Department of Mathematics
and Computer Science

Advanced Machine Learning

Neural Networks and Gradient-Based Learning

Lukas Galke

Spring 2026

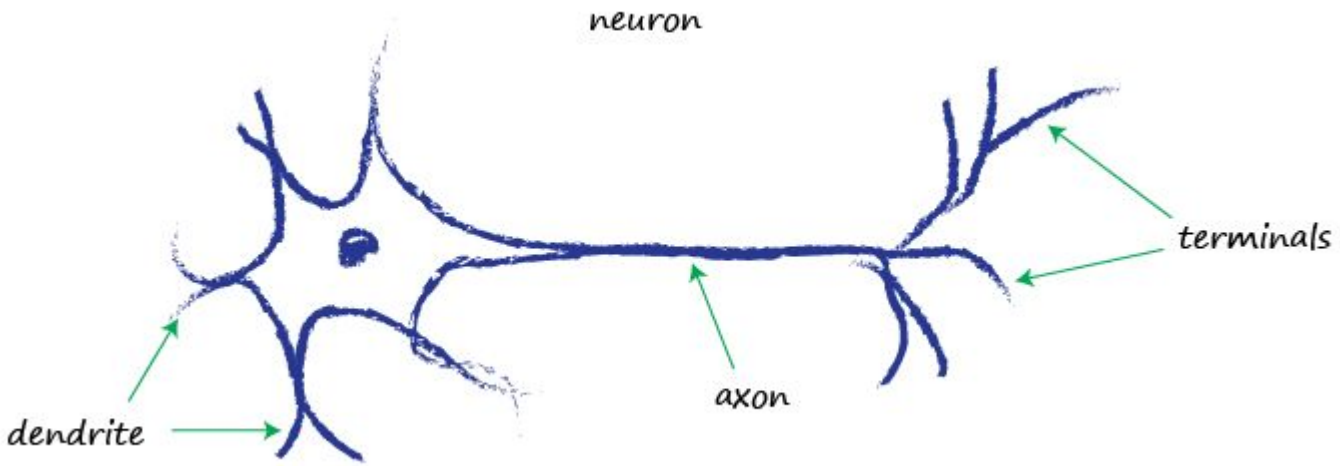
SDU

Slides based on Richard Röttger's material

Keywords for this session

- Neural Networks
- Gradient-based Learning

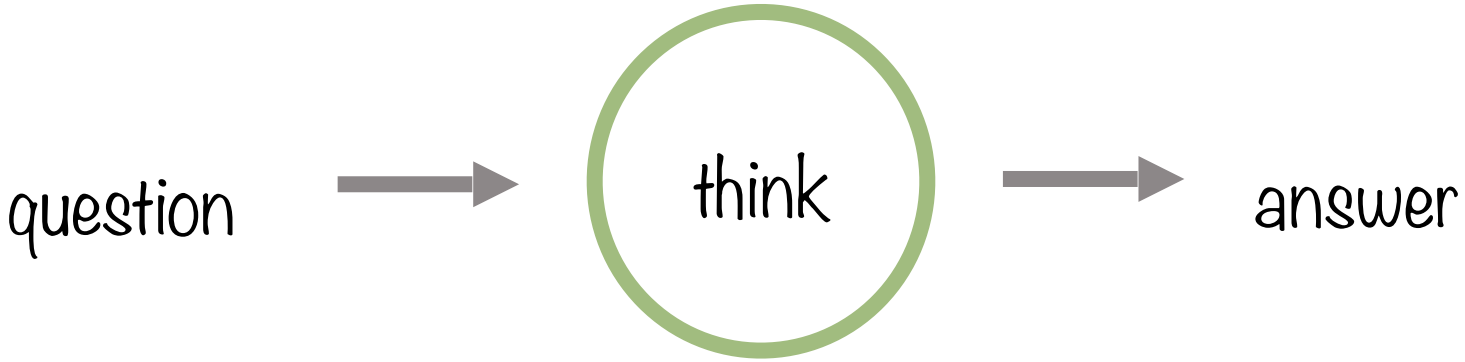
The Neuron … in Nature



That is nice … but

➢We will use Neurons as basic simple "prediction machines"

➢What does that mean?

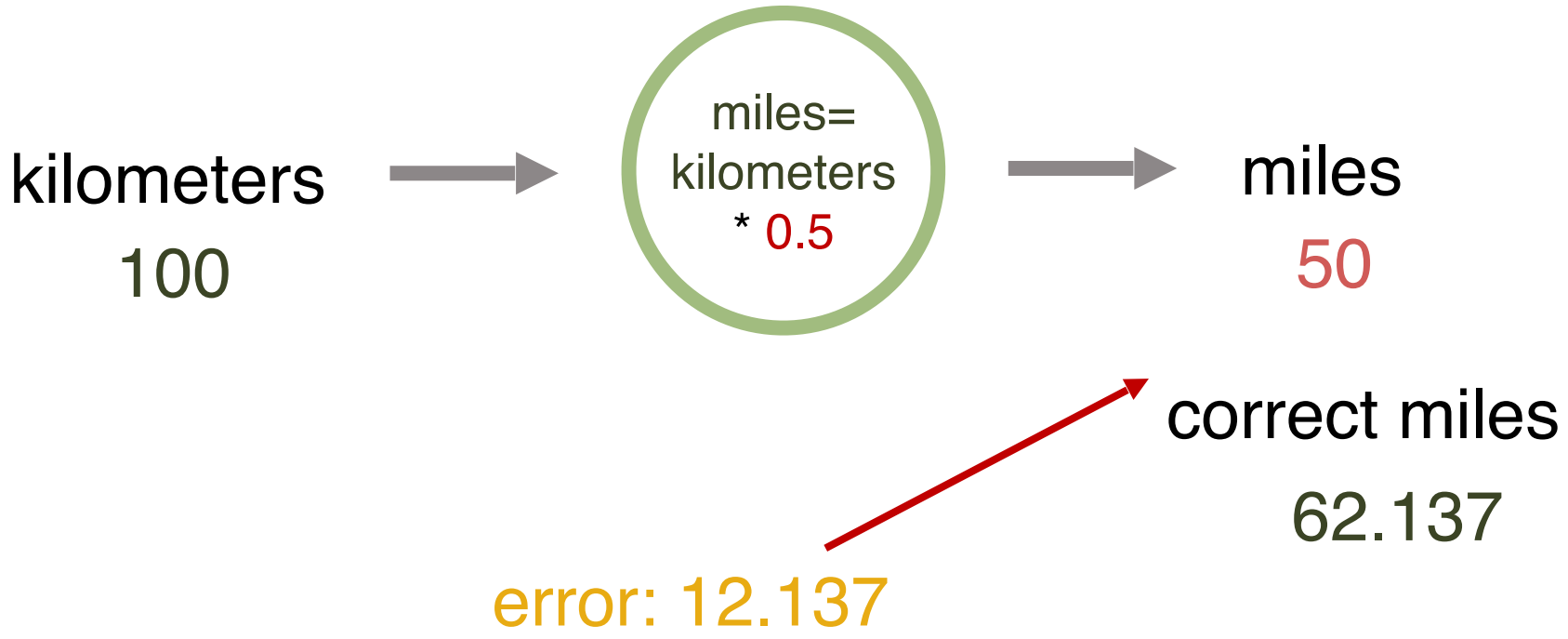


Example

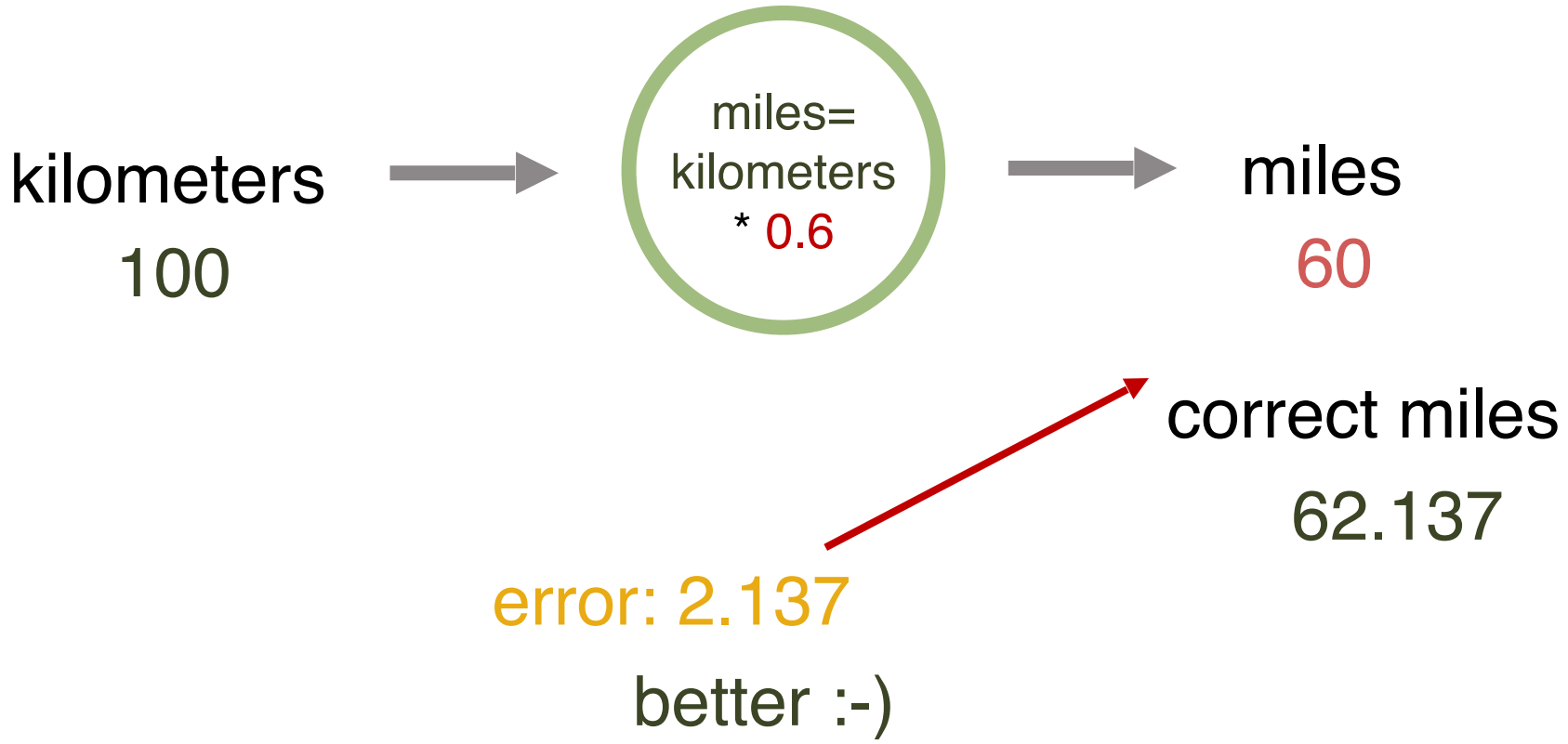
Example


kilometers
100
miles=
kilometers
* 0.5
miles
50
random starting point,
only model Parameter

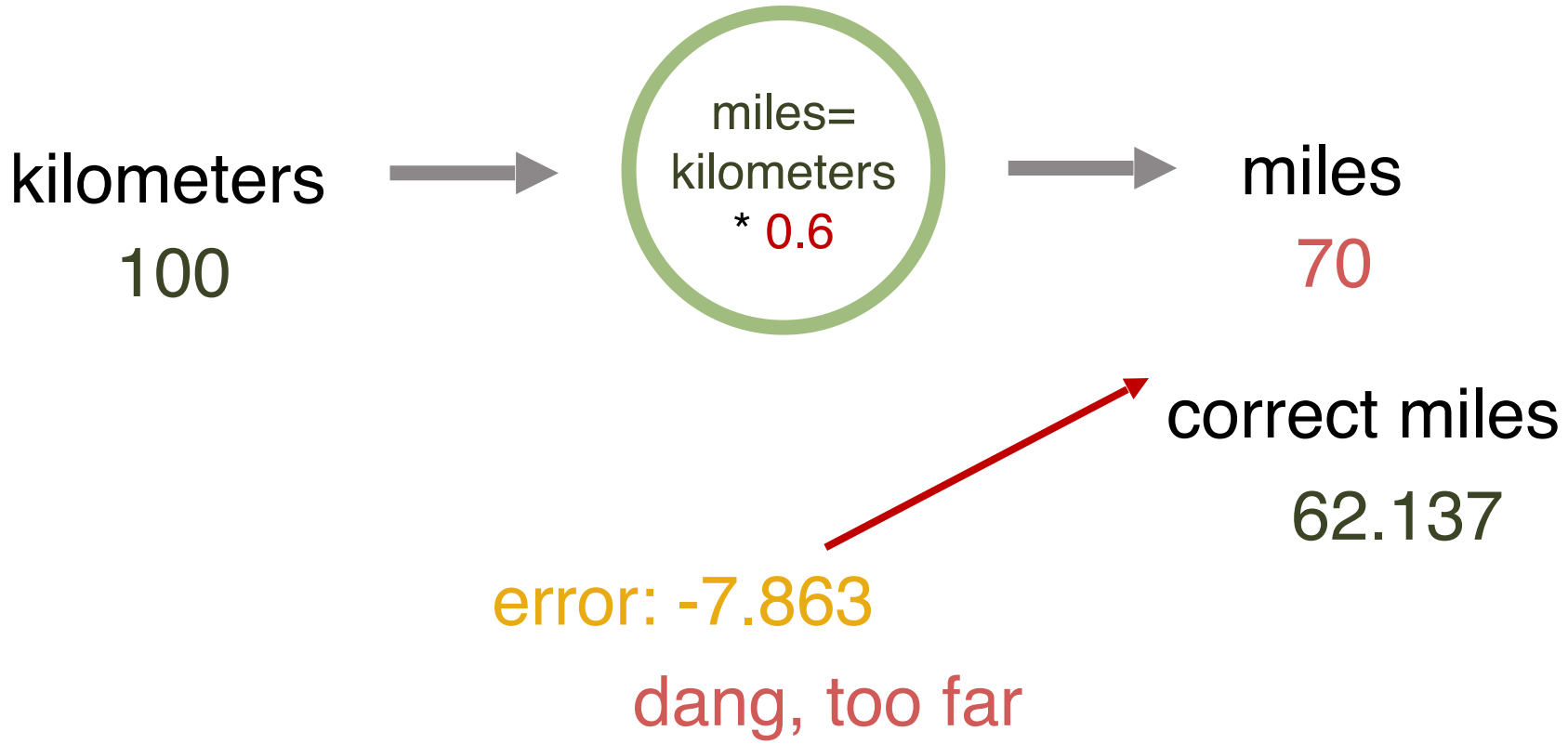
Example


kilometers
100
miles=
kilometers
* 0.5
miles
50
correct miles
62.137
error: 12.137

Example: Learning



Example: Learning



Now with more inputs

- This was extremely simplistic

- Now let's look at a more complicated thing:
  - You are thinking whether you should attend a sports event

- You base the decision on the following factors:
  1. Temperature?
  2. Ticket Price?
  3. Travel time?

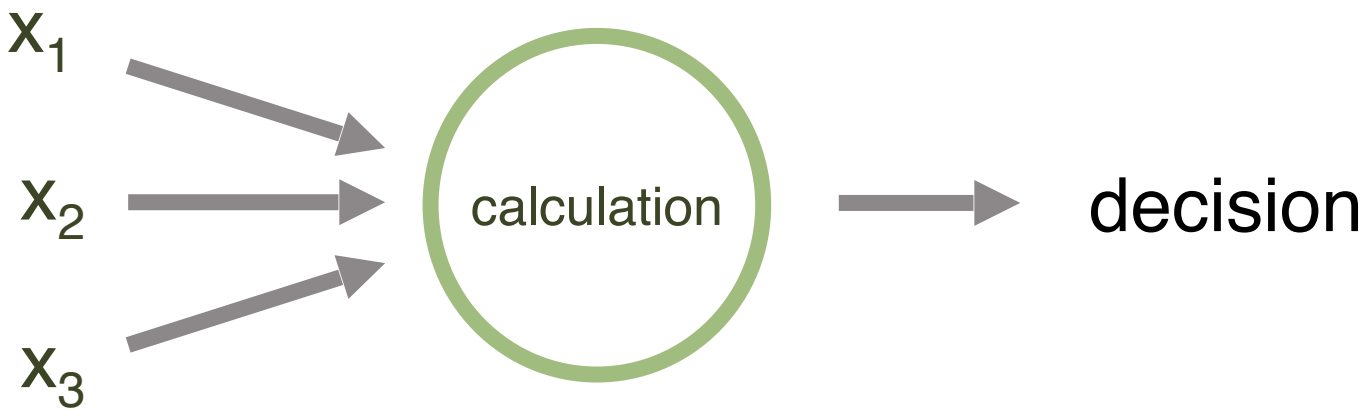
Now with more inputs

- This was extremely simplistic

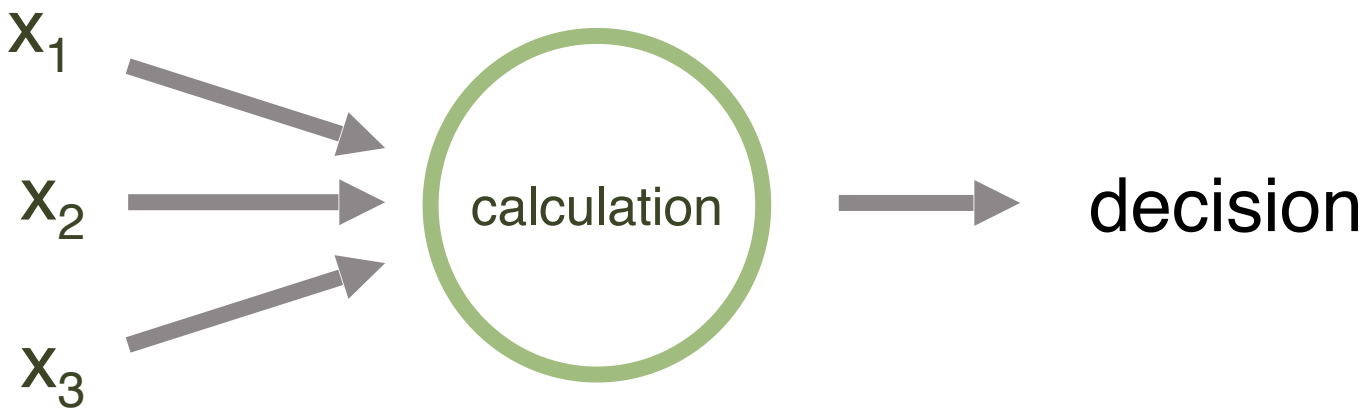
- Now let's look at a more complicated thing:
  - You are thinking whether you should attend a sports event

- You base the decision on the following factors:
  1. Temperature? x_1
  2. Ticket Price? x_2
  3. Travel time? x_3

Now with more inputs



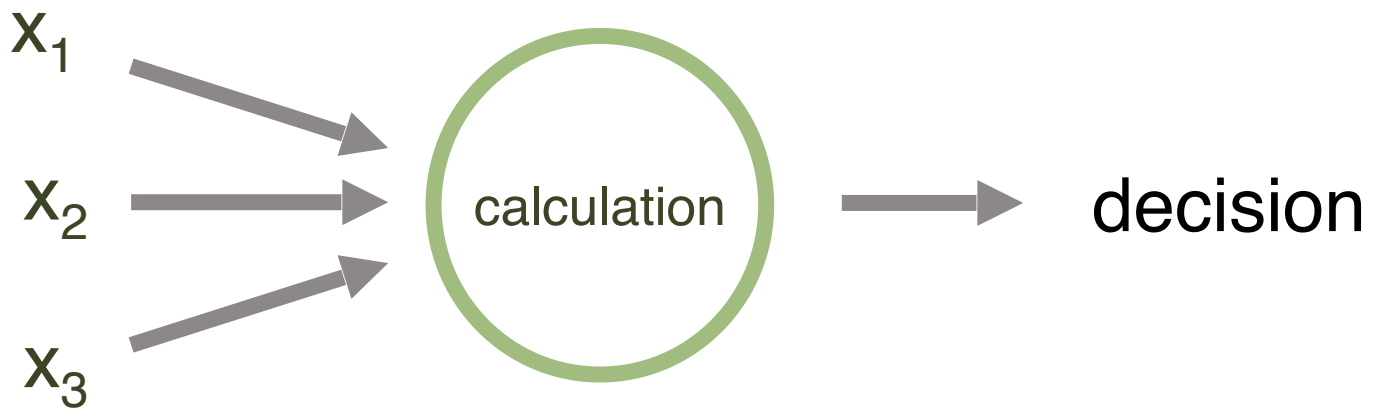
Now with more inputs



➢How to make the decision?

$$x_1 + x_2 + x_3 > \text{threshold?}$$

Now with more inputs

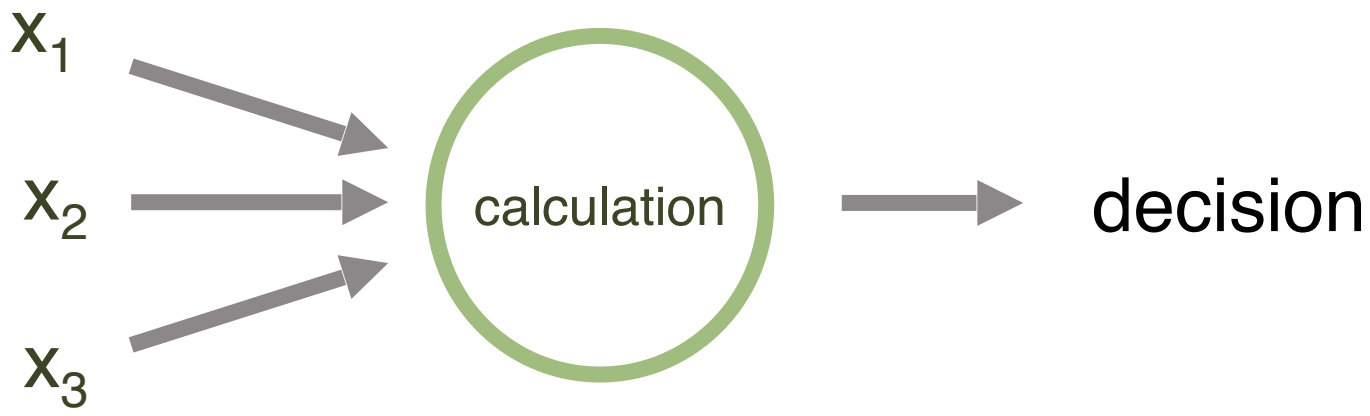


➢How to make the decision?

$$x_1 + x_2 + x_3 > \text{threshold?}$$

➢That would mean, the higher the price and the farther away, the more likely you are to go …

Now with more inputs



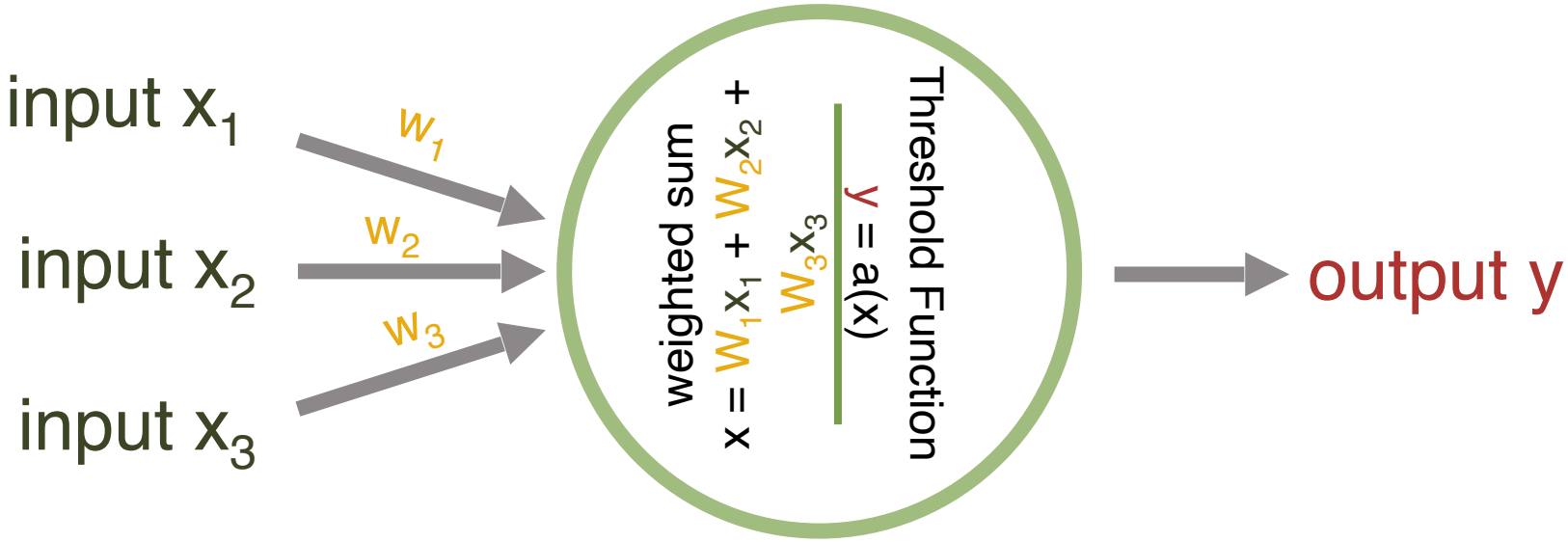
➢How to make the decision?

➢Therefore, introduce weights:

$$W_1x_1 + W_2x_2 + W_3x_3 > \text{threshold?}$$

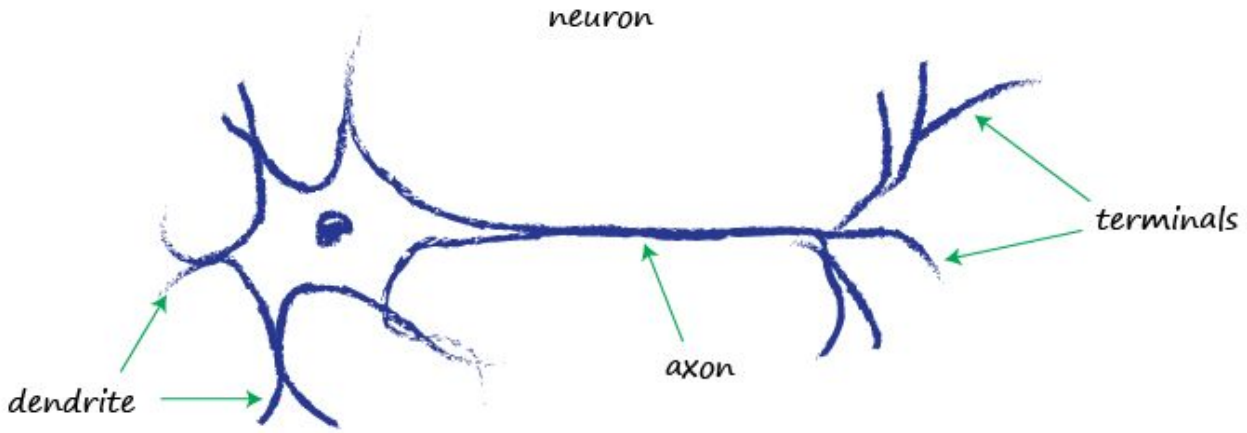
➢Now, you can set the weights and threshold according to your preference, e.g., 1, -4, -1

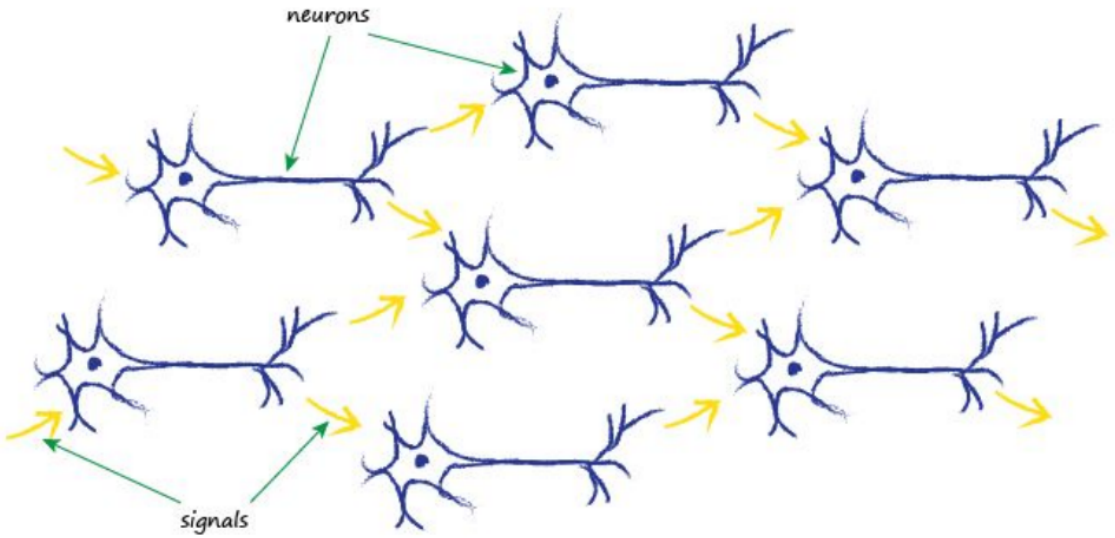
This is in fact how neurons work



- An artificial neuron (aka perceptron) consists of
  - A number of weighted inputs
  - An activation function
  - The generated output

From Neuron to "Brain"

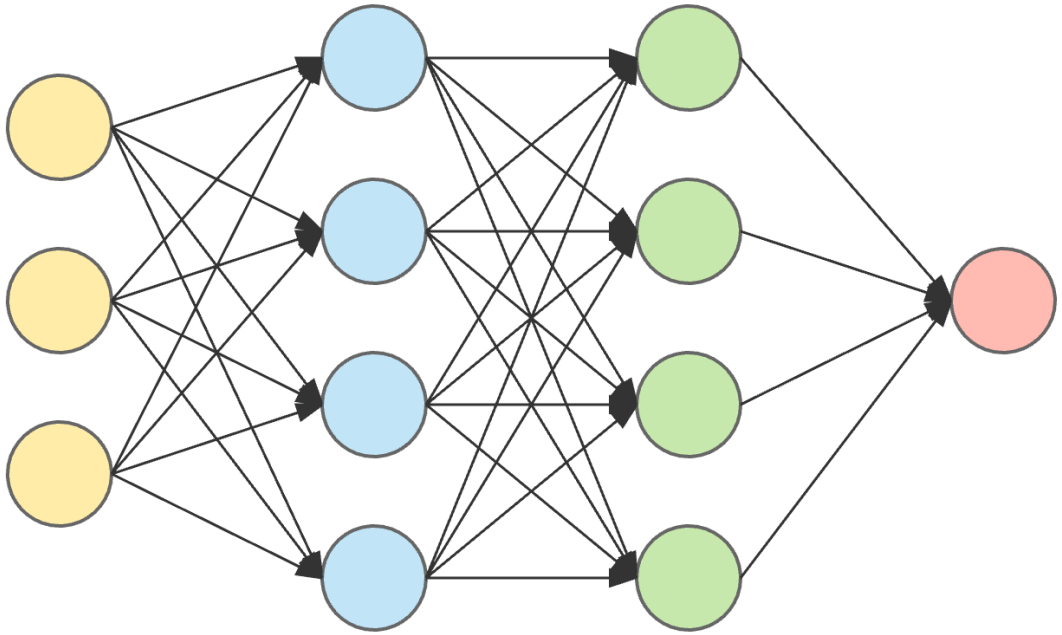




- We now can connect neurons
- The output of one neuron becomes the input of other neurons

In a more structured way

- We normally have more than one node
- Multiple Nodes are arranged in layers
- Each layer receives the generated output from the previous layer



How to build these networks?

- **Define the Architecture**
  - How many inputs?
  - How many hidden layers?
    - How many neurons per layer?
    - What activation function to use?
  - What is the desired output? Define output neurons
    - One-out-of-N classification: Softmax
    - Multiple-out-of-N classification: Sigmoid
    - Regression: linear output
- **Define the loss function**
  - Dependent on the output and task
- **Train the network**

How is a Network Trained?

- **General Procedure:**
  - We present the network with an example where we know the answer
  - We observe the answer of the network and adjust the weights accordingly

- **Ideal world:**
  - We simply look at the dataset and could exactly calculate the weights

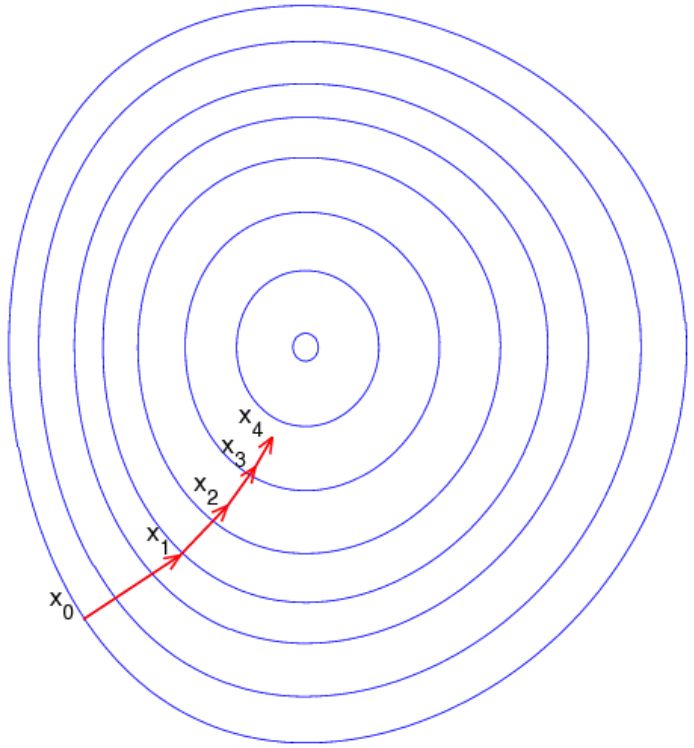
- **Reality:**
  - We define a **cost-function**, the so-called **loss function**
  - We iteratively approximate the best setting by trying to minimize the cost functions
  - We do this by a process called **gradient descent**

Keywords for this session

- Neural Networks
- **Gradient-based Learning**

The Central Idea

➢ Update the model parameters following the steepest slope of the loss function



More Mathematically

- Suppose function $y = f(x)$
- Derivative of function denoted: $f'(x)$ or as dy/dx
  - Derivative $f'(x)$ gives the slope of $f(x)$ at point x
  - It specifies how to scale a small change in input to obtain a corresponding change in the output:

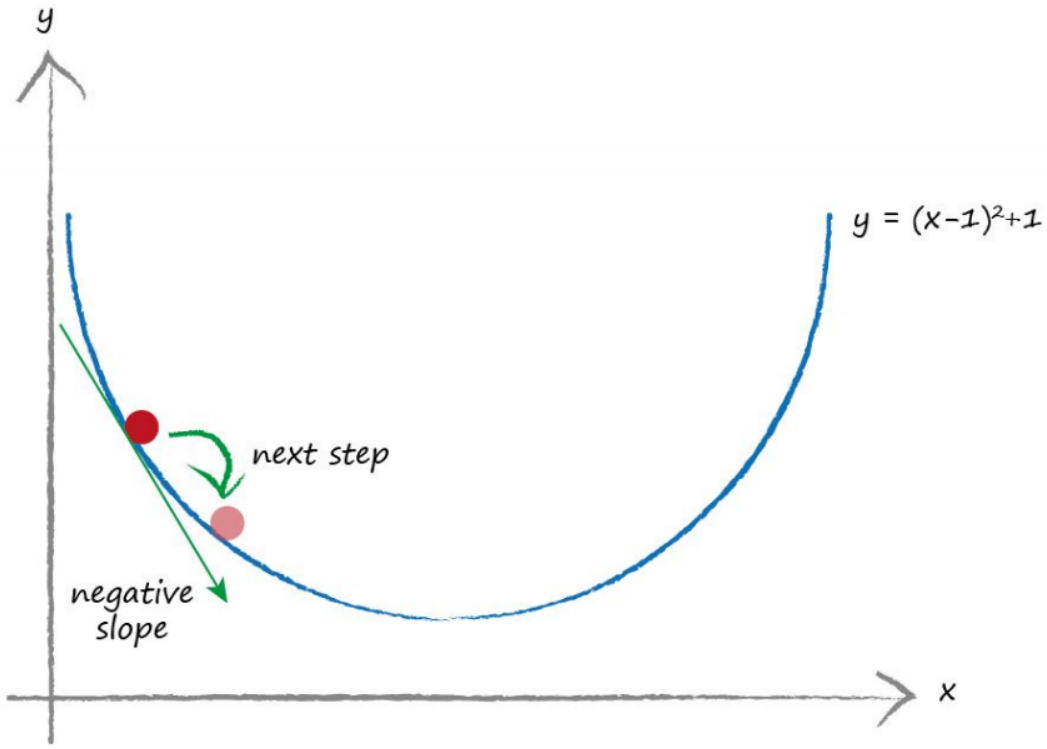
$$f(x + \varepsilon) \approx f(x) + \varepsilon f'(x)$$

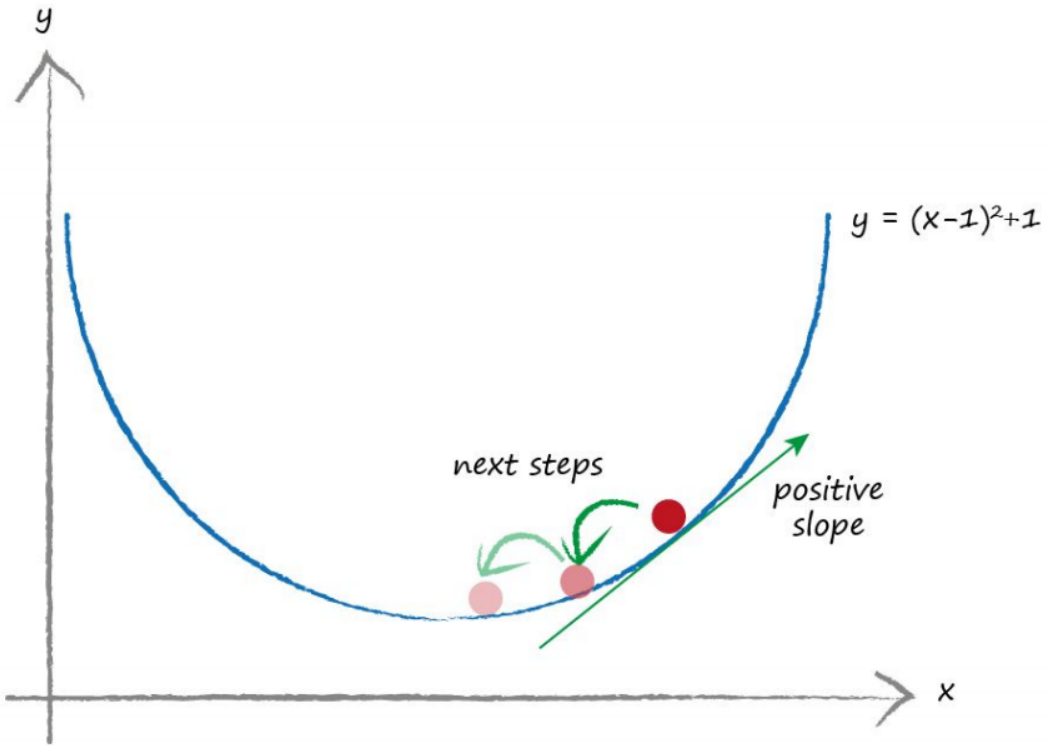
- We know that

$$f(x - \varepsilon \text{sign}(f'(x)))$$

  is less than $f(x)$ for small ε.
  - Thus we can reduce $f(x)$ by moving x in small steps with opposite sign of derivative
  - This technique is called gradient descent (Cauchy 1847)

Gradient Descent





The General Case

- Let us assume we a data dataset $X = \{x_1, \ldots, x_N\}$
- We have defined a neural network with the parameters θ
- We further have defined a cost function J:

$$J(\theta, \boldsymbol{x}, \boldsymbol{y})$$

  - Which give us per sample $\boldsymbol{x}$ and the true label $\boldsymbol{y}$, and the current parameters of the model a certain cost
    - If we, e.g., misclassify x we will get high costs, if we are correct, very low costs

The General Case

➢ For each object $\boldsymbol{x}_1$ we can now calculate the gradient:

$$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} \boldsymbol{J}\left(\boldsymbol{\theta}, \boldsymbol{x}_1, \boldsymbol{y}_1\right)$$

➢ This gradient tells us how to modify the weights in order to achieve a lesser weight, but is noisy and overly depended on $\boldsymbol{x}_1$

➢ Since we want to improve on the entire dataset, we calculate the overall gradient:

$$\boldsymbol{g} = \frac{1}{N} \sum_{X} \nabla_{\boldsymbol{\theta}} \boldsymbol{J}\left(\boldsymbol{\theta}, \boldsymbol{x}_i, \boldsymbol{y}_i\right) = \nabla_{\boldsymbol{\theta}} \boldsymbol{J}\left(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y}\right)$$

A bit more formally

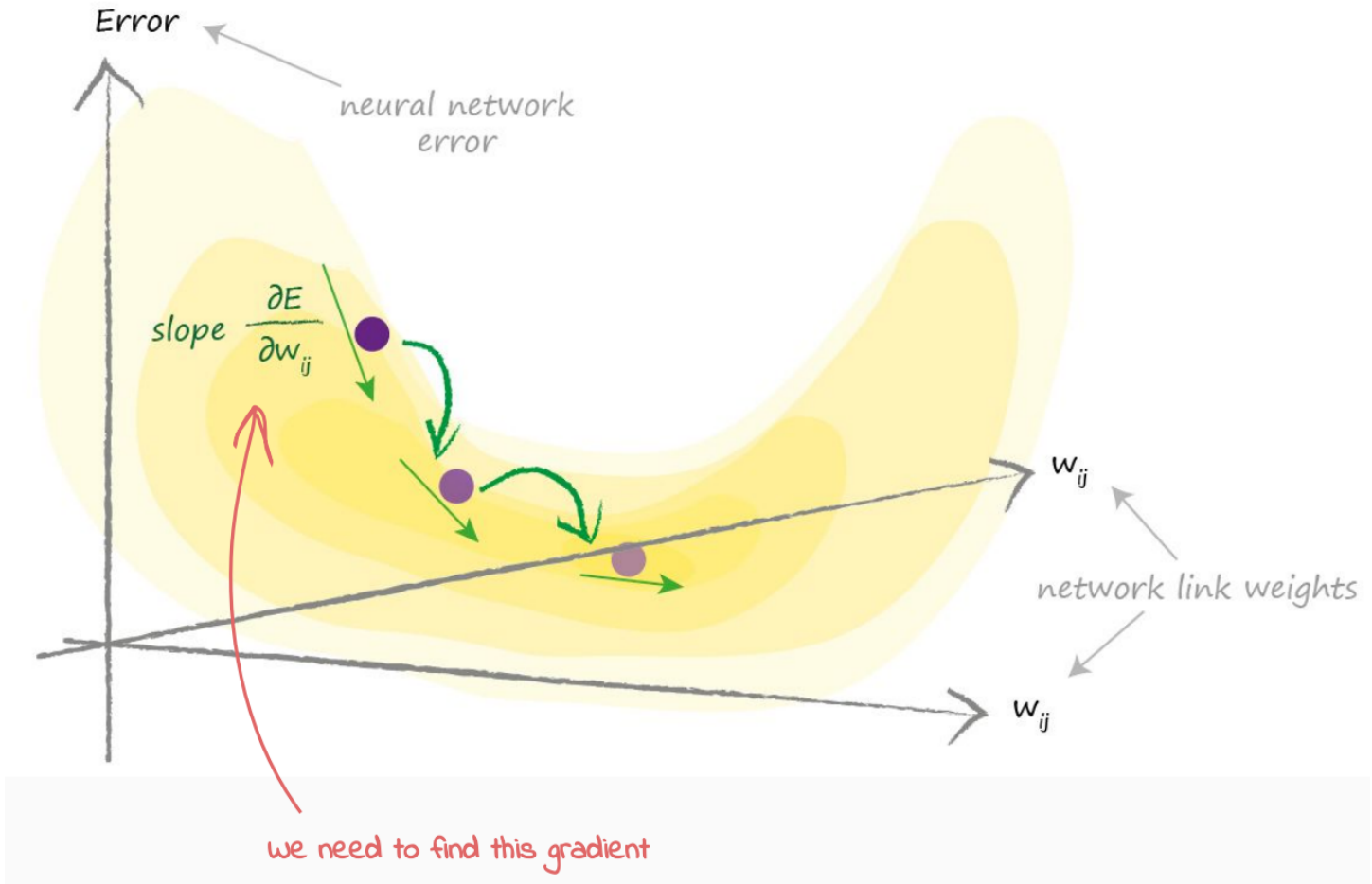
➢We now modify the weights according to the gradient descent method:

$$\boldsymbol{g} = \frac{1}{N} \sum_{X} \nabla_{\theta} \boldsymbol{J}\left(\boldsymbol{\theta}, \boldsymbol{x}_i, \boldsymbol{y}_i\right) = \nabla_{\theta} \boldsymbol{J}\left(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y}\right)$$

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{old} - \epsilon \boldsymbol{g}$$

- ➢ϵ is called the learning rate
- ➢Note, most of the time, we do not compute the gradient for all available data, but for randomly selected small portions of the dataset.
- ➢This is called **Stochastic Gradient Descent**

Gradient Descent



The Loss Function

➢We are often using the loss function called cross-entropy

$$H(p,q) = -\mathbb{E}[\log(q)]$$

➢For most of our (discrete) cases

$$H(p,q) = -\sum_{x \in X} p(x)\log\big(q(x)\big)$$

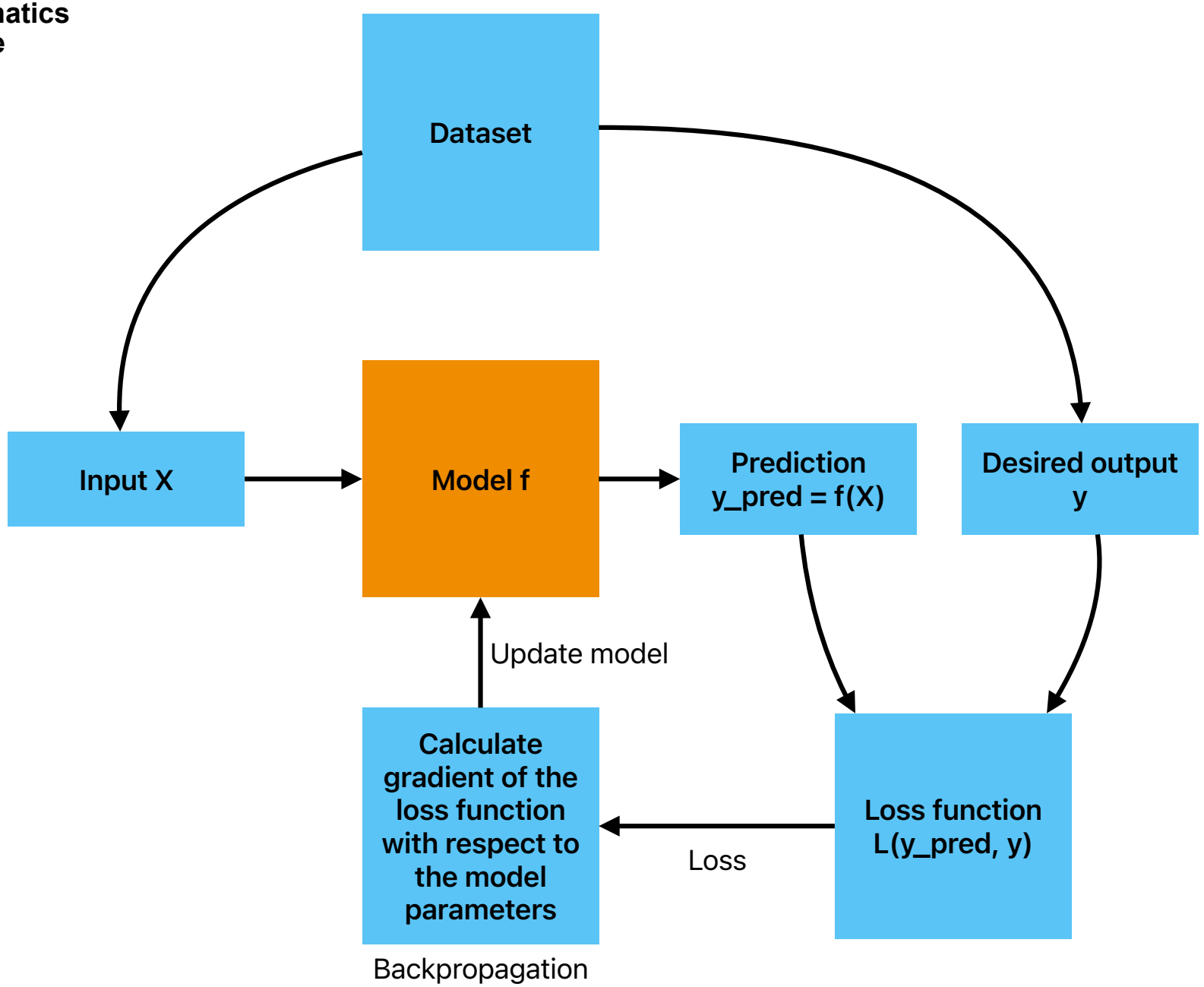
➢**Where $p(x)$ is the real data distribution, while $q(x)$ is our prediction**

➢Interpretation: The average number of bits needed to encode data distributed according to p(x), if you assume it follows q(x).

Keywords for this session

- Neural Networks
- Gradient-based Learning

Outlook


Dataset
Input X
Model f
Prediction
y_pred = f(X)
Desired output
y
Update model
Calculate
gradient of the
loss function
with respect to
the model
parameters
Loss function
L(y_pred, y)
Loss
Backpropagation

How’s my teaching?

https://www.admonymous.co/lukasgalke