

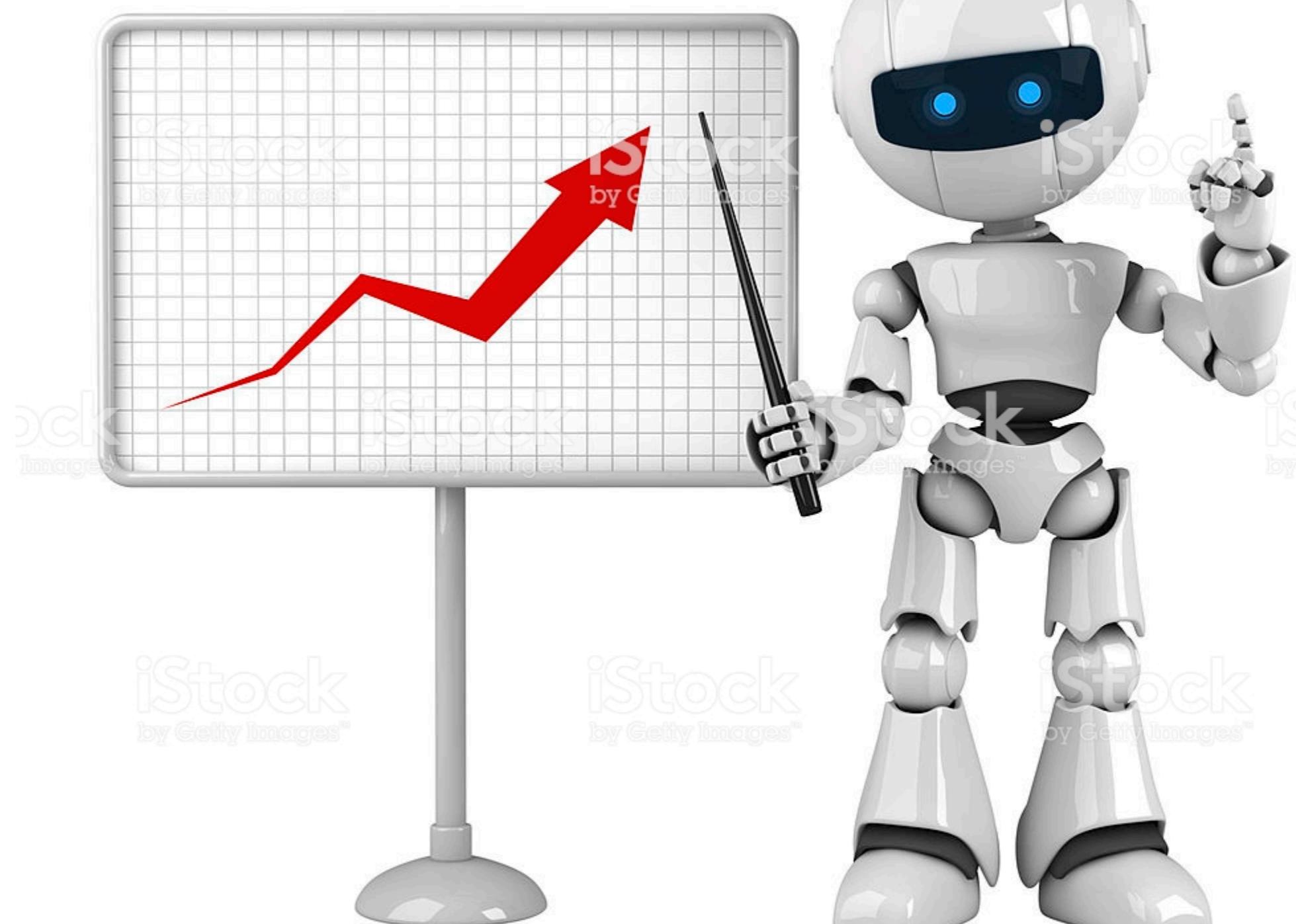


RNNs for Timeseries Analysis

Bruno Gonçalves

www.data4sci.com

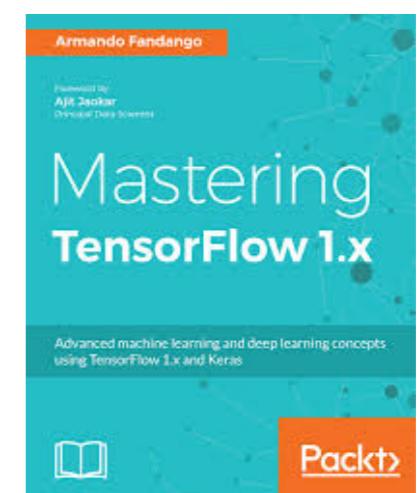
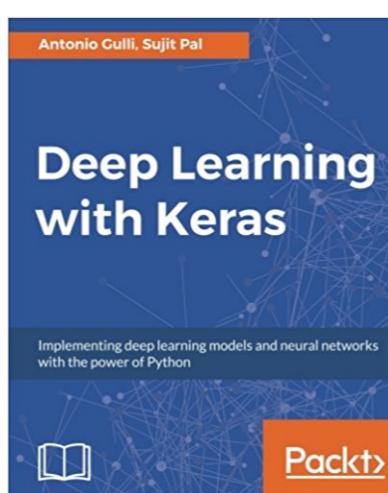
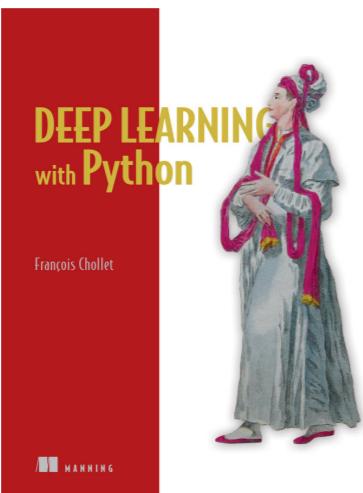
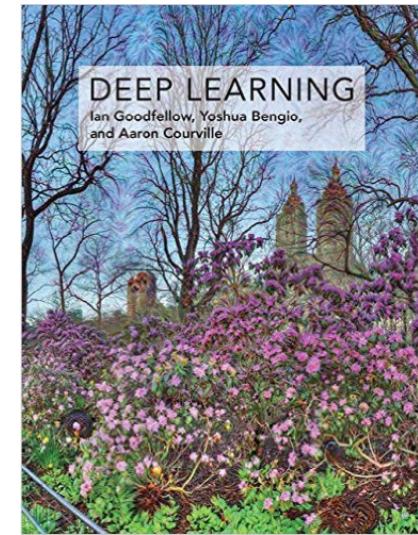
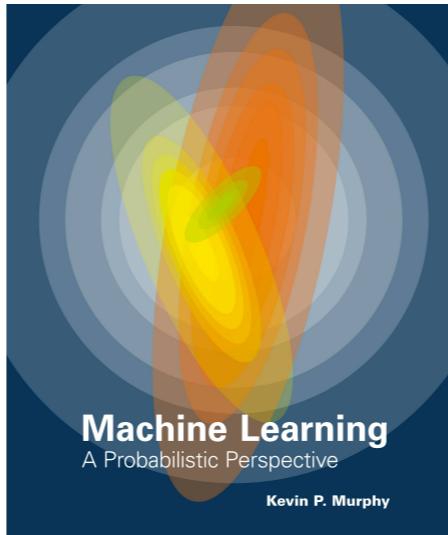
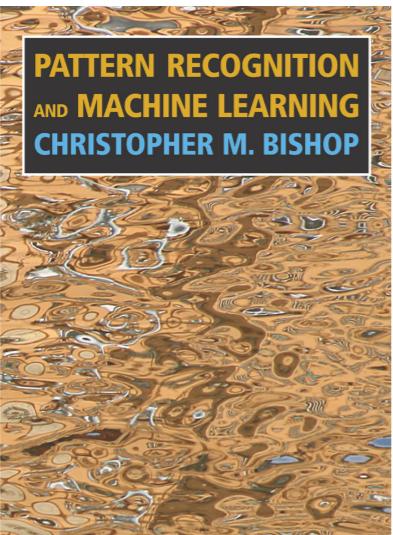
github.com/DataForScience/RNN



Disclaimer

The views and opinions expressed in this tutorial are those of the authors and do not necessarily reflect the official policy or position of my employer. The examples provided with this tutorial were chosen for their didactic value and are not meant to be representative of my day to day work.

References

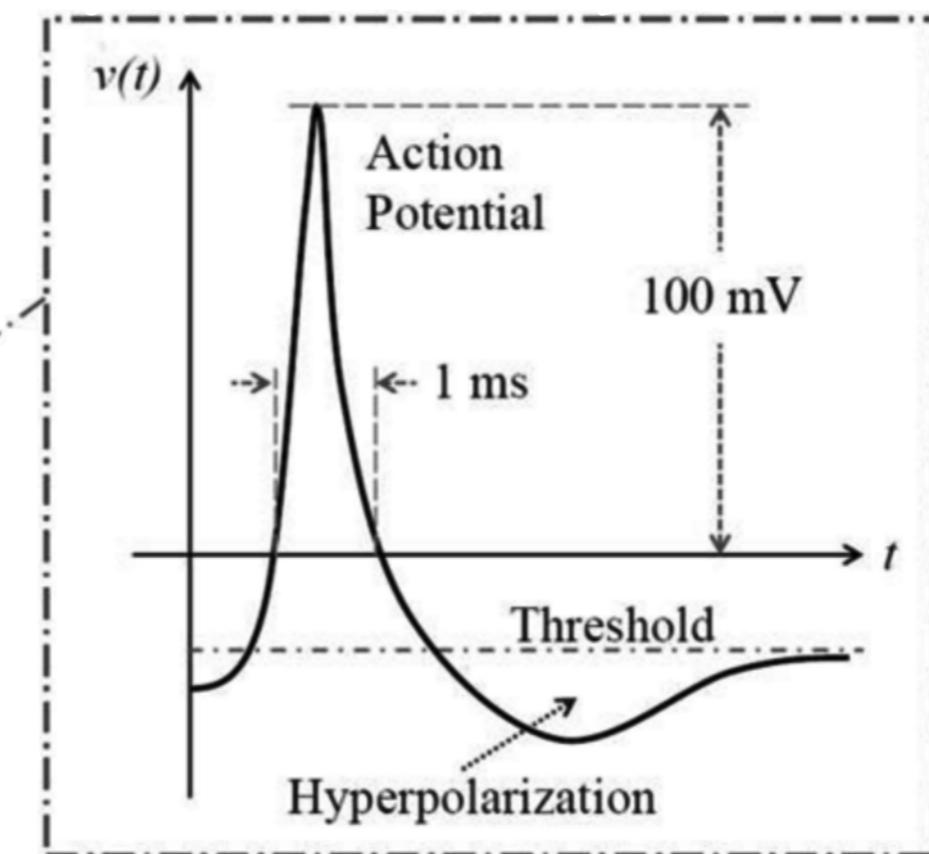
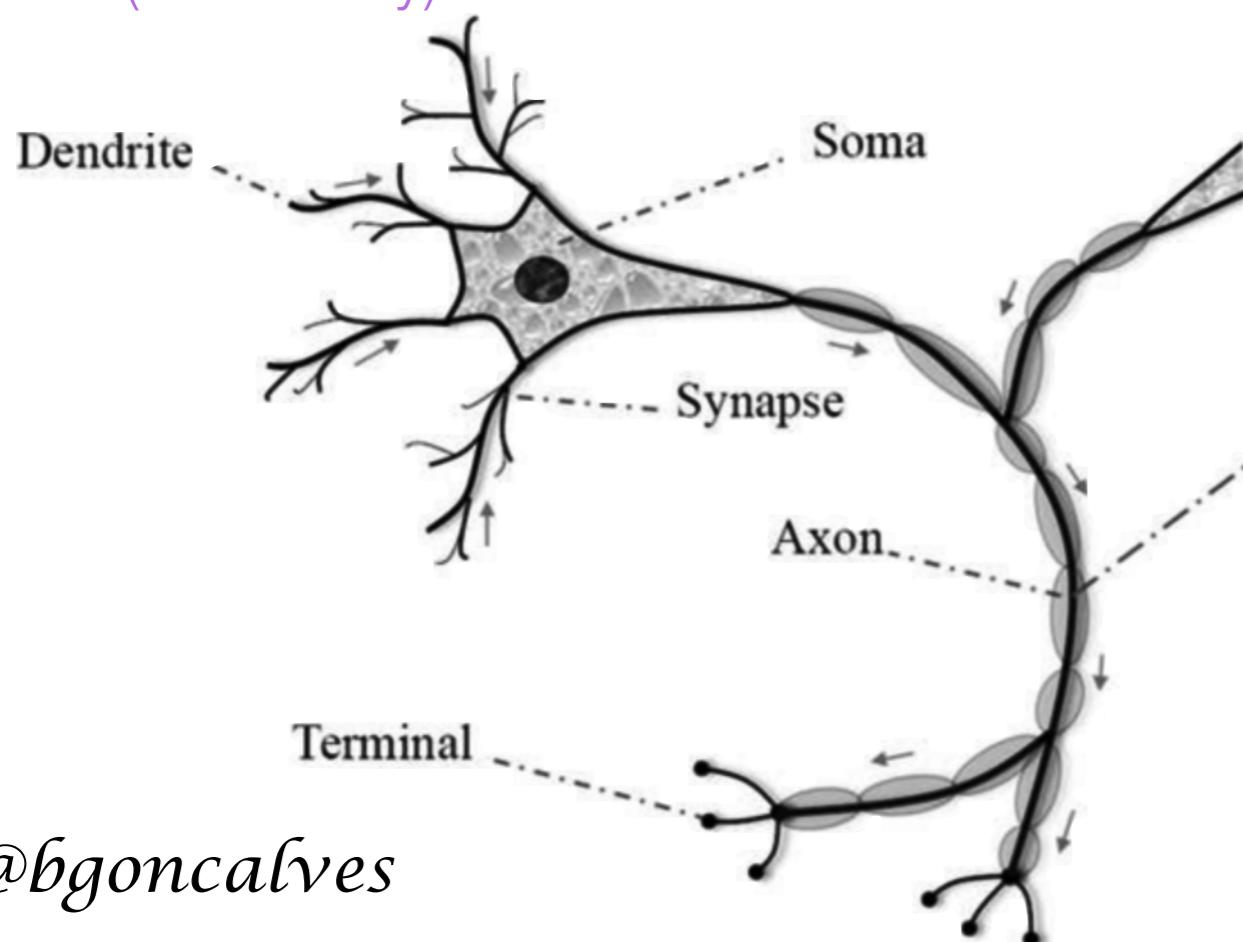
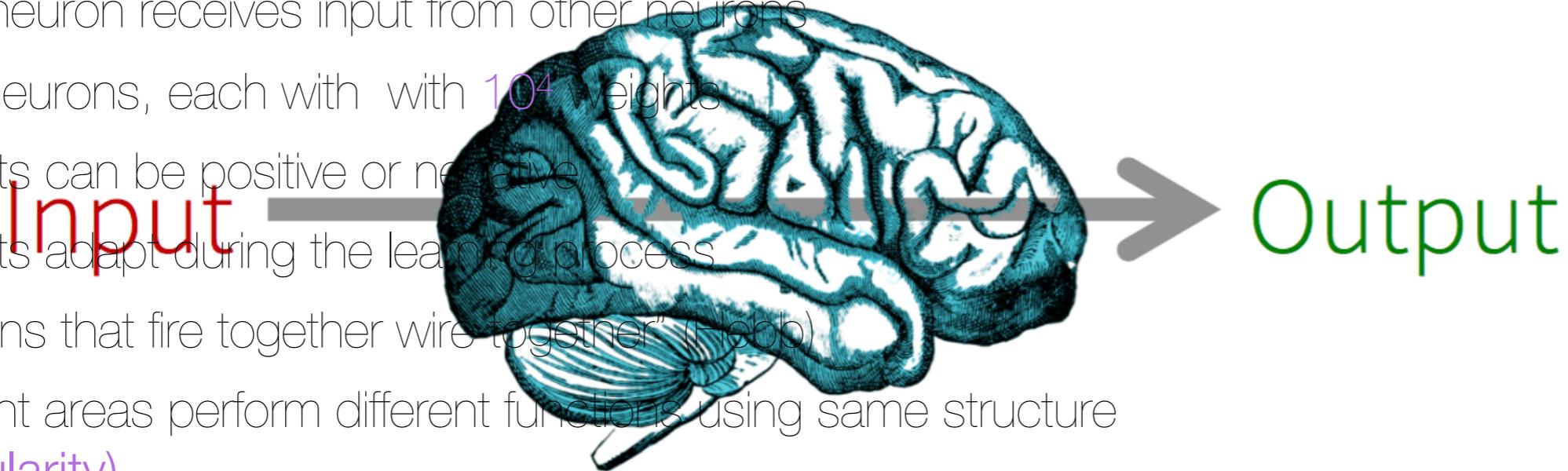


How the Brain “Works” (Cartoon version)

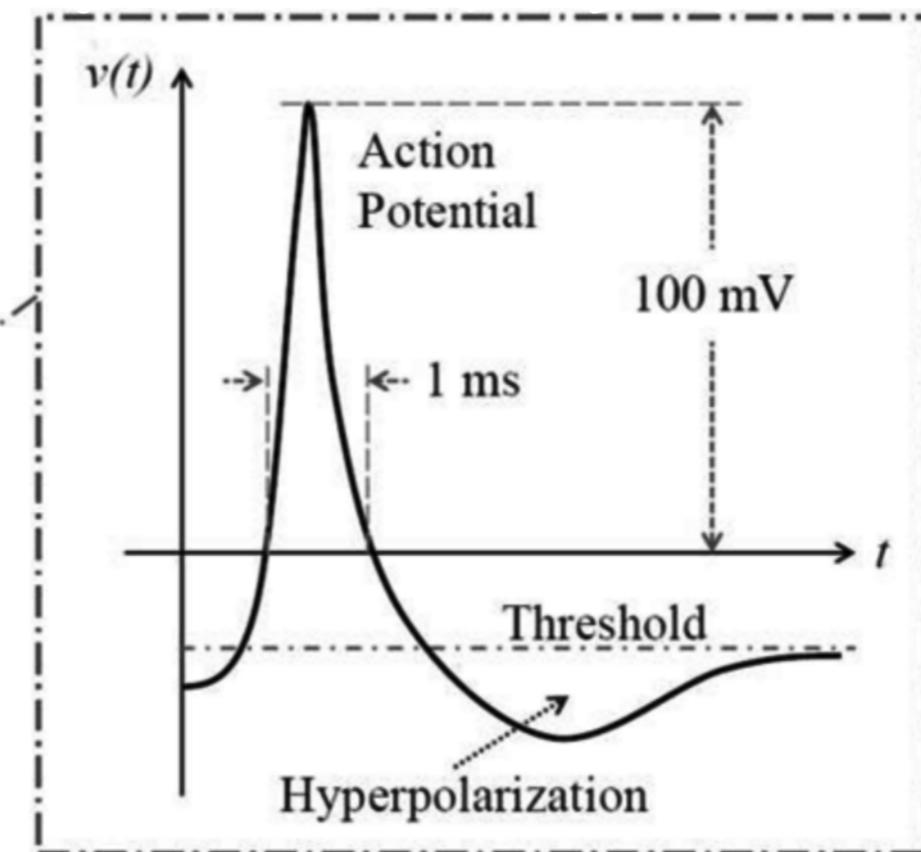
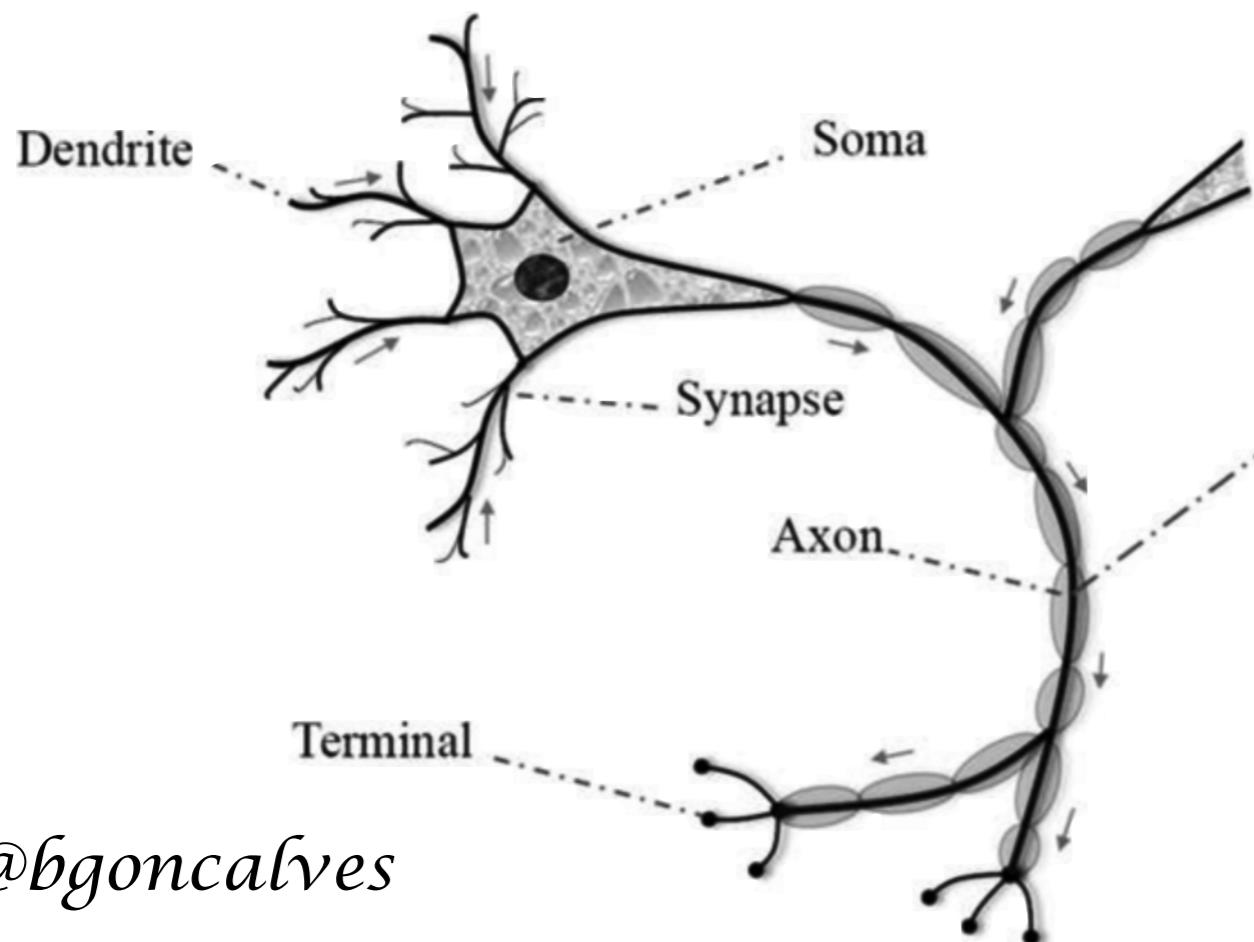
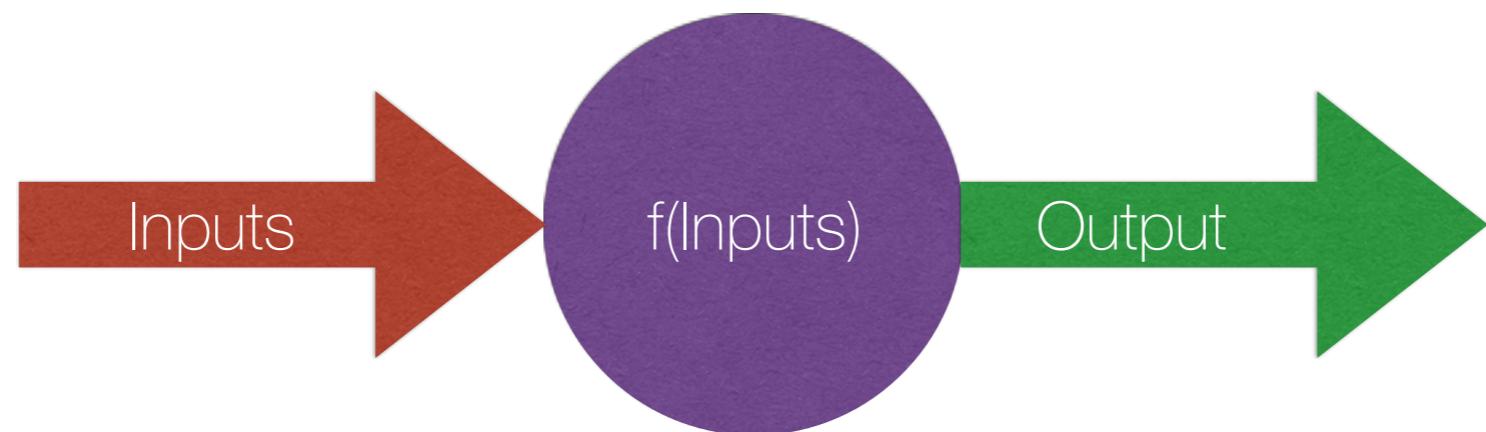


How the Brain “Works” (Cartoon version)

- Each neuron receives input from other neurons
- 10^{11} neurons, each with 10^4 weights
- Weights can be positive or negative
- Weights adapt during the learning process
- “neurons that fire together wire together” (Hebb)
- Different areas perform different functions using same structure
(Modularity)



How the Brain “Works” (Cartoon version)



Optimization Problem

- (Machine) Learning can be thought of as an optimization problem.
- Optimization Problems have 3 distinct pieces:
 - The constraints
 - The function to optimize
 - The optimization algorithm.

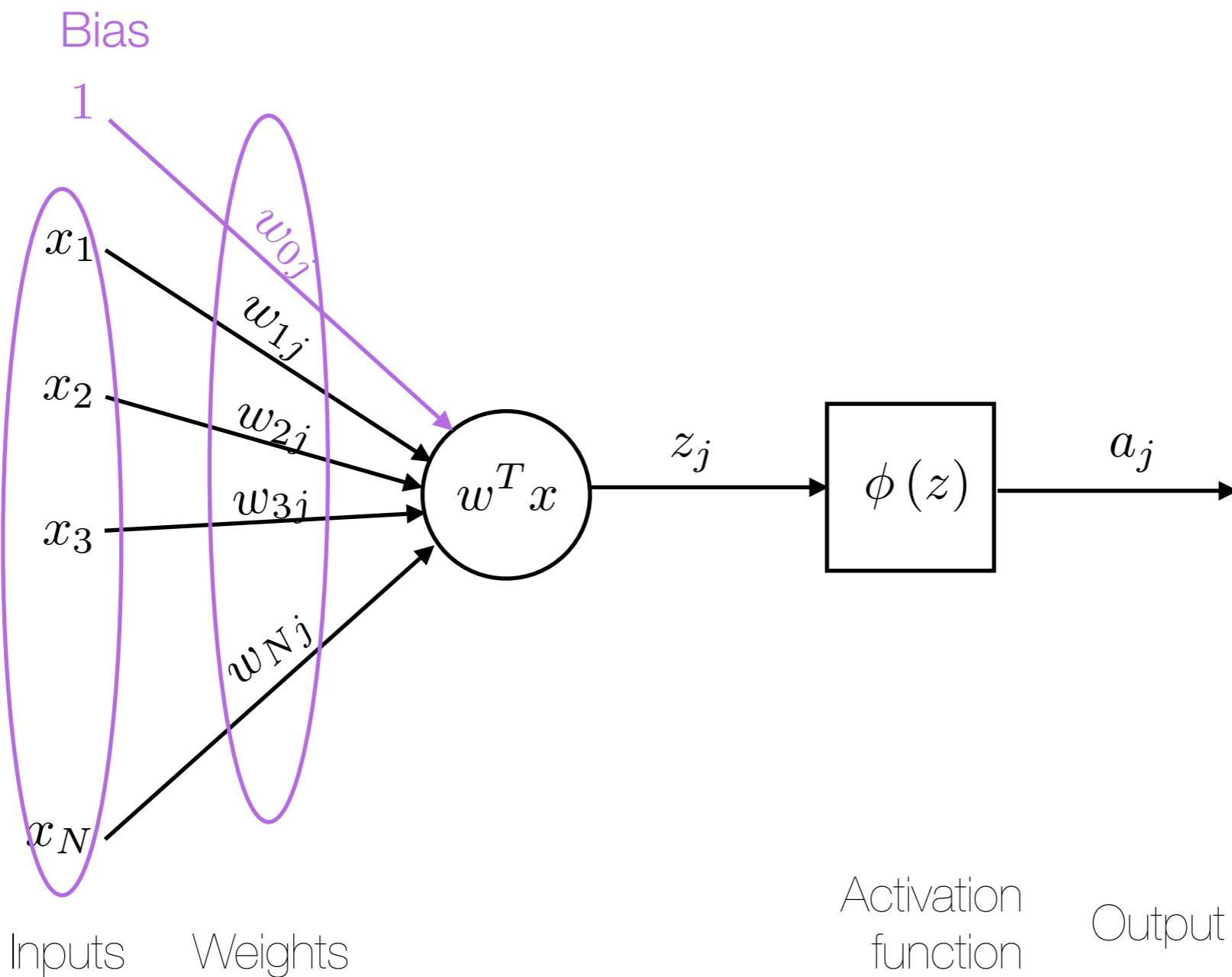
Neural Network

Prediction Error

Gradient Descent



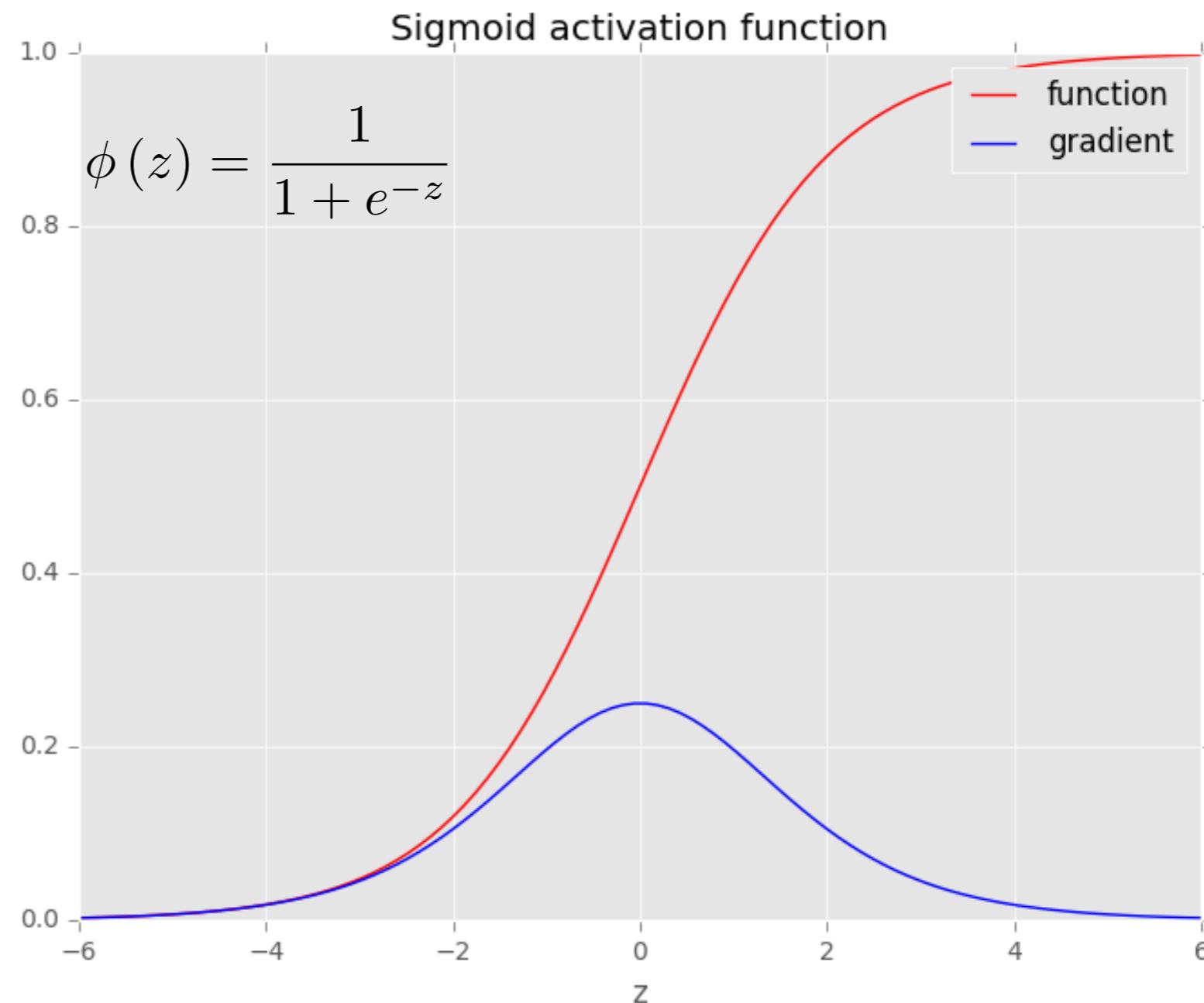
Artificial Neuron



Activation Function - Sigmoid

<http://github.com/bmtgoncalves/Neural-Networks>

- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- Perhaps the **most common**



Activation Function - tanh

<http://github.com/bmtgoncalves/Neural-Networks>

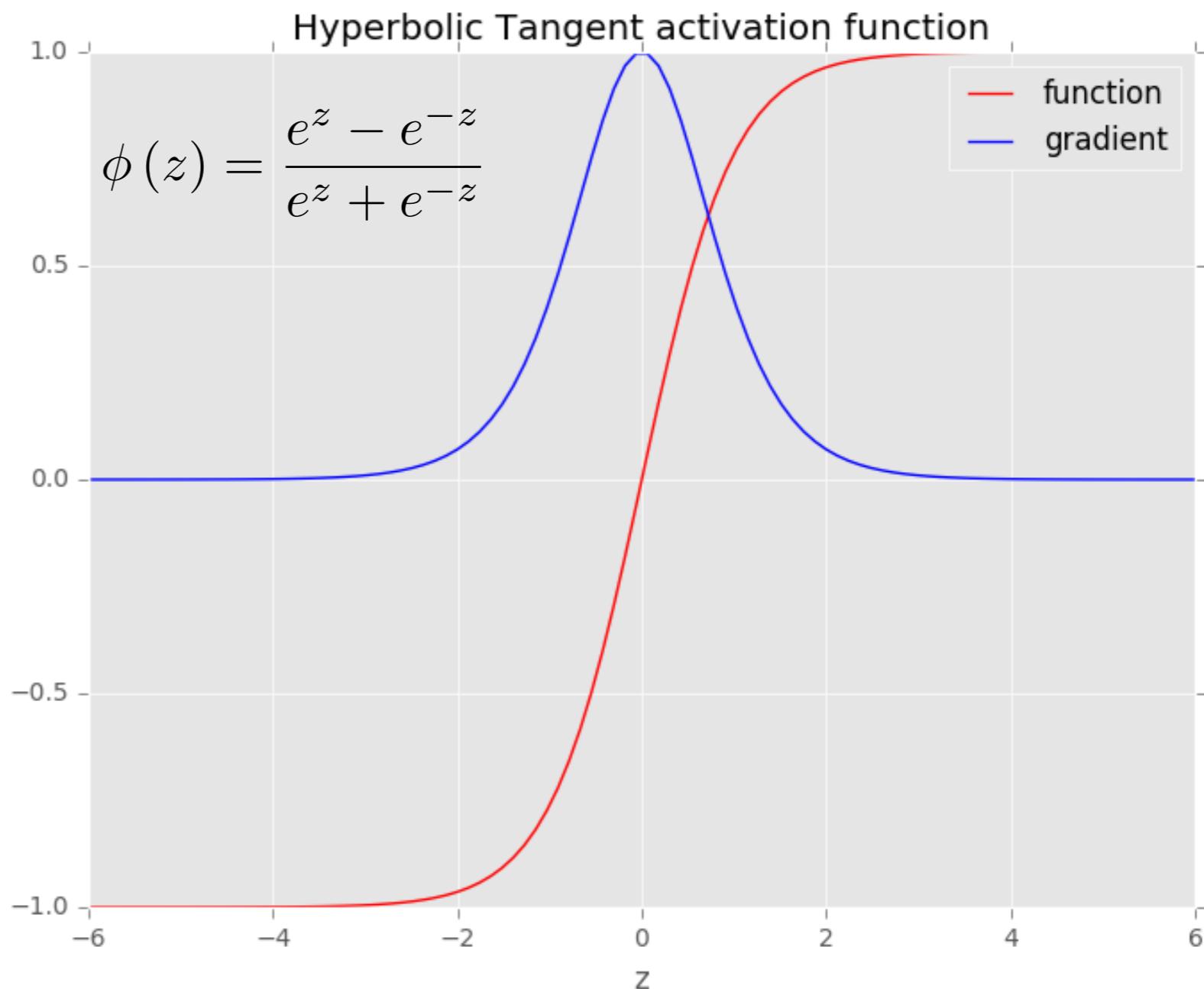
- Non-Linear function

- Differentiable

- non-decreasing

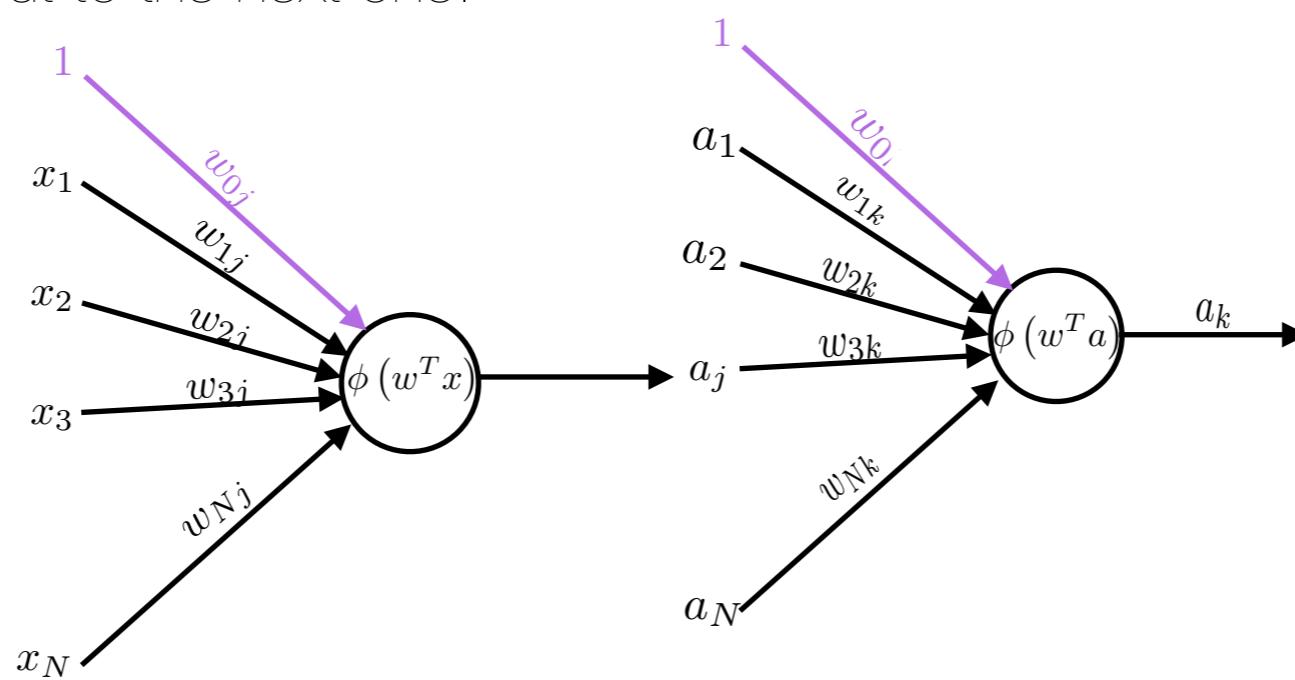
- Compute new sets of features

- Each layer builds up a more abstract representation of the data



Forward Propagation

- The output of a perceptron is determined by a sequence of steps:
 - obtain the inputs
 - multiply the inputs by the respective weights
 - calculate output using the activation function
- To create a multi-layer perceptron, you can simply use the output of one layer as the input to the next one.



Backward Propagation of Errors (BackProp)

- BackProp operates in two phases:
 - Forward propagate the inputs and calculate the deltas
 - Update the weights
- The error at the output is a **weighted average difference** between predicted output and the observed one.
- For inner layers there is no “real output”!

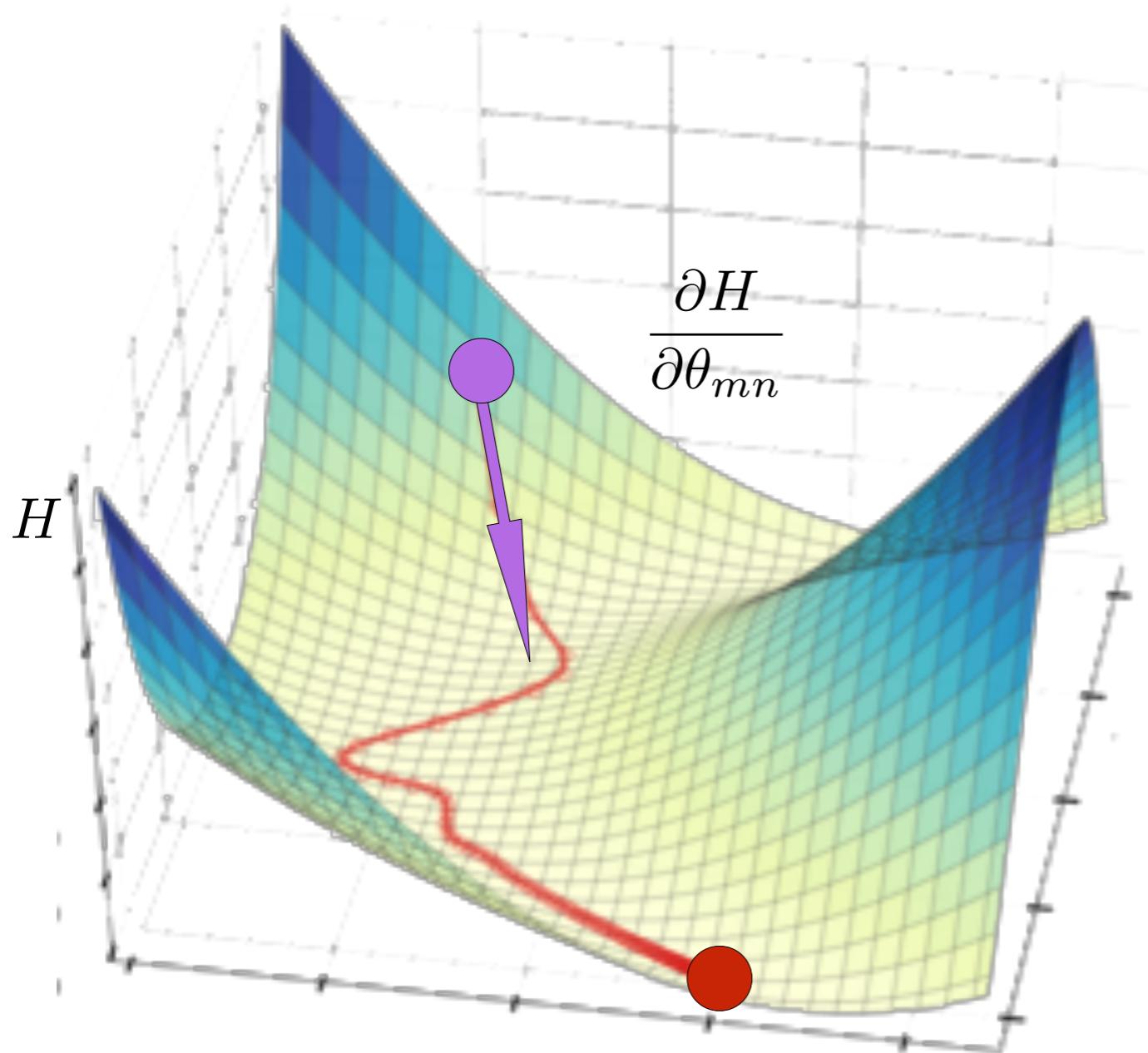
Loss Functions

- For learning to occur, we must quantify how far off we are from the desired output. There are two common ways of doing this:
 - Quadratic error function:
- Cross Entropy

$$E = \frac{1}{N} \sum_n |y_n - a_n|^2$$
$$J = -\frac{1}{N} \sum_n \left[y_n^T \log a_n + (1 - y_n)^T \log (1 - a_n) \right]$$

The **Cross Entropy** is complementary to **sigmoid** activation in the output layer and improves its stability.

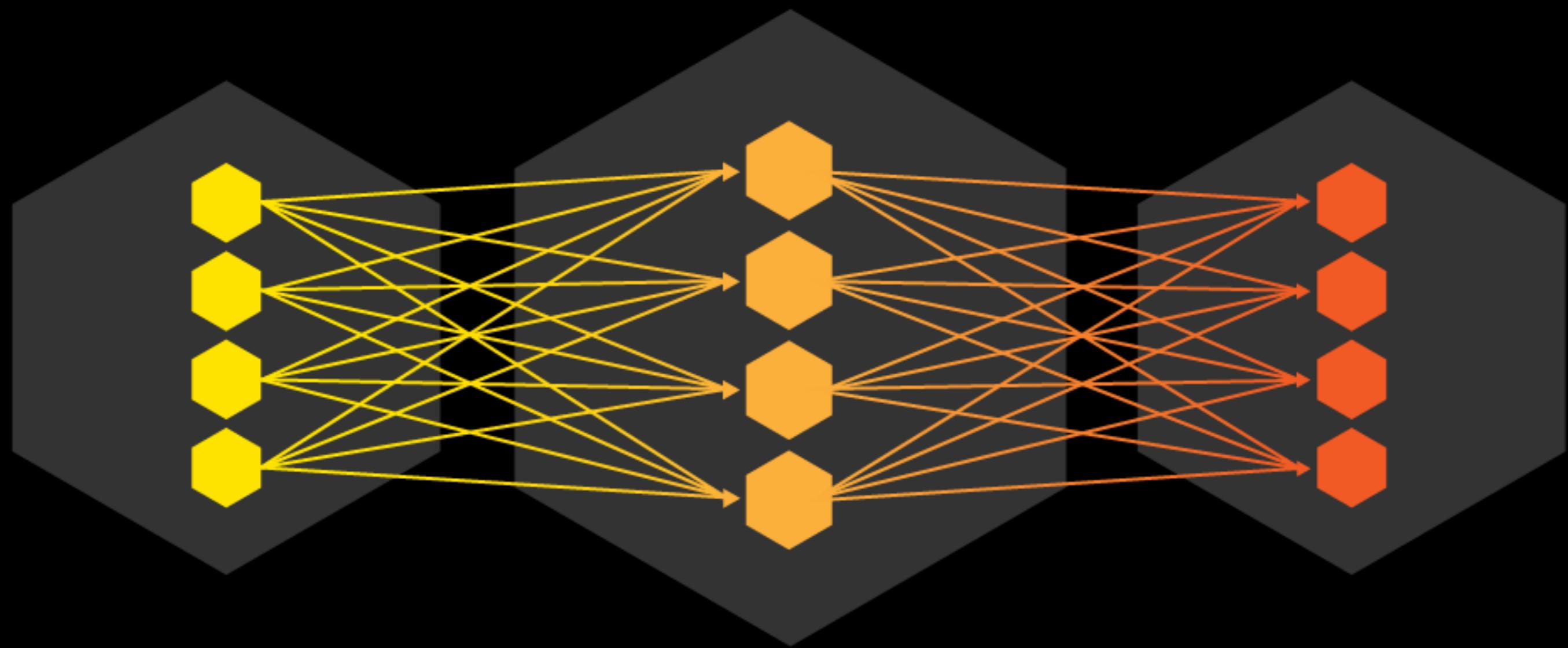
Gradient Descent



- Find the gradient for each training batch
- Take a step **downhill** along the direction of the gradient

$$\theta_{mn} \leftarrow \theta_{mn} - \alpha \frac{\partial H}{\partial \theta_{mn}}$$

- where α is the step size.
- Repeat until "convergence".



INPUT TERMS

FEATURES
PREDICTIONS
ATTRIBUTES
PREDICTABLE VARIABLES

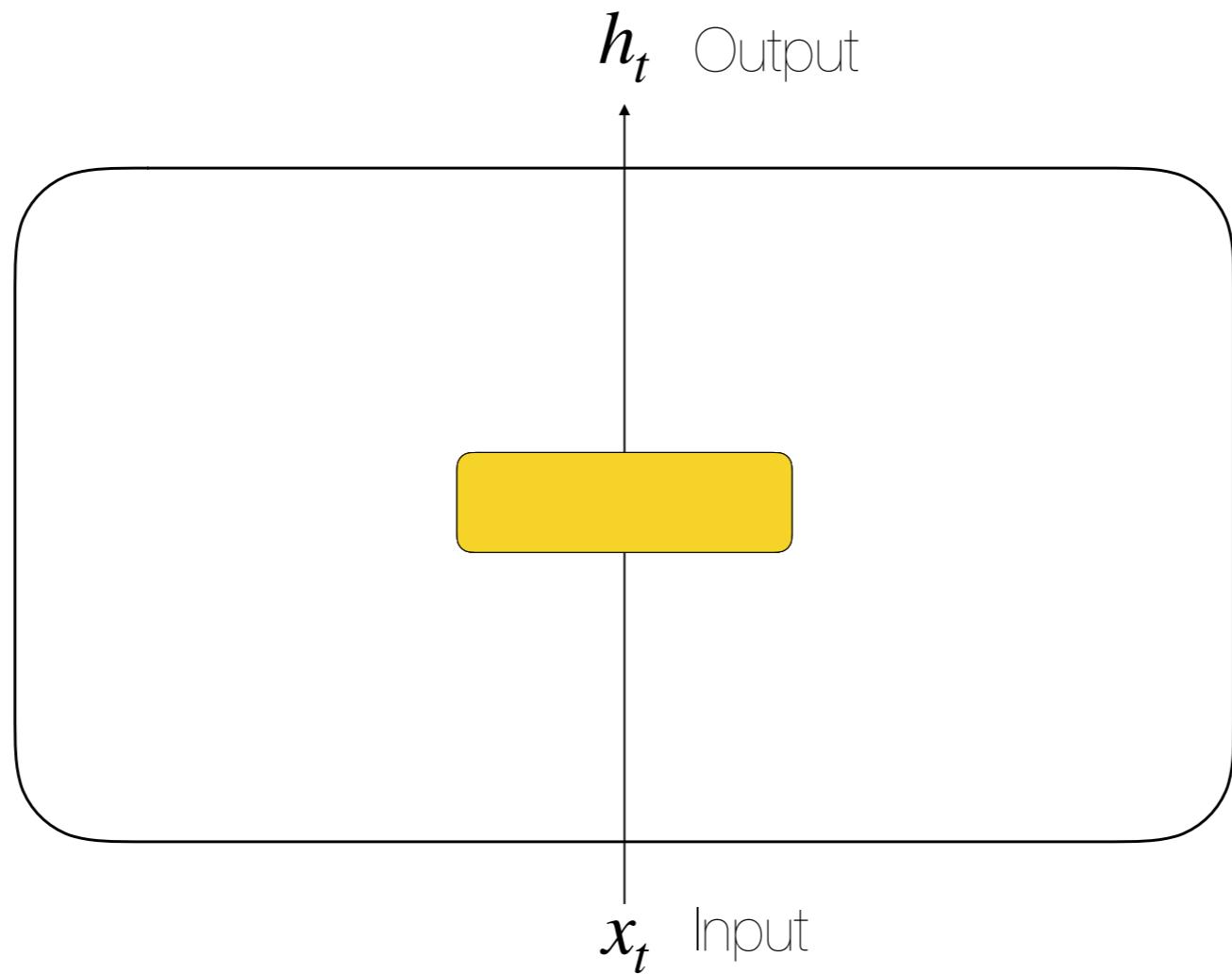
MACHINE

ALGORITHMS
TECHNIQUES
MODELS

OUTPUT TERMS

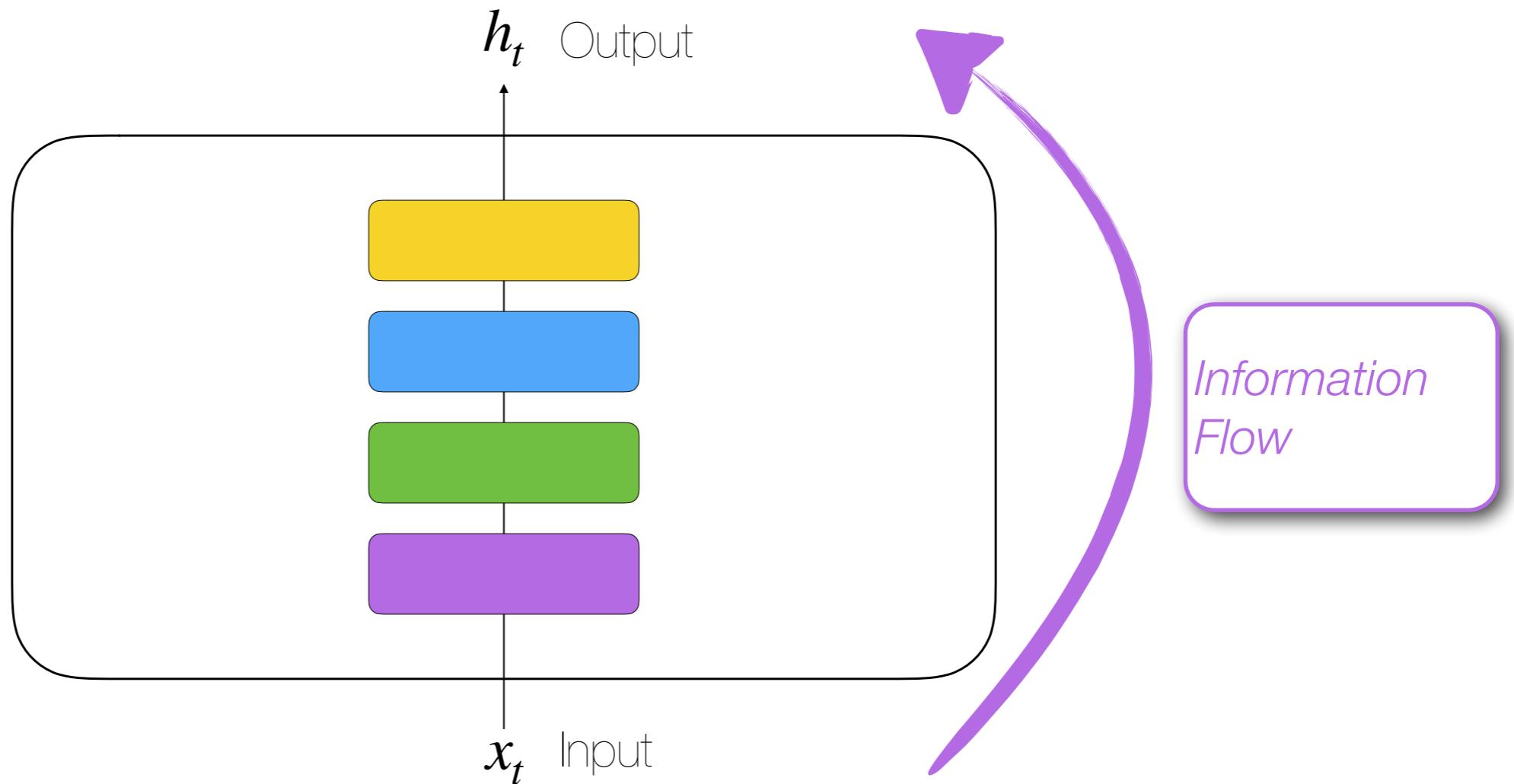
CLASSES
RESPONSES
TARGETS
DEPENDANT VARIABLES

Feed Forward Networks



$$h_t = f(x_t)$$

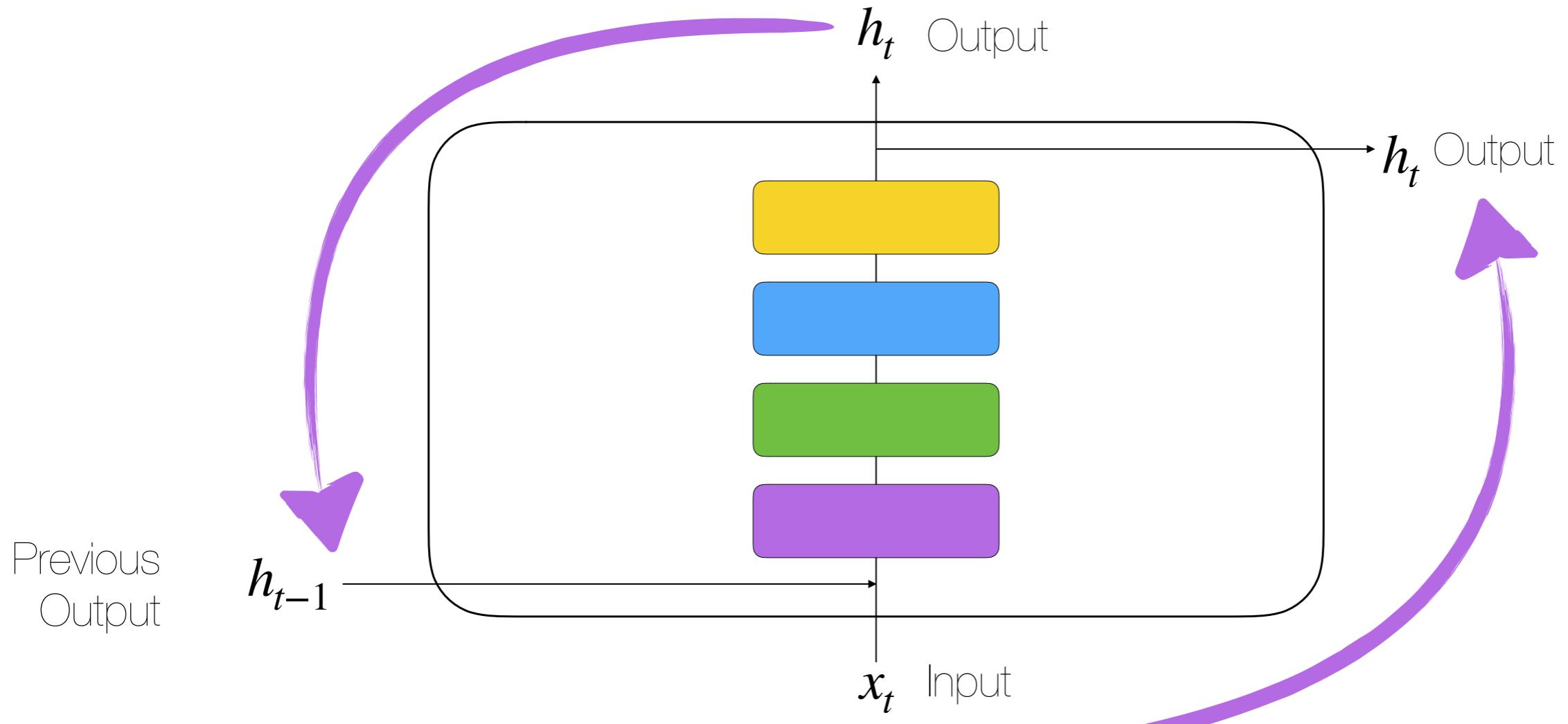
Feed Forward Networks



$$h_t = f(x_t)$$

Information
Flow

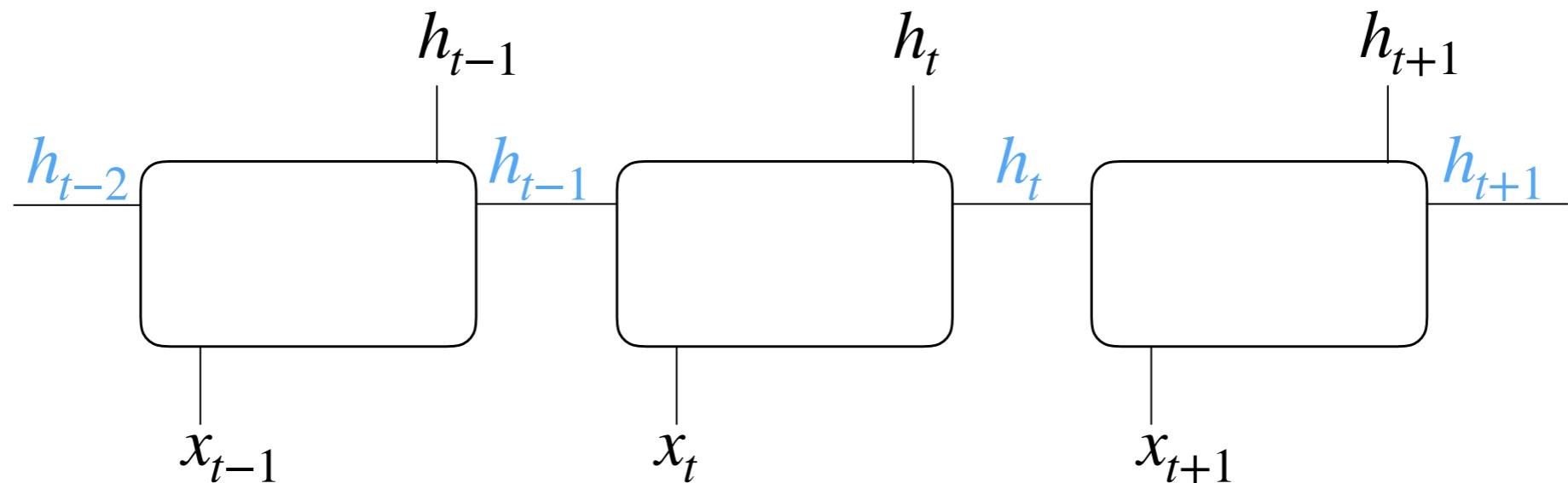
Recurrent Neural Network (RNN)



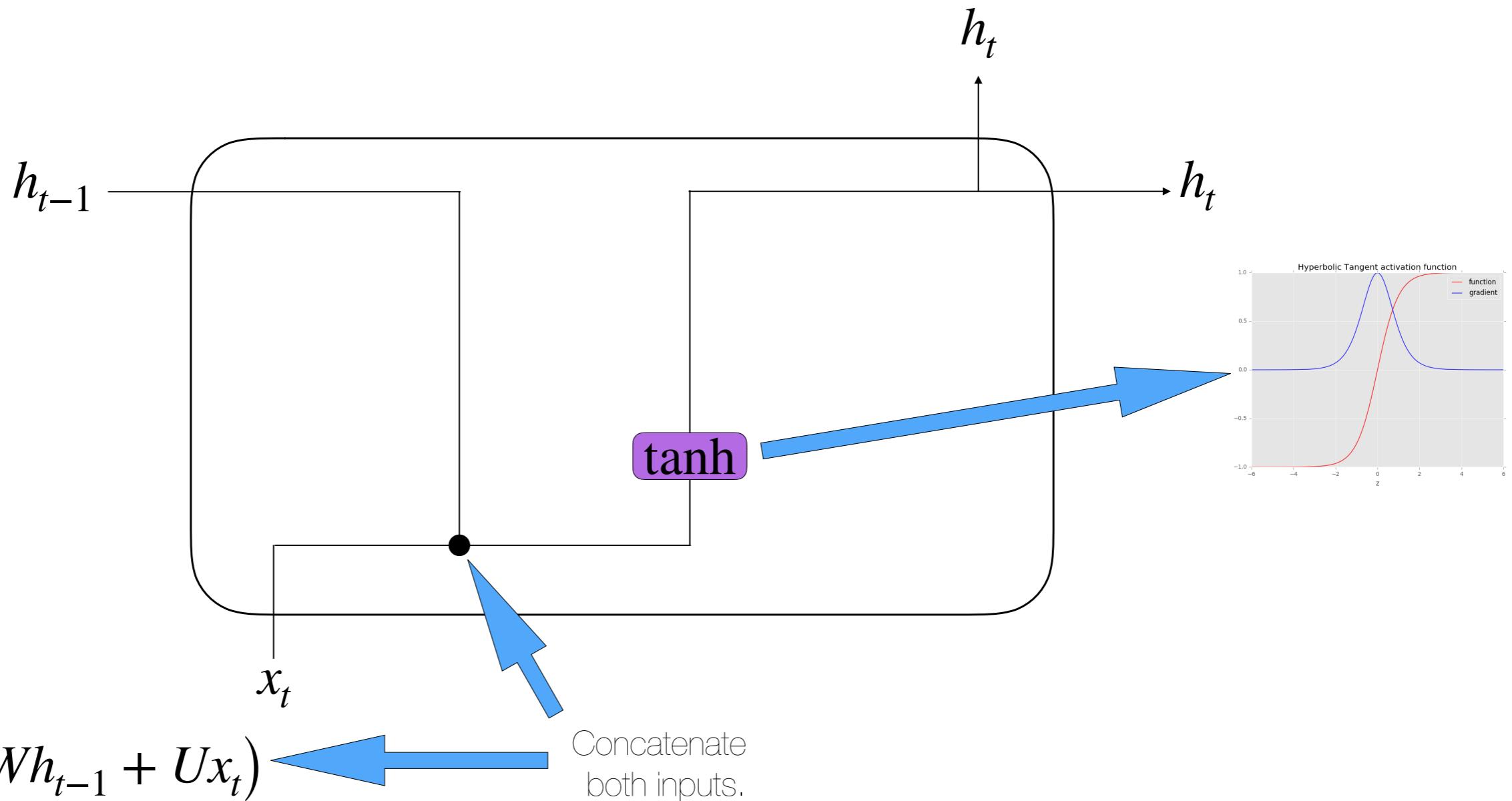
$$h_t = f(x_t, h_{t-1})$$

Recurrent Neural Network (RNN)

- Each output depends (implicitly) on all previous **outputs**.
- Input sequences generate output sequences (**seq2seq**)



Recurrent Neural Network (RNN)



$$h_t = \tanh (Wh_{t-1} + Ux_t)$$

Timeseries

- Temporal sequence of data points
- Consecutive points are strongly correlated
- Common in statistics, signal processing, econometrics, mathematical finance, earthquake prediction, etc

- Numeric (real or discrete) or symbolic data

- Supervised Learning problem:

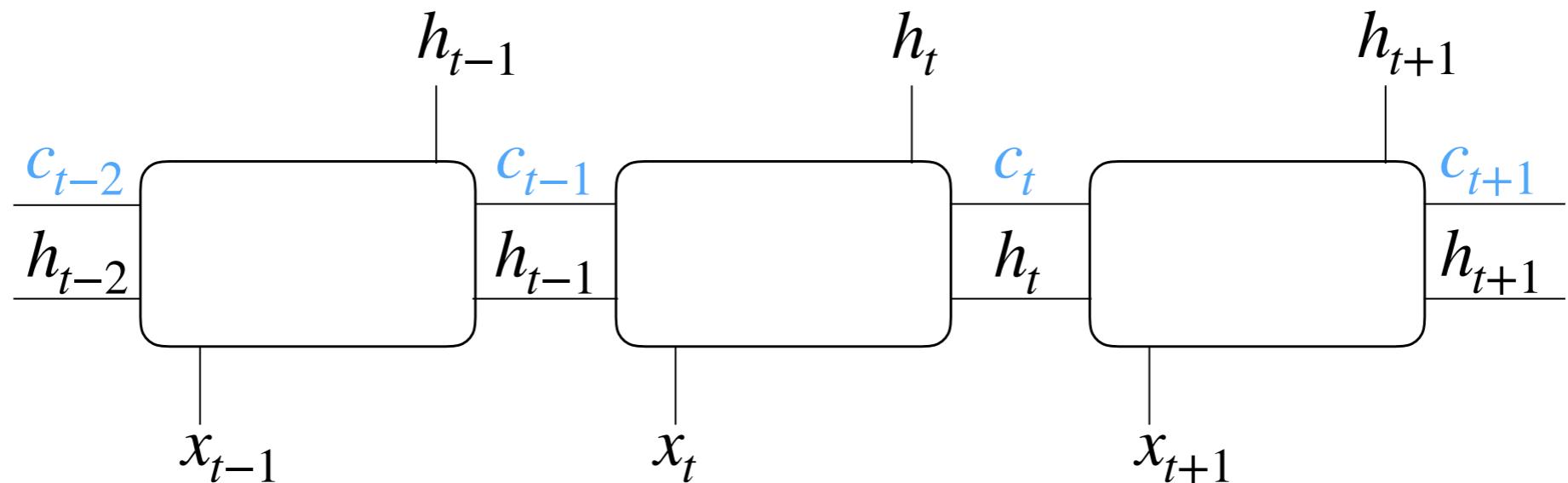
$$X_t = f(X_{t-1}, \dots, X_{t-n})$$



github.com/DataForScience/RNN

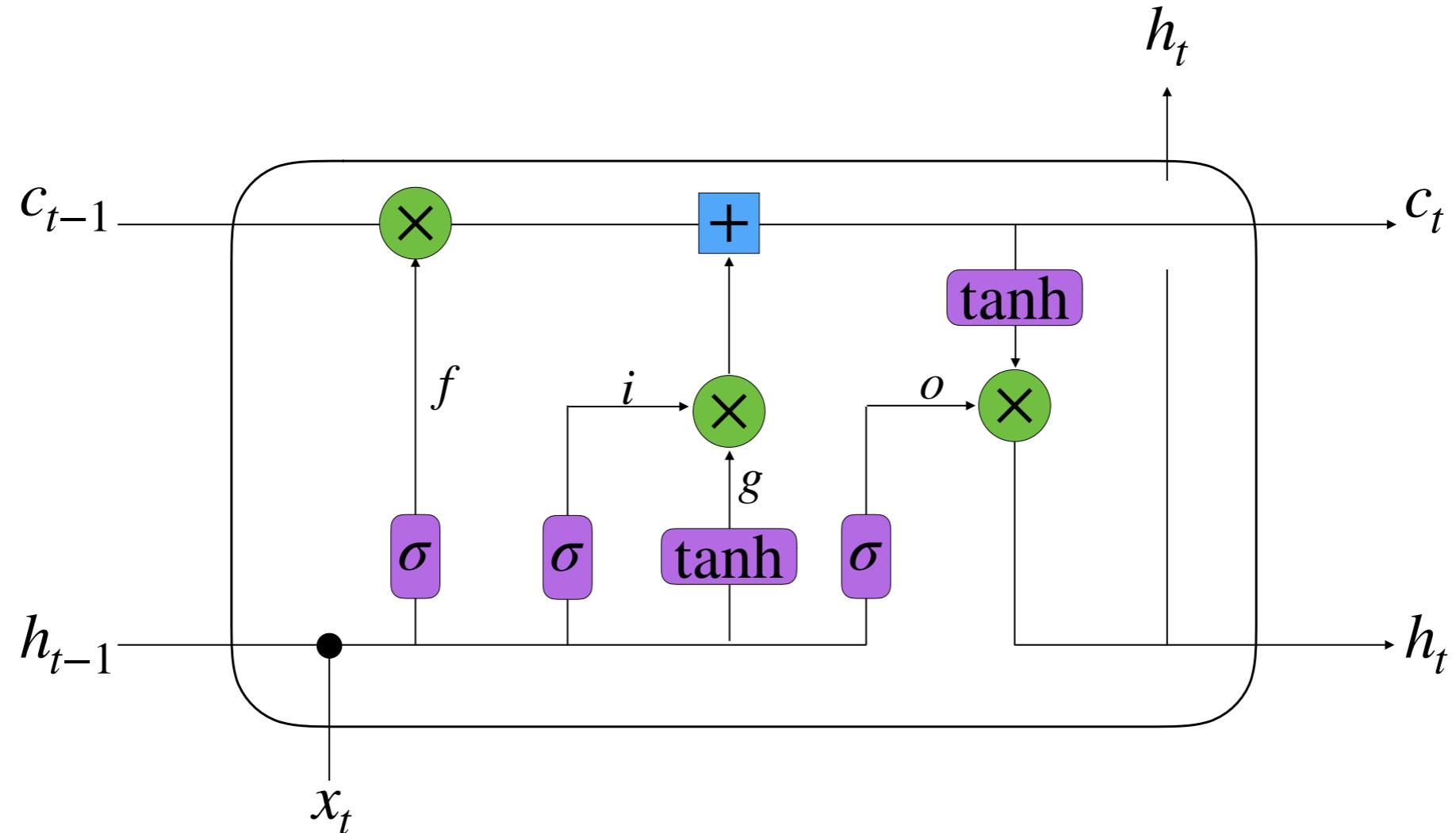
Long-Short Term Memory (LSTM)

- What if we want to keep explicit information about previous states (**memory**)?
- How much information is kept, can be controlled through gates.
- LSTMs were first introduced in **1997** by Hochreiter and Schmidhuber



Long-Short Term Memory (LSTM)

-  Element wise addition
-  Element wise multiplication
-  1 minus the input



$$f = \sigma(W_f h_{t-1} + U_f x_t)$$

$$g = \tanh(W_g h_{t-1} + U_g x_t)$$

$$i = \sigma(W_i h_{t-1} + U_i x_t)$$

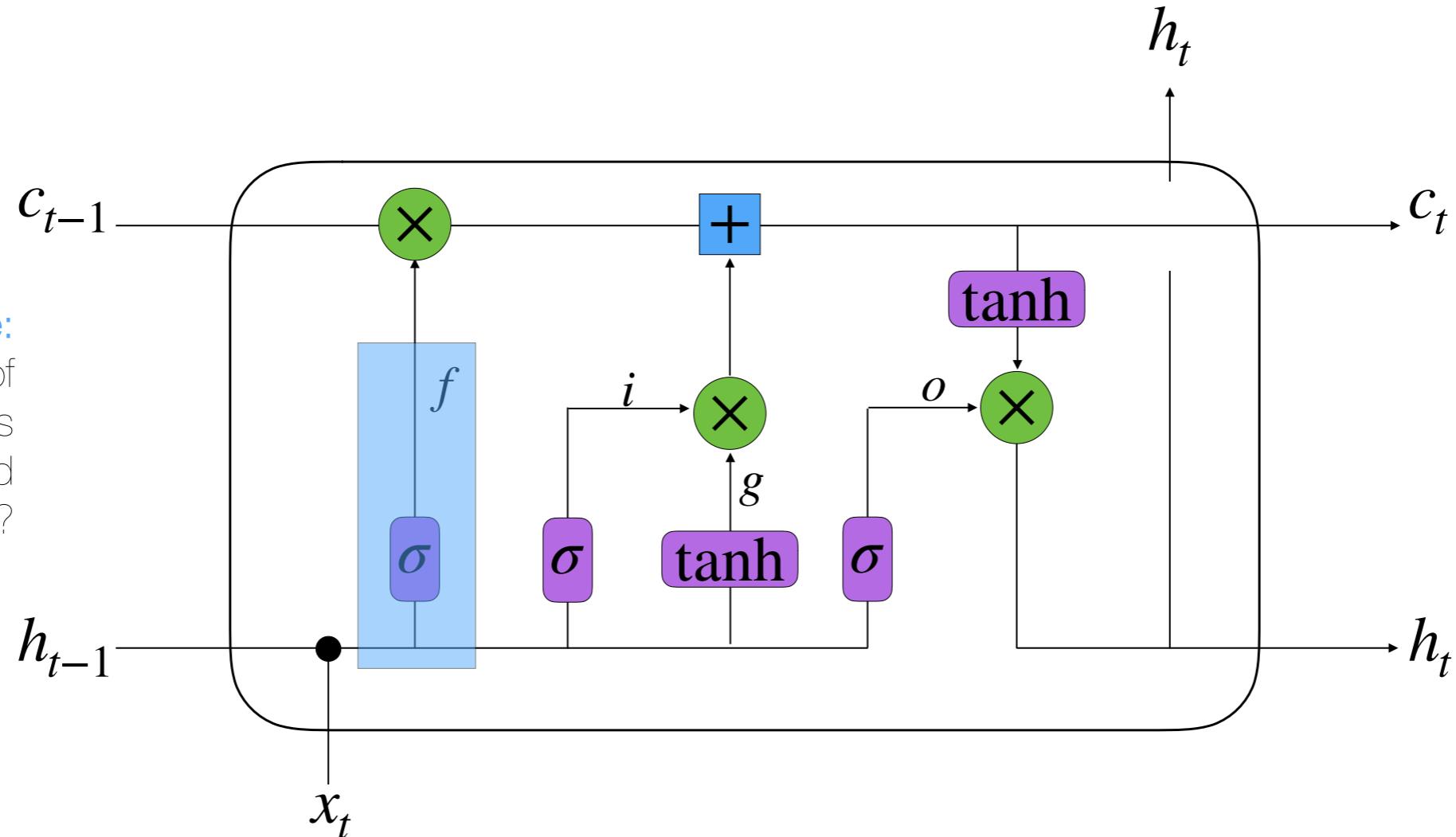
$$c_t = (c_{t-1} \otimes f) + (g \otimes i)$$

$$o = \sigma(W_o h_{t-1} + U_o x_t)$$

$$h_t = \tanh(c_t) \otimes o$$

Long-Short Term Memory (LSTM)

-  Element wise addition
-  Element wise multiplication
-  1 minus the input



$$f = \sigma(W_f h_{t-1} + U_f x_t)$$

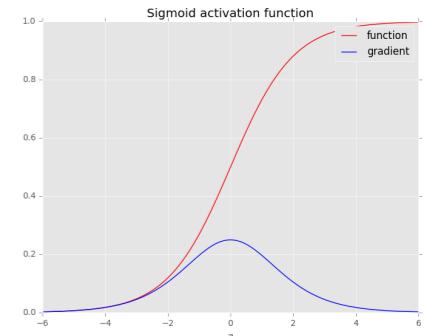
$$i = \sigma(W_i h_{t-1} + U_i x_t)$$

$$o = \sigma(W_o h_{t-1} + U_o x_t)$$

$$g = \tanh(W_g h_{t-1} + U_g x_t)$$

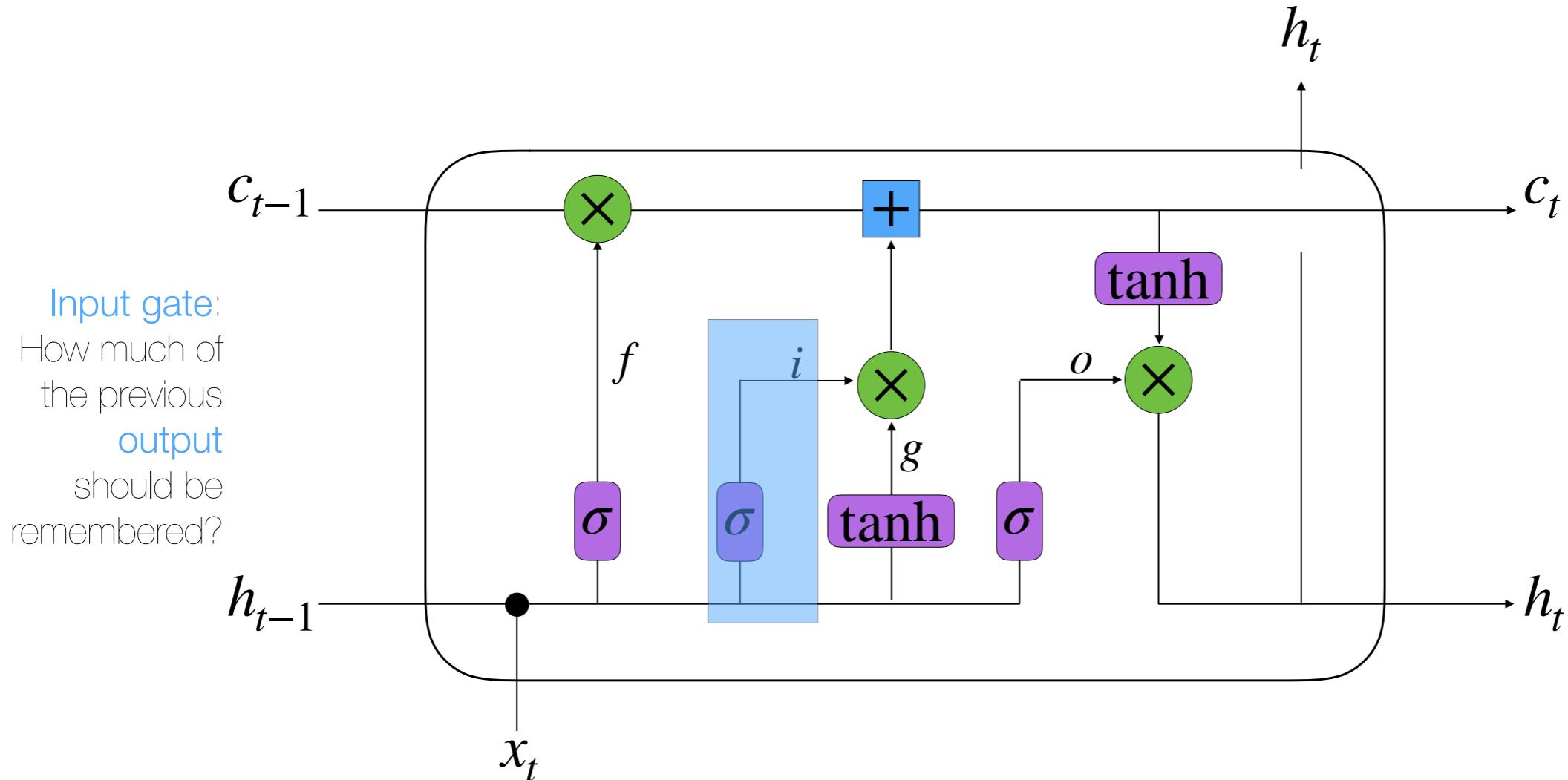
$$c_t = (c_{t-1} \otimes f) + (g \otimes i)$$

$$h_t = \tanh(c_t) \otimes o$$



Long-Short Term Memory (LSTM)

-  Element wise addition
-  Element wise multiplication
-  1 minus the input



$$f = \sigma(W_f h_{t-1} + U_f x_t)$$

$$g = \tanh(W_g h_{t-1} + U_g x_t)$$

$$i = \sigma(W_i h_{t-1} + U_i x_t)$$

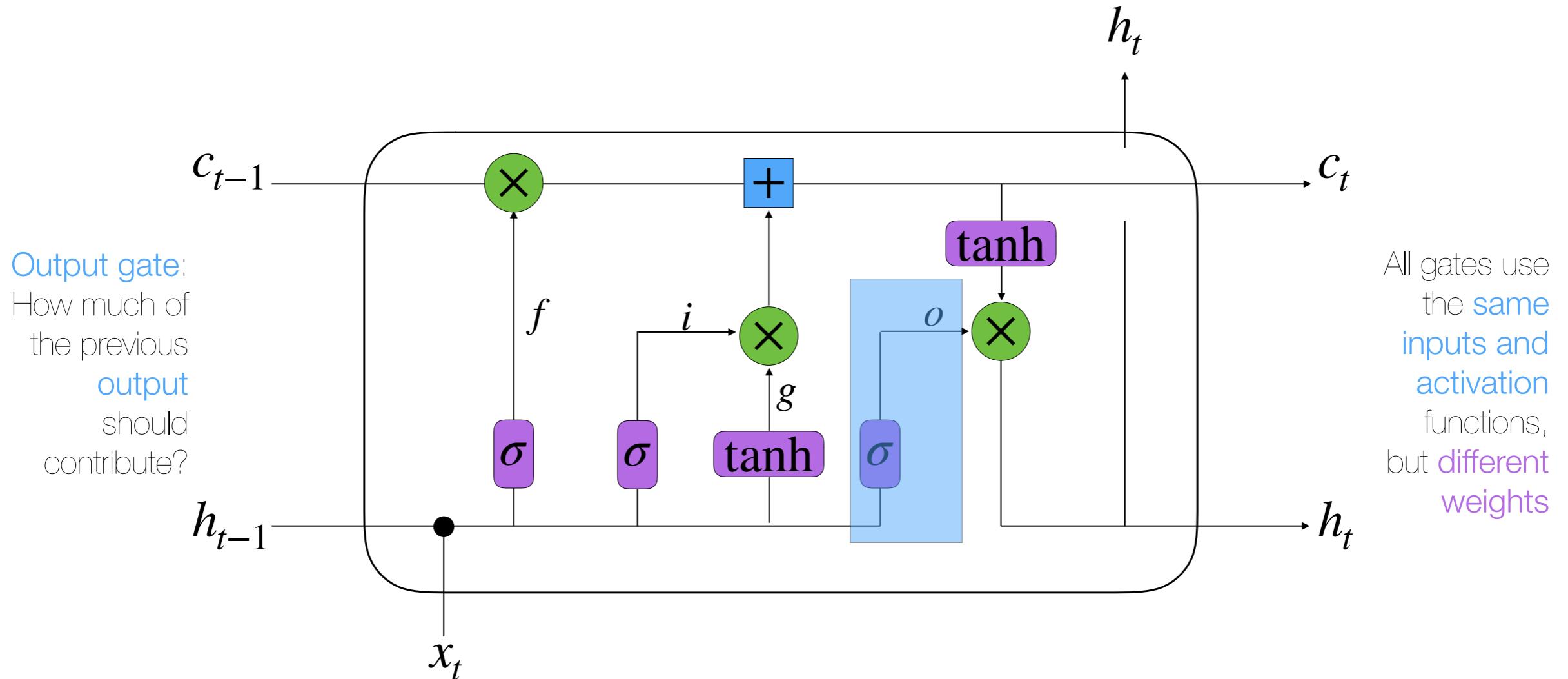
$$c_t = (c_{t-1} \otimes f) + (g \otimes i)$$

$$o = \sigma(W_o h_{t-1} + U_o x_t)$$

$$h_t = \tanh(c_t) \otimes o$$

Long-Short Term Memory (LSTM)

- + Element wise addition
- × Element wise multiplication
- 1- 1 minus the input



$$f = \sigma(W_f h_{t-1} + U_f x_t)$$

$$g = \tanh(W_g h_{t-1} + U_g x_t)$$

$$i = \sigma(W_i h_{t-1} + U_i x_t)$$

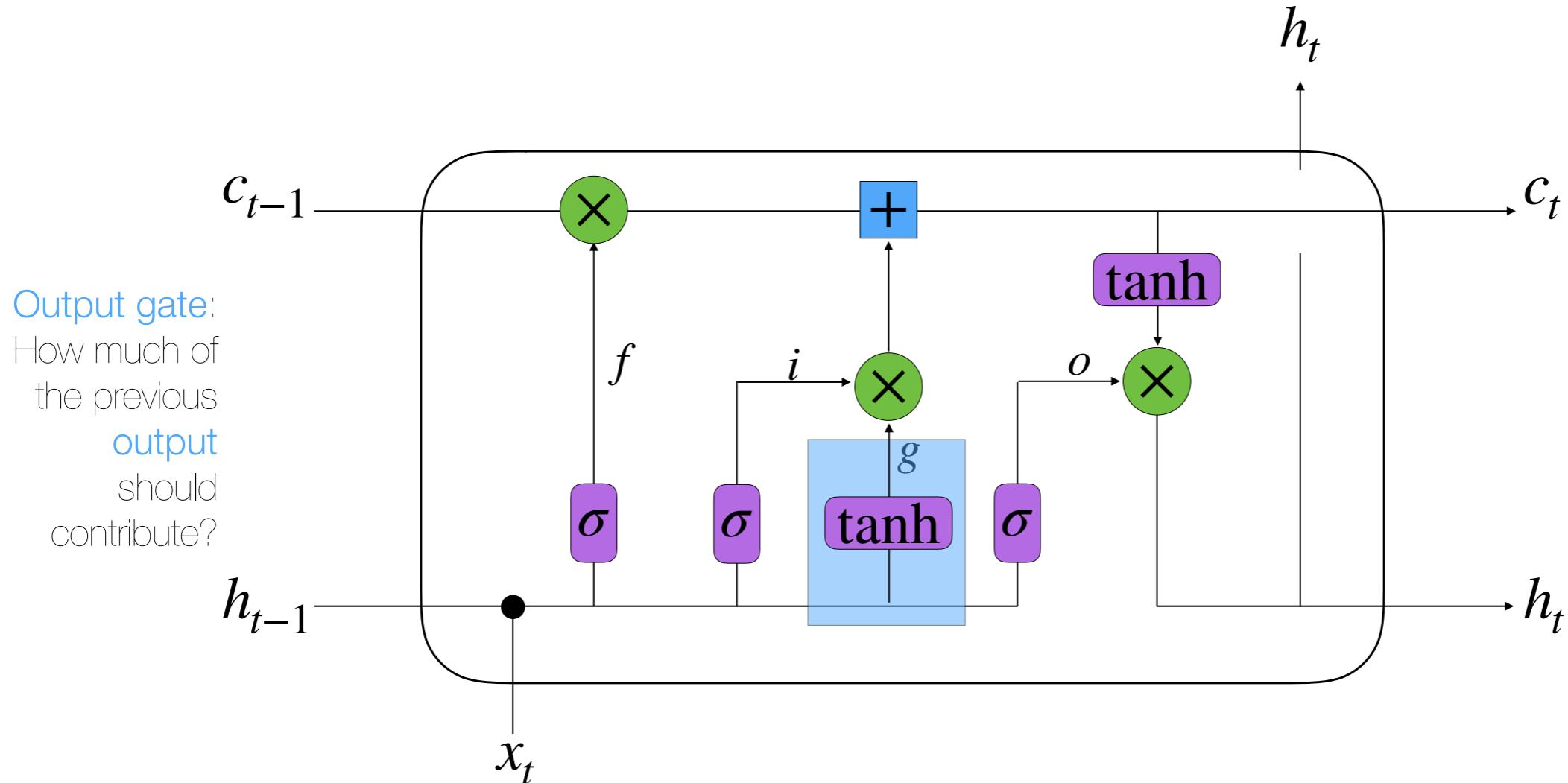
$$c_t = (c_{t-1} \otimes f) + (g \otimes i)$$

$$o = \sigma(W_o h_{t-1} + U_o x_t)$$

$$h_t = \tanh(c_t) \otimes o$$

Long-Short Term Memory (LSTM)

- + Element wise addition
- × Element wise multiplication
- 1- 1 minus the input



$$f = \sigma(W_f h_{t-1} + U_f x_t)$$

$$g = \tanh(W_g h_{t-1} + U_g x_t)$$

$$i = \sigma(W_i h_{t-1} + U_i x_t)$$

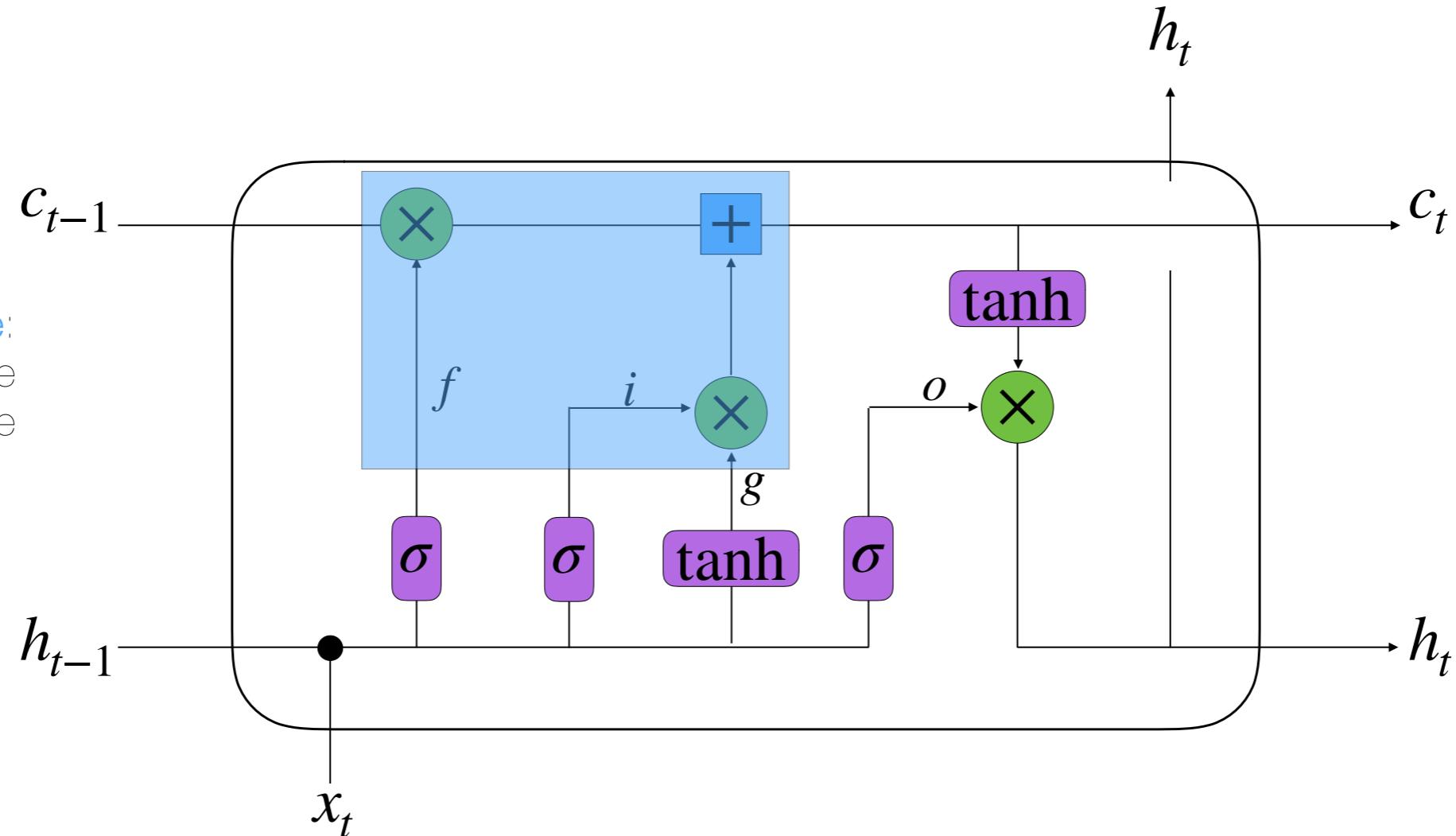
$$c_t = (c_{t-1} \otimes f) + (g \otimes i)$$

$$o = \sigma(W_o h_{t-1} + U_o x_t)$$

$$h_t = \tanh(c_t) \otimes o$$

Long-Short Term Memory (LSTM)

-  Element wise addition
-  Element wise multiplication
-  1 minus the input



$$f = \sigma(W_f h_{t-1} + U_f x_t)$$

$$g = \tanh(W_g h_{t-1} + U_g x_t)$$

$$i = \sigma(W_i h_{t-1} + U_i x_t)$$

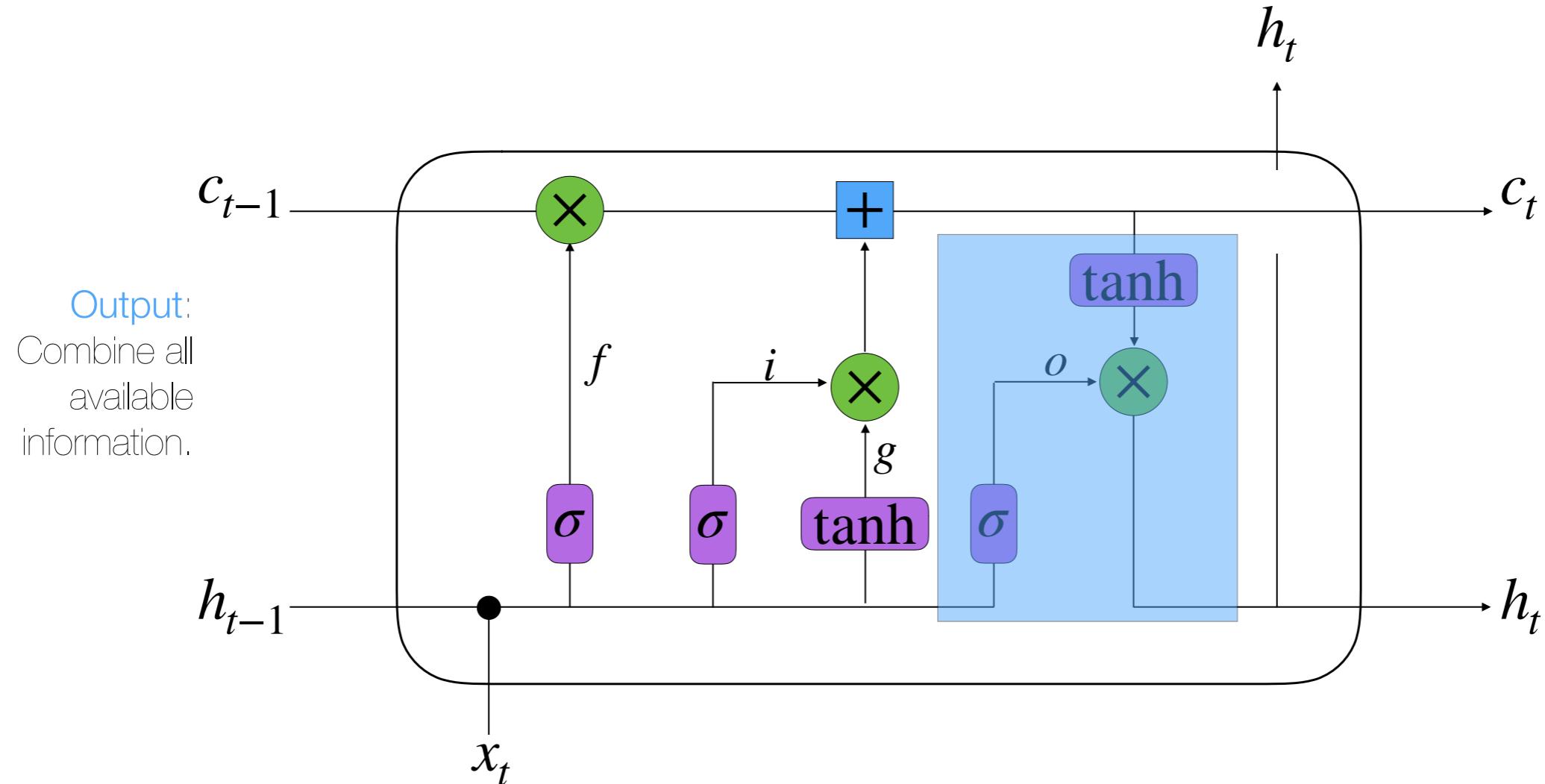
$$c_t = (c_{t-1} \otimes f) + (g \otimes i)$$

$$o = \sigma(W_o h_{t-1} + U_o x_t)$$

$$h_t = \tanh(c_t) \otimes o$$

Long-Short Term Memory (LSTM)

-  Element wise addition
-  Element wise multiplication
-  1 minus the input



$$f = \sigma(W_f h_{t-1} + U_f x_t)$$

$$g = \tanh(W_g h_{t-1} + U_g x_t)$$

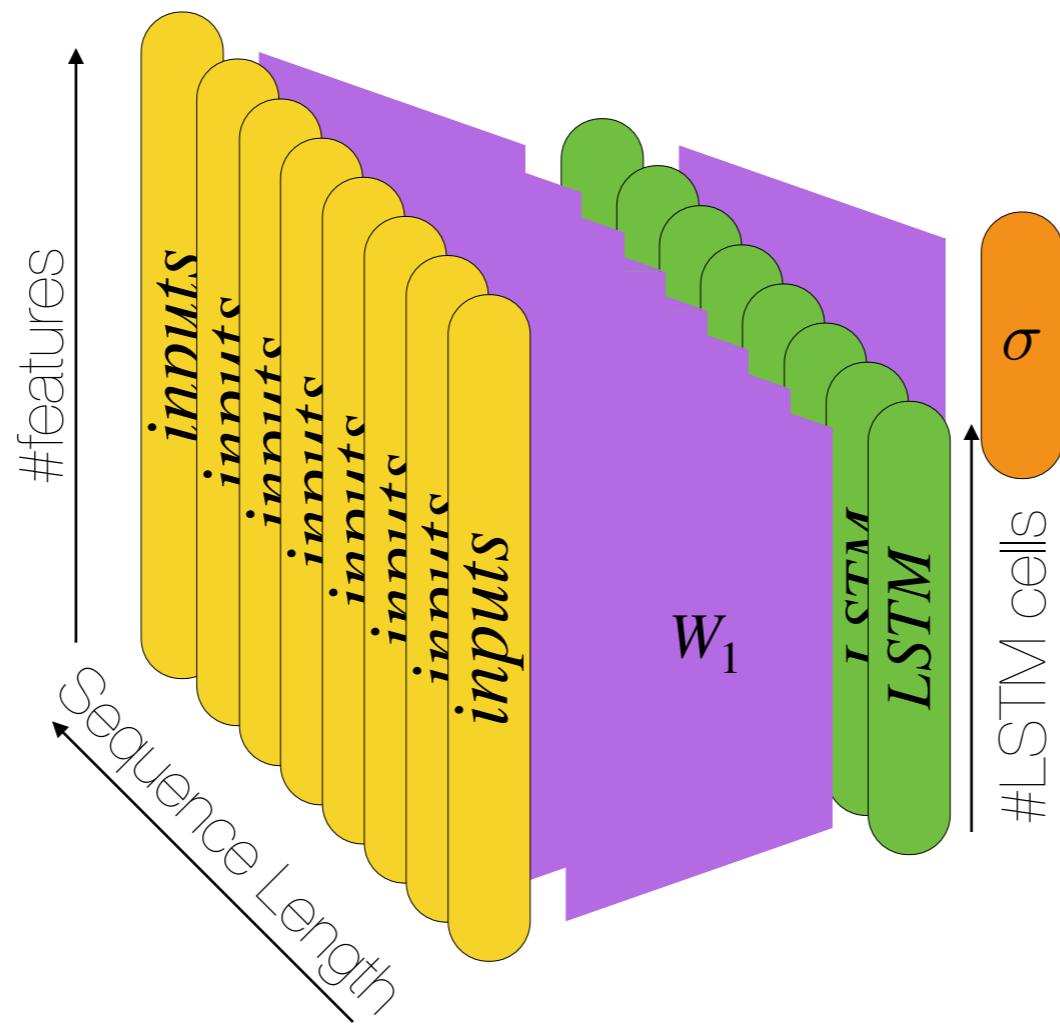
$$i = \sigma(W_i h_{t-1} + U_i x_t)$$

$$c_t = (c_{t-1} \otimes f) + (g \otimes i)$$

$$o = \sigma(W_o h_{t-1} + U_o x_t)$$

$$h_t = \tanh(c_t) \otimes o$$

Using LSTMs

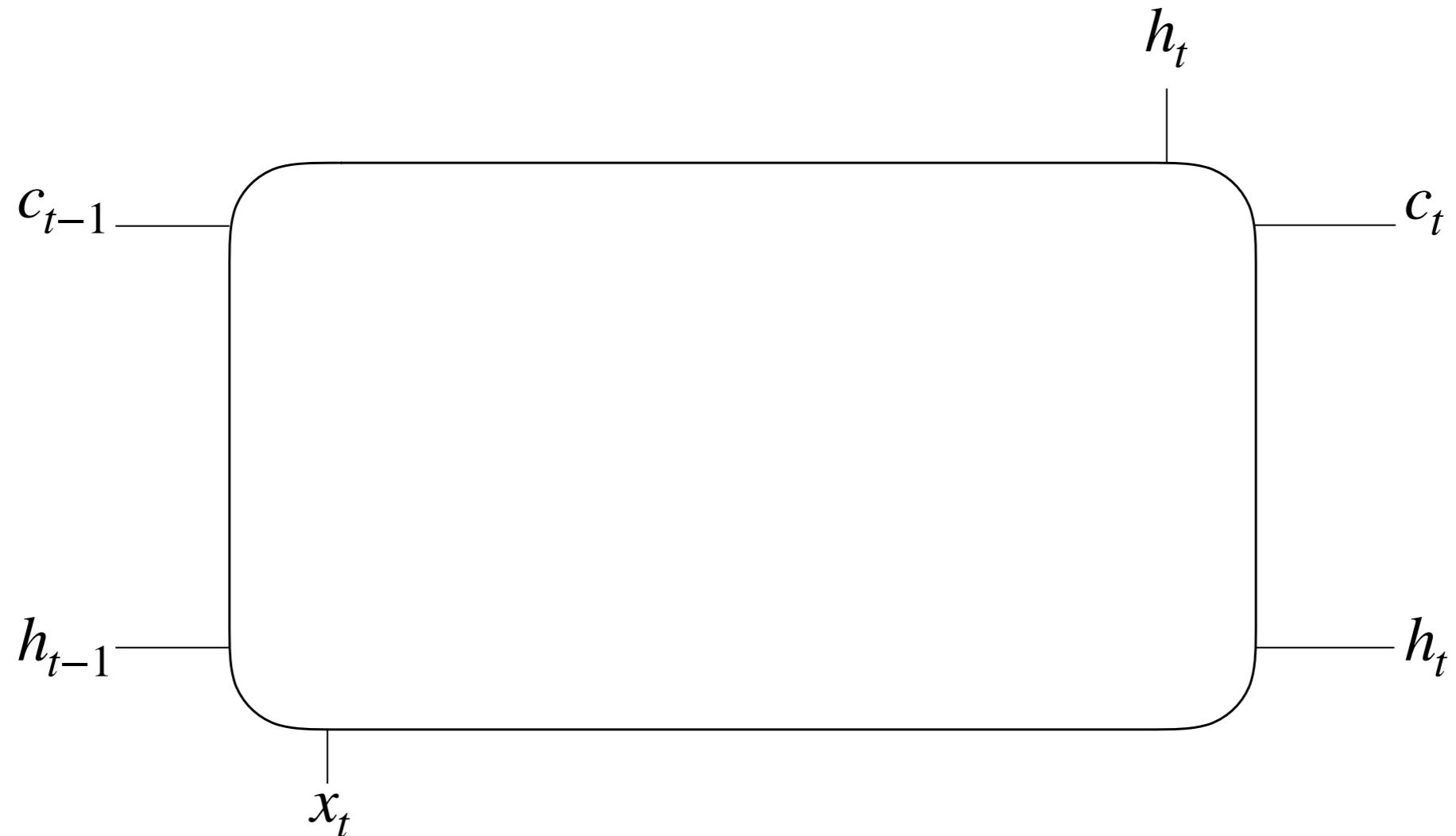


github.com/DataForScience/RNN

Applications

- Language Modeling and Prediction
- Speech Recognition
- Machine Translation
- Part-of-Speech Tagging
- Sentiment Analysis
- Summarization
- Time series forecasting

Neural Networks?



Or legos?

<https://keras.io>



Keras

<https://keras.io>

- Open Source neural network library written in Python
- TensorFlow, Microsoft Cognitive Toolkit or Theano backends
- Enables fast experimentation
- Created and maintained by François Chollet, a Google engineer.
- Implements Layers, Objective/Loss functions, Activation functions, Optimizers, etc...

- `keras.models.Sequential(layers=None, name=None)`- is the workhorse. You use it to build a model layer by layer. Returns the object that we will use to build the **model**

- **keras.layers**

- `Dense(units, activation=None, use_bias=True)` - None means linear activation. Other options are, '`tanh`', '`sigmoid`', '`softmax`', '`relu`', etc.
- `Dropout(rate, seed=None)`
- `Activation(activation)` - Same as the activation option to `Dense`, can also be used to pass `TensorFlow` or `Theano` operations directly.
- `SimpleRNN(units, input_shape, activation='tanh', use_bias=True, dropout=0.0, return_sequences=False)`
- `GRU(units, input_shape, activation='tanh', use_bias=True, dropout=0.0, return_sequences=False)`

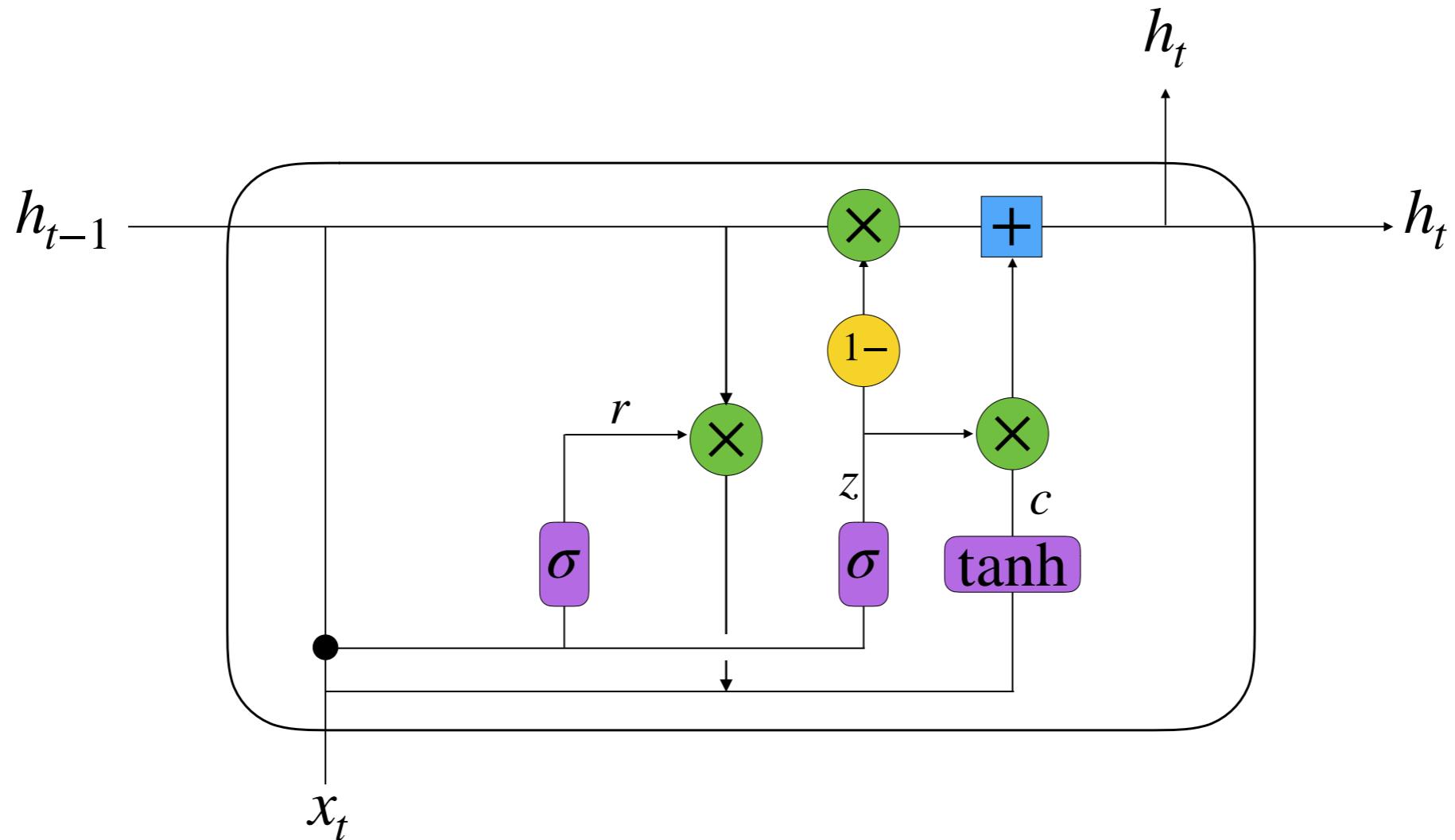
- `model = Sequential()`
- `model.add(layer)` - Add a layer to the top of the model
- `model.compile(optimizer, loss)` - We have to compile the model before we can use it
 - optimizer - ‘adam’, ‘sgd’, ‘rmsprop’, etc...
 - loss - ‘mean_squared_error’, ‘categorical_crossentropy’, ‘kullback_leibler_divergence’, etc...
- `model.fit(x=None, y=None, batch_size=None, epochs=1, verbose=1, validation_split=0.0, validation_data=None, shuffle=True)`
- `model.predict(x, batch_size=32, verbose=0)` - fit/predict interface

Gated Recurrent Unit (GRU)

- Introduced in [2014](#) by K. Cho
- Meant to solve the [Vanishing Gradient Problem](#)
- Can be considered as a [simplification of LSTMs](#)
- [Similar performance](#) to LSTM in some applications, [better performance](#) for [smaller datasets](#).

Gated Recurrent Unit (GRU)

-  Element wise addition
-  Element wise multiplication
-  1 minus the input



$$z = \sigma(W_z h_{t-1} + U_z x_t)$$

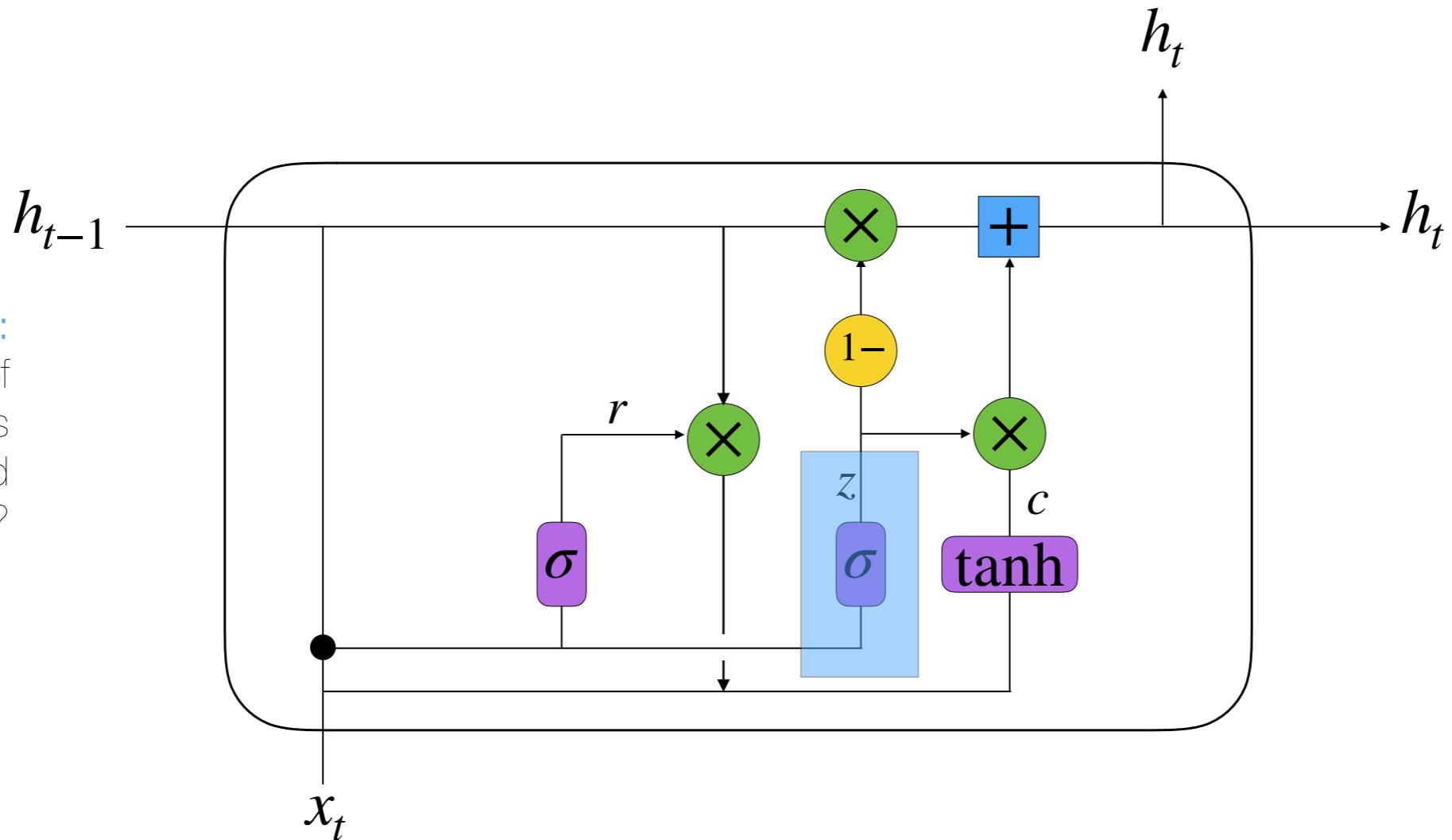
$$r = \sigma(W_r h_{t-1} + U_r x_t)$$

$$c = \tanh\left(W_c (h_{t-1} \otimes r) + U_c x_t\right)$$

$$h_t = (z \otimes c) + ((1 - z) \otimes h_{t-1})$$

Gated Recurrent Unit (GRU)

- + Element wise addition
- × Element wise multiplication
- 1- 1 minus the input



$$z = \sigma(W_z h_{t-1} + U_z x_t)$$

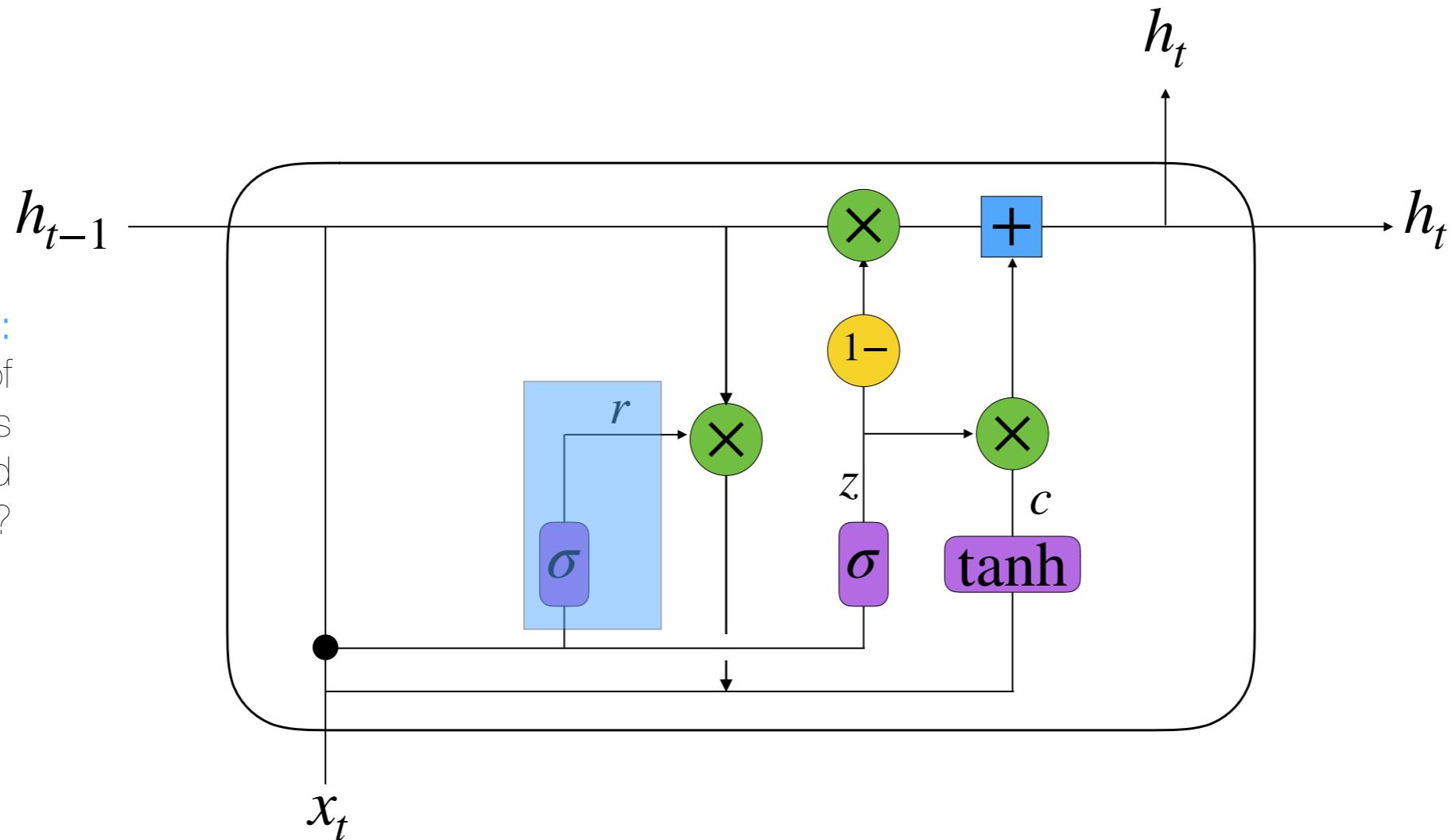
$$r = \sigma(W_r h_{t-1} + U_r x_t)$$

$$c = \tanh\left(W_c (h_{t-1} \otimes r) + U_c x_t\right)$$

$$h_t = (z \otimes c) + ((1 - z) \otimes h_{t-1})$$

Gated Recurrent Unit (GRU)

- + Element wise addition
- × Element wise multiplication
- 1- 1 minus the input



$$z = \sigma(W_z h_{t-1} + U_z x_t)$$

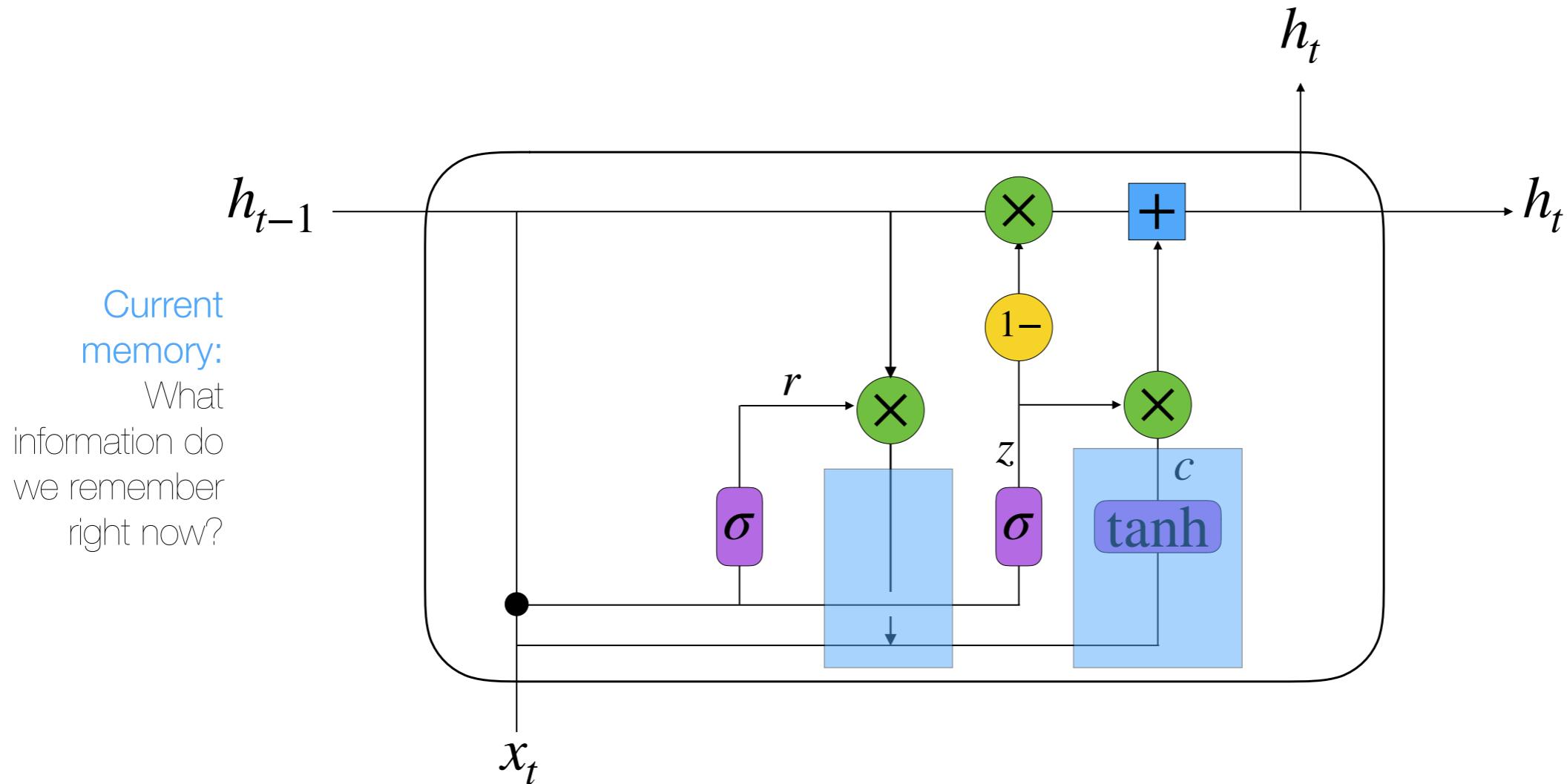
$$r = \sigma(W_r h_{t-1} + U_r x_t)$$

$$c = \tanh\left(W_c (h_{t-1} \otimes r) + U_c x_t\right)$$

$$h_t = (z \otimes c) + ((1 - z) \otimes h_{t-1})$$

Gated Recurrent Unit (GRU)

- + Element wise addition
- × Element wise multiplication
- 1- 1 minus the input



$$z = \sigma(W_z h_{t-1} + U_z x_t)$$

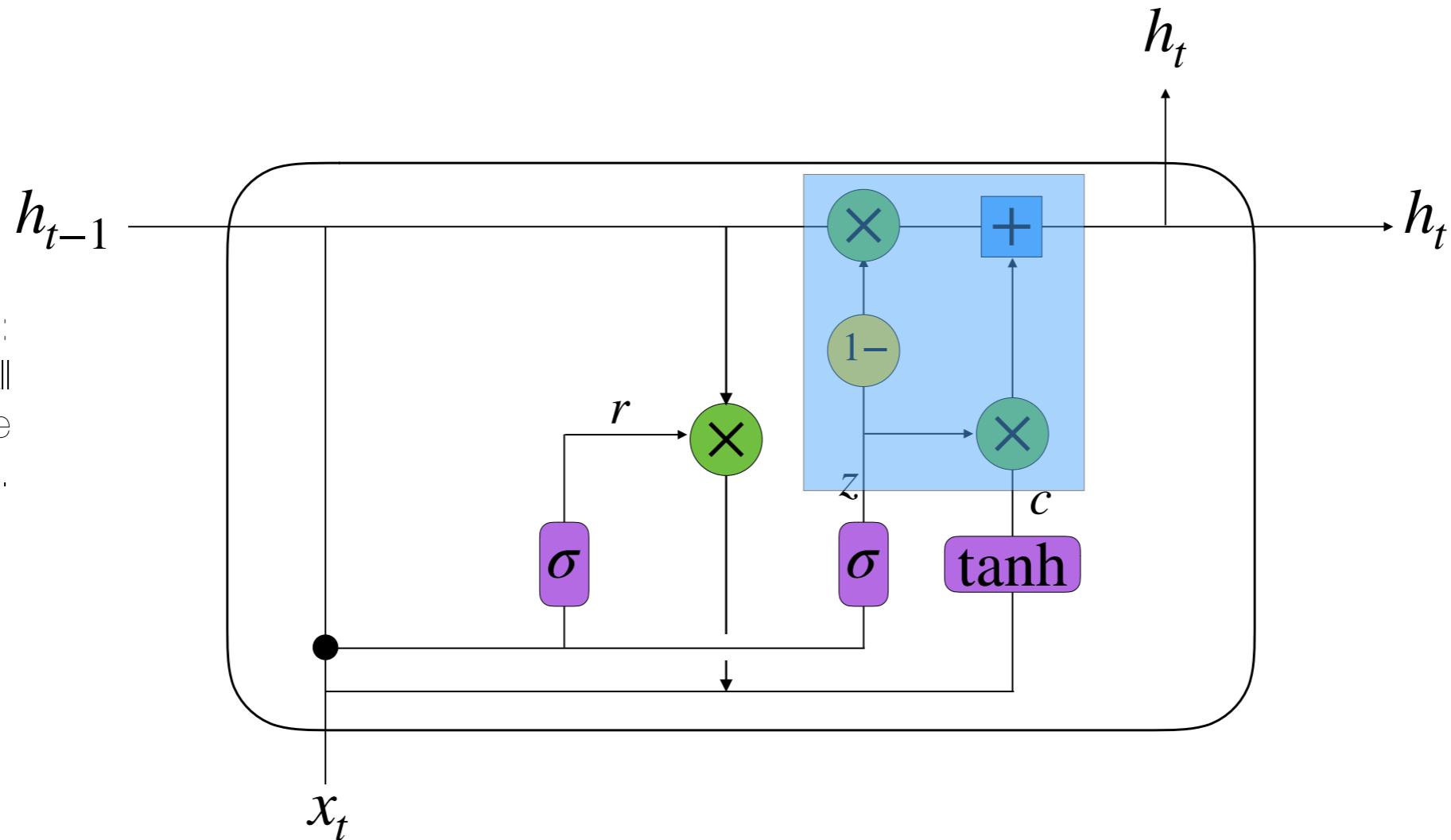
$$r = \sigma(W_r h_{t-1} + U_r x_t)$$

$$c = \tanh\left(W_c (h_{t-1} \otimes r) + U_c x_t\right)$$

$$h_t = (z \otimes c) + ((1 - z) \otimes h_{t-1})$$

Gated Recurrent Unit (GRU)

- + Element wise addition
- × Element wise multiplication
- 1- 1 minus the input



$$z = \sigma(W_z h_{t-1} + U_z x_t)$$

$$r = \sigma(W_r h_{t-1} + U_r x_t)$$

$$c = \tanh(W_c (h_{t-1} \otimes r) + U_c x_t)$$

$$h_t = (z \otimes c) + ((1 - z) \otimes h_{t-1})$$