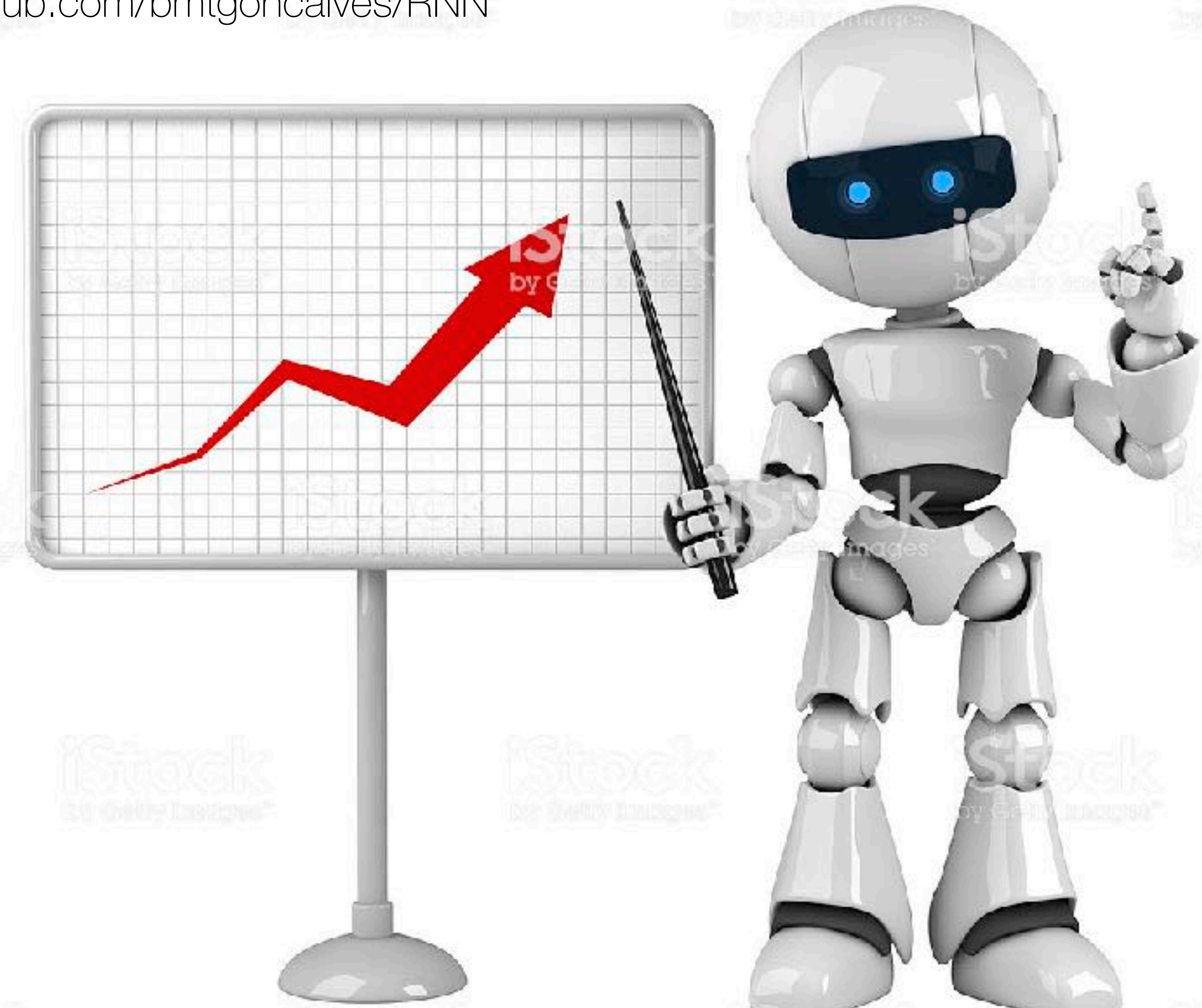
NYU DATA SCIENCE

RNNs for Timeseries Analysis

Bruno Gonçalves

www.bgoncalves.com

github.com/bmtgoncalves/RNN

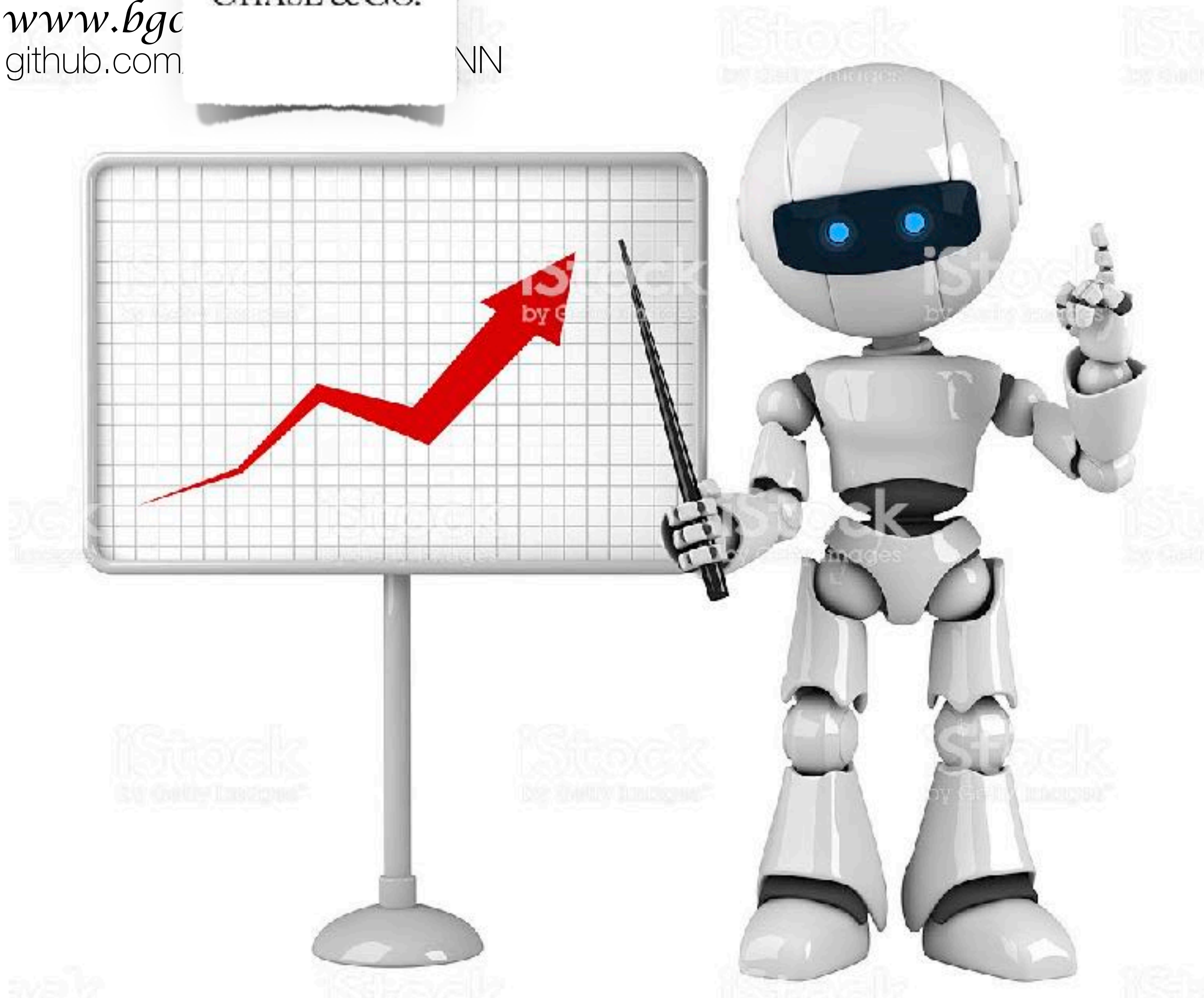


RNNs f ries Analysis

Bruno G

www.bg

github.com. NN



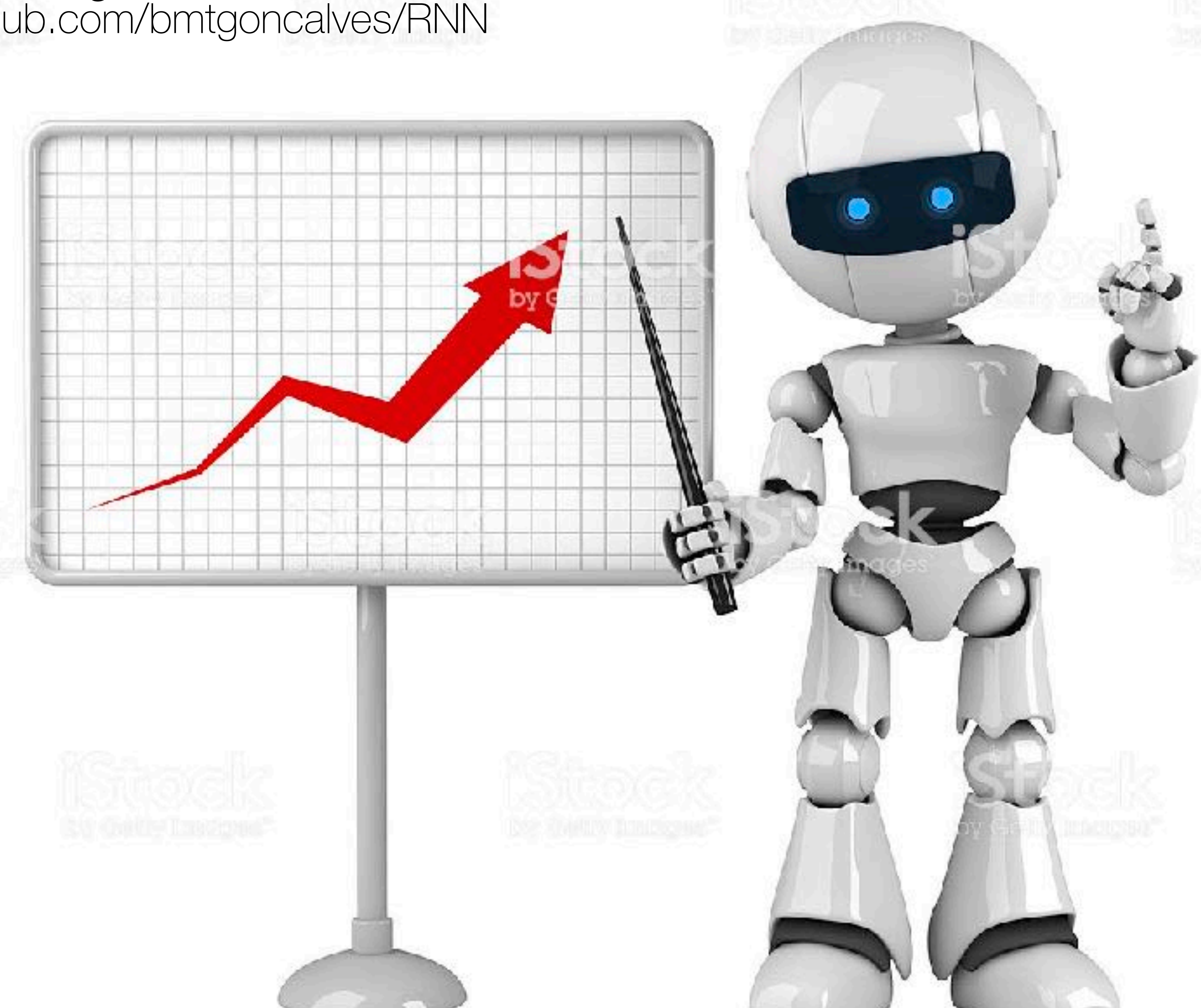
JPMorgan Chase & Co.

RNNs for Timeseries Analysis

Bruno Gonçalves

www.bgoncalves.com

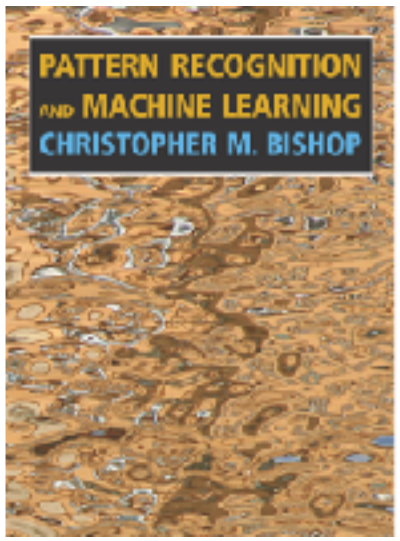
github.com/bmtgoncalves/RNN

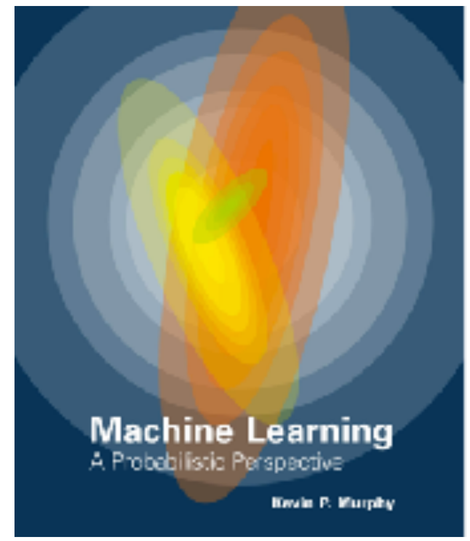


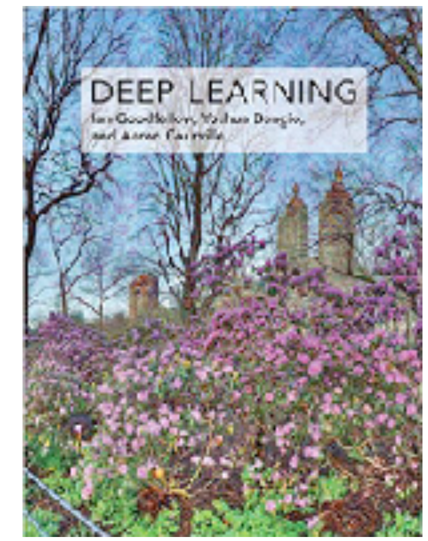
Disclaimer

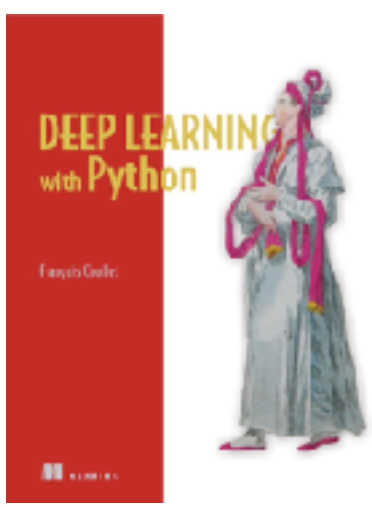
The views and opinions expressed in this article are those of the authors and do not necessarily reflect the official policy or position of my employer. The examples provided with this tutorial were chosen for their didactic value and are not mean to be representative of my day to day work.

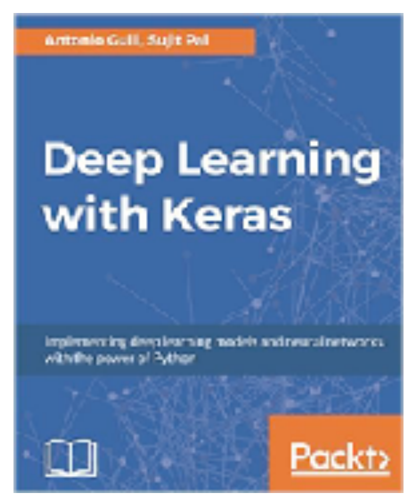
References

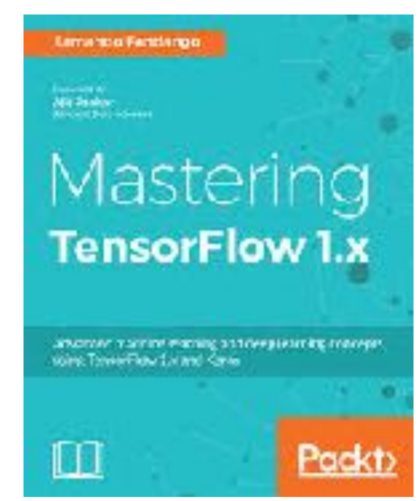

PATTERN RECOGNITION
AND MACHINE LEARNING
CHRISTOPHER M. BISHOP


Machine Learning
A Probabilistic Perspective
Kevin P. Murphy


DEEP LEARNING


DEEP LEARNING
with Python

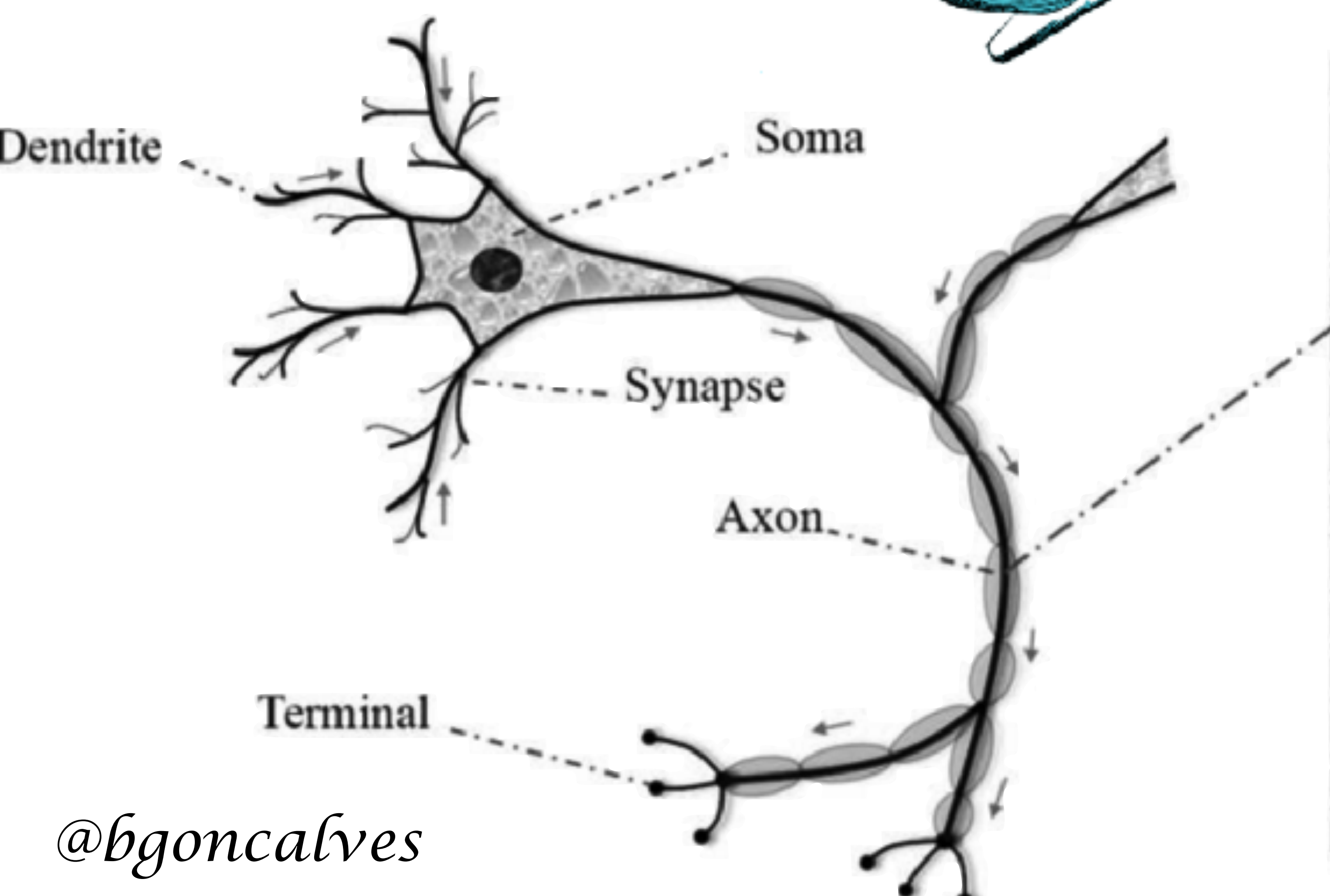

Antonio Gulli, Sujit Pal
Deep Learning with Keras
Packt>

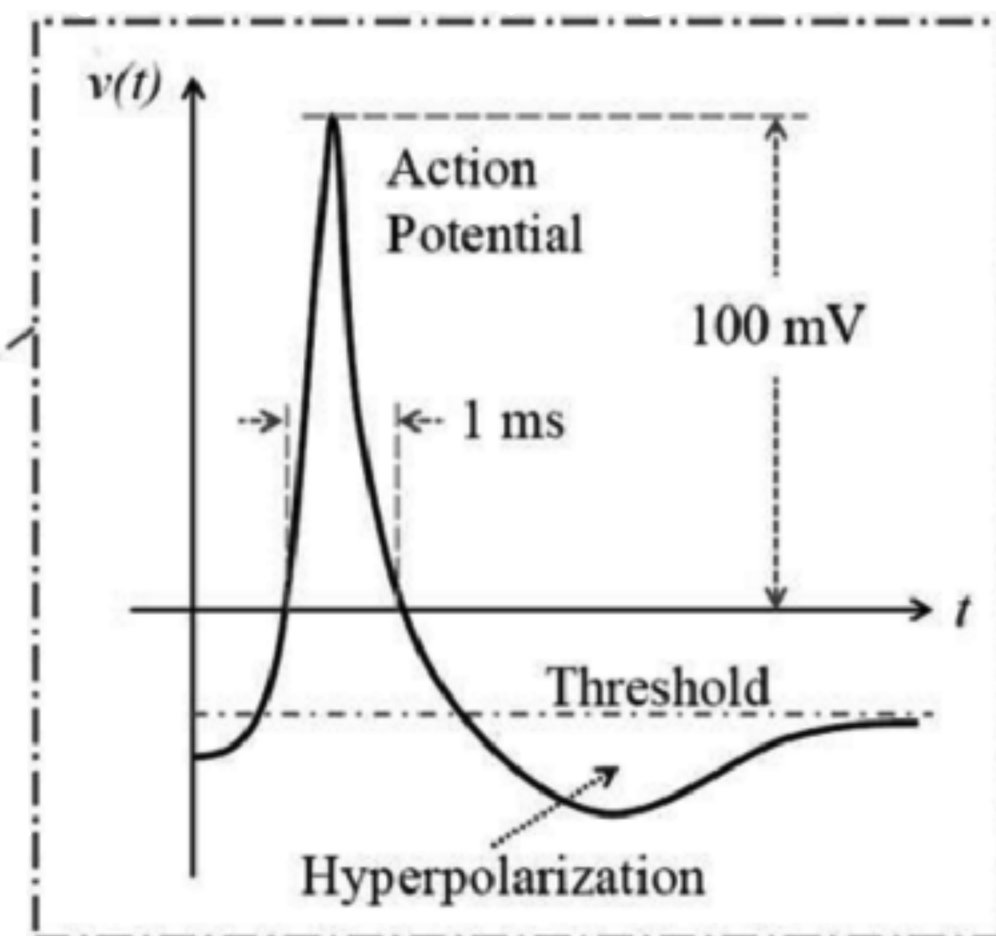

Mastering
TensorFlow 1.x
Packt>

How the Brain "Works" (Cartoon version)

How the Brain "Works" (Cartoon version)

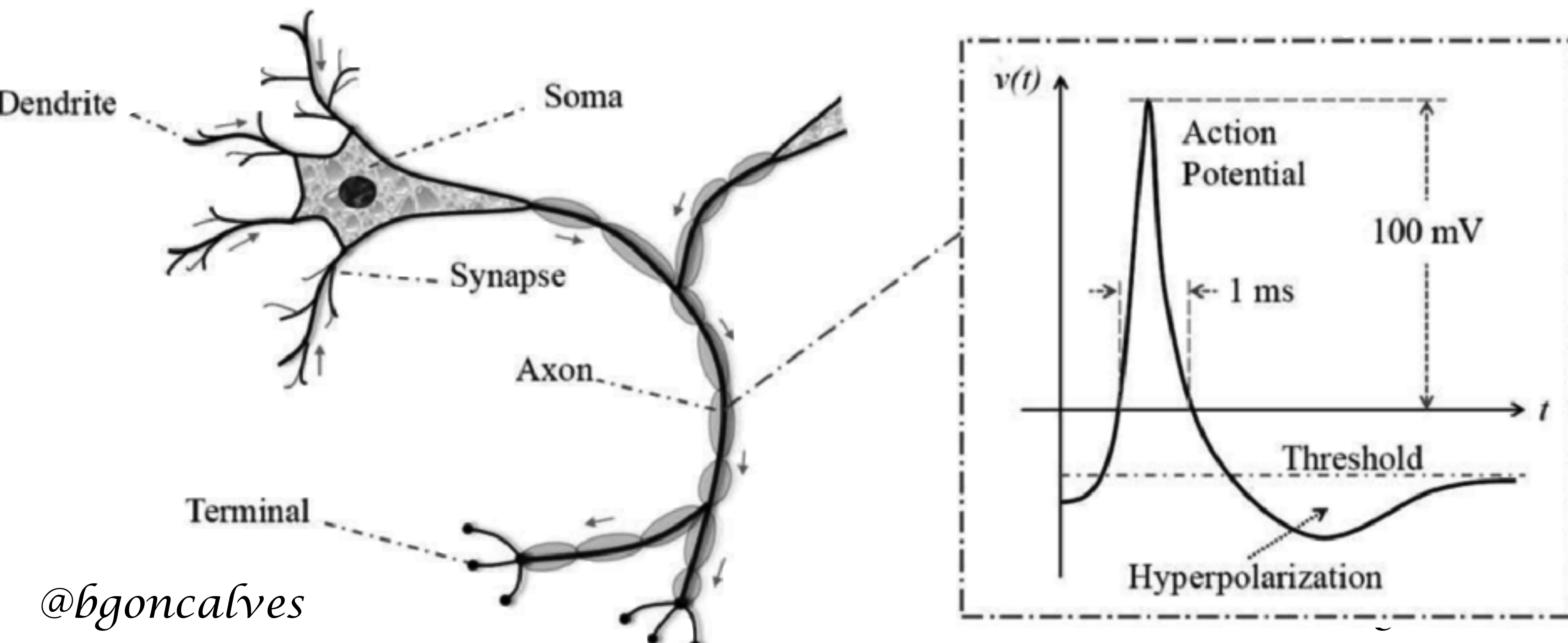
How the Brain "Works" (Cartoon version)





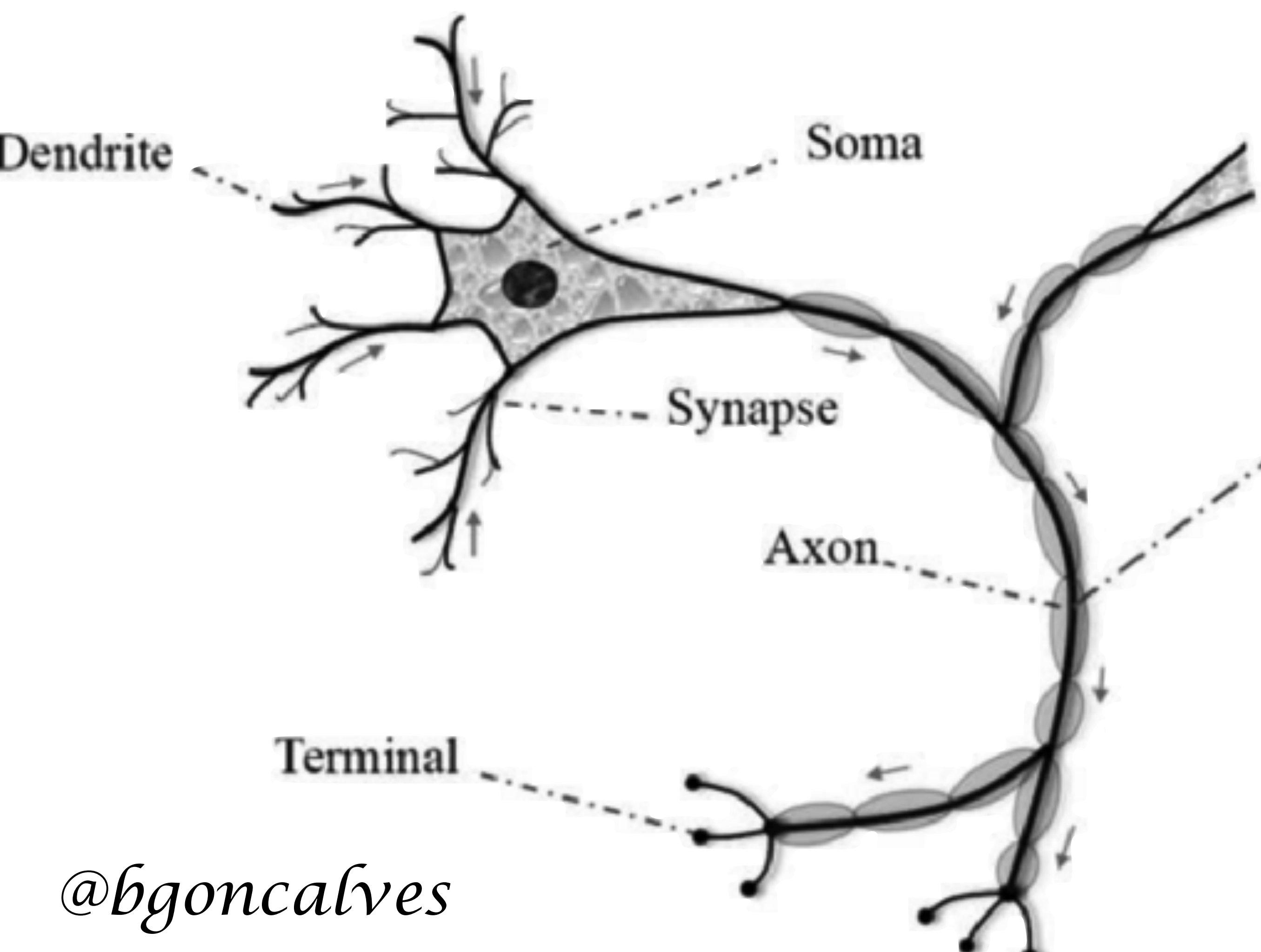
How the Brain “Works” (Cartoon version)

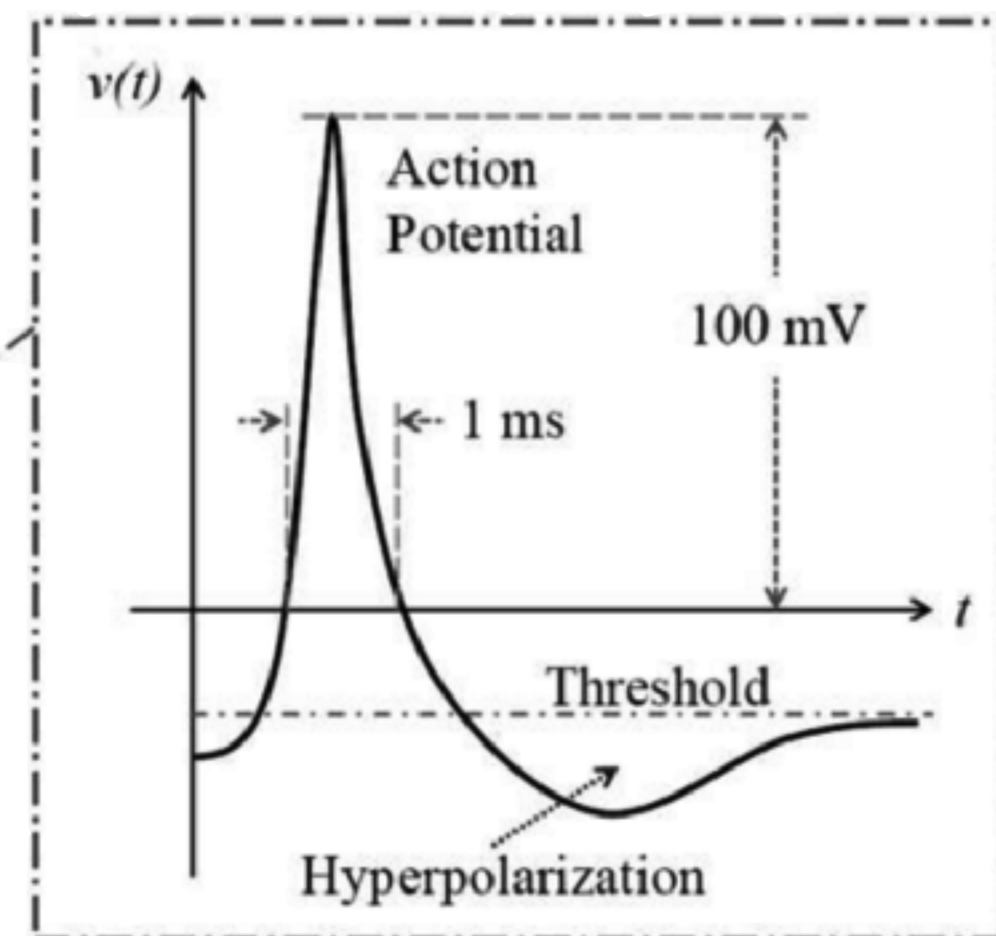
- Each neuron receives input from other neurons



How the Brain "Works" (Cartoon version)

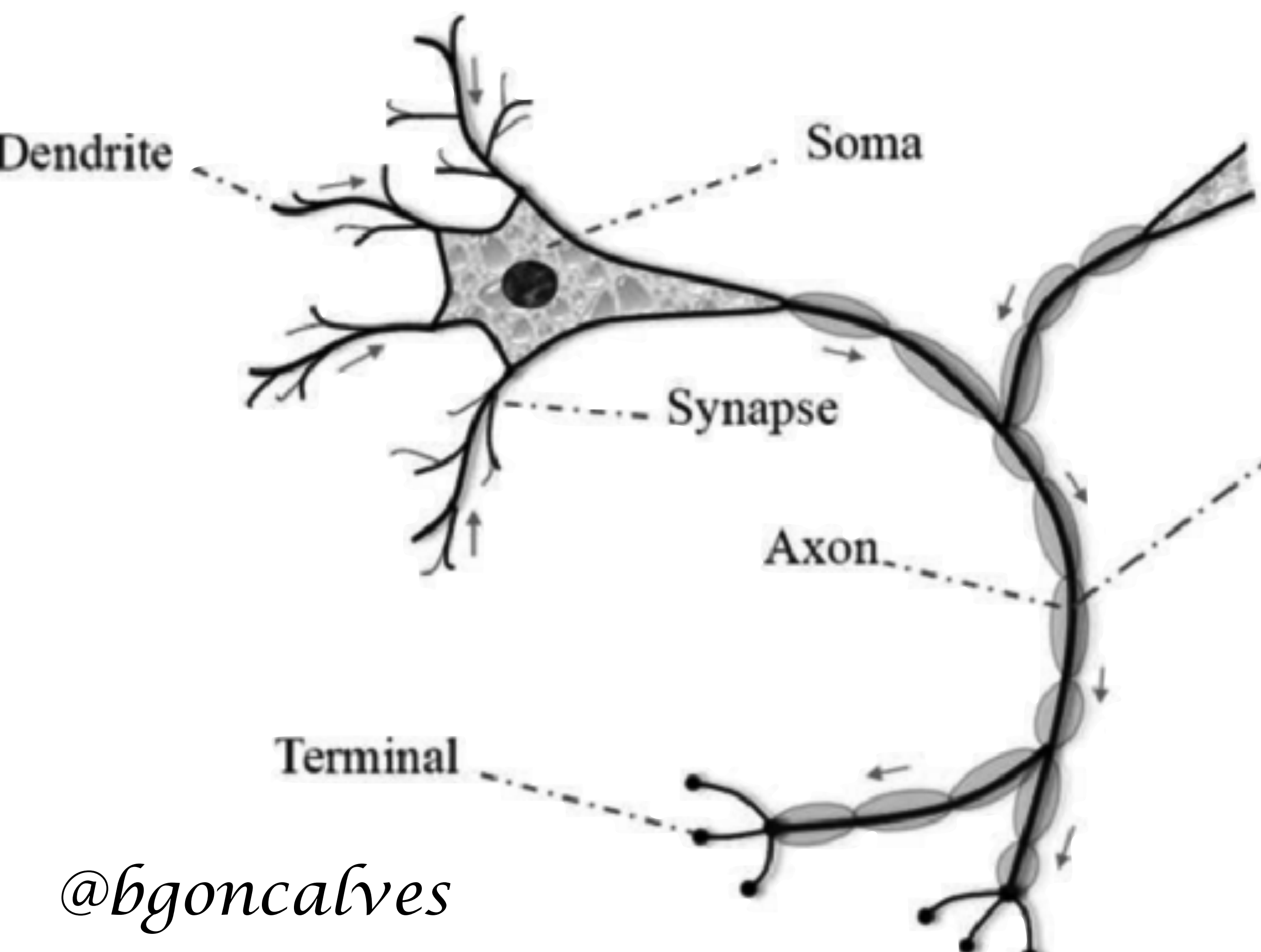
- Each neuron receives input from other neurons
- 10^{11} neurons, each with with 10^{4} weights

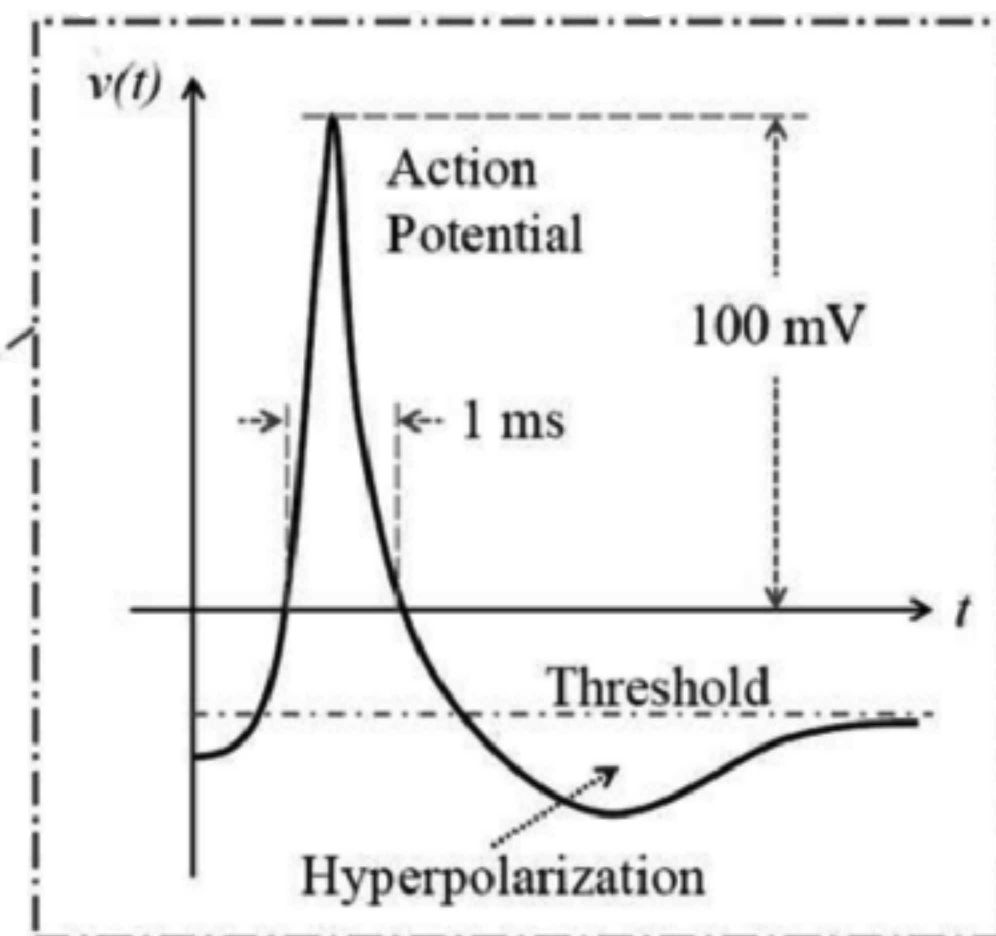




How the Brain "Works" (Cartoon version)

- Each neuron receives input from other neurons
- 10^{11} neurons, each with with 10^{4} weights
- Weights can be positive or negative

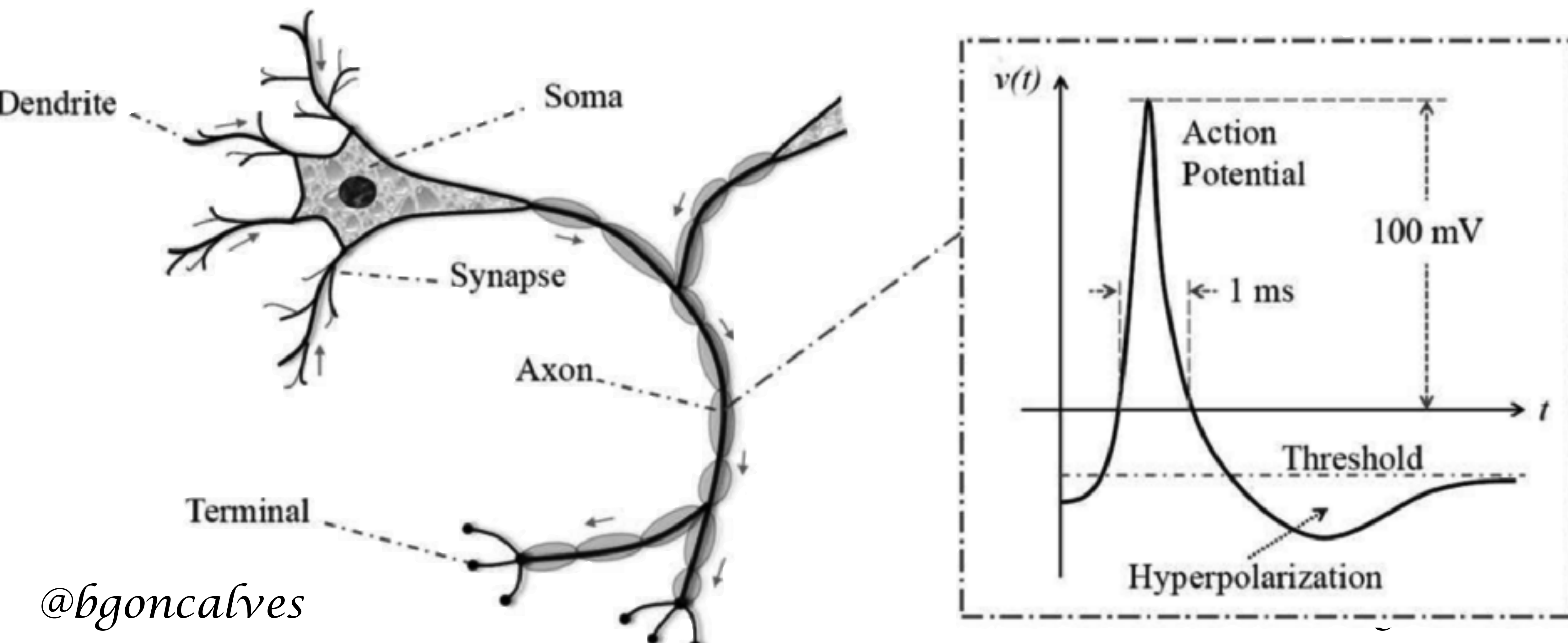




@bgoncalves

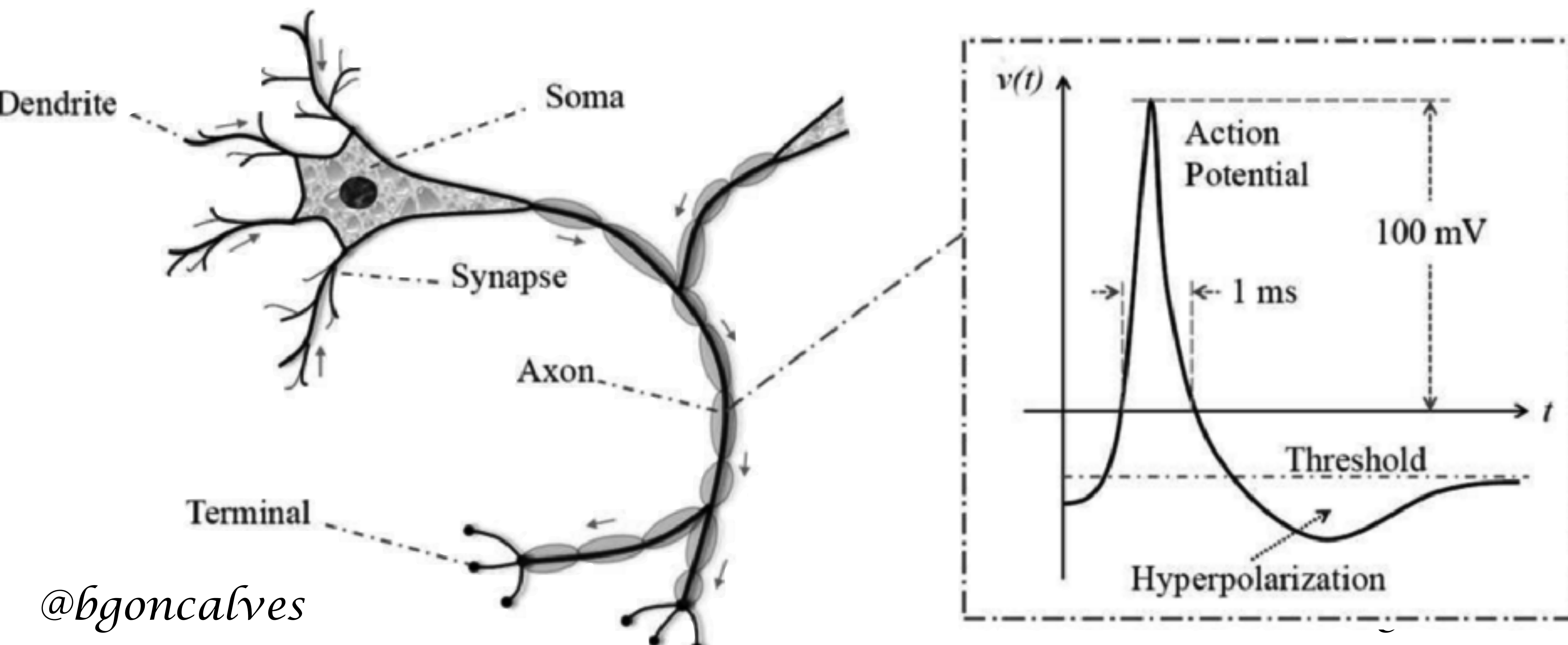
How the Brain “Works” (Cartoon version)

- Each neuron receives input from other neurons
- 10^{11} neurons, each with with 10^4 weights
- Weights can be positive or negative
- Weights adapt during the learning process



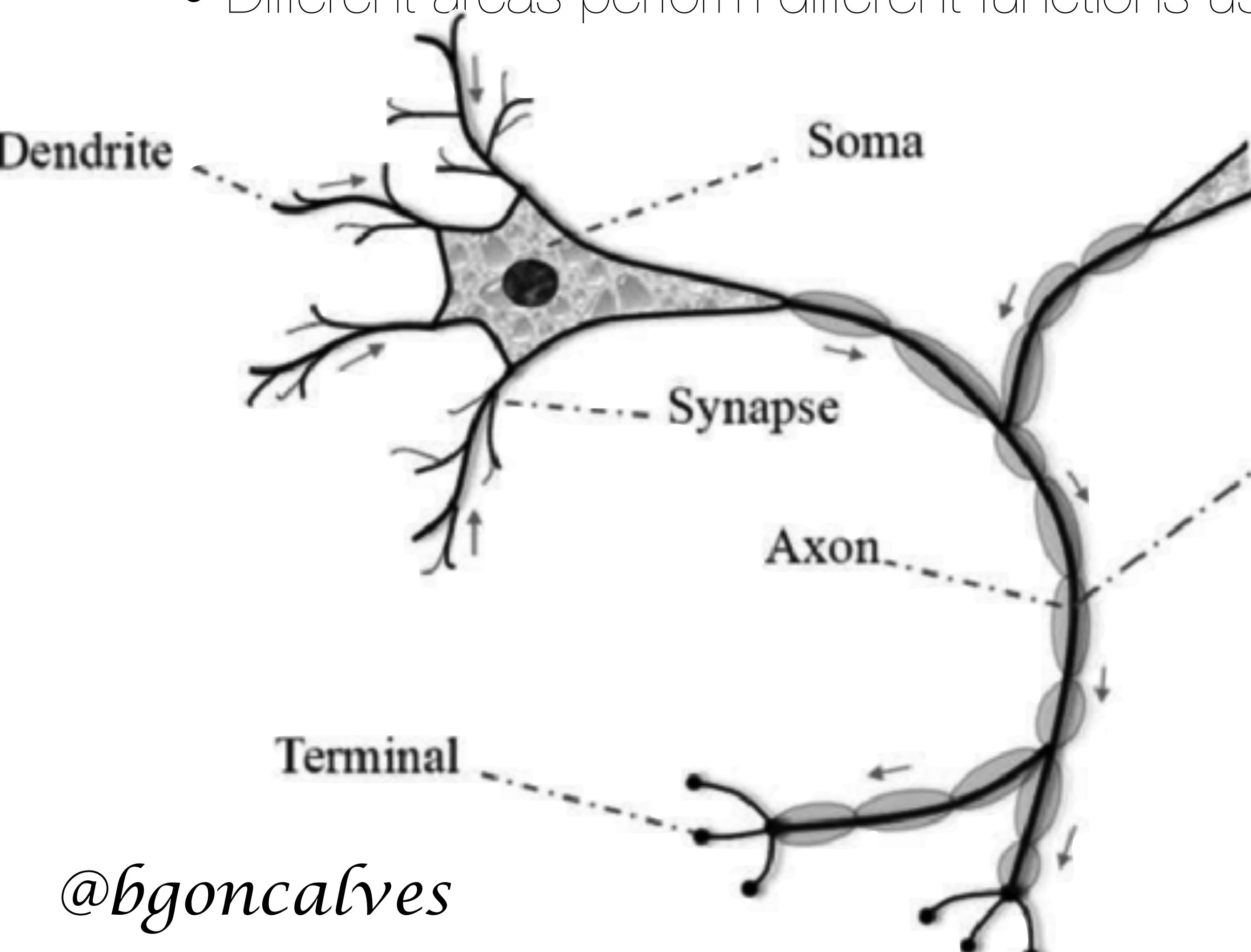
How the Brain "Works" (Cartoon version)

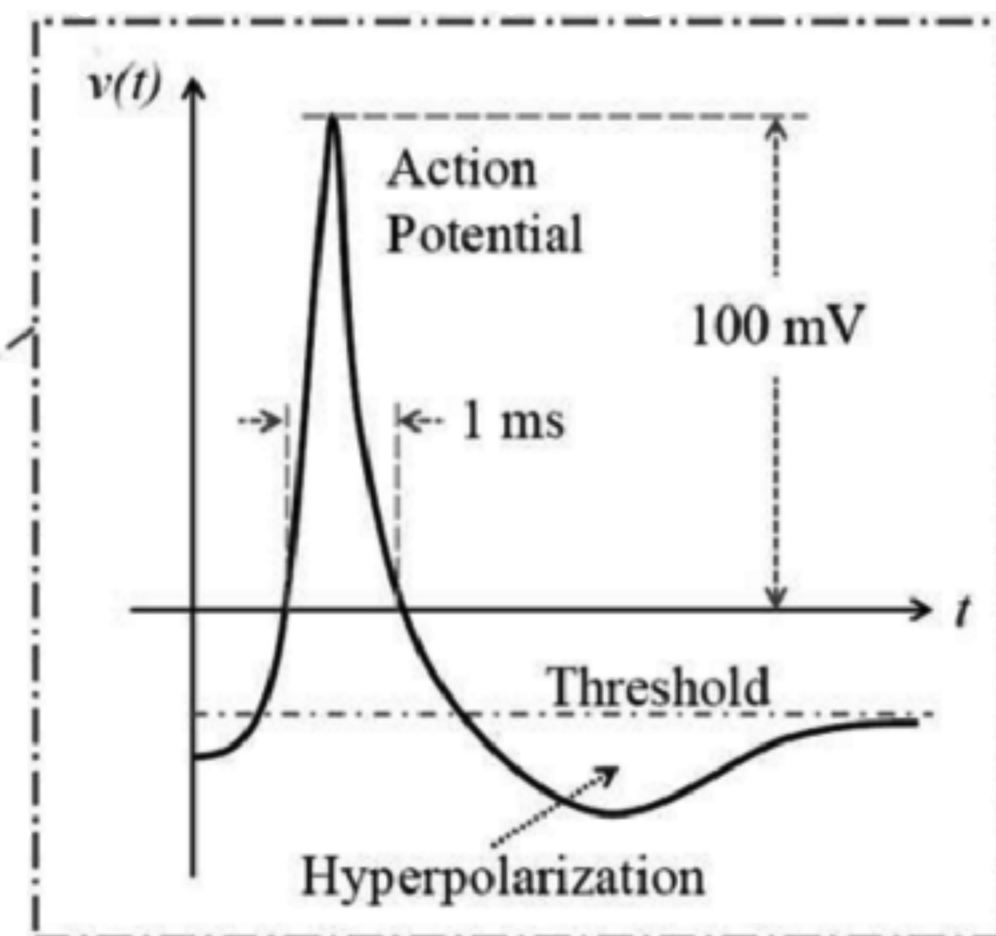
- Each neuron receives input from other neurons
- 10^{11} neurons, each with with 10^{4} weights
- Weights can be positive or negative
- Weights adapt during the learning process
- "neurons that fire together wire together" (Hebb)



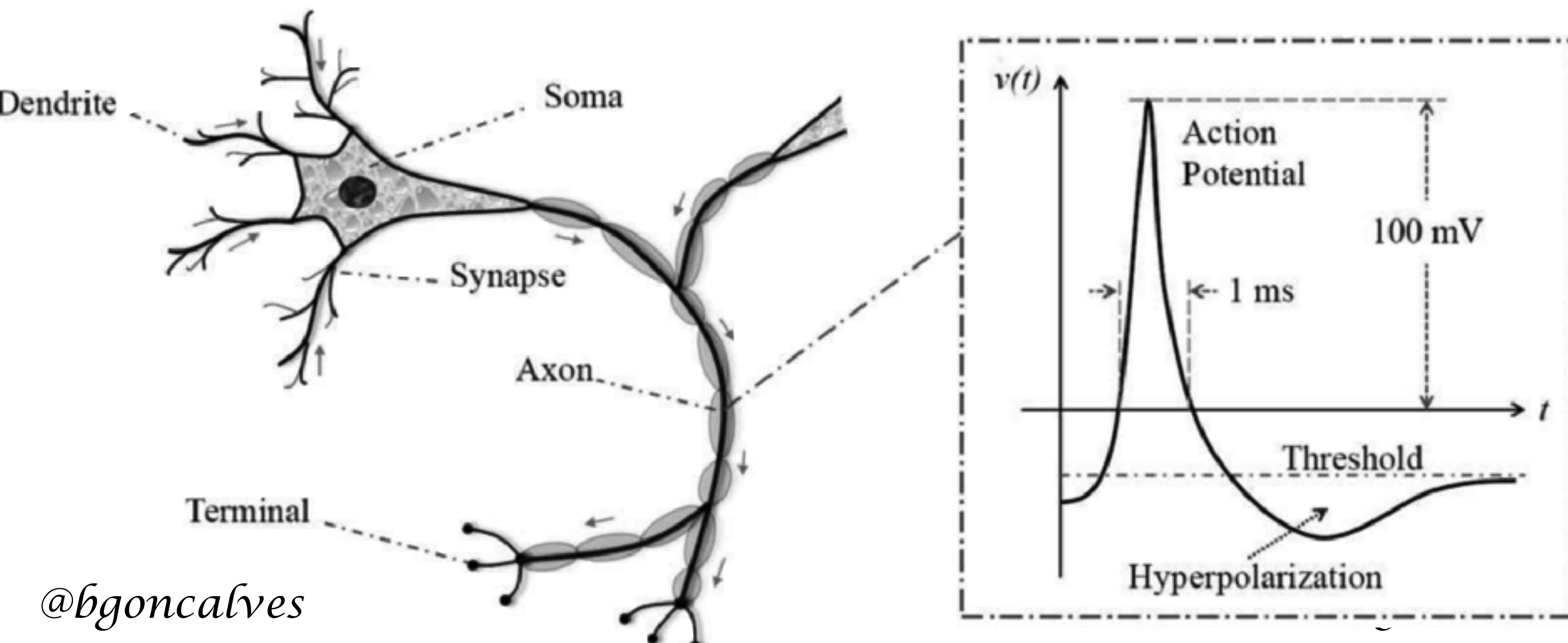
How the Brain "Works" (Cartoon version)

- Each neuron receives input from other neurons
- 10^{11} neurons, each with 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)



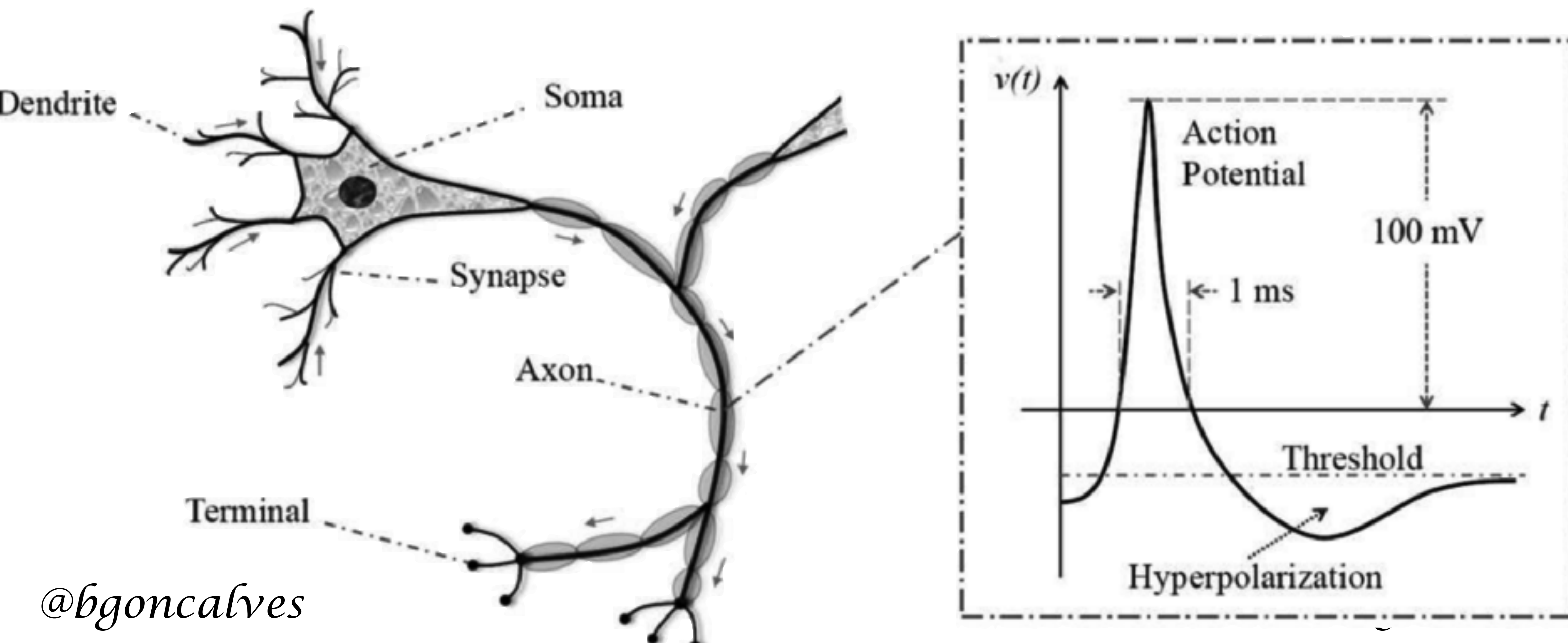


How the Brain "Works" (Cartoon version)

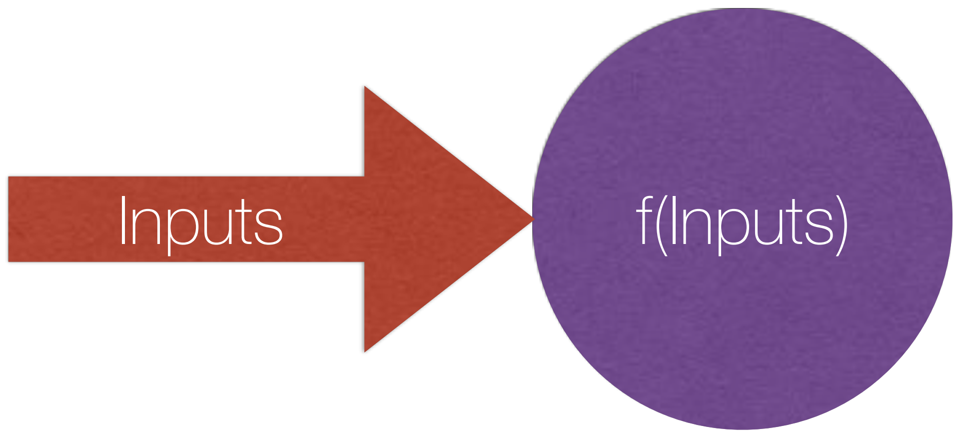


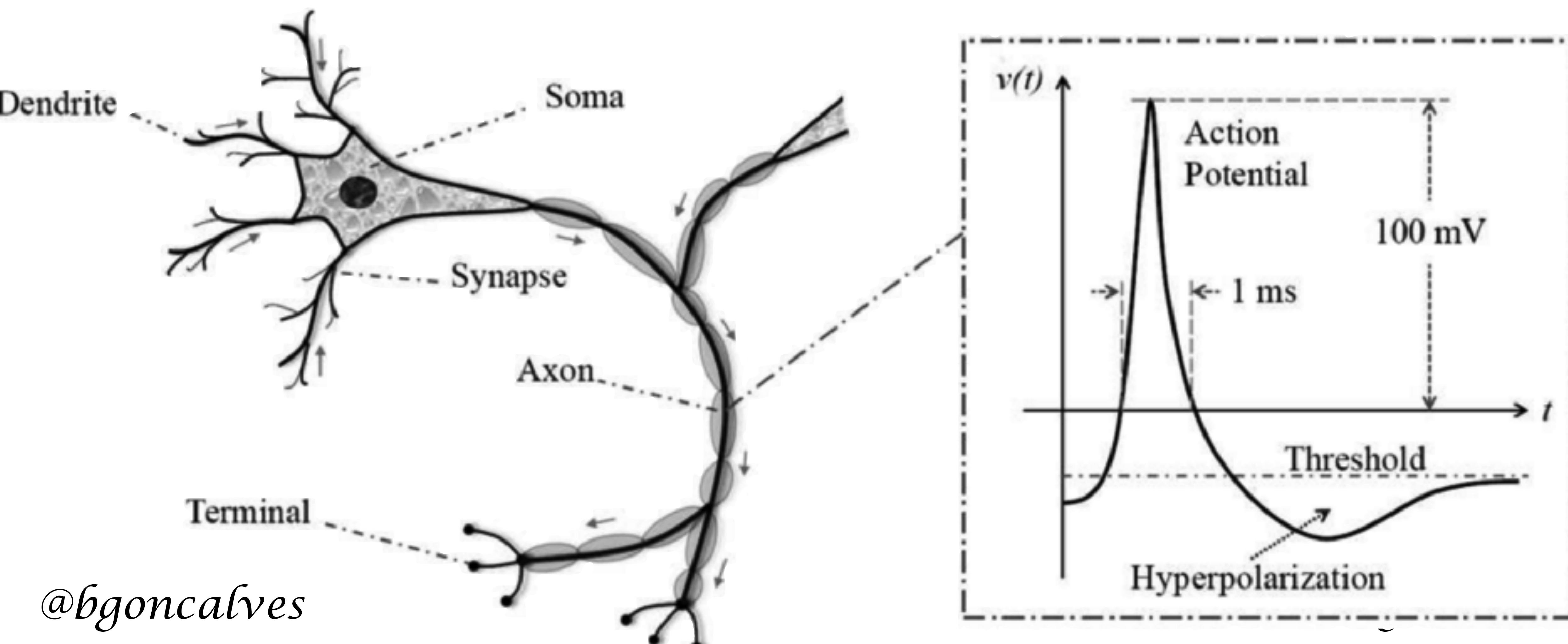
How the Brain "Works" (Cartoon version)

Inputs

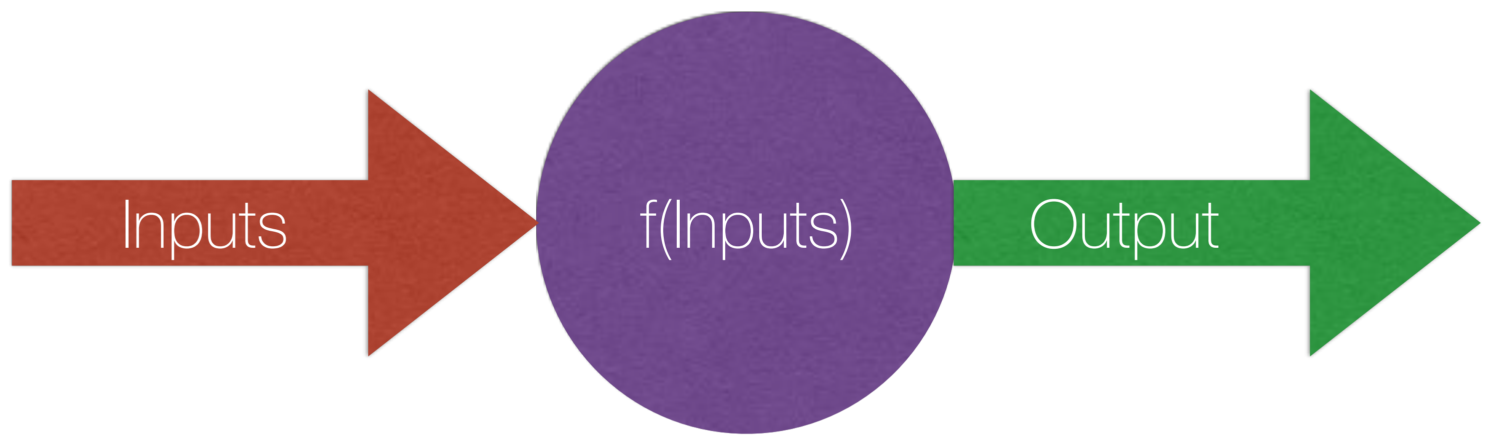


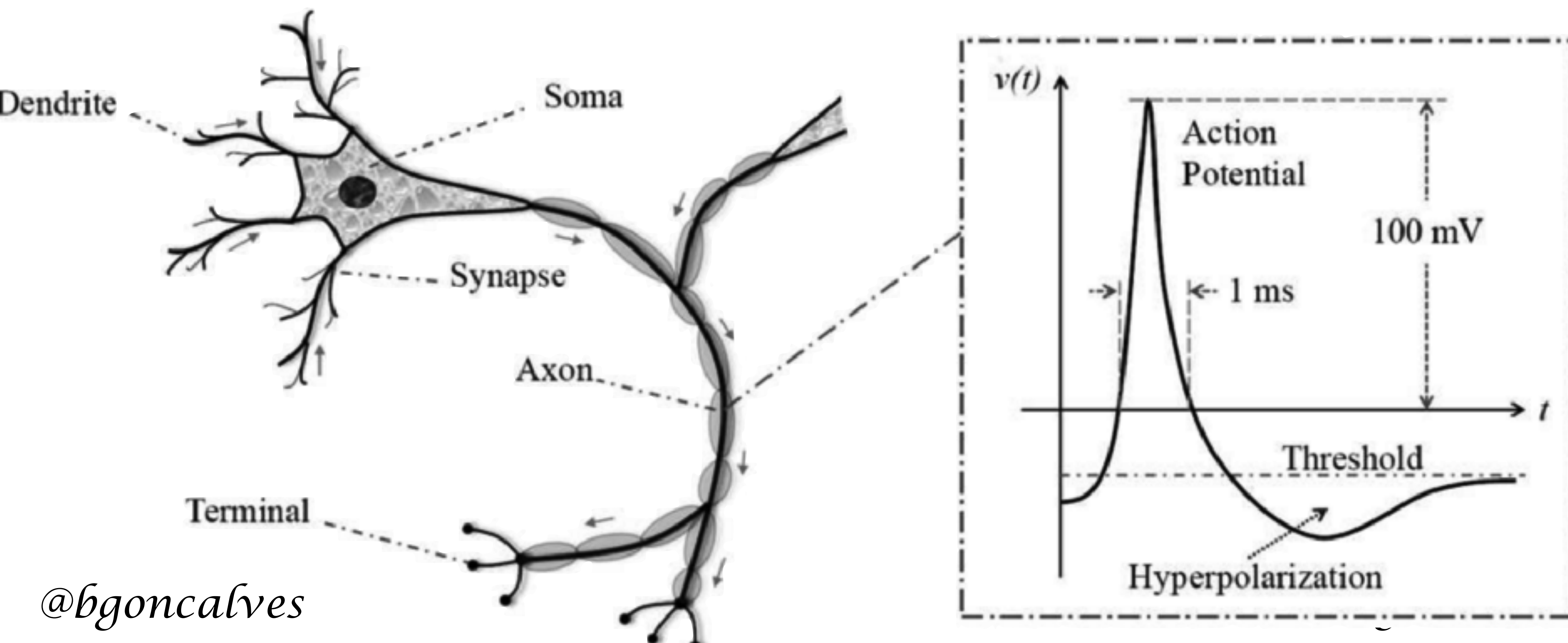
How the Brain "Works" (Cartoon version)





How the Brain "Works" (Cartoon version)





Optimization Problem

Optimization Problem

- (Machine) Learning can be thought of as an optimization problem.

Optimization Problem

- (Machine) Learning can be thought of as an optimization problem.
- Optimization Problems have 3 distinct pieces:

Optimization Problem

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

Optimization Problem

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

Optimization Problem

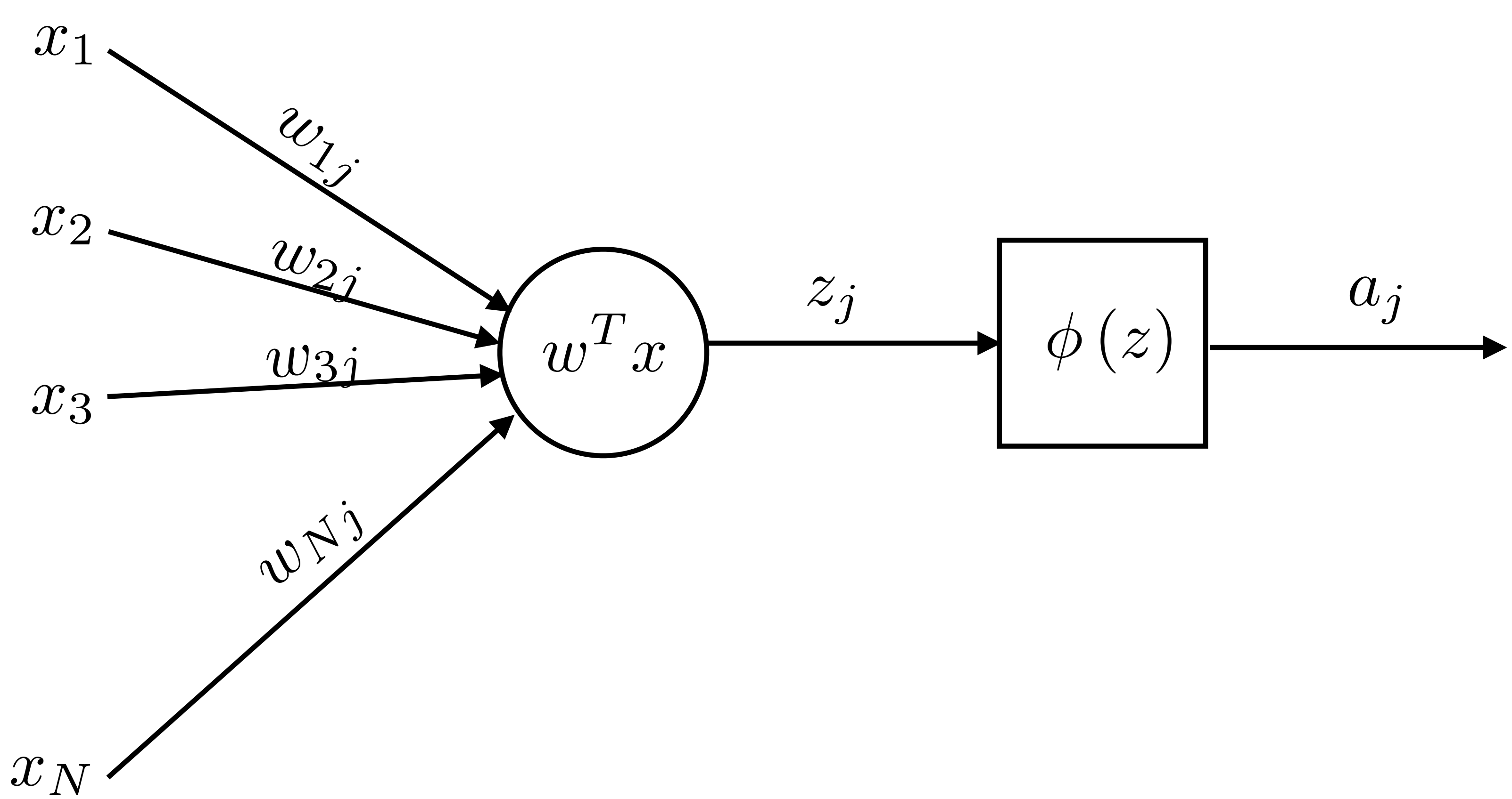
- (Machine) Learning can be thought of as an optimization problem.
- Optimization Problems have 3 distinct pieces:
    - The constraints
    - The function to optimize
    - The optimization algorithm.

Optimization Problem

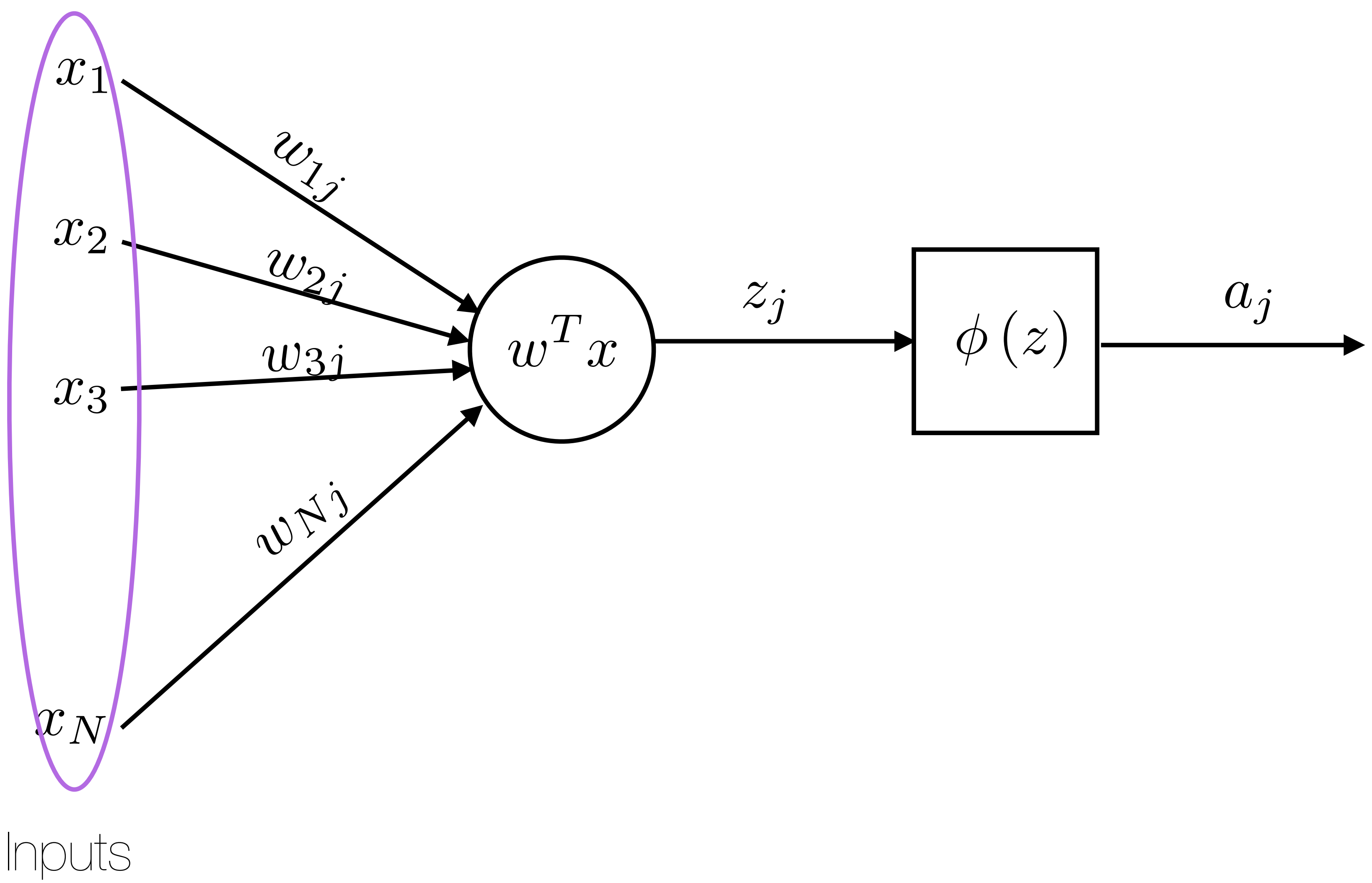
- (Machine) Learning can be thought of as an optimization problem.
- Optimization Problems have 3 distinct pieces:
  - The constraints — Neural Network
  - The function to optimize — Prediction Error
  - The optimization algorithm. — Gradient Descent

Artificial Neuron

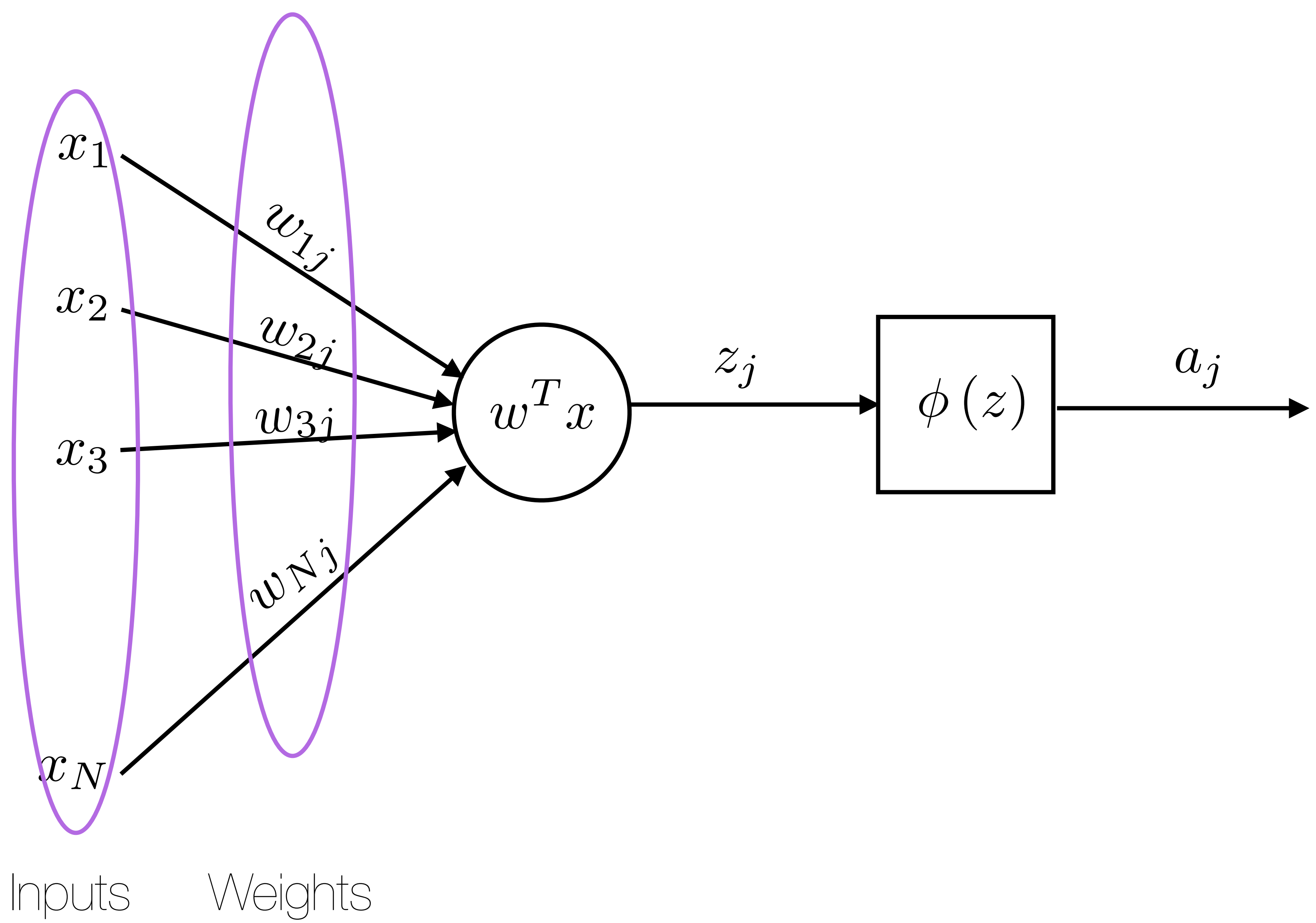
Artificial Neuron



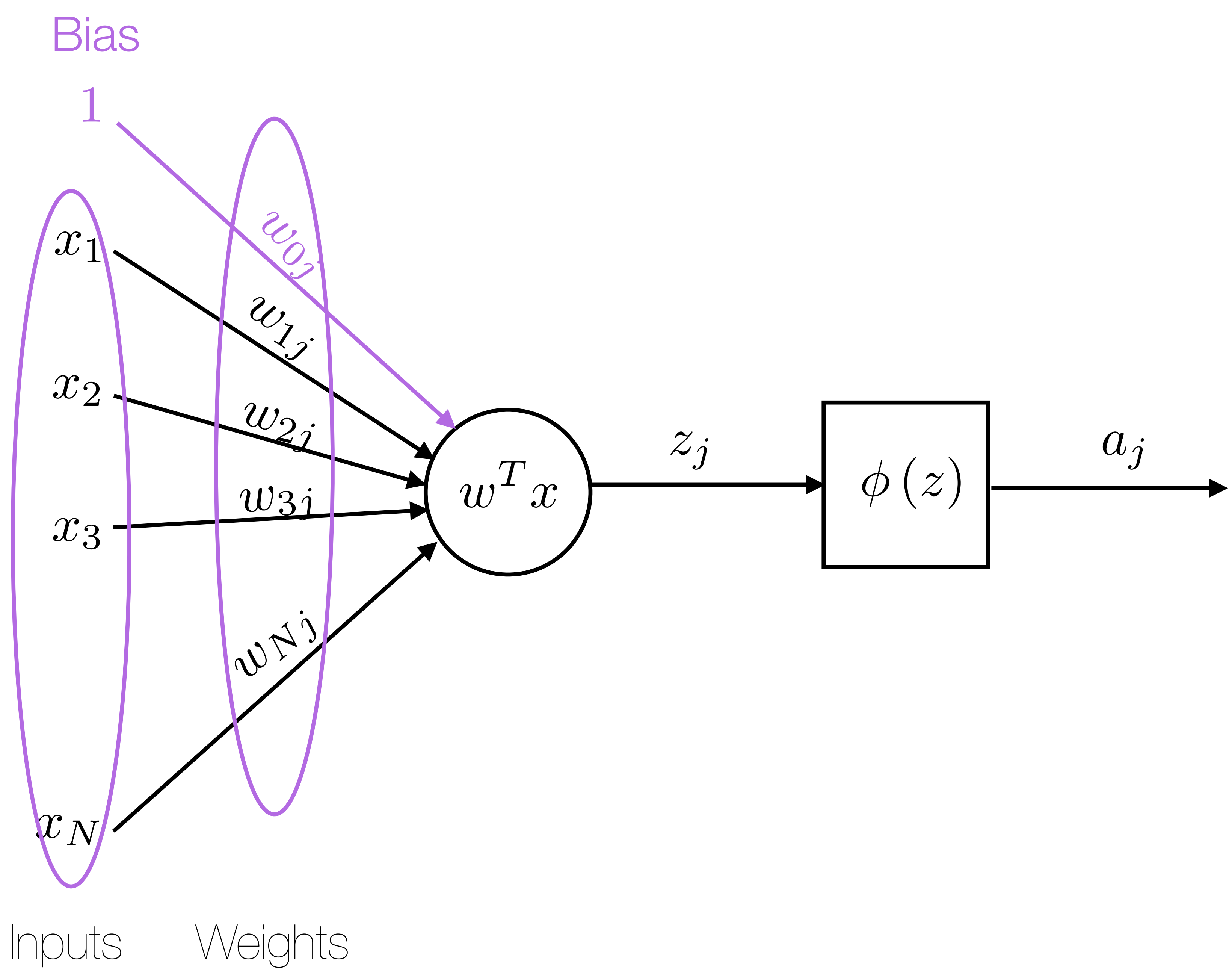
Artificial Neuron



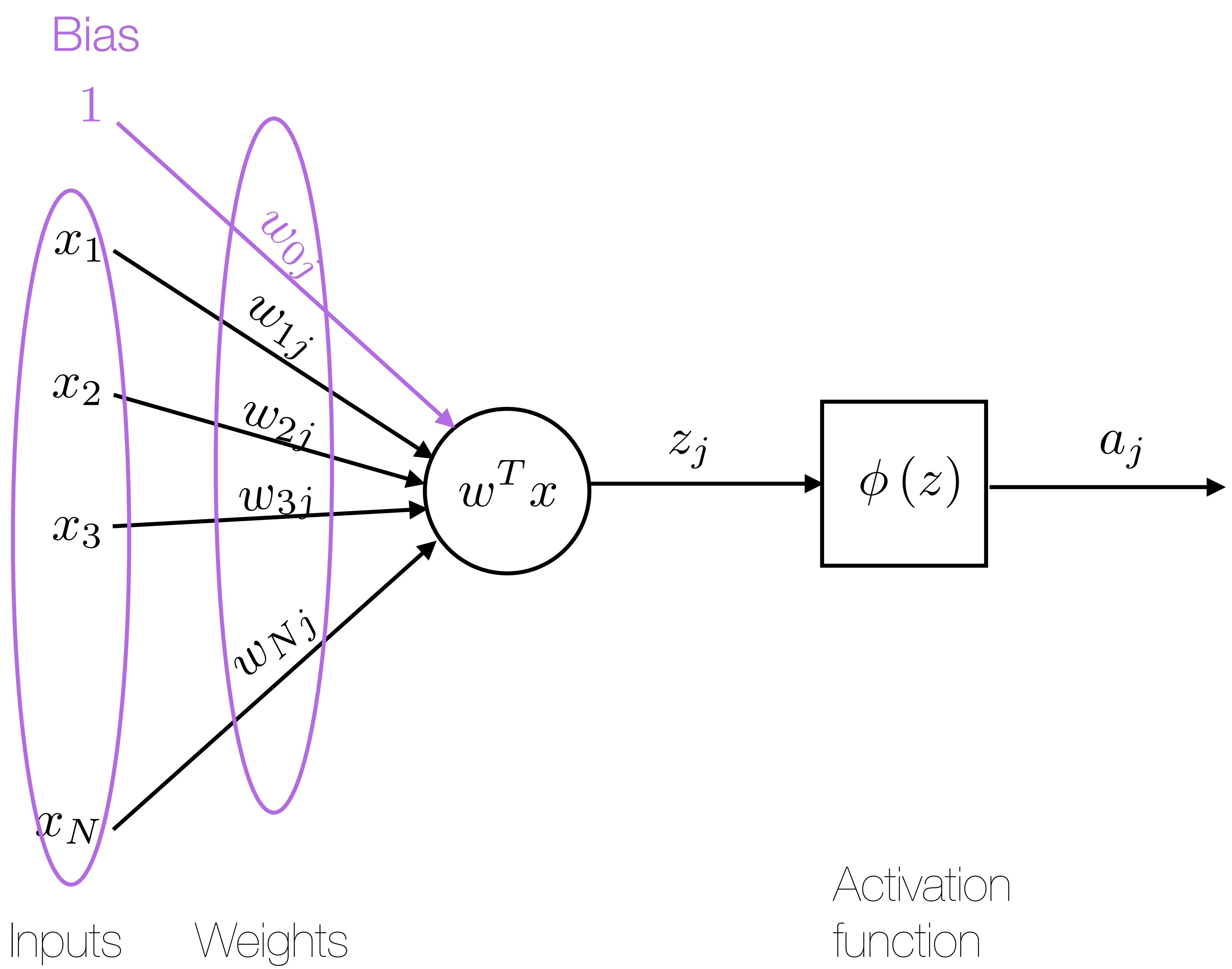
Artificial Neuron



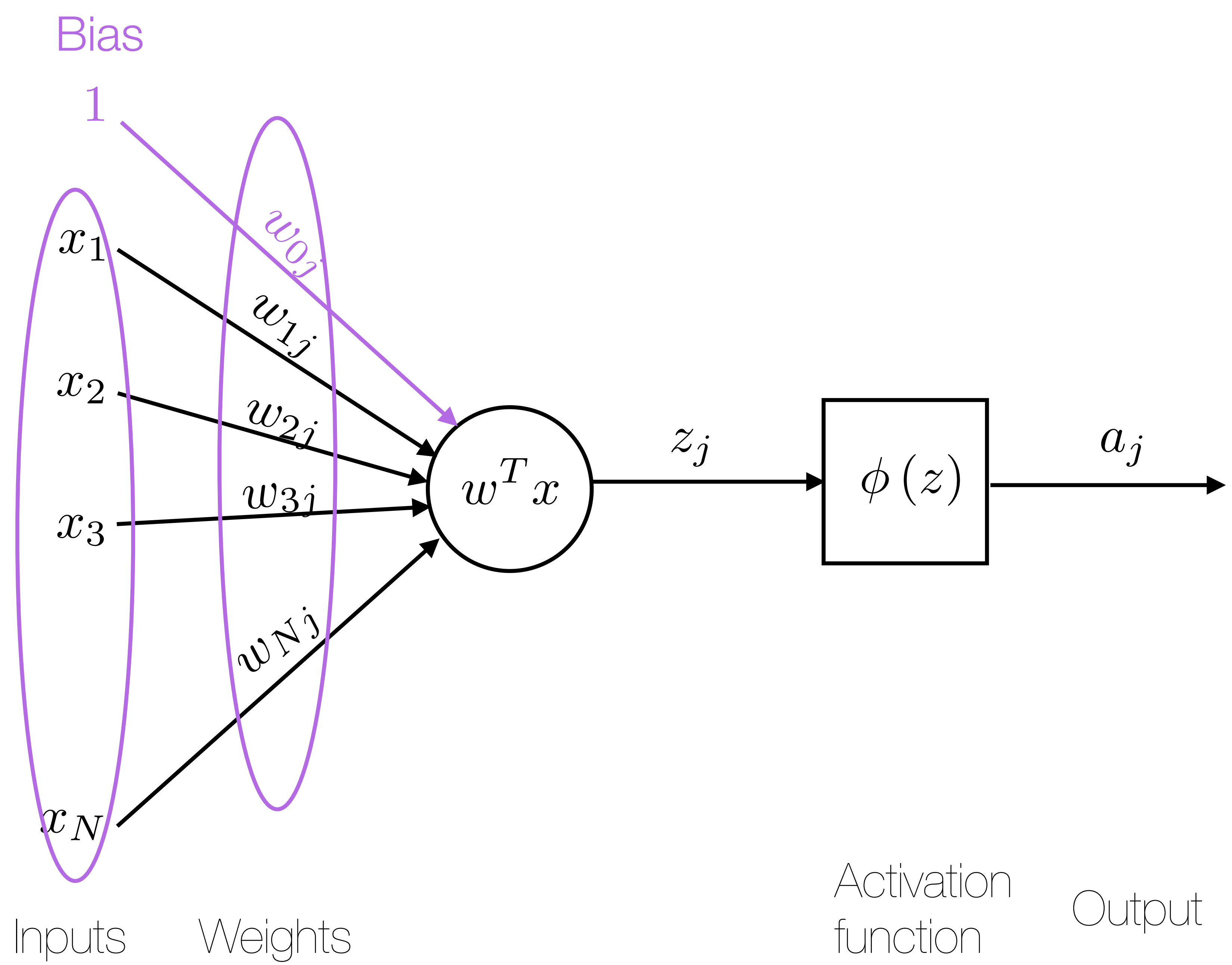
Artificial Neuron



Artificial Neuron

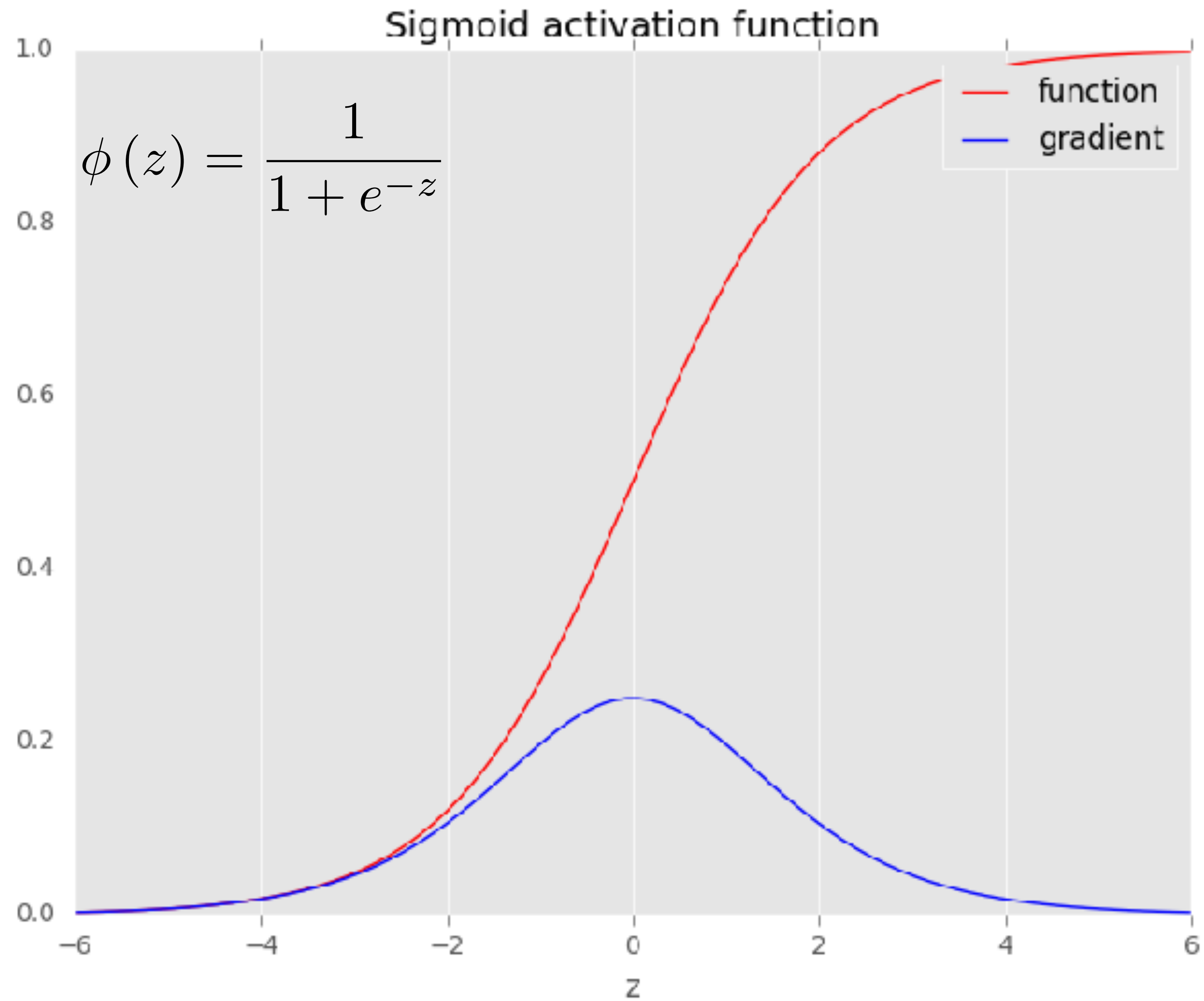


Artificial Neuron



Activation Function - Sigmoid

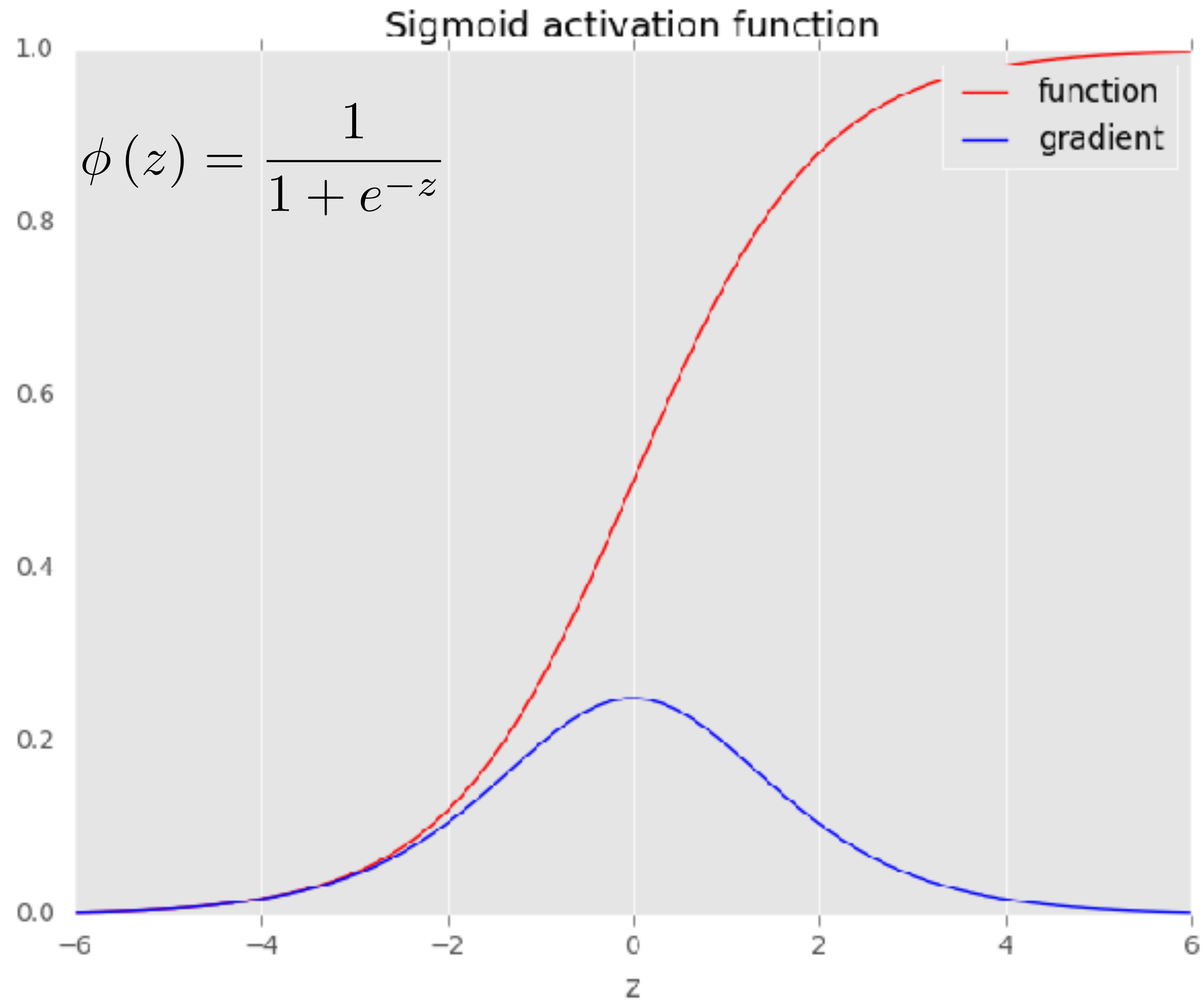
http://github.com/bmtgoncalves/Neural-Networks



Activation Function - Sigmoid

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

- Non-Linear function

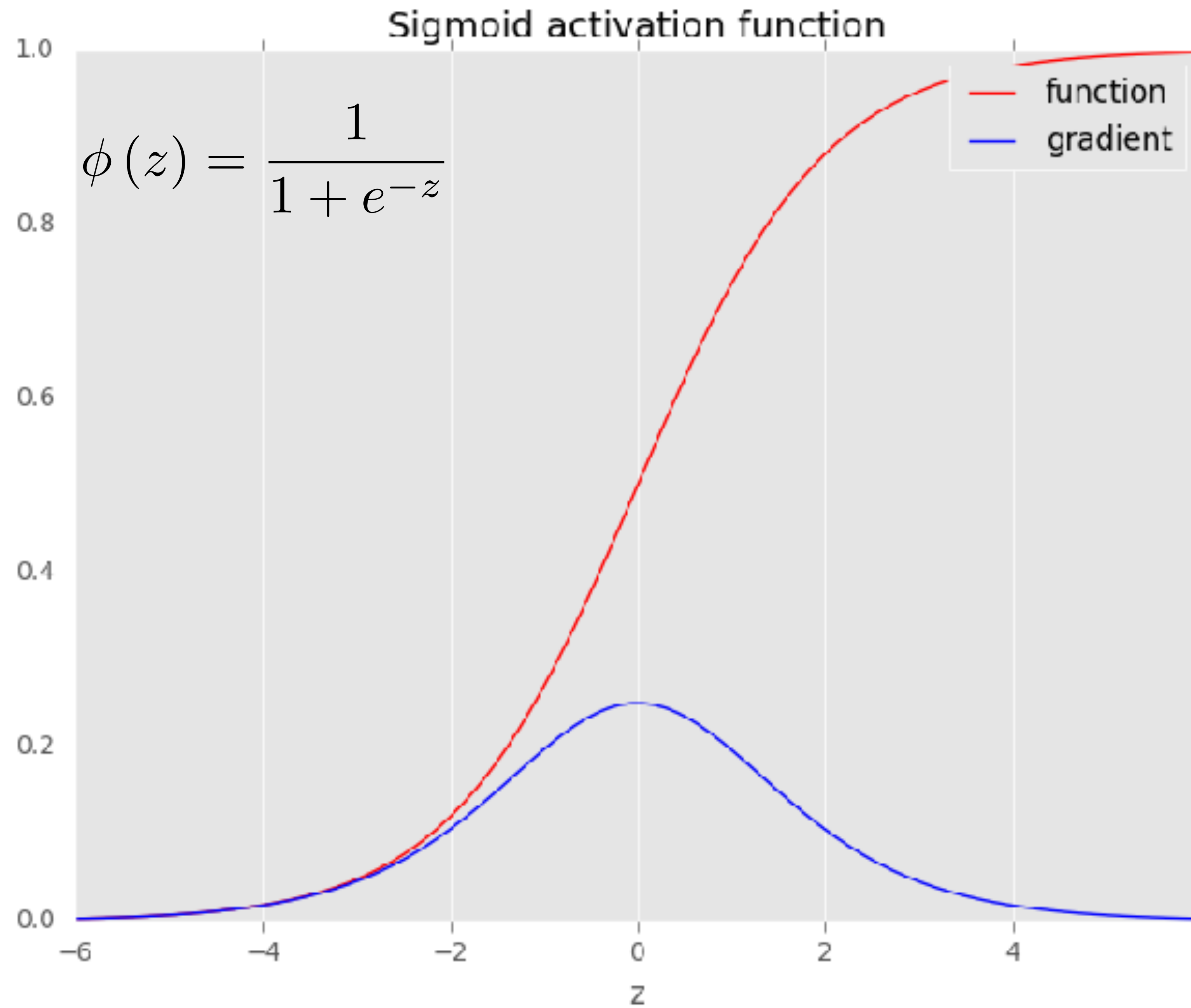


Activation Function - Sigmoid

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

- Non-Linear function

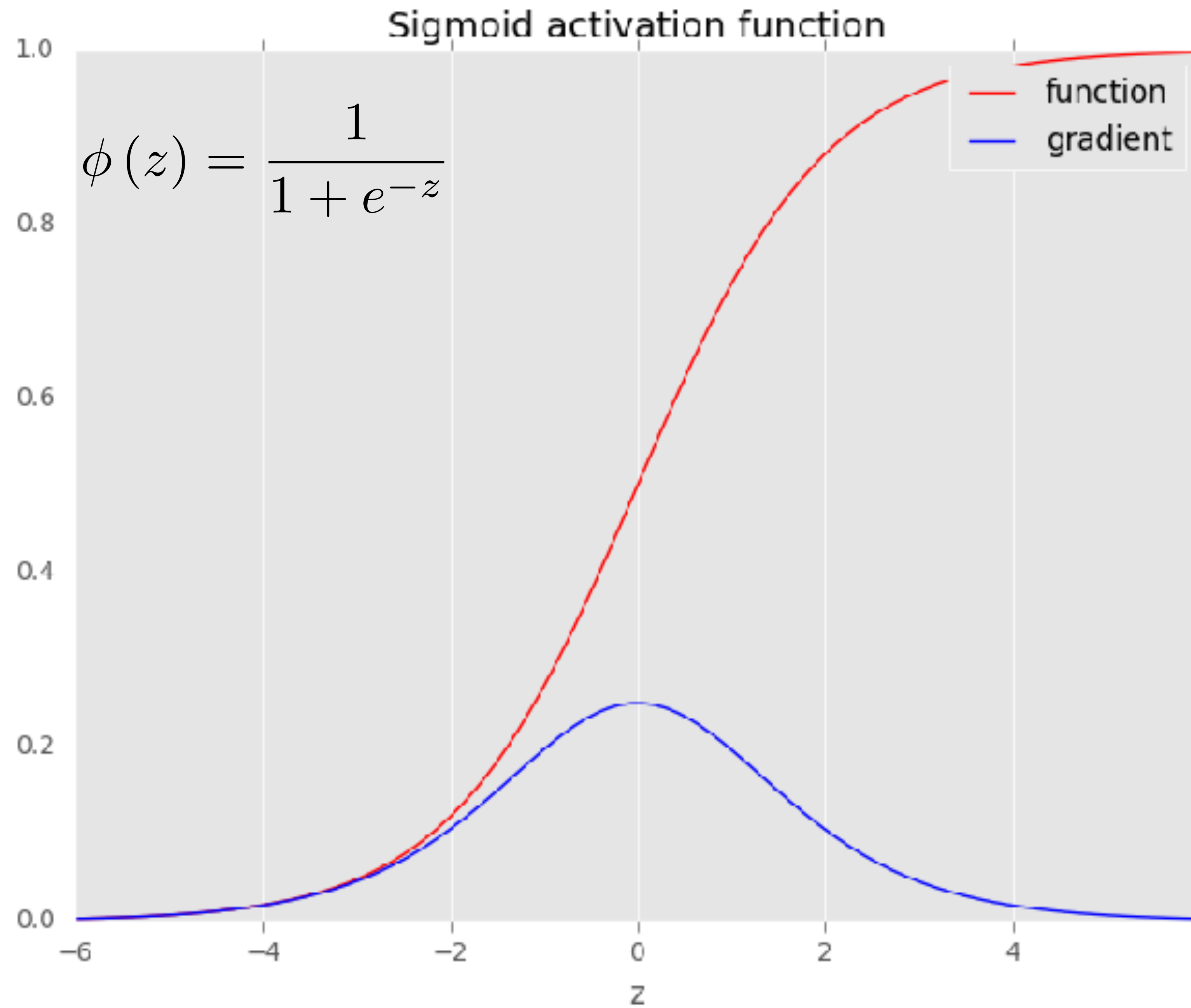
- Differentiable



Activation Function - Sigmoid

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

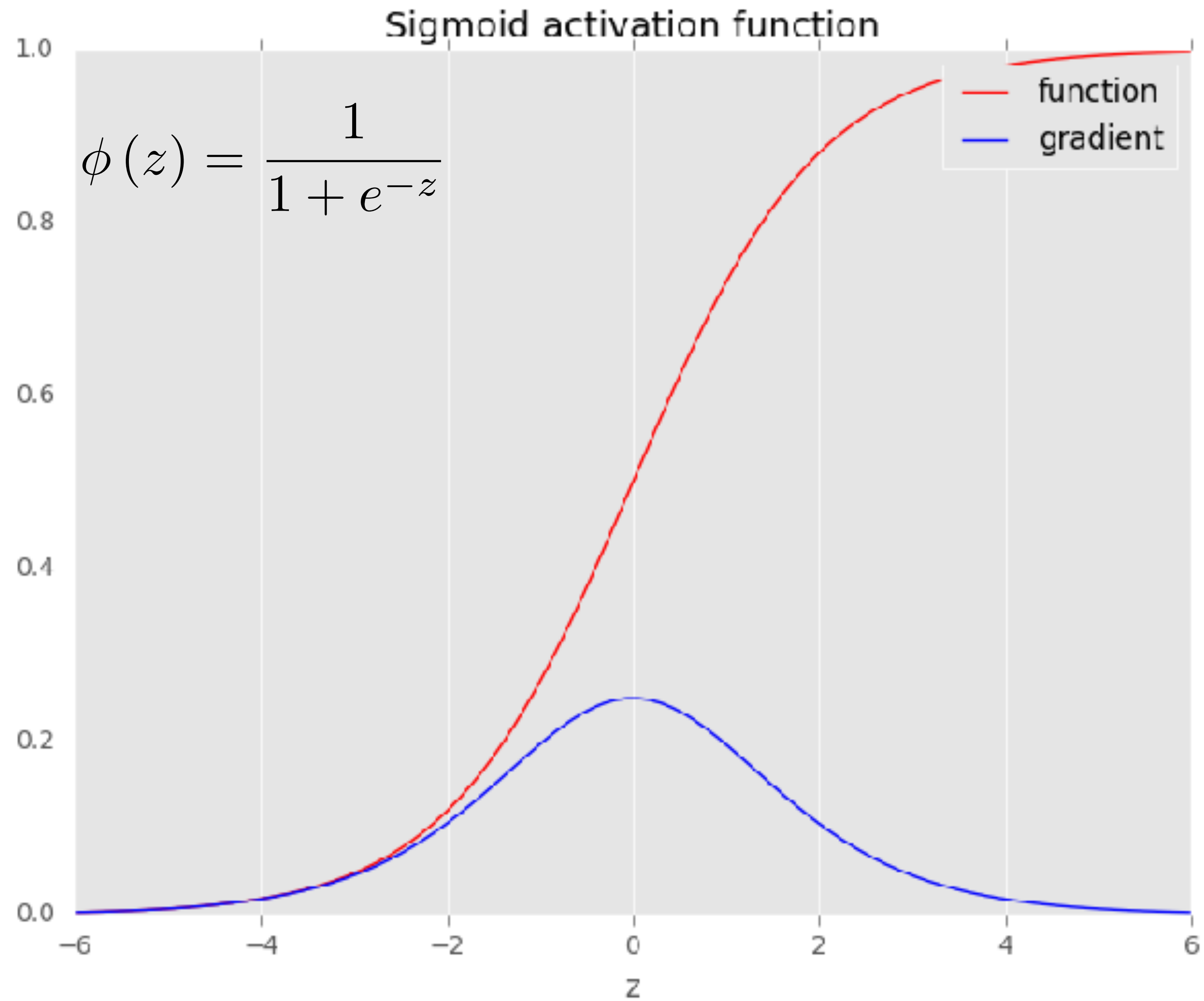
- Non-Linear function
- Differentiable
- non-decreasing



Activation Function - Sigmoid

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

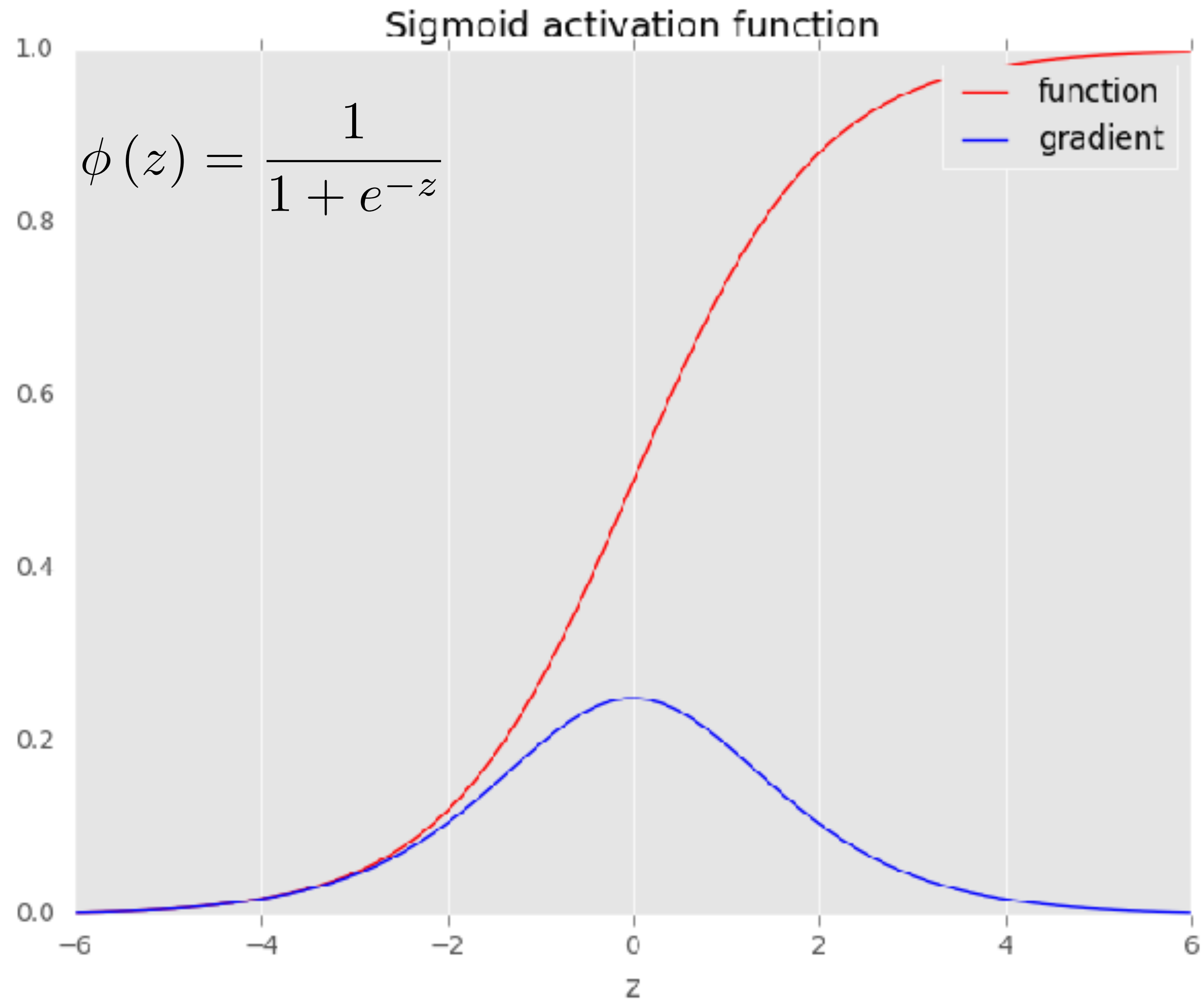
- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features



Activation Function - Sigmoid

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

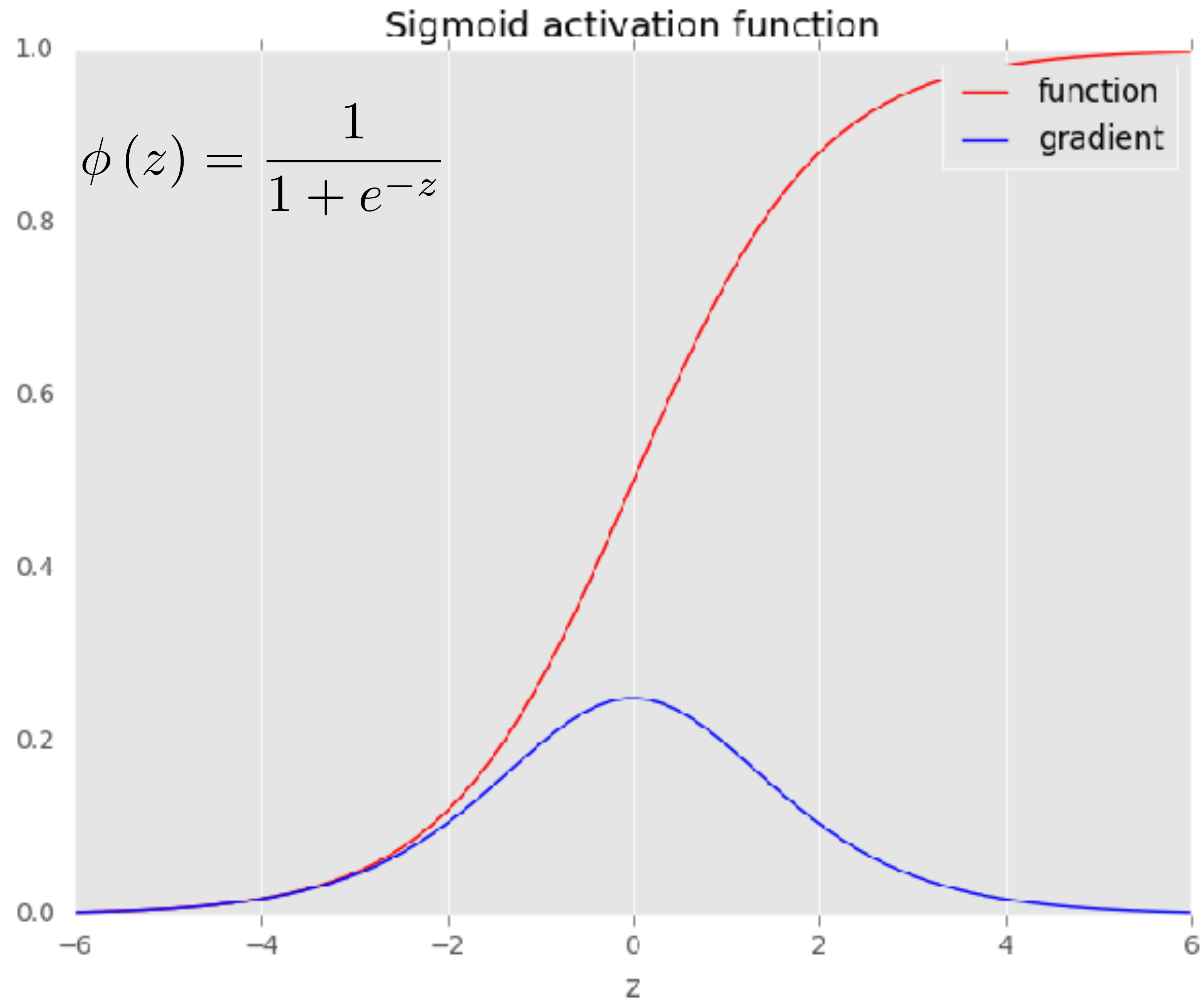
- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data



Activation Function - Sigmoid

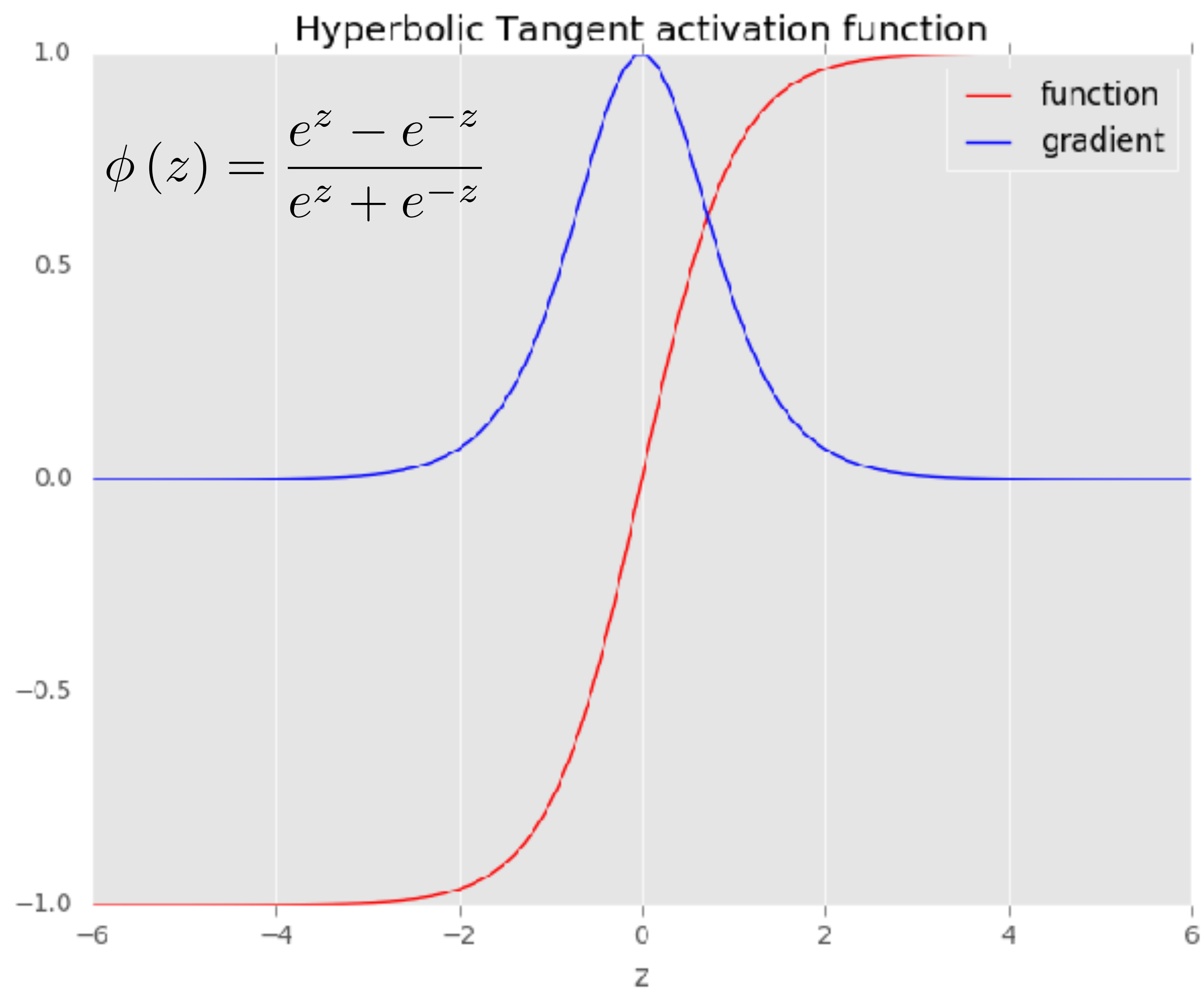
http://github.com/bmtgoncalves/Neural-Networks

- 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

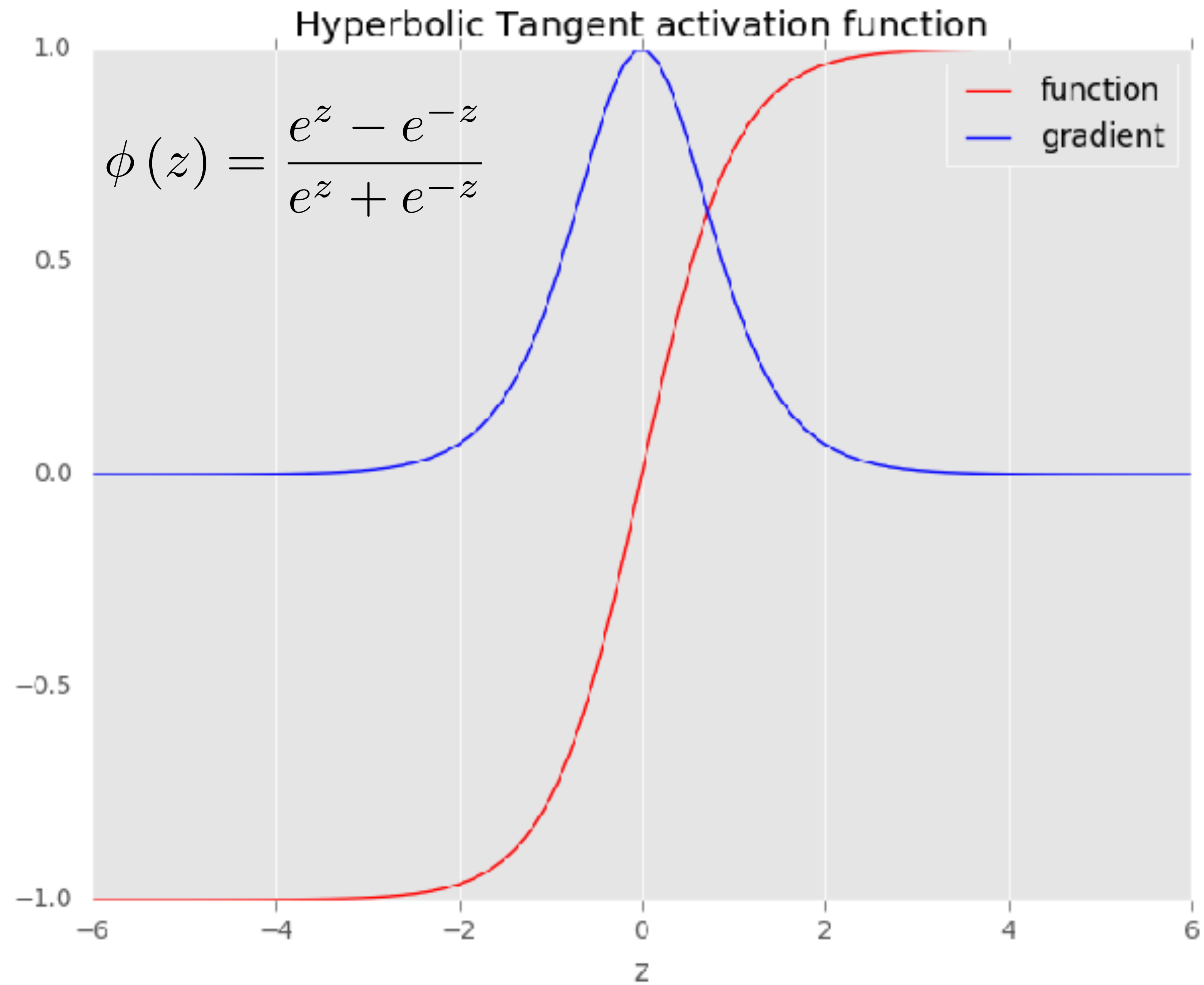
http://github.com/bmtgoncalves/Neural-Networks



Activation Function - tanh

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

- Non-Linear function

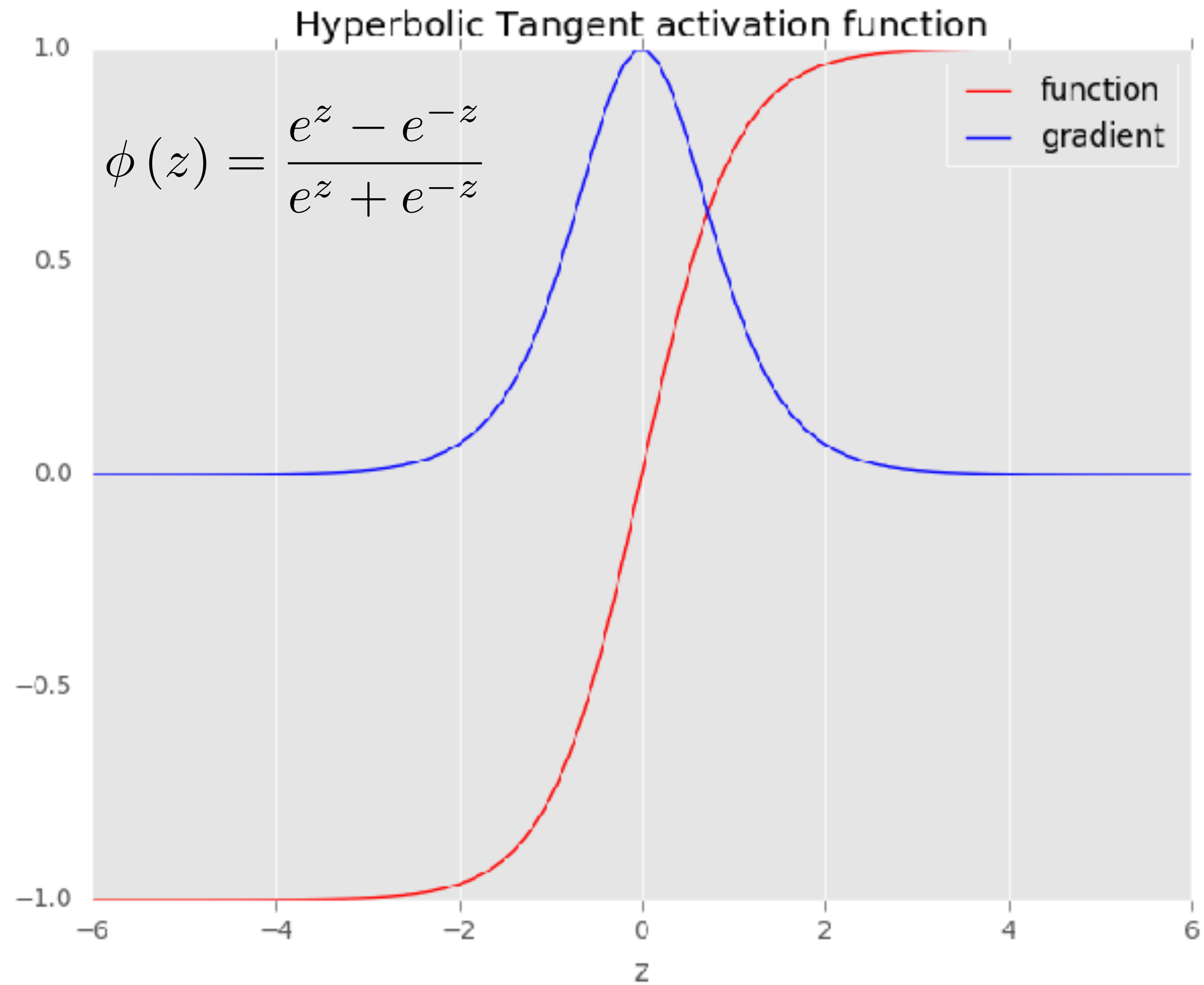


Activation Function - tanh

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

- Non-Linear function

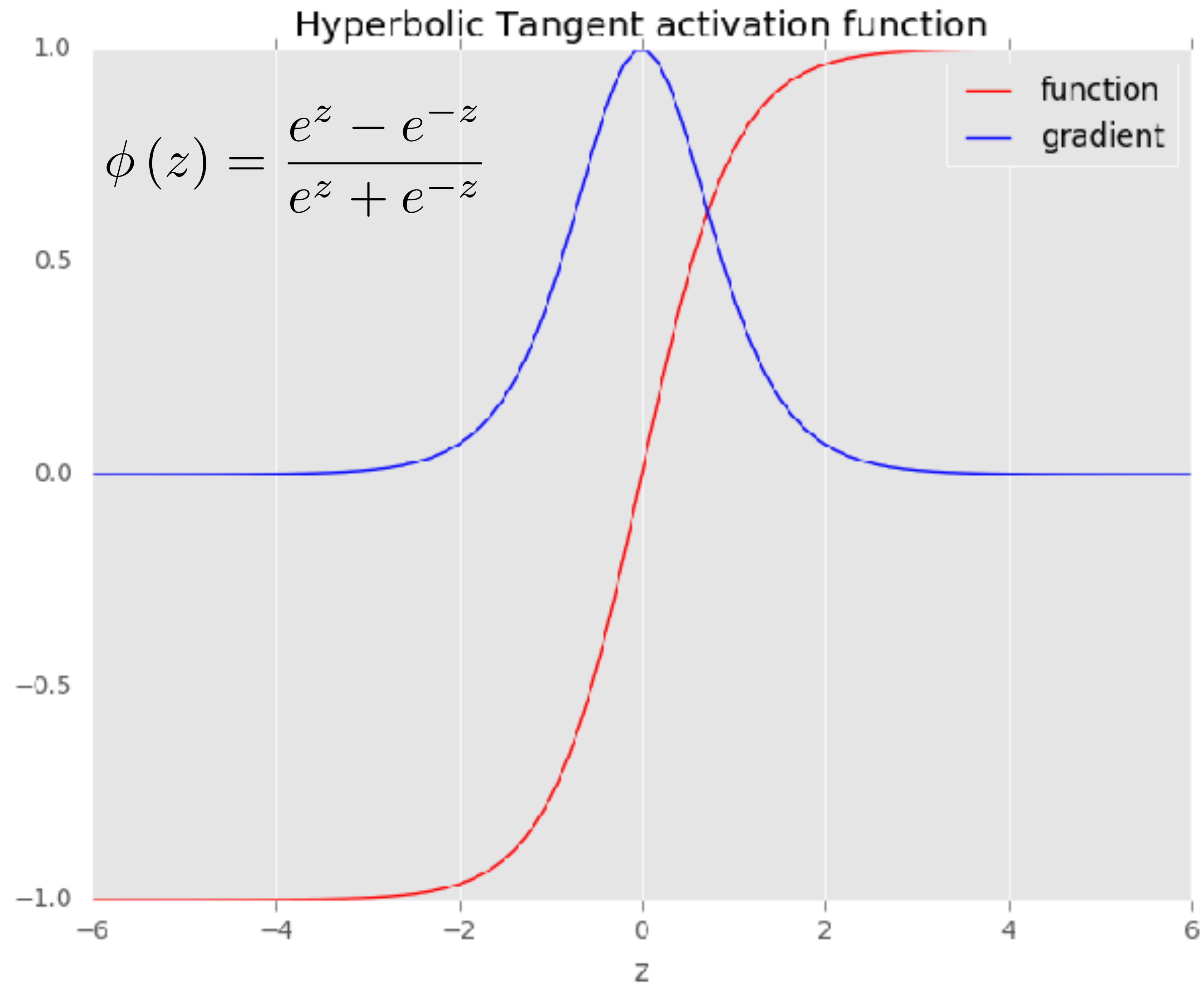
- Differentiable



Activation Function - tanh

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

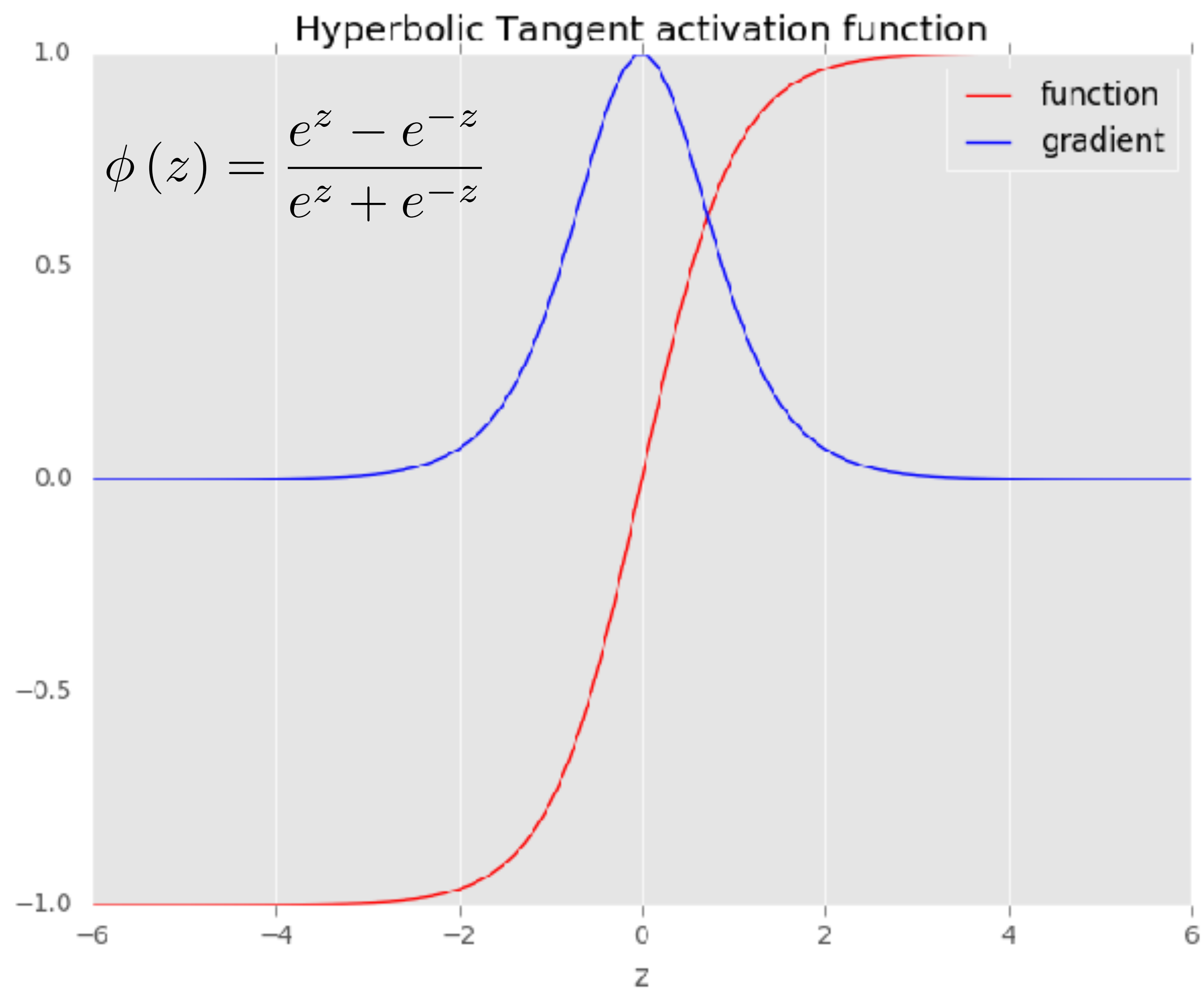
- Non-Linear function
- Differentiable
- non-decreasing



Activation Function - tanh

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

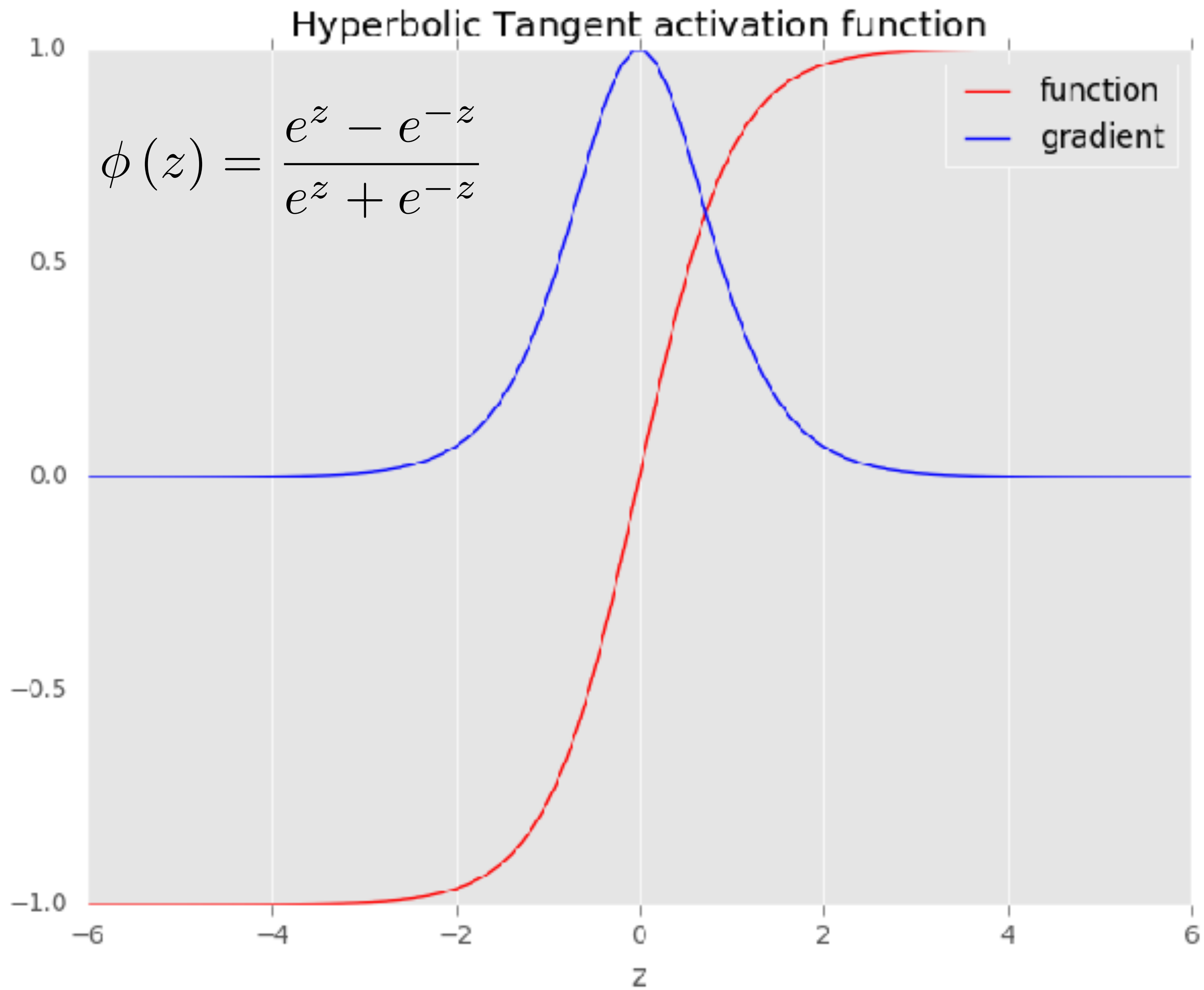
- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features



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

Forward Propagation

- The output of a perceptron is determined by a sequence of steps:

Forward Propagation

- The output of a perceptron is determined by a sequence of steps:
    - obtain the inputs

Forward Propagation

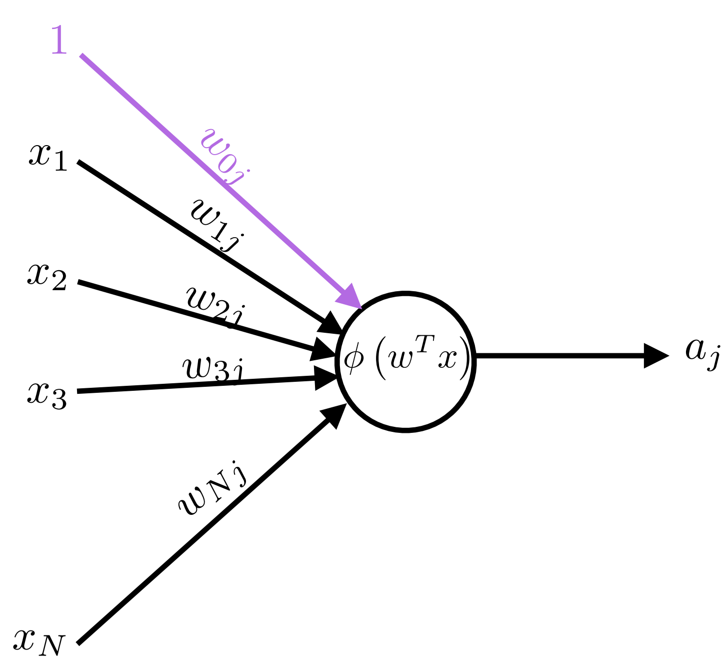
- The output of a perceptron is determined by a sequence of steps:
  - obtain the inputs
  - multiply the inputs by the respective weights

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

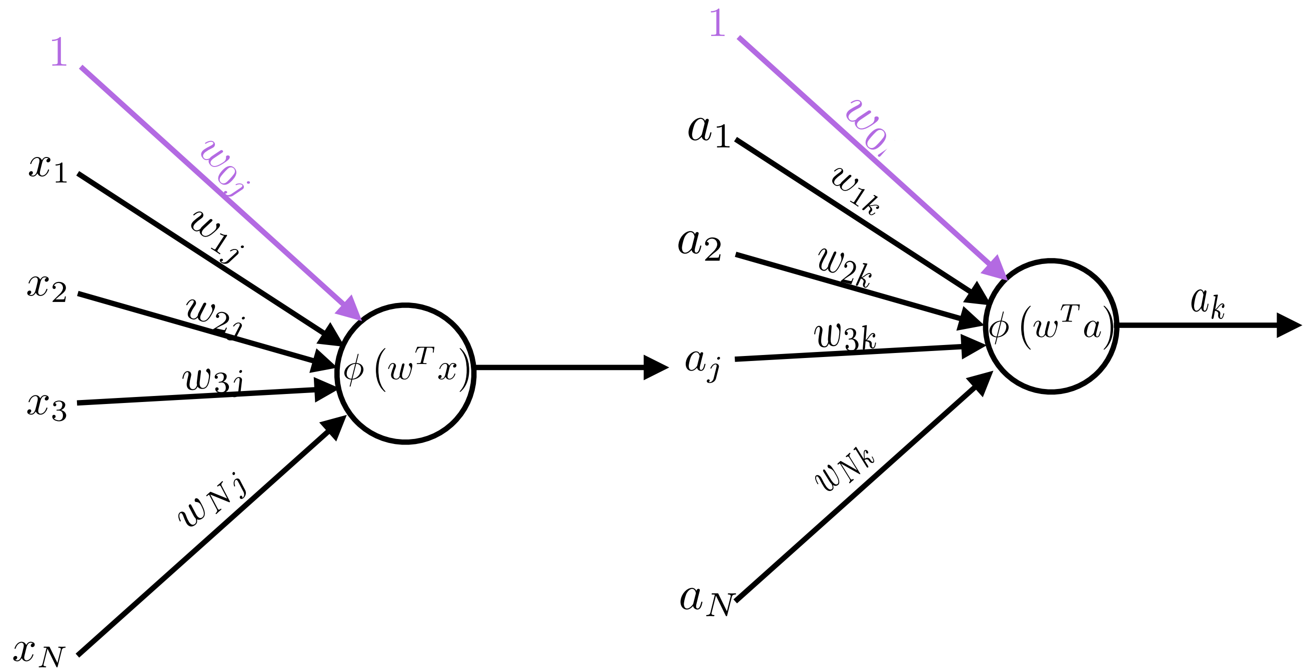
Forward Propagation

- 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.



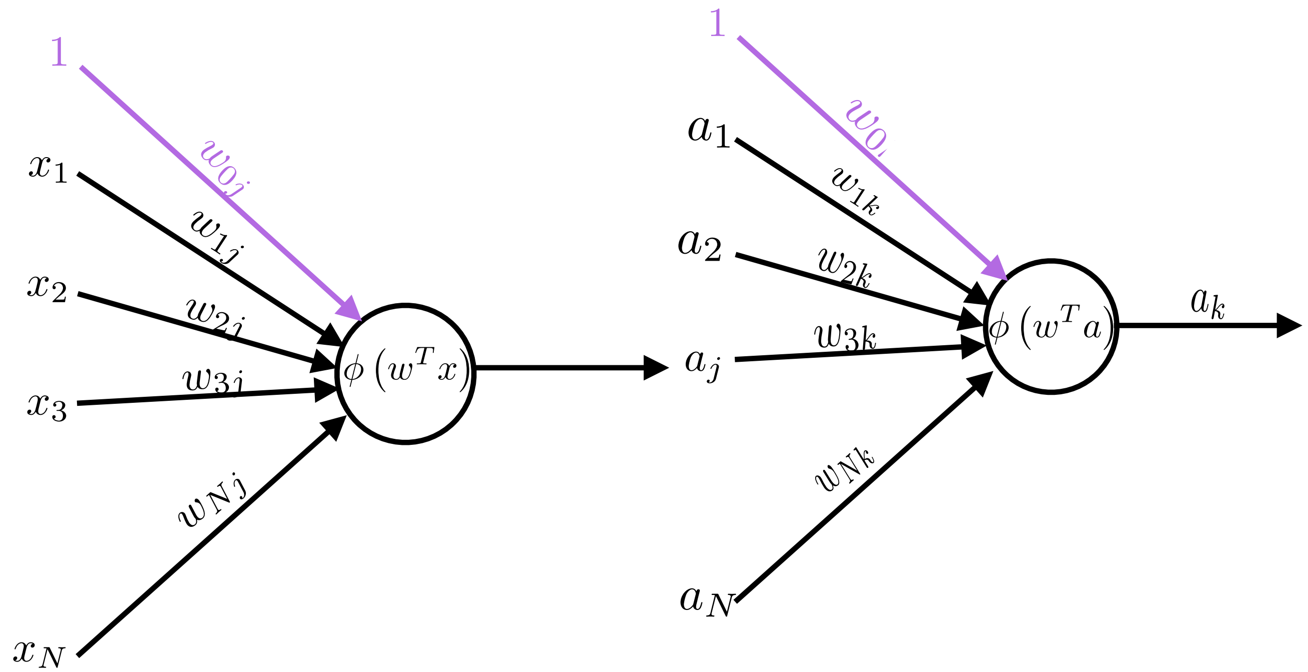
Forward Propagation

- 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.



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.



- But how can we propagate back the errors and update the weights?

Backward Propagation of Errors (BackProp)

Backward Propagation of Errors (BackProp)

- BackProp operates in two phases:

Backward Propagation of Errors (BackProp)

- BackProp operates in two phases:
  - Forward propagate the inputs and calculate the deltas

Backward Propagation of Errors (BackProp)

- BackProp operates in two phases:
    - Forward propagate the inputs and calculate the deltas
    - Update the weights

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.

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

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:

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:

$$E = \frac{1}{N}\sum_n |y_n - a_n|^2$$

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:

$$E = \frac{1}{N} \sum_n |y_n - a_n|^2$$

    - Cross Entropy

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

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:

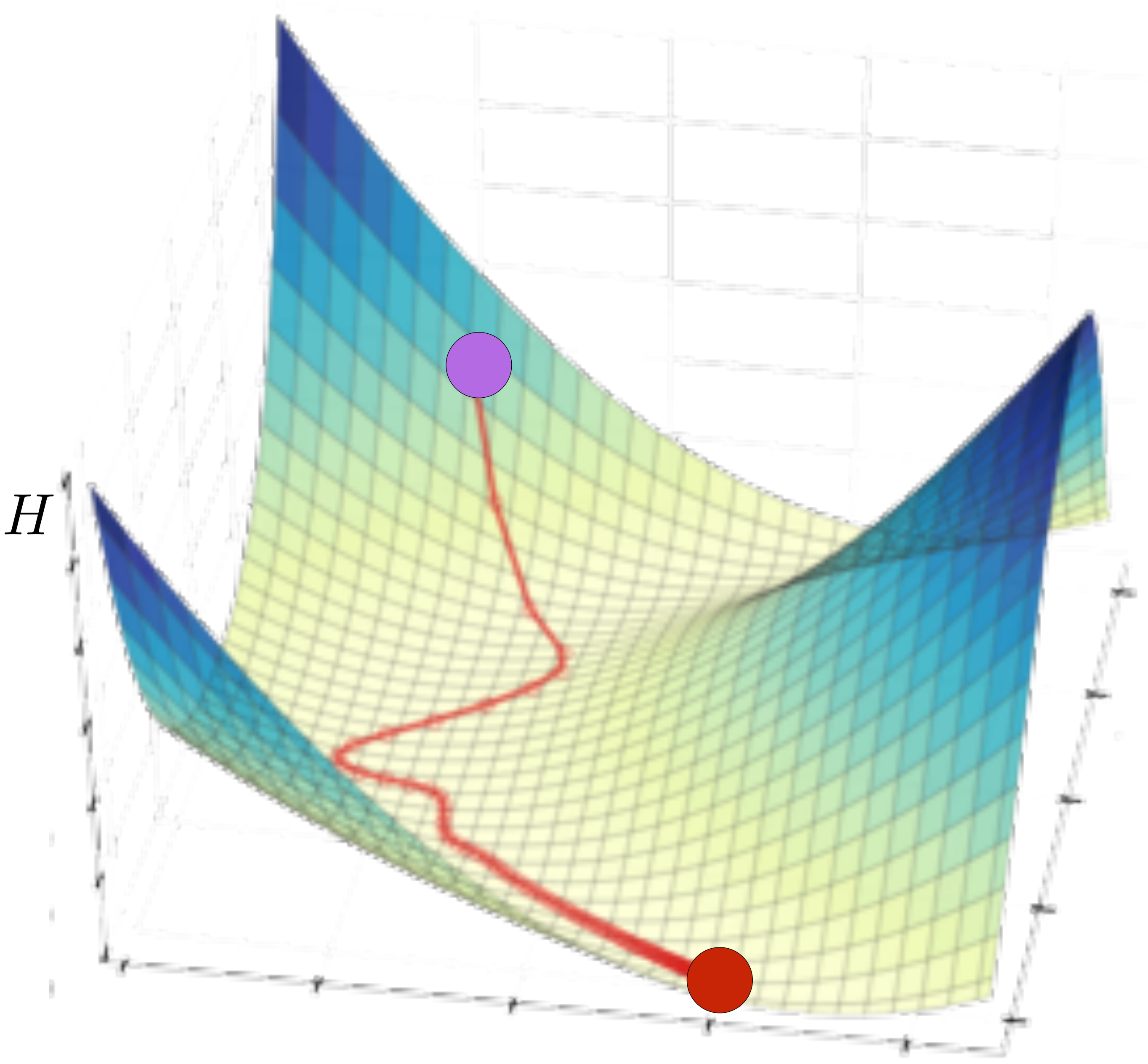
$$E = \frac{1}{N}\sum_n |y_n - a_n|^2$$

    - Cross Entropy

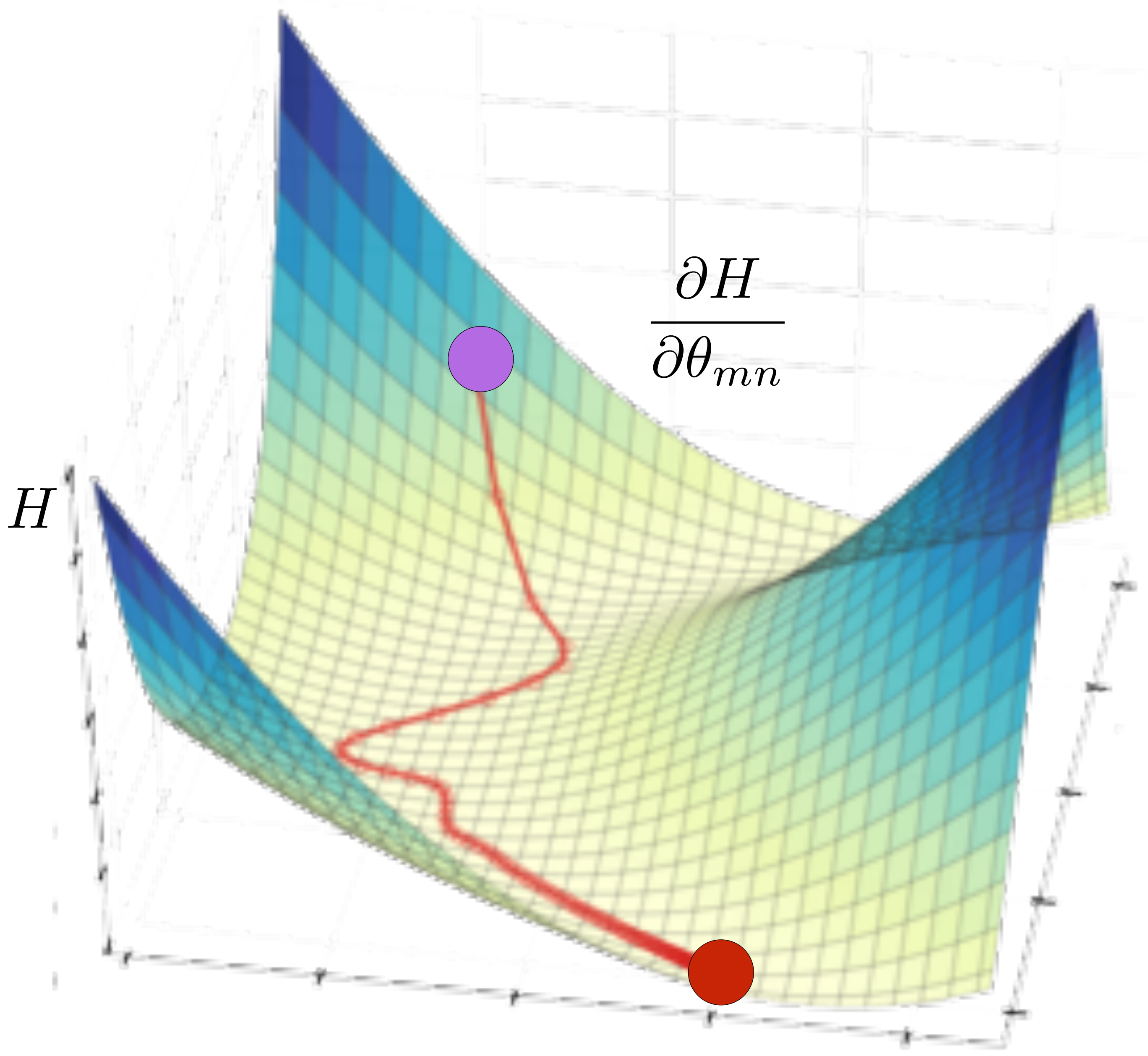
$$J = -\frac{1}{N}\sum_n \left[y_n^T \log a_n + (1 - y_n)^T \log(1 - a_n) \right]$$

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

Gradient Descent

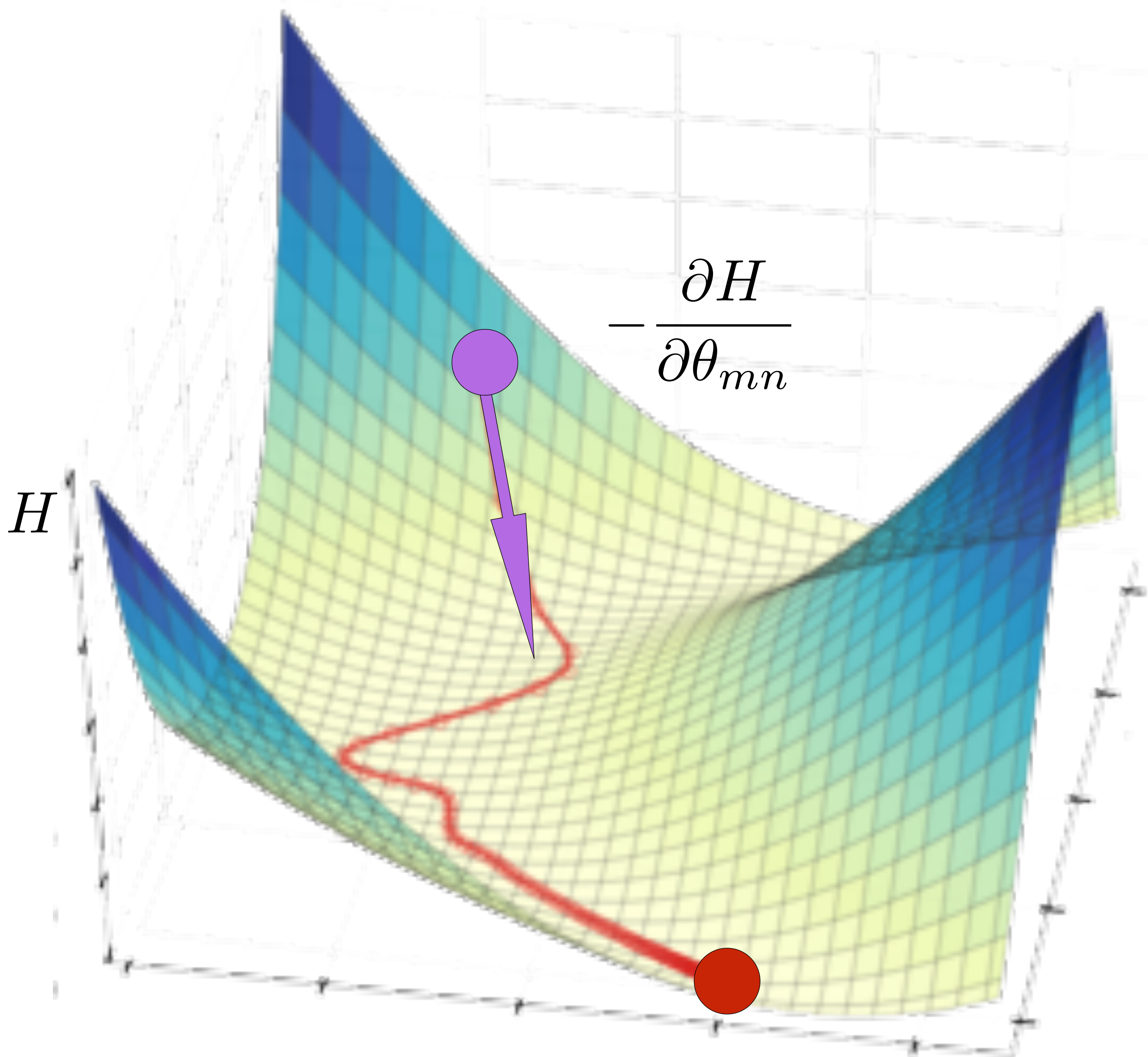


Gradient Descent



- Find the gradient for each training batch

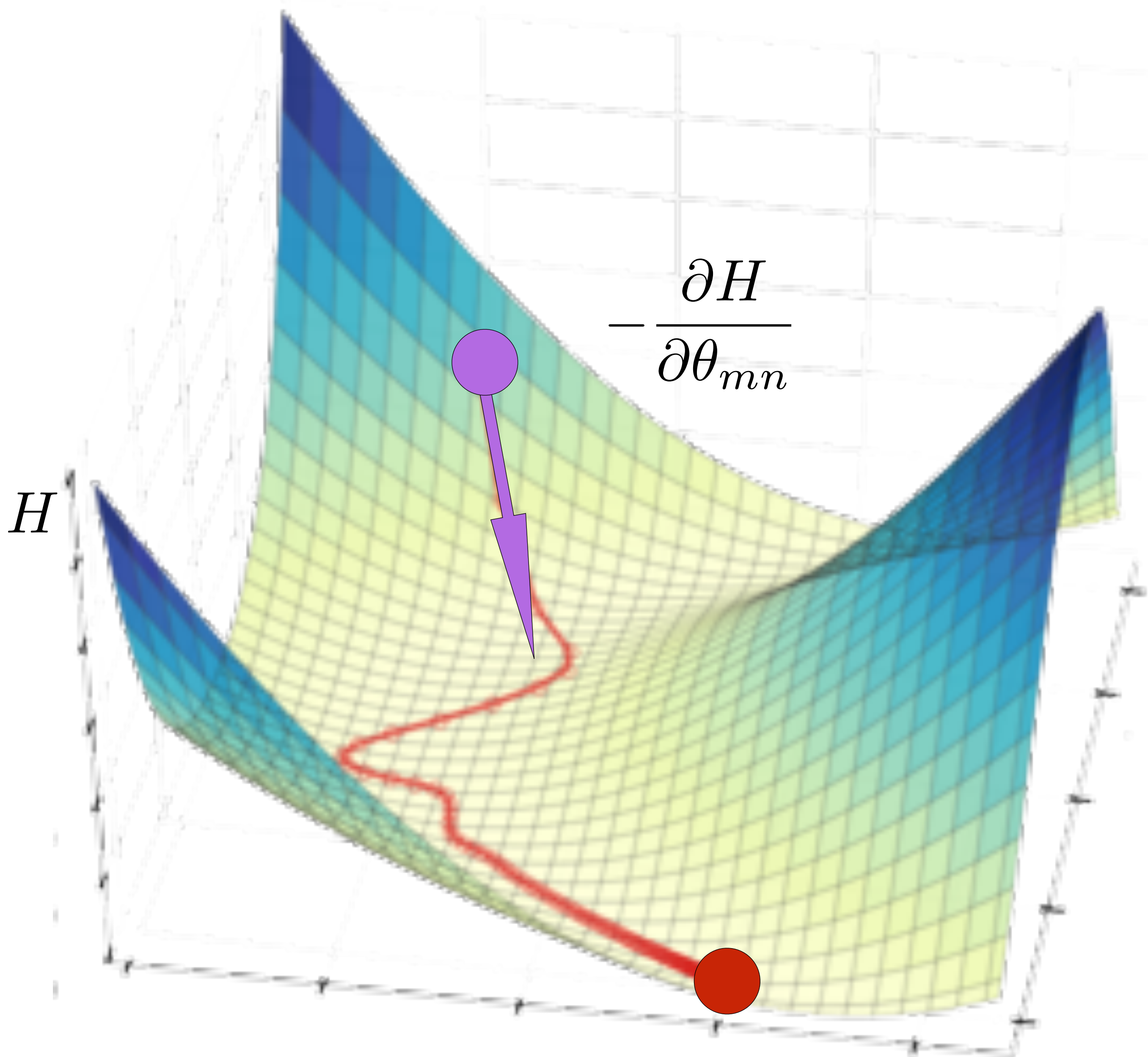
Gradient Descent



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

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

Gradient Descent

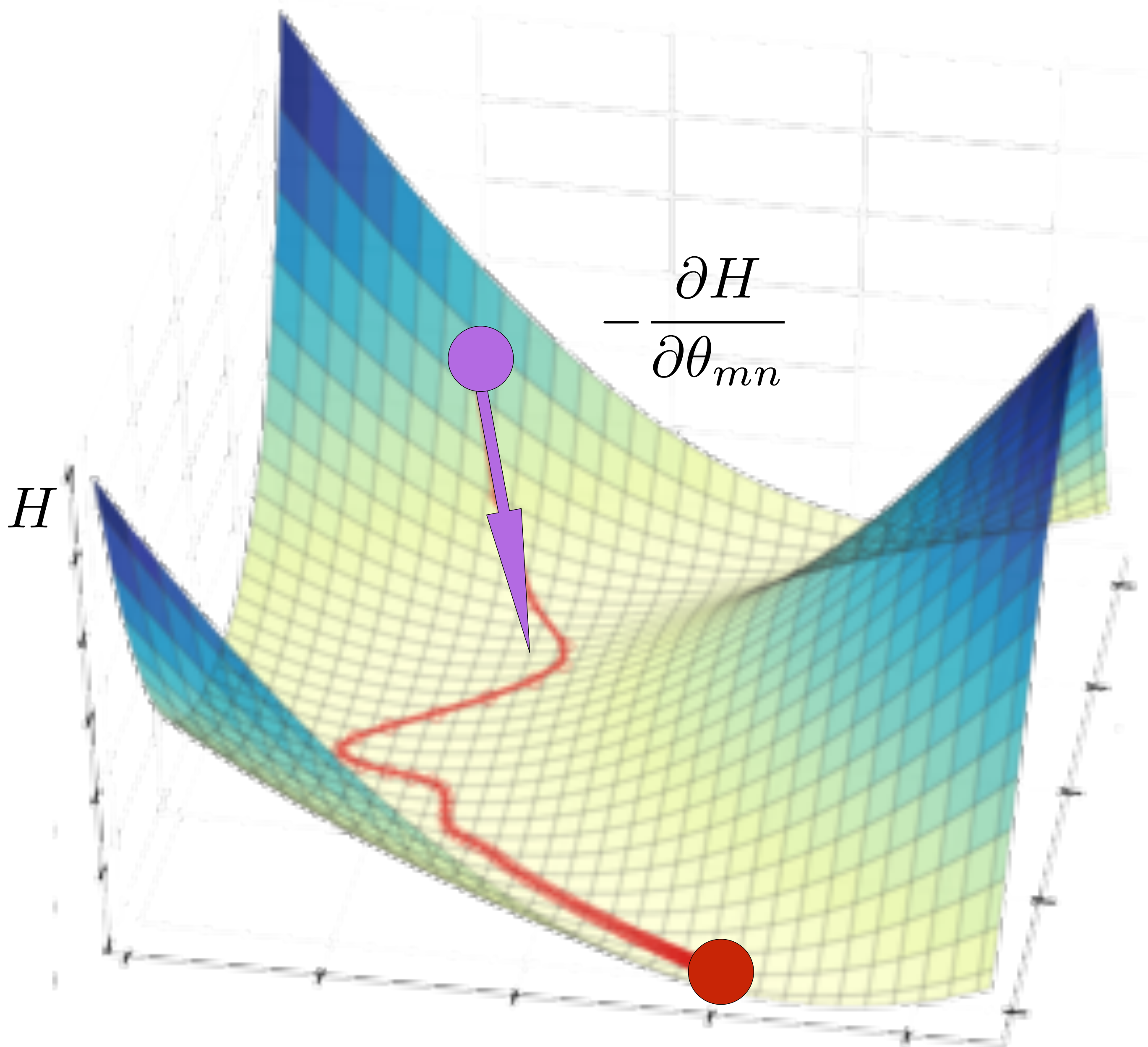


- Find the gradient for each training batch
- Take a step **downhill** along the direction of the gradient

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

- where α is the step size.

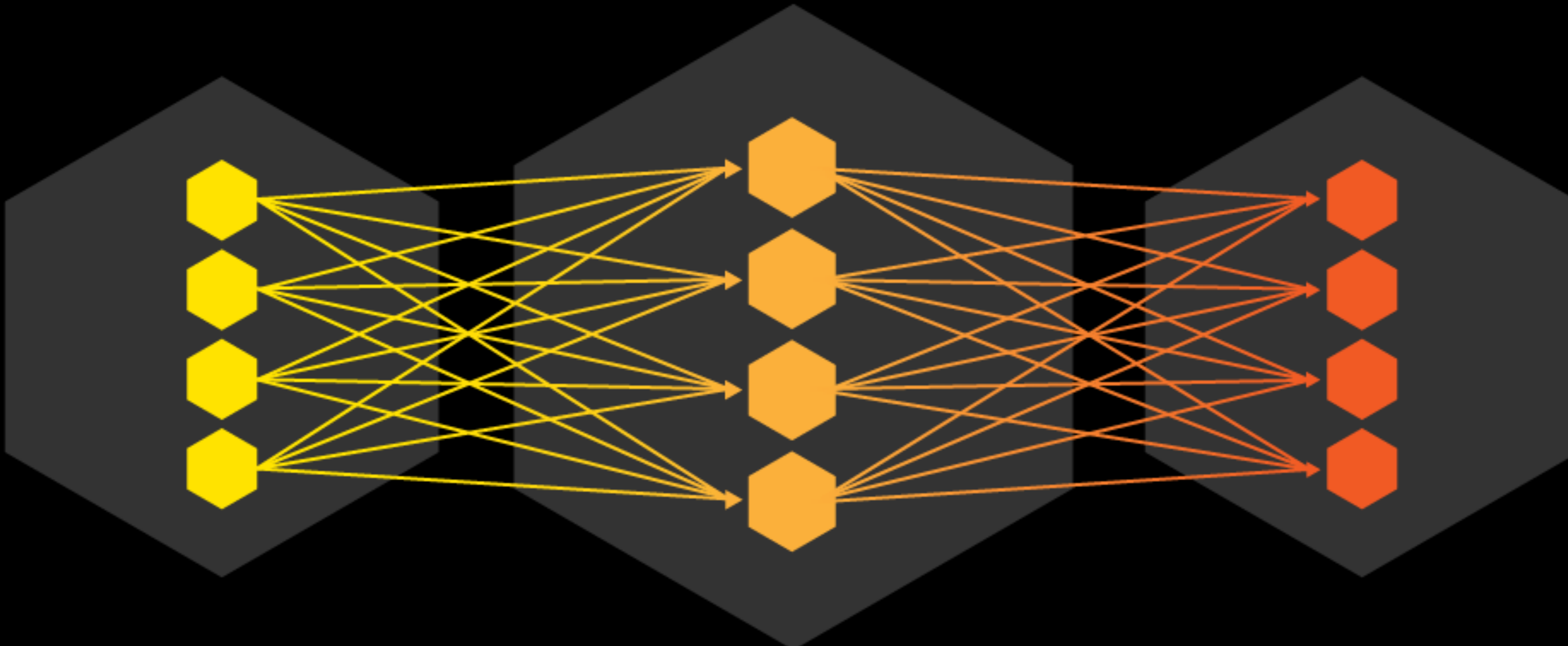
Gradient Descent



- 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

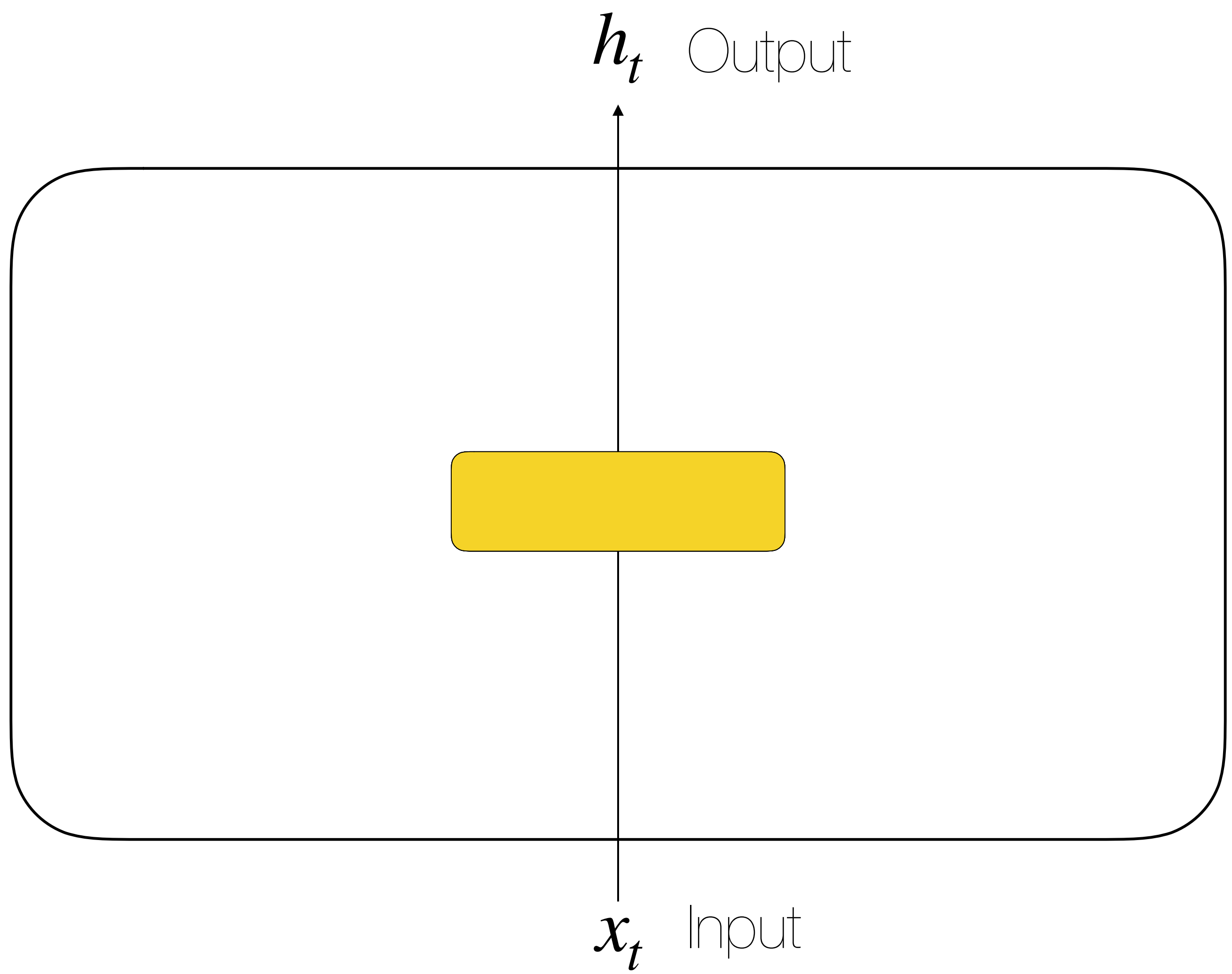
MACHINE

ALGORITHMS
TECHNIQUES
MODELS

OUTPUT TERMS

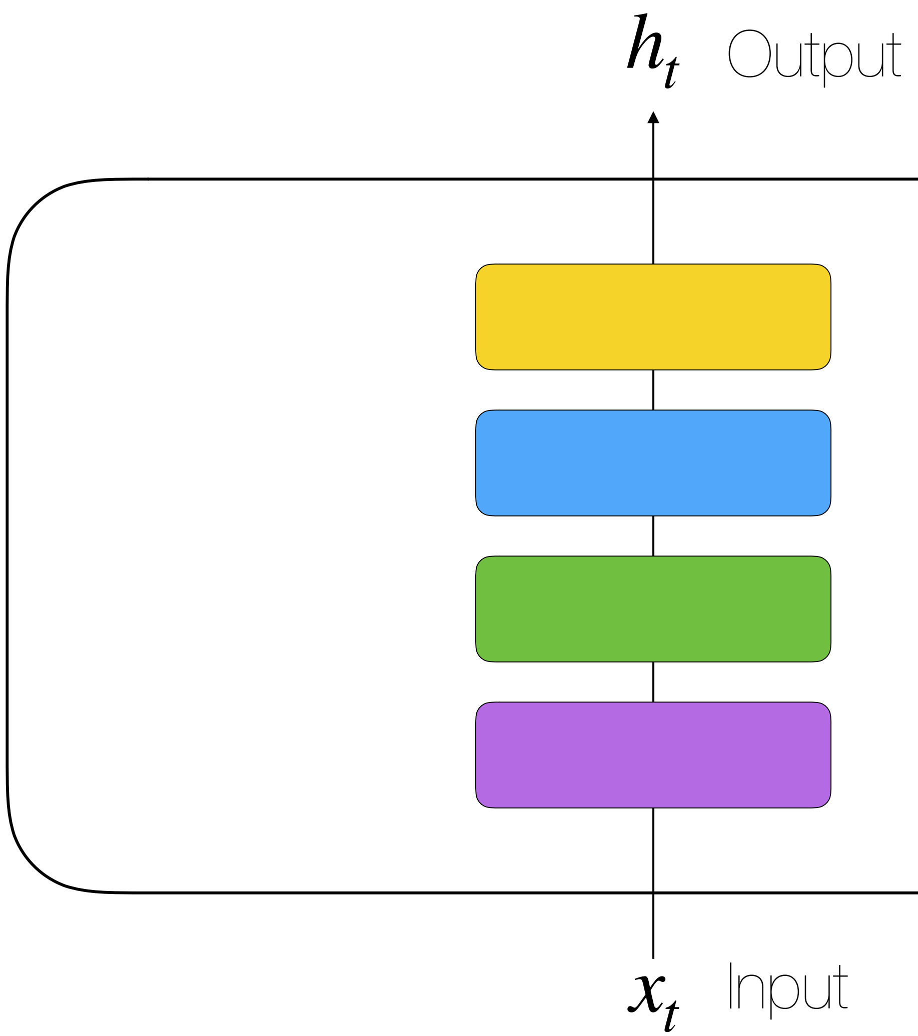
CLASSES
RESPONSES
TARGETS
DEPENDANT VARIABLES

Feed Forward Networks



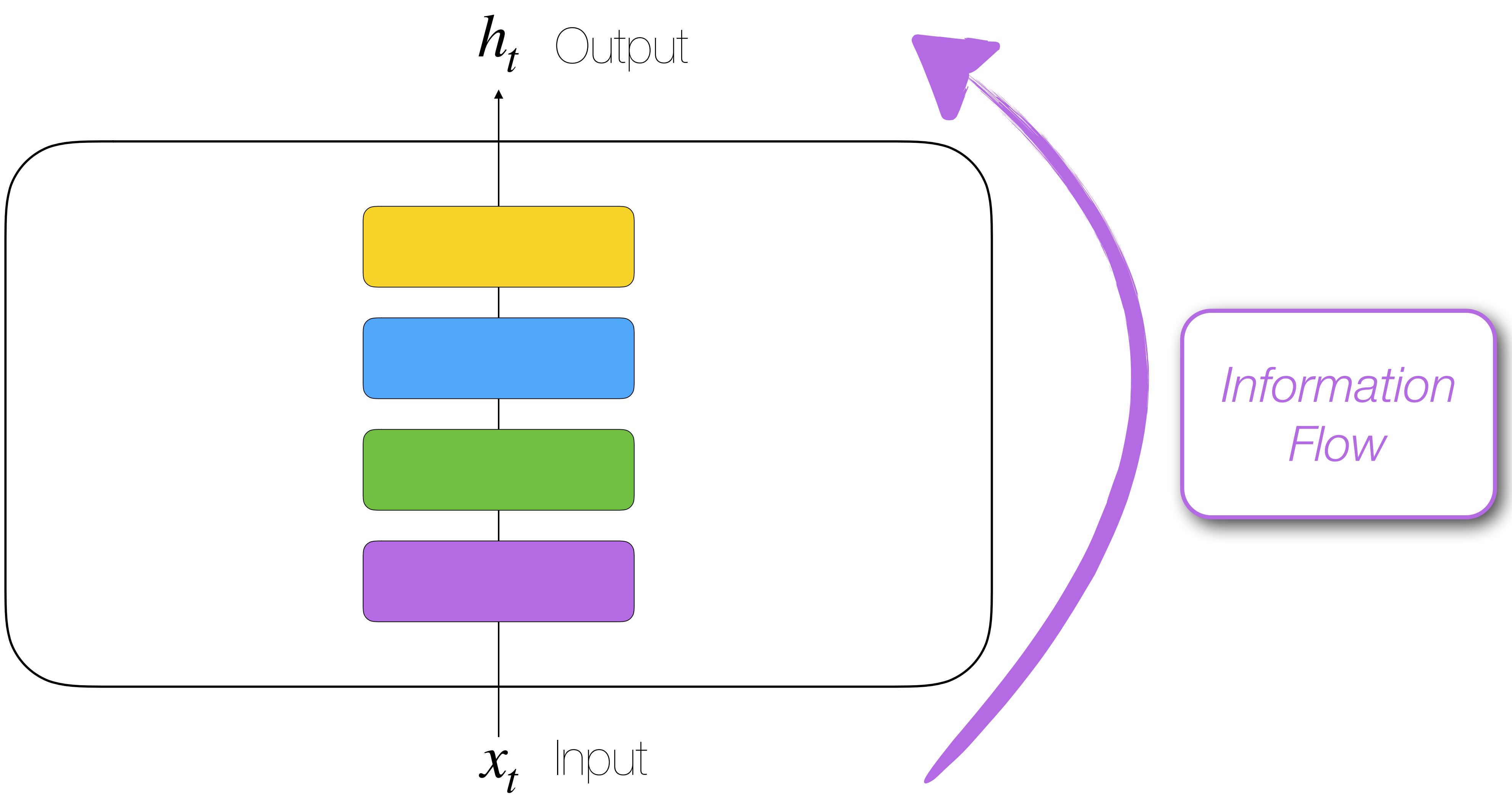
$$h_t = f\left(x_t\right)$$

Feed Forward Networks



$$h_t = f\left(x_t\right)$$

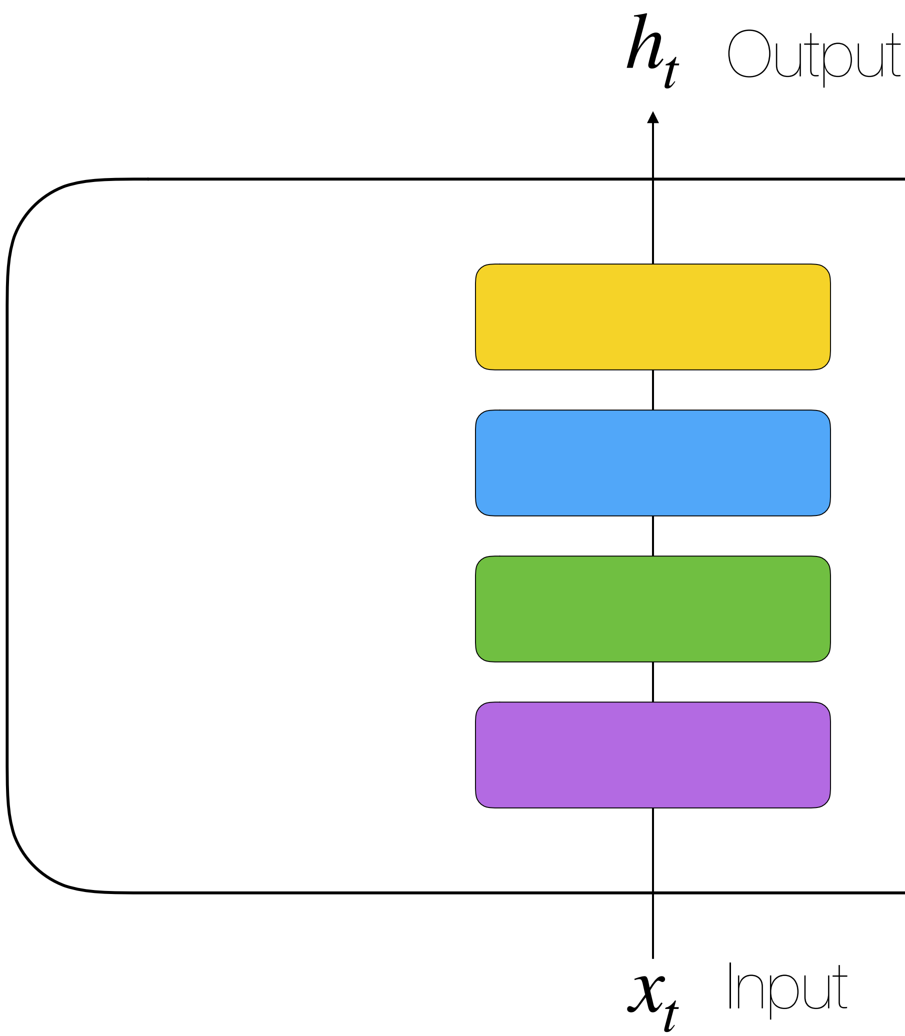
Feed Forward Networks



$$h_t = f\left(x_t\right)$$

Recurrent Neural Network (RNN)

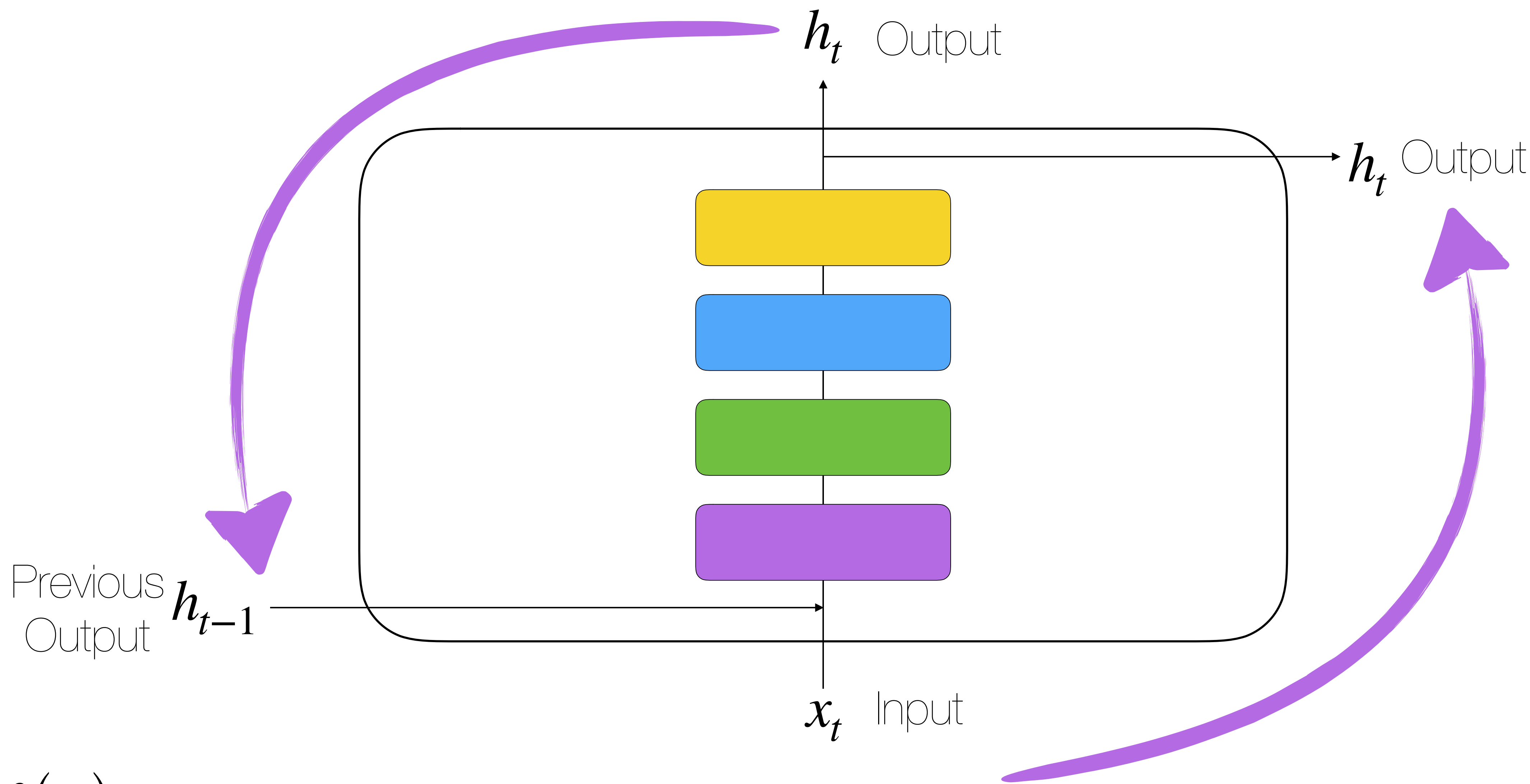
Information Flow



$$h_t = f\left(x_t\right)$$

Recurrent Neural Network (RNN)

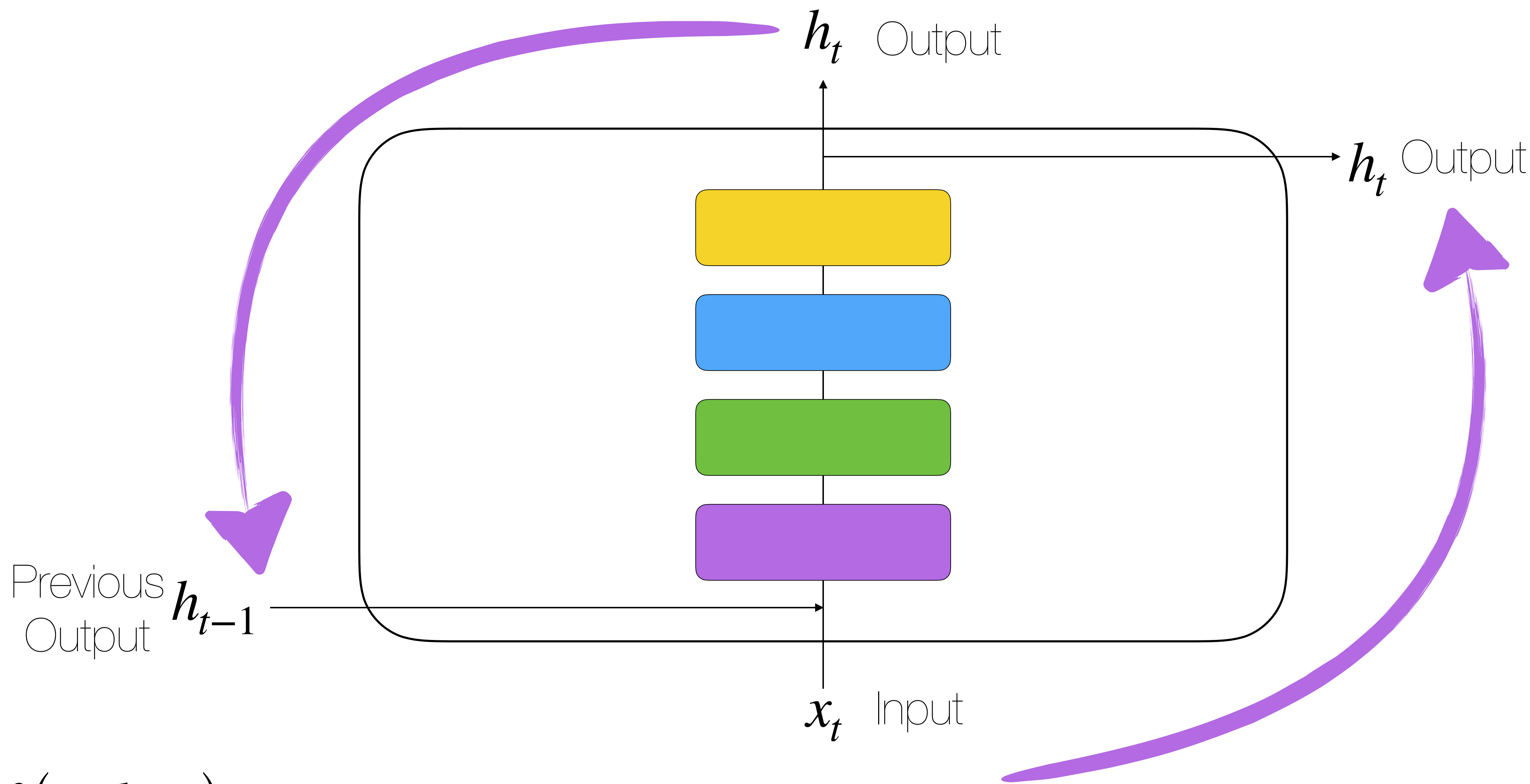
Information Flow



$$h_t = f\left(x_t\right)$$

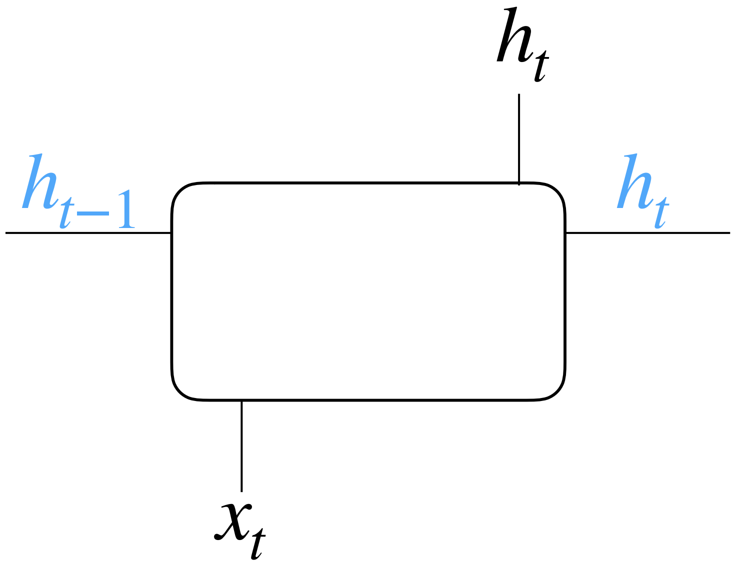
Recurrent Neural Network (RNN)

Information Flow



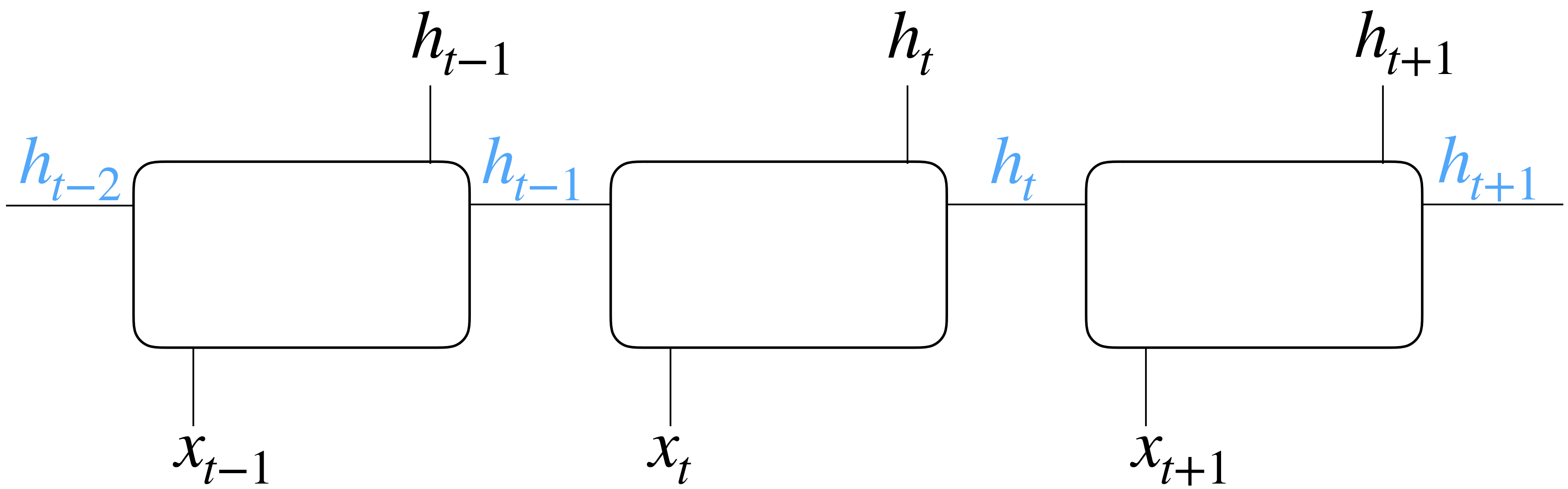
$$h_t = f\left(x_t, h_{t-1}\right)$$

Recurrent Neural Network (RNN)



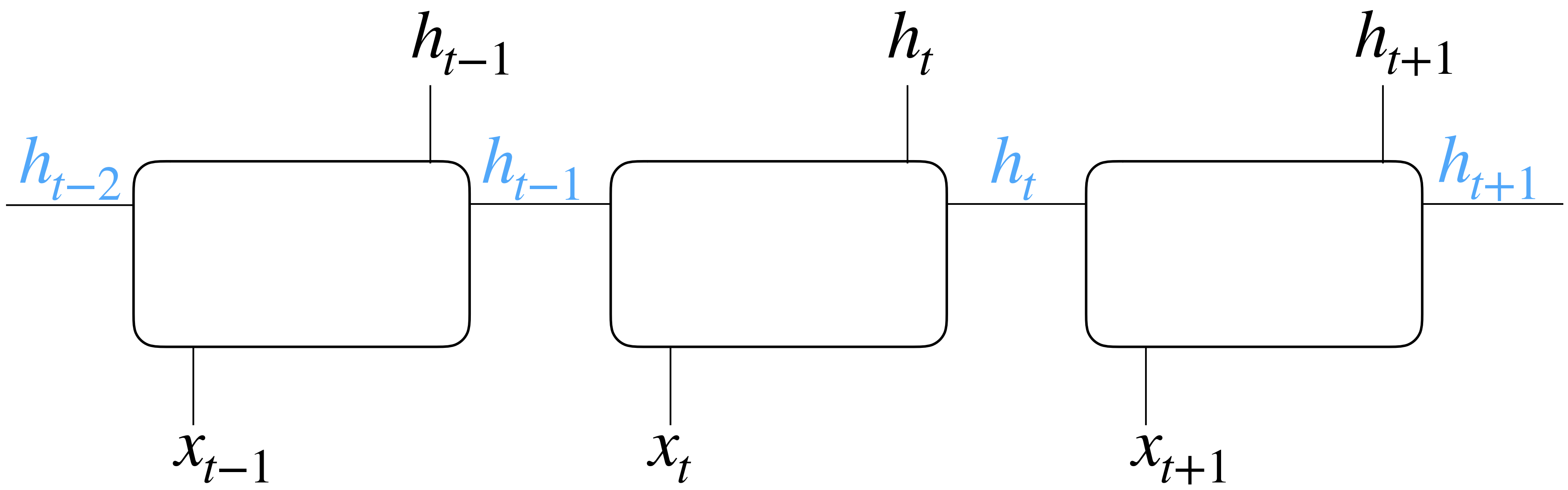
Recurrent Neural Network (RNN)

- Each output depends (implicitly) on all previous outputs.

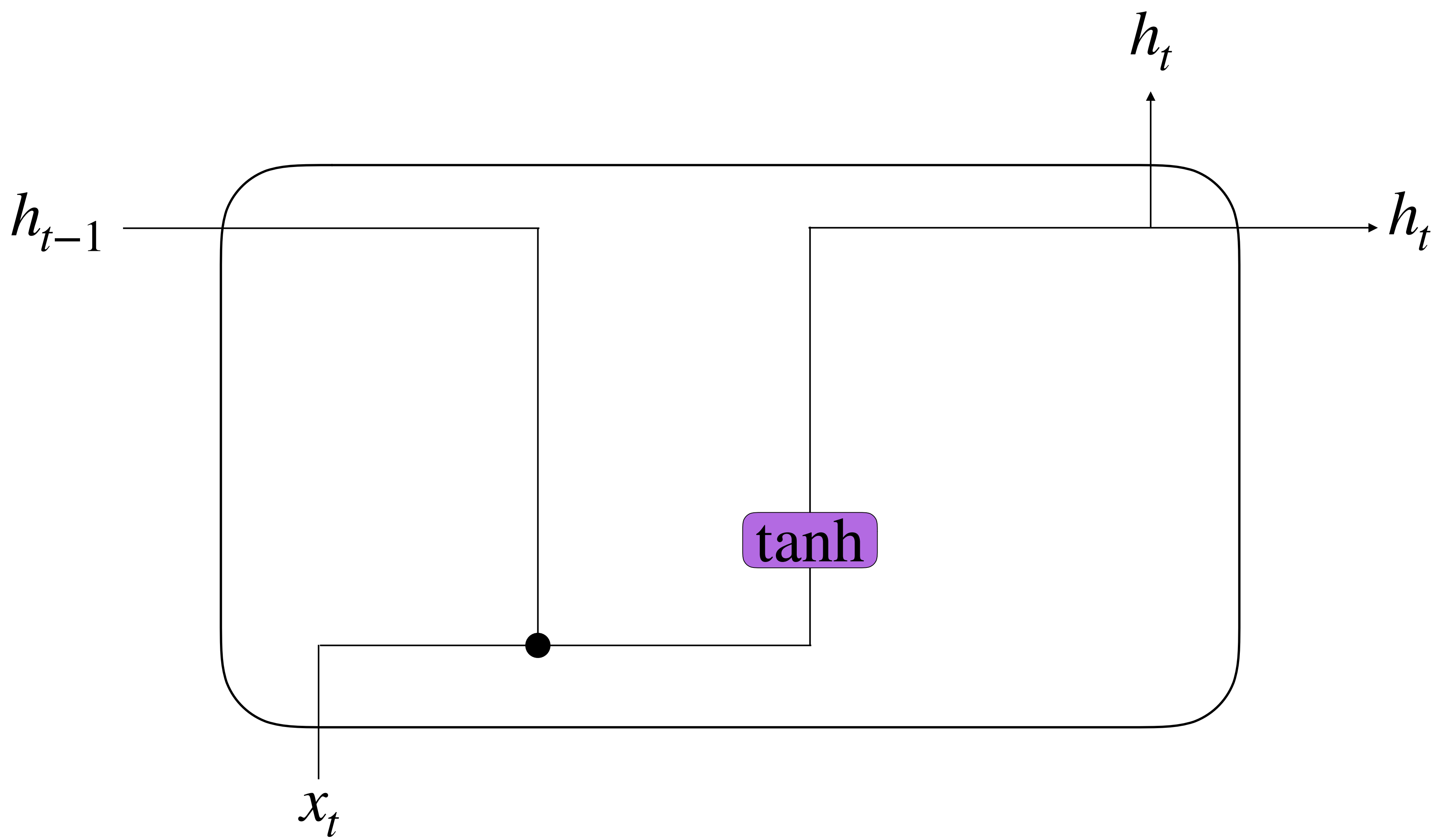


Recurrent Neural Network (RNN)

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

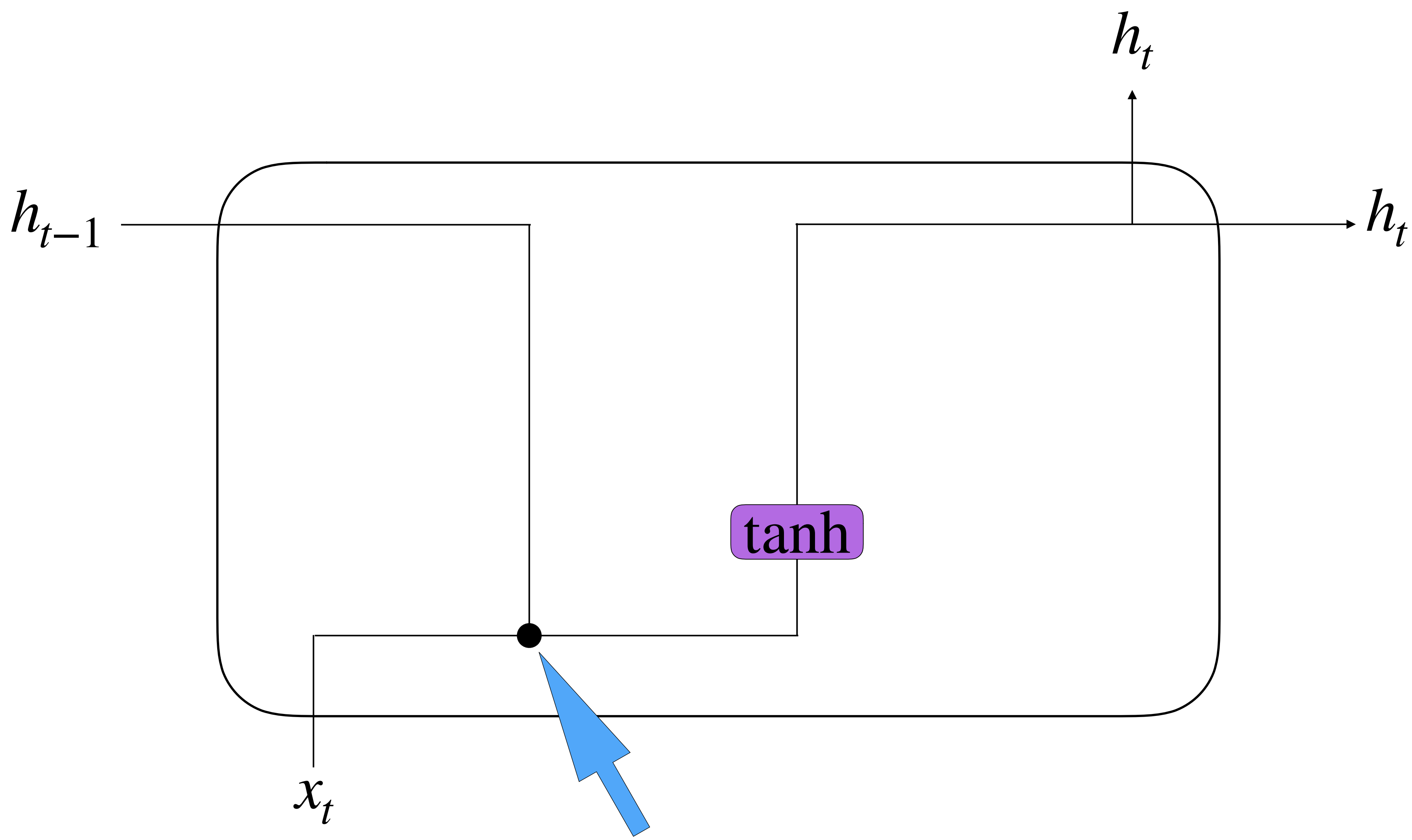


Recurrent Neural Network (RNN)



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

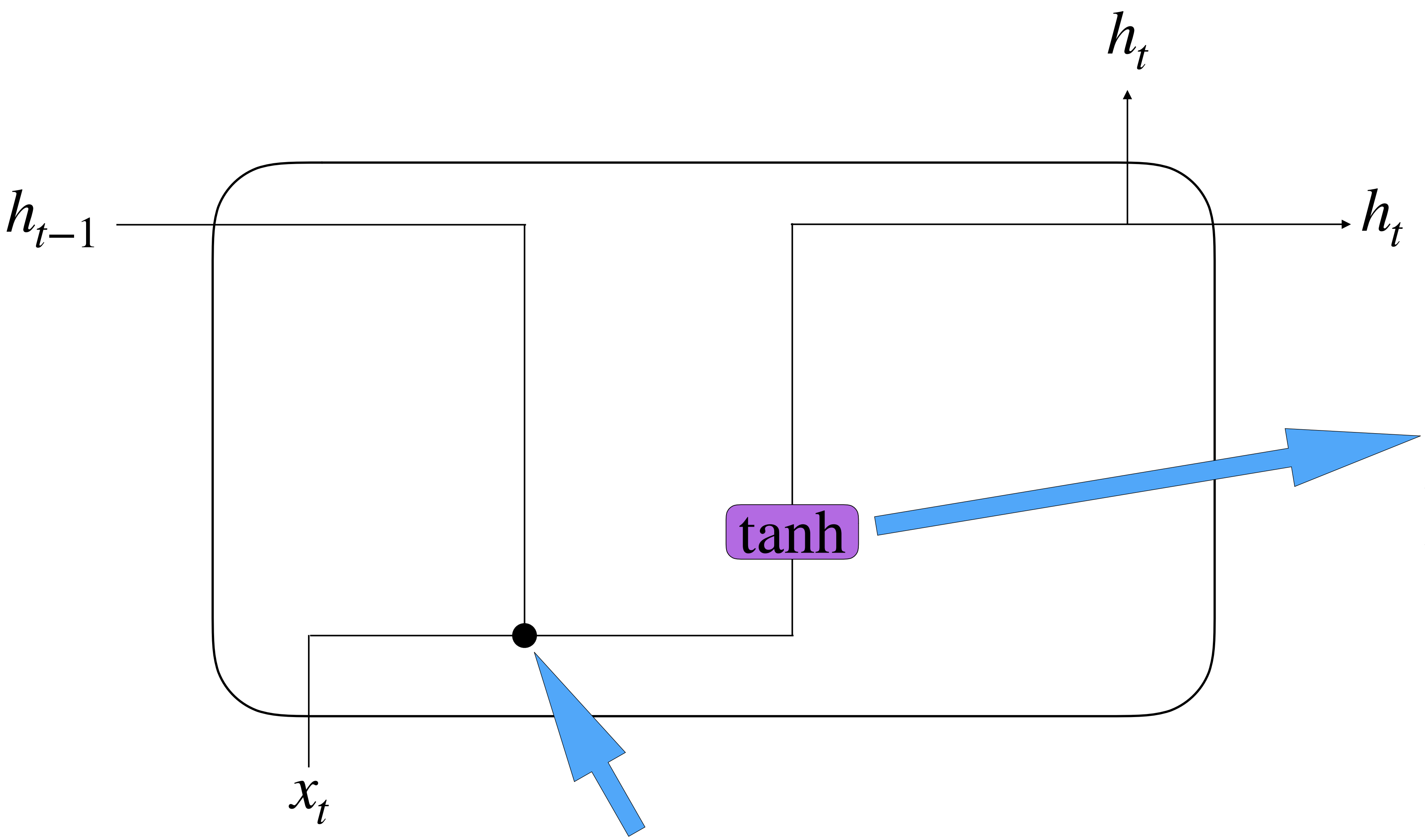
Recurrent Neural Network (RNN)

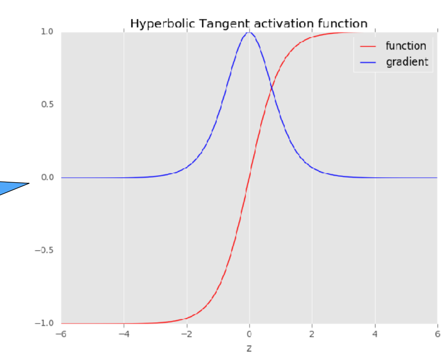


$$h_t = \tanh\left(Wh_{t-1} + Ux_t\right)$$

Concatenate
both inputs.

Recurrent Neural Network (RNN)





Concatenate
both inputs.

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

Timeseries

Timeseries

- Temporal sequence of data points

Timeseries

- Temporal sequence of data points
- Consecutive points are strongly correlated

Timeseries

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

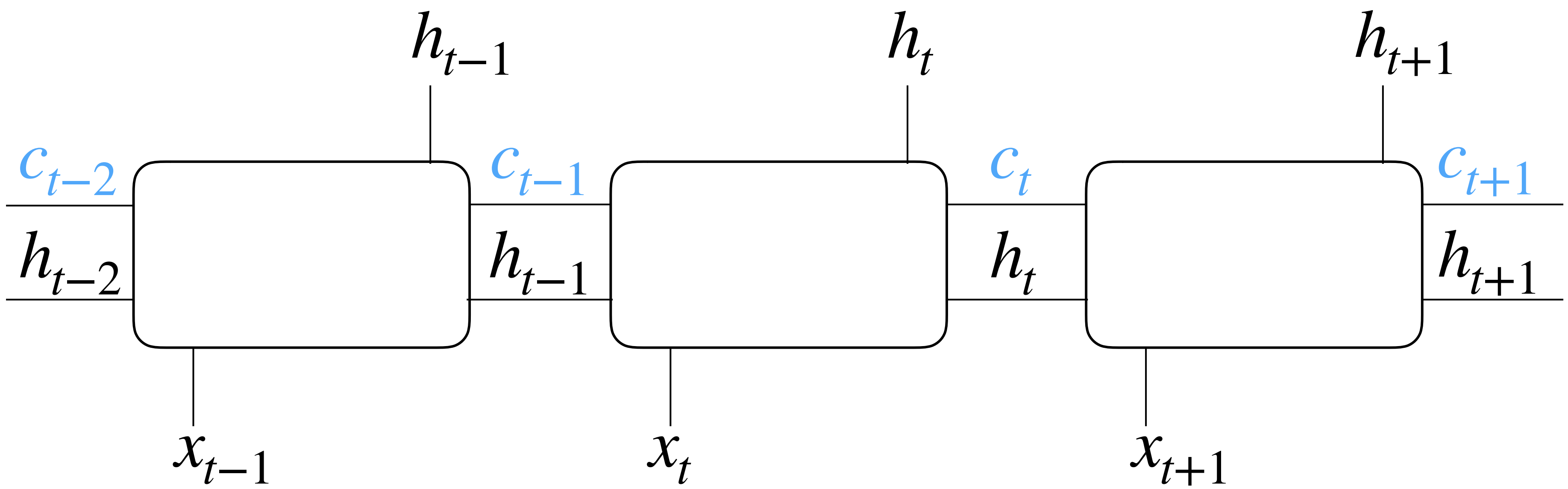
Timeseries

- 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

Timeseries

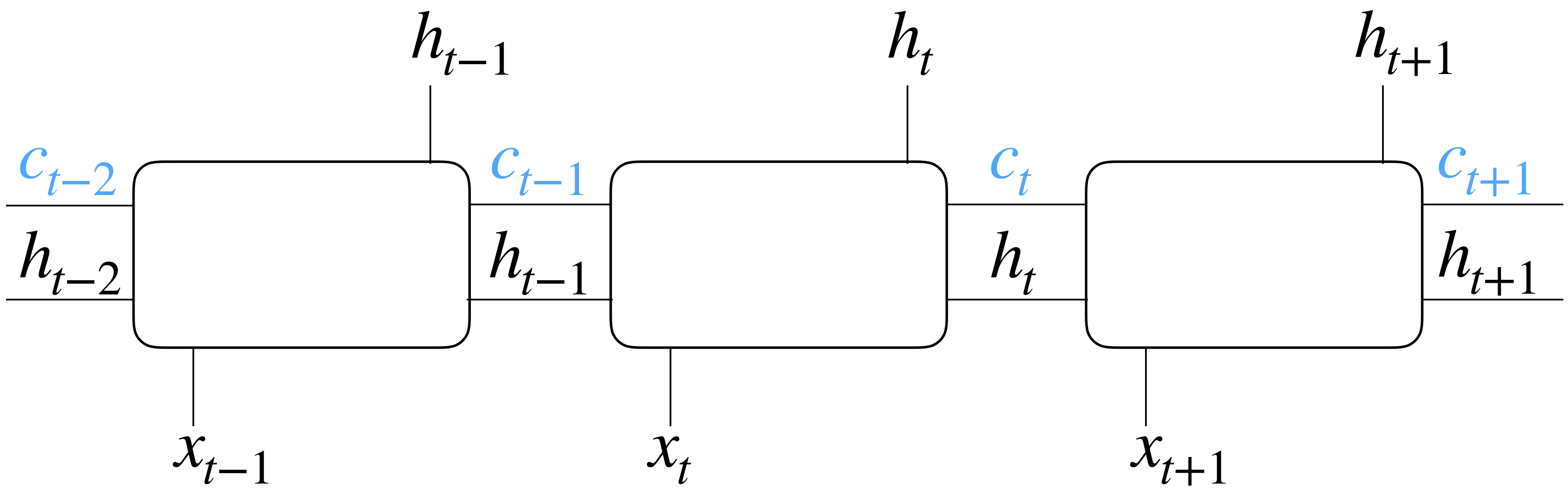
- Temporal sequence of data points
- Consecutive points are strongly correlated
- Common in statistics, signal processing, econometrics, mathematical finance, earthquake prediction, etc
- Numeric (real or discrete) or symbolic data

Long-Short Term Memory (LSTM)



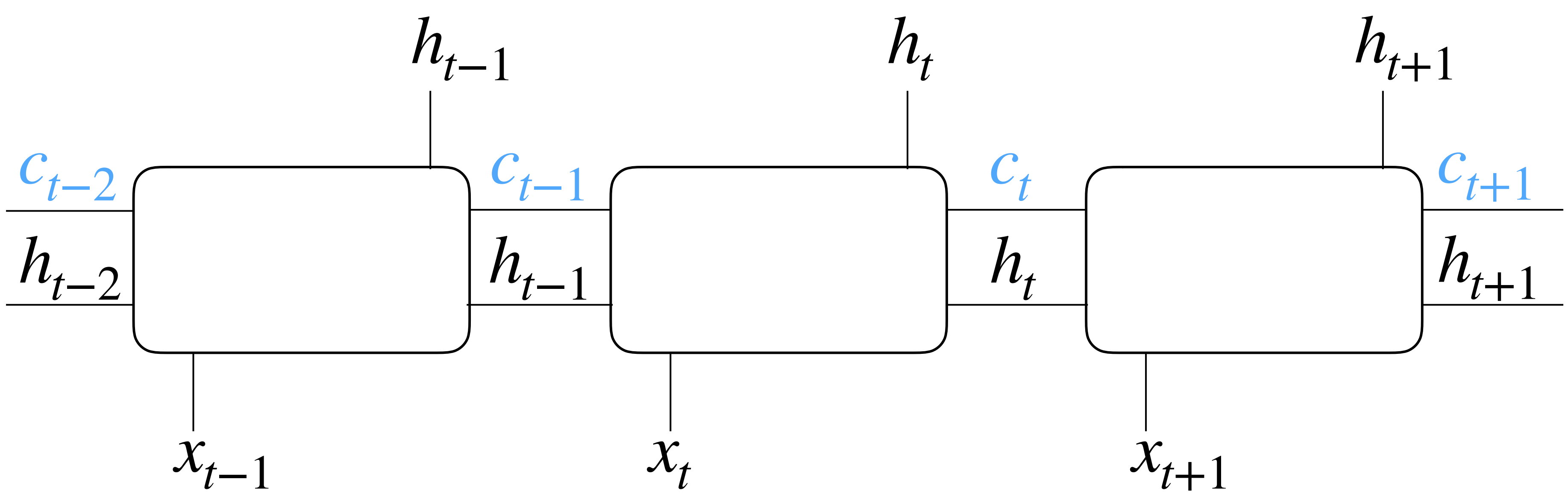
Long-Short Term Memory (LSTM)

- What if we want to keep explicit information about previous states (memory)?



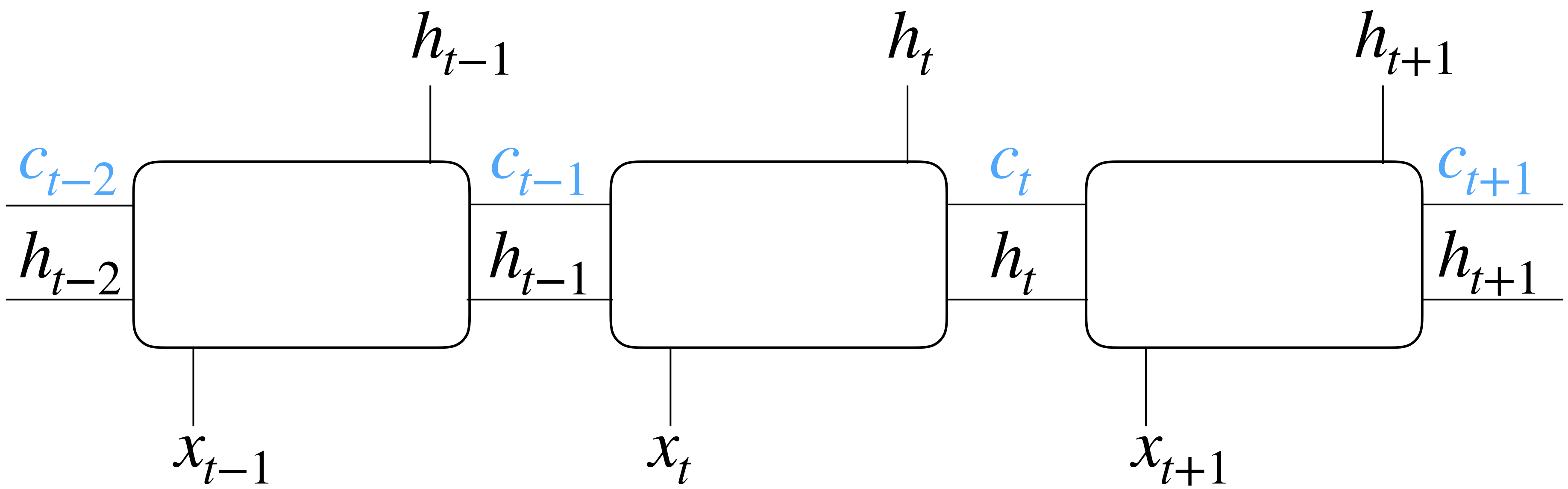
Long-Short Term Memory (LSTM)

- What if we want to keep explicit information about previous states (memory)?
- How much information is kept, can be controlled through gates.

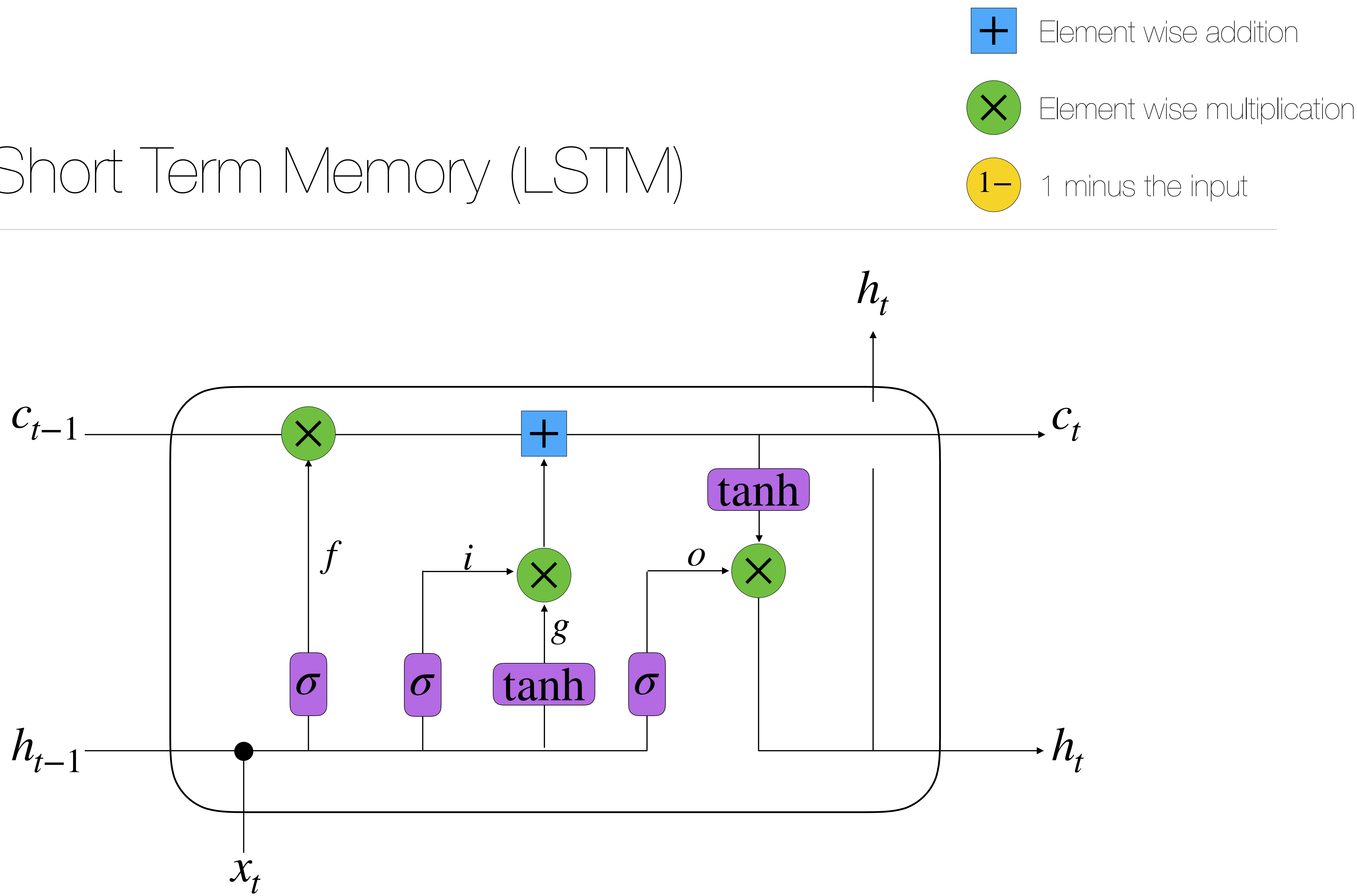


Long-Short Term Memory (LSTM)

- 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)



$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right) \qquad g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$

$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right) \qquad c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$

$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right) \qquad h_t = \tanh\left(c_t\right) \otimes o$$

www.bgoncalves.com

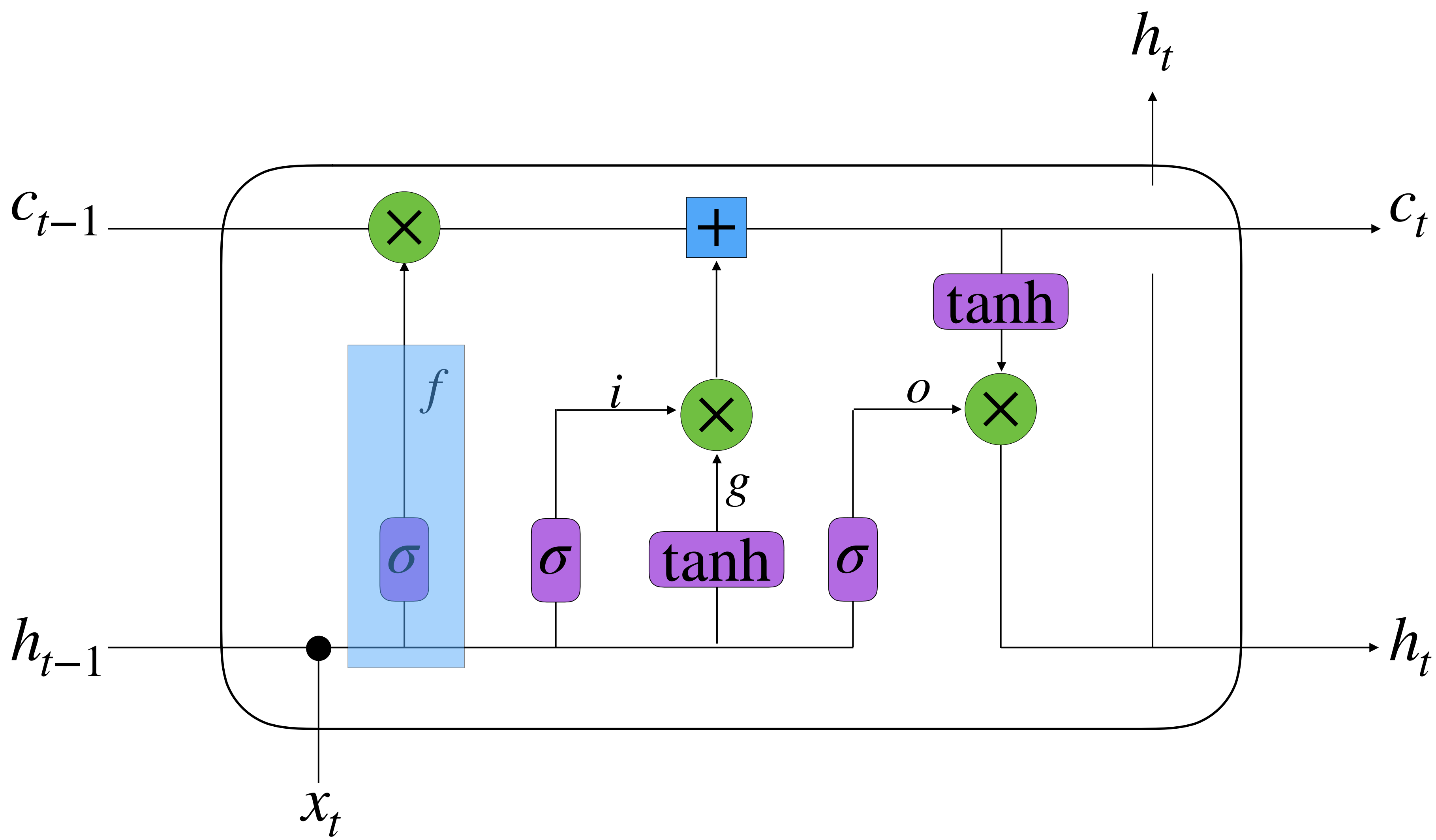
Element wise addition

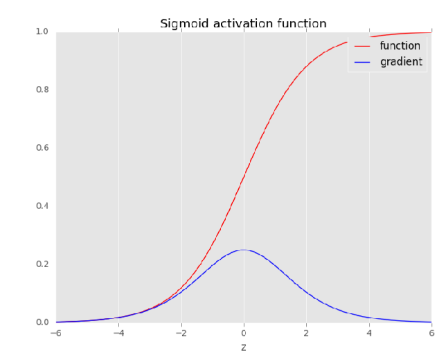
Element wise multiplication

1 minus the input

Long-Short Term Memory (LSTM)

Forget gate:
How much of
the previous
state should
be kept?





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

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

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

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

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

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

www.bgoncalves.com

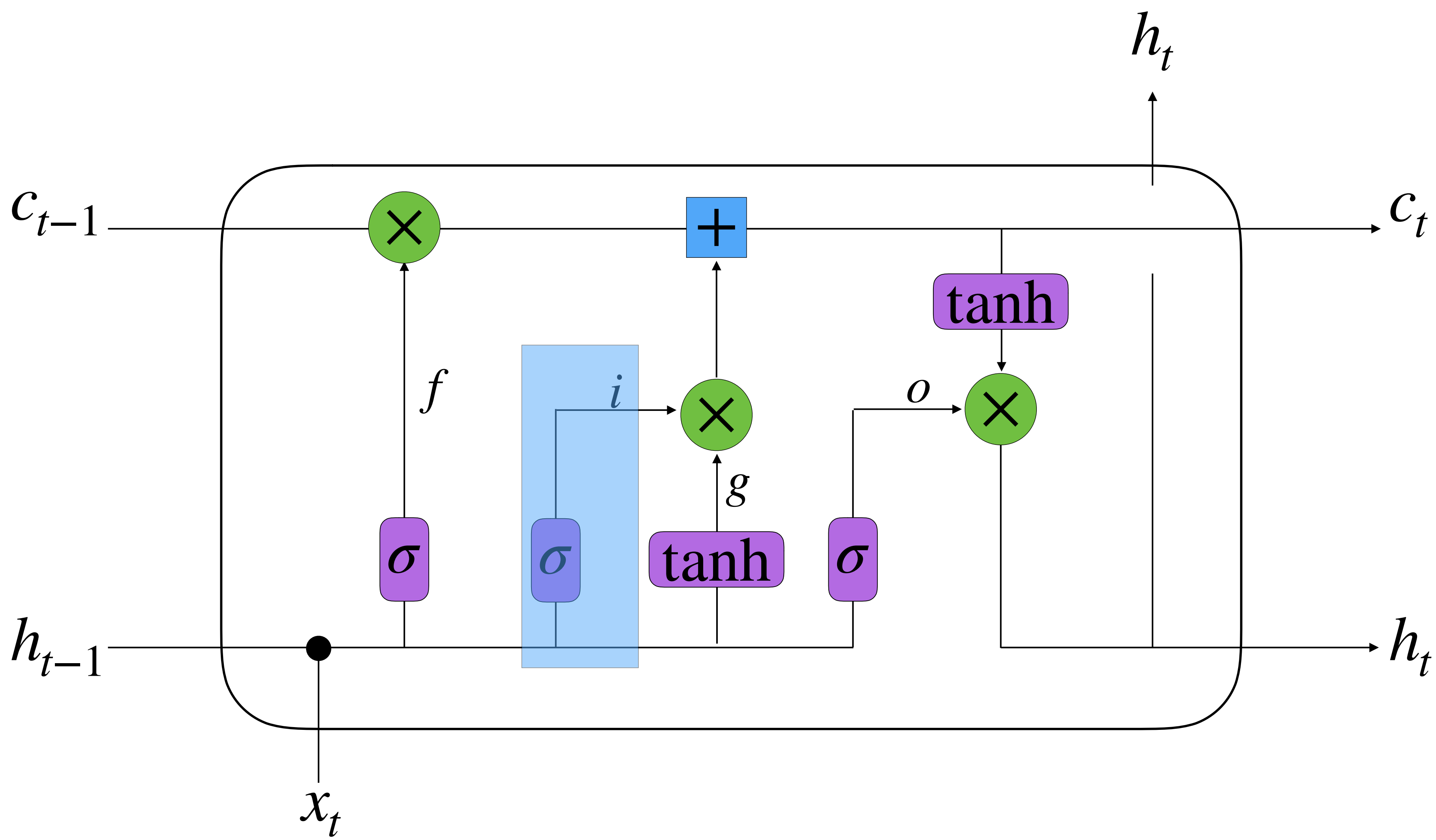
Element wise addition

Element wise multiplication

1 minus the input

Long-Short Term Memory (LSTM)

Input gate:
How much of
the previous
output
should be
remembered?



$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right) \qquad g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$

$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right) \qquad c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$

$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right) \qquad h_t = \tanh\left(c_t\right) \otimes o$$

www.bgoncalves.com

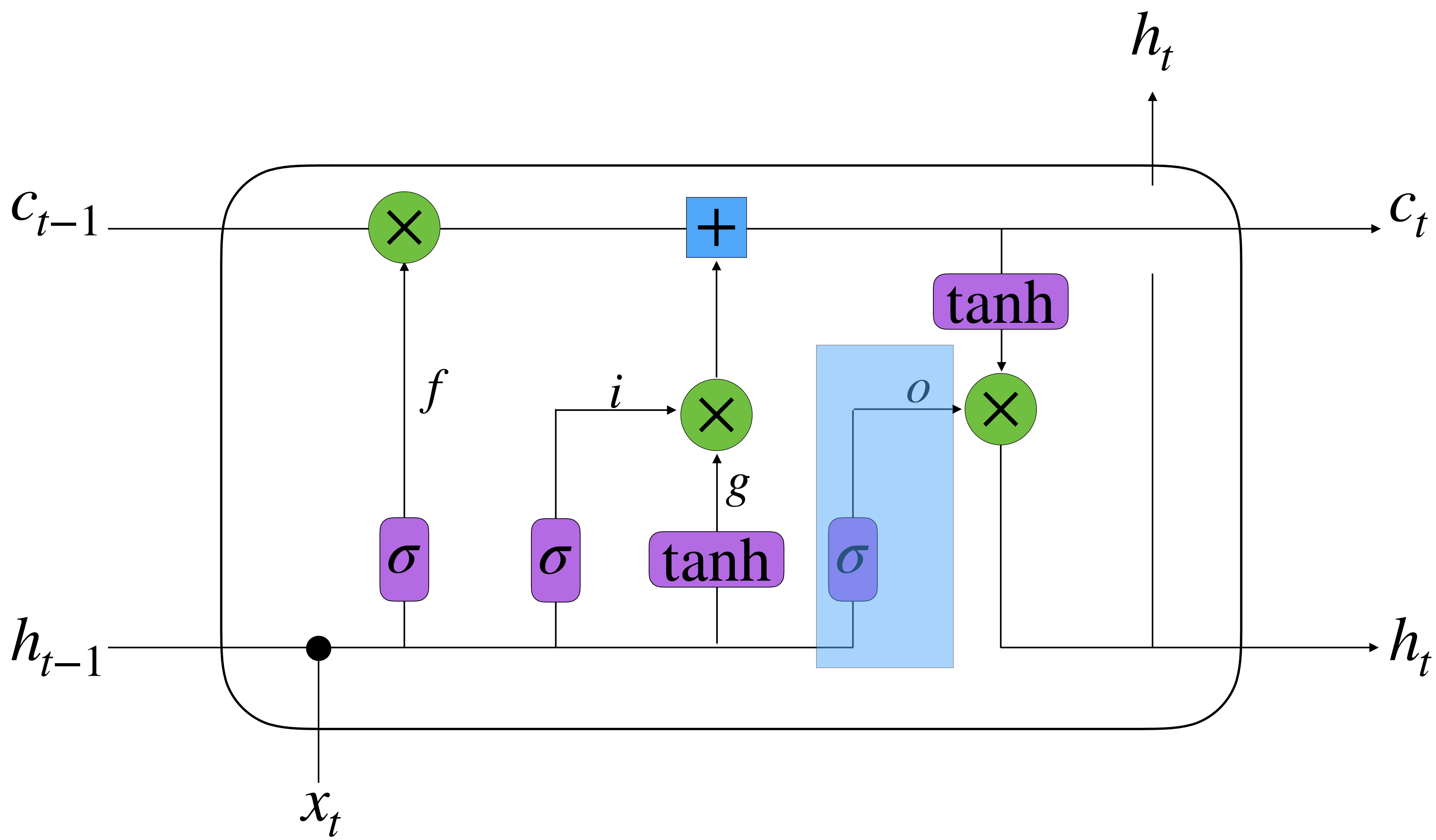
Long-Short Term Memory (LSTM)

Element wise addition

Element wise multiplication

1 minus the input

Output gate: How much of the previous output should contribute?



All gates use the same inputs and activation functions, but different weights

$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right) \qquad g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$

$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right) \qquad c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$

$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right) \qquad h_t = \tanh\left(c_t\right) \otimes o$$

www.bgoncalves.com

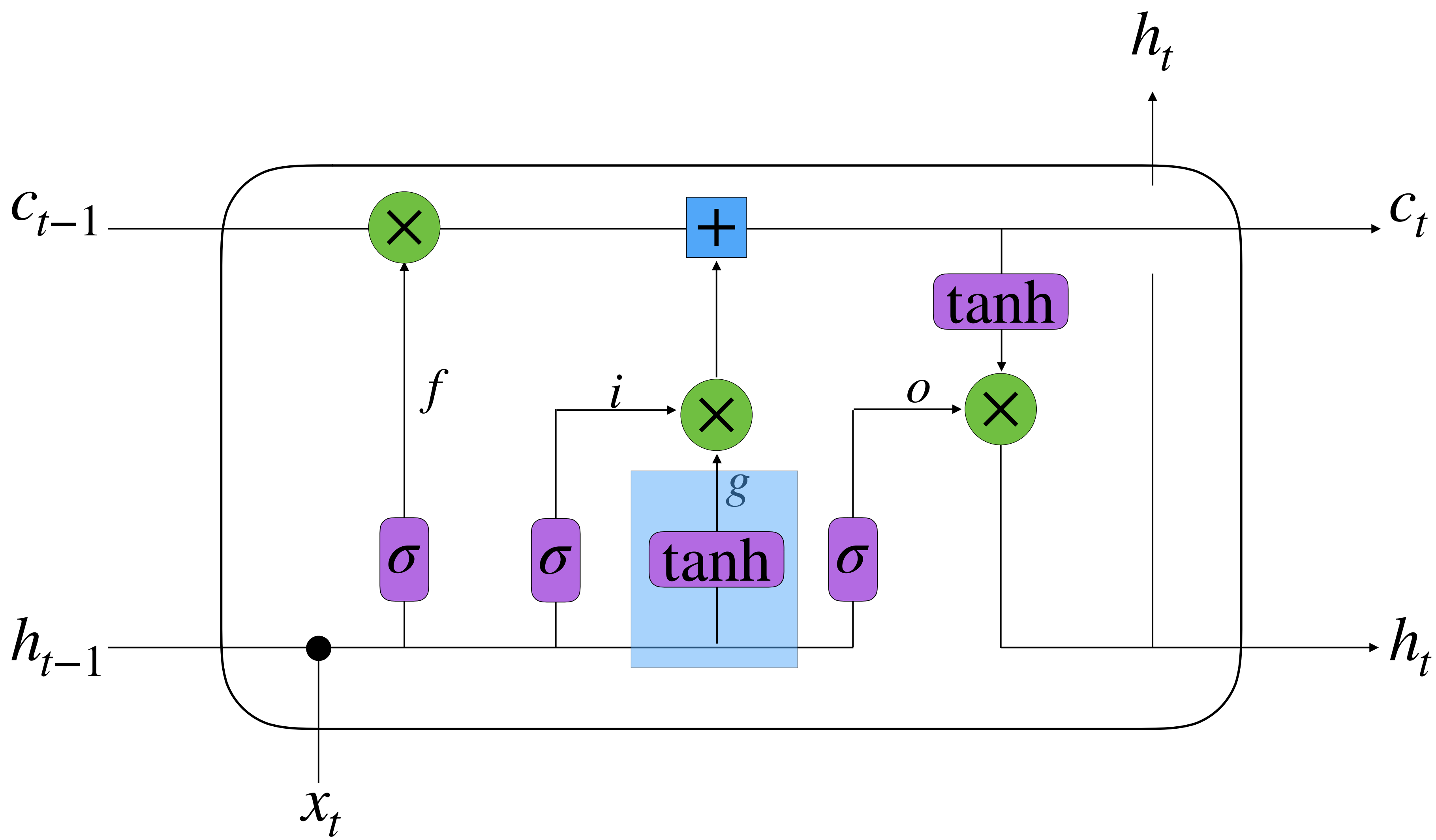
Long-Short Term Memory (LSTM)

+ Element wise addition

× Element wise multiplication

1− 1 minus the input

Output gate: How much of the previous output should contribute?



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

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

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

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

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

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

www.bgoncalves.com

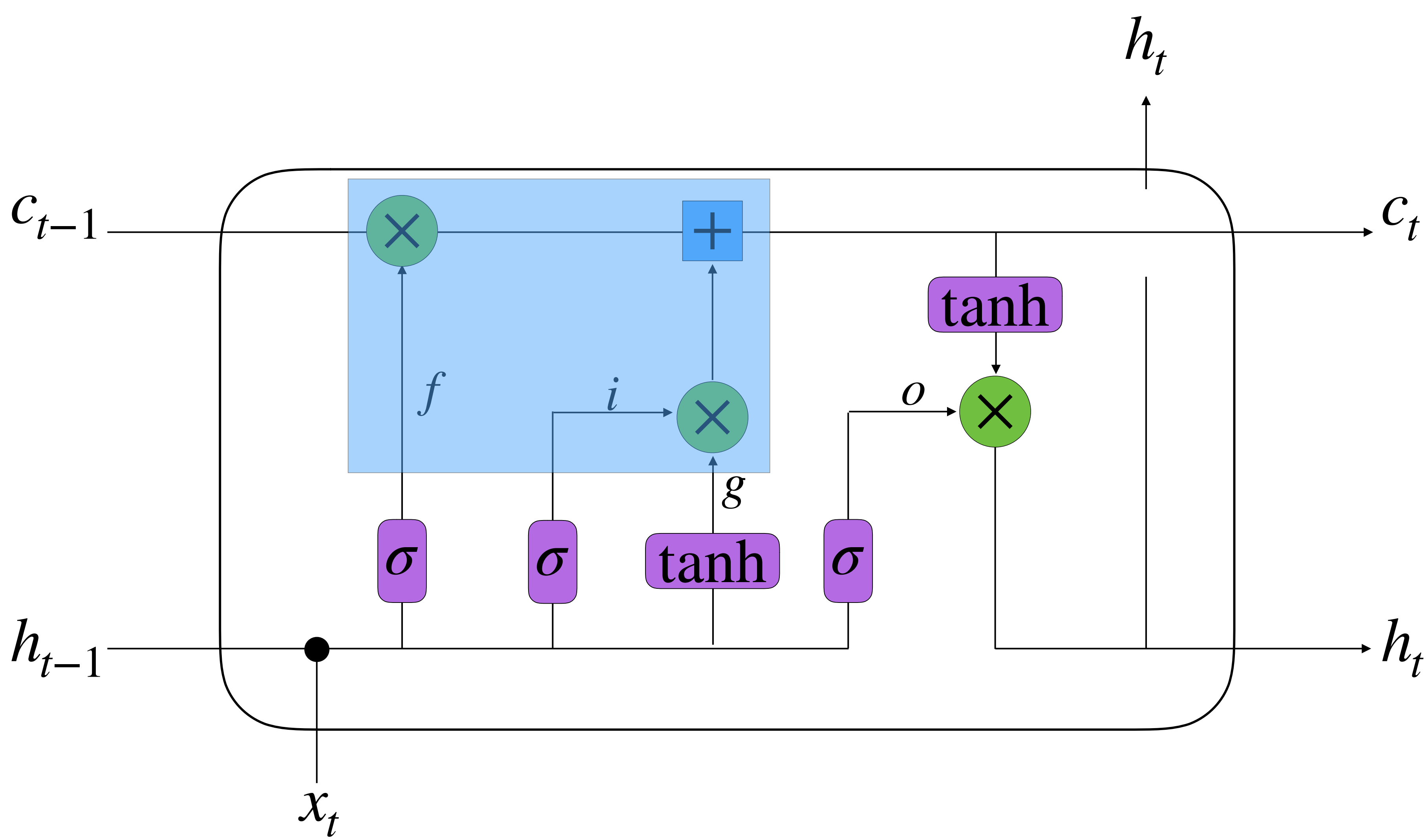
Long-Short Term Memory (LSTM)

Element wise addition

Element wise multiplication

1 minus the input

State:
Update the
current state



$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right) \qquad g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$

$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right) \qquad c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$

$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right) \qquad h_t = \tanh\left(c_t\right) \otimes o$$

www.bgoncalves.com

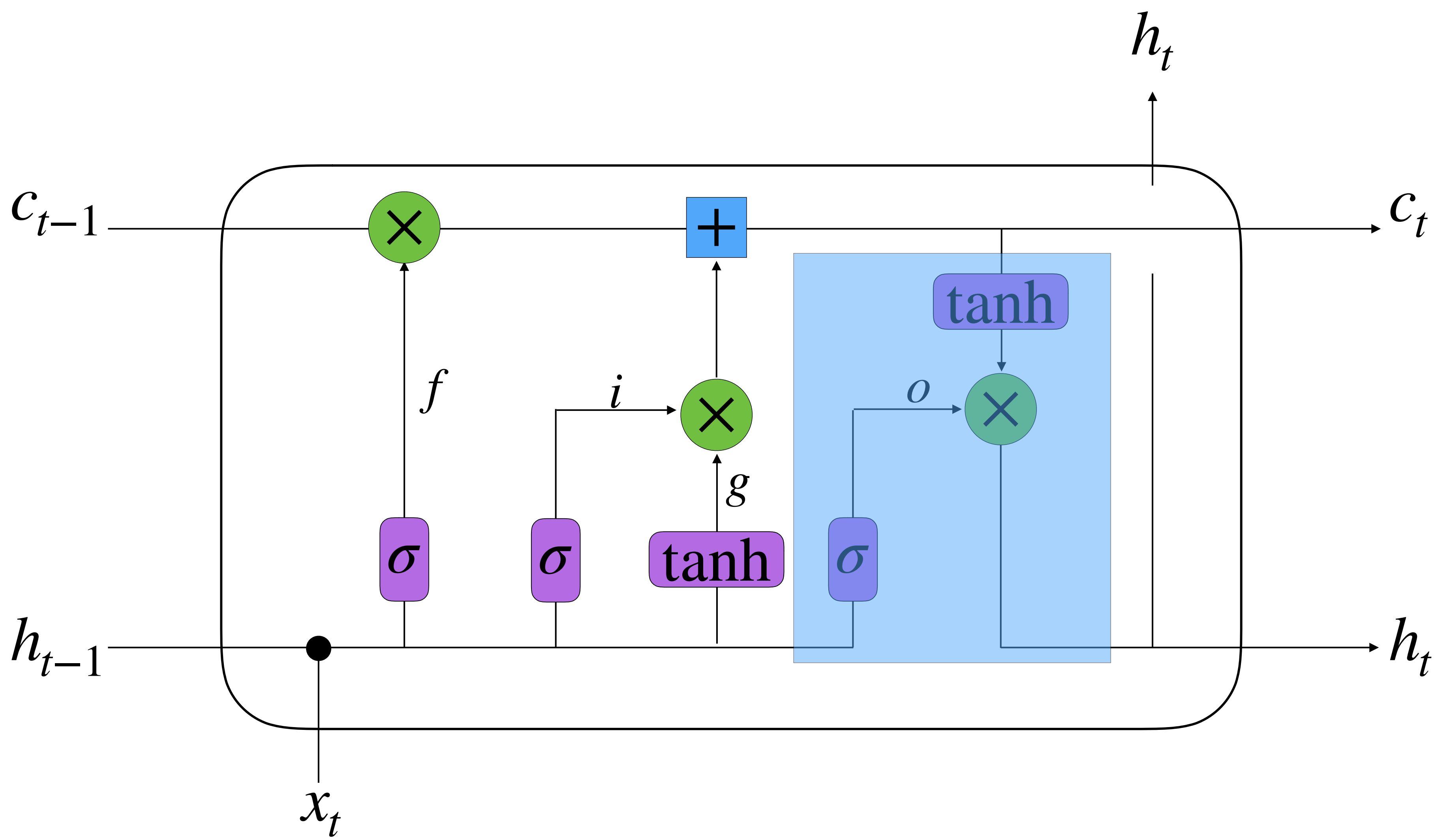
Element wise addition

Element wise multiplication

1 minus the input

Long-Short Term Memory (LSTM)

Output:
Combine all available information.



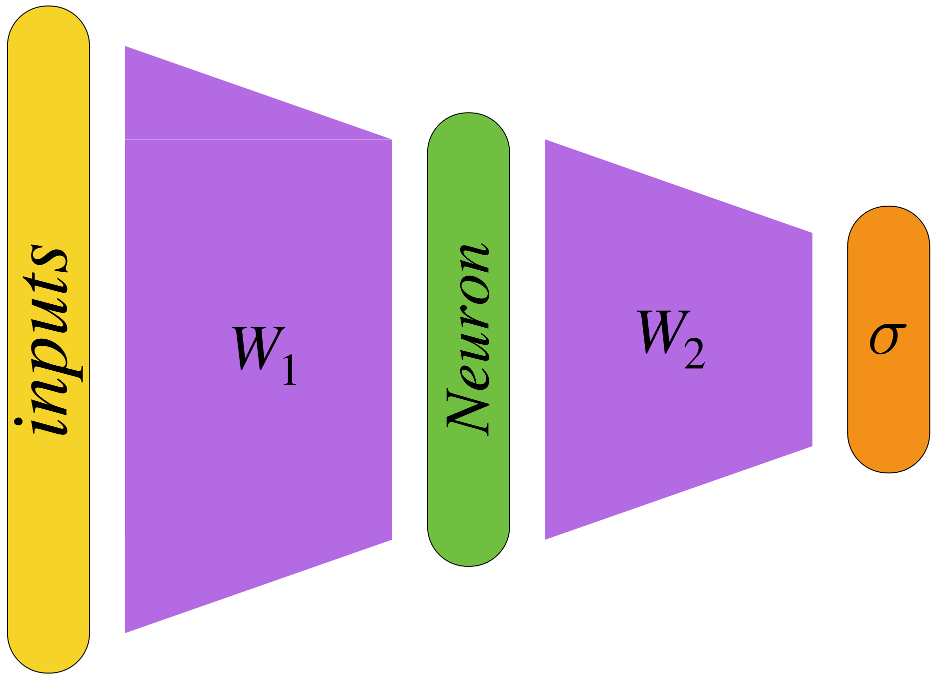
$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right) \qquad g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$

$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right) \qquad c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$

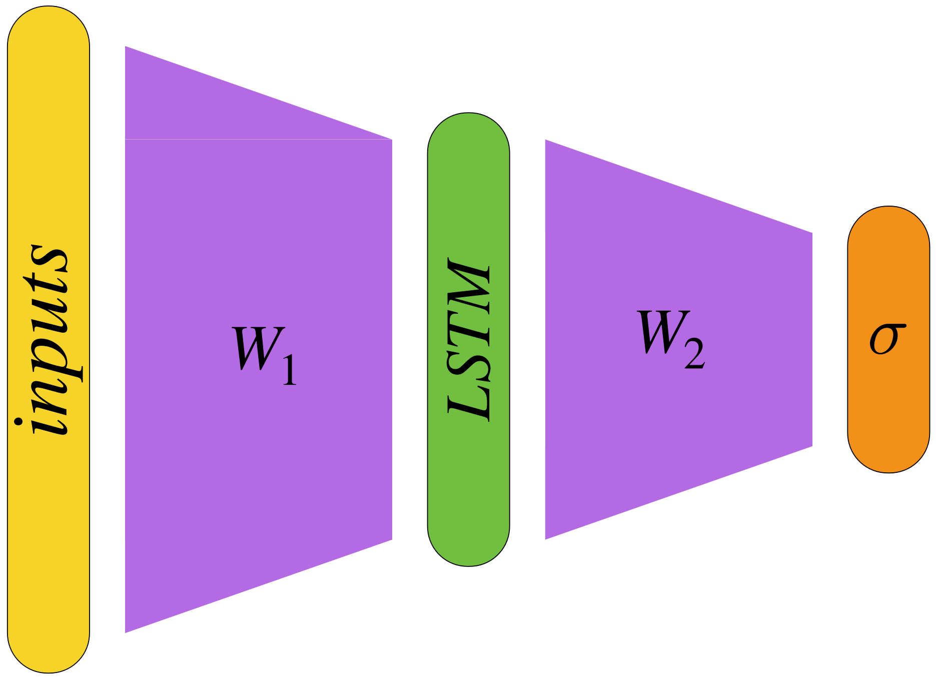
$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right) \qquad h_t = \tanh\left(c_t\right) \otimes o$$

www.bgoncalves.com

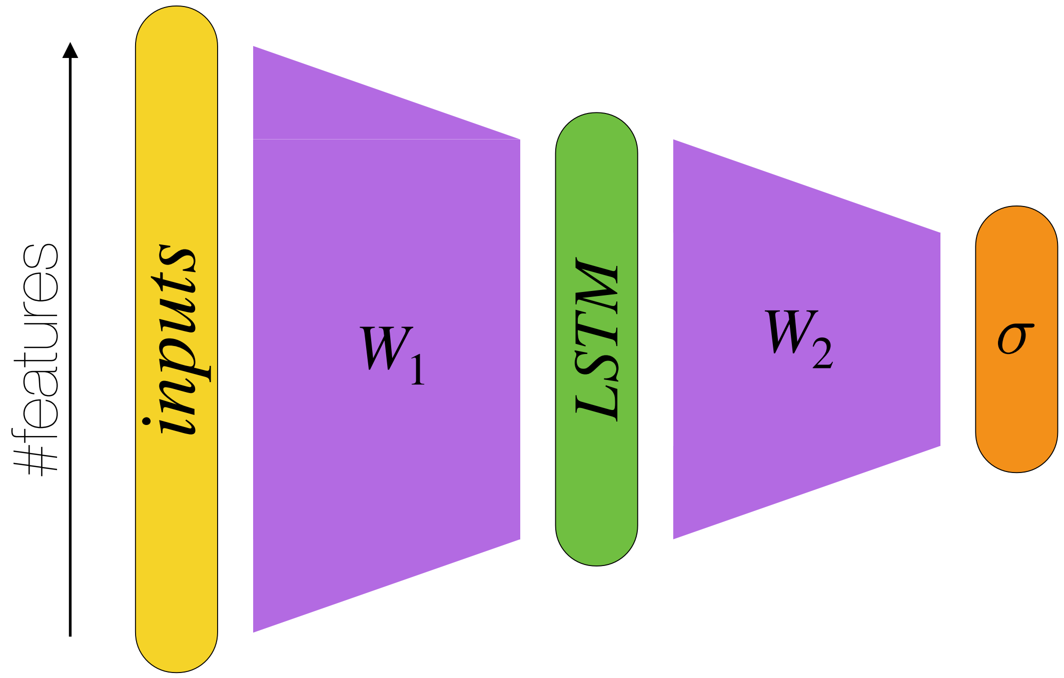
Using LSTMs



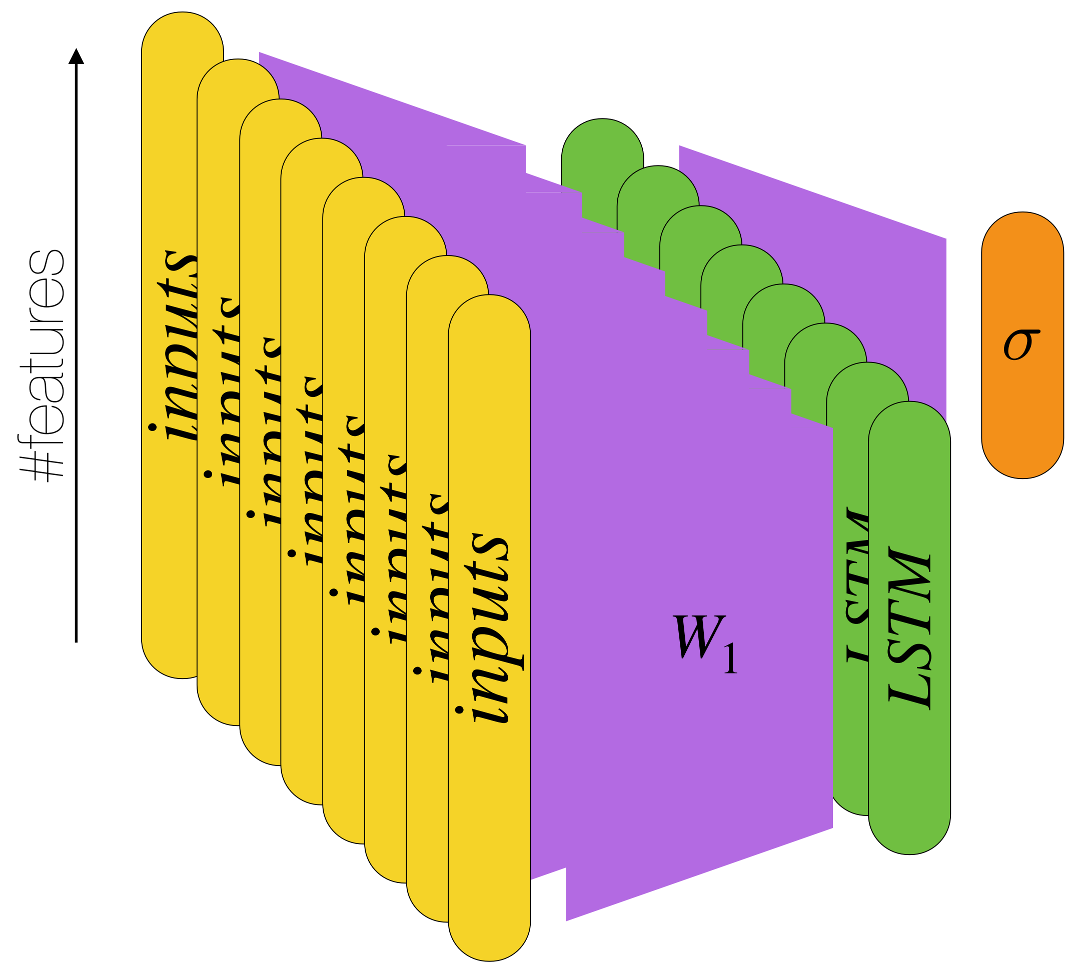
Using LSTMs



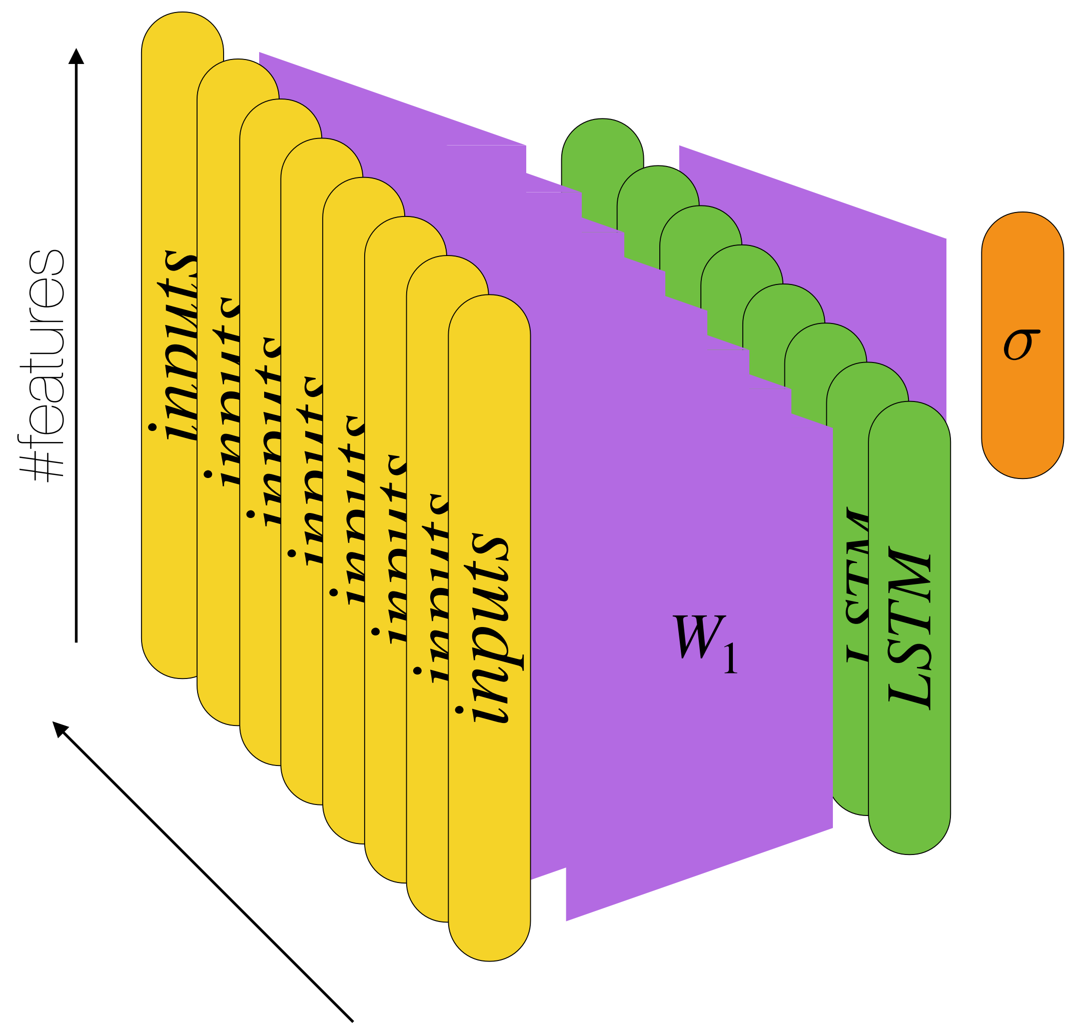
Using LSTMs



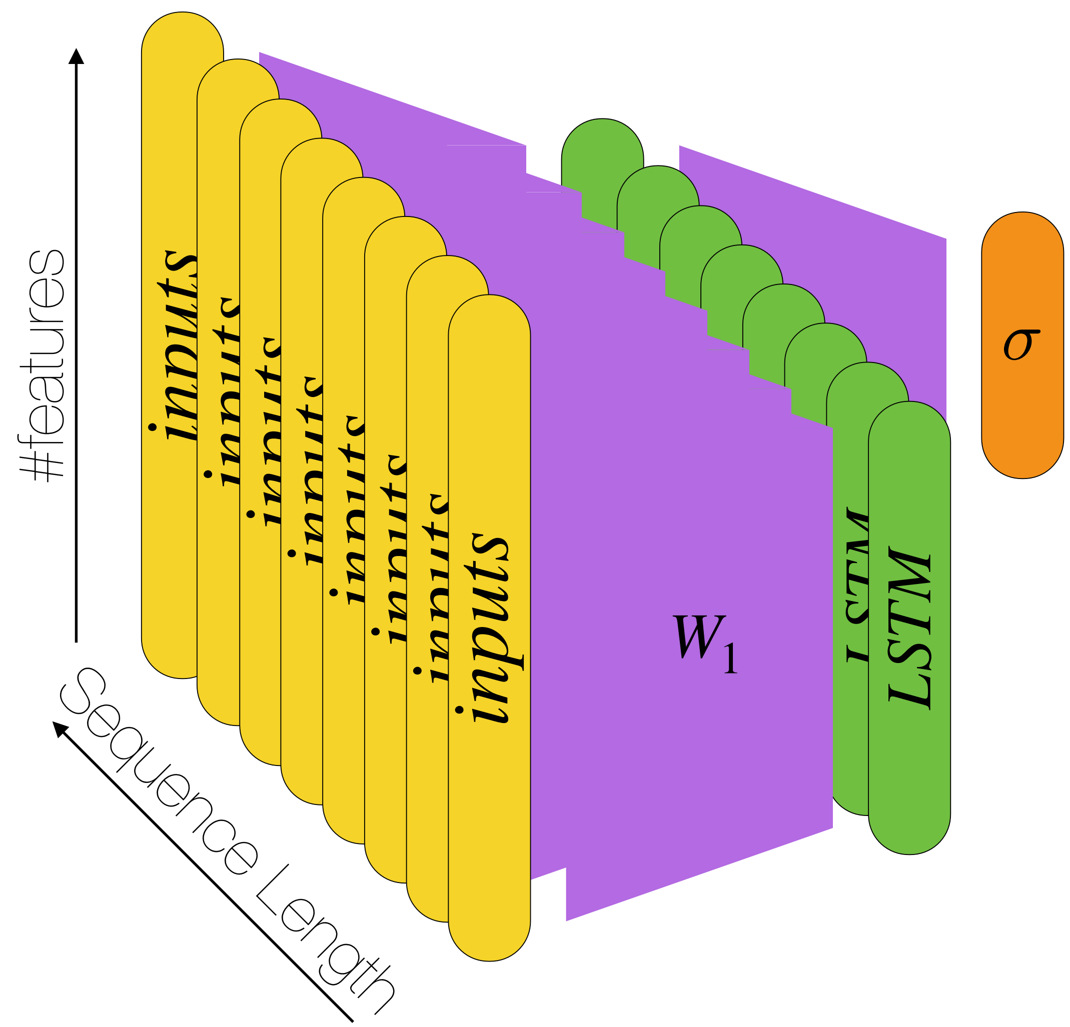
Using LSTMs


#features
inputs
inputs
inputs
inputs
inputs
inputs
inputs
inputs
W_1
LSTM
LSTM
σ

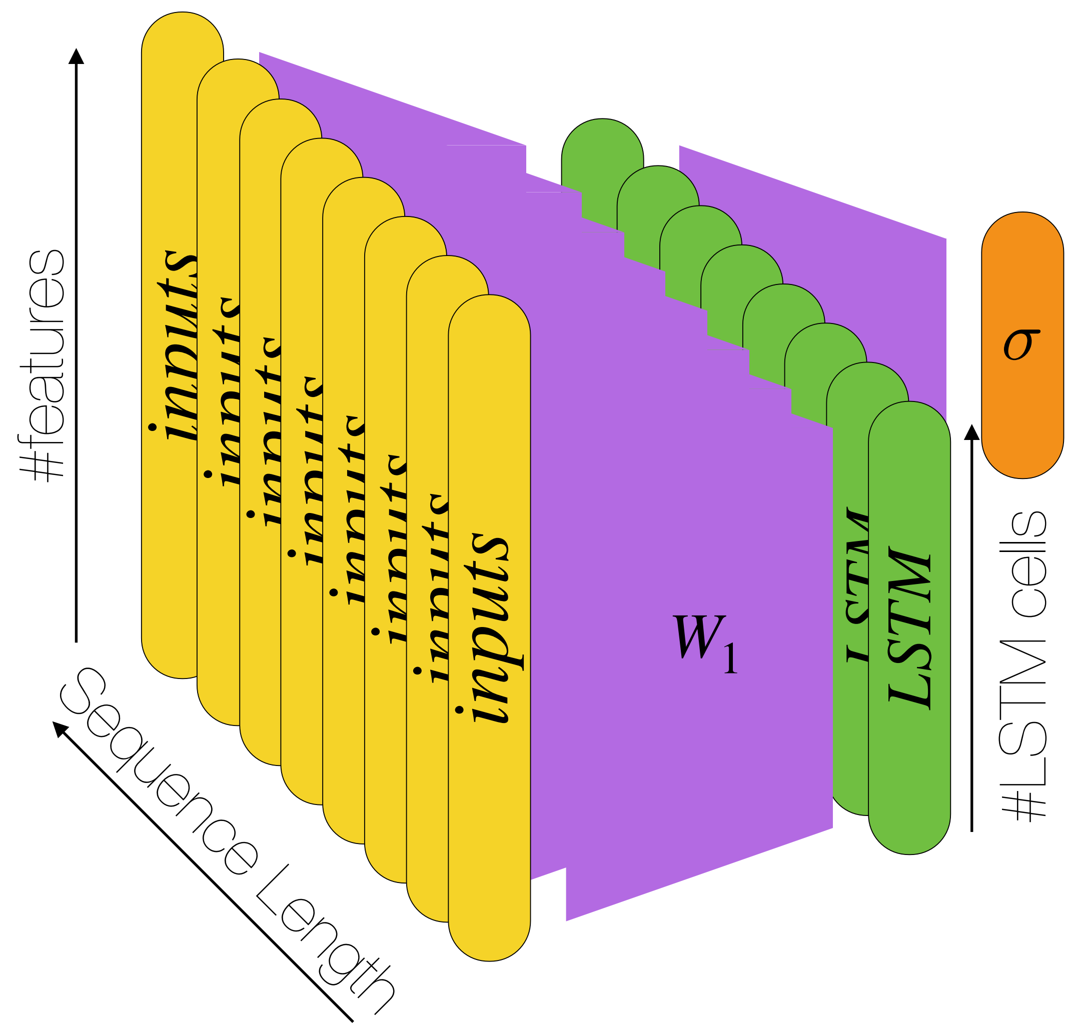
Using LSTMs



Using LSTMs


#features
Sequence Length
inputs
inputs
inputs
inputs
inputs
inputs
inputs
inputs
W_1
LSTM
LSTM
σ

Using LSTMs



#features
inputs
inputs
inputs
inputs
inputs
inputs
inputs
inputs
Sequence Length
W_1
LSTM
LSTM
#LSTM cells
σ

Applications

Applications

- Language Modeling and Prediction

Applications

- Language Modeling and Prediction
- Speech Recognition

Applications

- Language Modeling and Prediction
- Speech Recognition
- Machine Translation

Applications

- Language Modeling and Prediction
- Speech Recognition
- Machine Translation
- Part-of-Speech Tagging

Applications

- Language Modeling and Prediction
- Speech Recognition
- Machine Translation
- Part-of-Speech Tagging
- Sentiment Analysis

Applications

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

Applications

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

Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU)

- Introduced in 2014 by Cho

Gated Recurrent Unit (GRU)

- Introduced in 2014 by Cho
- Meant to solve the Vanishing Gradient Problem

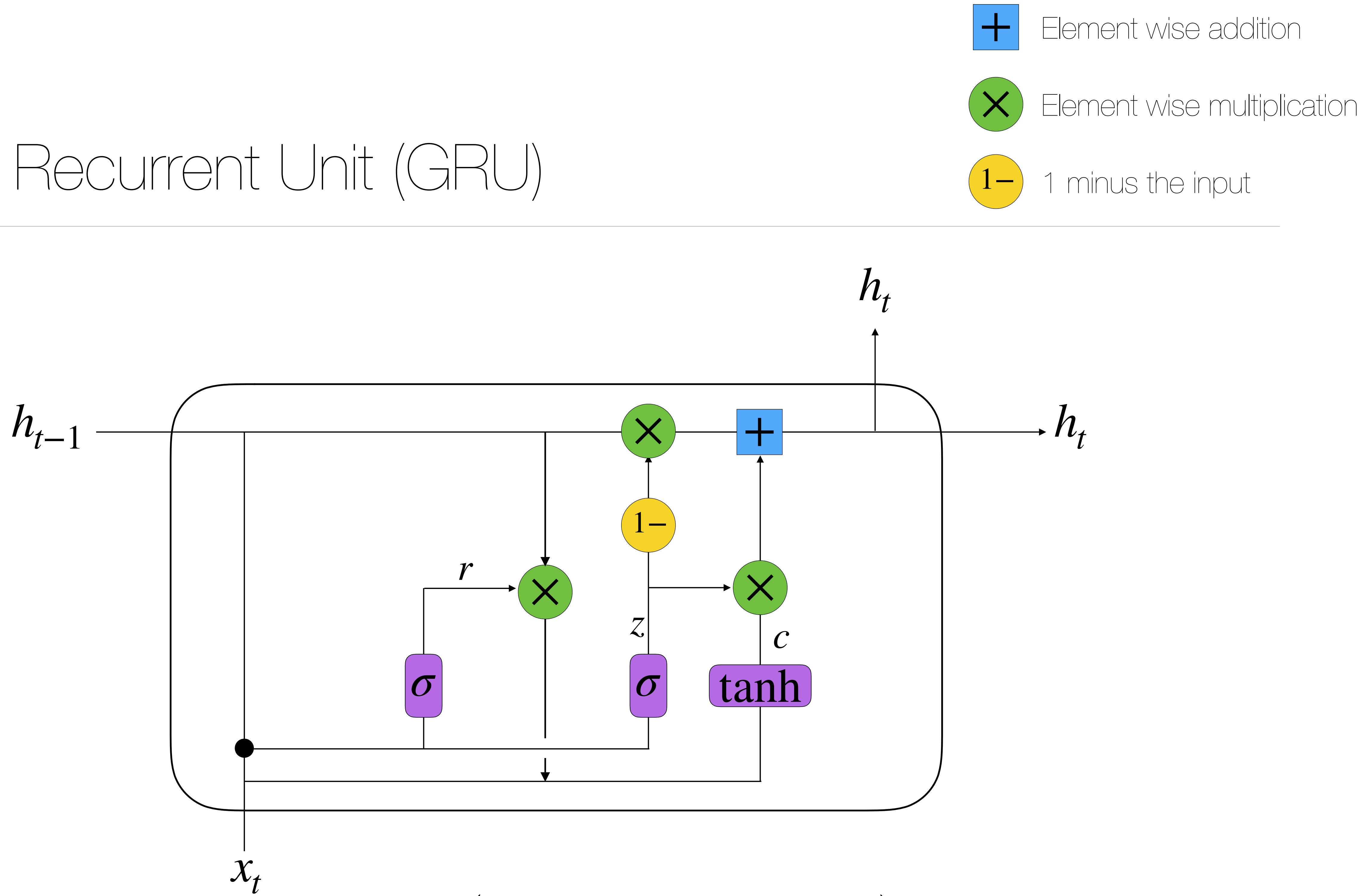
Gated Recurrent Unit (GRU)

- Introduced in 2014 by Cho
- Meant to solve the Vanishing Gradient Problem
- Can be considered as a simplification of LSTMs

Gated Recurrent Unit (GRU)

- Introduced in 2014 by 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)



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

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

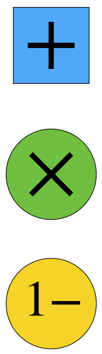
$$c = \tanh\left(W_c\left(h_{t-1} \otimes r\right) + U_c x_t\right)$$

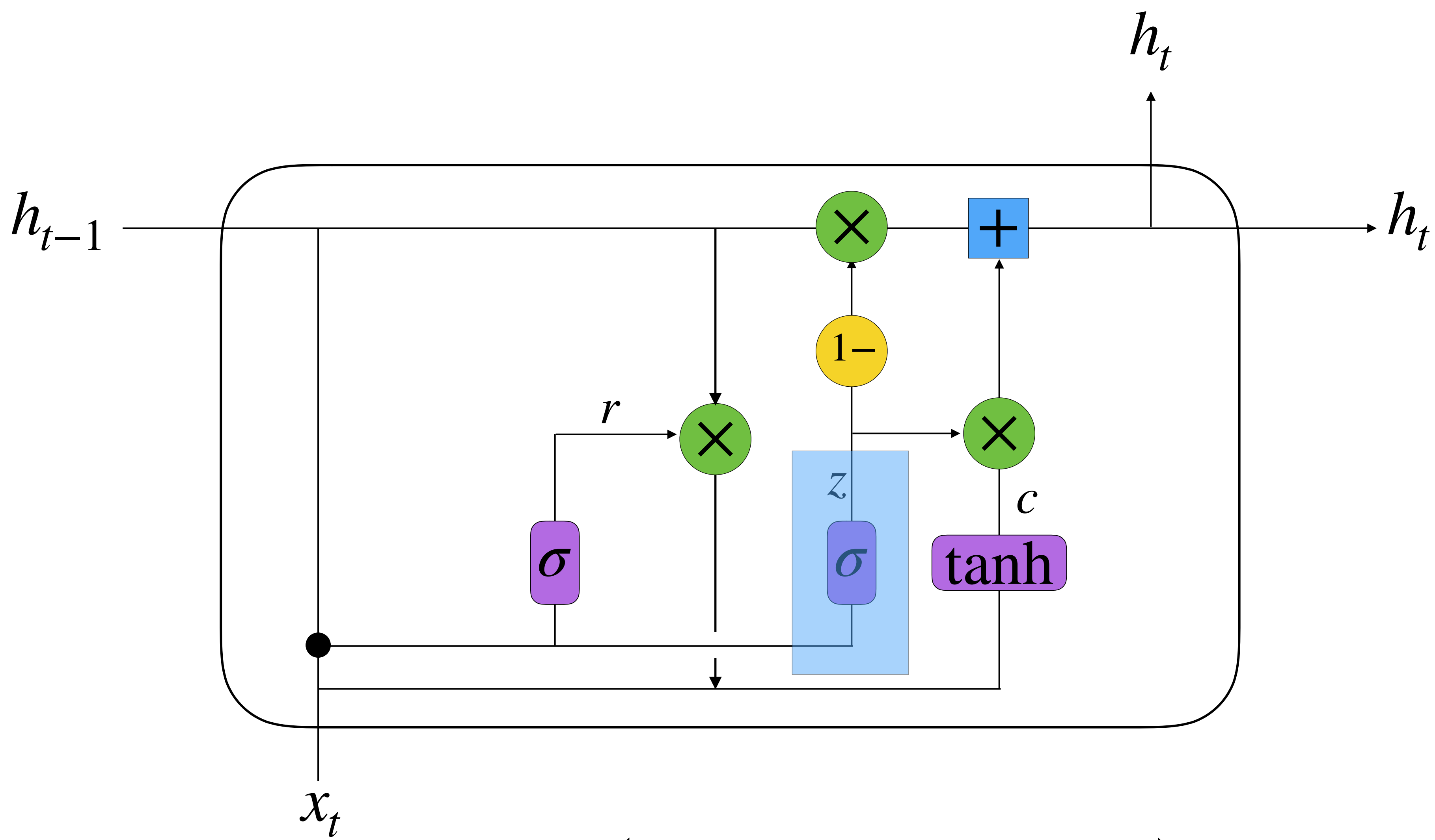
$$h_t = (z \otimes c) + \left((1 - z) \otimes h_{t-1}\right)$$

@bgoncalves

www.bgoncalves.com

Gated Recurrent Unit (GRU)





Update gate: How much of the previous state should be kept?

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

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

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

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

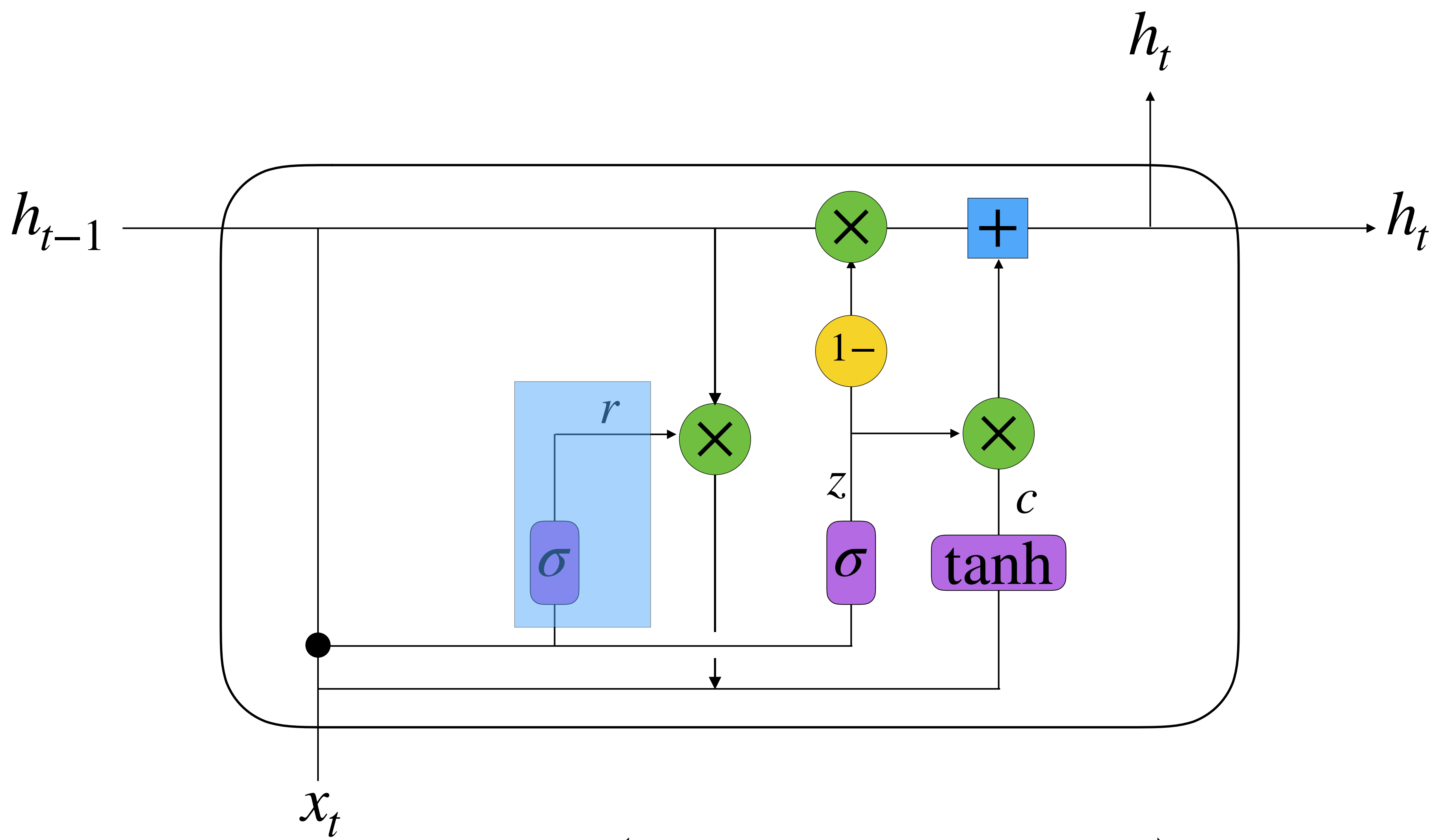
Gated Recurrent Unit (GRU)

+ Element wise addition

× Element wise multiplication

1− 1 minus the input

Reset gate:
How much of
the previous
output should
be removed?



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

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

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

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

@bgoncalves

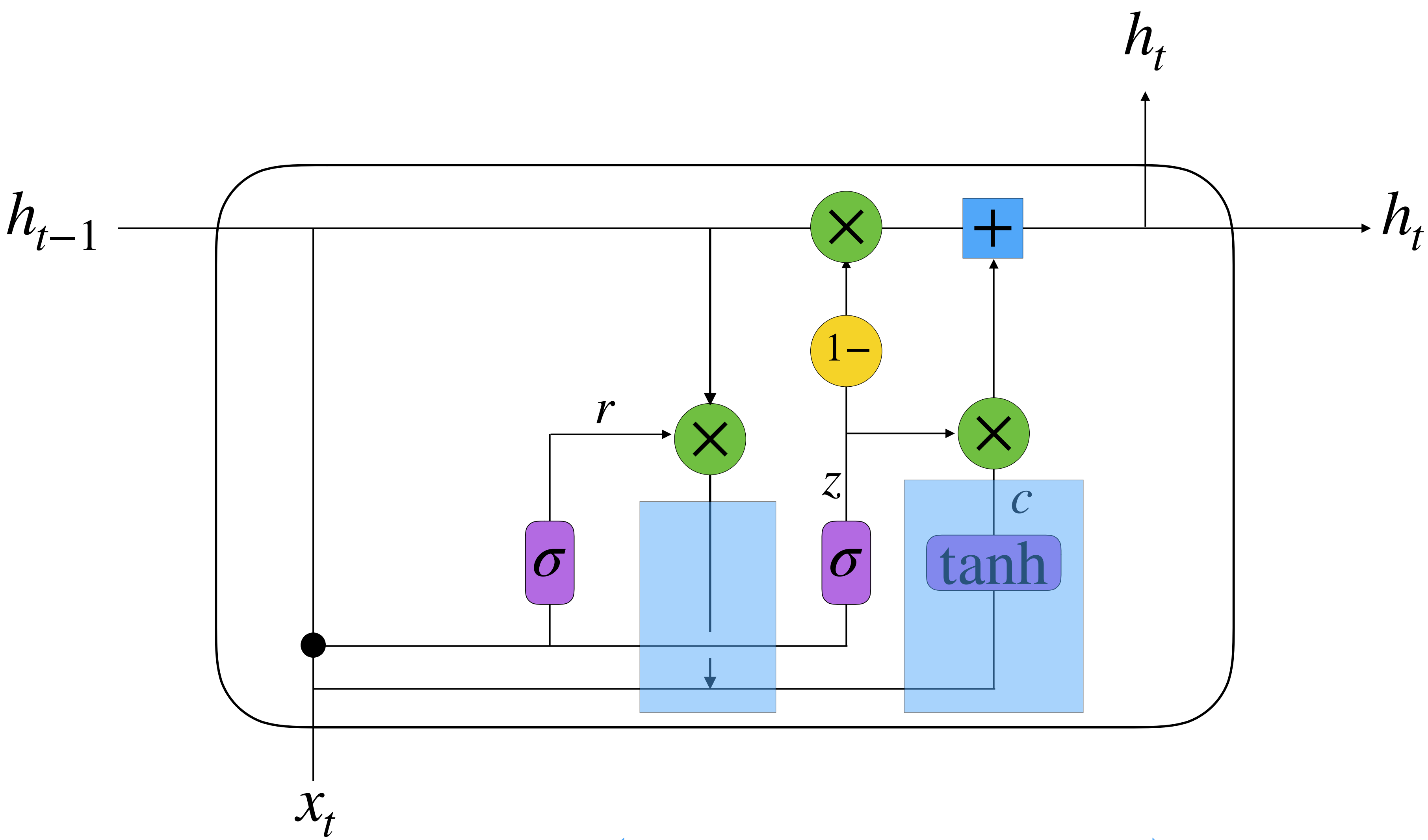
www.bgoncalves.com

Gated Recurrent Unit (GRU)

Element wise addition

Element wise multiplication

1 minus the input



Current memory: What information do we remember right now?

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

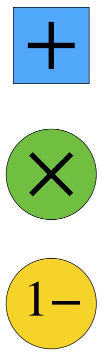
$$z = \sigma\left(W_z h_{t-1} + U_z x_t\right)$$

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

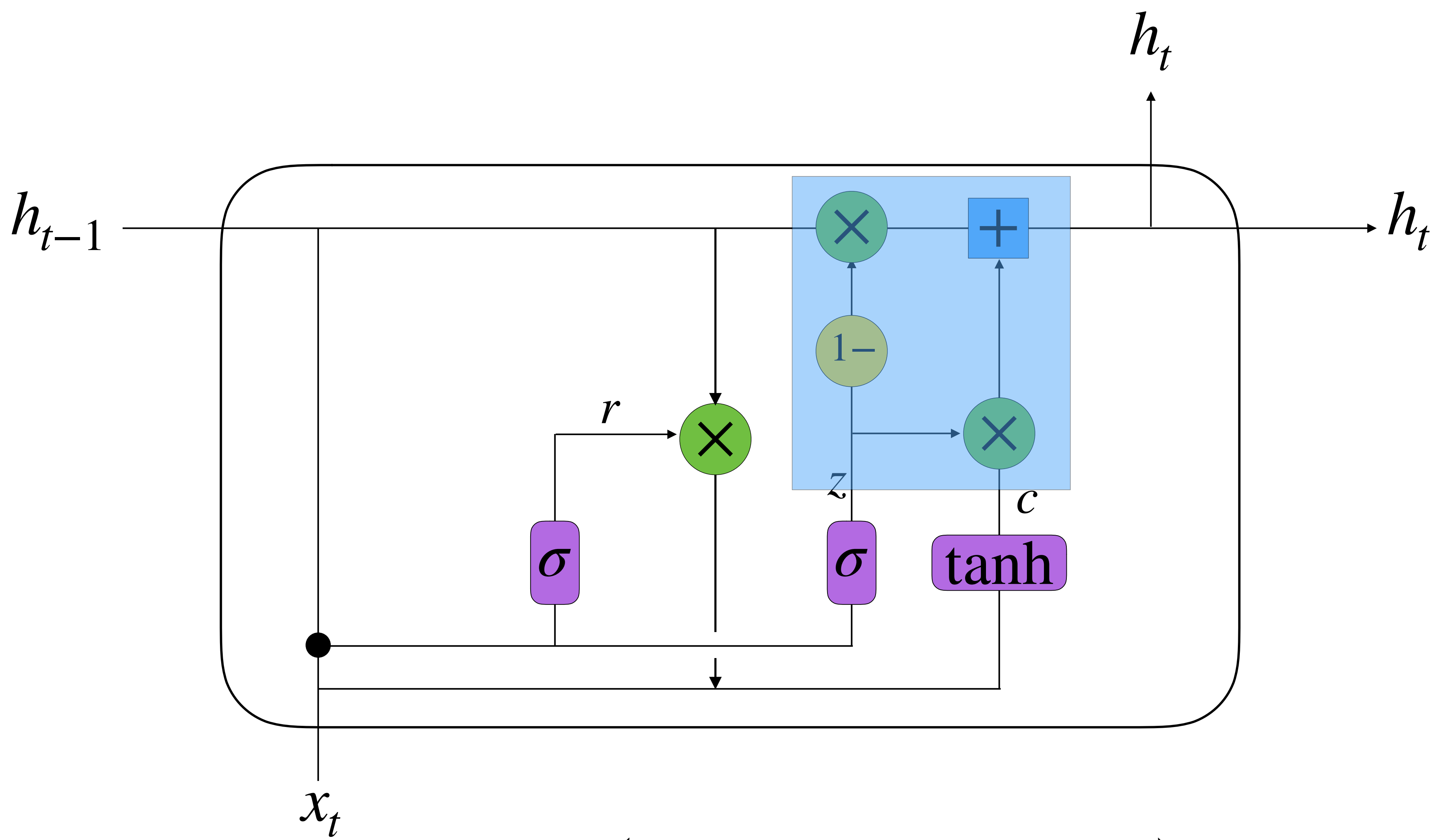
$$h_t = (z \otimes c) + \left((1 - z) \otimes h_{t-1}\right)$$

@bgoncalves www.bgoncalves.com

Gated Recurrent Unit (GRU)



Output:
Combine all
available
information.



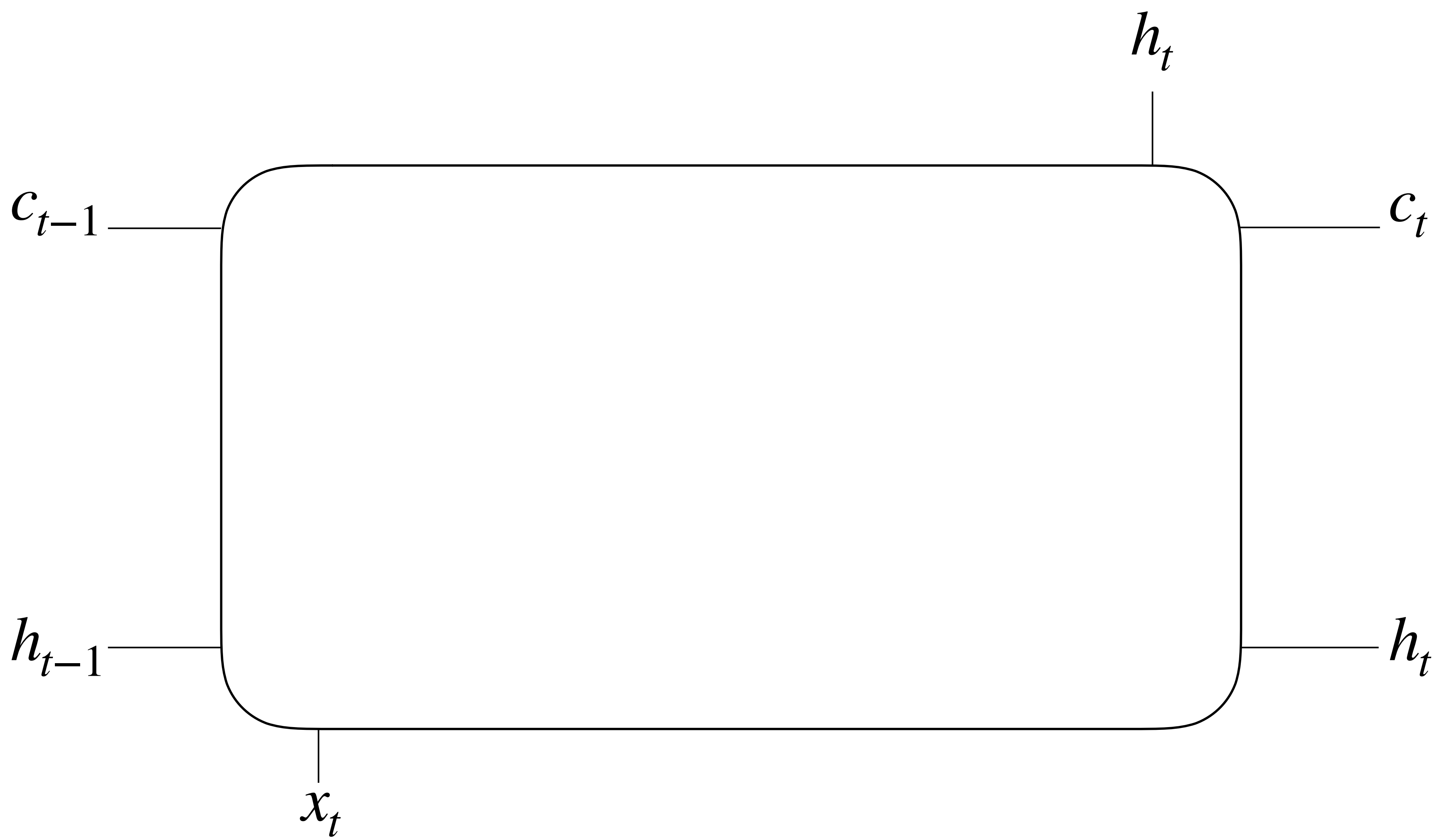
$$z = \sigma\left(W_z h_{t-1} + U_z x_t\right)$$

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

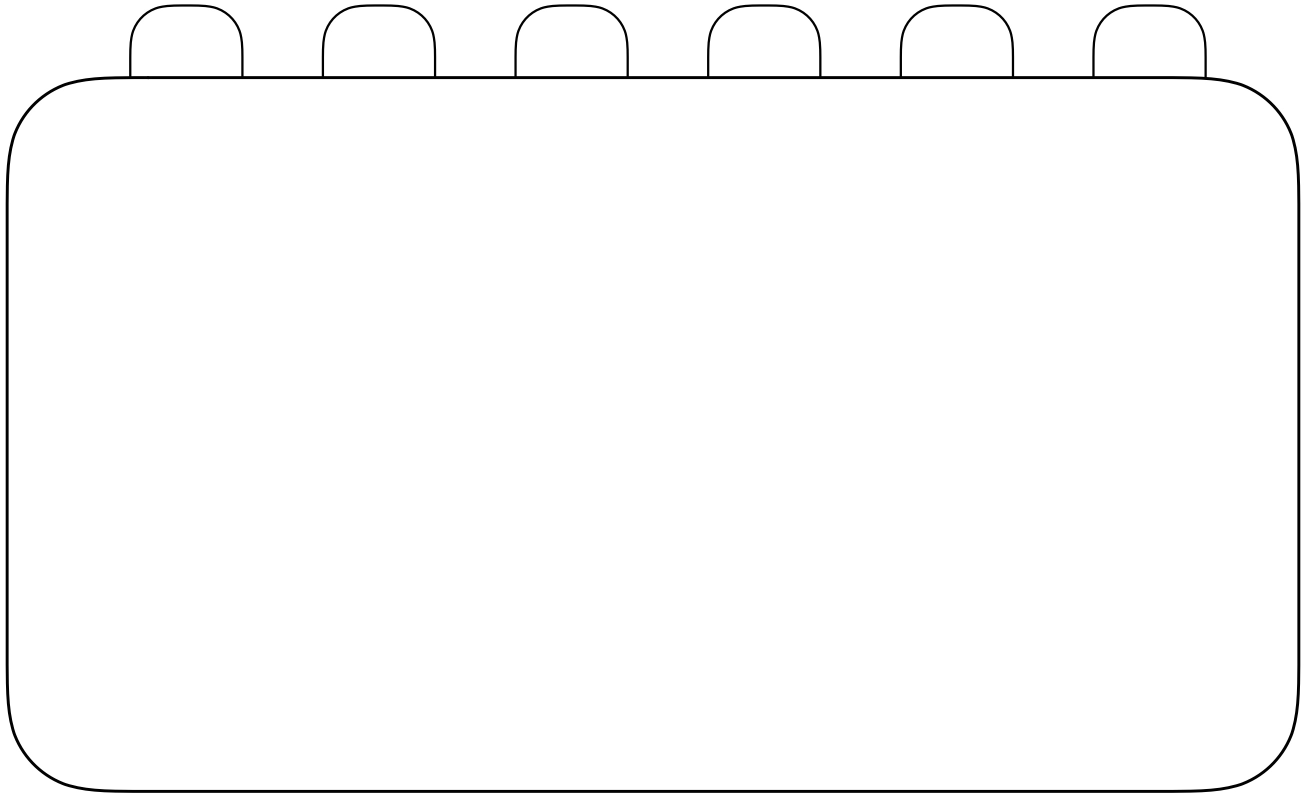
$$c = \tanh\left(W_c\left(h_{t-1} \otimes r\right) + U_c x_t\right)$$

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

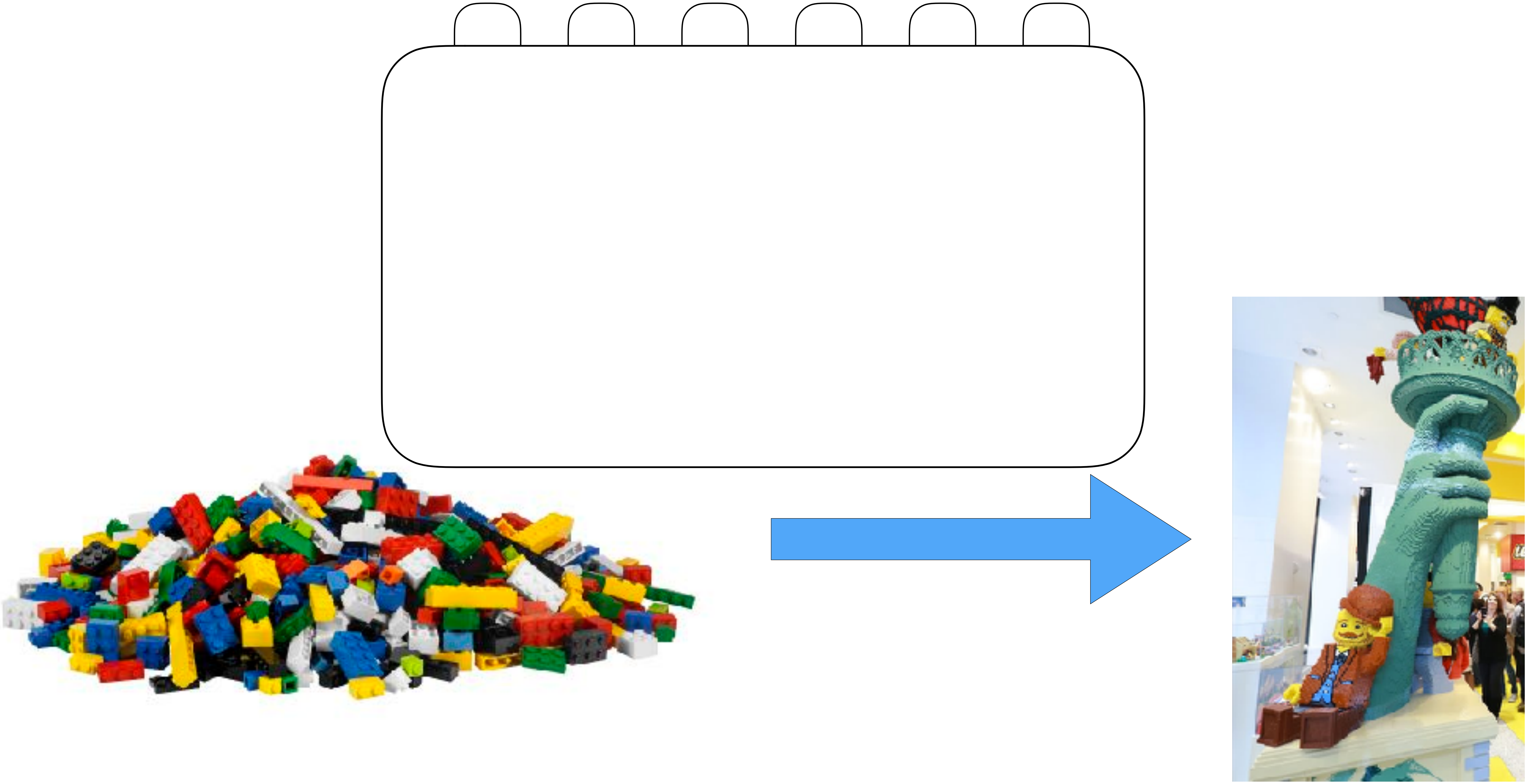
Neural Networks?



Or legos?



Or legos?



Or legos?

https://keras.io

Keras

@bgoncalves

www.bgoncalves.com

Keras

https://keras.io

Keras

https://keras.io

- Open Source neural network library written in Python

Keras

https://keras.io

- Open Source neural network library written in Python

- TensorFlow, Microsoft Cognitive Toolkit or Theano backends

Keras

https://keras.io

- Open Source neural network library written in Python
- TensorFlow, Microsoft Cognitive Toolkit or Theano backends
- Enables fast experimentation

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.

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

https://keras.io

Keras

https://keras.io

- 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

https://keras.io

- 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

Keras

https://keras.io

- 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.

Keras

https://keras.io

- 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)

Keras

https://keras.io

- 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.

Keras

https://keras.io

- 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)

Keras

https://keras.io

- 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)

Keras

https://keras.io

- 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)
    - LSTM(units, input_shape, activation='tanh', use_bias=True, dropout=0.0, return_sequences=False)

Keras

https://keras.io

Keras

https://keras.io

- model = Sequential()

Keras

https://keras.io

- model = Sequential()
- model.add(layer) - Add a layer to the top of the model

Keras

https://keras.io

- 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

Keras

https://keras.io

- 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…

Keras

https://keras.io

- 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…

Keras

https://keras.io

- 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)

Keras

https://keras.io

- 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 similar to sklearn.

Keras

https://keras.io

- 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 similar to sklearn.
- model.summary() - Output a textual representation of the model

github.com/bmtgoncalves/RNN