COMPUTATIONAL ASTROPHYSICS

Observatorio
Astronómico
Nacional

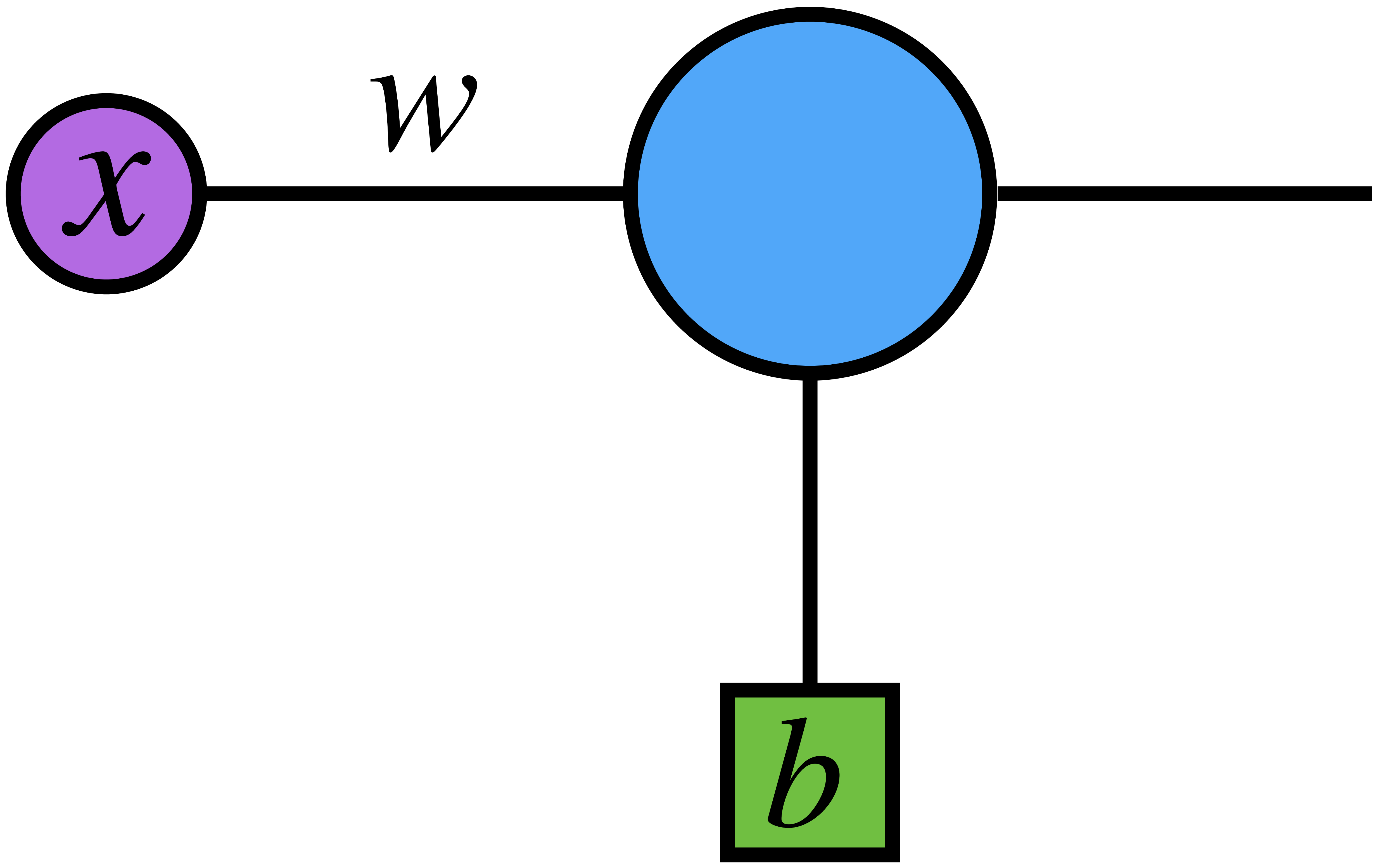
Computational Astrophysics

Neural Networks. Classification

Eduard Larrañaga
Observatorio Astronómico Nacional
Universidad Nacional de Colombia

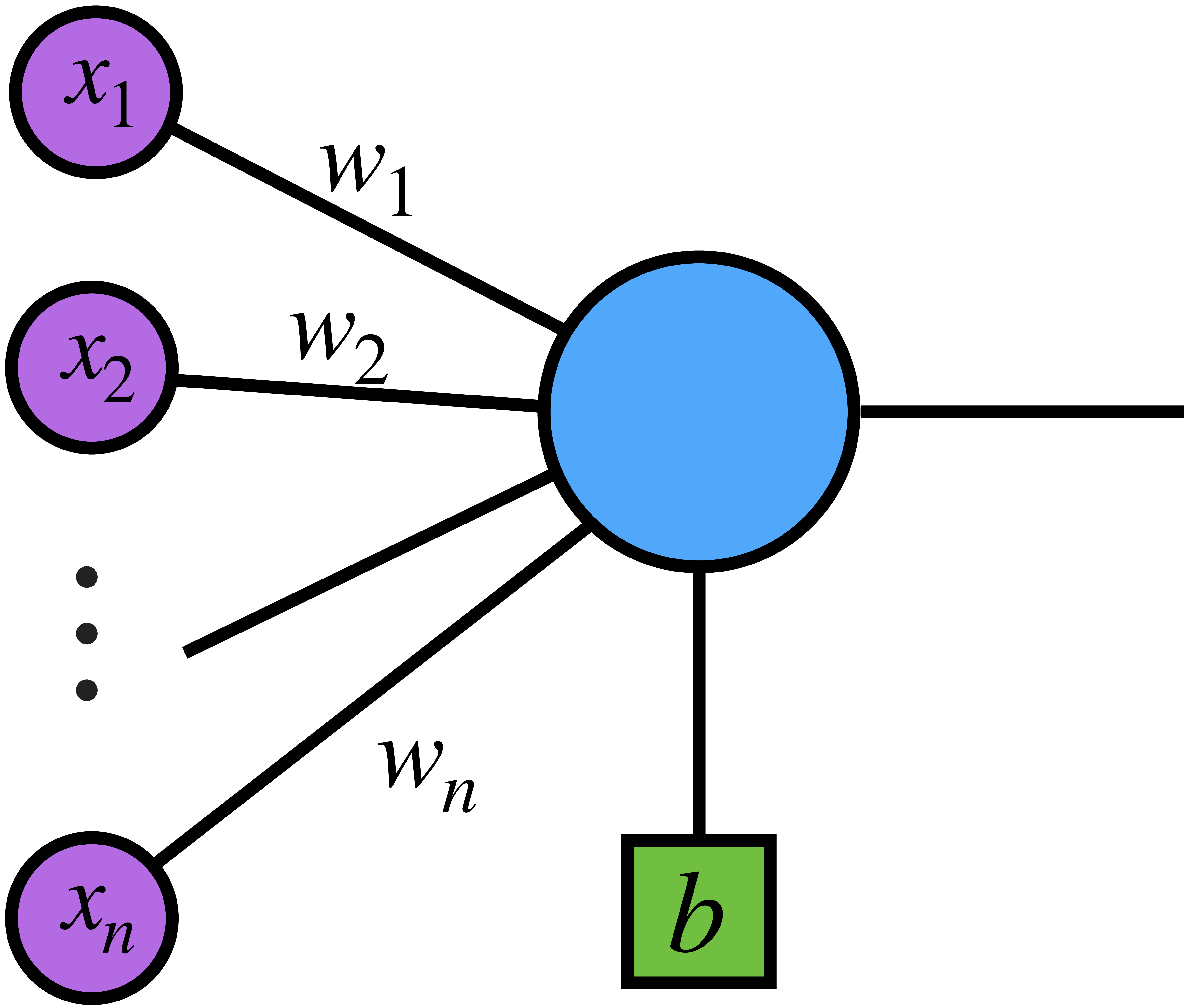
A Perceptron

Perceptron

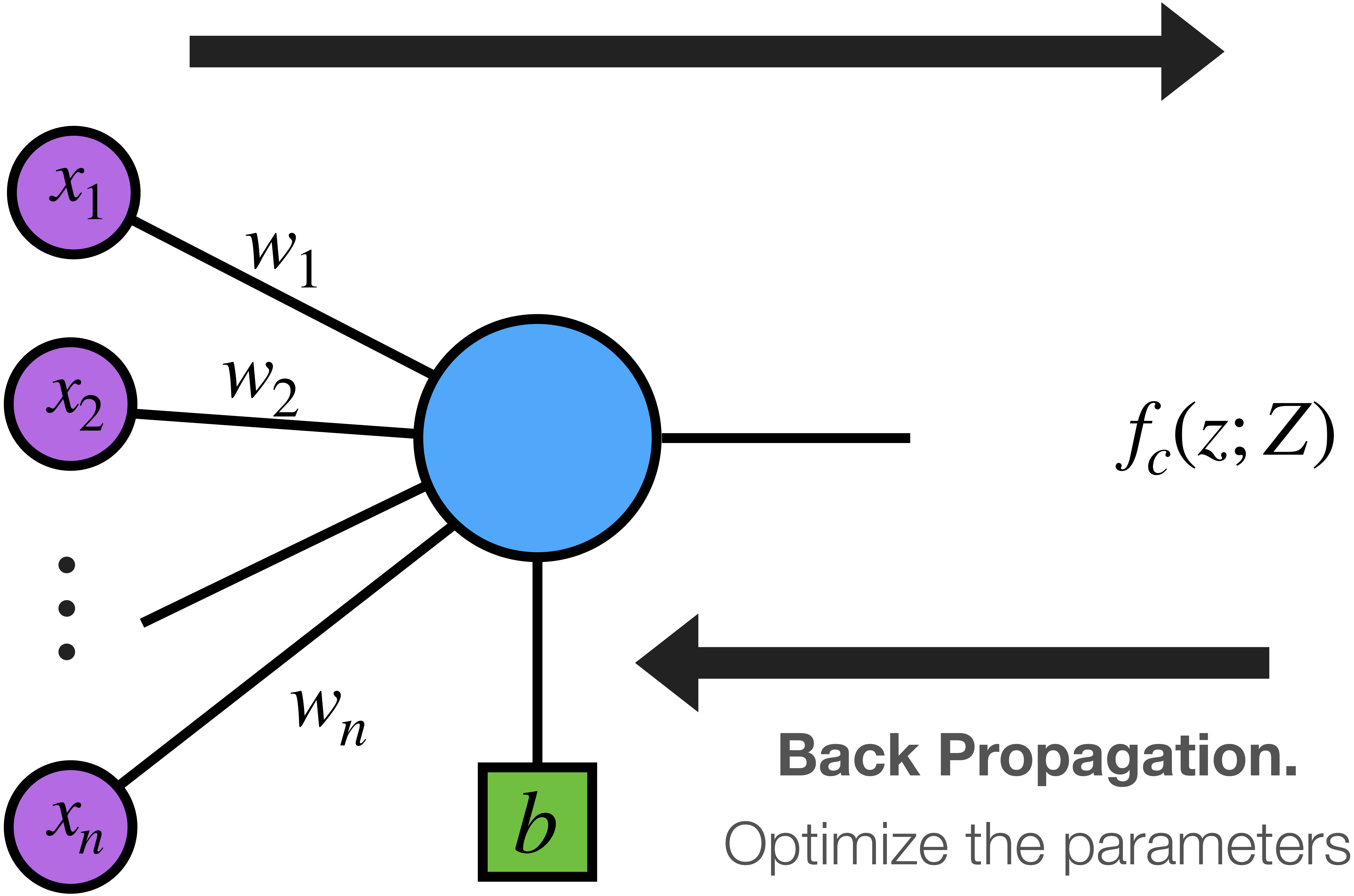


$$z = wx + b$$

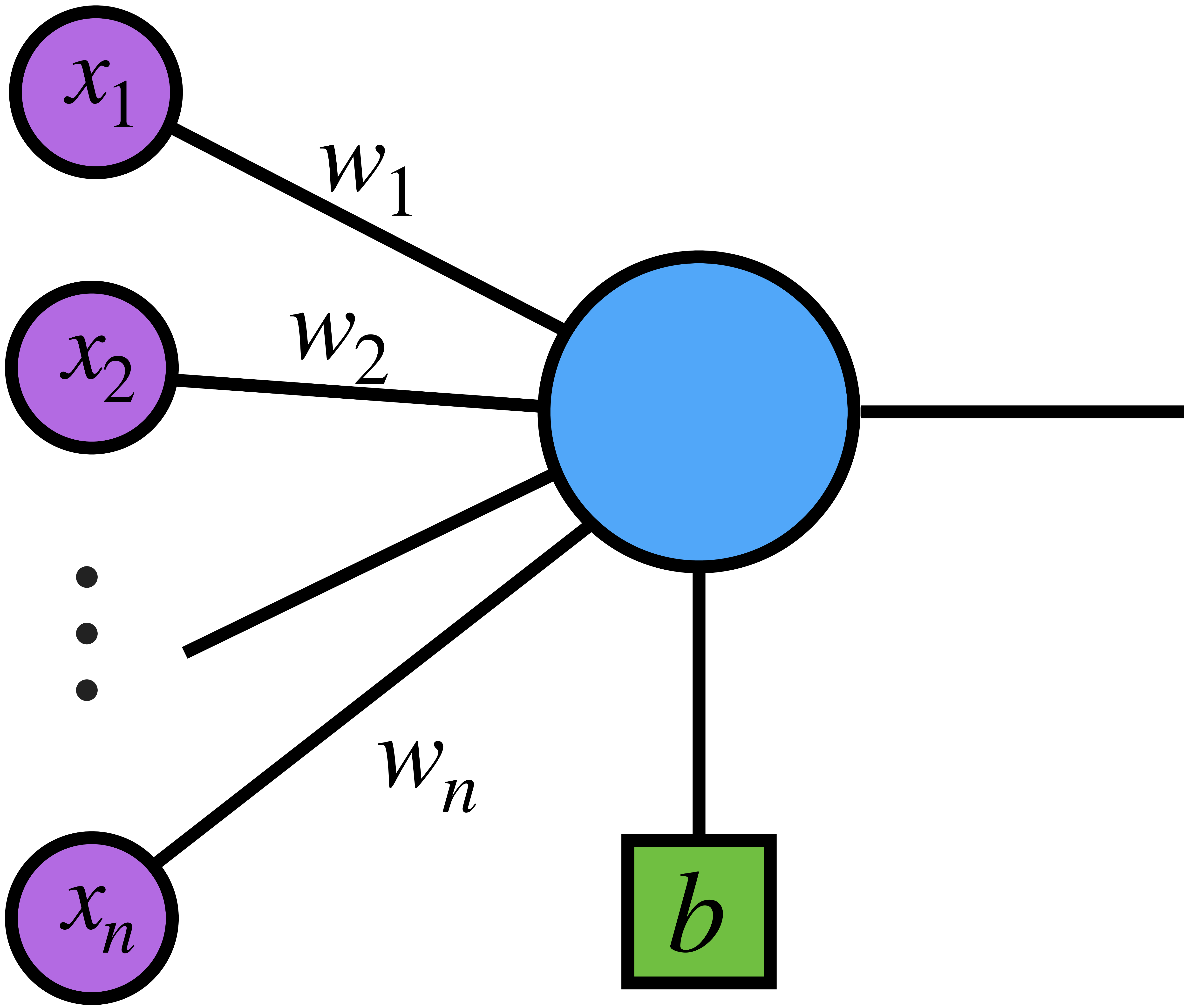
Perceptron



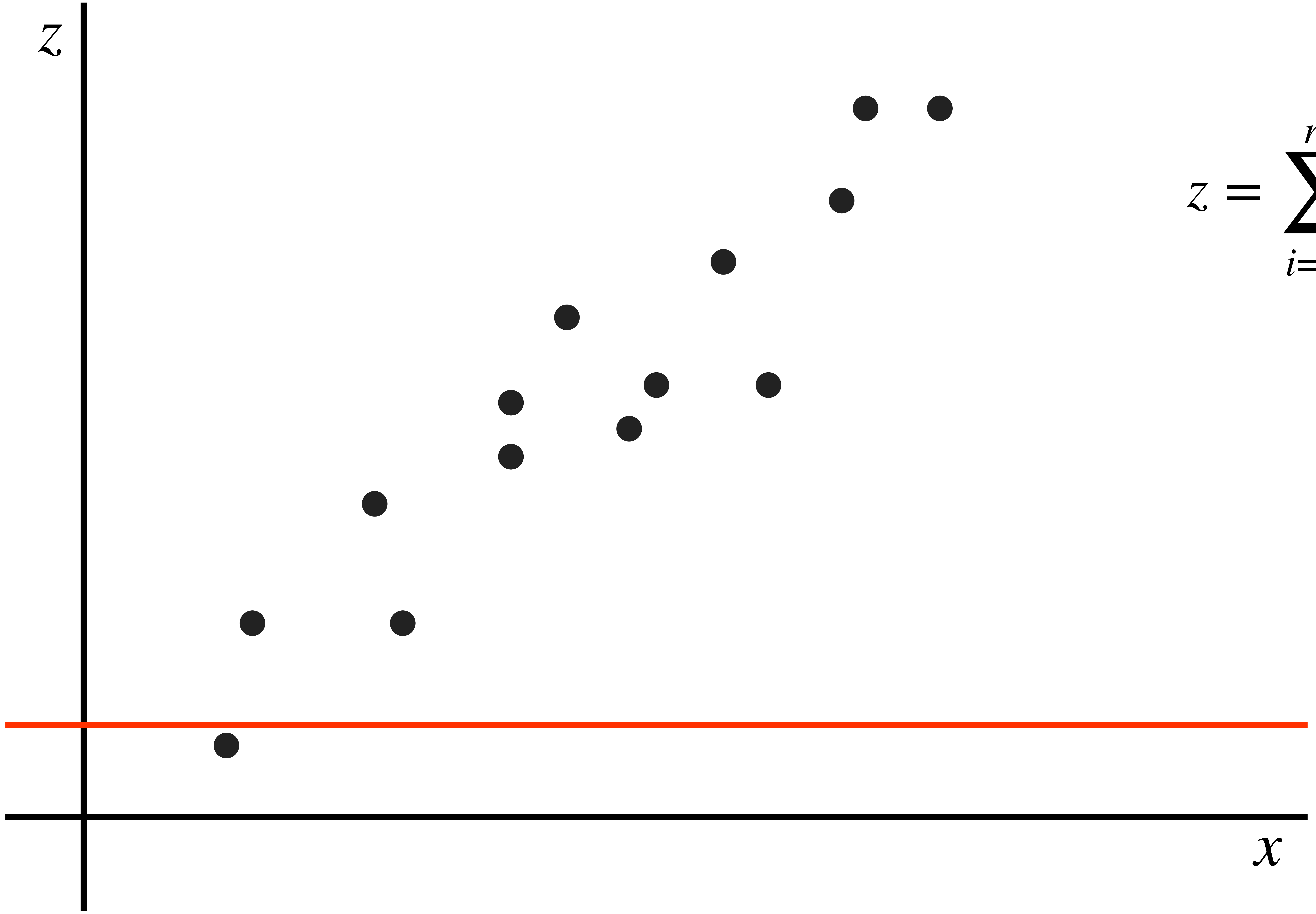
$$z = \sum_{i=1}^{n} w_i x_i + b$$


x_1
x_2
x_n
w_1
w_2
w_n
b
$f_c(z; Z)$
Back Propagation.
Optimize the parameters

Perceptron

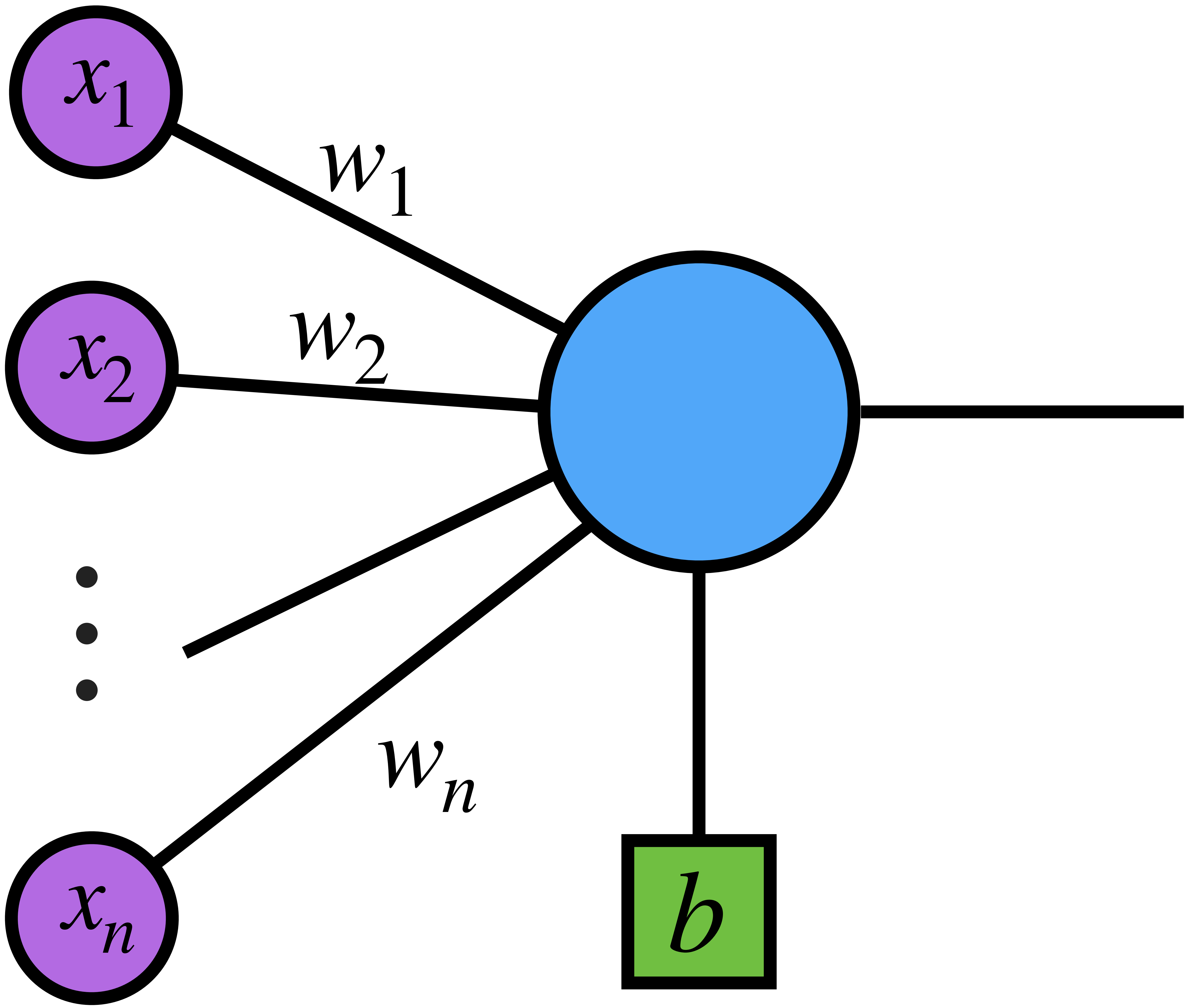


$$z = \sum_{i=1}^{n} w_i x_i + b$$



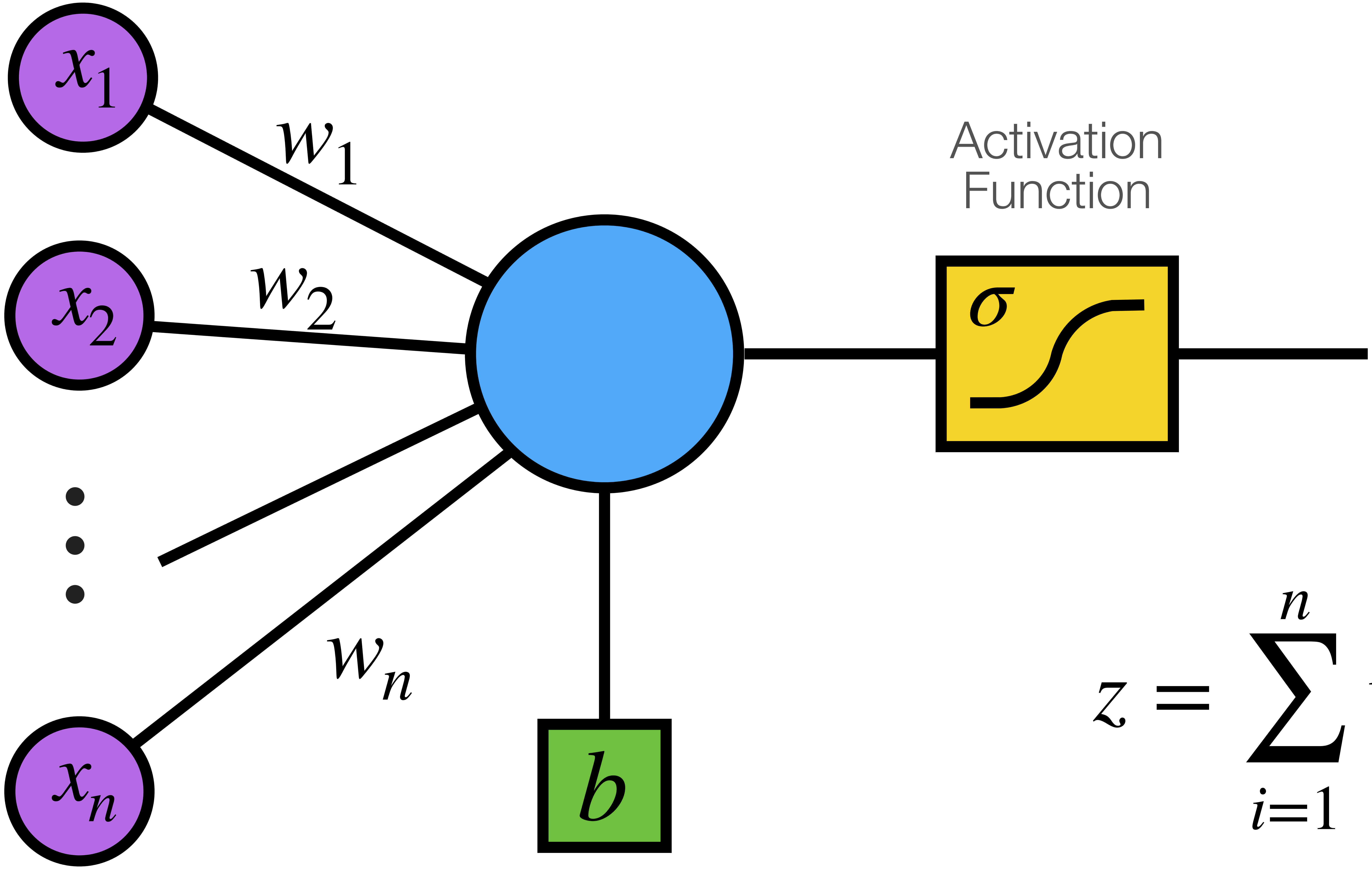
$$z = \sum_{i=1}^{n} w_i x_i + b$$

Perceptron



$$z = \sum_{i=1}^{n} w_i x_i + b$$

Perceptron

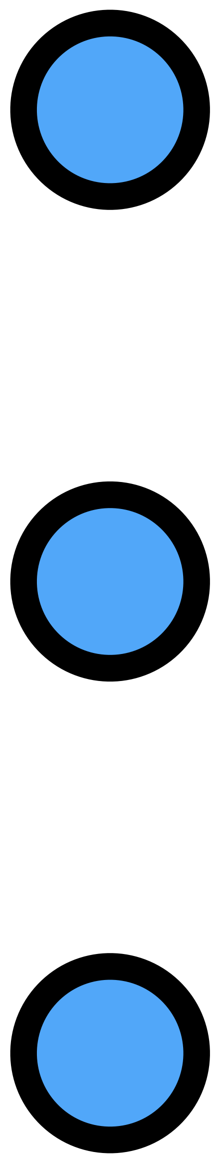


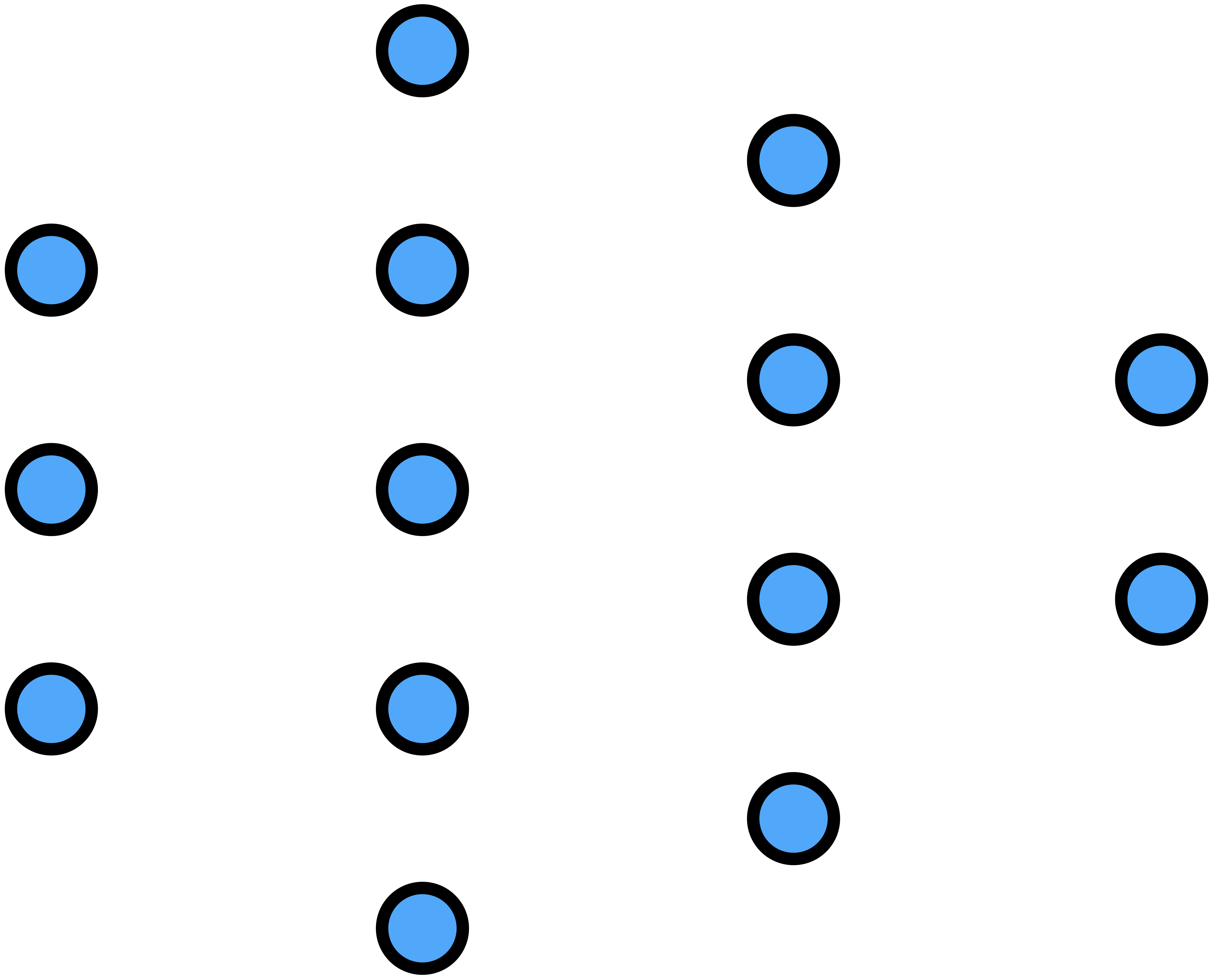
$$y = \sigma(z)$$

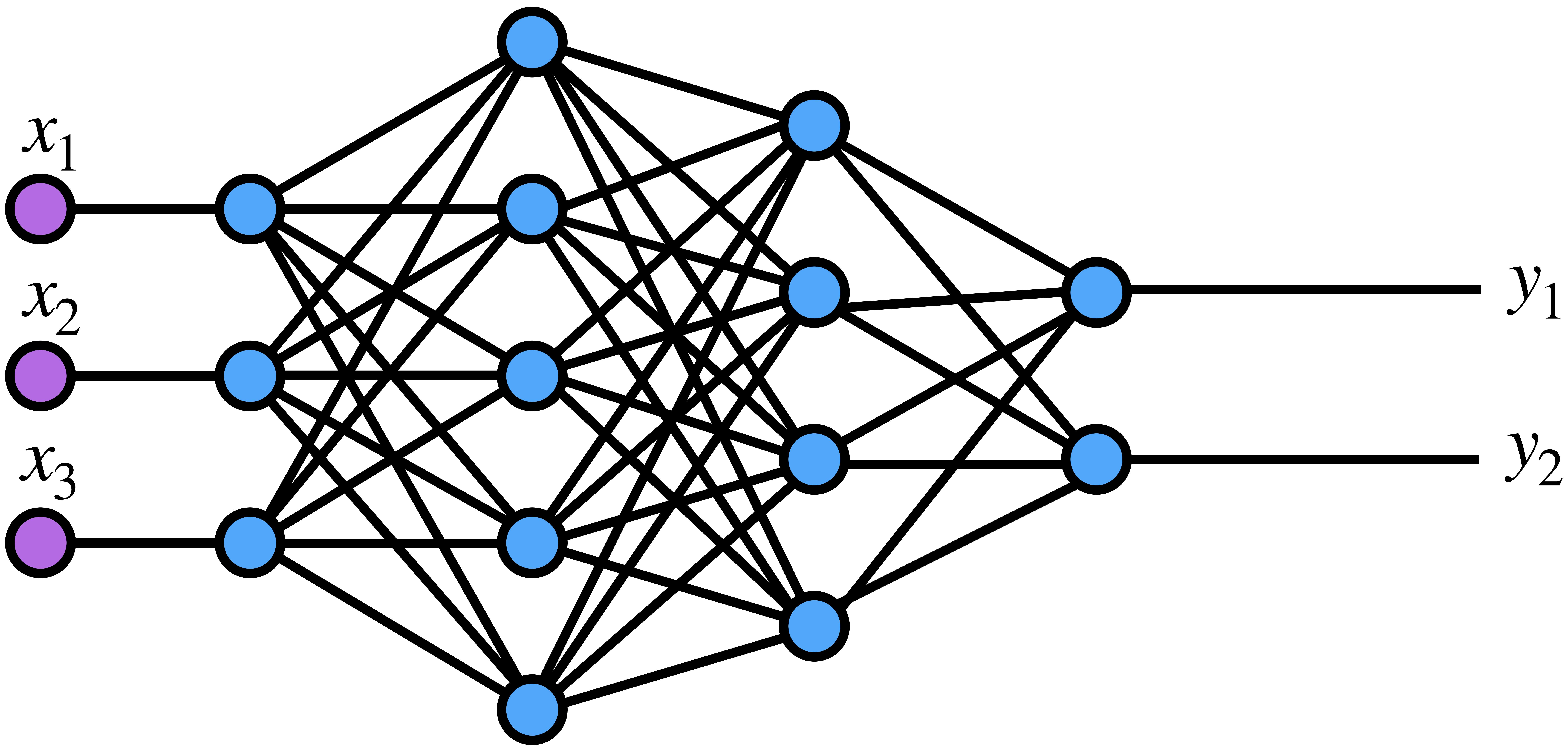
$$z = \sum_{i=1}^{n} w_i x_i + b$$

Neural Networks

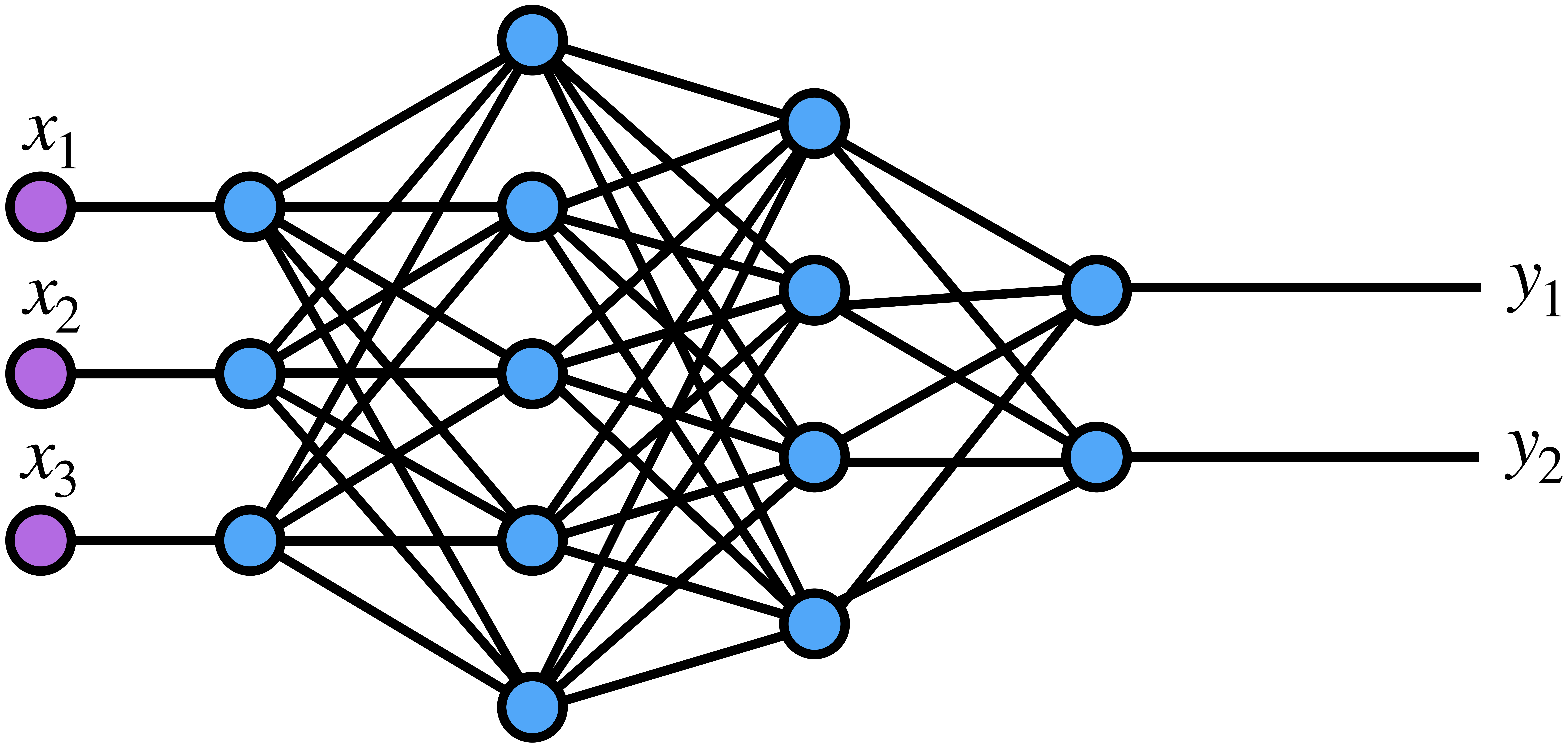
Layers






x_1
x_2
x_3
y_1
y_2

Neural Network



Loss Function

Loss Function

The **loss (or cost) function** measures the difference between predictions of the model and known targets. It is the function that is minimized using the optimizer.

Loss Function

Regression losses

- MeanSquaredError class
- MeanAbsoluteError class
- MeanAbsolutePercentageError class
- MeanSquaredLogarithmicError class
- CosineSimilarity class
- mean_squared_error function
- mean_absolute_error function
- mean_absolute_percentage_error function
- mean_squared_logarithmic_error function
- cosine_similarity function
- Huber class
- huber function
- LogCosh class
- log_cosh function

Probabilistic losses

- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- Poisson class
- binary_crossentropy function
- categorical_crossentropy function
- sparse_categorical_crossentropy function
- poisson function
- KLDivergence class
- kl_divergence function

Optimizer

Optimizer

The **optimizer** defines how to adjust the parameters of the neural network.

The simplest method is based on the **gradient descent,**

$$w_{t+1} = w_t - \alpha \frac{\partial f_C}{\partial w}$$

α : learning rate

Optimizer

The **Momentum** method incorporates the average of past gradients to accelerate the convergence**,**

$$w_{t+1} = w_t - \alpha m_t$$

$$m_t = (1 - \beta)\frac{\partial f_C}{\partial w} + \beta m_{t-1}$$

m_t : aggregate of gradients at time t

β : average parameter ($\sim 0.9)$

Optimizer

The **R**oot **M**ean **S**quare **P**ropagation **(RMPS)** method takes an exponential moving average of the gradients,

$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t + \epsilon)^{1/2}} \frac{\partial f_C}{\partial w}$$

$v_t = \beta v_{t-1} + (1 - \beta)\left(\frac{\partial f_C}{\partial w}\right)^2$: weighted sum of past gradients

ϵ : small positive parameter ($\sim 10^{-8}$)

Optimizers

- SGD (Stochastic Gradient Descent)
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam
- Ftrl

ADAM

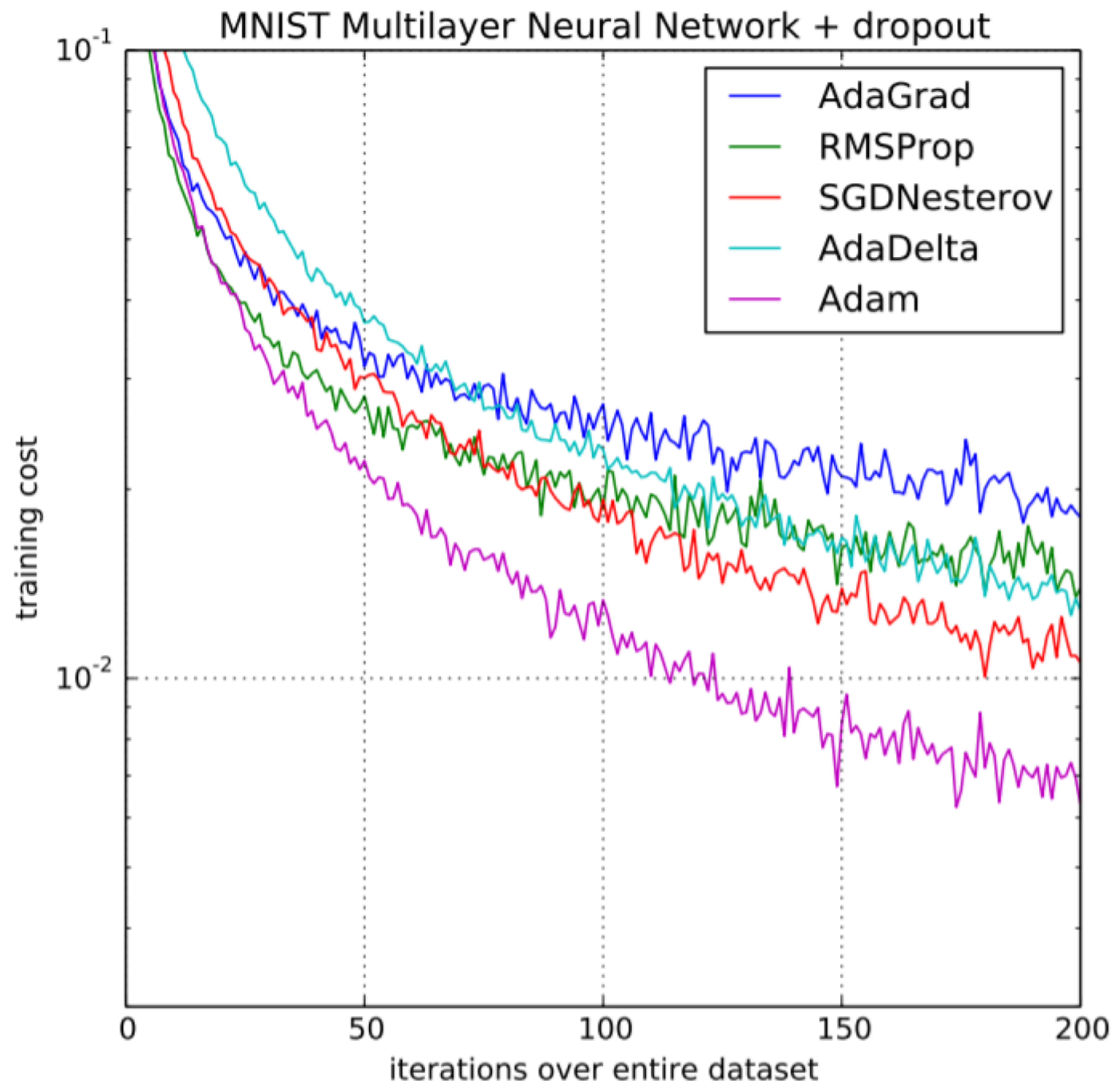
ADAptive **M**oment Estimation [see arXiV:1412.6980]

Is one of the simpler optimizers. Uses a combination of **gradient descent** and **momentum methods**.

According to its documentation, this optimizer is efficient, requires little memory and it is useful for problems with a large number of data or parameters.

Only one argument: learning_rate (the default is 0.001). Measures how much will adjust the parameters in each epoch.

ADAM



MNIST Multilayer Neural Network + dropout
10^{-1}
10^{-2}
training cost
0
50
100
150
200
iterations over entire dataset
AdaGrad
RMSProp
SGDNesterov
AdaDelta
Adam

@astronomiaOAN

AstronomiaOAN

@astronomiaOAN