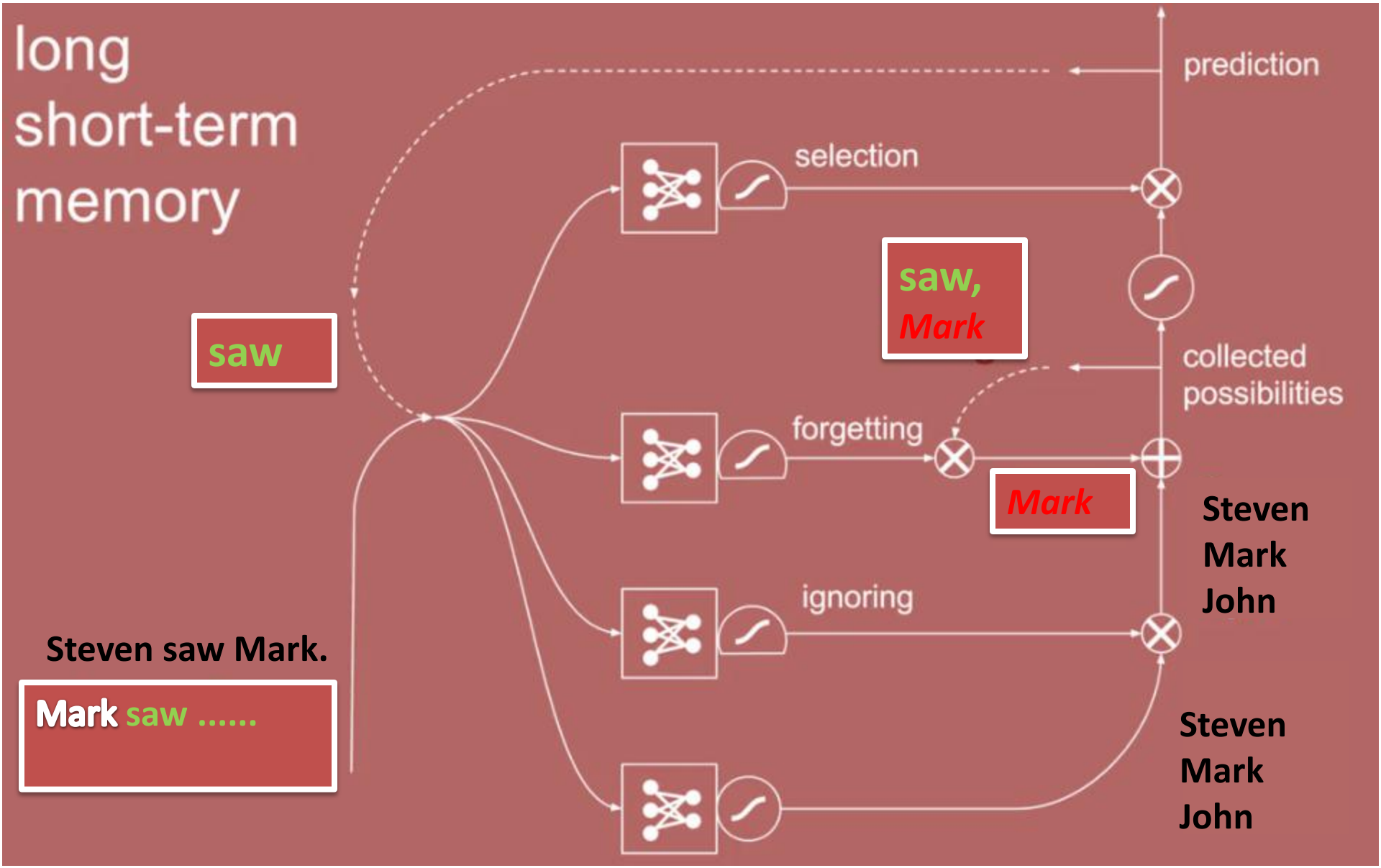

long
short-term
memory
Steven
Mark
John
Steven saw Mark.
Mark .....
prediction
selection
forgetting
memory
ignoring
saw,
Mark
saw,
Mark
saw,
Mark

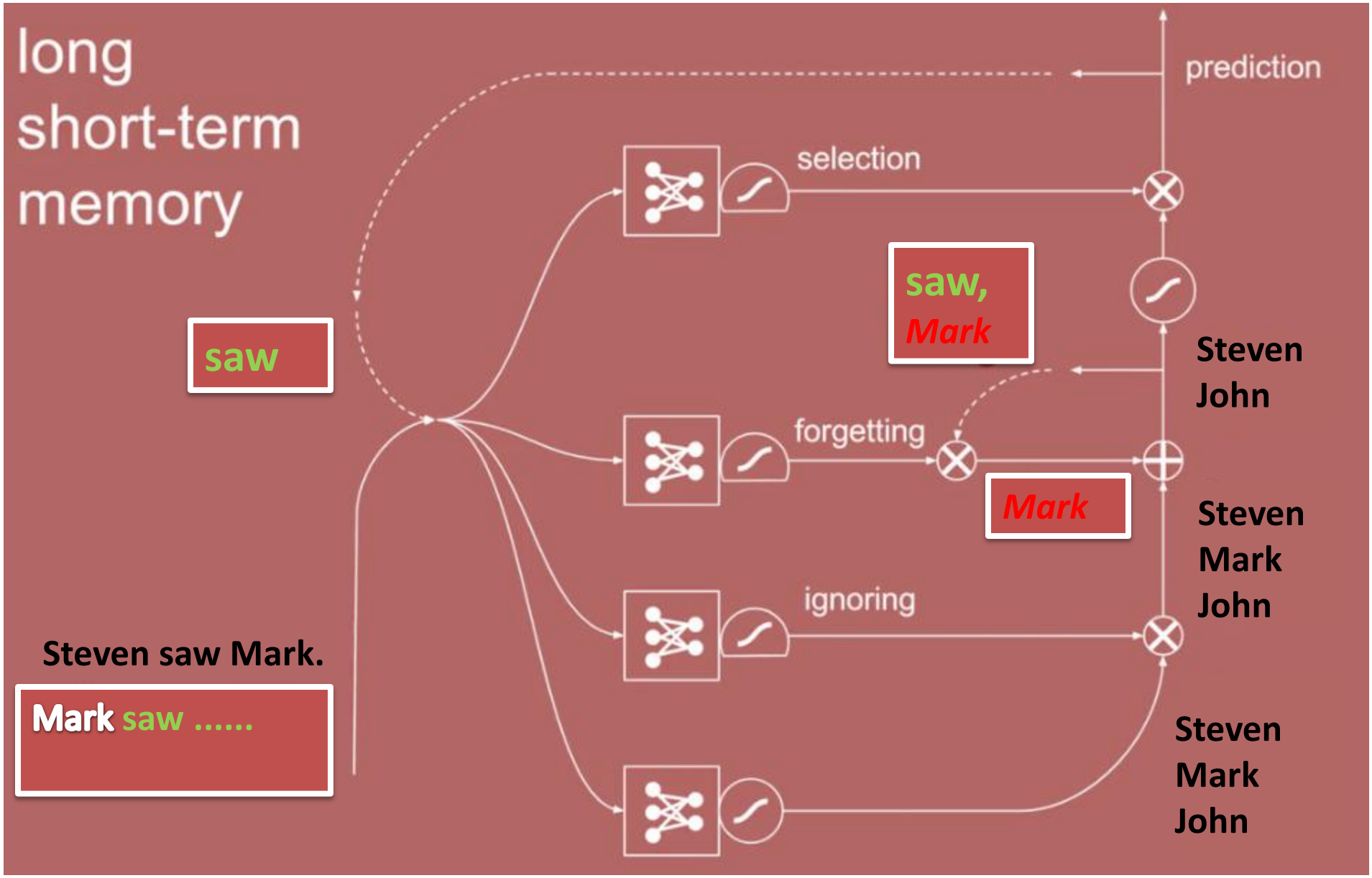

long
short-term
memory
Steven
Mark
John
Steven saw Mark.
Mark .....
selection
forgetting
memory
ignoring
saw
saw,
Mark
saw,
Mark
saw,
Mark

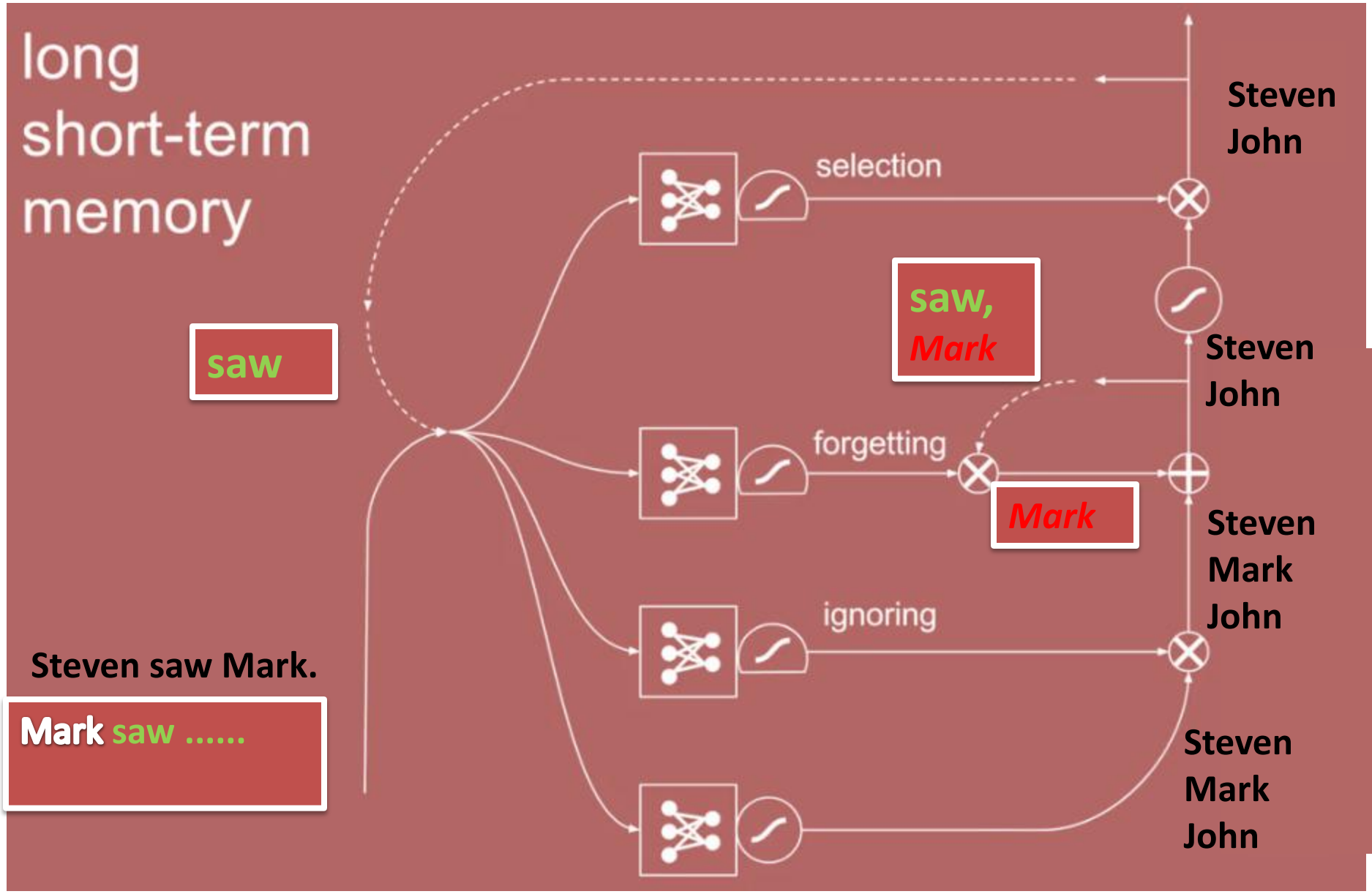

long
short-term
memory
saw
prediction
selection
collected
possibilities
forgetting
memory
filtered
possibilities
ignoring
new
information
possibilities


long
short-term
memory
prediction
selection
saw,
Mark
saw
collected
possibilities
forgetting
Mark
filtered
possibilities
ignoring
possibilities
Steven saw Mark.
Mark saw .....


long
short-term
memory
prediction
selection
saw,
Mark
saw
collected
possibilities
forgetting
Mark
filtered
possibilities
ignoring
Steven saw Mark.
Mark saw .....
Steven
Mark
John


long
short-term
memory
prediction
selection
saw,
Mark
saw
collected
possibilities
forgetting
Mark
Steven
Mark
John
ignoring
Steven saw Mark.
Mark saw .....
Steven
Mark
John


long
short-term
memory
prediction
selection
saw,
Mark
saw
Steven
John
forgetting
Mark
Steven
Mark
John
ignoring
Steven saw Mark.
Mark saw .....
Steven
Mark
John


long
short-term
memory
saw
selection
saw,
Mark
forgetting
Mark
ignoring
Steven
John
Steven
John
Steven
Mark
John
Steven
Mark
John
Steven saw Mark.
Mark saw .....

Recurrent Neural Network

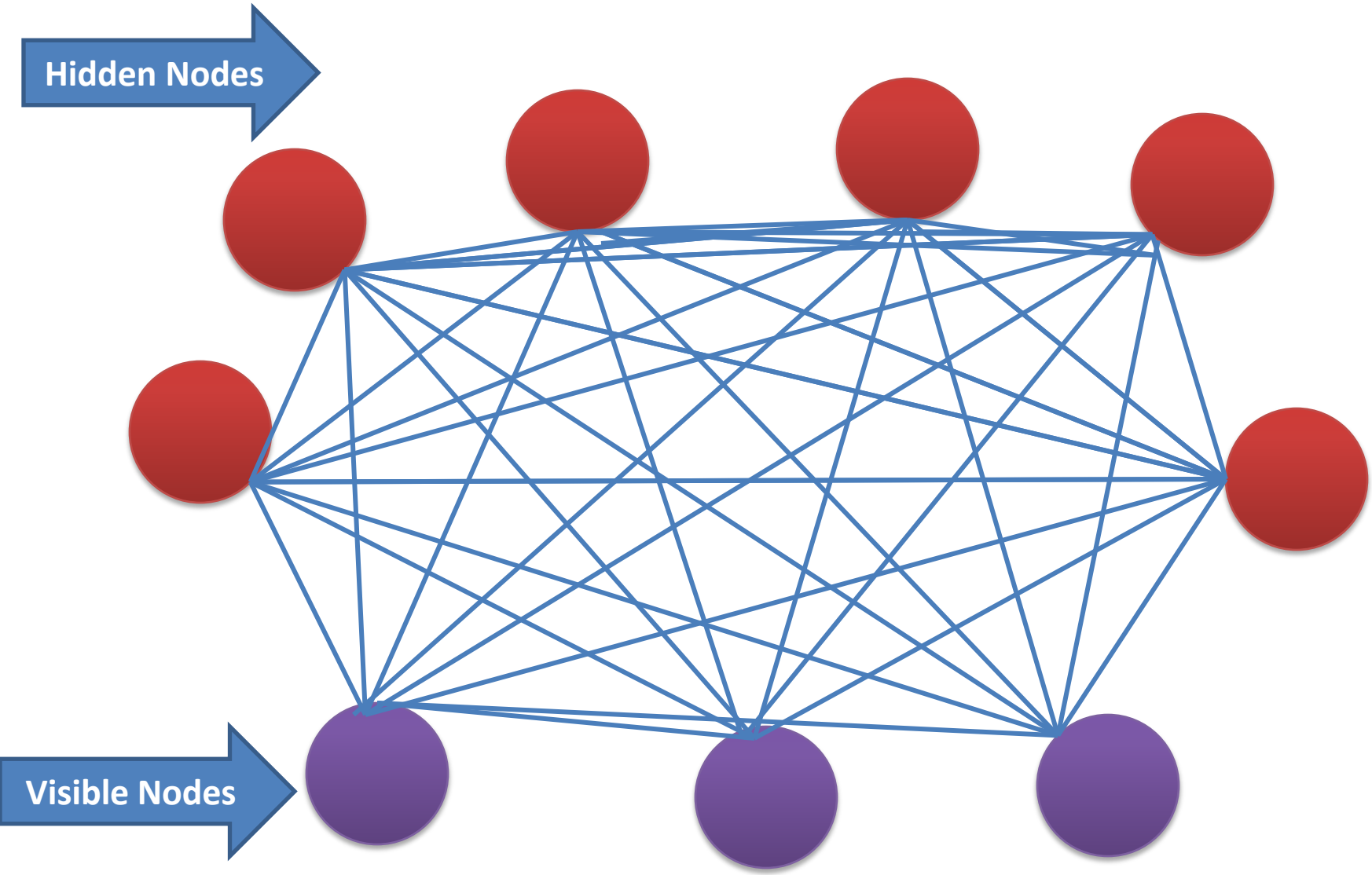
- **Use cases of recurrent neural networks**
- Machine translation (English --> French)
- Speech to text
- Market prediction
- Scene labelling (Combined with CNN)
- Car wheel steering. (Combined with CNN)

UnSupervised Learning

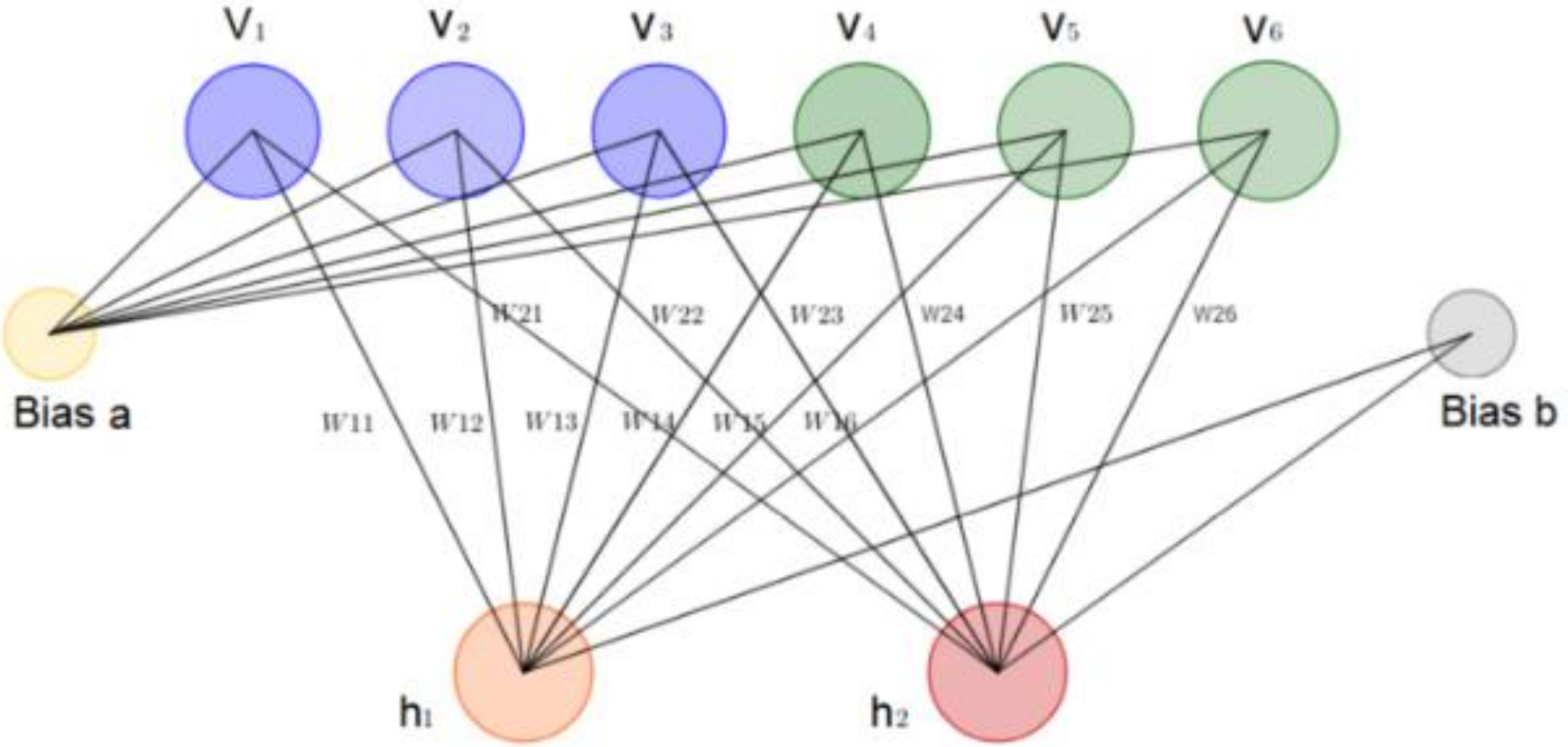
Boltzmann Machine

- Are undirected models

Boltzmann Machine



Restricted Boltzmann Machine



An Energy-Based-Model

- Energy is a term that may not be associated with deep learning in the first place.
- Rather is energy a quantitative property of physics. E.g. gravitational energy describes the potential energy a body with mass has in relation to another massive object due to gravity.
- Yet some deep learning architectures use the idea of energy as a metric for measurement of the models quality.

An Energy-Based-Model

- One purpose of deep learning models is to encode dependencies between variables.
- Capturing of dependencies happen through associating of a scalar energy to each configuration of the variables, which serves as a measure of compatibility.
- A high energy means a bad compatibility. An energy based model tries always to minimize a predefined energy function.

Energy Function

$$E(\mathbf{v}, \mathbf{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

E energy

v visible nodes

h hidden nodes

a bias for visible nodes

b bias for hidden nodes

i number of visible nodes

j number of hidden nodes

w weight

Energy Function

$$E(\mathbf{v}, \mathbf{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

- As it can be noticed the value of the energy function depends on the configurations of visible/input states, hidden states, weights and biases.
- The training of RBM consists in finding of parameters for given input values so that the energy reaches a minimum.

A probabilistic Model

- Restricted Boltzmann Machines are probabilistic.
- As opposed to assigning discrete values the model assigns probabilities.
- At each point in time the RBM is in a certain state.
- The state refers to the values of neurons in the visible and hidden layers **v** and **h**.
- The probability that a certain state of **v** and **h** can be observed is given by the following joint distribution:

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})}$$

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$$

Joint Distribution for v and h

- Here **Z** is called the 'partition function' that is the summation over all possible pairs of visible and hidden vectors.
- The joint distribution is known in Physics as the Boltzmann Distribution which gives the probability that a particle can be observed in the state with the energy *E*.

- Unfortunately it is very difficult to calculate the joint probability due to the huge number of possible combination of **v** and **h** in the partition function **Z**.
- Much easier is the calculation of the conditional probabilities of state **h** given the state **v** and conditional probabilities of state **v** given the state **h**:

$$p(\mathbf{h}|\mathbf{v}) = \prod_i p(h_i|\mathbf{v})$$

$$p(\mathbf{v}|\mathbf{h}) = \prod_i p(v_i|\mathbf{h})$$

Conditional probabilities for h and v

- Each neuron in a RBM can only exist in a binary state of 0 or 1
- The most interesting factor is the probability that a hidden or visible layer neuron is in the state 1 — hence activated.
- Given an input vector **v** the probability for a single hidden neuron *j* being activated is:

$$p(h_j = 1|\mathbf{v}) = \frac{1}{1 + e^{(-(b_j + W_j v_i))}} = \sigma(b_j + \sum_i v_i w_{ij})$$

Conditional probability for one hidden neuron, given v.

- the probability that a binary state of a visible neuron *i* is set to 1 is:

$$p(v_i = 1|\mathbf{h}) = \frac{1}{1 + e^{(-(a_i + W_i h_j))}} = \sigma(a_i + \sum_j h_j w_{ij})$$

Conditional probability for one visible neuron, given h.

Collaborative Filtering with Restricted Boltzmann Machines

Recognizing Latent Factors in The Data

- Assume some people were asked to rate a set of movies on a scale of 1–5 stars.
- In classical factor analysis each movie could be explained in terms of a set of latent factors.
- For example, movies like ***Harry Potter*** and ***Fast and the Furious*** might have strong associations with a latent factors of ***fantasy*** **and** ***action***.
- On the other hand users who like ***Toy Story*** **and** ***Cars*** might have strong associations with latent ***Pixar*** **factor.**

Recognizing Latent Factors in The Data

- RBMs are used to analyse and find out these underlying factors.
- After some epochs of the training phase the neural network has seen all ratings in the training data set of each user multiple times.
- At this time the model should have learned the underlying hidden factors based on users preferences and corresponding collaborative movie tastes of all users.

Recognizing Latent Factors in The Data

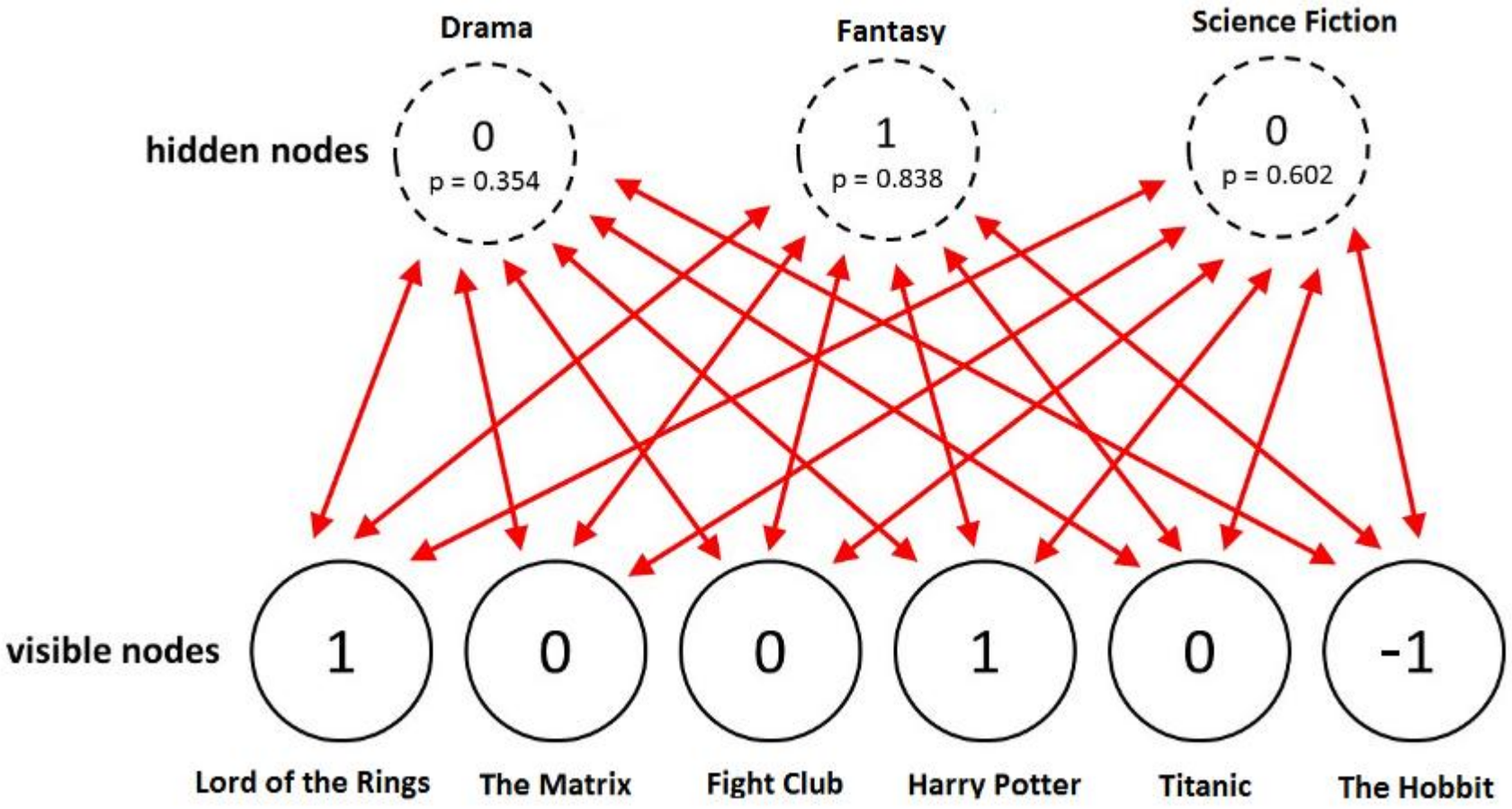
- The analysis of hidden factors is performed in a binary way.
- The binary rating values represent the inputs for the visible layer.
- Given the inputs the RMB then tries to discover latent factors in the data that can explain the movie choices.
- Each hidden neuron represents one of the latent factors.

Recognizing Latent Factors in The Data

- It is necessary to give yet unrated movies also a value, e.g. -1.0

- So that the network can identify the unrated movies during training time and ignore the weights associated with them.

Identification of Latent factor



Given these inputs the Boltzmann Machine may identify three hidden factors *Drama*, *Fantasy* and *Science Fiction* which correspond to the movie genres.

Identification of Latent factor

- Given the movies the RMB assigns a probability **p(h|v)** for each hidden neuron.

$$p(h_j = 1|\mathbf{v}) = \frac{1}{1 + e^{(-(b_j + W_j v_i))}} = \sigma(b_j + \sum_i v_i w_{ij})$$

Conditional probability for one hidden neuron, given v.

- The final binary values of the neurons are obtained by sampling from **Bernoulli distribution** using the probability **p**.

Bernoulli distribution

- A **Bernoulli distribution** has only two possible outcomes, namely 1 (success) and 0 (failure), and a single trial.

- So the random variable X which has a Bernoulli distribution can take value 1 with the probability of success, say p, and the value 0 with the probability of failure, say q or 1-p.

- The probability mass function is given by:
  $p^x(1-p)^{1-x}$ where x € (0, 1)

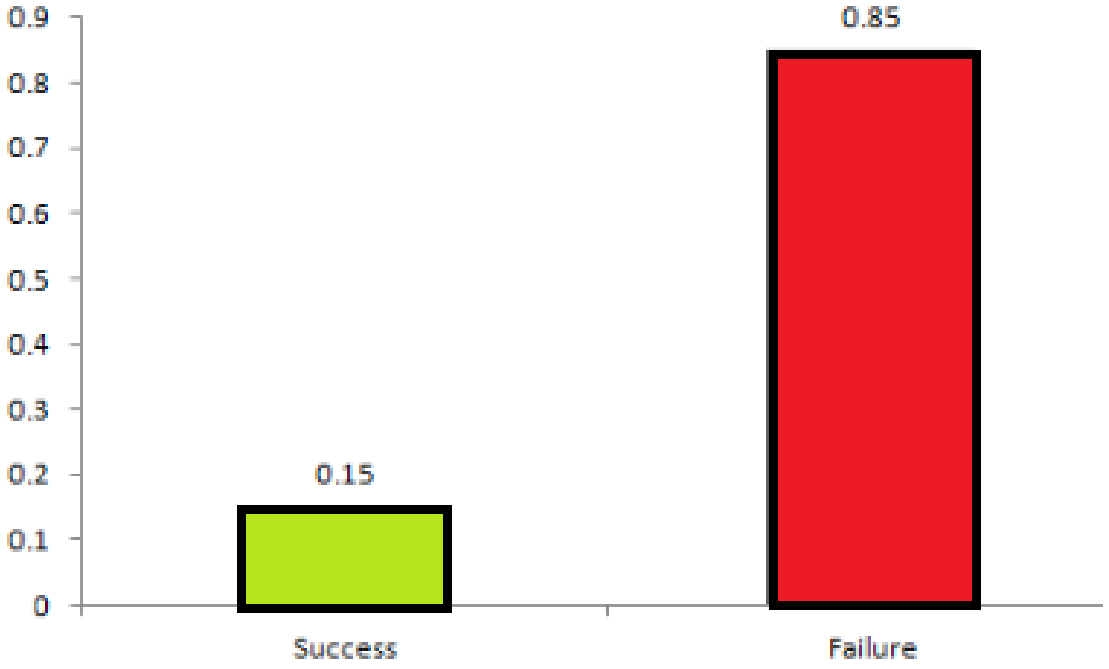
  It can also be written as $$P(x) = \begin{cases} 1-p, & x = 0 \\ p, & x = 1 \end{cases}$$

Bernoulli distribution

- The probabilities of success and failure need not be equally likely

The chart below shows the Bernoulli Distribution of a chess game between a normal person and grandmaster Viswanathan Anand

Viswanathan Anand is pretty much certain to win. So in this case probability of a normal person's success is 0.15 while that person's failure is 0.85



Bernoulli distribution

- Expected value of any distribution is the mean of the distribution.
- The expected value of a random variable X from a Bernoulli distribution is found as follows:

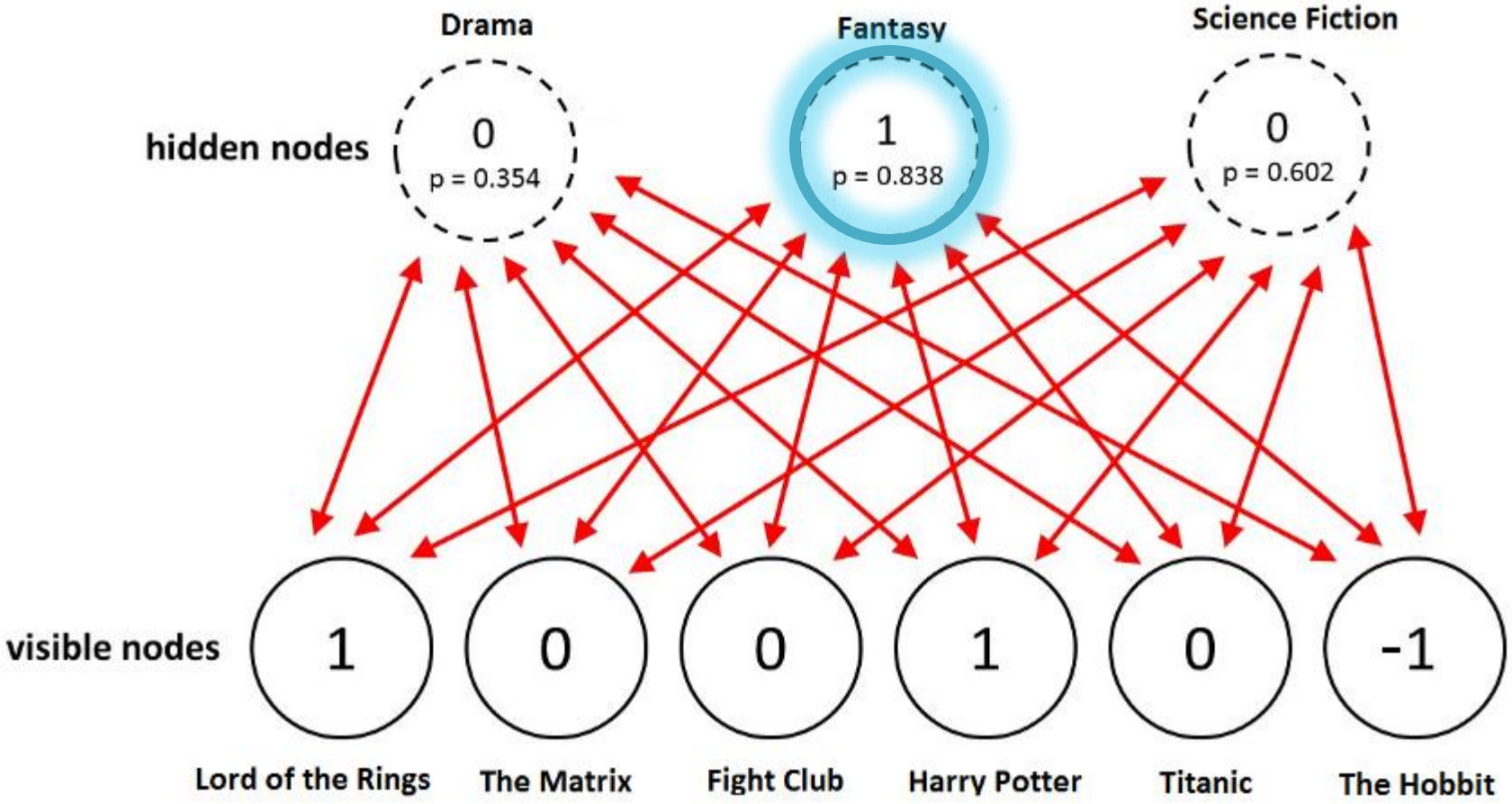
$$\mathbf{E(X) = 1*p + 0*(1-p) = p}$$

- The variance of a random variable from a Bernoulli distribution is:

$$\mathbf{V(X) = E(X^2) - [E(X)]^2 = p - p^2 = p(1-p)}$$

Identification of Latent factor

- In this example only the hidden neuron that represents the genre *Fantasy* becomes activate.



Using Latent Factors for Prediction

- After the training phase the goal is to predict a binary rating for the movies that had not been seen yet.

- Given the training data of a specific user the network is able to identify the latent factors based on this users preference.

Using Latent Factors for Prediction

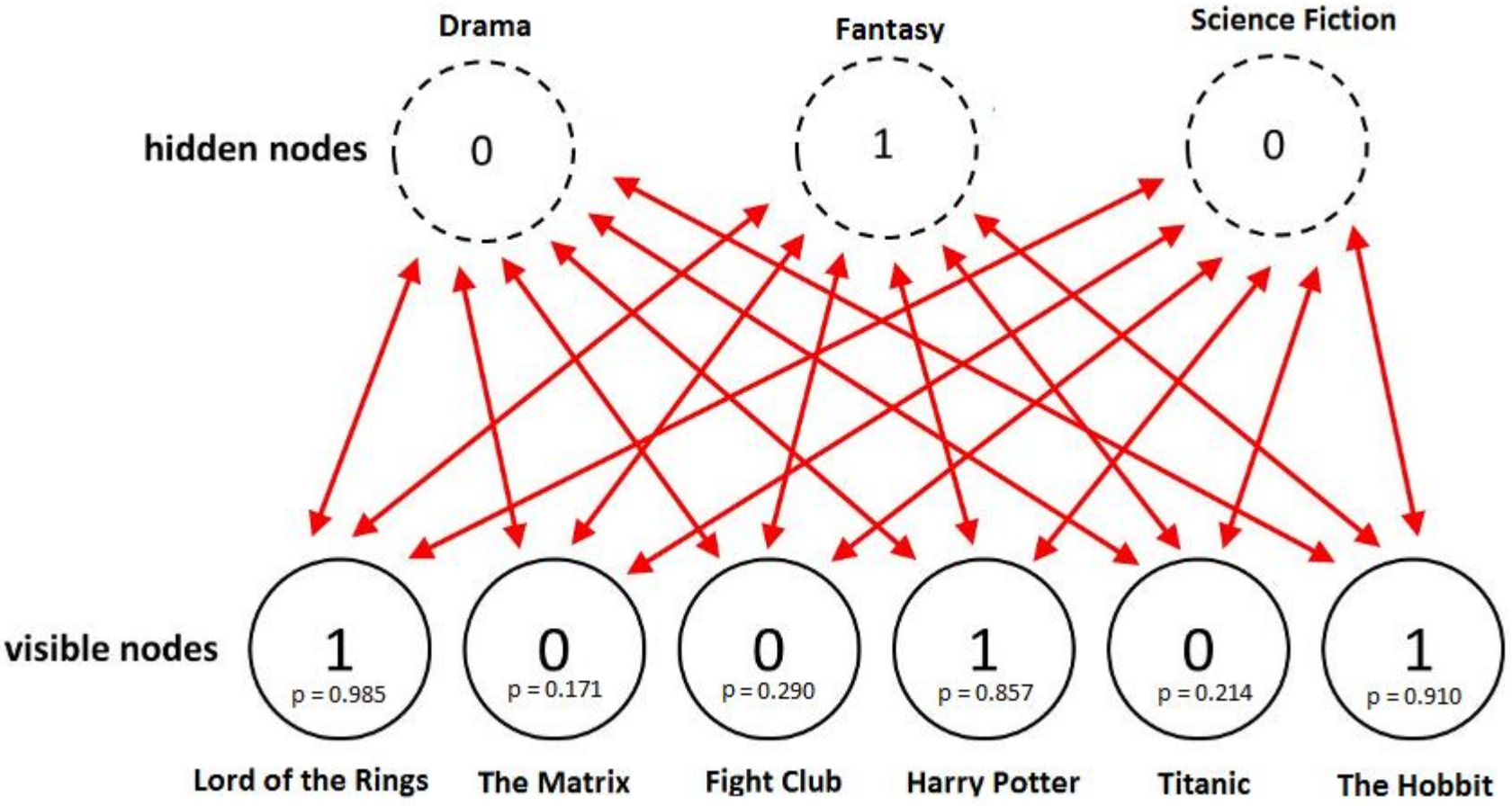
- Since the latent factors are represented by the hidden neurons we can use **p(v|h)**

$$p(v_i = 1|\mathbf{h}) = \frac{1}{1 + e^{(-(a_i + W_i h_j))}} = \sigma(a_i + \sum_j h_j w_{ij})$$

Conditional probability for one visible neuron, given h.

and sample from Bernoulli distribution to find out which of the visible neurons now become active.

Using the hidden neuron for inference



New ratings after using the hidden neuron values for the inference. The network did identified *Fantasy* as the preferred movie genre and rated *The Hobbit* as a movie the user would like.

To summarize the process from training to the prediction phase

1. Train the network on the data of all users
2. During inference time take the training data of a specific user
3. Use this data to obtain the activations of hidden neurons
4. Use the hidden neuron values to get the activations of input neurons
5. The new values of input neurons show the rating the user would give yet unseen movies

Training

- The training of the Restricted Boltzmann Machine differs from the training of a regular neural networks via stochastic gradient descent.
- The first part of the training is called ***Gibbs Sampling*.**
- The first part of the training is called ***Contrastive Divergence***.

Gibbs Sampling

- Given an input vector **v** we are using **p(h|v)** for prediction of the hidden values **h.**

$$p(h_j = 1|\mathbf{v}) = \frac{1}{1 + e^{(-(b_j + W_j v_i))}} = \sigma(b_j + \sum_i v_i w_{ij})$$

Conditional probability for one hidden neuron, given v.

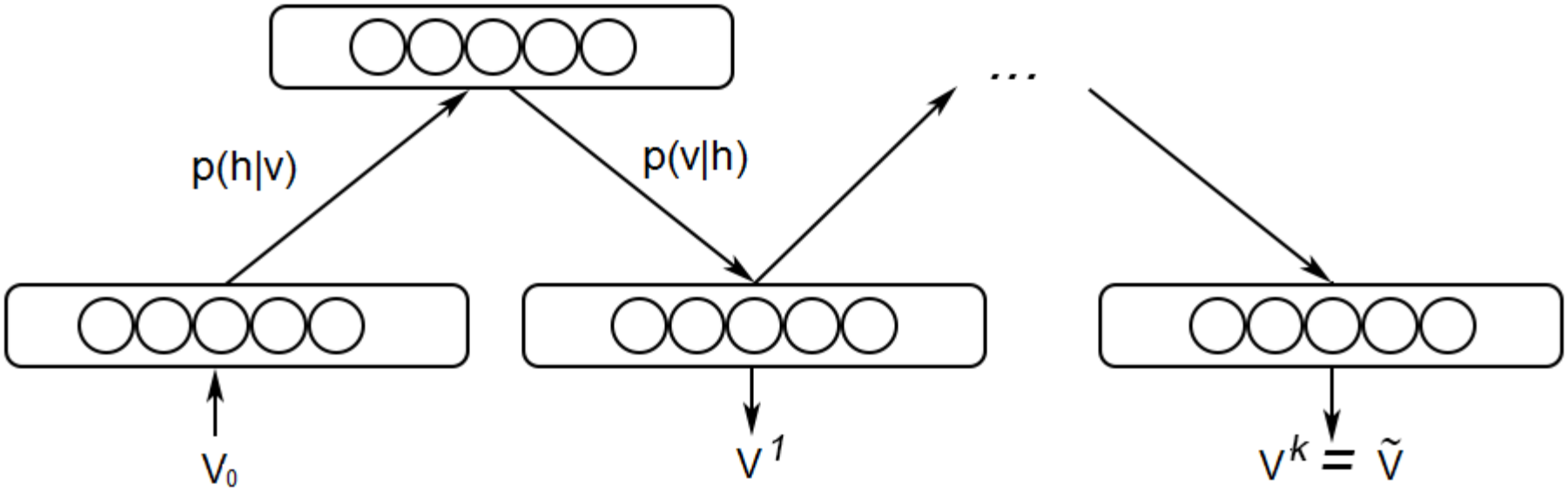
- Knowing the hidden values we use **p(v|h)** for prediction of new input values **v**.

$$p(v_i = 1|\mathbf{h}) = \frac{1}{1 + e^{(-(a_i + W_i h_j))}} = \sigma(a_i + \sum_j h_j w_{ij})$$

Conditional probability for one visible neuron, given h.

Gibbs Sampling

- This process is repeated *k* times.
  After *k* iterations we obtain an other input vector **v_k** which was recreated from original input values **v_0**.



Contrastive Divergence

- The update of the weight matrix happens during the *Contrastive Divergence* step.
- Vectors **v_0** and **v_k** are used to calculate the activation probabilities for hidden values **h_0** and **h_k**

$$p(h_j = 1|\mathbf{v}) = \frac{1}{1 + e^{(-(b_j + W_j v_i))}} = \sigma(b_j + \sum_i v_i w_{ij})$$

Conditional probability for one hidden neuron, given v.

Contrastive Divergence

- The difference between the outer products of those probabilities with input vectors **v_0** and **v_k** results in the update matrix:

$$\triangle W = \mathbf{v}_0 \otimes p(\mathbf{h}_0|\mathbf{v}_0) - \mathbf{v}_k \otimes p(\mathbf{h}_k|\mathbf{v}_k)$$

Update Matrix

Contrastive Divergence

- Using the update matrix the new weights can be calculated with gradient **ascent,** given by:

$$W_{new} = W_{old} + \triangle W$$

Update rule for the weights

Contrastive Divergence

- In RBMs there are no hidden-to-hidden or visible-to-visible connections, so influence flows just between hidden and visible units. Meaning we can now update them in parallel — first compute hidden units' activations, then visible, then hidden again, and so on until the network settles down to equilibrium. **That's called contrastive divergence**. And it turns out, this method works even if we make *one* iteration of it — when the network is certainly not in equilibrium, but still gets updated in the right direction.