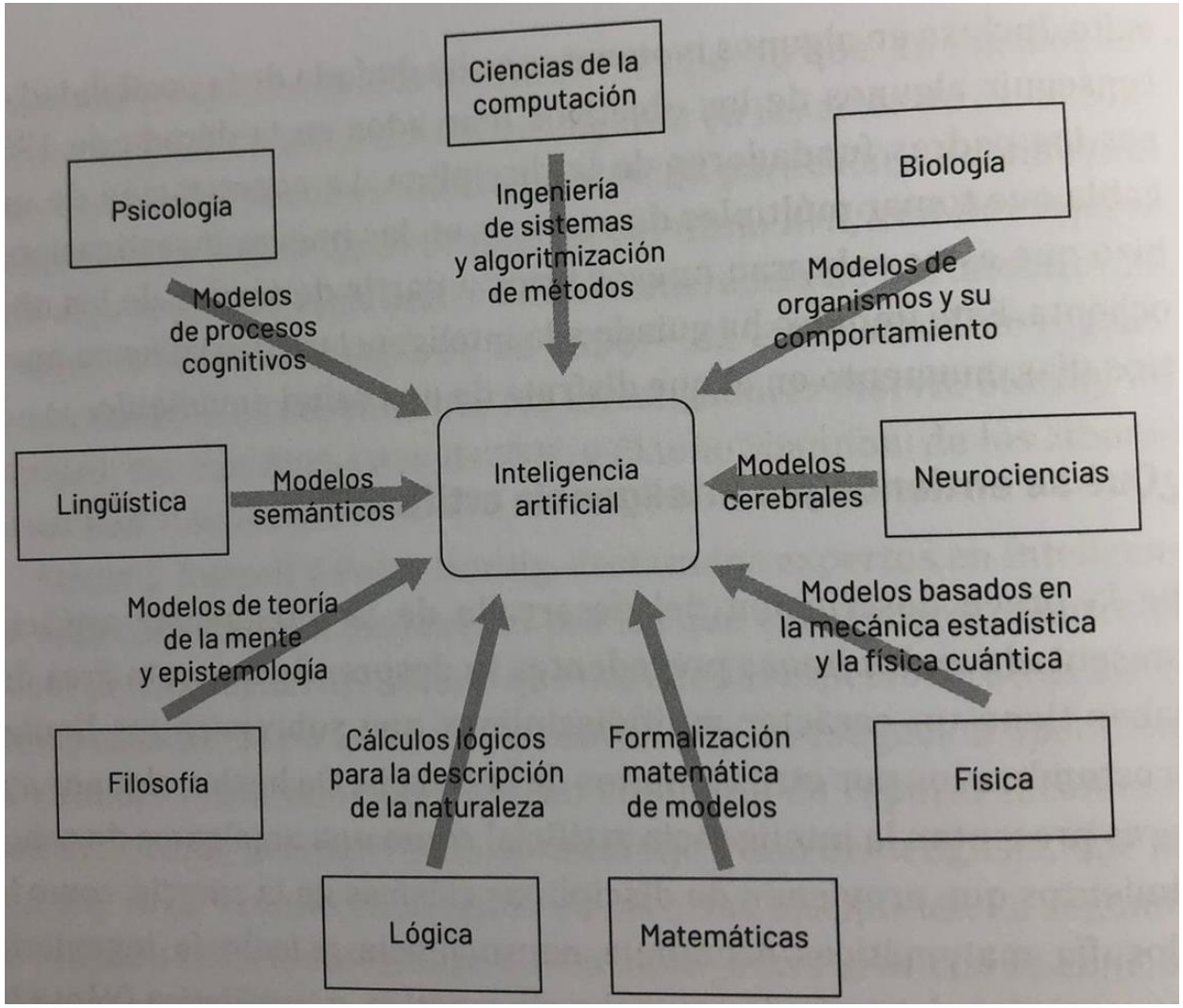
Introducción



Introducción

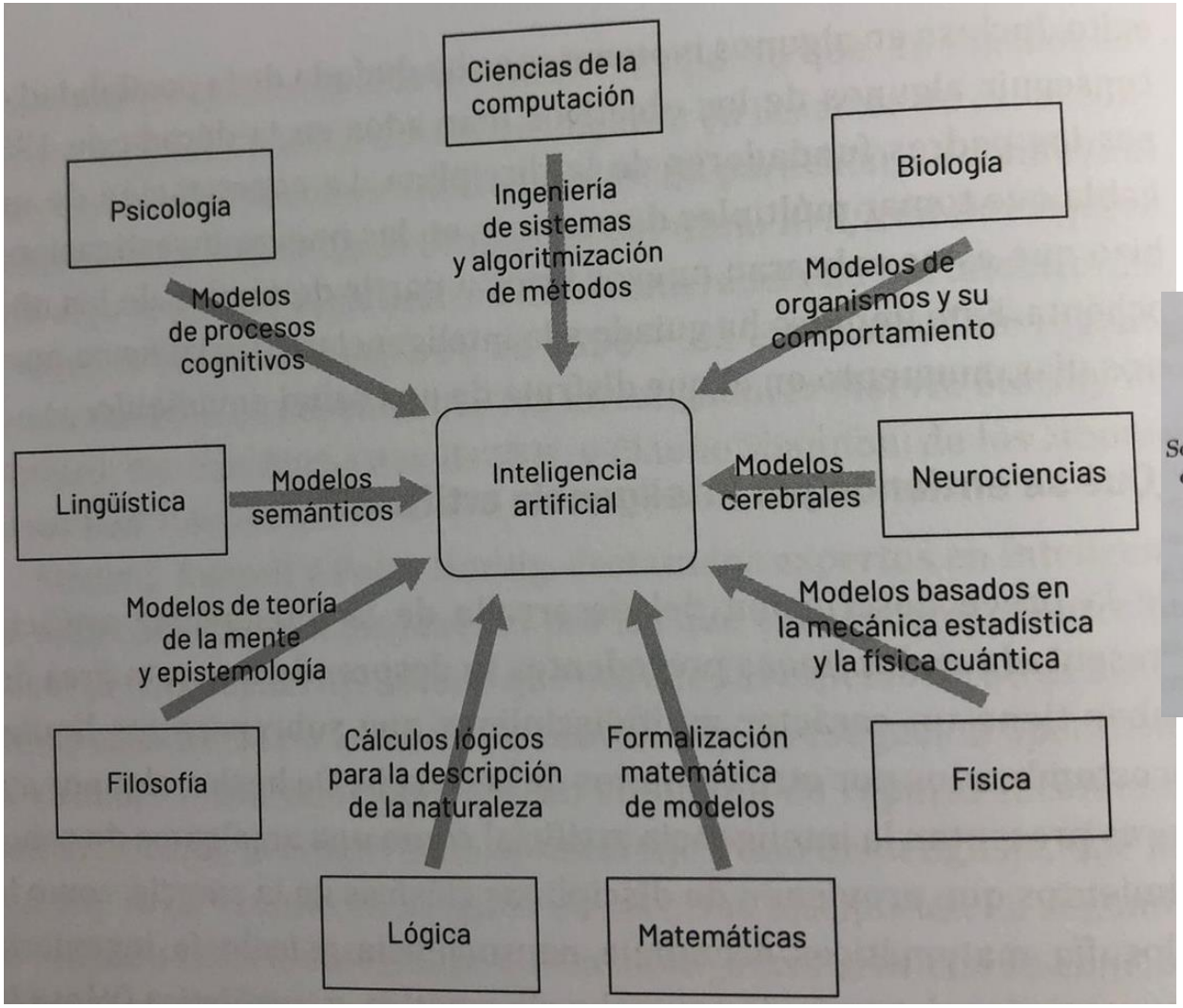
- Un componente importante de la inteligencia es la capacidad de aprender

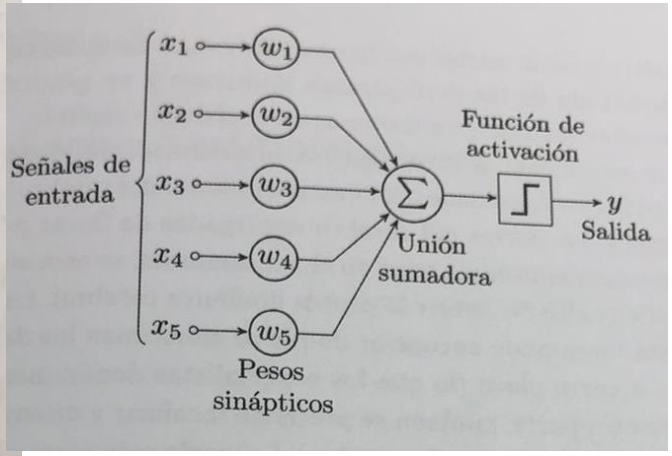
- Se distinguen tres pasos elementales:

  - Recogida de datos (estamos en el siglo de los datos)
  - Transformación - Representación de la información entregada
  - Generalización de dicha información para la toma de decisiones

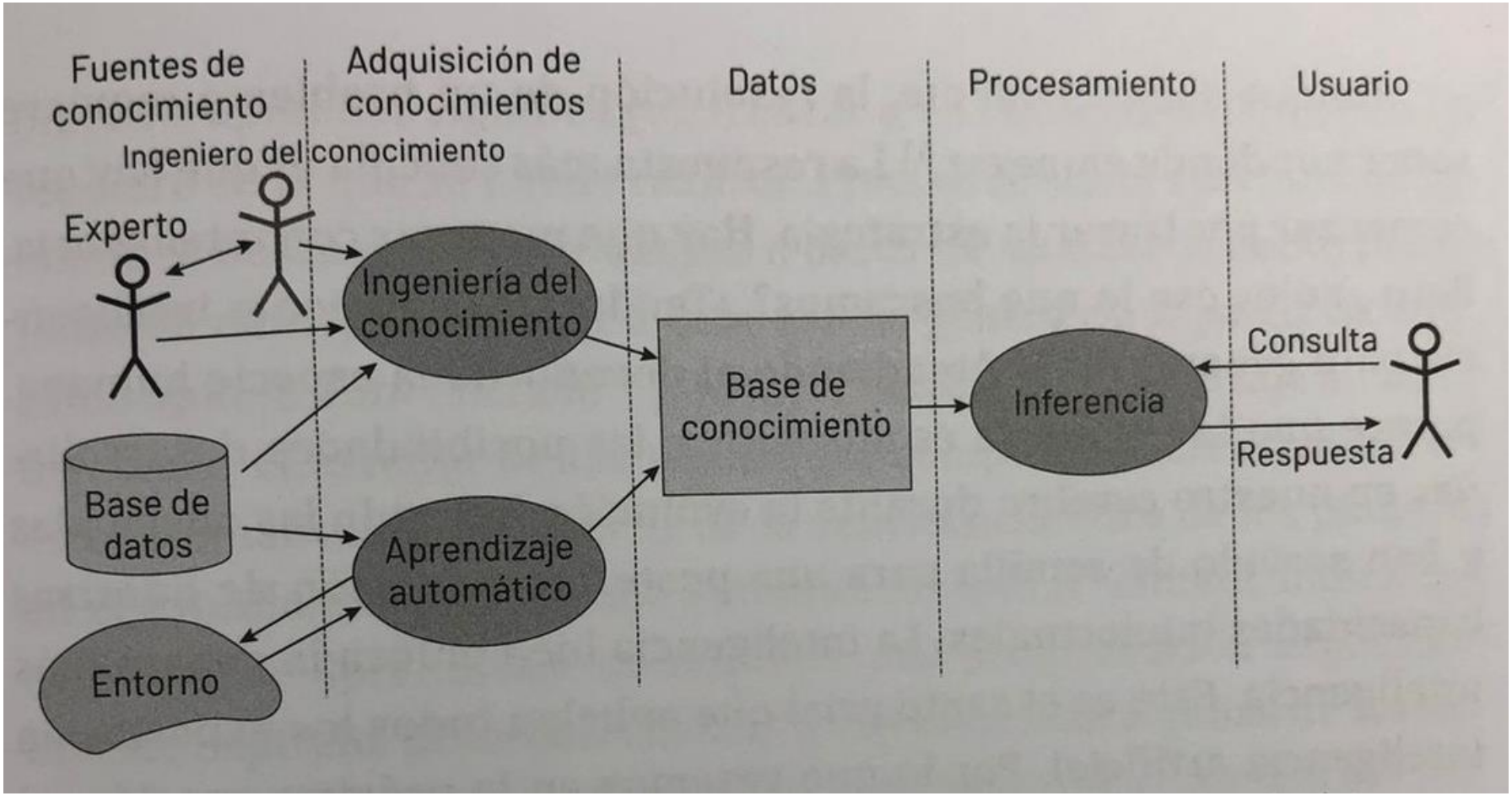
- La llegada de los datos está haciendo el trabajo más empirico (inductivo) que formal (deductivo)

Introducción





Introducción



Introducción

MUNDO NUEVO

Deep Patient, el futuro médico de los pacientes con Inteligencia Artificial

Se prevé que el modelado predictivo con datos de registros médicos electrónicos impulsará la medicina personalizada y mejorará la calidad de la atención médica.

Mar 14, 2021 | Info general |

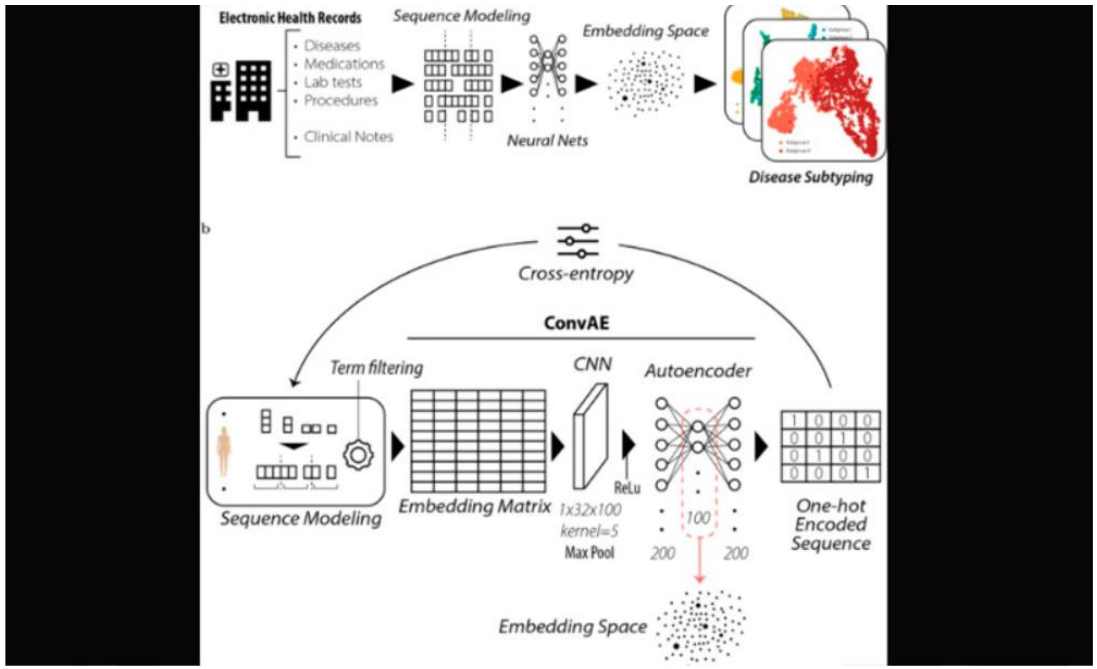


Foto: *un marco que permite el análisis de estratificación de pacientes a partir de representaciones profundas de HCE sin supervisión; Detalles de la arquitectura de aprendizaje de representación de ConvAE.*

https://www.conclusion.com.ar/info-general/deep-patient-el-futuro-medico-de-los-pacientes-con-inteligencia-artificial/03/2021/#:~:text=El%20Sistema%20de%20inteligencia%20artificial,por%20su%20excelencia%20en%20la

Introducción

Bertalan Meskó, MD, PhD • 3er+
Director of The Medical Futurist Institute (Keynote Speaker, Aut...
2 semanas •

+ Seguir •••

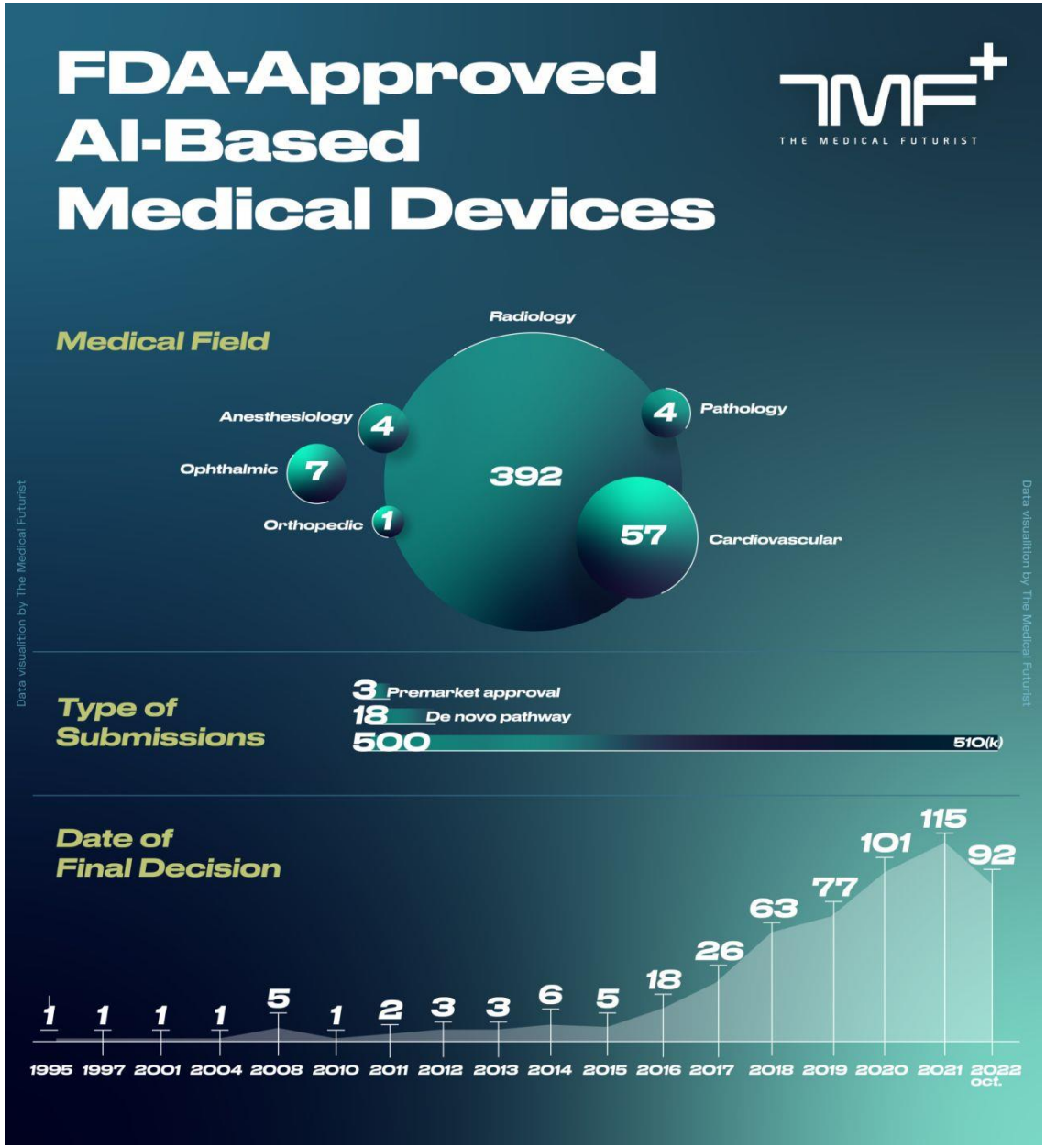
I just analyzed all the FDA-approved, AI-based medical technologies.

This infographic is based on the official list of the FDA and displays the medical fields, types of submissions, and final decision dates all in one place.

One interesting finding is that 510k submissions were by far the most popular. This may be due to their relative ease, but it's also worth noting that the field of radiology is currently experiencing a surge in AI-based technology ideas.

It's fascinating to see how AI is rapidly transforming the medical industry!

You can also check out the full list of FDA-approved AI-based medical devices here:



https://www.linkedin.com/posts/bertalanmesko_i-just-analyzed-all-the-fda-approved-ai-based-activity-7055179254489038848-fA0X/?utm_source=share&utm_medium=member_desktop&utm_source=The+Medical+Futurist+Newsletter&utm_campaign=2e2cce779f-EMAIL_CAMPAIGN_2022_02_01_COPY_01&utm_medium=email&utm_term=0_efd6a3cd08-2e2cce779f-420631673&mc_cid=2e2cce779f&mc_eid=54c7e2f5dd

Introducción

Date of Final Decision	Submission Number	Device	Company	Panel (Lead)	Primary Product Code
01/02/2020	K183089	Air Next	NuvoAir AB	Anesthesiology	BZG
05/05/2017	K163665	MATRx Plus	Zephyr Sleep Technologies	Anesthesiology	MNR
11/05/2015	K150102	SnoreSounds	Appian Medical Inc.	Anesthesiology	MNR
03/11/1997	K955841	Compumedics Sleep Monitoring System	Computmedics Sleep Pty. Ltd.	Anesthesiology	MNR
07/27/2022	K210822	DeepRhythmAI	Medicalgorithmics S.A.	Cardiovascular	DQK
07/19/2022	K213357	Study Watch with Irregular Pulse Monitor (Home), Study Watch with Irregular Pulse Monitor	Verily Life Sciences LLC	Cardiovascular	DXH
07/19/2022	K213409	ZEUS System (Zio Watch)	iRhythm Technologies, Inc.	Cardiovascular	DQK
06/29/2022	K213794	Eko Murmur Analysis Software (EMAS)	Eko Devices, Inc.	Cardiovascular	DQD
06/11/2021	K210484	LINQ II Insertable Cardiac Monitor, Zelda AI ECG Classification System	Medtronic, Inc.	Cardiovascular	MXD
06/03/2022	K213971	Atrial Fibrillation History Feature	Apple Inc.	Cardiovascular	QDB
05/31/2022	K220766	eMurmur Heart AI	CSD Labs GmbH	Cardiovascular	DQD
04/26/2022	K212662	AliveCor QT Service	AliveCor, Inc.	Cardiovascular	DQK
04/01/2022	K213657	DEEPVESSEL FFR	KeyaMed NA Inc.	Cardiovascular	PJA

https://www.fda.gov/medical-devices/software-medical-device-samd/artificial-intelligence-and-machine-learning-aiml-enabled-medical-devices

molecular pharmaceutics

Review

pubs.acs.org/molecularpharmaceutics

Applications of Deep Learning in Biomedicine

Polina Mamoshina,[†] Armando Vieira,[‡] Evgeny Putin,[†] and Alex Zhavoronkov*[,†]

[†]Artificial Intelligence Research, Insilico Medicine, Inc, ETC, Johns Hopkins University, Baltimore, Maryland 21218, United States

[‡]RedZebra Analytics, 1 Quality Court, London, WC2A 1HR, U.K.

HOSTED BY

ELSEVIER

Genomics Proteomics Bioinformatics

www.elsevier.com/locate/gpb
www.sciencedirect.com

REVIEW

Deep Learning and Its Applications in Biomedicine

Chensi Cao [1,#,a], Feng Liu [2,#,b], Hai Tan [3,#,c], Deshou Song [3,d], Wenjie Shu [2,e], Weizhong Li [4,f], Yiming Zhou [1,5,*,g], Xiaochen Bo [2,*,h], Zhi Xie [3,*,i]

Introduction

- The amount of biomedical data in public repositories is rapidly increasing, but the heterogeneous nature of this data renders analysis problems.

- Computational biology methods are essential in various fields of biomedicine from **biomarker development** to **drug discovery**, and machine learning methods are increasingly applied.

- **Deep learning** is a class of **ML techniques** that show particular promise in extracting high level abstractions from **raw data** of very large datasets.

Introduction

- Artificial intelligence was born in the 1950s, when pioneers from of computer science started asking whether computers could be made to "**think**".

- AI is a general field that encompasses machine learning and deep learning, but also includes approaches that don't involve **learning**.

- Early chess programs, for instance, only involved hardcoded rules crafted by programmers.
  - Many experts believed that human-level artificial intelligence could be achieved by having programmers handcraft a sufficiently large set of explicit rules for manipulating knowledge (***symbolic AI***)

Introduction

- Machine learning only started to flourish in the 1990s
  - It has quickly become the most popular and most successful subfield of AI
  - A trend driven by the availability of faster hardware and larger datasets.

- Machine learning is tightly related to mathematical statistics
  - unlike statistics, machine learning tends to deal with large, complex datasets for which classical statistical analysis would be impractical.

Introduction

- A machine-learning system is ***trained*** rather than explicitly programmed:
  - **It's presented with many examples relevant to a task**, and it finds statistical structure in these examples that eventually allows the system to come up with rules for automating the task.

- For instance, if you wished to automate the task of tagging your vacation pictures, you could present a machine-learning system with many examples of pictures already tagged by humans, and the system would learn statistical rules for associating specific pictures to specific tags

Introduction

- Machine learning can take place in one of two forms:
  - conventional, **“shallow”** learning (neural networks with a single hidden layer or support vector machines)
  - **deep** learning (neural networks with many hierarchical layers of nonlinear information processing).

- **shallow learning does not deal well with raw data**, requiring extensive human input to set up and maintain, whereas deep learning can be largely unsupervised once set in motion.

The development of ANNs

- As a basis for deep learning, ANNs were inspired by biological processes in the 1960s, when it was discovered that different visual cortex cells were activated when cats visualized different objects, implying that the information was processed **layer by layer** in the visual system.

- ANNs mimicked the perception of objects by connecting artificial neurons within layers that could extract the features of objects.

- However, ANN research stagnated after the 1960s, due to the limited computational capacity of computers at that time.

The development of ANNs

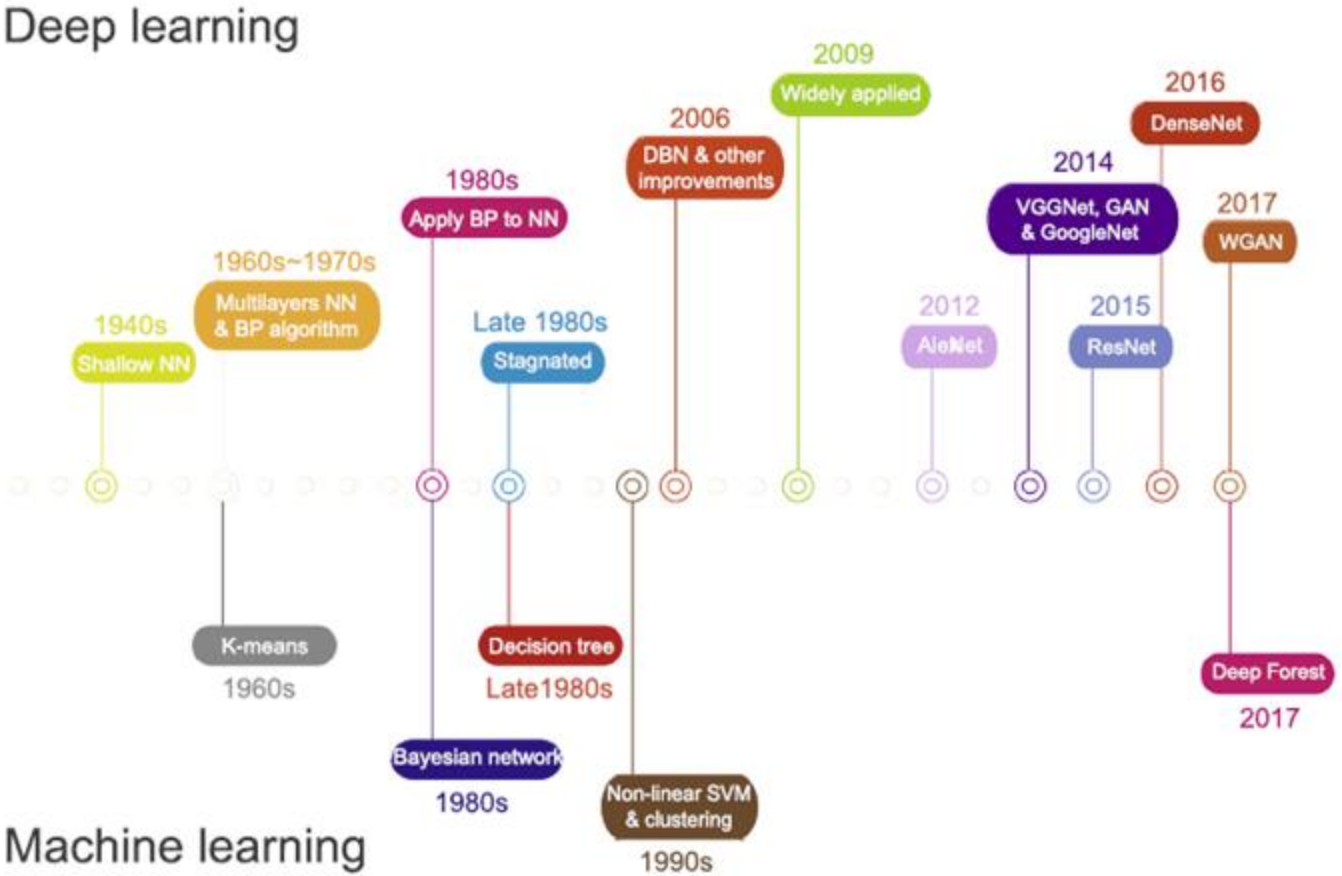
- Thanks to the improvement in computer capabilities and methodologies, ANNs with efficient backpropagation (BP) facilitated studies on **pattern recognition**.

- In a neural network with BP, classifications are first processed by the ANN model and weights are then modified by **evaluating the difference between the predicted and the true class labels**.

- Through improving the steeper gradients with BP, several learning methods were proposed: momentum, adaptive learning rate, least-squares methods, quasi-Newton methods, and conjugate gradient (CG)

The development of ANNs

- Other simple machine learning algorithms, such as support vector machines (SVMs), random forest, and k-nearest neighbors algorithms (k-NN), overtook ANNs in popularity



Potential applications in biomedicine

- **Biomarkers:** One important task in biomedicine is the translation of biological data to valid biomarkers that reflect phenotype and physical states, such as disease.

Biomarkers are critical in assessing clinical trial outcomes and detecting and monitoring disease.

Identification of sensitive specific biomarkers is a great challenge for modern translational

- **Genomic / Transcriptomic / Proteomic**

- **Drug discovery**

Challenges and limitations of deep learning systems

- **The "Black Box"**. One of the major limitations relates to quality control and interpretation.

- Most DNNs are "black boxes" that learn by simple associations and co-occurrences.

- DNNs thus lack the transparency and interpretability of other methods and are unable to **uncover** complex causal and structural relationships common in biology.

Challenges and limitations of deep learning systems

- **The Need for Large Data Sets**. Another limitation is the requirement of large training data sets. Many commonly used machine learning methods outperform DNNs where experimental data is scarce.

- When data sets are not sufficiently large, one of the major challenges is overfitting (training error is low but test error is high).

- There are ways to regularize the DNN. But overfitting is often still a threat within small biological data sets, especially with unbiased and noisy data.

Challenges and limitations of deep learning systems

- **The Selection Problem.** With many types of DNNs available, task-appropriate selection is not always straightforward.

- **The Computation Costs**. The training process is usually computationally intensive and time-consuming and often requires access to and programming knowledge for graphics processing units (GPU).

- Thus, despite the current enthusiasm for deep learning, traditional models still play an important role especially when **the amount of data is not very large or when the number of variables is large.**

Optimization

- Mathematical optimization is to find, among different possibilities, the best option given a criteria

$$\min_{x \in \Omega} C(x)$$

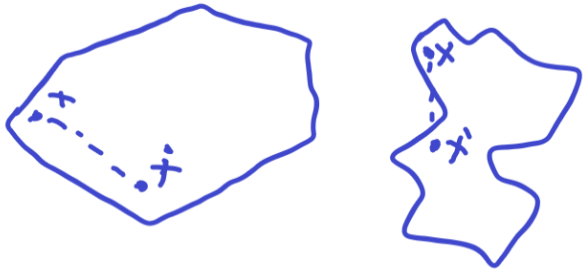
- When we use conditions over the space of solutions we have restrictions

$$\min_{x \in \Omega} C(x)$$
$$\text{given } x \gtreqless k$$

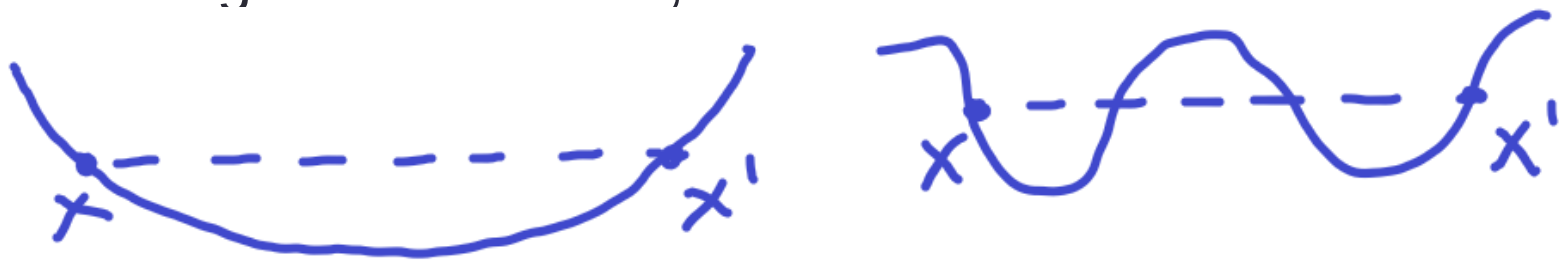
- C is commonly known as objective (cost) function

Convex and non convex problems

- A set Ω is convex if for any pair of points (x,x') all the points of the segment that join x with x' are contained in Ω



- An optimization problem is convex if the search space Ω and the objective function is convex
  - In this problems exist only one global minimum
- For the non convex problems there are not guarantee to find the global minimum,



Optimization problems

- Linear programming
- Integer programming
- Non linear programming
- Combinatory programming
- Variational calculus
- Optimal control
- Stochastic optimization

Analytic solution methods

- When the solution space Ω is continuum and the cost function is differentiable

$$C(\vec{x}) = f(\vec{x}) : \mathbb{R}^m \rightarrow \mathbb{R}$$

- There are simple and well stablished methods to find the minimum (or maximum)

- In the case where the solution are real numbers, for example m = 1, Ω = [a,b], the border numbers are evaluated as f(a) and f(b) as possible global minimum

Analytic solution methods

- Later, we evaluate the critic points x* of f that corresponds

$$\left.\frac{\delta f(x)}{dx}\right|_{x=x^*} = 0$$

$$\left.\frac{\delta^2 f(x)}{dx^2}\right|_{x=x^*} > 0 \Rightarrow x^* \text{ mínimo local}$$

$$\left.\frac{\delta^2 f(x)}{dx^2}\right|_{x=x^*} < 0 \Rightarrow x^* \text{ máximo local}$$

Analytic solution methods

- For m > 1 we use the gradient

$$\nabla f(\vec{x})\Big|_{x=x^*} = \left(\frac{\partial f\vec{x}}{\partial x_1}, \frac{\partial f\vec{x}}{\partial x_2}, \ldots, \frac{\partial f\vec{x}}{\partial x_m} \right)\Bigg|_{x=\vec{x}^*} = 0$$

- And the Hessian's matrix indicates if the point is minimum or maximum

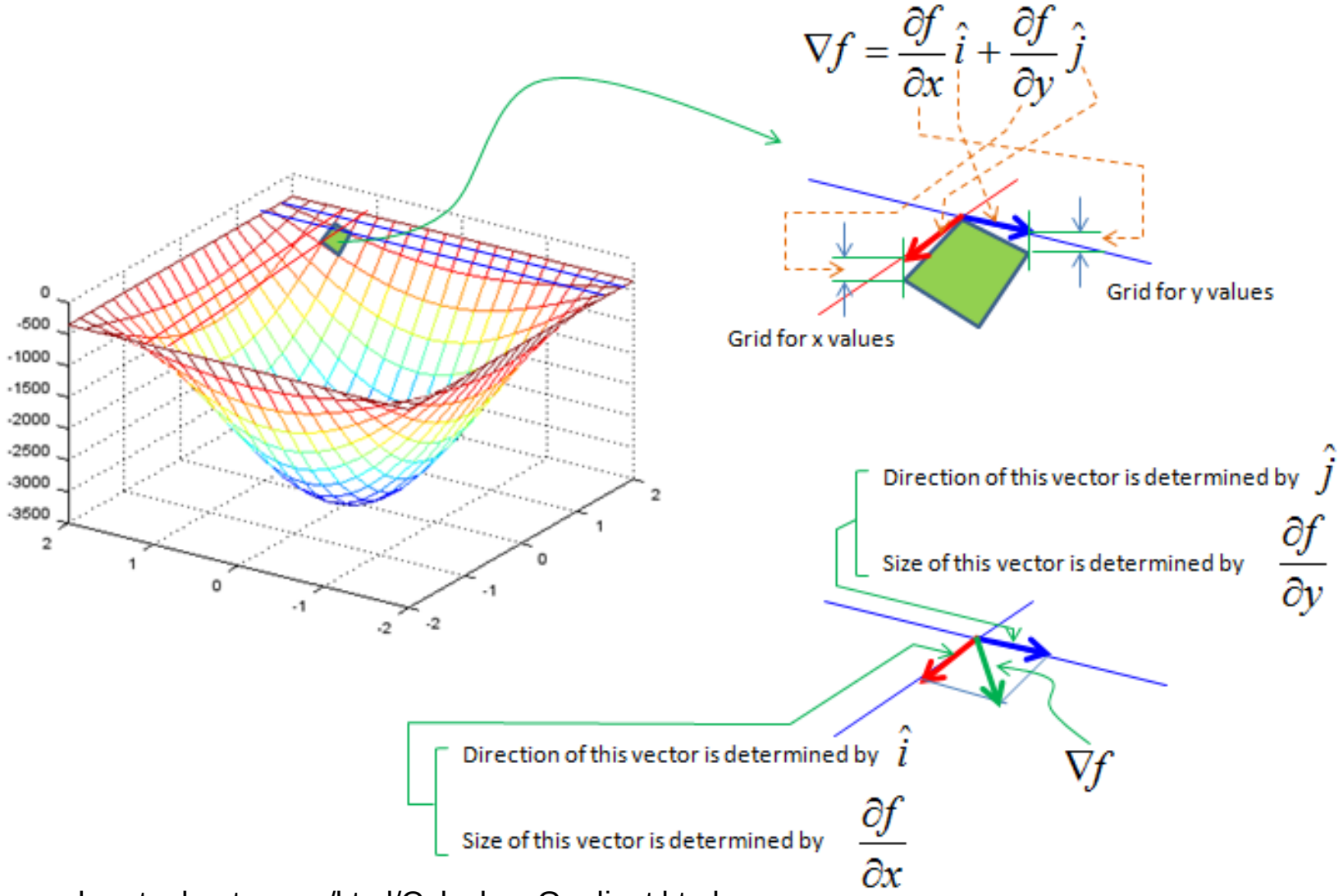
Analytic solution methods

- In practical terms, f might be differentiable but complicated to obtain the derivative or gradient being impossible to find the point analytical
- There might be multiple local minimal (even millions) being impossible to know all of them to determine the global minima
- In the previous cases is necessary to use numerical methods (for example Newton-Raphson)

Descent of the gradient

- When the solution space is a subspace of R^m and the cost function is differentiable, being in an initial point x the cost function reach the higher variation if we move in the direction of the gradient of the cost function
- Multiple iteration of the previous step tends to local minima
  - A problem is when we begin in a local minima, so, in some cases, we might want to move in a direction even when the gradient indicates a contrary direction

Descent of the gradient



https://www.sharetechnote.com/html/Calculus_Gradient.html

Lecture 2

Machine Learning for Beginners: An Introduction to Neural Networks

A simple explanation of how they work and how to implement one from scratch in Python.

Victor Zhou Follow

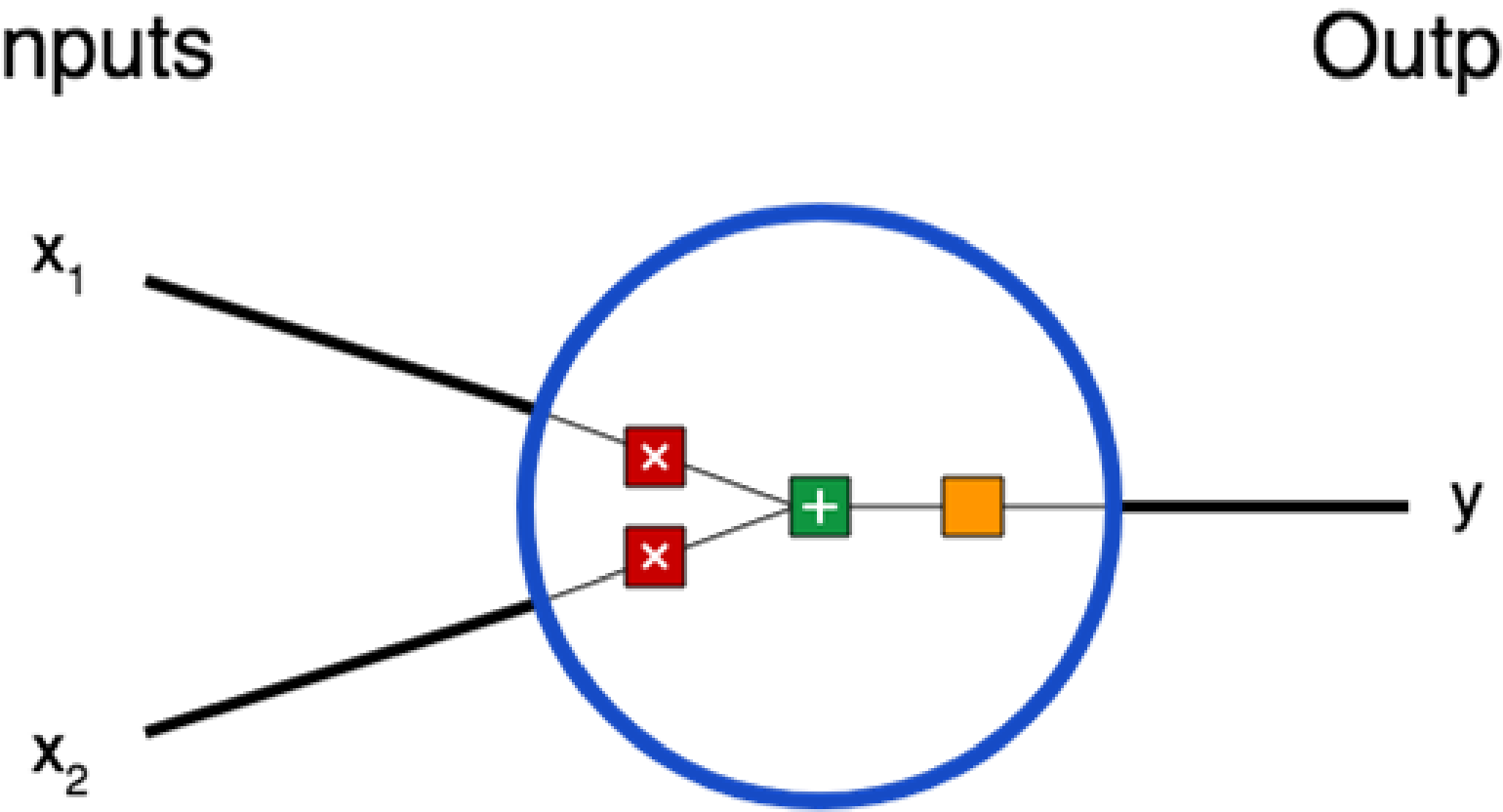
Mar 5, 2019 · 9 min read ★

https://victorzhou.com/blog/intro-to-neural-networks/

"En matemáticas no entiendes las cosas. Solo te acostumbras a ellas" - John Von Neumann

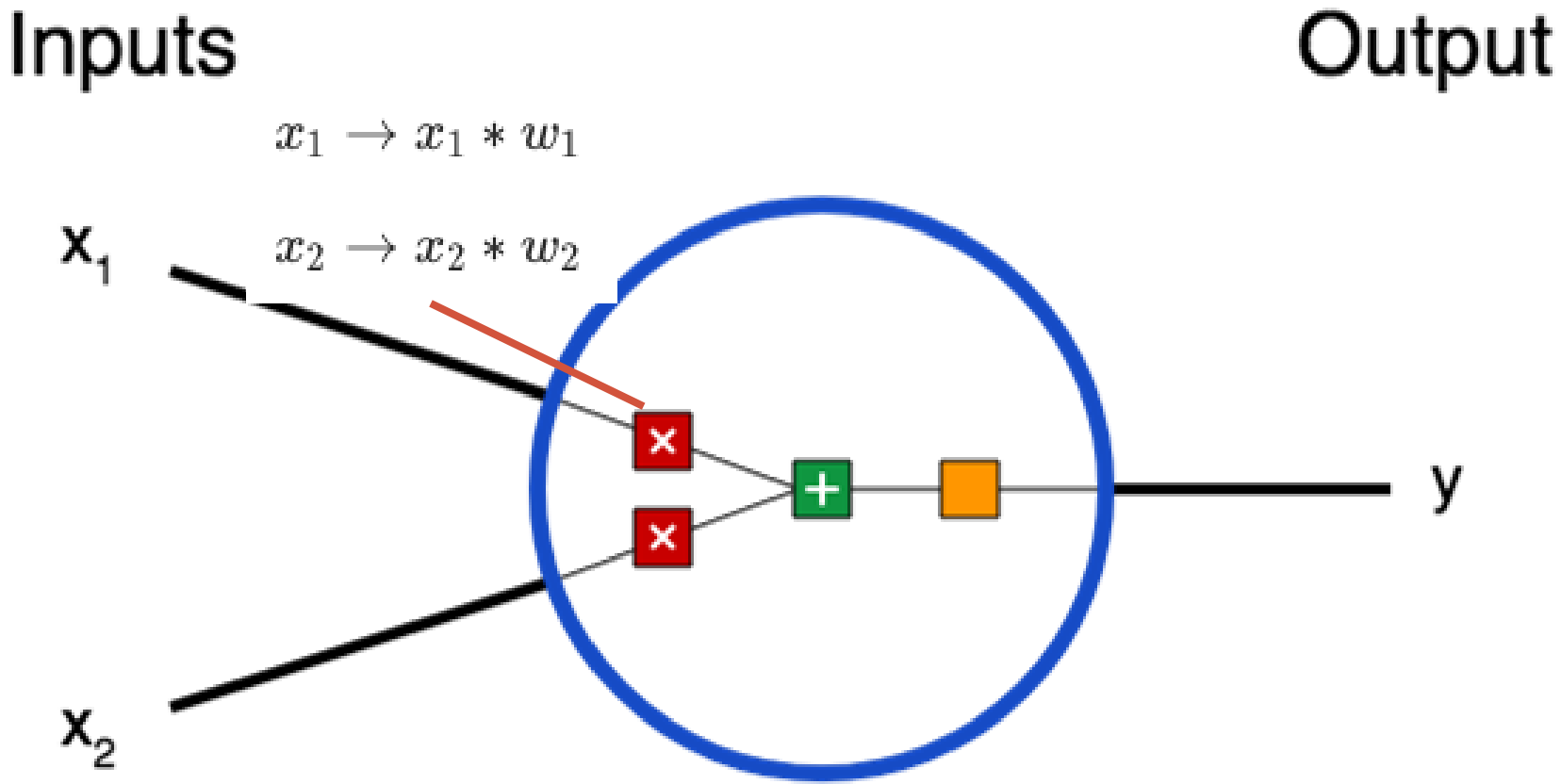
Building blocks: Neurons

- A neuron takes inputs, does some math with them, and produces one output.



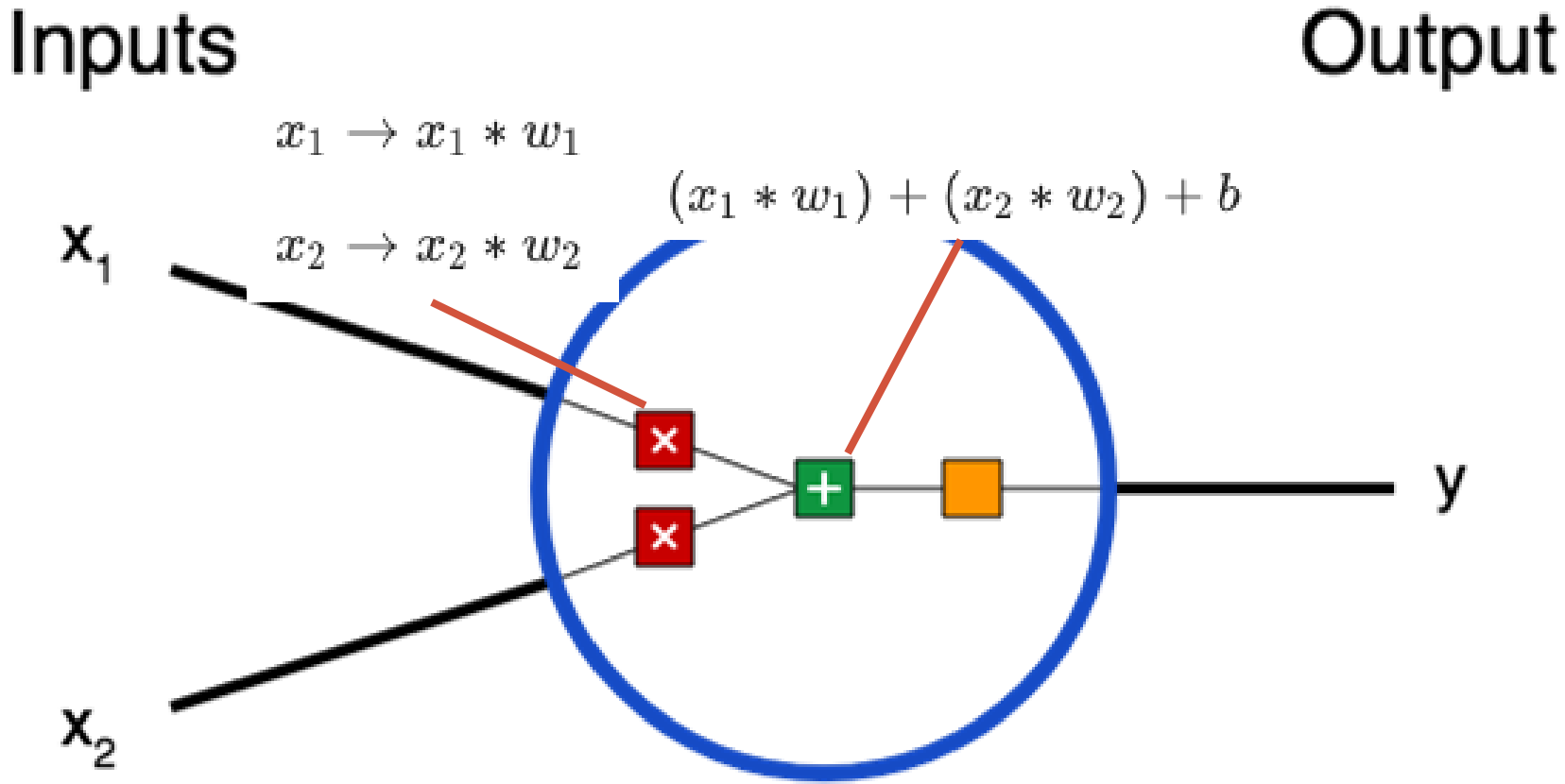
Building blocks: Neurons

- A neuron takes inputs, does some math with them, and produces one output.



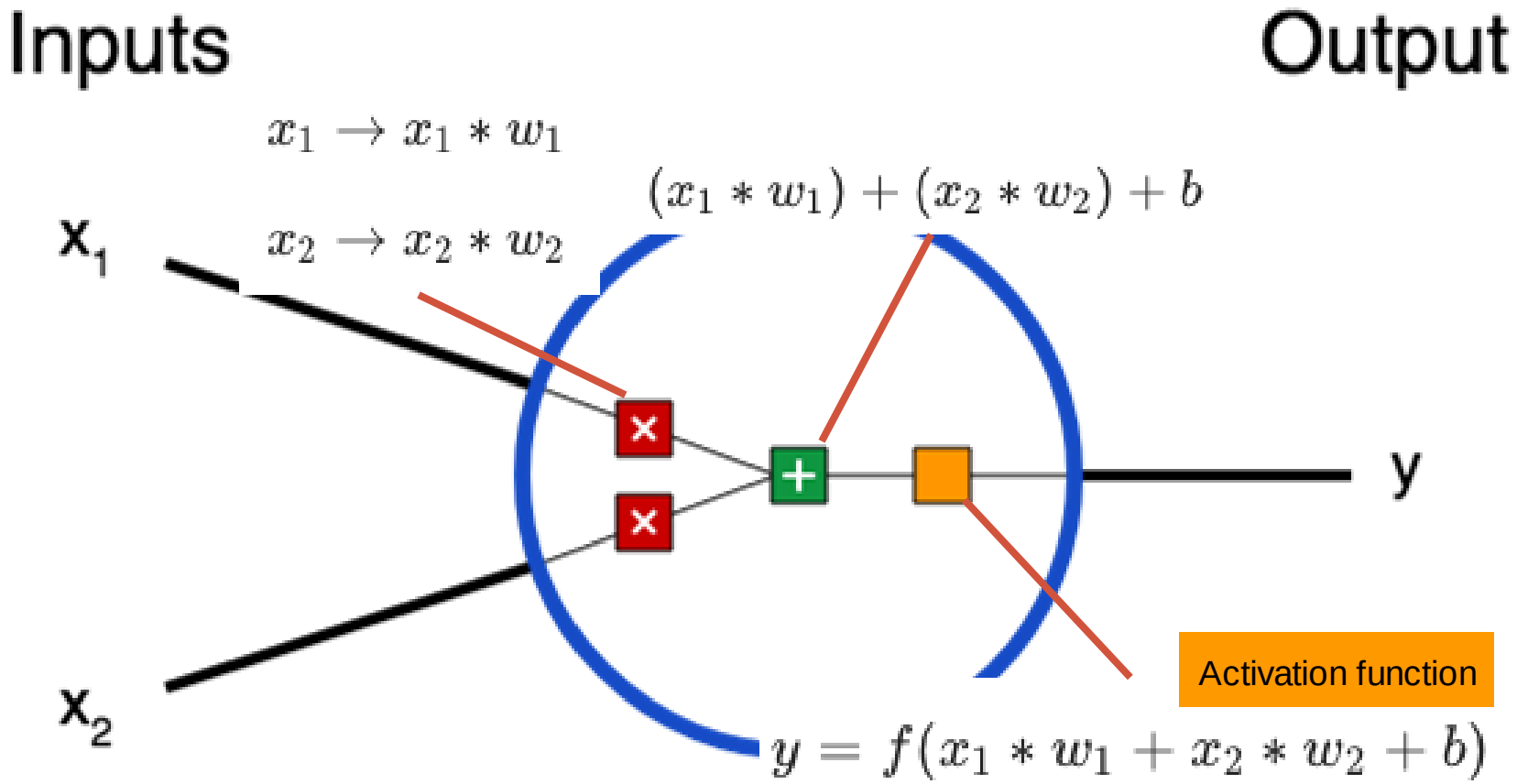
Building blocks: Neurons

- A neuron takes inputs, does some math with them, and produces one output.



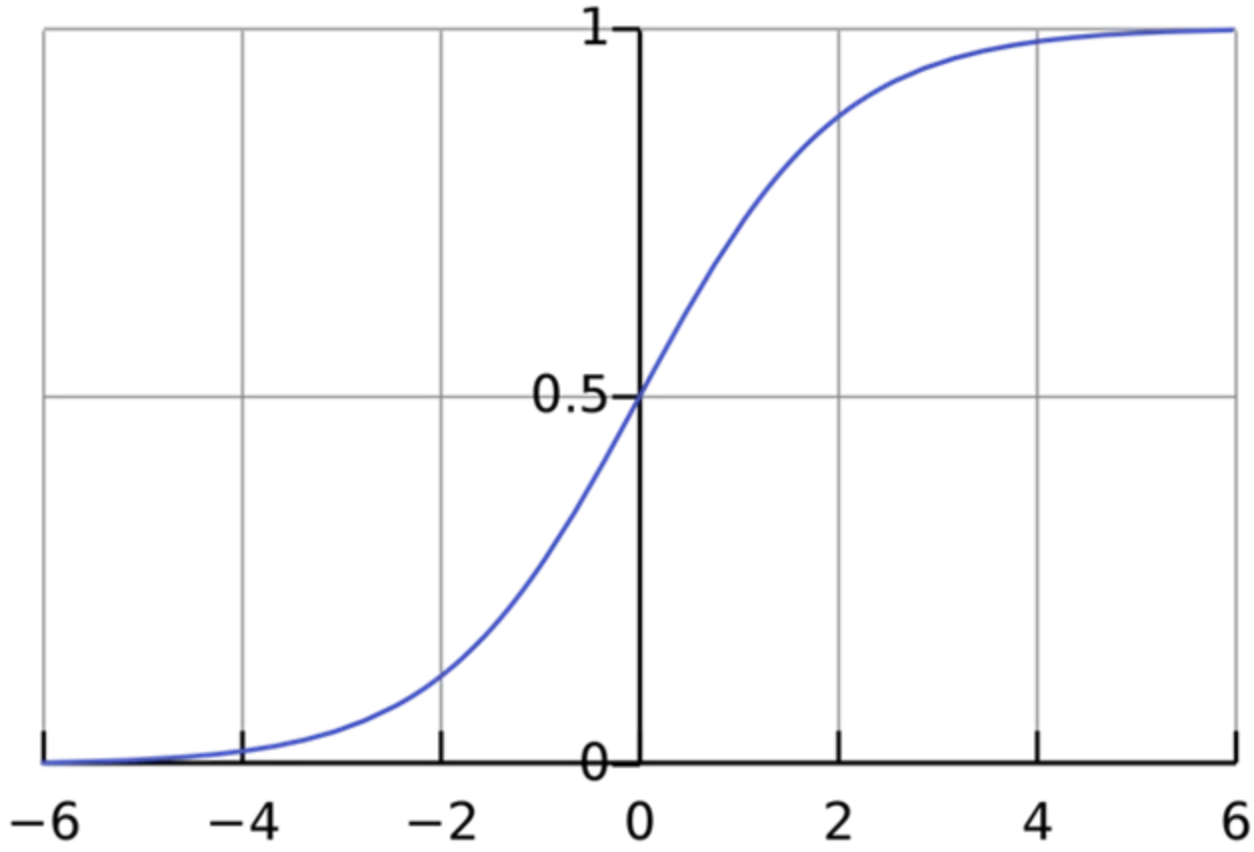
Building blocks: Neurons

- A neuron takes inputs, does some math with them, and produces one output.



Building blocks: Neurons

- The activation function is used to turn an unbounded input into an output that has a nice, predictable form

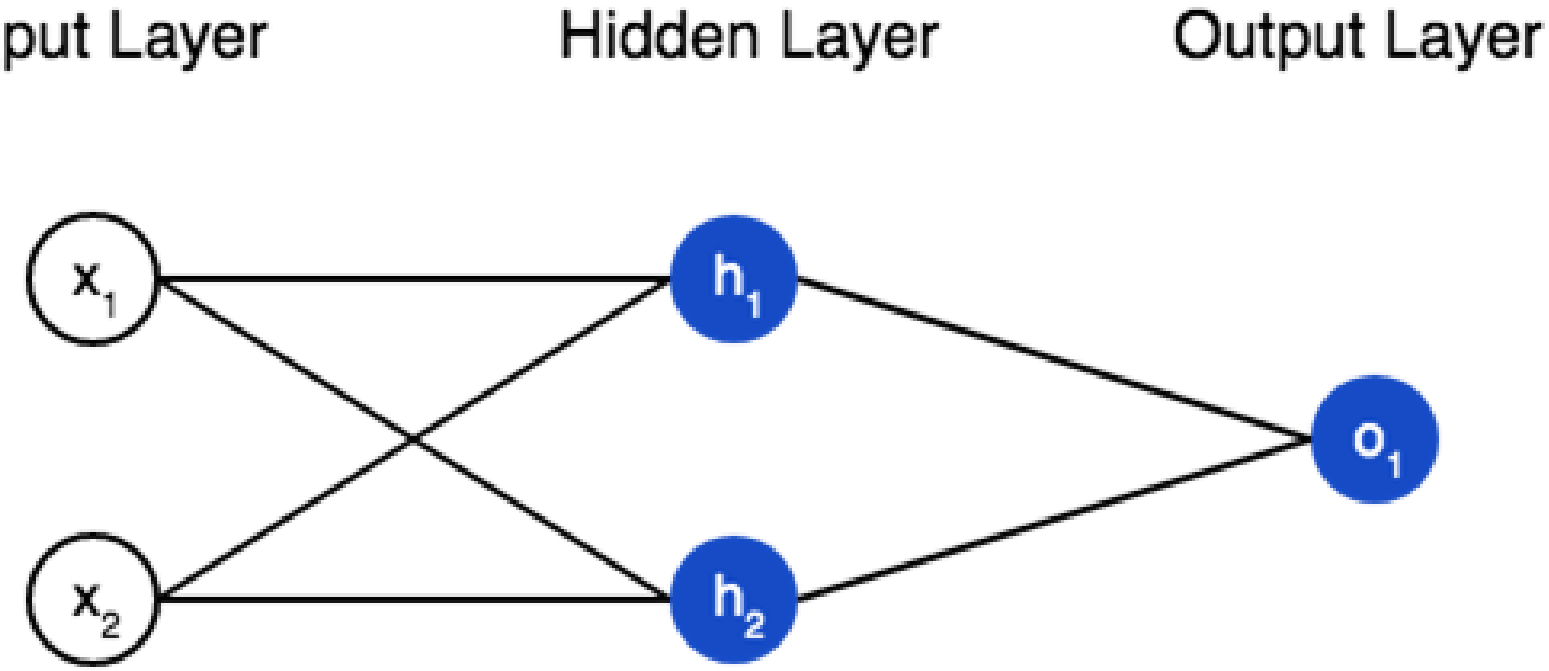


Building blocks: Neurons

```
import numpy as np

def sigmoid(x):
  # Our activation function: f(x) = 1 / (1 + e^(-x))
  return 1 / (1 + np.exp(-x))

class Neuron:
  def __init__(self, weights, bias):
    self.weights = weights
    self.bias = bias

  def feedforward(self, inputs):
    # Weight inputs, add bias, then use the activation function
    total = np.dot(self.weights, inputs) + self.bias
    return sigmoid(total)

weights = np.array([0, 1]) # w1 = 0, w2 = 1
bias = 4                   # b = 4
n = Neuron(weights, bias)

x = np.array([2, 3])       # x1 = 2, x2 = 3
print(n.feedforward(x))    # 0.9990889488055994
```

Combining Neurons into a Neural Network

- A neural network is collection of neurons connected.



This network has 2 inputs, a hidden layer with 2 neurons and an output layer with 1 neuron.

Combining Neurons into a Neural Network

- A neural network can have any number of layers with any number of neurons in those layers.
- The basic idea is: feed the input(s) forward through the neurons in the network to get the output(s) at the end.

Combining Neurons into a Neural Network

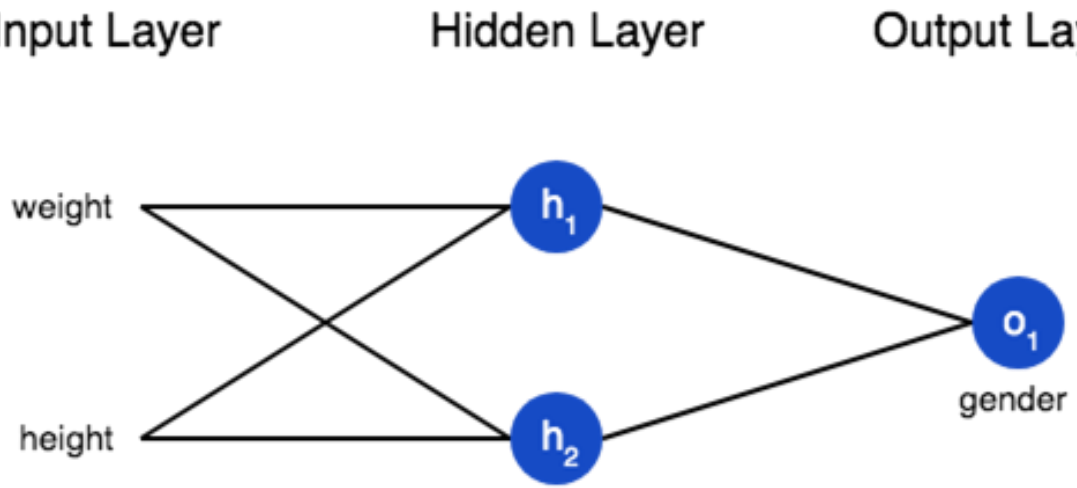
```
import numpy as np

# ... code from previous section here

class OurNeuralNetwork:
  '''
  A neural network with:
    - 2 inputs - a hidden layer with 2 neurons (h1, h2) - an output layer with 1 neuron (o1)
  Each neuron has the same weights and bias:
    - w = [0, 1] - b = 0
  '''
  def __init__(self):
    weights = np.array([0, 1])
    bias = 0

    # The Neuron class here is from the previous section
    self.h1 = Neuron(weights, bias)
    self.h2 = Neuron(weights, bias)
    self.o1 = Neuron(weights, bias)

  def feedforward(self, x):
    out_h1 = self.h1.feedforward(x)
    out_h2 = self.h2.feedforward(x)

    # The inputs for o1 are the outputs from h1 and h2
    out_o1 = self.o1.feedforward(np.array([out_h1, out_h2]))

    return out_o1
```

```
network = OurNeuralNetwork()
x = np.array([2, 3])
print(network.feedforward(x)) # 0.7216325609518421
```

Training a Neural Network, Part 1

- Let's train the network to predict gender

Name	Weight (lb)	Height (in)	Gender
Alice	133	65	F
Bob	160	72	M
Charlie	152	70	M
Diana	120	60	F



Training a Neural Network, Part 1

- Organize the data (remove the mean, replace NaN, etc)

Name	Weight (lb)	Height (in)	Gender
Alice	133	65	F
Bob	160	72	M
Charlie	152	70	M
Diana	120	60	F

Name	Weight (minus 135)	Height (minus 66)	Gender
Alice	-2	-1	1
Bob	25	6	0
Charlie	17	4	0
Diana	-15	-6	1

Training a Neural Network, Part 1

Loss

- Is necessary a way to quantify how “good” is classifying the networks so that it can try to do “better”.

- The mean squared error (MSE) loss:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{pred})^2$$

- Better predictions = Lower loss.

- Training a network = trying to minimize its loss.

Training a Neural Network, Part 1

Loss

- Example:

Name	y_{true}	y_{pred}	$(y_{true} - y_{pred})^2$
Alice	1	0	1
Bob	0	0	0
Charlie	0	0	0
Diana	1	0	1

$$\text{MSE} = \frac{1}{4}(1 + 0 + 0 + 1) = \boxed{0.5}$$

Training a Neural Network, Part 1

Loss

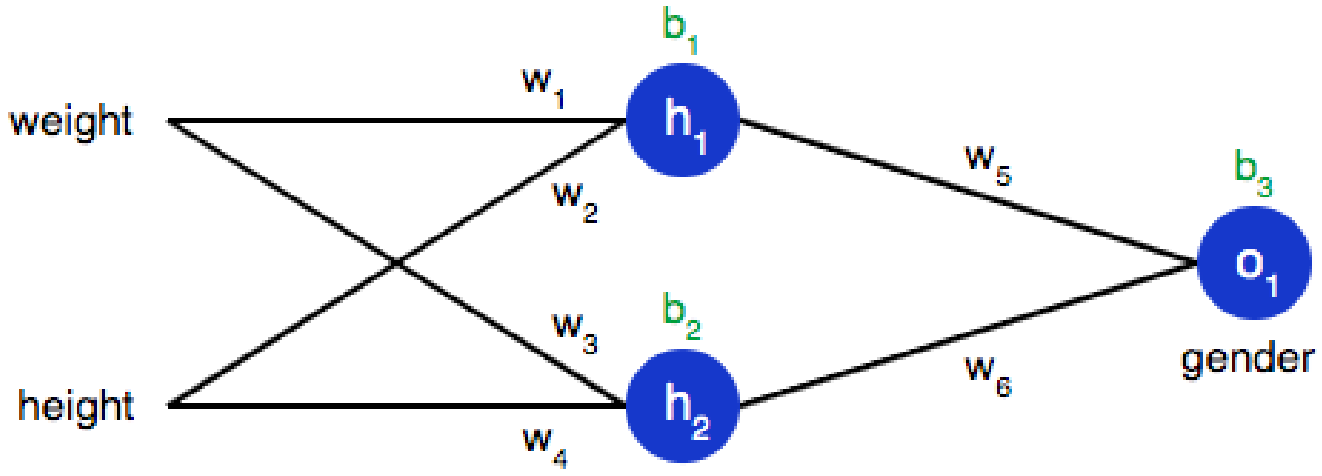
- MSE in python

```
import numpy as np

def mse_loss(y_true, y_pred):
  # y_true and y_pred are numpy arrays of the same length.
  return ((y_true - y_pred) ** 2).mean()

y_true = np.array([1, 0, 0, 1])
y_pred = np.array([0, 0, 0, 0])

print(mse_loss(y_true, y_pred)) # 0.5
```

Training a Neural Network, Part 2

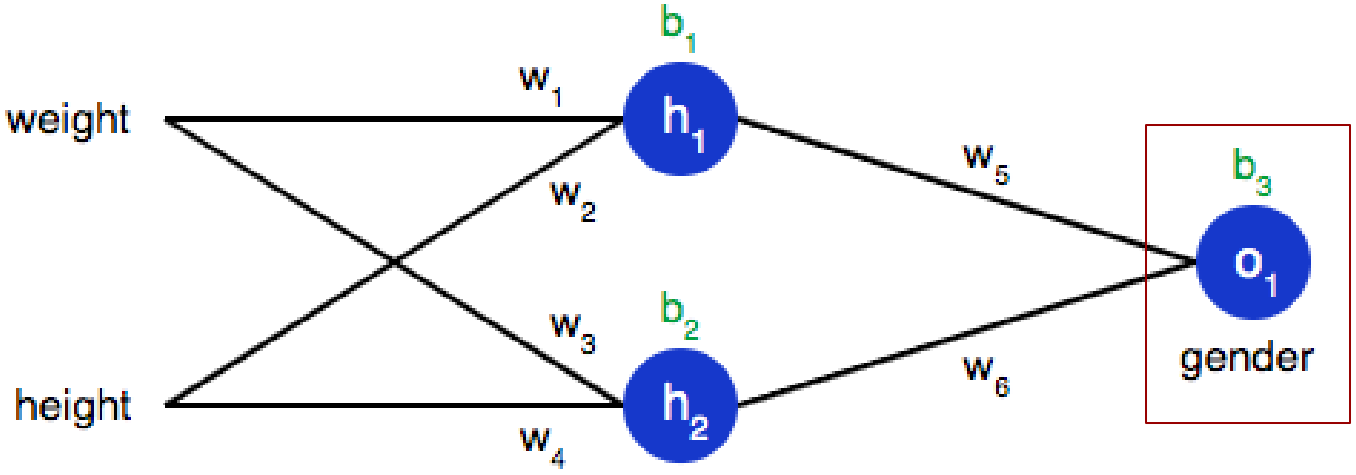
The Loss function depends of the parameters of the network

$$\begin{aligned}\text{MSE} &= \frac{1}{1}\sum_{i=1}^{1}(y_{true} - y_{pred})^2 \\ &= (y_{true} - y_{pred})^2 \\ &= (1 - y_{pred})^2\end{aligned}$$



$$L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$$

Training a Neural Network, Part 2



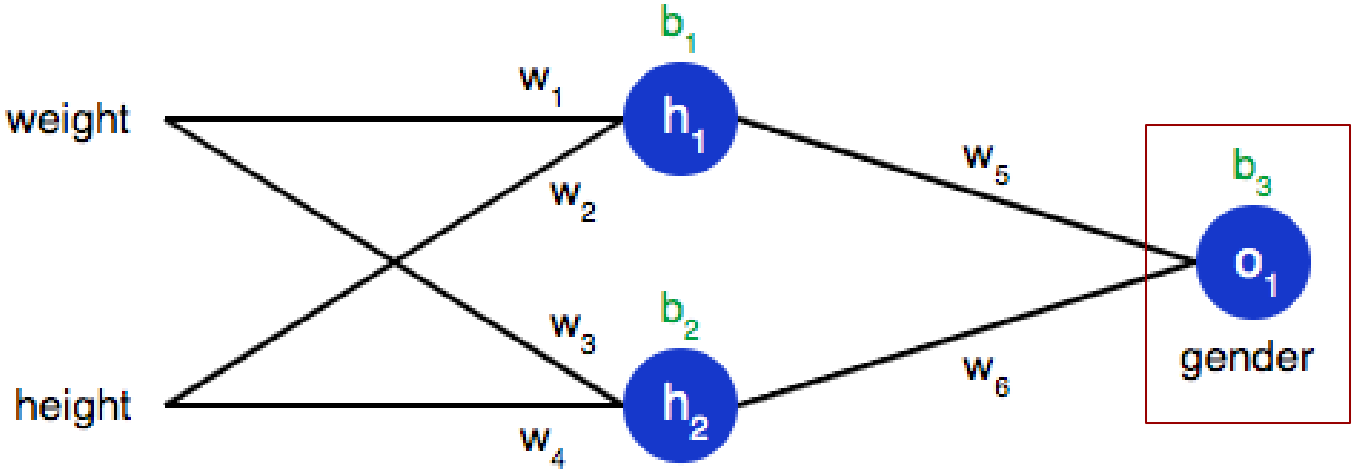
How would loss L change if we changed w1?

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial w_1}$$

$$\frac{\partial L}{\partial y_{pred}} = \frac{\partial (1 - y_{pred})^2}{\partial y_{pred}} = \boxed{-2(1 - y_{pred})}$$

$$y_{pred} = o_1 = f(w_5 h_1 + w_6 h_2 + b_3)$$

Training a Neural Network, Part 2



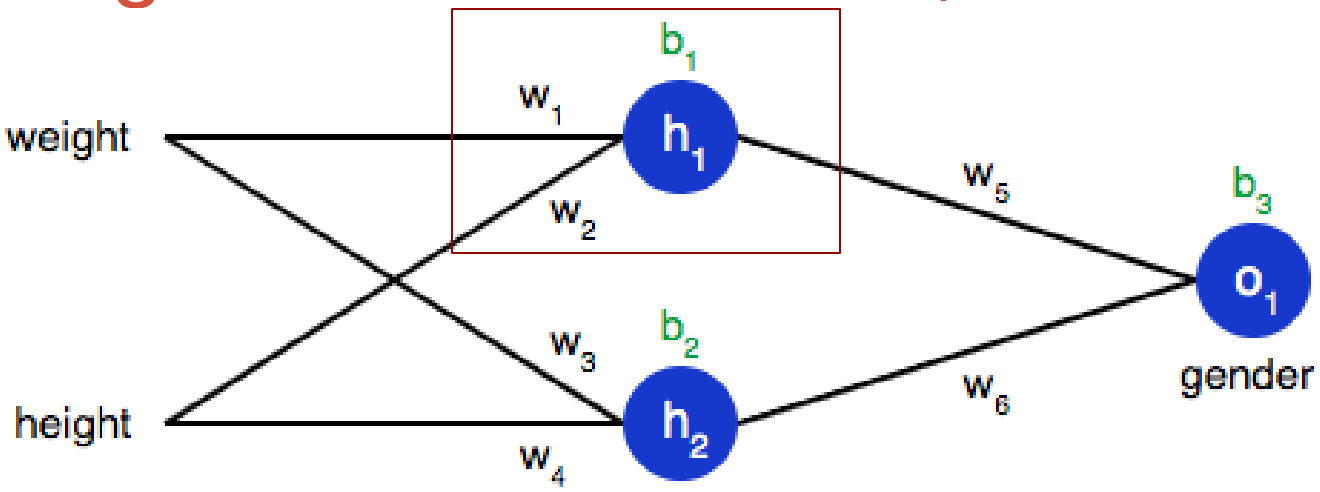
How would loss L change if we changed w1?

$$y_{pred} = o_1 = f(w_5 h_1 + w_6 h_2 + b_3)$$

$$\frac{\partial y_{pred}}{\partial w_1} = \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}$$

$$\frac{\partial y_{pred}}{\partial h_1} = \boxed{w_5 * f'(w_5 h_1 + w_6 h_2 + b_3)}$$

Training a Neural Network, Part 2



How would loss L change if we changed w1?

$$h_1 = f(w_1 x_1 + w_2 x_2 + b_1)$$

$$\frac{\partial h_1}{\partial w_1} = \boxed{x_1 * f'(w_1 x_1 + w_2 x_2 + b_1)}$$

We are using the same activation function for each neuron

$$f(x) = \frac{1}{1 + e^{-x}} \qquad f'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = f(x) * (1 - f(x))$$

Training a Neural Network, Part 2

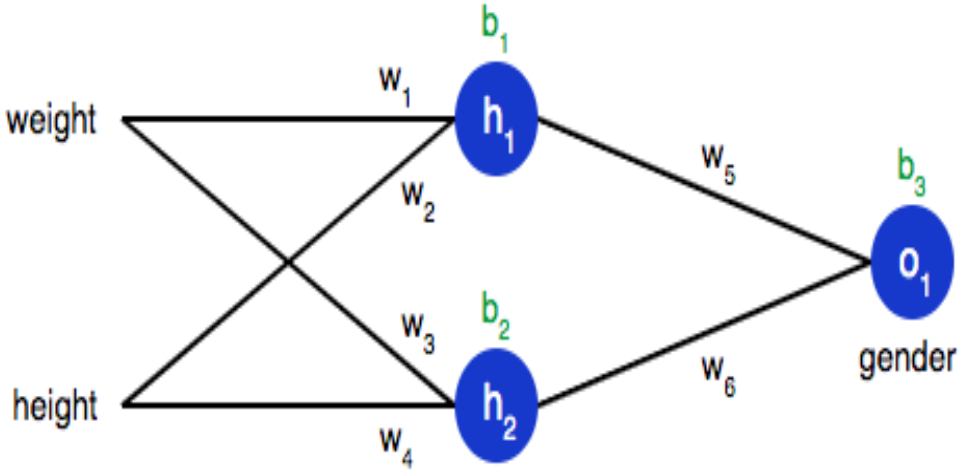
This system of calculating partial derivatives by working backwards is known as **backpropagation**, or “backprop”.

Training a Neural Network, Part 2

Backprop example

Name	Weight (minus 135)	Height (minus 66)	Gender
Alice	-2	-1	1

Let's initialize all the weights to 1 and all the biases to 0. If we do a feedforward pass through the network, we get:



$$h_1 = f(w_1 x_1 + w_2 x_2 + b_1)$$
$$= f(-2 + -1 + 0)$$
$$= 0.0474$$

$$h_2 = f(w_3 x_1 + w_4 x_2 + b_2) = 0.0474$$

$$o_1 = f(w_5 h_1 + w_6 h_2 + b_3)$$
$$= f(0.0474 + 0.0474 + 0)$$
$$= 0.524$$

Training a Neural Network, Part 2

Backprop example

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}$$

$$\begin{aligned}\frac{\partial L}{\partial y_{pred}} &= -2(1 - y_{pred}) \\ &= -2(1 - 0.524) \\ &= -0.952\end{aligned}$$

$$\begin{aligned}\frac{\partial y_{pred}}{\partial h_1} &= w_5 * f'(w_5 h_1 + w_6 h_2 + b_3) \\ &= 1 * f'(0.0474 + 0.0474 + 0) \\ &= f(0.0948) * (1 - f(0.0948)) \\ &= 0.249\end{aligned}$$

$$\begin{aligned}\frac{\partial h_1}{\partial w_1} &= x_1 * f'(w_1 x_1 + w_2 x_2 + b_1) \\ &= -2 * f'(-2 + -1 + 0) \\ &= -2 * f(-3) * (1 - f(-3)) \\ &= -0.0904\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial w_1} &= -0.952 * 0.249 * -0.0904 \\ &= \boxed{0.0214}\end{aligned}$$

This tells us that if we were to increase w1 then L would increase around 0.0214.

Training: Stochastic Gradient Descent

To change our weights and biases, to minimize loss, we use the following rule (derived from the gradient method):

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

η is a constant called the **learning rate** that controls how fast we train

If we do this for every weight and bias in the network, the loss will slowly decrease and our network will improve.

Training: Stochastic Gradient Descent

Our training process will look like this:

1. Choose one sample from our dataset. We only operate on one sample at a time.
2. Calculate all the partial derivatives of loss with respect to weights or biases.
3. Use the update equation to update each weight and bias.
4. Go back to step 1.

Training: Stochastic Gradient Descent

```
def train(self, data, all_y_trues):
    '''
    - data is a (n x 2) numpy array, n = # of samples in the dataset.
    - all_y_trues is a numpy array with n elements.
      Elements in all_y_trues correspond to those in data.
    '''
    learn_rate = 0.1
    epochs = 1000 # number of times to loop through the entire dataset

    for epoch in range(epochs):
      for x, y_true in zip(data, all_y_trues):
        # --- Do a feedforward (we'll need these values later)
        sum_h1 = self.w1 * x[0] + self.w2 * x[1] + self.b1
        h1 = sigmoid(sum_h1)

        sum_h2 = self.w3 * x[0] + self.w4 * x[1] + self.b2
        h2 = sigmoid(sum_h2)

        sum_o1 = self.w5 * h1 + self.w6 * h2 + self.b3
        o1 = sigmoid(sum_o1)
        y_pred = o1

        # --- Calculate partial derivatives.
        # --- Naming: d_L_d_w1 represents "partial L / partial w1"
        d_L_d_ypred = -2 * (y_true - y_pred)
```

Training: Stochastic Gradient Descent

```
    # Neuron o1
    d_ypred_d_w5 = h1 * deriv_sigmoid(sum_o1)
    d_ypred_d_w6 = h2 * deriv_sigmoid(sum_o1)
    d_ypred_d_b3 = deriv_sigmoid(sum_o1)

    d_ypred_d_h1 = self.w5 * deriv_sigmoid(sum_o1)
    d_ypred_d_h2 = self.w6 * deriv_sigmoid(sum_o1)

    # Neuron h1
    d_h1_d_w1 = x[0] * deriv_sigmoid(sum_h1)
    d_h1_d_w2 = x[1] * deriv_sigmoid(sum_h1)
    d_h1_d_b1 = deriv_sigmoid(sum_h1)

    # Neuron h2
    d_h2_d_w3 = x[0] * deriv_sigmoid(sum_h2)
    d_h2_d_w4 = x[1] * deriv_sigmoid(sum_h2)
    d_h2_d_b2 = deriv_sigmoid(sum_h2)

    # --- Update weights and biases
    # Neuron h1
    self.w1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w1
    self.w2 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w2
    self.b1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_b1
```

Training: Stochastic Gradient Descent

```
        # Neuron h2
        self.w3 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w3
        self.w4 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w4
        self.b2 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_b2

        # Neuron o1
        self.w5 -= learn_rate * d_L_d_ypred * d_ypred_d_w5
        self.w6 -= learn_rate * d_L_d_ypred * d_ypred_d_w6
        self.b3 -= learn_rate * d_L_d_ypred * d_ypred_d_b3

      # --- Calculate total loss at the end of each epoch
      if epoch % 10 == 0:
        y_preds = np.apply_along_axis(self.feedforward, 1, data)
        loss = mse_loss(all_y_trues, y_preds)
        print("Epoch %d loss: %.3f" % (epoch, loss))
```

Machine Learning and Scikit learn

Machine Learning: Is the field of teaching machines and computers to learn from existing data to make predictions on new (unseen) data.

There are three types of Machine Learning Algorithms:

- Supervised
- Unsupervised
- Reinforcement Learning

Machine Learning and Scikit learn

Supervised Learning: we have a target/outcome variable which is to be predicted from a given set of features/independent variables.

The training process is done until the model achieves the desired level of accuracy on the training data, which is then used on the new unseen data.

Machine Learning and Scikit learn

There are two types of supervised machine learning algorithms: Classification and Regression.

- Classification models predict a categorical label (health vs disease).
- Regression models predict a continuous label like a clinical scale

Machine Learning and Scikit learn

Scikit - Learn, or sklearn, is one of the most popular libraries in Python for doing supervised machine learning.

It integrates well with the SciPy stack, making it robust and powerful.

Scikit-learn can be used for both classification and regression problems.

Machine Learning and Scikit learn

Classification

An Aesop's Fable: The Boy Who Cried Wolf (*compressed*)

A shepherd boy gets bored tending the town's flock. To have some fun, he cries out, "Wolf!" even though no wolf is in sight. The villagers run to protect the flock, but then get really mad when they realize the boy was playing a joke on them.

[Iterate previous paragraph *N* times.]

One night, the shepherd boy sees a real wolf approaching the flock and calls out, "Wolf!" The villagers refuse to be fooled again and stay in their houses. The hungry wolf turns the flock into lamb chops. The town goes hungry. Panic ensues.

Let's make the following definitions:

- "Wolf" is a **positive class**.
- "No wolf" is a **negative class**.

Machine Learning and Scikit learn

Classification

Let's make the following definitions:

- "Wolf" is a **positive class**.
- "No wolf" is a **negative class**.

True Positive (TP): • Reality: A wolf threatened. • Shepherd said: "Wolf." • Outcome: Shepherd is a hero.	**False Positive (FP):** • Reality: No wolf threatened. • Shepherd said: "Wolf." • Outcome: Villagers are angry at shepherd for waking them up.
False Negative (FN): • Reality: A wolf threatened. • Shepherd said: "No wolf." • Outcome: The wolf ate all the sheep.	**True Negative (TN):** • Reality: No wolf threatened. • Shepherd said: "No wolf." • Outcome: Everyone is fine.

Machine Learning and Scikit learn

Classification

A **true positive (TP)** is an outcome where the model correctly predicts the **positive class**.

A **true negative (TN)** is an outcome where the model correctly predicts the negative class.

A **false positive (FP)** is an outcome where the model incorrectly predicts the p**ositive class**.

A **false negative (FN)** is an outcome where the model incorrectly predicts the **negative class**.

Machine Learning and Scikit learn

Evaluation Metrics

1. Accuracy: is the fraction of cases correctly classified. For a binary classifier, it is represented as accuracy = (TP+TN)/(TP+TN+FP+FN)

True Positive (TP): • Reality: Malignant • ML model predicted: Malignant • **Number of TP results: 1**	**False Positive (FP):** • Reality: Benign • ML model predicted: Malignant • **Number of FP results: 1**
False Negative (FN): • Reality: Malignant • ML model predicted: Benign • **Number of FN results: 8**	**True Negative (TN):** • Reality: Benign • ML model predicted: Benign • **Number of TN results: 90**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{1 + 90}{1 + 90 + 1 + 8} = 0.91$$

Machine Learning and Scikit learn

Evaluation Metrics

1. Accuracy: is the fraction of cases correctly classified. For a binary classifier, it is represented as accuracy = (TP+TN)/(TP+TN+FP+FN)

True Positive (TP):

- Reality: Malignant
- ML model predicted: Malignant
- **Number of TP results: 1**

False Positive (FP):

- Reality: Benign
- ML model predicted: Malignant
- **Number of FP results: 1**

False Negative (FN):

- Reality: Malignant
- ML model predicted: Benign
- **Number of FN results: 8**

True Negative (TN):

- Reality: Benign
- ML model predicted: Benign
- **Number of TN results: 90**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{1 + 90}{1 + 90 + 1 + 8} = 0.91$$

Machine Learning and Scikit learn

Evaluation Metrics

2 Precision: What proportion of positive identifications was actually correct?
P = TP / (TP+ FP)

Following the previous example:

True Positives (TPs): 1	False Positives (FPs): 1
False Negatives (FNs): 8	True Negatives (TNs): 90

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{1}{1 + 1} = 0.5$$

Our model has a precision of 0.5—in other words, when it predicts a tumor is malignant, it is correct 50% of the time.

A model that produces no false positives has a precision of 1.0.

Machine Learning and Scikit learn

Evaluation Metrics

3 Recall: What proportion of actual positives was identified correctly? R = TP / (TP+FN)

True Positives (TPs): 1	False Positives (FPs): 1
False Negatives (FNs): 8	True Negatives (TNs): 90

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{1}{1 + 8} = 0.11$$

Our model has a recall of 0.11—in other words, it correctly identifies 11% of all malignant tumors.

A model that produces no false negatives has a recall of 1.0.

Machine Learning and Scikit learn

Evaluation Metrics

4 F1-score: is a weighted average of precision and recall. It is represented as F1 =2(PR) / (P+R)

Layers

A layer groups a number of neurons together.

- There will always be an input and output layer.

We can have zero or more hidden layers in a neural network.

The learning process of a neural network is performed with the layers.

The neurons, within each of the layer of a neural network, perform the same function.

- They simply calculate the weighted sum of inputs and weights, add the bias and execute an activation function.

Layers

Input Layer

The input layer is responsible for receiving the inputs.

There must always be one input layer in a neural network.

The input layer takes in the inputs, performs the calculations via its neurons and then the output is transmitted onto the subsequent layers.

Layers

Input Layer

The number of neurons in an input layer is dependent on the shape of your training data.

$$\text{number\_neurons} = \text{number\_features} + 1$$

One additional node is to capture the bias term.

Layers

Output Layer

The output layer is responsible for producing the final result. There must always be one output layer in a neural network.

Layers

Output Layer

How Many Neurons In Output Layer?

- If your neural network is a regressor, then the output layer has a single node.
- If your neural network is a classifier, then it also has a single node.
- If you use a probabilistic activation function such as softmax then the output layer has one node per class label in your model.

Layers

Hidden Layer

Hidden layers reside in-between input and output layers and this is the primary reason why they are referred to as hidden.

There could be zero or more hidden layers in a neural network.

Layers

Table: Determining the Number of Hidden Layers

Num Hidden Layers	Result
none	Only capable of representing linear separable functions or decisions.
1	Can approximate any function that contains a continuous mapping from one finite space to another.
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.
>2	Additional layers can learn complex representations (sort of automatic feature engineering) for layer layers.

Layers

Hidden Layer

Usually, each hidden layer contains the same number of neurons.

The larger the number of hidden layers in a neural network, the longer it will take for the neural network to produce the output and the more complex problems the neural network can solve.

Layers

Hidden Layer

The optimum number of neurons hidden layers can be determined by:

number_neurons = (trading data samples)/(factor*(input_neurons+output_neurons))

The factor is used to prevent overfitting and it is a number between 1–10.

Layers

Hidden Layer

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

https://www.heatonresearch.com/2017/06/01/hidden-layers.html

Cross-validation: evaluating estimator performance

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake

- A model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data.

Cross-validation: evaluating estimator performance

This situation is called overfitting.

To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a test set X_test, y_test. N

Cross-validation: evaluating estimator performance

When evaluating different settings (“hyperparameters”) for estimators there is still a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally.

Cross-validation: evaluating estimator performance

This way, knowledge about the test set can “leak” into the model and evaluation metrics no longer report on generalization performance.

- To solve this problem, yet another part of the dataset can be held out as a so-called “validation set”: training proceeds on the training set, after which evaluation is done on the validation set, and when the experiment seems to be successful, final evaluation can be done on the test set.

Cross-validation: evaluating estimator performance

However, by partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets.

Cross-validation: evaluating estimator performance

A solution to this problem is a procedure called cross-validation (CV).

- A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV.

Cross-validation: evaluating estimator performance

In the basic approach, called k-fold CV, the training set is split into k smaller sets (other approaches follow the same principles). The following procedure is followed for each of the k “folds”:

- A model is trained using k-1 of the folds as training data;
- the resulting model is validated on the remaining part of the data (it is used as a test set to compute a performance measure such as accuracy).

Cross-validation: evaluating estimator performance

The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop.

This approach can be computationally expensive, but does not waste too much data, which is a major advantage in problem when the number of samples is very small.