THE UNIVERSITY OF MELBOURNE

Neural networks I: Perceptron

Semester 1, 2021

Kris Ehinger

Outline

- Representation learning
- Introduction to neural networks
- Perceptron
- Multilayer perceptron

Representation learning

"yes"

"no"

Example: Text representation

- How to represent the main idea of a text?
- Simple option: "bag of words"
  - Vector representing word frequencies
  - Values within the vector represent word count
  - Discards word order and context


intelligence
can
human
system
artificial
used
machine
systems
humans
machines
will
consider
people
become
reward
world
society
time
companies
training
way
intelligent
better
Artificial
work
phase
already
like
animals
just
new
right
ways
learning
labour
keep
might
capable
beings
attention
suffering
create
instances
many
tools
individuals
comes
progress
adversaries
bring
happen
Robot
aversion
weapons
well
raters
trucks
frontier
created
video
bias
next
becomes
make
ethical
performs
genetic
three
Tech
software
cancer
form
faster
jobs
hand
Eugene
Musk
fooled
algorithms
company
mechanisms
economy
majority
fair
look
revenues
service
Google
building
robots
cognitive
wealth
point
advanced
opportunity
physical
games
bots
day
now
use
wrong
may
think
economic
act
hourly
happens
example
Alphabet
means
relationships
similar
Valley
question
according
Elon
biggest
fewer
consequences
examples
workforce
technology
negative
Goostman
patterns
Silicon
families
input
whether
complex
rely
still
one
Intelligence
control
headlines
often
self-driving
using
risk
even
end
see
becoming
everyone
we're


dive
species
operators
Marine
site
coral
Reef
diving
sharks
sea
fish
reef
Bay
experience
Park
diver
Island
Shoals
Rowley
Australia
Ningaloo
sites
trips
Eric
Peninsula
underwater
one
you're
might
Port
Hole
can
says
Phillip
dragon
Cod
Lord
Flinders
Fish
Howe
Rapid
waters
Tasmania's
Australia's
Australian
South
website
Queensland
Rock
Rocks
Tasman
temperate
located
check
world
day
two
Moreton
Cave
continental
novice
National
marine
Epinephelus
adventure
rockcod
creatures
eels
includes
encounter
see
Neptune
Several
southern
whales
cage
water
information
Heads
sizable
also
giant
West
Jetty
Dive
run
around
cave
QLD
find
just
depth
Cape
found
north
green
deep
leafy
supports
qualified
include
World
ocean
way
much
live-aboard
Heritage-listed
Islands
November
home
often
new
surface
moray
NSW
close
turtles
rising
option
runs
list
will
jetty

Example: Text representation

- "Bag of words" works quite well for many tasks
- Problems?
  - Missing context, order, understanding of synonyms and phrases
  - Curse of dimensionality -- "words" space is very high-dimensional (~170,000 words in English)

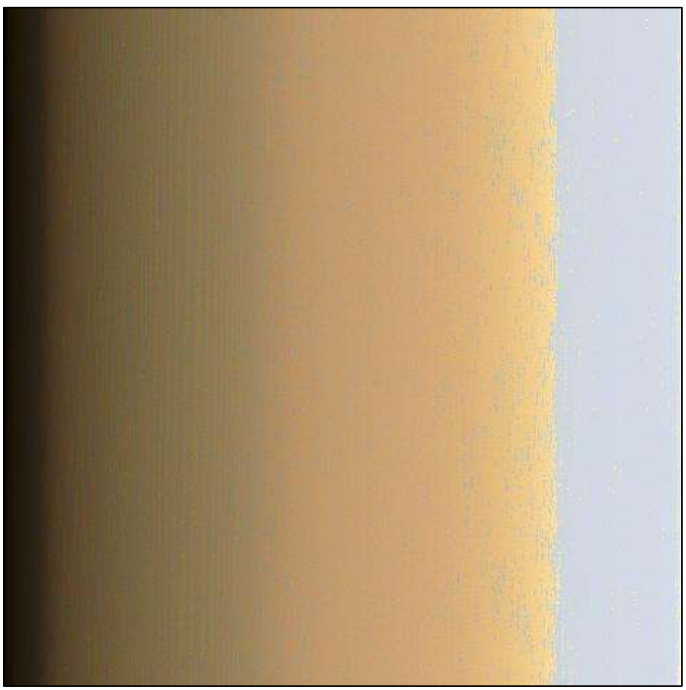
① 缺少顺序，上下文联系

② feature 过多，curse of dimensionality

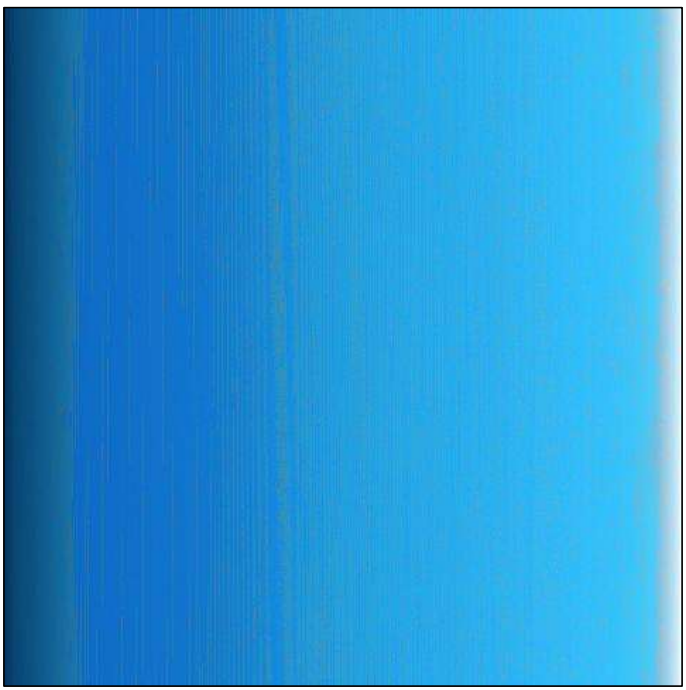
Example: Image representation

- How to represent the main idea of an image?
- Images are made up of pixel values, each pixel has an RGB value
- How much information can you get from “bags of pixels”?

Pixels



A



B

Example: Image representation

- Pixel-level representations can work for constrained tasks
- But more complex visual tasks require more complex features (shapes, objects)
  - How to define these?
- Curse of dimensionality – raw pixel space is very high-dimensional (~700,000 dimensions in the example images)

Representation learning in NNs

- Representation learning is a common application for neural networks
- Networks learn a feature hierarchy – from simple combinations of the input to more complex features (sometimes called **embeddings**)
- Embeddings are:
  - Low-dimensional representations of the input
  - Often useful for a range of tasks (not just the task on which the network was originally trained)

Introduction to neural networks

Convolutional neural network

"AlexNet": Krizhevsky, Sutskever, & Hinton (2012)

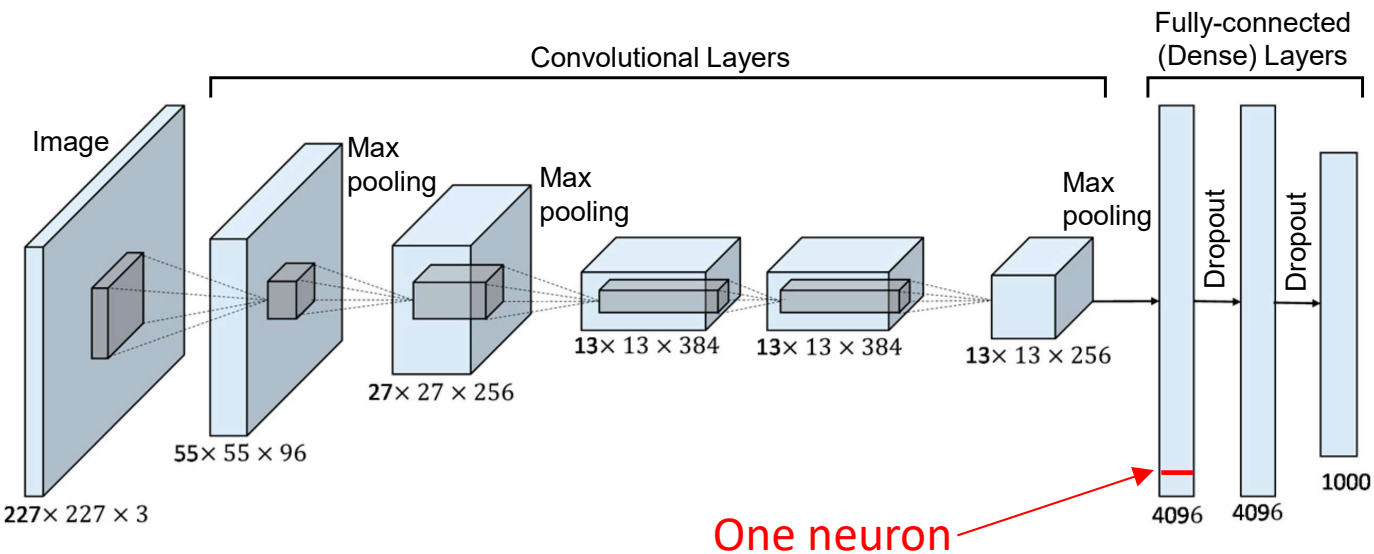


Image: Han, Zhong, Cao, & Zhang (2017)

Biological basis

- Hebbian learning (Hebb, 1949) – model for how neural connections change during learning
- “neurons wire together if they fire together” (Löwel & Singer, 1992)
- Over time, more weight on features associated with a target (like a class label), low weight on features not associated with the target

Biological neuron

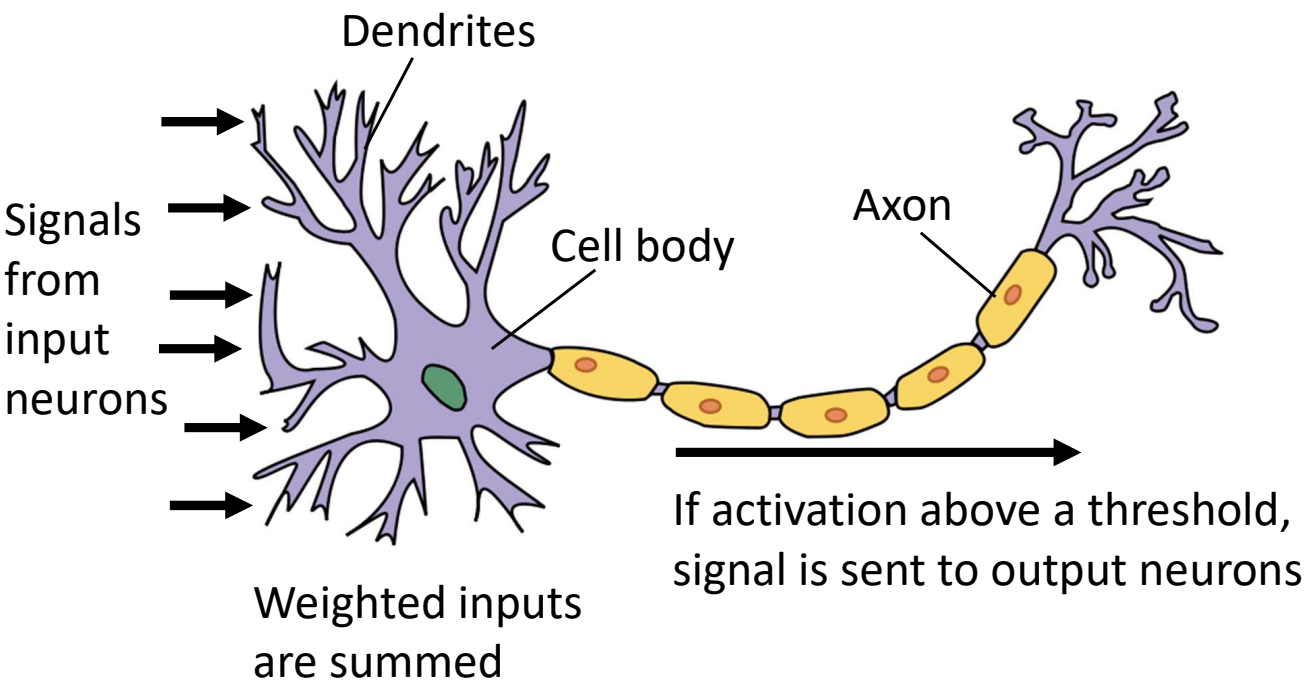
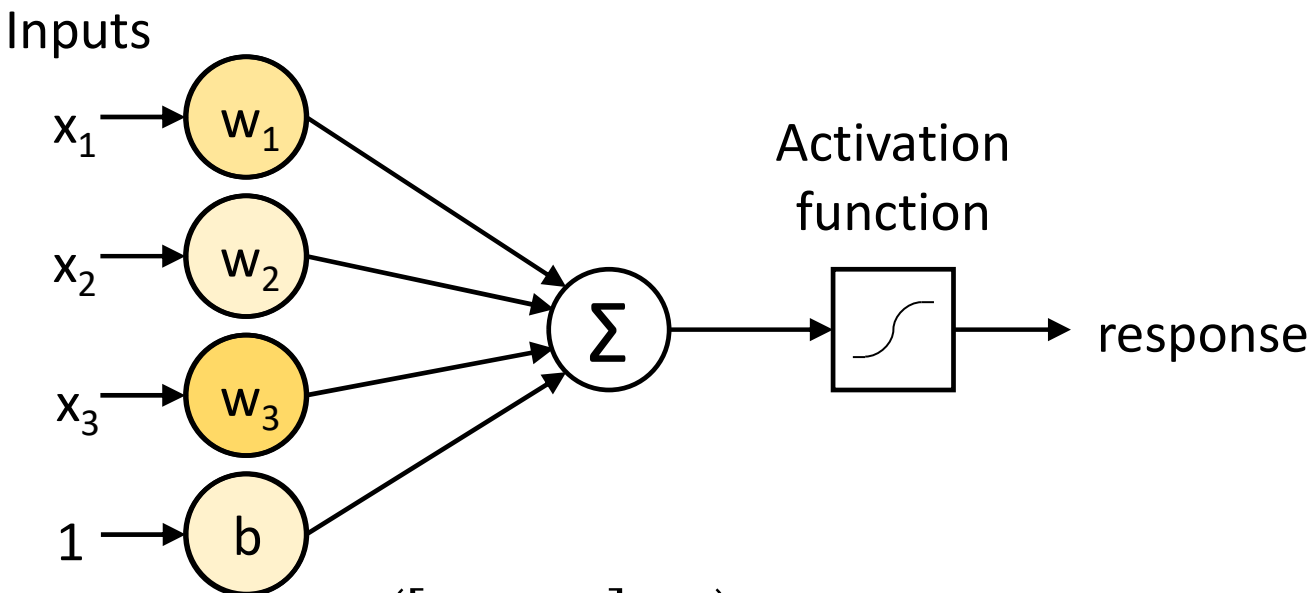


Image: *Anatomy and Physiology*, US National Cancer Institute SEER Program

Artificial neuron



$$y_i = f\left(\left[\sum_j w_j x_{ij}\right] + b\right) = f(\mathbf{w} \cdot \mathbf{x_i} + b)$$

Artificial neuron

- The basic unit of a neural network is the neuron, which is defined as follows:
  - input = a vector $\mathbf{x_i}$ ($\langle x_{i1}, x_{i2}, \ldots x_{in} \rangle \in \mathbb{R}^n$)
  - output = a scalar $y_i \in \mathbb{R}^n$
  - hyper-parameter: an activation function f
  - parameters: a vector of weights $\mathbf{w}$ ($\langle w_1, w_2, \ldots w_n \rangle \in \mathbb{R}^n$) plus a bias term b ($b \equiv w_0$)
- Mathematically:

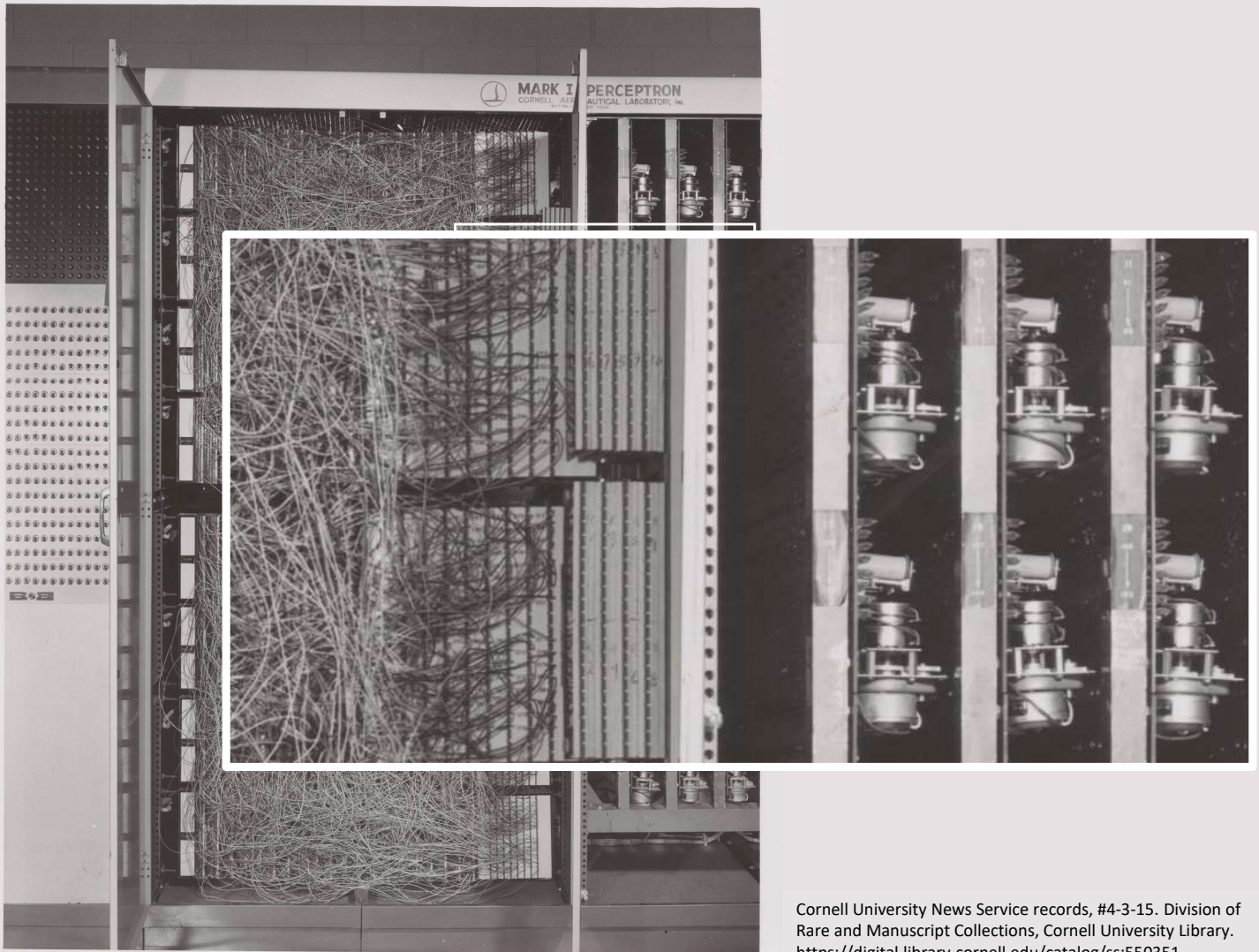
$$y_i = f\left(\left[\sum_j w_j x_{ij}\right] + b\right) = f(\mathbf{w} \cdot \mathbf{x_i} + b)$$

Perceptron

Perceptron

- Perceptron = a neural network with just a single neuron
- A perceptron is a binary linear classifier, which can be written as

$$f(\mathbf{w} \cdot \mathbf{x_i} + b) = \begin{cases} 1 & if\ \mathbf{w} \cdot \mathbf{x_i} + b \geq 0 \\ 0 & otherwise \end{cases}$$



Cornell University News Service records, #4-3-15. Division of Rare and Manuscript Collections, Cornell University Library. https://digital.library.cornell.edu/catalog/ss:550351

Training a perceptron

- Training a perceptron entails finding the weights $\mathbf{w}$ which minimize errors on the training data
- The classic way to train a perceptron is by iterating over examples in a training dataset. Each iteration over the whole dataset is called an **epoch**.
- Perceptron algorithm
  - For training examples, compute y based on current weights
  - Update weights based on the difference between predicted y and true label

Training a perceptron

Initialize weight vector $\mathbf{w}$ to random values

repeat

  for each training instance $(\mathbf{x_i}, y_i)$ **do**

    compute $\hat{y_i} = f(\mathbf{w} \cdot \mathbf{x_i} + b)$

    for each weight w_j **do**

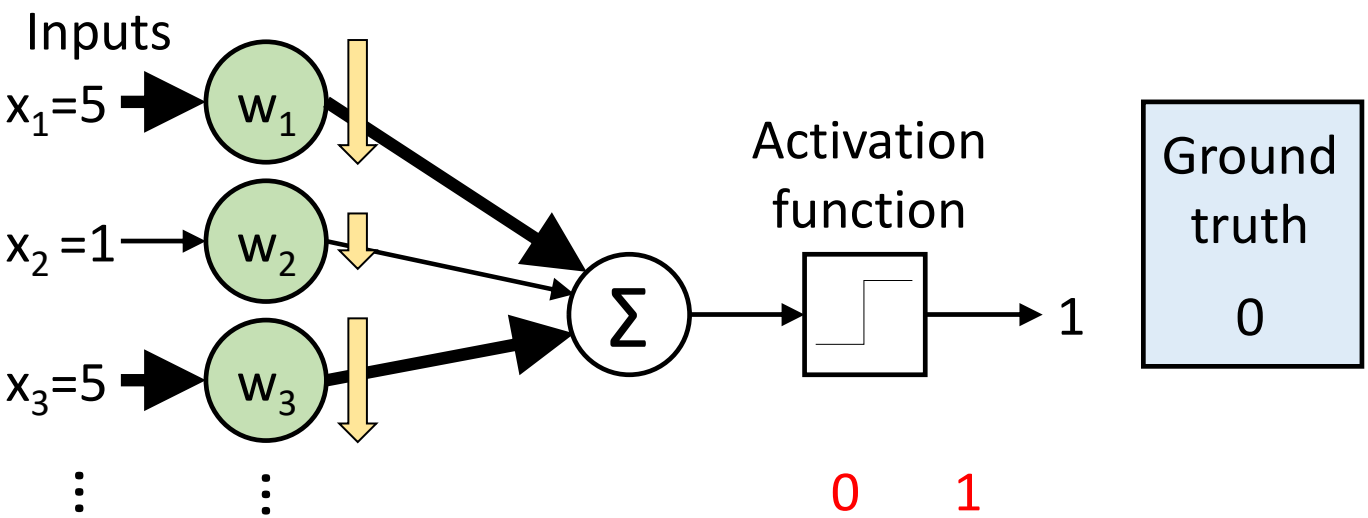
      update $w_j \leftarrow w_j + \lambda(y_i - \hat{y_i})x_{ij}$

    end for

  end for

until stopping condition is met

Weight update

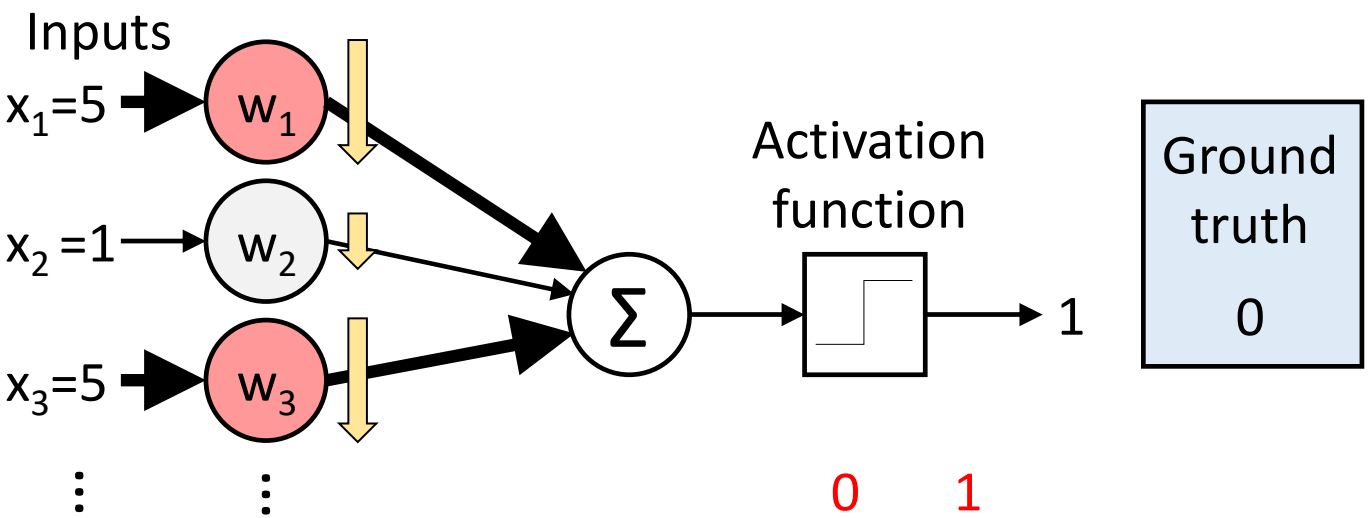


$$w_j = w_j + \lambda(y_i - \hat{y}_i)x_{ij}$$

0 1

Learning rate

Weight update



$$w_j = w_j + \lambda(y_i - \hat{y}_i)x_{ij}$$

0 1

Learning rate

Numeric example

Training instances

$\langle x_{i1}, x_{i2} \rangle$	y_i
1,1	1
1,2	1
0,0	0
-1,0	0

Learning rate

$\lambda = 1$

$\langle b, w_1, w_2 \rangle$ = 0, 0, 0

Epoch 1:

$\langle x_{i1}, x_{i2} \rangle$	$b + w_1 \cdot x_{i1} + w_2 \cdot x_{i2}$	$\hat{y}_i$	y_i
1,1	0 + 0 + 0 = 0	1	1
1,2	0	1	1
0,0		1	0

$b \leftarrow 0 + (-1) \cdot 1 \cdot 1 = -1$

$w_1 \leftarrow 0 + (-1) \cdot 0 \cdot 0 = 0$

$w_2 \leftarrow 0 + (-1) \cdot 0 \cdot 0 = 0$

$\langle -1, 0, 0 \rangle$

$$f(\mathbf{w} \cdot \mathbf{x_i} + b) = \begin{cases} 1 & if\ \mathbf{w} \cdot \mathbf{x_i} + b \geq 0 \\ 0 & otherwise \end{cases}$$

Numeric example

Training instances

$\langle x_{i1}, x_{i2} \rangle$	y_i
1,1	1
1,2	1
0,0	0
-1,0	0

Learning rate

λ = 1

$\langle b, w_1, w_2 \rangle$ = -1, 0, 0

Epoch 2:

$\langle x_{i1}, x_{i2} \rangle$	$b + w_1 \cdot x_{i1} + w_2 \cdot x_{i2}$	$\hat{y}_i$	y_i
1,1	-1 + 0 + 0 = -1	0	1

$b \leftarrow 1 + 1 \cdot 1 \cdot 1 = 0$

$w_1 \leftarrow 0 + 1 \cdot 1 \cdot 1 = 1$

$w_2 \leftarrow 0 + 1 \cdot 1 \cdot 1 = 1$

$\langle 0, 1, 1 \rangle$

1,2 $\quad 0 + 1 + 2 = 3$ $\quad$ 1 $\quad$ 1 $\quad$ ✓

0,0 $\quad 0 + 0 + 0$ $\quad$ 1 $\quad$ 0 $\quad$ ✗

$b \leftarrow -1 + (-1) \cdot 1 \cdot 0 = -1$

$w_1 \leftarrow 1 + (-1) \cdot 1 \cdot 0 = 1$

$w_2 \leftarrow 1 + (-1) \cdot 1 \cdot 0 = 1$

$\langle -1, 1, 1 \rangle$

(-1,0) $\quad -1 + 0 = -1$ $\quad$ 0 $\quad$ 0 $\quad$ ✓

$$f(\mathbf{w} \cdot \mathbf{x_i} + b) = \begin{cases} 1 & if\ \mathbf{w} \cdot \mathbf{x_i} + b \geq 0 \\ 0 & otherwise \end{cases}$$

Numeric example

Training instances

$\langle x_{i1}, x_{i2} \rangle$	y_i
1,1	1
1,2	1
0,0	0
-1,0	0

Learning rate

λ = 1

$\langle b, w_1, w_2 \rangle$ = -1, 1, 1

Epoch 3:

$\langle x_{i1}, x_{i2} \rangle$	$b + w_1 \cdot x_{i1} + w_2 \cdot x_{i2}$	$\hat{y}_i$	y_i	
1,1	-1 + 1 + 1 = 1	1	1	✓
1,2	-1+1+2 = 2	1	1	✓
0,0	-1+0+0 = -1	0	0	✓
-1,0	-1+(-1)+0 = -2	0	0	✓

Convergence – there were no updates in this epoch, so end training

$$f(\mathbf{w} \cdot \mathbf{x_i} + b) = \begin{cases} 1 & if\ \mathbf{w} \cdot \mathbf{x_i} + b \geq 0 \\ 0 & otherwise \end{cases}$$

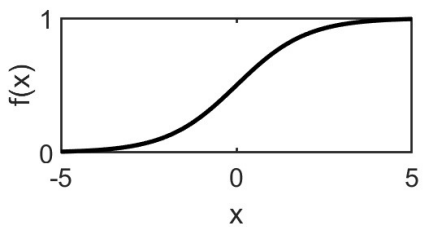
Activation function

- What does it do?
- In a perceptron
  - Maps the linear response to the range we want (class labels 0 or 1)
- In a multilayer perceptron / neural network:
  - Adds a non-linearity after each linear operation
  - More about this later!

Activation function

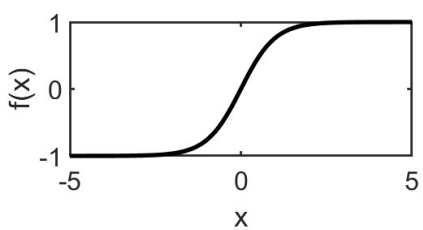
- Common choices:
- (logistic) sigmoid (σ):

$$f(x) = \frac{1}{1 + e^{-x}}$$



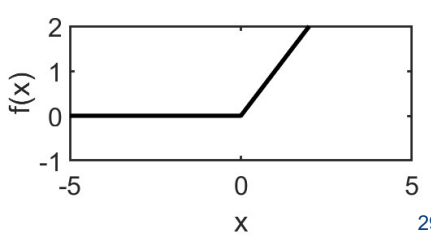
- hyperbolic tan (tanh):

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



- rectified linear unit (ReLU):

$$f(x) = \max(0, x)$$



Perceptron properties

- The perceptron algorithm is guaranteed to converge for linearly-separable data, but the convergence point (class boundary) will depend on:
  - The initial values of the weights and bias
  - The learning rate
- Not guaranteed to maximise the margin between classes
- Not guaranteed to converge over non-linearly-separable data

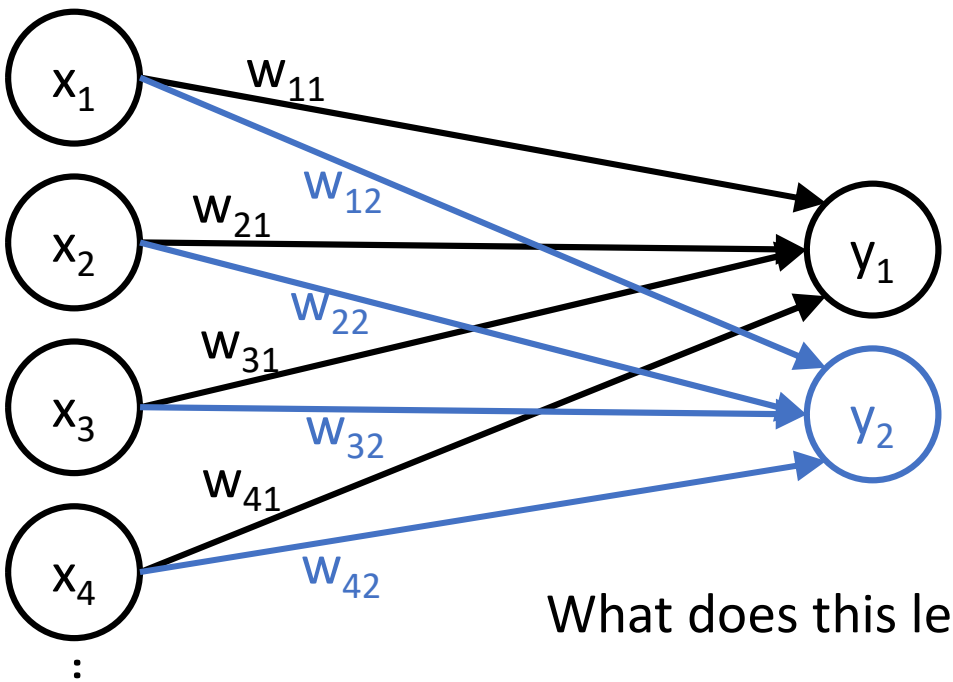
Perceptron properties

- A perceptron with a sigmoid activation function (plus a binary step to convert the output to 0 or 1) is equivalent to logistic regression:

$$y_i = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x_i} + b)}}$$

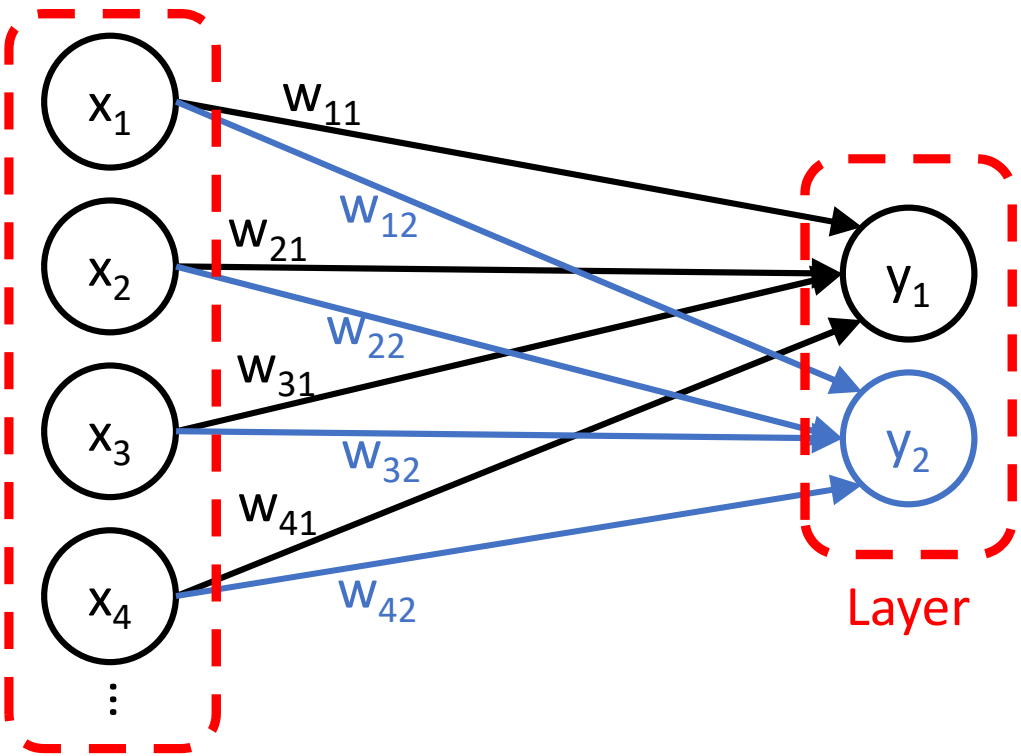
Multilayer perceptron

Extension 1: Multiple outputs

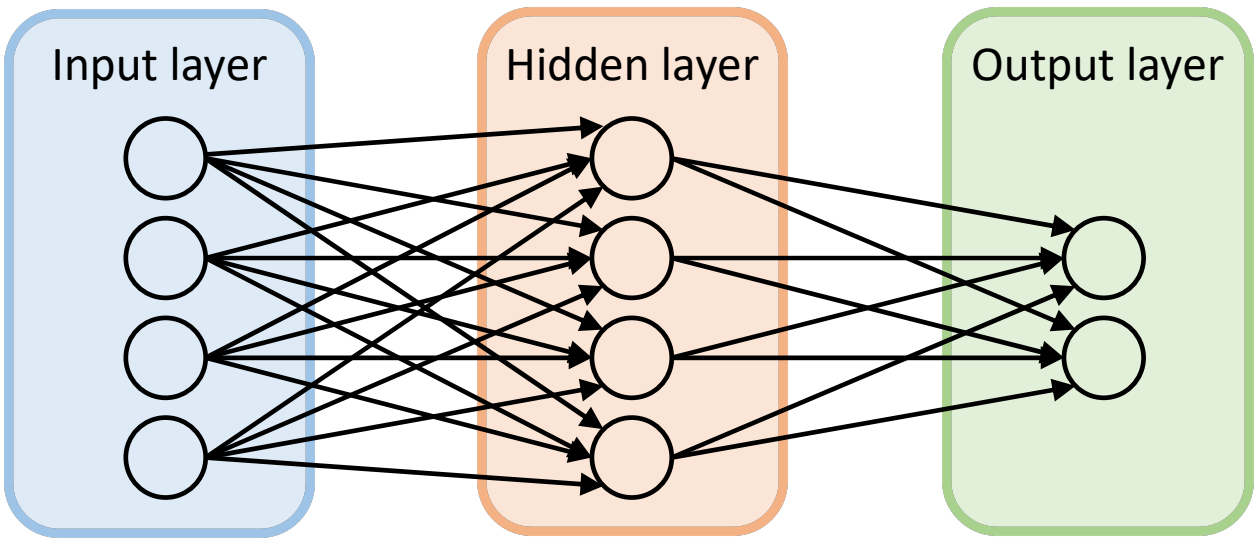


What does this learn?

Extension 1: Multiple outputs



Extension 2: Multiple layers



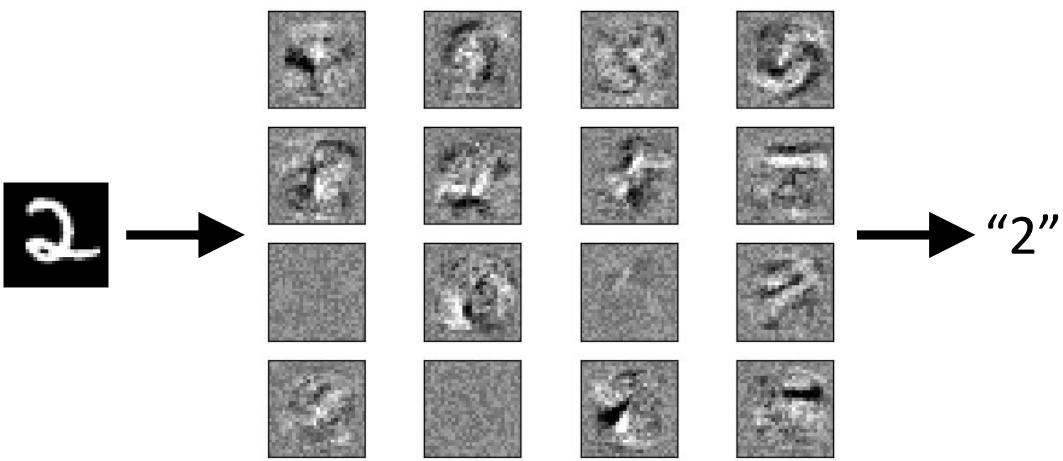
What does this learn?

Multilayer perceptron

- Layer = a group of neurons working in parallel on the same input
- Multilayer perception (MLP):
  - An **input layer**
  - At least one **hidden layer**. Each hidden layer receives the previous layer's output as its input
  - An **output layer** which receives input from a hidden layer and outputs the class label

Hidden layer

- What does it learn?
  - Representation learning – hidden layers learn some features that are useful for the output layer



https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html

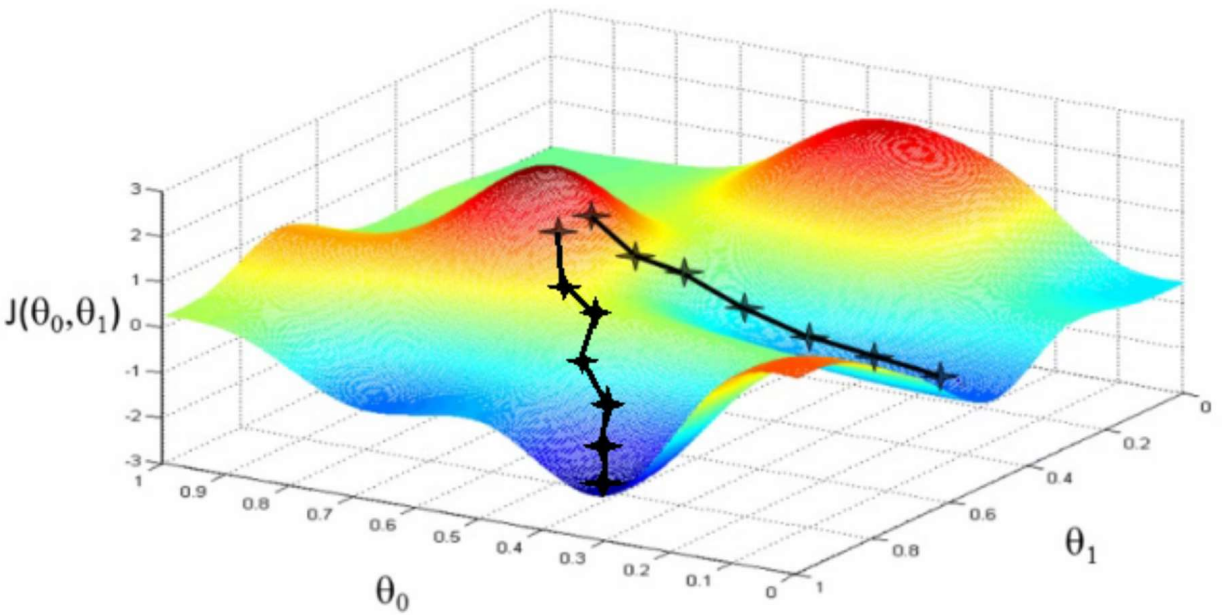
Hidden layer

- How many neurons?
  - Given that we don't know exactly what features this layer should learn, how to pick the right number of neurons?
  - In theory, depends on factors like complexity of decision boundary
  - In practice, often just pick an arbitrary value in between the input and output size

Hidden layer

- How does it learn?
  - There's not just one "correct" output for the neurons in a hidden layer, like there is for the output layer
  - How can the hidden layer(s) be trained?
- Solution: backpropagation
  - **Stochastic** gradient descent (randomly choose sample)
  - Find parameters (weights+biases) to minimize loss on the training dataset

Gradient descent



Backpropagation

- Form of gradient descent – iterative process for finding the weight parameters that minimize error
- Just like single perceptron training, requires a learning rate (hyperparameter)
- Idea: look at the derivative of error and update weights in the direction that would reduce error
  - Compute errors at the output layer w.r.t. each weight using partial differentiation
  - Propagate errors back to each of the earlier layers
  - Chain rule used to efficiently compute derivatives

Output layer

- Structure depends on task
- Common options:
  - Binary classification – 1 output neuron with step activation function
  - N-way classification – N output neurons and softmax activation function
  - Regression – 1 output neuron with identity activation function

Neural network/MLP properties

- Universal approximation theorem: a feed-forward neural network with a single hidden layer (with finite neurons) is able to approximate any continuous function on $\mathbb{R}^n$
- Means the network can learn any (linear or non-linear) basis function dynamically, unlike many other methods (e.g., SVM where kernel is a hyperparameter)
- Note that this requires non-linearities

Non-linearities

- With a non-linear activation function:
  $$f(\mathbf{w_{L2}} \cdot f(\mathbf{w_{L1}} \cdot f(\mathbf{w_{L0}} \cdot \mathbf{x_i} + \mathbf{b_{L0}}) + \mathbf{b_{L1}}) + \mathbf{b_{L2}})$$

- Without non-linearity:
  $$\mathbf{w_{L2}} \cdot (\mathbf{w_{L1}} \cdot (\mathbf{w_{L0}} \cdot \mathbf{x_i} + \mathbf{b_{L0}}) + \mathbf{b_{L1}}) + \mathbf{b_{L2}} = \mathbf{w}_\alpha \cdot \mathbf{x_i} + \mathbf{b_\alpha}$$
  - Multiple layers don't accomplish anything – the same computations could have been done in one layer

Neural network/ MLP properties

- True or false?

- Since NNs are universal approximators, they are guaranteed to generalise better than any other machine learning method. not guaranteed to be better on generalised

- Since NNs learn their own representations in hidden layers, they don't require any feature engineering.

Neural network/ MLP properties

- MLP advantages
  - Can be adapted to many types of problems (classification, regression)
  - Universal approximator – can model arbitrary basis functions
  - Representation learning in hidden layers
- MLP disadvantages
  - Very high number of parameters – slow to train, prone to overfitting, high memory requirements
  - Stochastic gradient descent – not guaranteed to converge to the same solution every time