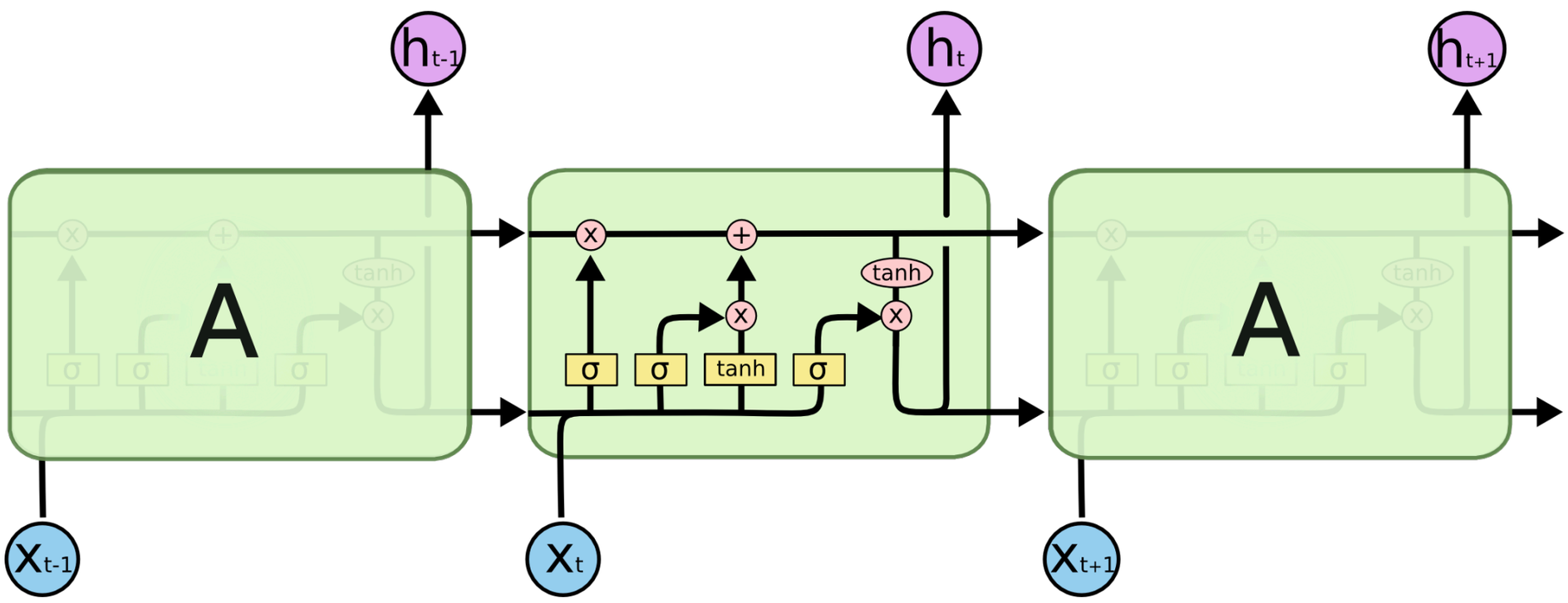
UNIVERSIDAD DEL ROSARIO

CENTER FOR ASTROPHYSICS
HARVARD & SMITHSONIAN

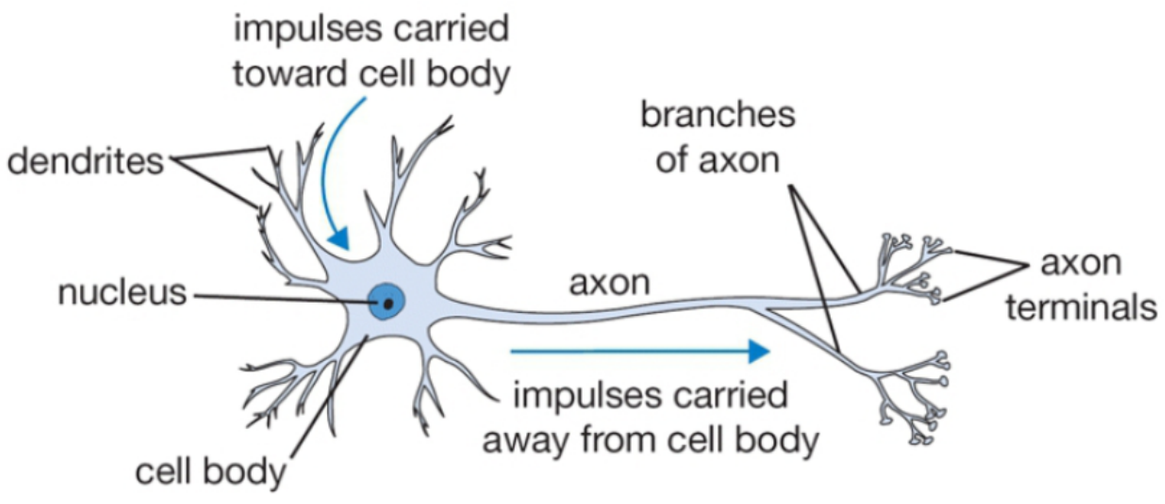
Recurrent Neural Networks (RNNs)

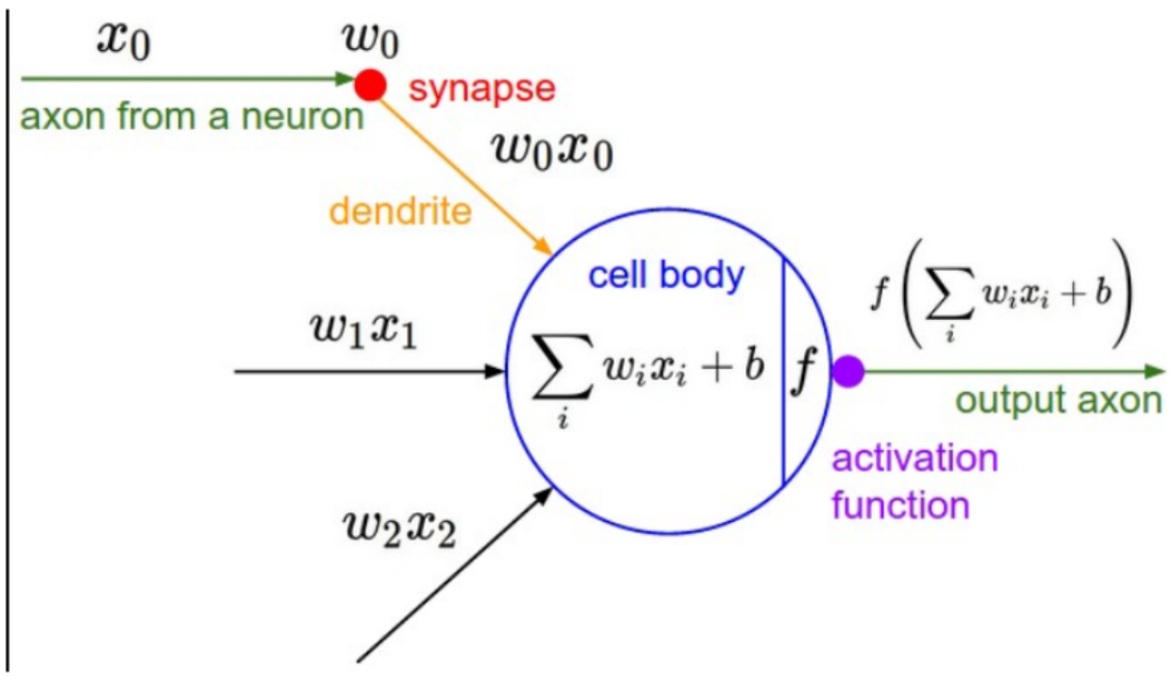
Rafael Martínez-Galarza



BigDataCo, December 12, 2020

A simple perceptron



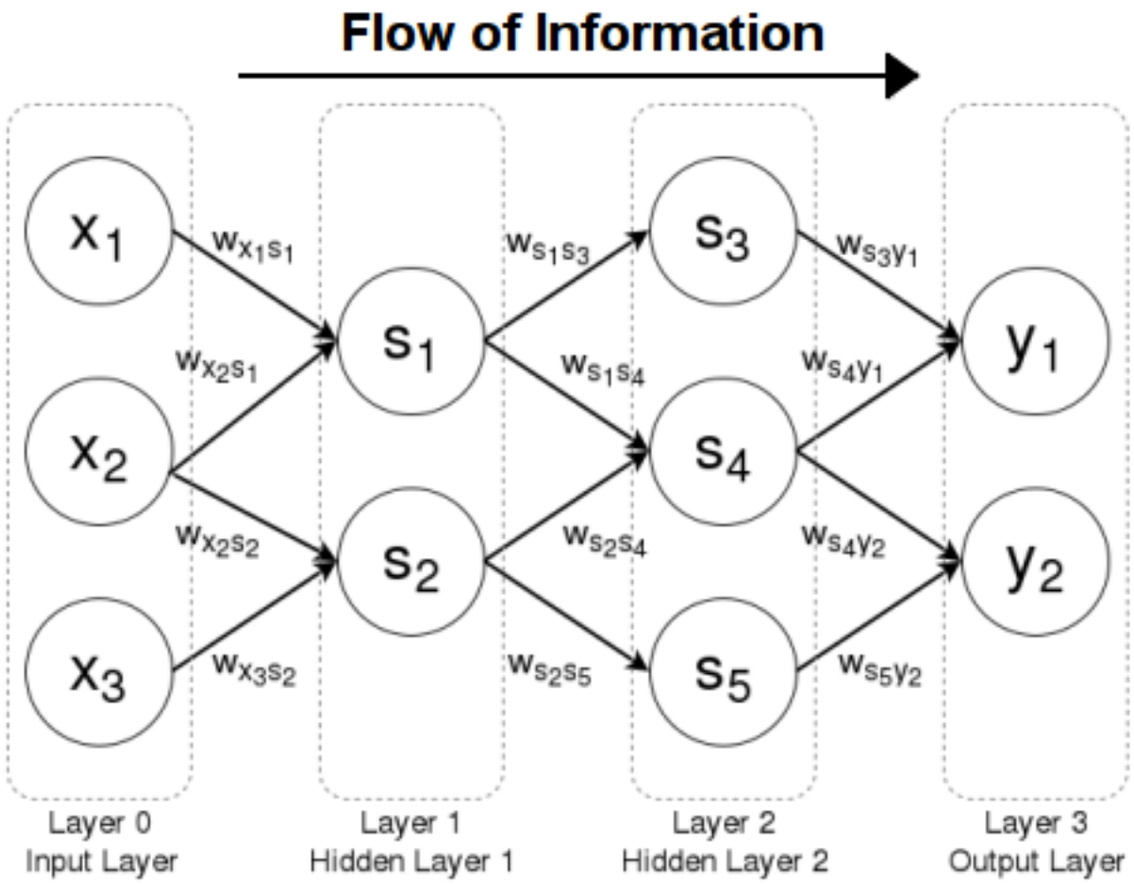


- A non-linear activation function is applied to the linear combination of the features.
- The loss (error) function that we want to minimize is minus the logarithm of the joint probability of the labels observed in the training set:

$$J(\theta) = -\sum_i \left(y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right)$$

Feedforward Networks

More in general

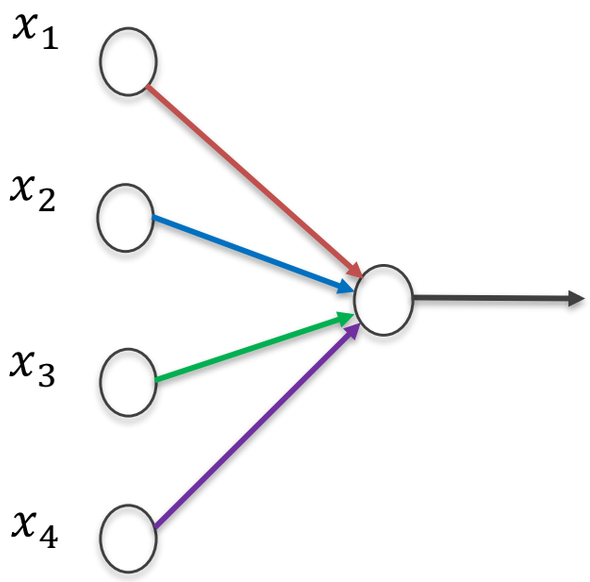


Feedforward: connections do not form a cycle

Computes a function *f* on fixed size input *x* such that *f(x) ~ y* for training pairs *(x,y)*

What is learning?

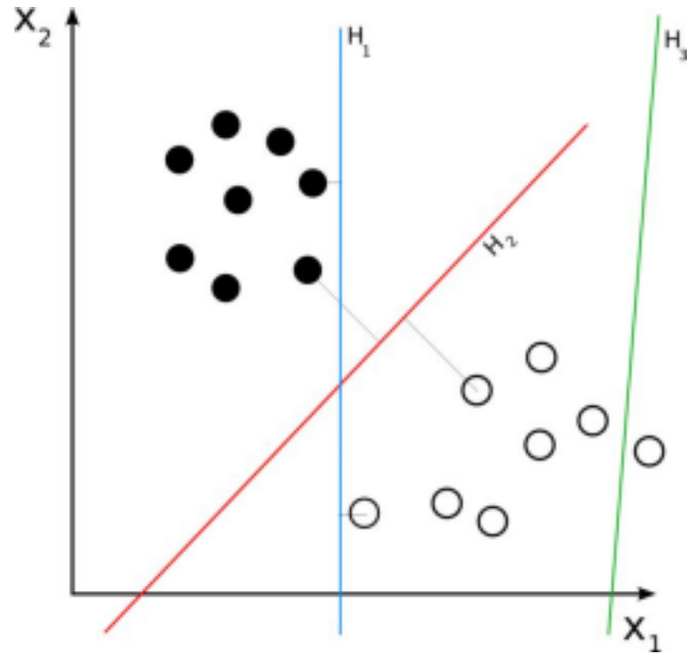
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



$$Y = \sigma(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4)$$

Intercept or **Bias**

Coefficients or **Weights**



$$o = g(\vec{w} \cdot \vec{x} + b).$$

- Learning is adjusting the weights and bias to find the best possible decision boundary.
- Learning is performed on the TRAINING DATA.
- Another way to see it: adjust the weights and bias so that g(w*x+b) -the probability- is high if dot is black and low otherwise.

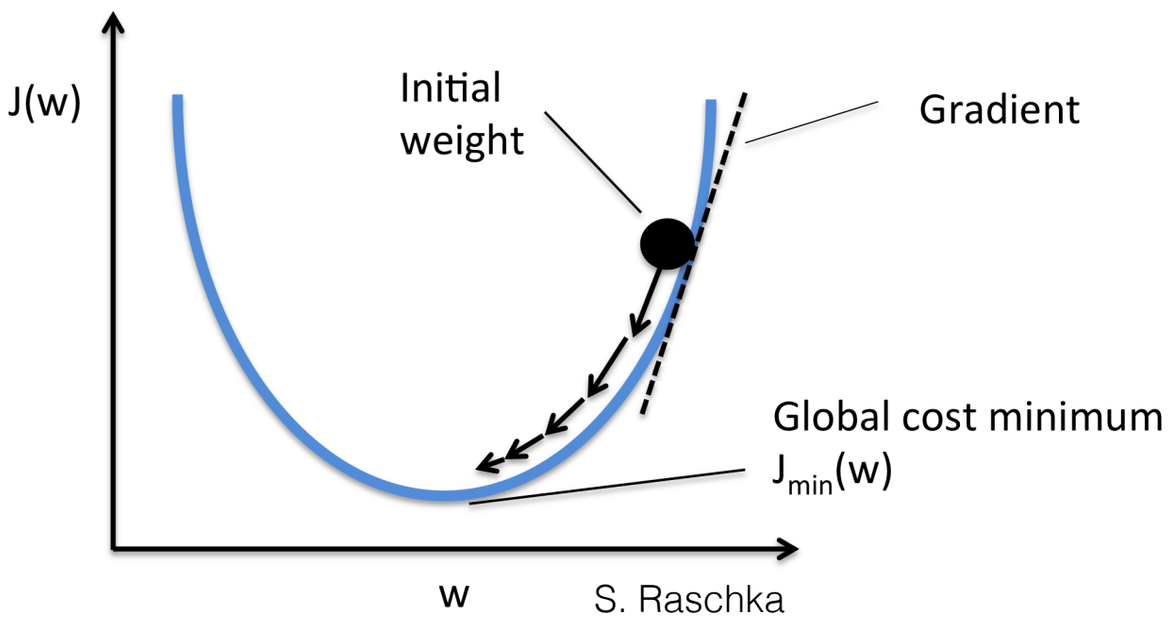
Error Function

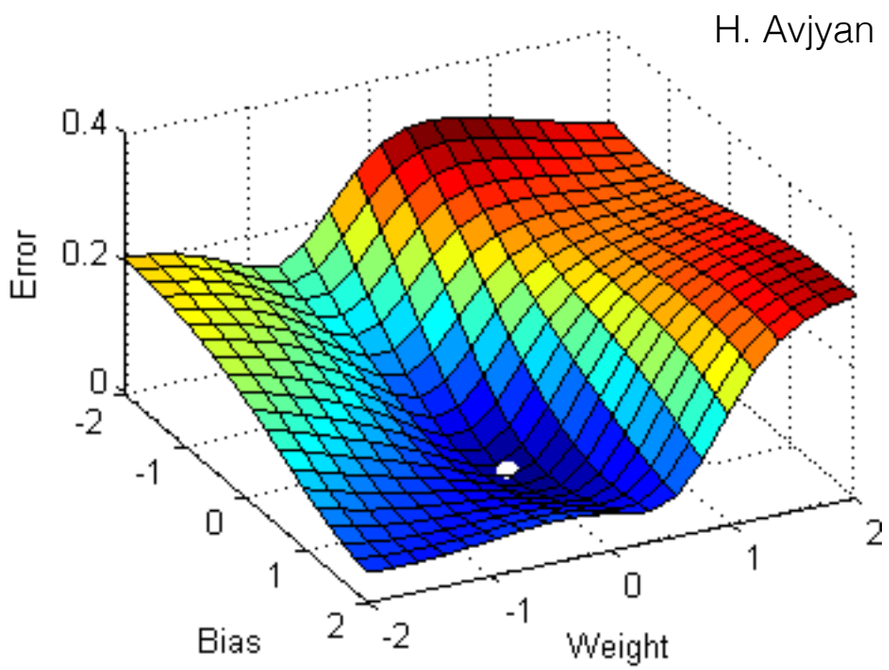
- How do we adjust the weights?
- We need to quantify the difference between the perceptron output and the true value y for an input **x**, over a set of input-output pairs.
- The mean square error (MSE) is a natural choice:

$$E(X) = \frac{1}{2N}\sum_{i=1}^{N}(o_i - y_i)^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(g(\vec{w}\cdot\overrightarrow{x_i} + b) - y_i\right)^2$$

- Note that E(X) = 0 when $o_i = y_i$ for all input-output pairs.
- Therefore, we want to MINIMIZE E(X)
- How do we do that?

Gradient Descent





- Start with random bias and weights
- Use gradient of error function w.r.t. the bias and weights at each point to update those values.
- Iterate until certain convergence criterium is met.

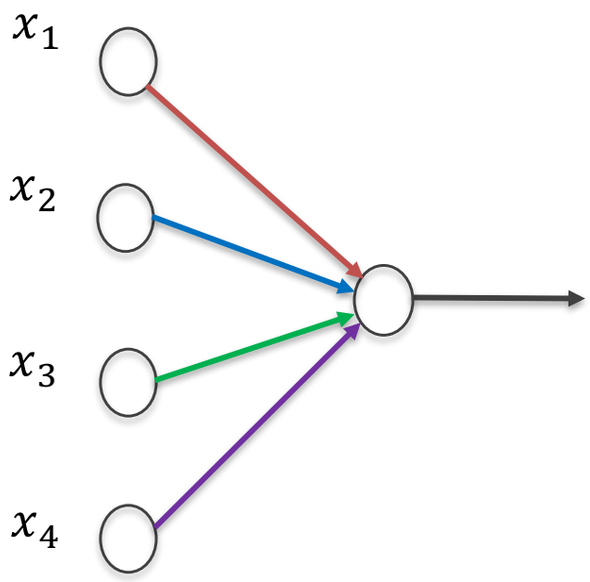
$$\overrightarrow{w_{i+1}} = \overrightarrow{w_i} - \alpha \frac{\partial E(X)}{\partial \overrightarrow{w_i}}$$

$$b_{i+1} = b_i - \alpha \frac{\partial E(X)}{\partial b_i}$$

Learning rate!

What is learning?

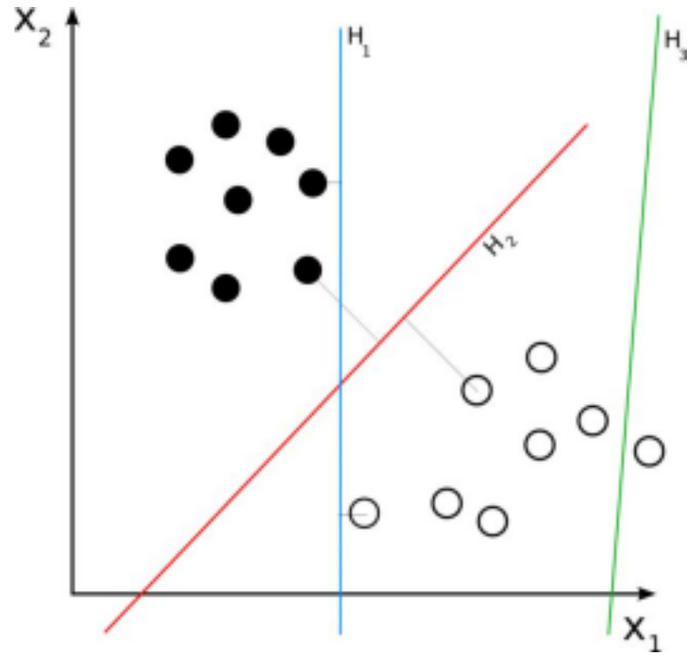
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



$$Y = \sigma(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4)$$

Intercept or **Bias**

Coefficients or **Weights**



$$o = g(\vec{w} \cdot \vec{x} + b).$$

- Learning is adjusting the weights and bias to find the best possible decision boundary.
- Learning is performed on the TRAINING DATA.
- Another way to see it: adjust the weights and bias so that g(w*x+b) -the probability- is high if dot is black and low otherwise.

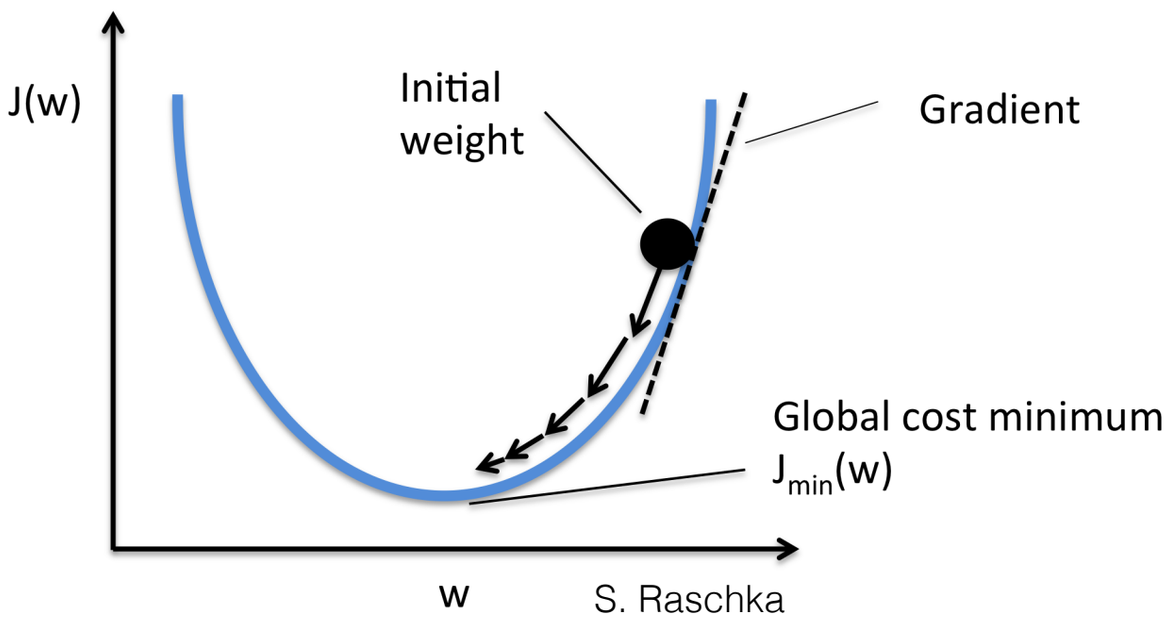
Error Function

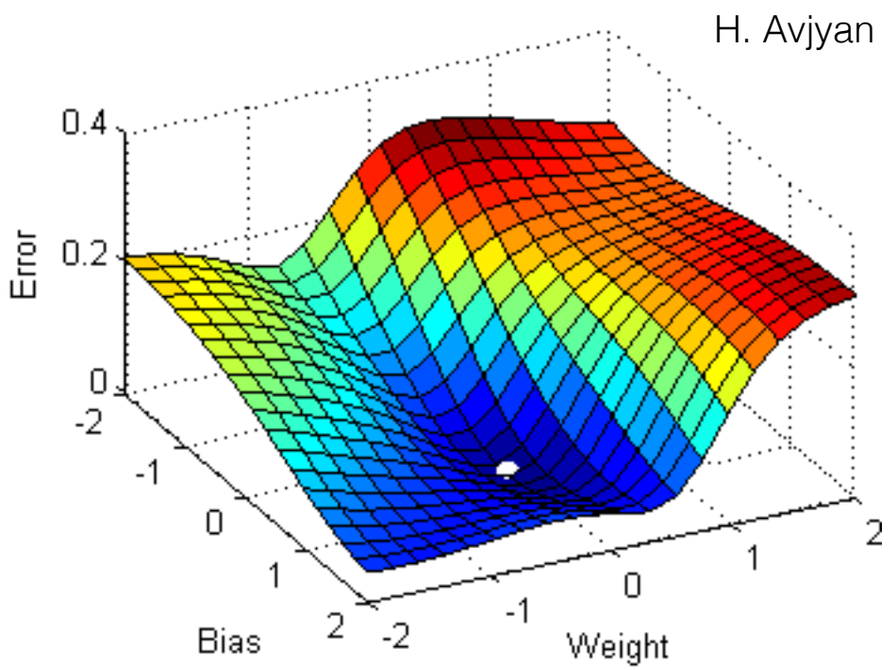
- How do we adjust the weights?
- We need to quantify the difference between the perceptron output and the true value y for an input **x**, over a set of input-output pairs.
- The mean square error (MSE) is a natural choice:

$$E(X) = \frac{1}{2N}\sum_{i=1}^{N}(o_i - y_i)^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(g(\vec{w}\cdot\overrightarrow{x_i} + b) - y_i\right)^2$$

- Note that E(X) = 0 when $o_i = y_i$ for all input-output pairs.
- Therefore, we want to MINIMIZE E(X)
- How do we do that?

Gradient Descent





- Start with random bias and weights
- Use gradient of error function w.r.t. the bias and weights at each point to update those values.
- Iterate until certain convergence criterium is met.

$$\overrightarrow{w_{i+1}} = \overrightarrow{w_i} - \alpha \frac{\partial E(X)}{\partial \overrightarrow{w_i}}$$

$$b_{i+1} = b_i - \alpha \frac{\partial E(X)}{\partial b_i}$$

Learning rate!

Parenthesis: Stochastic GD

- Remember that the error function is a sum over all examples in the training set:

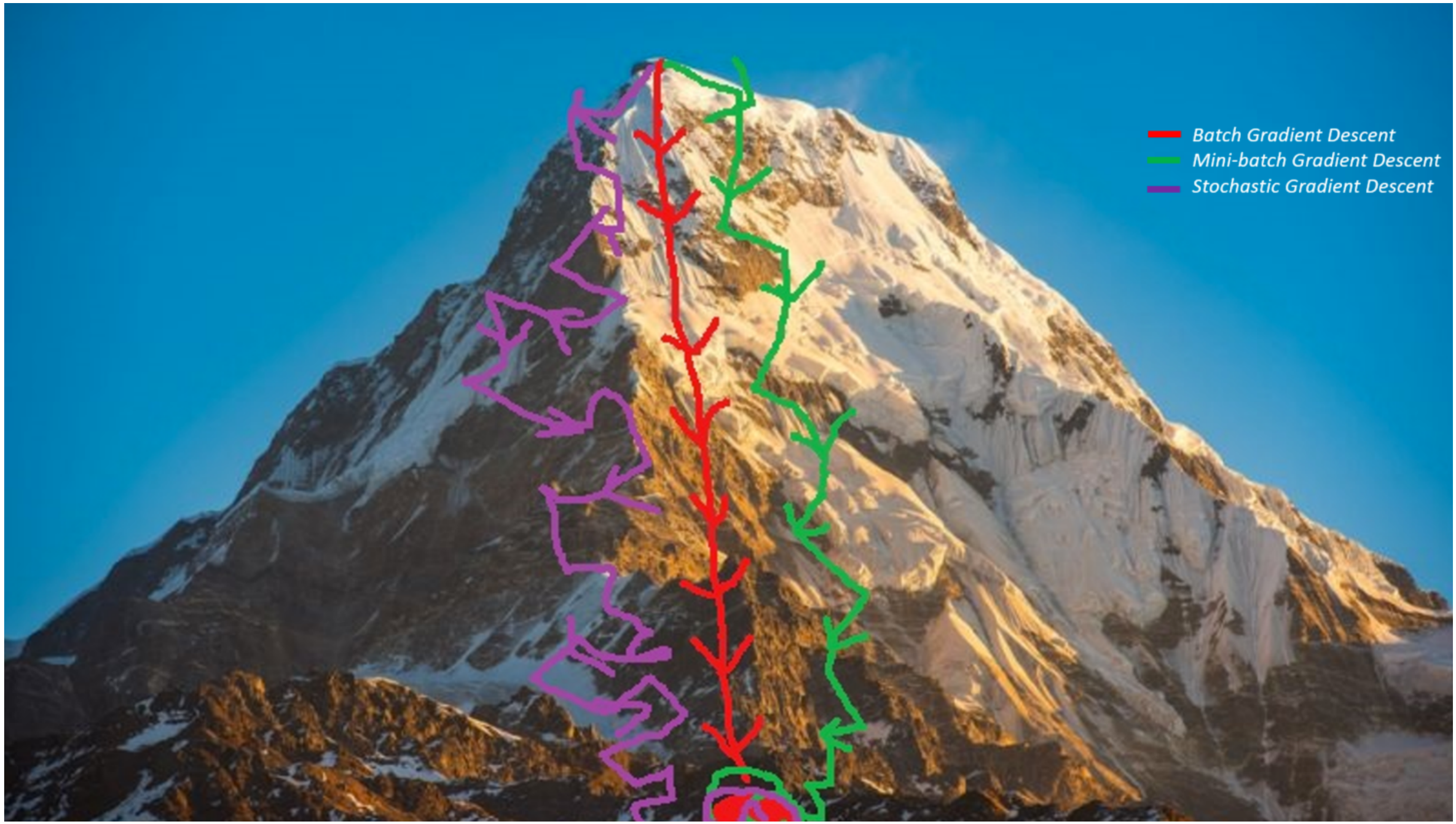
$$E(X) = \frac{1}{2N}\sum_{i=1}^{N}(o_i - y_i)^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(g(\vec{w}\cdot\overrightarrow{x_i} + b) - y_i\right)^2$$

- For a single update of weights, I need to go over the entire data set. Not efficient.

- Solution 1: approximate the gradient using a single random example. Not too accurate.

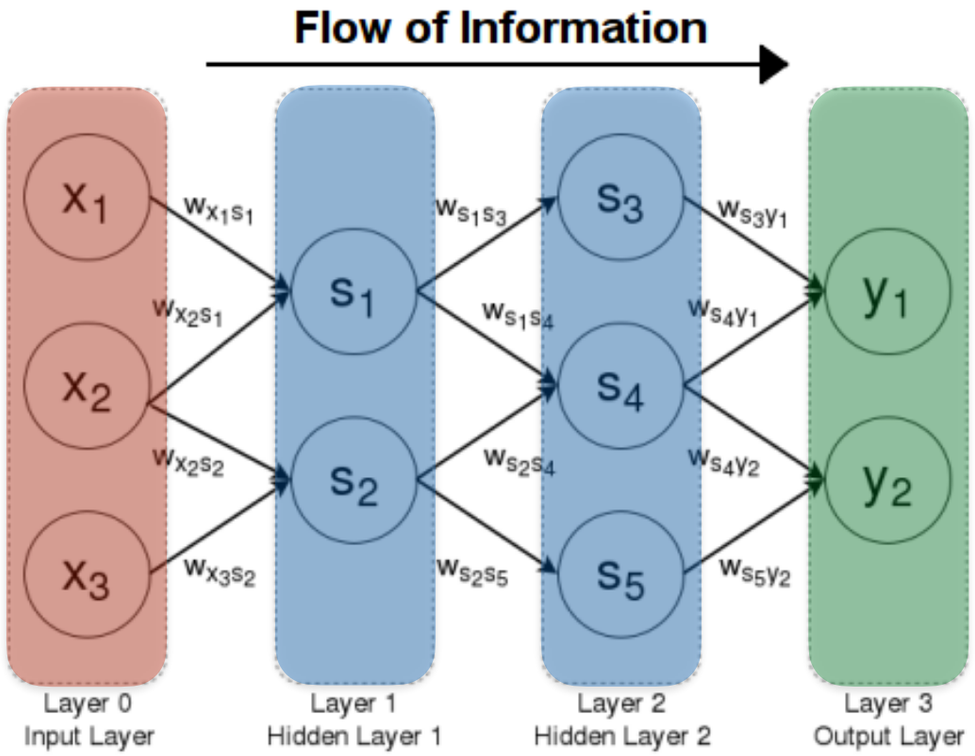
- Solution 2: Perform an update using a mini-batch of examples.

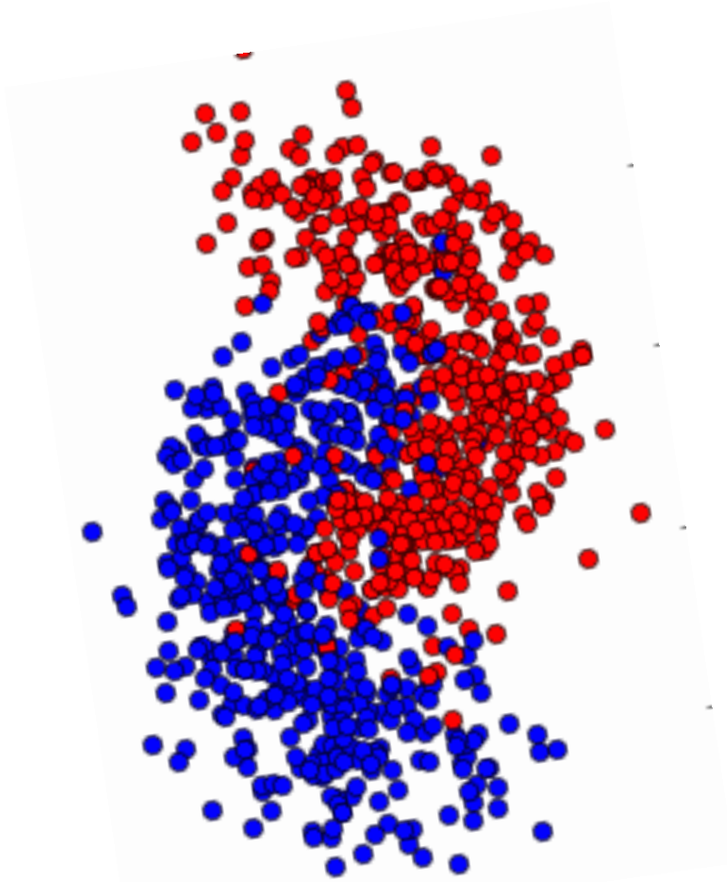
Parenthesis: Stochastic GD



I. Dabbura

Multi-layer perceptron





- ANN composed on many perceptrons.
- Capable of learning non-linearly separable functions.
- Good for regression and classification in supervised learning.
- The MLP is organized in layers: an input layer, some hidden layers, and an output layer.
- Regression: single neuron in the output layer
- Classification: more than one neuron in output layer
- The MLP above is said to be FULLY CONNECTED (all neurons in a given layer are connected to all neurons in the next)

Convolutional Neural Networks

Finding structure in images

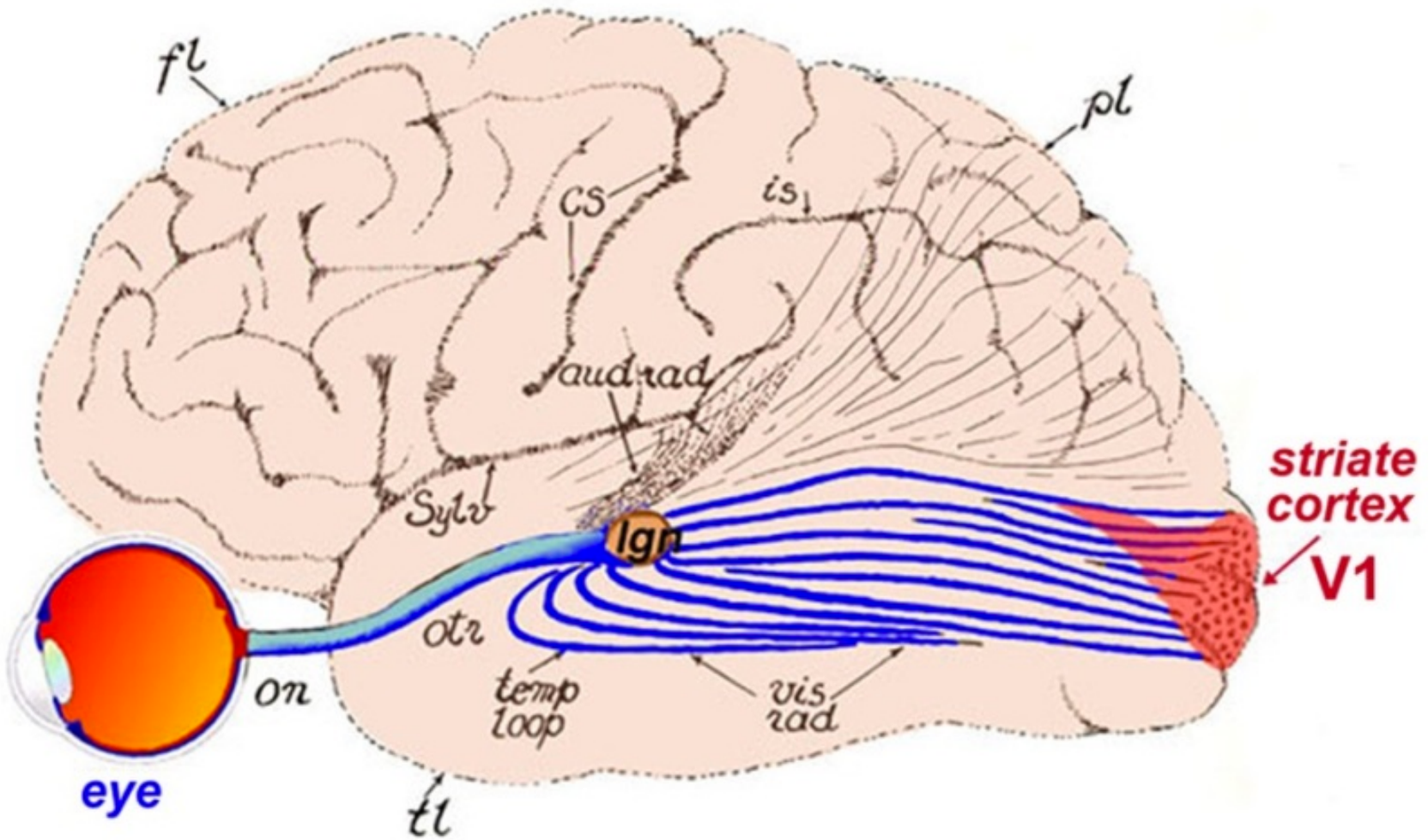
Nearby pixels are more strongly related than distant ones.

Objects are built up out of smaller parts.

A fully connected network would be very inefficient at learning patterns here.

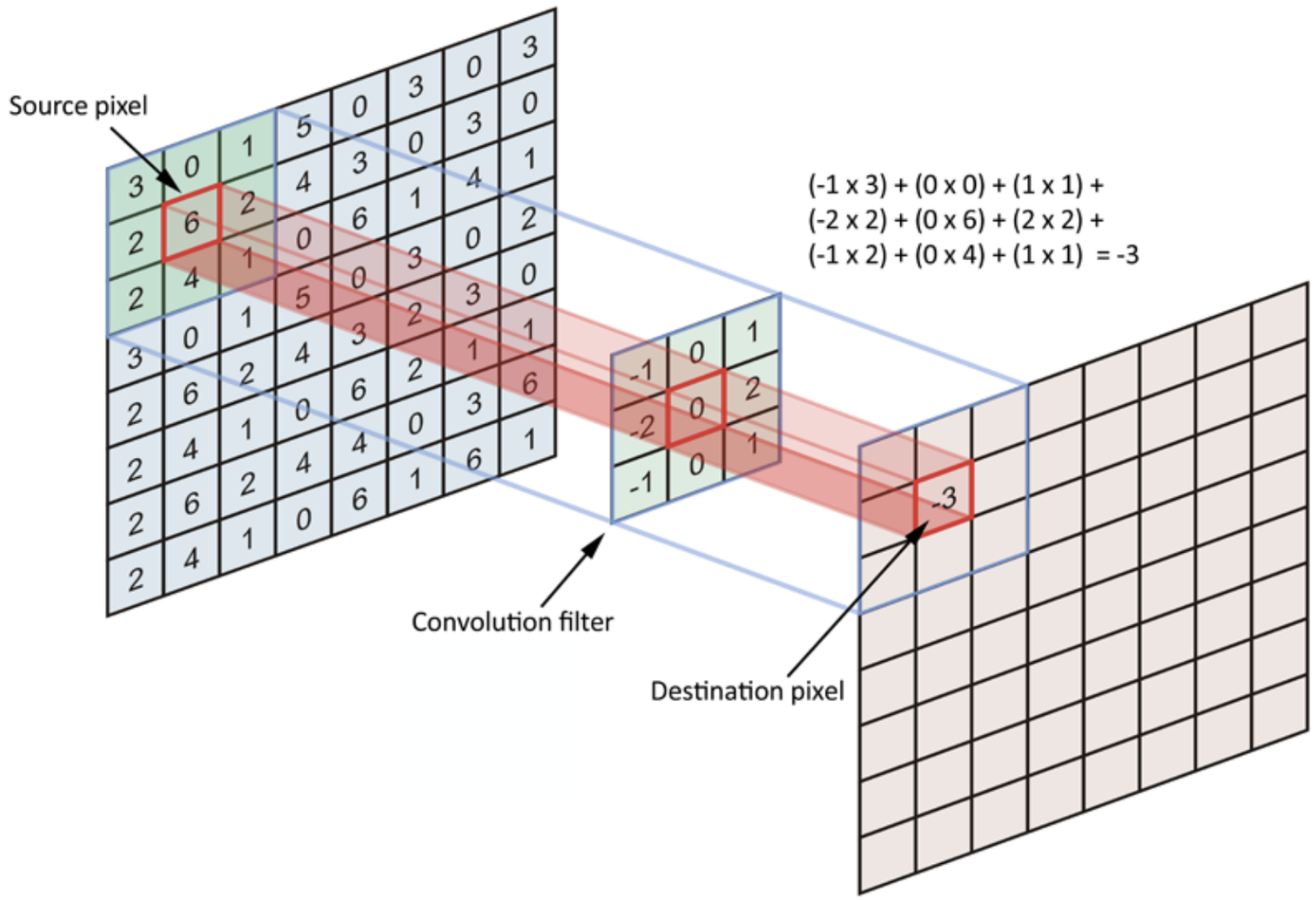
Instead, we can learn from nature.

Vision in nature

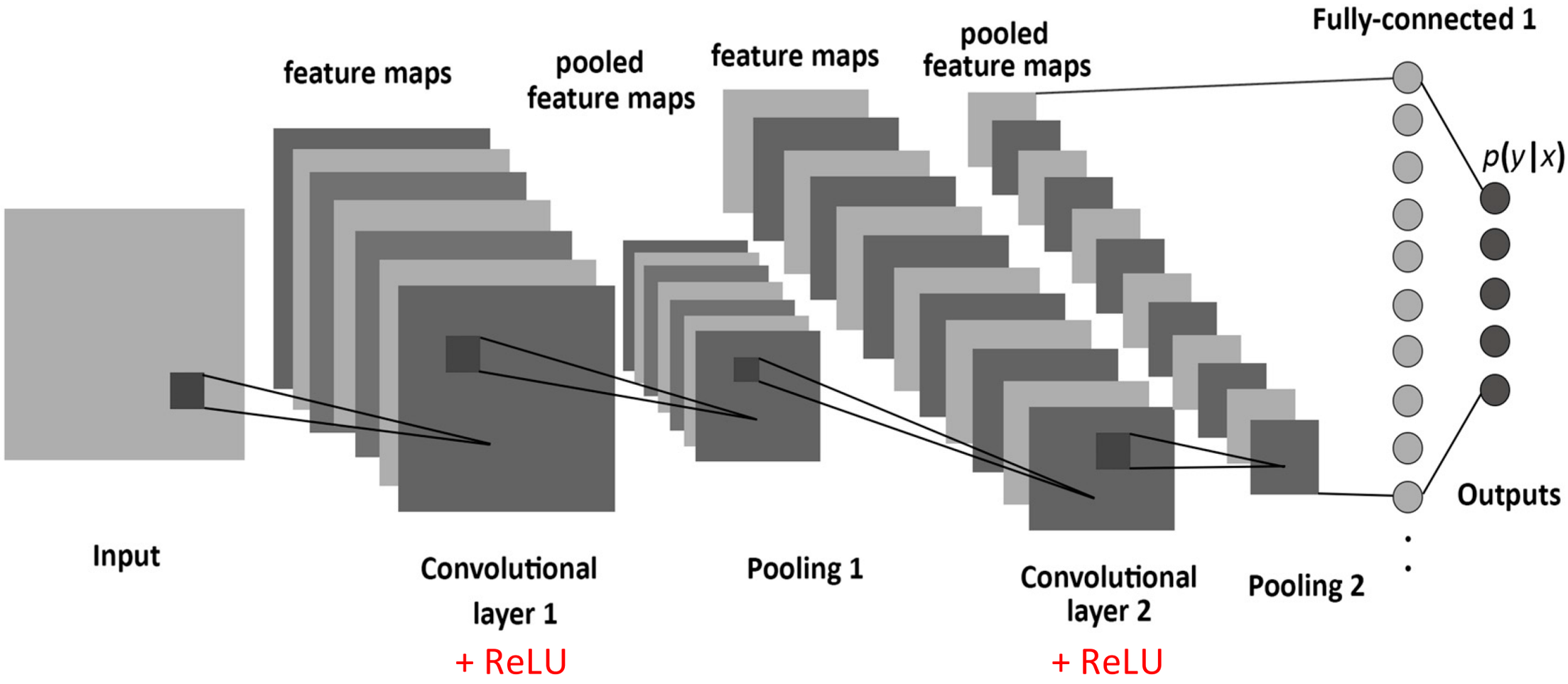


Cell in the visual cortex are not sensitive to the entire visual field
First neurons connected to the retina specialize in detecting edited
Neurons in further layers identify other types of patterns

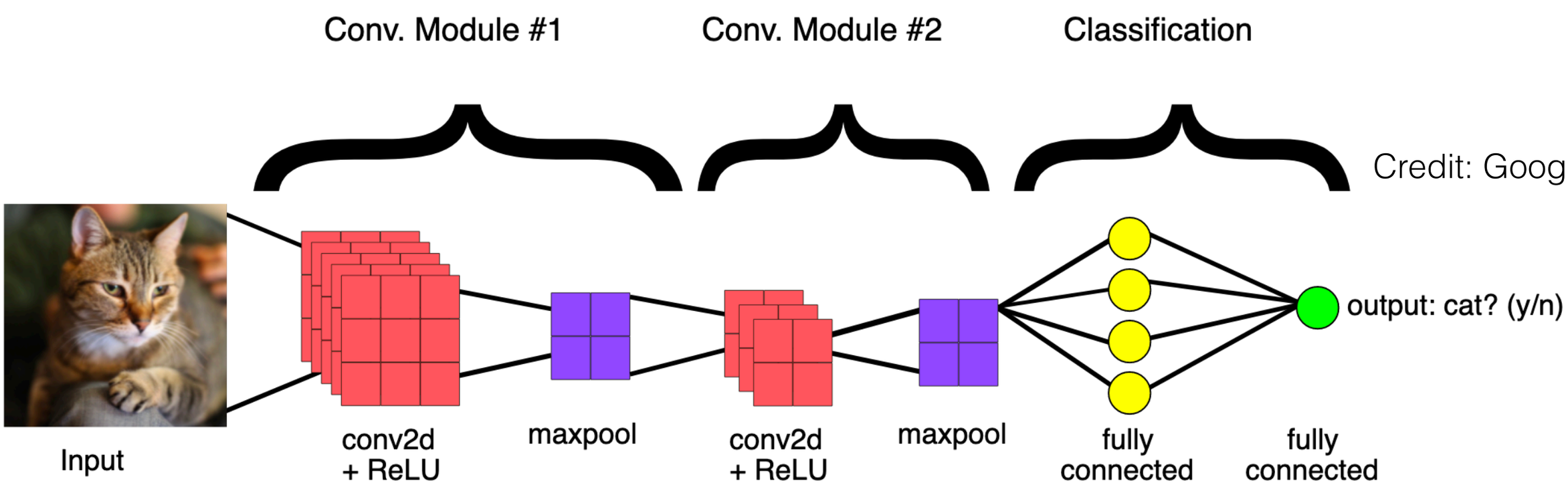
“Convolution” operation



A convolutional network



Fully connected NNs and CNNs work well for classification and regression



Credit: Google.

But most interesting things, i.e., language, weather, are not static, but sequential and time-dependent.

Recurrent Neural Networks

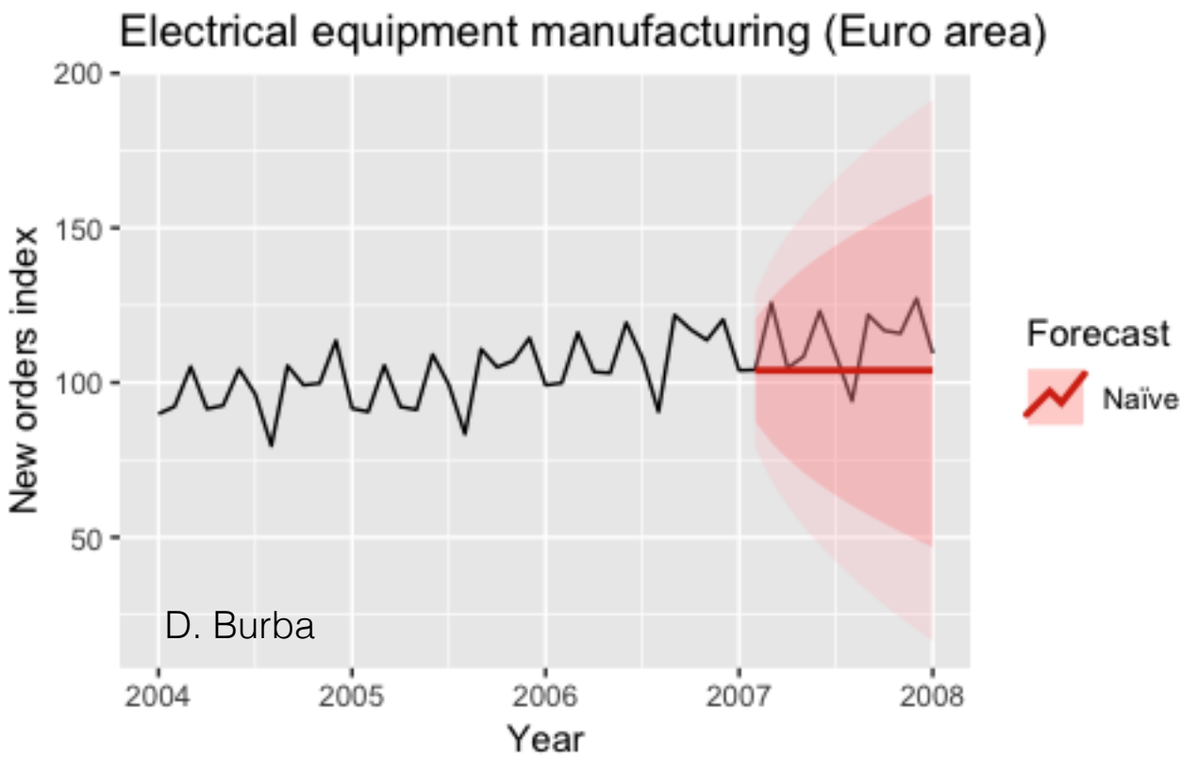
The importance of persistence

I grew up in France, I speak fluent… French

The importance of persistence

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

The power of prediction



- Forecasting is of extreme importance for finance, weather prediction, etc.
- In Machine Learning, only RNNs can do this efficiently
- Forecasting has value even in the case when it fails: it tends to fail more with truly anomalous time series

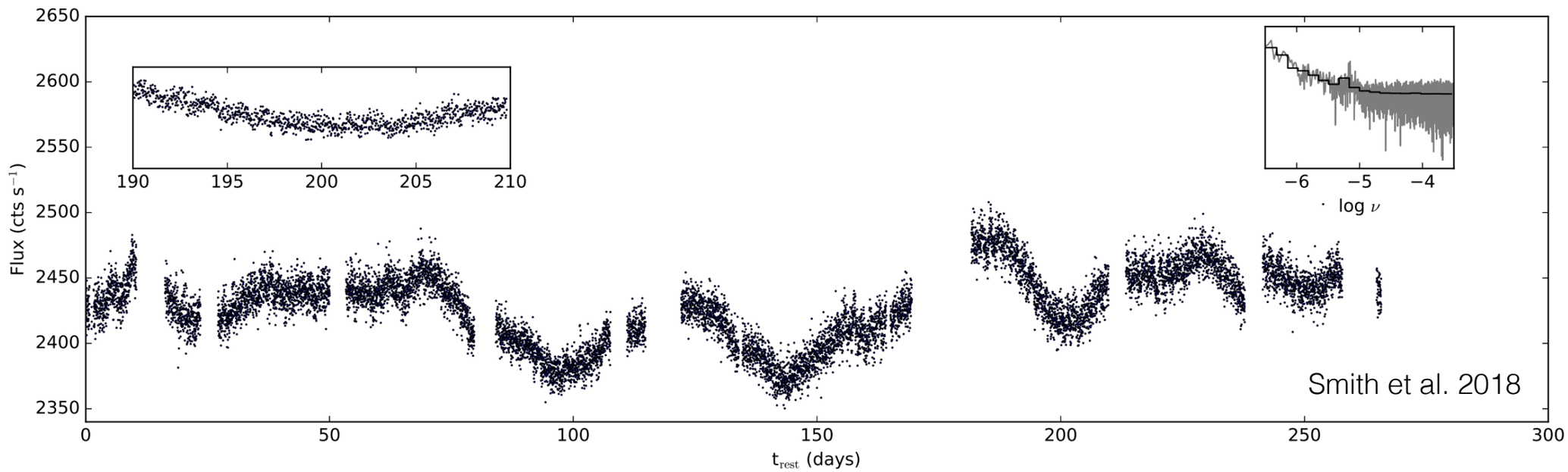
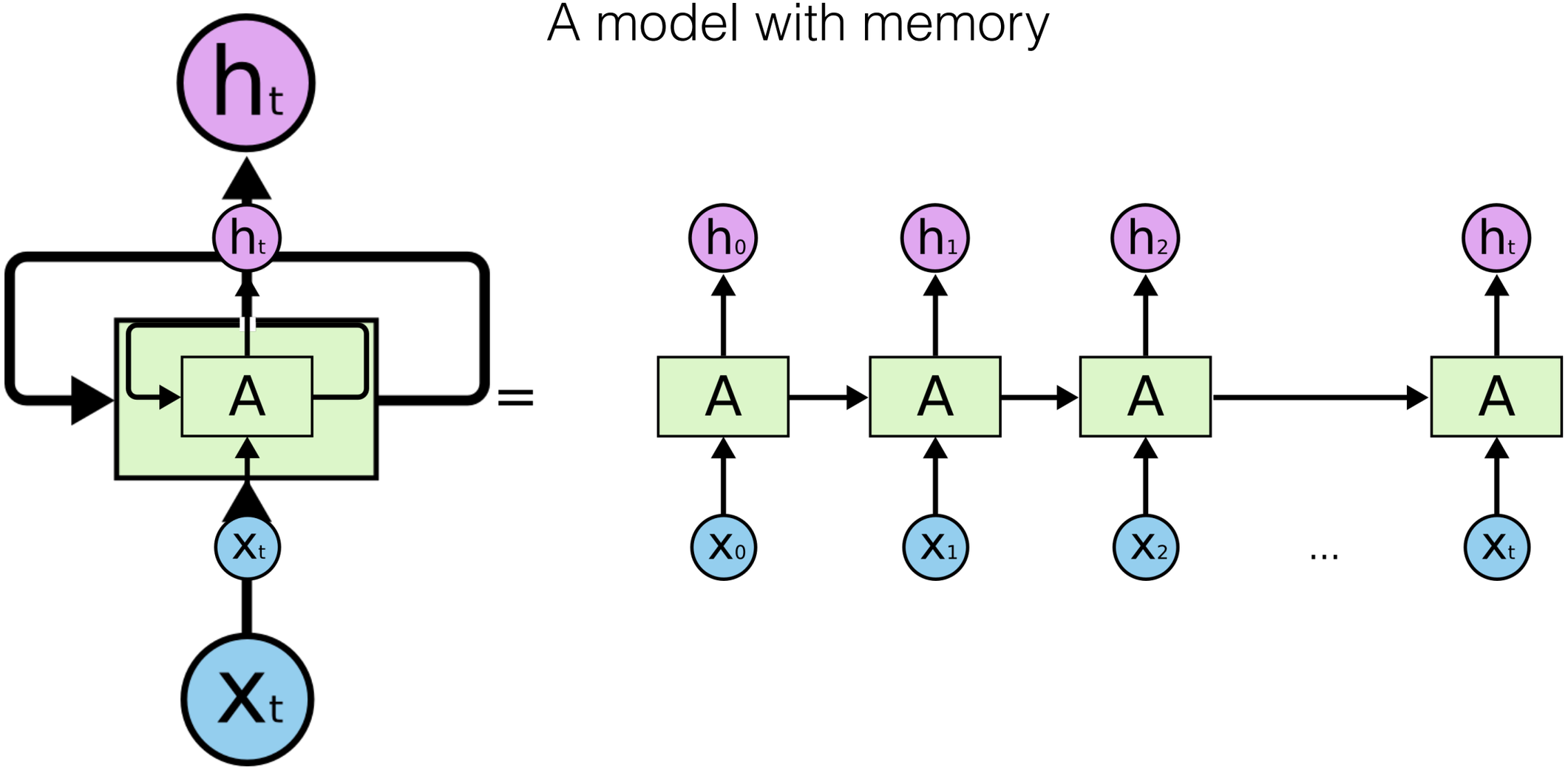


FIG. 1.— Light curve of spectroscopic AGN KIC 11614932, an object with stellar contamination within the extraction aperture. An excerpt of the light curve (left) and the power spectrum of the full light curve with the periodic signal visible (right) are shown as insets.

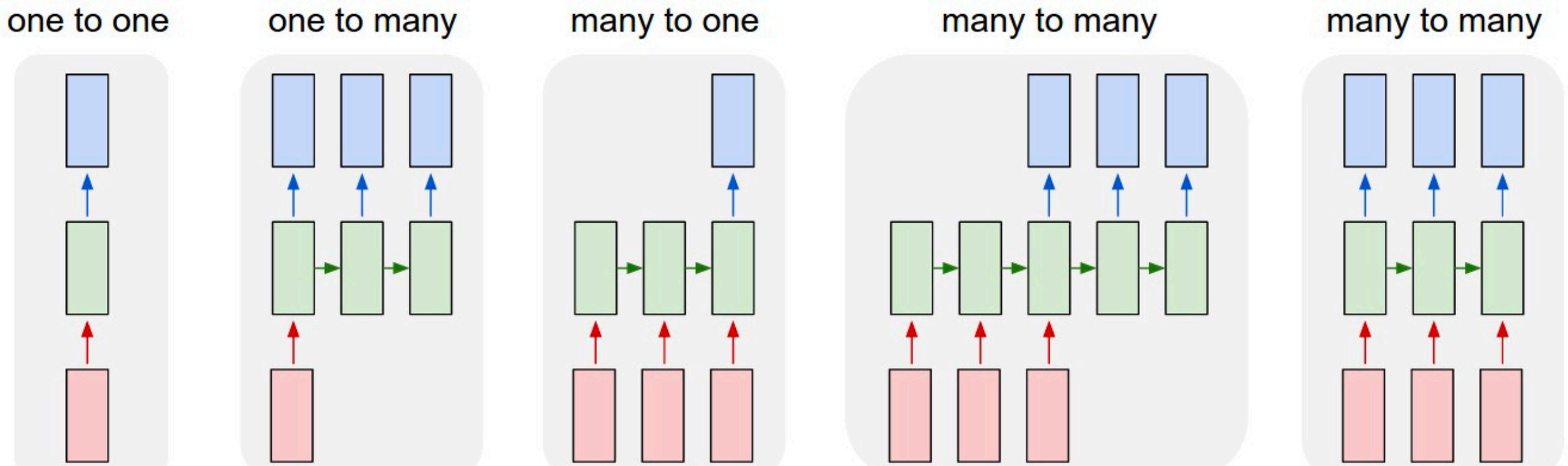
How do RNNs work?

A model with memory



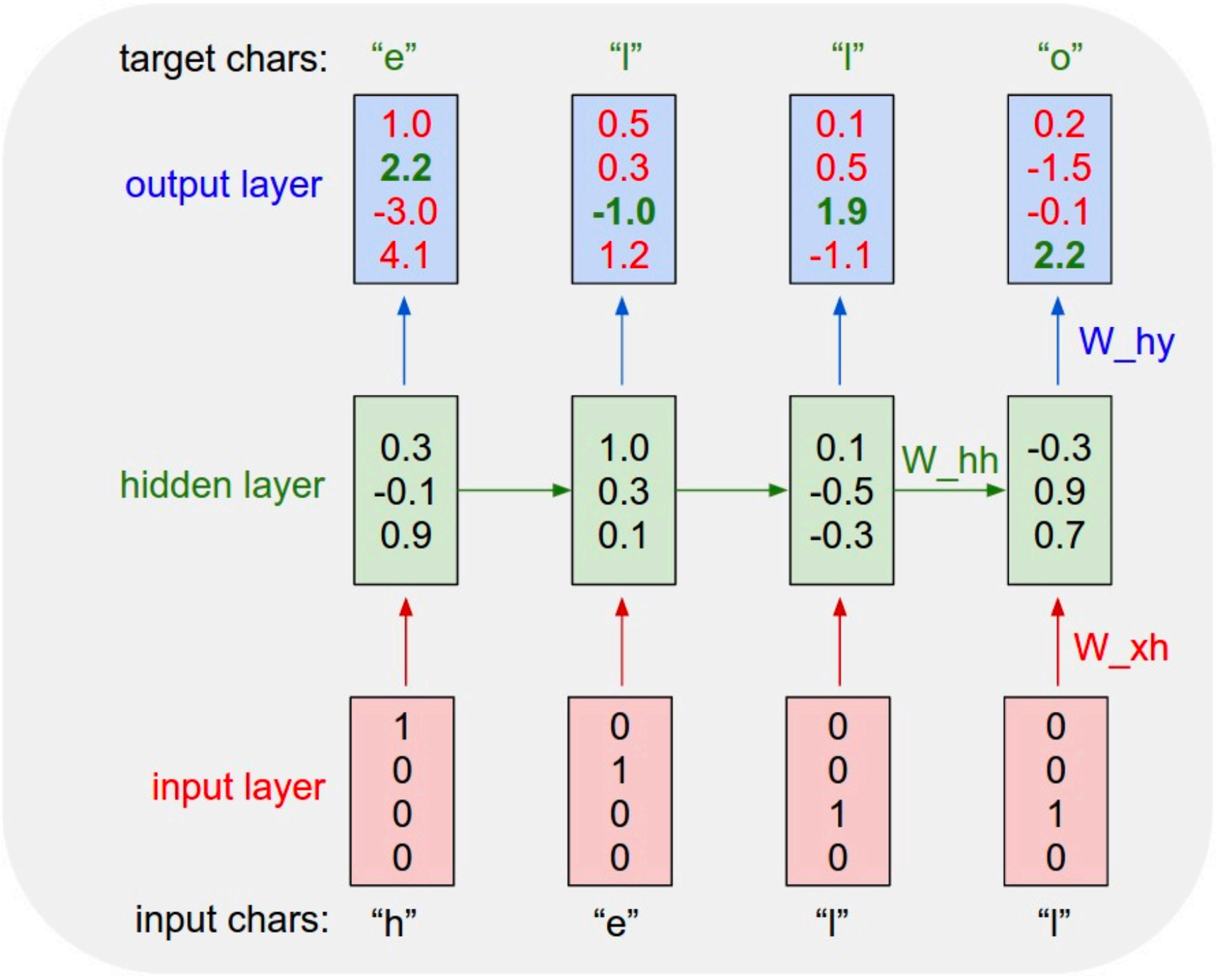
- The input is a sequence “x” (for example, over time). The output is also a sequence, h
- Hidden neural units “A” have an internal state that gets updated based on the input “x_t”, and on the **previous internal state.**
- The sequence of internal states make the “hidden” (unobservable) vector, h
- Therefore, the content of the output h depends not only on the current input vector x, but on **the entire history of previous inputs**.

RNNs operate on sequences of vectors!



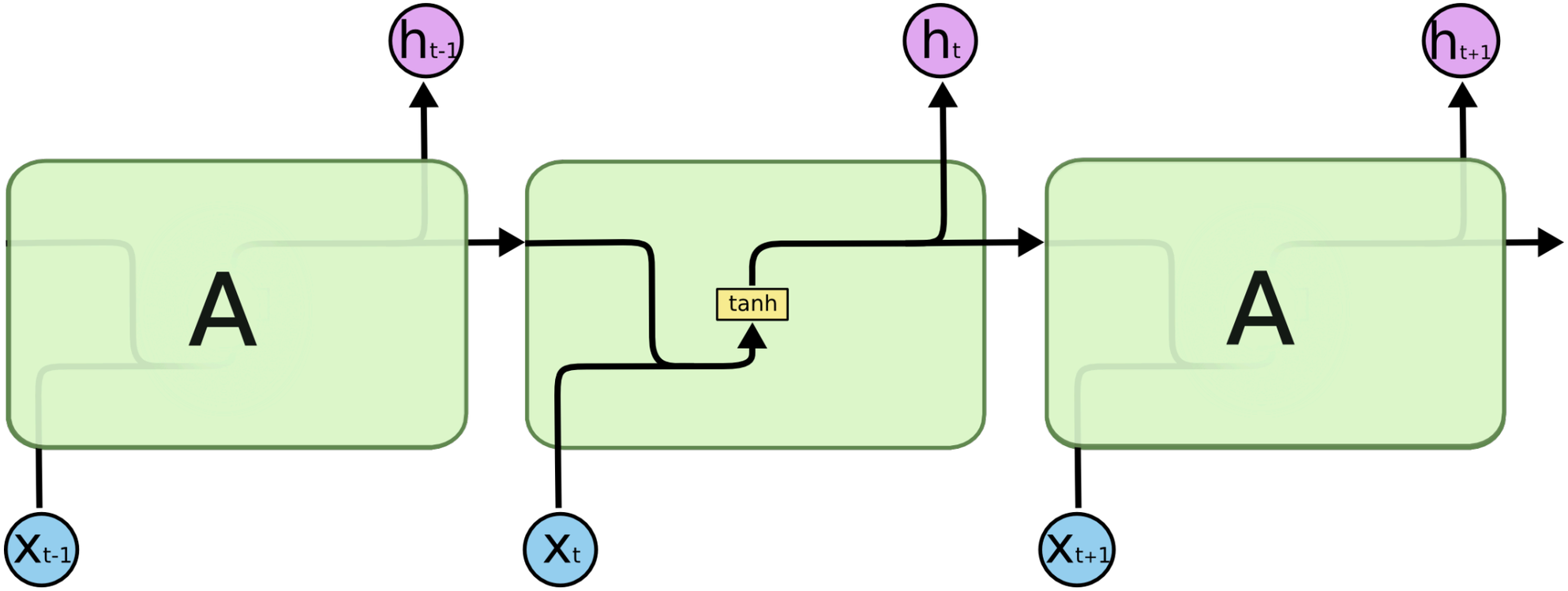
- One to one: standard forward fed NN (image classification)
- One to many: Input single vector, output sequence (image captioning)
- Many to one: Input sequence, output vector (sentiment analysis)
- Many to Many: both input and output are sequences: language translation, image sequence classification

Training RNNs



- The three matrices (W_xh, W_hh, and W_hy) are initialized randomly. Then weights are adjusted (using gradient descent) to obtain desired output.

The math behind a RNN

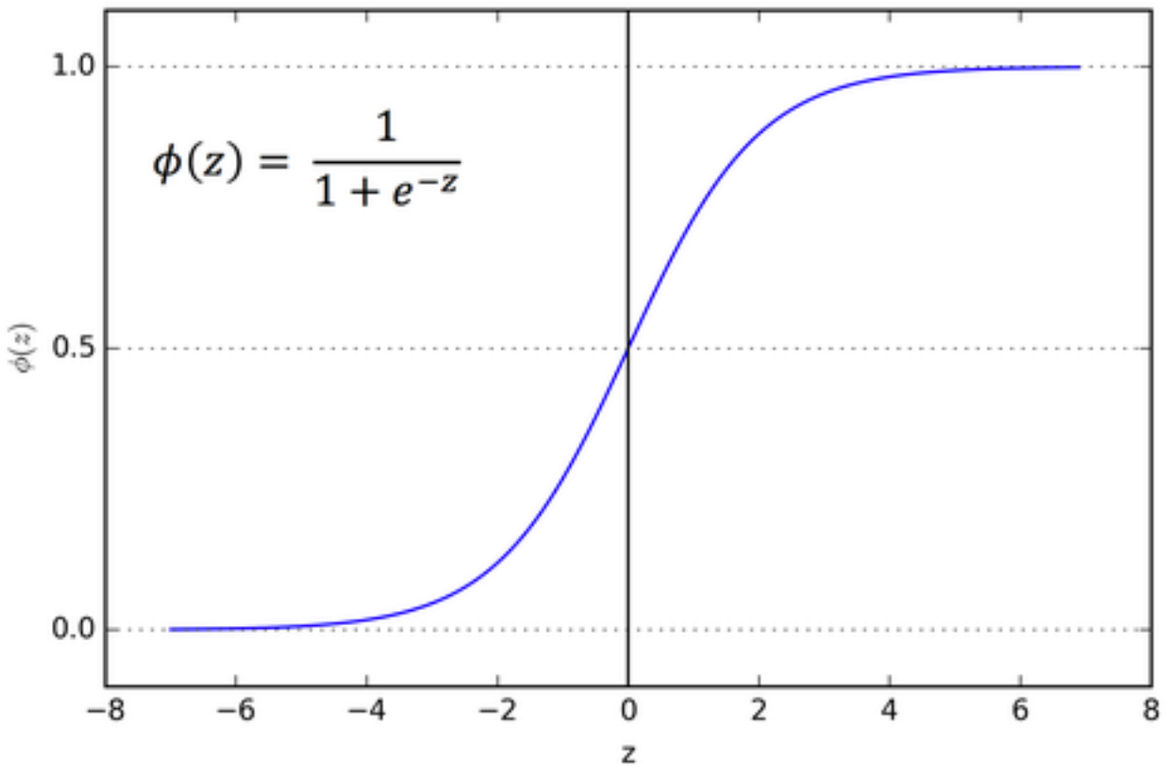


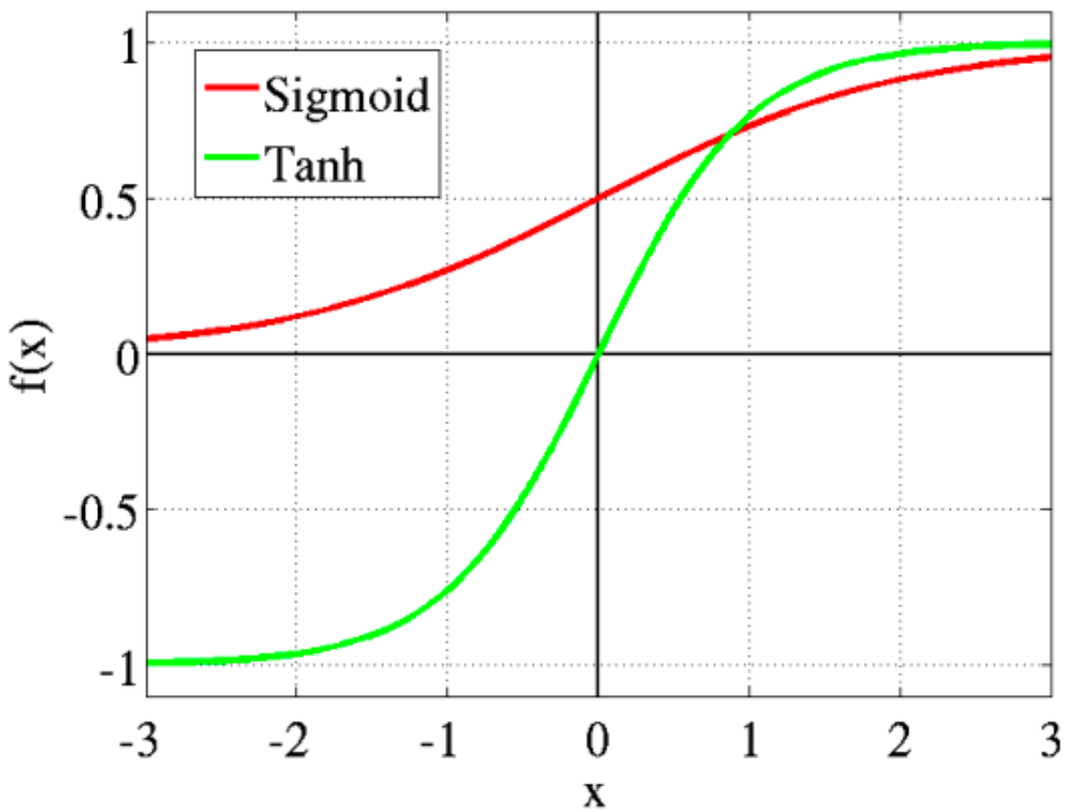
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Code:

```
class RNN:
  # ...
  def step(self, x):
    # update the hidden state
    self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
    # compute the output vector
    y = np.dot(self.W_hy, self.h)
    return y
```

Activation functions





Sigmoid:

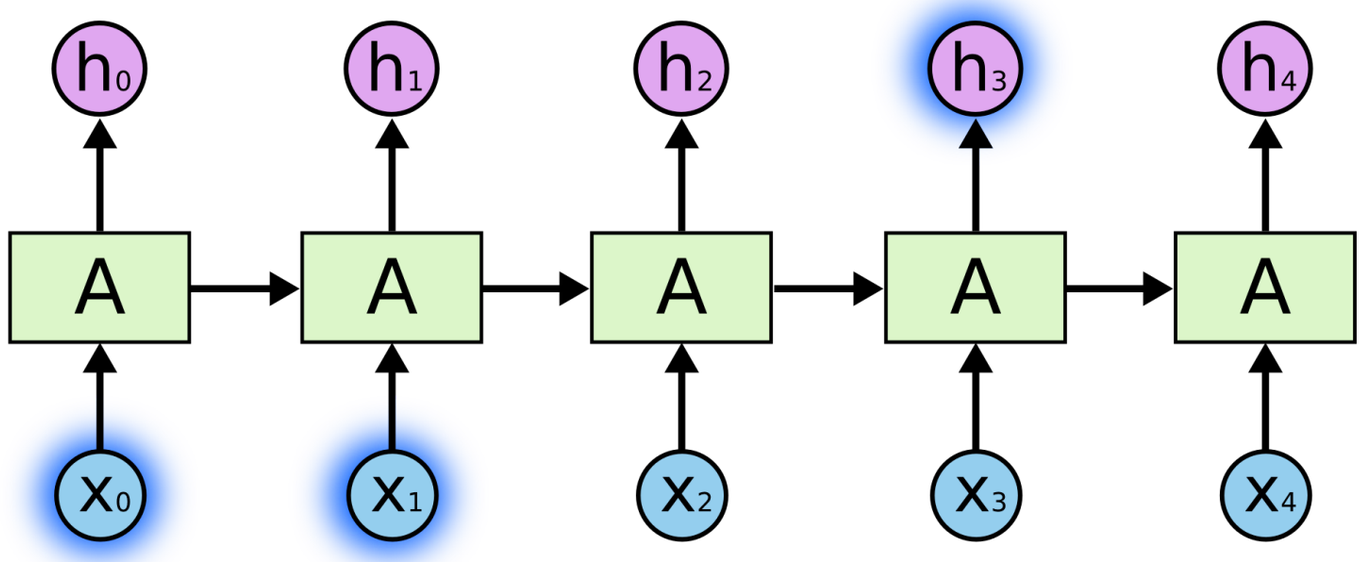
- Differentiable
- Squeezes input in range [0,1]
- Easily interpretable as probability (classification)

tanh:

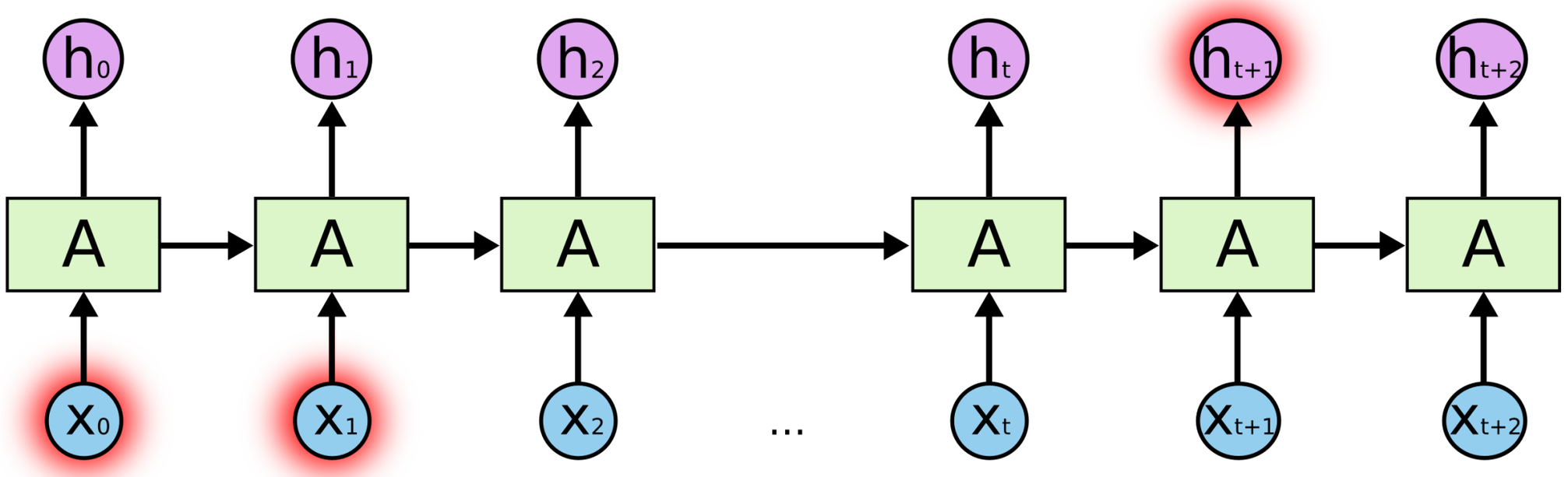
- Differentiable
- Squeezes input in range [-1,1]
- Negative and neutral values are taken into account

Short-term vs. long-term memory

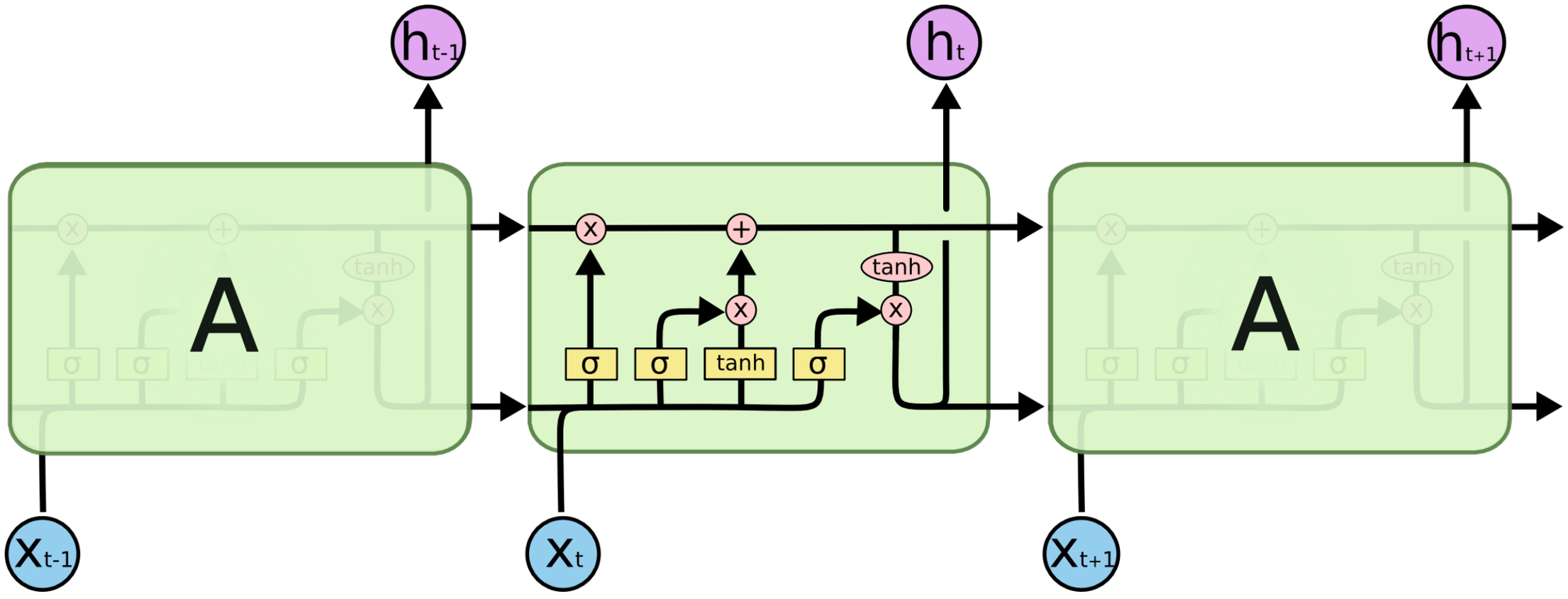
Las nubes están en el… cielo



Yo crecí en Colombia, hablo perfecto… español



Long Short Term Memory (LSTM) Networks



- The concept remains the same: output depends in previous hidden state, and on current input
- However, a new path is aded (horizontal line at the top). This is the cell state.
- Using neural operations wit the cell state, we can forget old information, or add new one, according to the input we are seeing.

Some key features

- **Sparse interactions**: neurons in a given layer are sensitive only to a sub-field of nearby neurons in the previous layer.

- **Parameter sharing**: same kernel applied to different regions of the input image. Learn kernel only.

- **Pooling**: reduce spatial dimensions of input volume, by averaging over small square windows of the field.

- **Padding**: add additional layer to the border of the image to avoid loss of information.