Multi-layer Perceptrons (MLPs)

An introduction

Elizabeth Williamson

LONDON SCHOOL of HYGIENE &TROPICAL MEDICINE

Learning outcomes

In this session, we will:

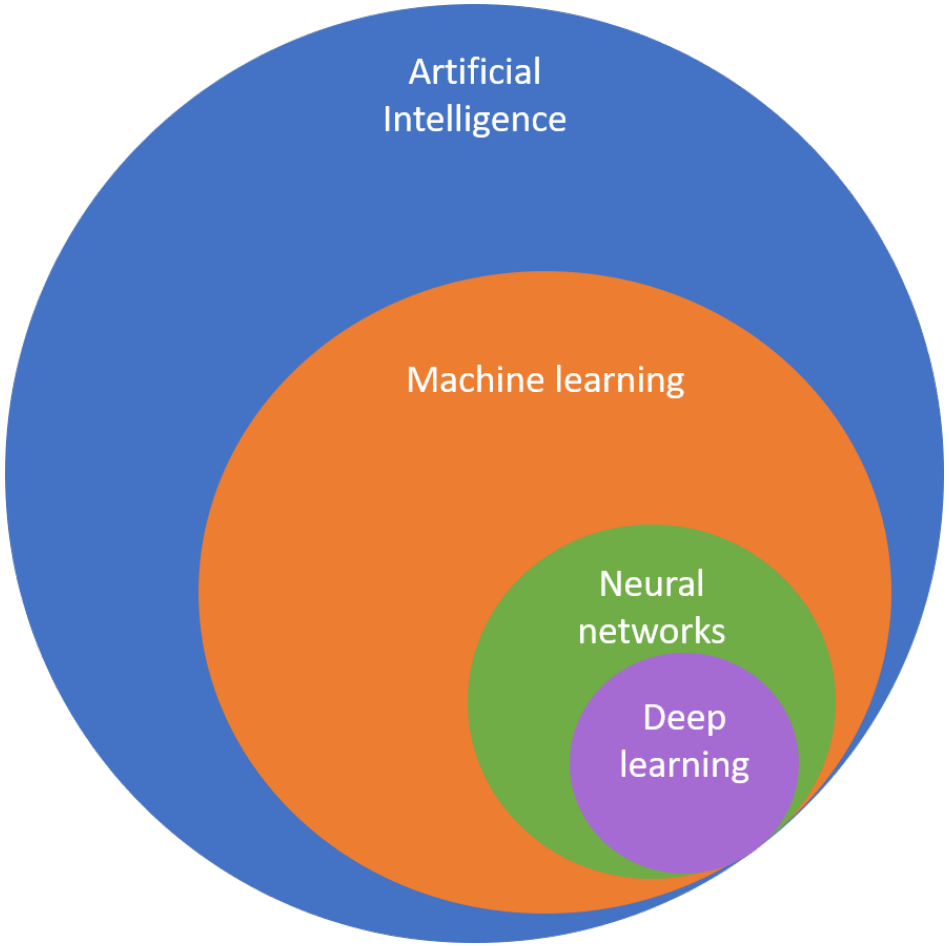
- Extend simple logistic/multinomial regression to a Multi-Layer Perceptron
- Think about different activation functions
- Explore a simple example (MNIST)
- Think about connections to other types of neural network

Table of Contents

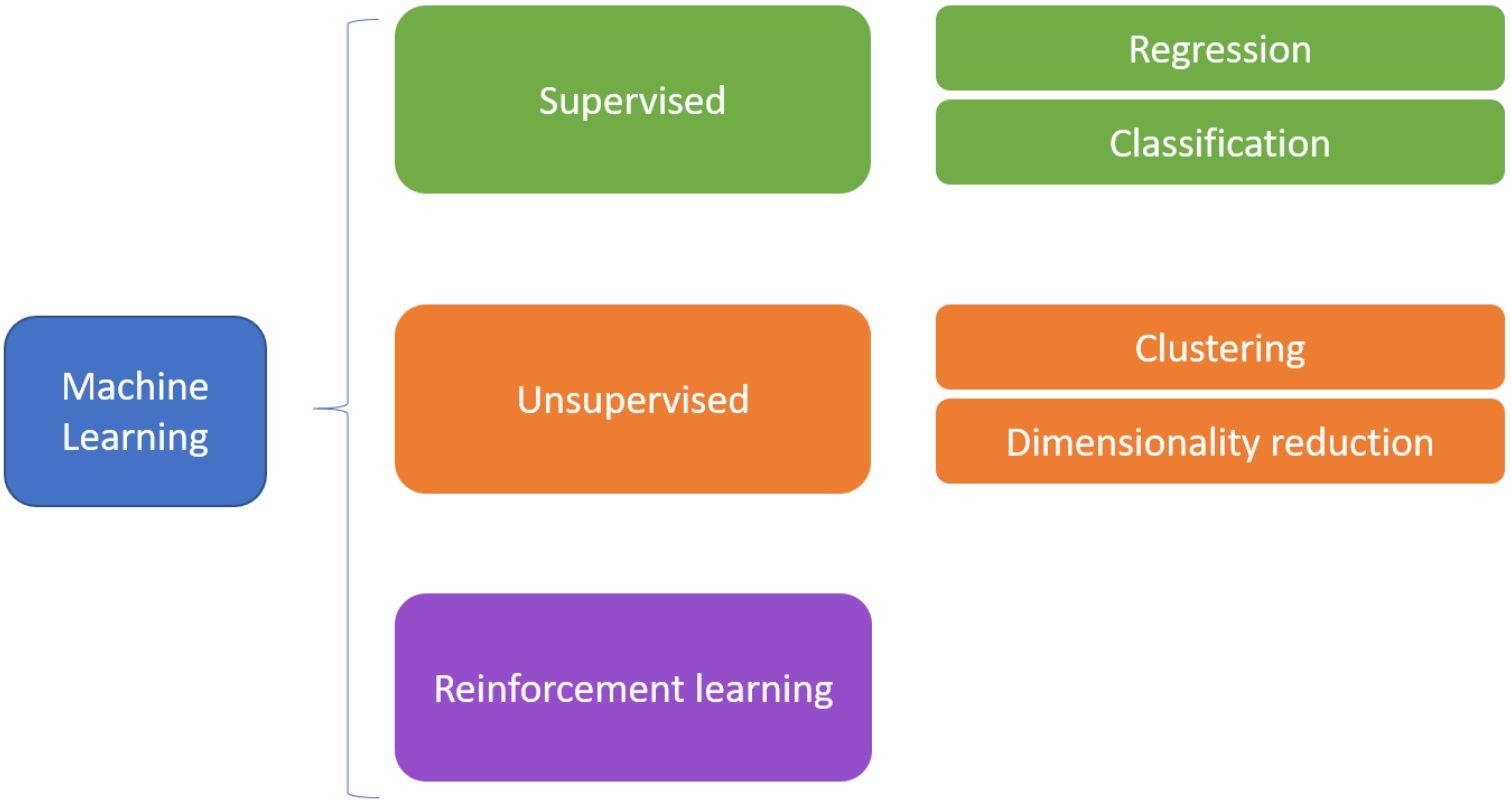
Deep learning

- Deep Learning is a branch of Machine Learning using artificial neural networks (ANNs)
- ANNs attempt to mimic the way that biological neurons signal to one another in the brain
- Statistically, ANNs are mathematical functions that map a set of inputs to one or more outputs.

Deep learning - connections to ML and AI



Artificial neural networks

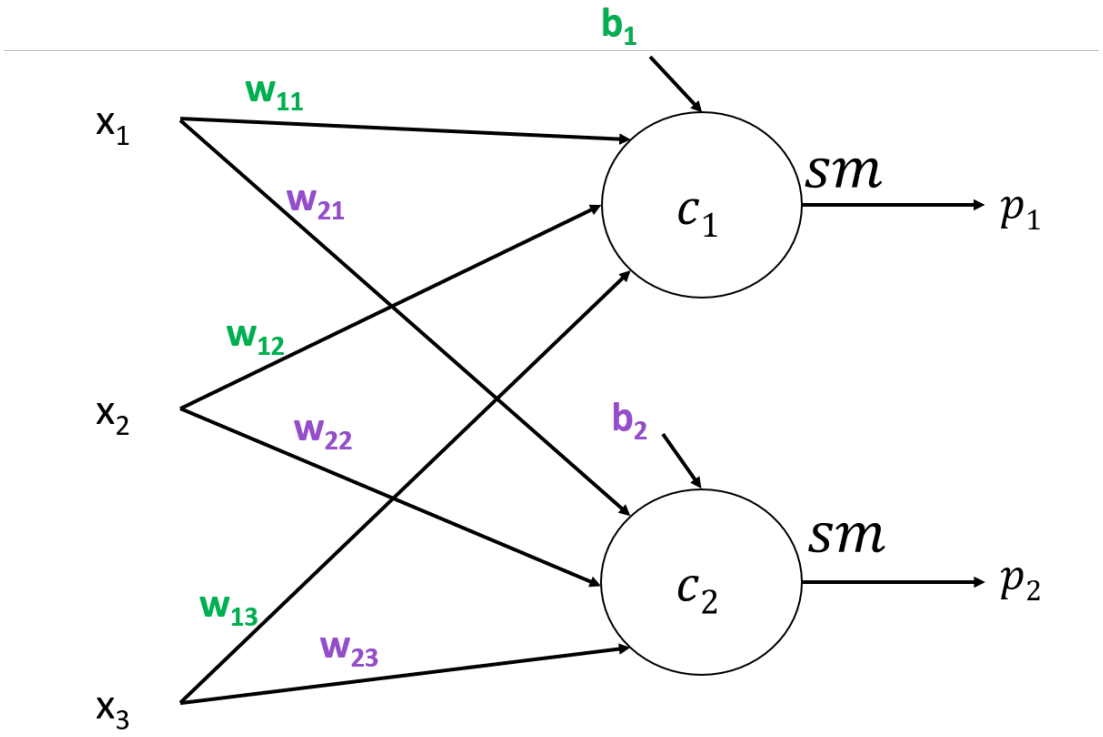


Neural networks can be used within all of these branches of machine learning.

Table of Contents

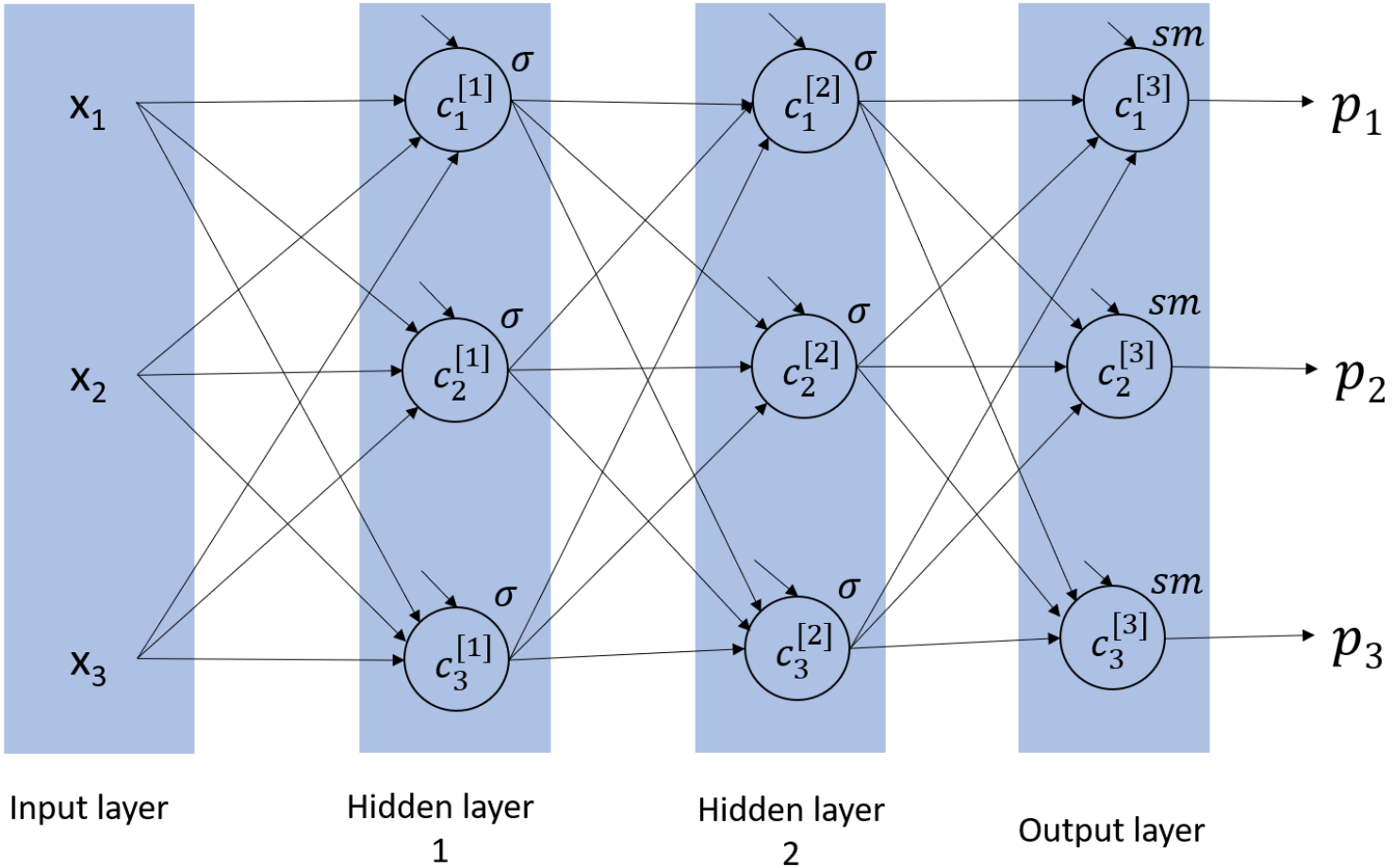
Extending logistic/multinomial regression

We have seen how to combine perceptrons to extend logistic regression to more than two outcome classes



We will now add *layers* to this network.

Multi-layer perceptrons



Multi-layer perceptrons

- As well as the input layer and the output layer (the confidence values), we now have one or more *hidden* layers
- A ‘deep’ network has at least 3 layers (the input layer is not counted)
- The layers are called: dense layers, affine layers, linear layers, fully connected layers (fcl).

Cross-entropy loss

The likelihood of the data is:

$$L(\theta) = \prod_{i=1}^{N} \mathbb{P}(y = y_i | \mathbf{x} = \mathbf{x}_i)$$

The log-likelihood is:

$$l(\theta) = \sum_{i=1}^{N} log\left(\mathbb{P}(y = y_i | \mathbf{x} = \mathbf{x}_i)\right)$$

If we let p_{ic} be the probability that observation y_i is in class c, given $\mathbf{x}_i$, then we can write the negative log-likelihood as the *cross-entropy loss function*. ($I[.]$ is the indicator function.)

$$J(\theta) = -l(\theta) = -\sum_{i=1}^{N}\sum_{c=1}^{K} I[y_i \text{ in class } c] log(p_{ic})$$

Softmax function

For K categories

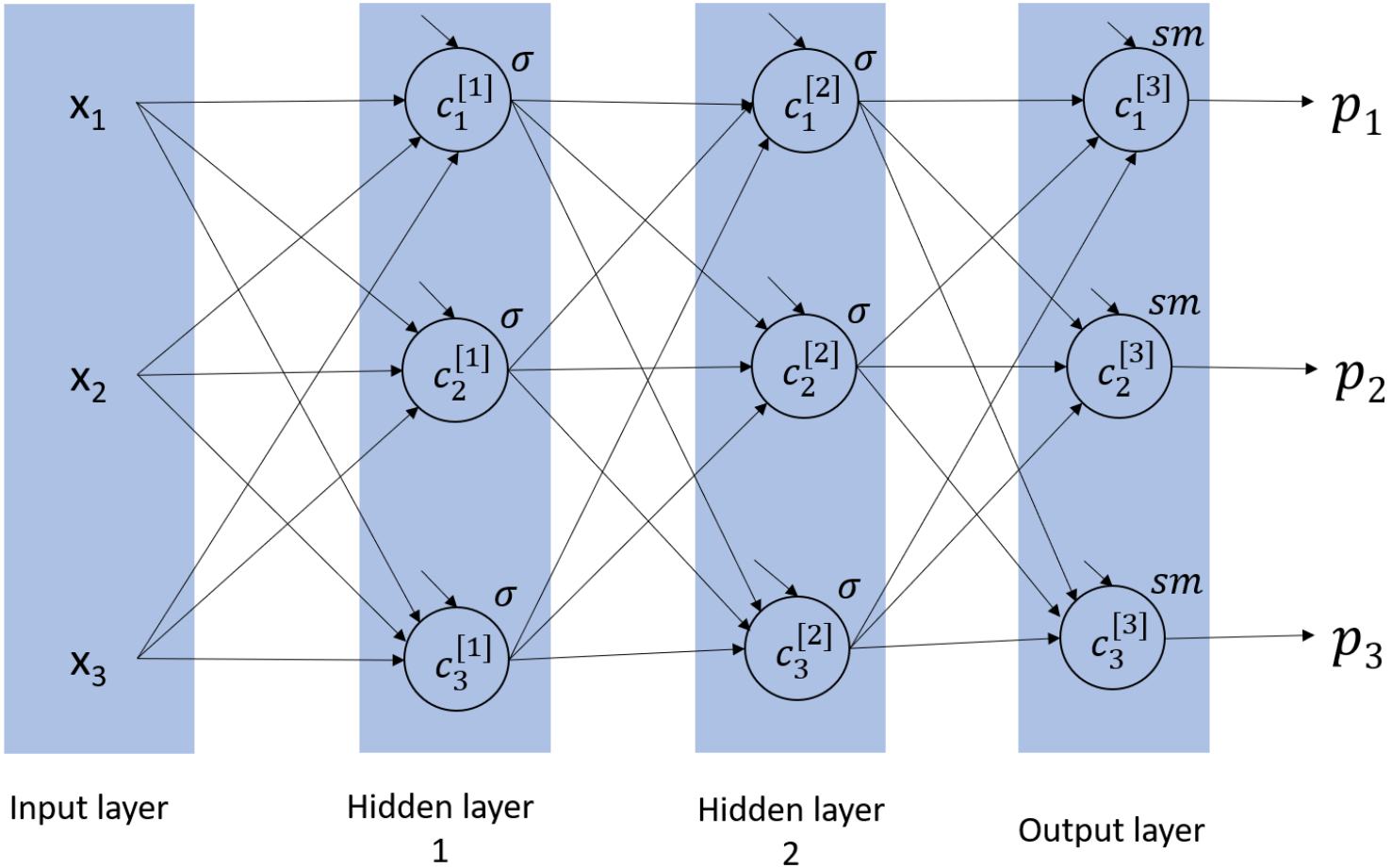
- K linear combinations are outputted from the final output layer, $\{c_j^{[\text{out}]}; j = 1, ..., K\}$, say
- These are converted to a probability distribution using the *soft-max* function (rather than the sigmoid function that we use for other layers)

For class c, $c = 1, ..., K$, the softmax function gives:

$$sm(c_c^{[\text{out}]}) = \frac{e^{c_c^{[\text{out}]}}}{\sum_{d=1}^{K} e^{d_d^{[\text{out}]}}}$$

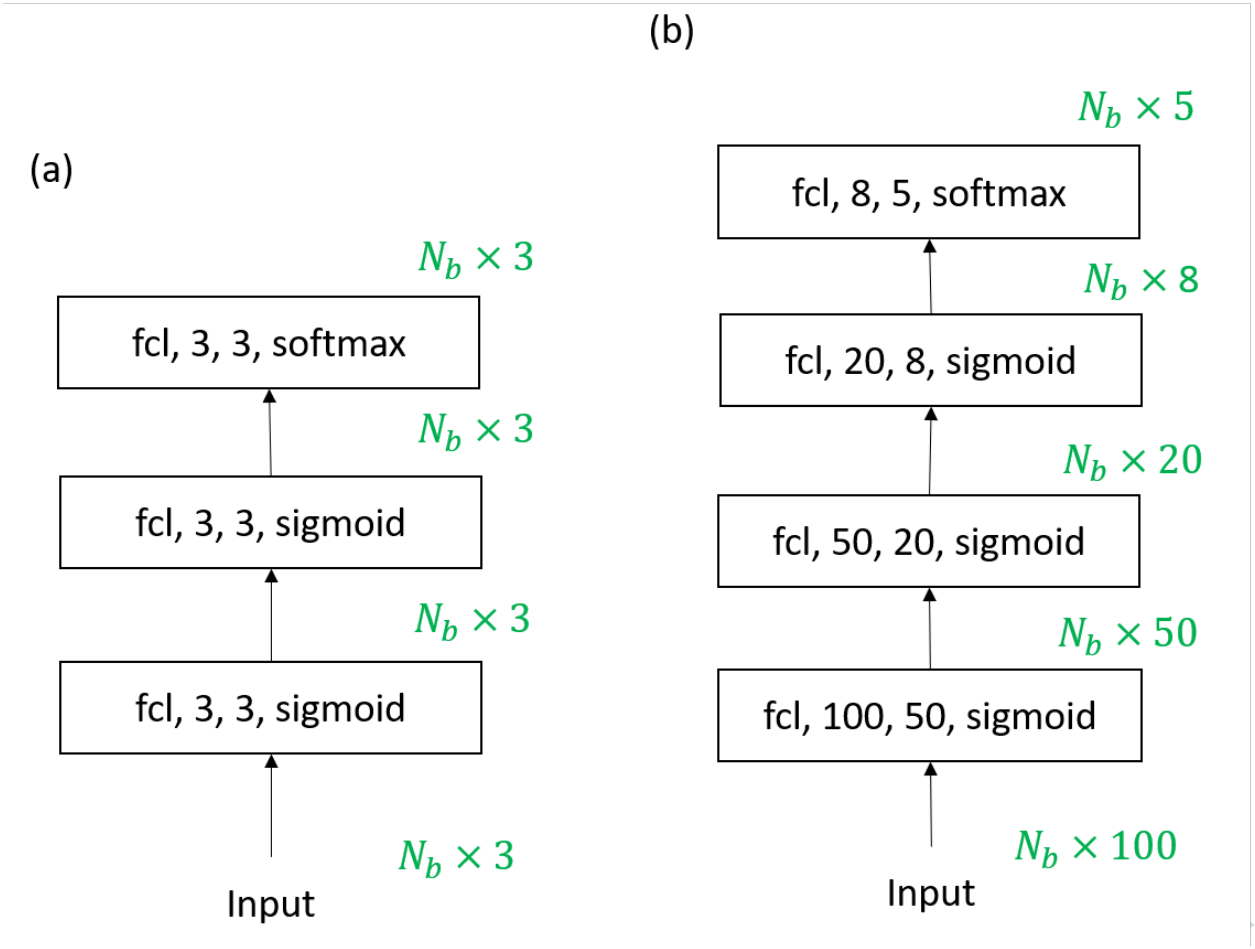
- An instance (sampling unit) is classified as the class with the highest probability

Multi-layer perceptrons



Neural network architecture

(a) Simple setting, 3 covariates, 2 hidden layers each with 3 nodes (neurons); (b) 100 covariates and 3 hidden layers. N_b = number instances in a batch.



Number of parameters

- There are a number of parameters (to be estimated / involved in the optimization process)
- More than in a typical logistic regression model
- In the previous slide:
  - Example (a) had 36 parameters.
  - Example (b) had 6283.

- There are also hyper-parameters:
  - Learning rate
  - Batch size
  - Number of layers
  - Number of neurons/perceptrons per layer

Table of Contents

Different activation functions

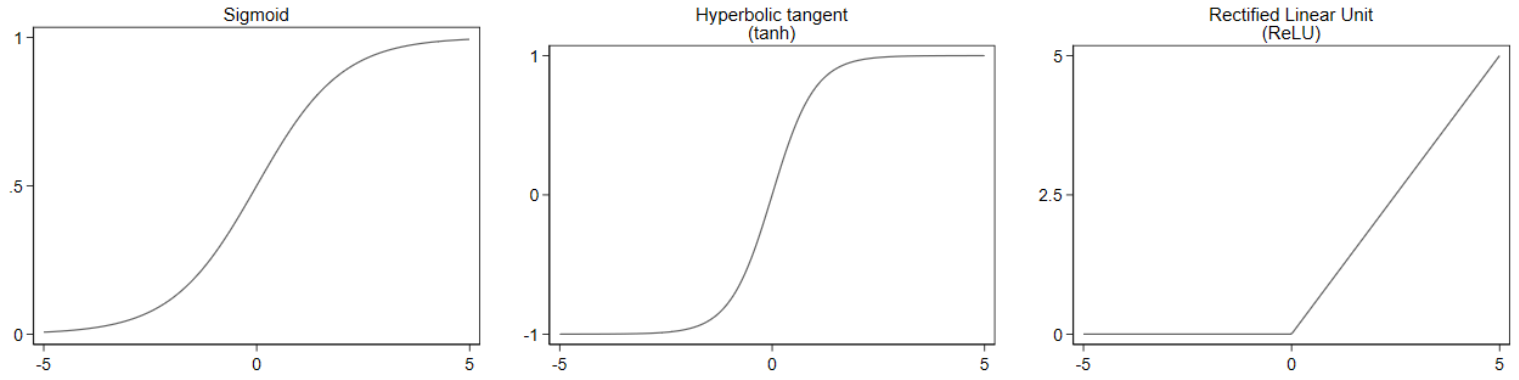
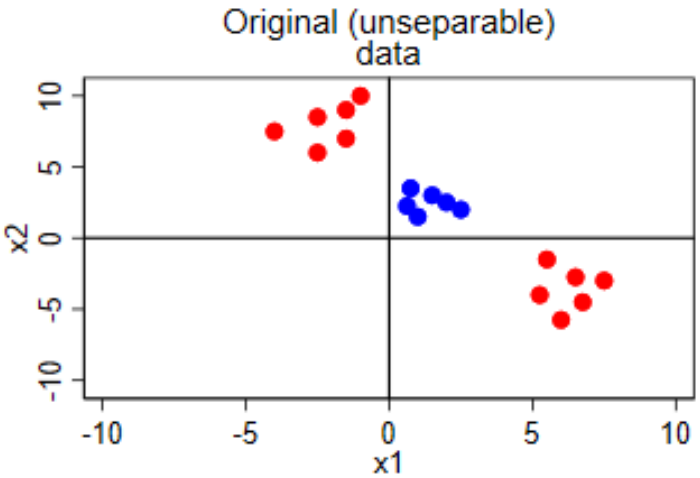
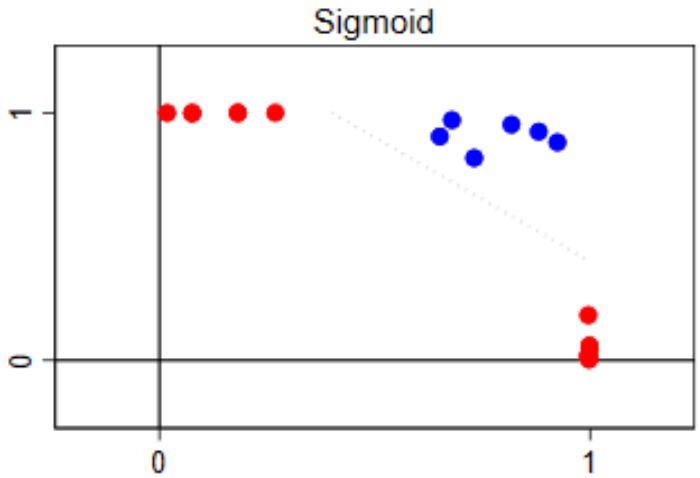


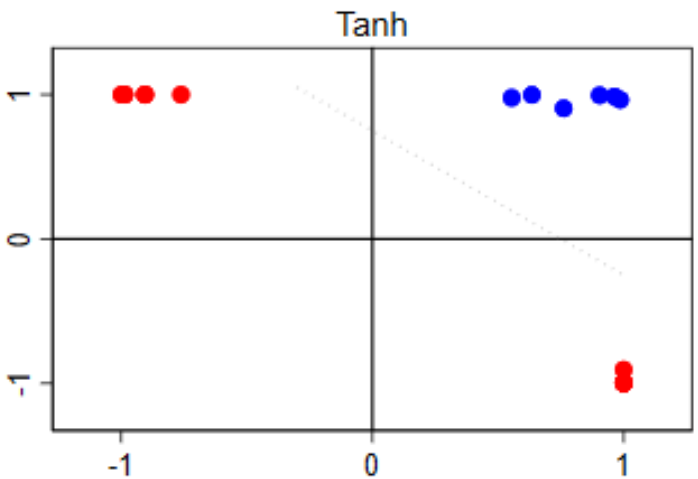
Figure: Activation functions

Table of Contents

Different activation functions







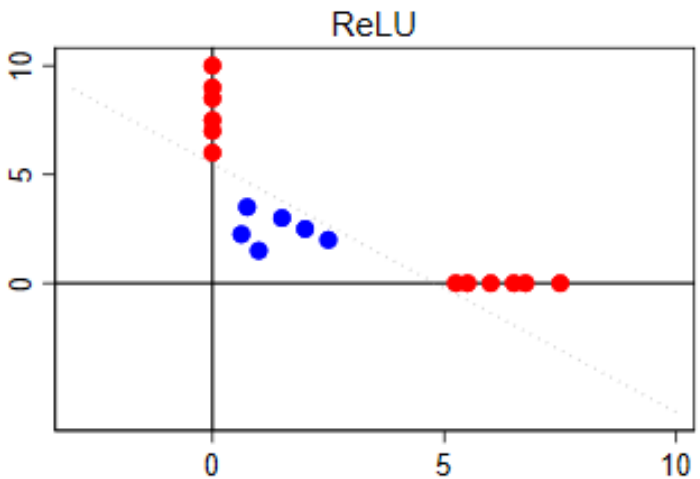


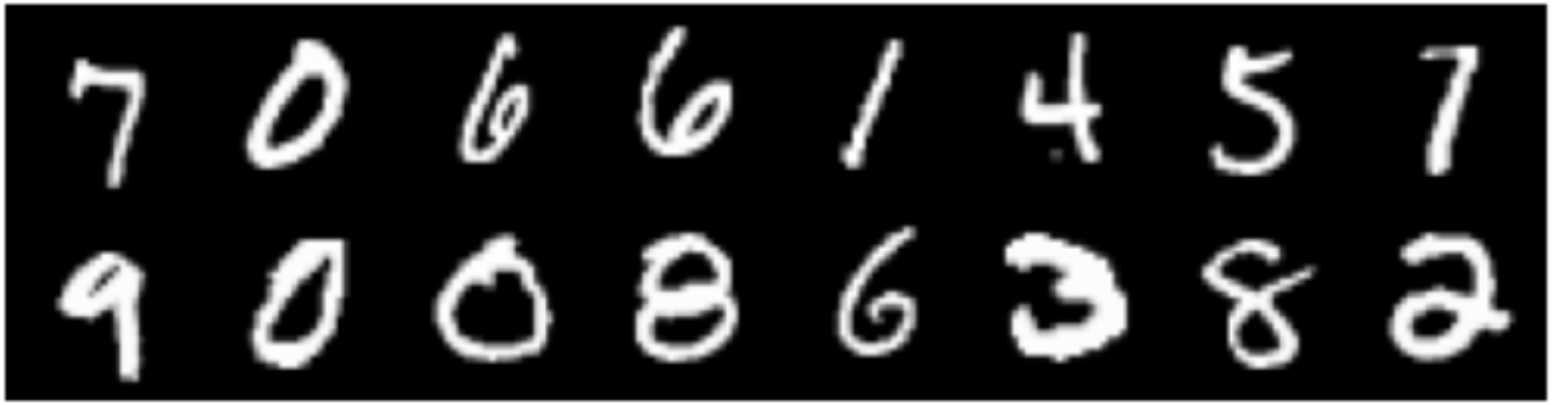
Table of Contents

The MNIST database

The MNIST database (Modified National Institute of Standards and Technology database):

- is a large database of handwritten digits
- commonly used for training various image processing systems.
- 60,000 training images of handwritten digits (i.e. each contains a single number: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9).
- 10,000 test images

The MNIST database



- each picture is a 28 x 28 greyscale image (with elements in [0,1]).
- each picture is converted to a 1D vector of length 28x28=784.
- This is called *flattening*

Architectures explored - Single layer network

In a simple single layer network we have:

- 784 (28 x 28) input nodes and 10 output neurons
- Parameters:
  - 784 x 10 = 7840 weight parameters
  - 10 bias parameters

Architectures explored - Two layer network

An example of a two-layer network:

- 784 (28 x 28) input nodes
- 300 nodes in a hidden layer
- 10 output neurons
- Parameters:
  - 784 x 300 = 235,200 weight parameters in 1st layer
  - 300 bias parameters in 1st layer
  - 300 x 10 = 3000 weight parameters in 2nd layer
  - 10 bias parameters in 2nd layer
  - 238,510 parameters in total

Optimisation

- Cross-entropy loss used
- Training data batches of size 16
- Stochastic gradient descent
- Learning rate 0.001
- Momentum of 0.9

Performance - One layer classifier

True	Predicted									
	0	1	2	3	4	5	6	7	8	9
0	958	0	2	3	0	7	5	4	1	0
1	0	1110	4	2	0	2	4	2	11	0
2	5	7	938	11	9	3	13	9	34	3
3	4	1	19	914	0	25	4	11	26	6
4	1	2	7	3	915	0	8	4	10	32
5	10	2	3	34	11	778	15	7	29	3
6	9	3	9	1	8	14	911	2	1	0
7	1	7	23	4	7	1	0	956	2	27
8	7	8	8	17	8	27	11	9	867	12
9	10	8	1	8	24	7	0	23	8	920

Table: Confusion matrix (10,000 test images)

Performance - One layer classifier

Overall accuracy: 92.3%

Class	Precision (%) (PPV)	Recall (%) (Sensitivity)
0	95.0	97.9
1	96.6	97.8
2	93.3	89.1
3	91.3	90.8
4	92.5	93.4
5	89.8	87.2
6	93.3	95.1
7	91.8	92.6
8	87.3	88.6
9	91.3	90.0

Table: Performance on 10,000 test images

Performance - two layer classifier

True	Predicted									
	0	1	2	3	4	5	6	7	8	9
0	967	0	2	1	1	3	2	2	1	1
1	0	1126	3	0	0	1	2	1	2	0
2	5	1	1010	1	2	0	3	4	5	1
3	0	0	3	990	0	3	0	5	4	5
4	1	0	2	1	958	1	4	1	1	13
5	4	1	0	7	2	866	5	1	4	2
6	6	3	1	0	4	6	935	0	3	0
7	0	7	11	3	2	0	0	995	1	9
8	3	0	3	4	3	4	3	3	948	3
9	3	3	0	7	9	3	1	6	0	977

Table: Confusion matrix (10,000 test images)

Performance - Two layer classifier

Overall accuracy: 97.7%

Class	Precision (%) (PPV)	Recall (%) (Sensitivity)
0	97.8	98.7
1	98.7	99.2
2	97.6	97.9
3	97.6	98.0
4	97.7	97.6
5	97.6	97.1
6	97.9	97.6
7	97.7	96.8
8	97.8	97.3
9	96.6	96.8

Table: Performance on 10,000 test images

Comparison

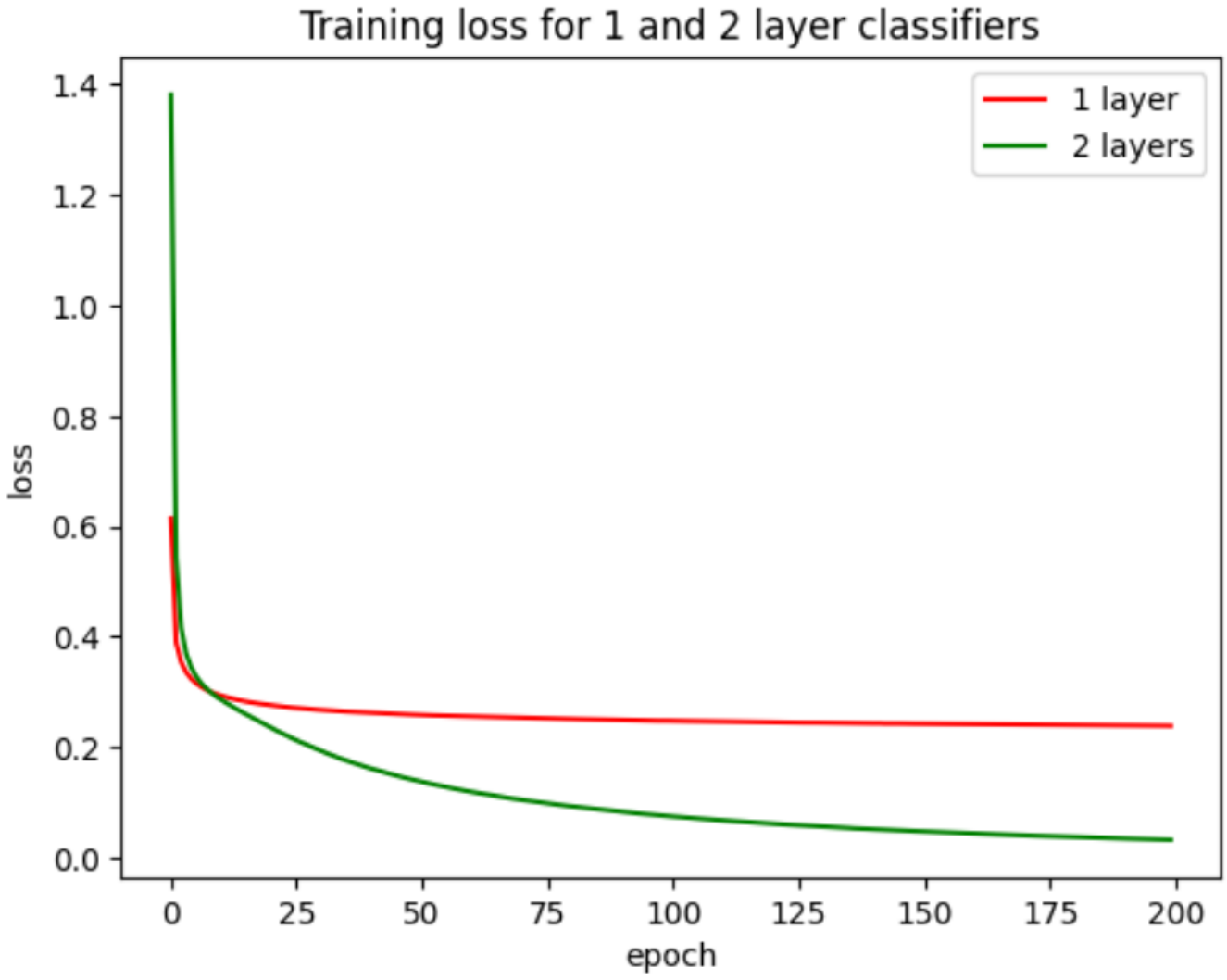

Training loss for 1 and 2 layer classifiers
1 layer
2 layers
loss
epoch
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
25
50
75
100
125
150
175
200

Table of Contents

Some types of neural network

- Convolutional neural network (CNN)
  - Typically used when the input is images
  - Convolution describes the mathematical operation used to extract features from images
- Recurrent neural network (RNN)
  - Typically used for text input
- Transformers
  - Often performs better than RNNs
- Generative adversarial networks (GANs)
  - Create new data instances that resemble the training data.
  - Composed of: a generator, which learns to generate fake data, and a discriminator, which learns to distinguish real from fake data.

Feature (representation) learning

- Machine learning techniques start with a set of variables or *features*. These are used for, for example, classification
- But what if instead of variables your input is a picture? You need to decide what aspects or *features* of the picture are relevant for the task at hand
- In earlier work, feature engineering was done manually. This has proven hard to do well with pictures, audio etc.

Deep learning:

- provides a way of automatically discovering the feature patterns in the data; it learns the relevant features
- allows the model to both learn the features and use them to perform a task (e.g. classification)

Feature (representation) learning

Table of Contents

Final thoughts

- Typically neural networks are much more complex than traditional regression models, and require huge sample sizes and a long compute time
- MLPs can be seen as an extension to logistic regression
- CNNs build on these ideas by adding in ‘convolution’ — a mathematical operation to extract spacial features from images
- RNNs/Transformers build on these ideas by building vector representations of words using an idea called ‘embedding’