Towards Artificial Neural Networks (ANNs)

An introduction

Elizabeth Williamson

LONDON SCHOOL of HYGIENE &TROPICAL MEDICINE

Learning outcomes

In this session, we will:

- Revisit simple logistic regression and write it in a form resembling an artificial neural network
- Meet key concepts in neural networks
  - Gradient descent, back-propagation, mini-batches etc.
- Extend the logistic regression to more than two categories of outcome (analogous to multinomial regression)
- Explore a simple example (MNIST)

Table of Contents

Classic logistic regression model

Suppose we have

- data for $i = 1, 2, ..., N$ sampling units (often people)
- a binary outcome y_i
- three covariates, x_{1i}, x_{2i}, x_{3i}

We will put the covariates in a vector, $\mathbf{x} = (1, x_1, x_2, x_3)^T$

Classic logistic regression model

The classic logistic regression model is:

$$log\left(\frac{p_i}{1-p_i}\right) = \boldsymbol{\theta}^T \mathbf{x}_i$$

where $p_i = \mathbb{P}(y_i = 1|\mathbf{x}_i)$

Parameters:

- The unknown parameter to be estimated is $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2, \theta_3)^T$
- θ_0 is the constant or intercept term
- $\theta_1, \theta_2, \theta_3$ are the regression coefficients

Classic logistic regression model

Logistic regression model:

$$log\left(\frac{p_i}{1-p_i}\right) = \theta^T \mathbf{x}_i$$

Equivalently, we can rewrite this as:

$$p_i = \frac{exp(\theta^T \mathbf{x}_i)}{1 + exp(\theta^T \mathbf{x}_i)}$$
$$= \frac{1}{1 + exp(-\theta^T \mathbf{x}_i)}$$

Classic logistic regression model - maximum likelihood

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:

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

We want to find parameters θ that

- *maximise* the likelihood
- *maximise* the log-likelihood
- *minimise* the negative log-likelihood, (let $J(\theta) = -l(\theta)$)

Estimation of logistic regression parameters

There are many methods for parameter estimation (or numerical optimization). Some familiar ones are:

- Newton-Raphson
- Fisher Scoring
- Iteratively Reweighted Least Squares

Newton Raphson

- Newton-Raphson is an iterative process.
- To find the value θ that minimises the function $-l(\theta) = J(\theta)$ we take a step from the current value to what we hope is a closer value:

$$\theta_{j+1} = \theta_j - \left(\frac{\partial^2}{\partial^2 \theta} J(\theta)\right)^{-1} \left(\frac{\partial}{\partial \theta} J(\theta)\right)$$

- We continue until we have convergence.

Newton Raphson

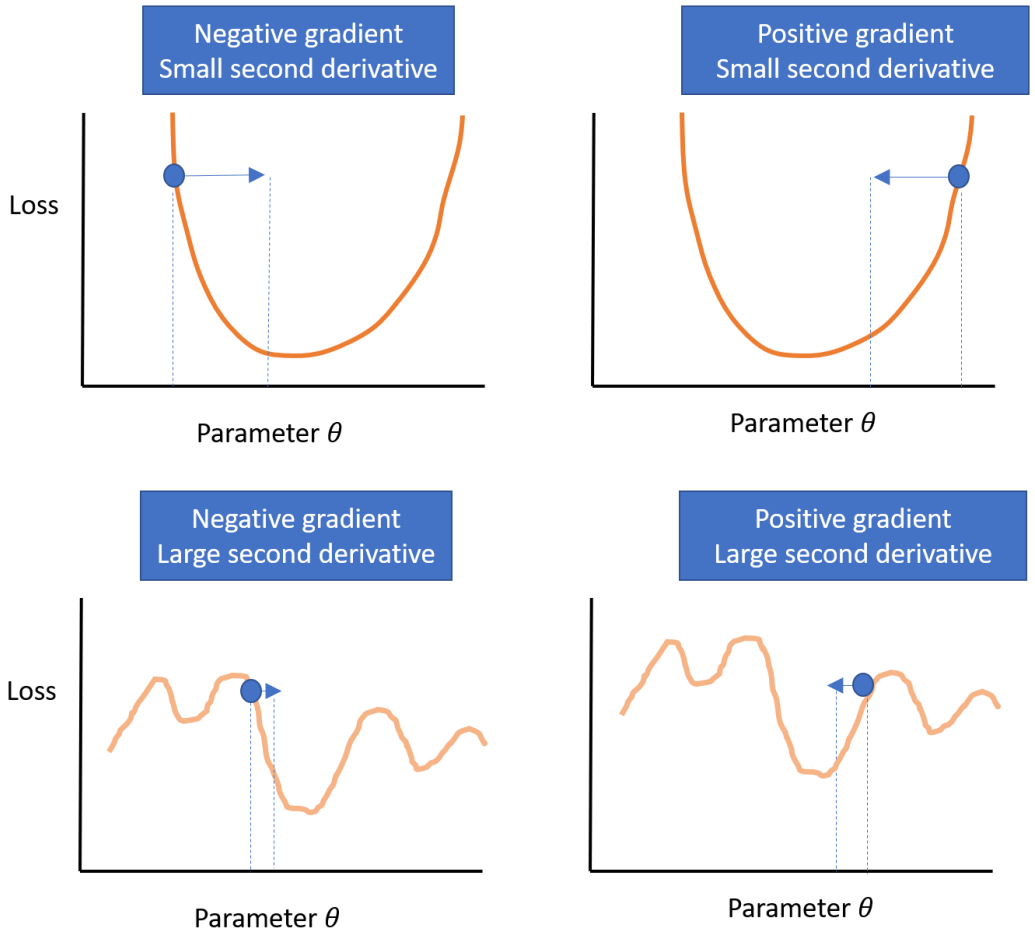


Figure: Newton Raphson updates

Newton Raphson: problems

Rough interpretation:

- We are taking the step in the direction of the steepest gradient
- The size of the step is determined by the second derivative
  - if things are changing rapidly we take a small step
  - if things are changing slowly we take a large one.

Problems:

- The algorithm can diverge away from the minimum (in general settings)
- We need to evaluate the second derivative and invert (a large matrix). This is computationally intensive. In big data this is a real problem.

Gradient Descent

In Newton-Raphson the problem is the calculation of the size of the step. Can we replace this with something easier?

$$\boldsymbol{\theta}_{j+1} = \boldsymbol{\theta}_j - \left(\frac{\partial^2}{\partial^2 \boldsymbol{\theta}} J(\boldsymbol{\theta})\right)^{-1} \left(\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta})\right)$$

becomes

$$\boldsymbol{\theta}_{j+1} = \boldsymbol{\theta}_j - \alpha \left(\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta})\right)$$

If α is a scalar hyper-parameter then this is *Gradient Descent*

Gradient Descent - learning rate

$$\boldsymbol{\theta}_{j+1} = \boldsymbol{\theta}_j - \alpha \left(\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) \right)$$

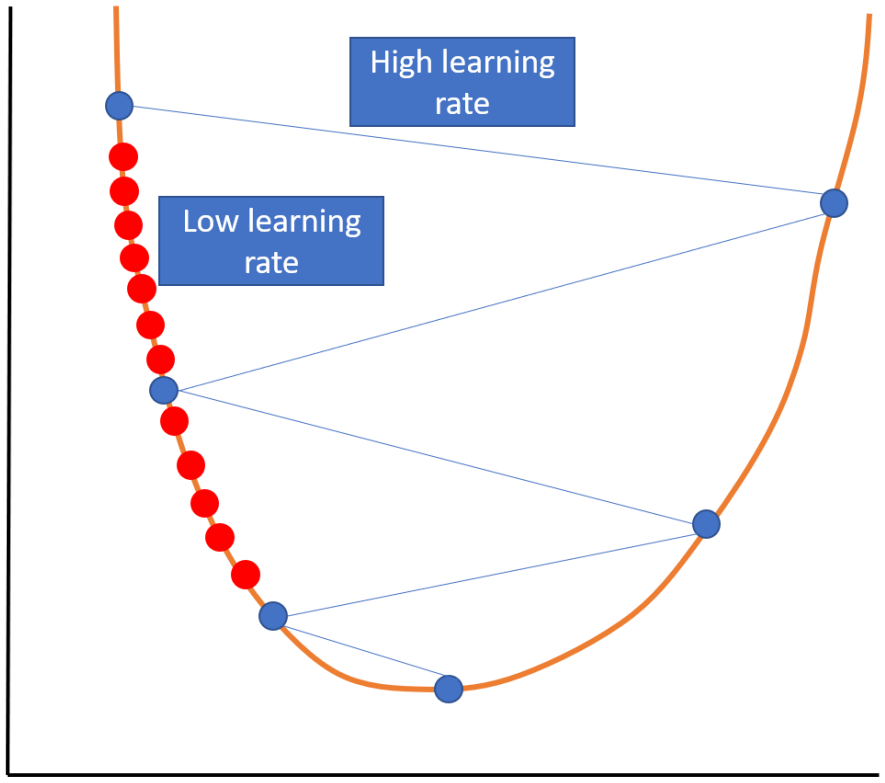
α is

- the *learning rate*
- a *hyperparameter*

How to choose α?

- If α is very small, it would take long time to converge and become computationally expensive.
- If α is large, it can fail to converge and overshoot the minimum.
- Typically chosen as 0.001, 0.003, 0.01, 0.03, 0.1, 0.3.

Gradient Descent - learning rate



Gradient Descent - momentum

- Instead of changing direction completely each step, we can think of the update process as having *momentum*
- i.e. we're moving in a particular direction, and then the update gives a suggested new direction, which nudges it from its current direction.

Gradient Descent

For each iteration of Gradient Descent, in deep learning this is thought of as three separate steps:

1. A *forward pass*: take the variables ($\mathbf{x}$) and current parameter values θ_j and use them to calculate the loss function.
   - Although this is not strictly necessary to obtain the gradient, neural networks use a version of the chain rule to obtain the gradients, which requires this forward pass to be done before the next step.
2. *Back-propagation*: calculation of the first derivative (the gradient)
3. *Parameter update*

Note: we think of this process as parameter estimation. In deep learning, this is thought of as *numerical optimization*. The focus is taken away from individual parameter estimates.

Gradient Descent in large samples

- In very large samples even Gradient Descent can be slow.
- Instead we can randomly split the data into *mini-batches* of e.g. 64 people and do a parameter update on each in turn.
- This is called *Stochastic Gradient Descent**
- Each pass of a mini-batch through the optimisation algorithm is called an *iteration*.
- Once every person in the sample has been used once within a parameter update this is called an *epoch*
- This is faster than standard Gradient Descent
- Needs less memory
- Convergence can be more noisy

* Terminology varies - some call this ‘mini-batch gradient descent’

Notation change

- We will split our parameter θ into two parts:

$$\theta = (b, \mathbf{w}), \text{where } \mathbf{w} = (w_1, w_2, w_3)^T$$

  - The constant term b is known as the *bias* term
  - The regression coefficients $\mathbf{w}$ are known as the *weights*

- We will drop the constant from the covariate vector:

$\mathbf{x} = (x_1, x_2, x_3)^T$

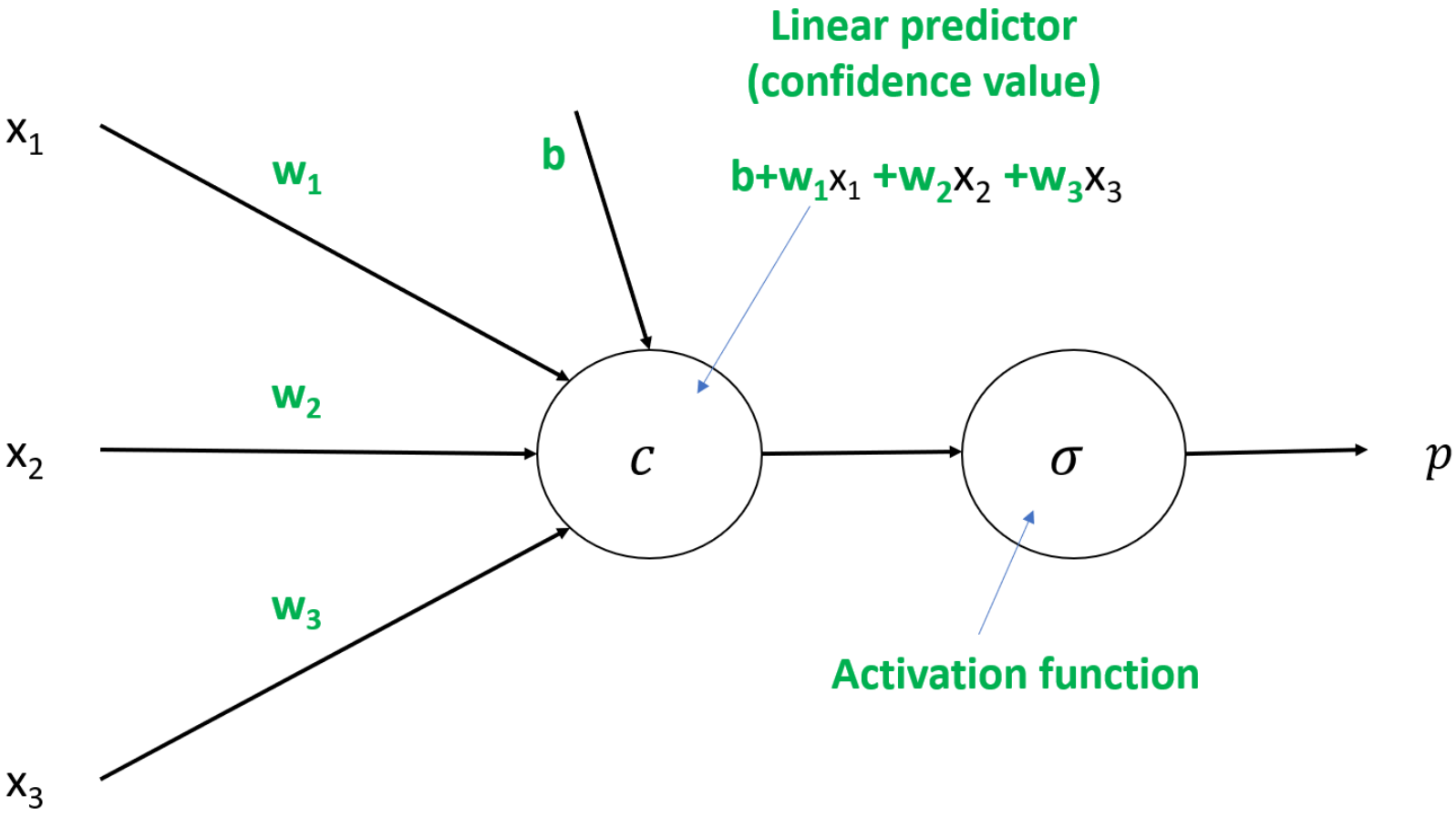
Then

$$p_i = \sigma\left(b + \mathbf{w}^T\mathbf{x}\right)$$

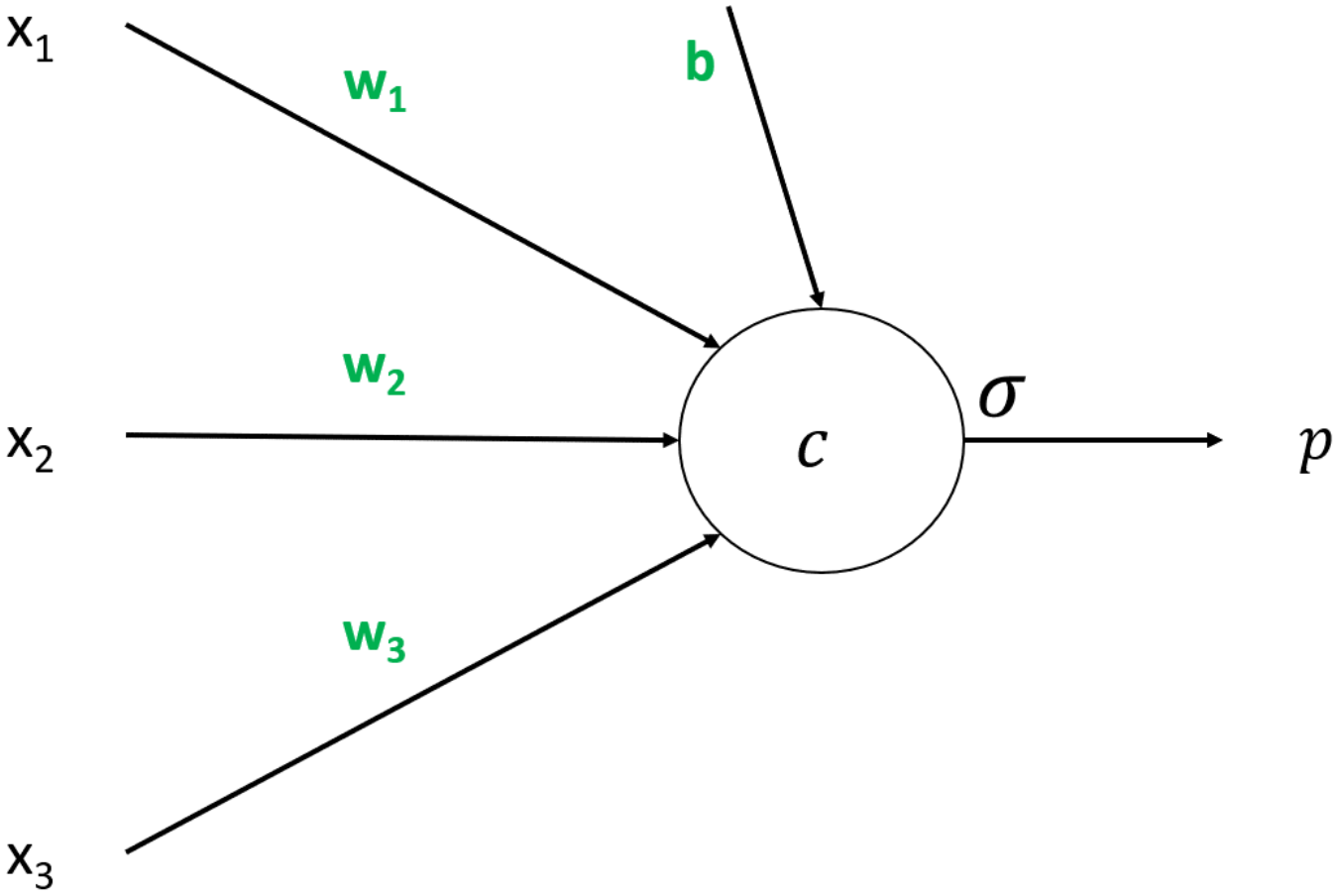
where

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Logistic regression - a perceptron (neuron)



Logistic regression - a perceptron (neuron)



Classic logistic regression model

Same concepts but different words:

Classic logistic regression	Neural network
Sample, unit, individual	Instance
Variable	Input
Constant, intercept	Bias term
Regression or beta coefficients	Weights
Linear predictor	Confidence value
Link function	Activation function
(maps $\mathbb{E}[Y]$ to linear predictor)	(maps linear predictor to $\mathbb{E}[Y]$)
Max. (log-)likelihood	Min. loss function, objective function

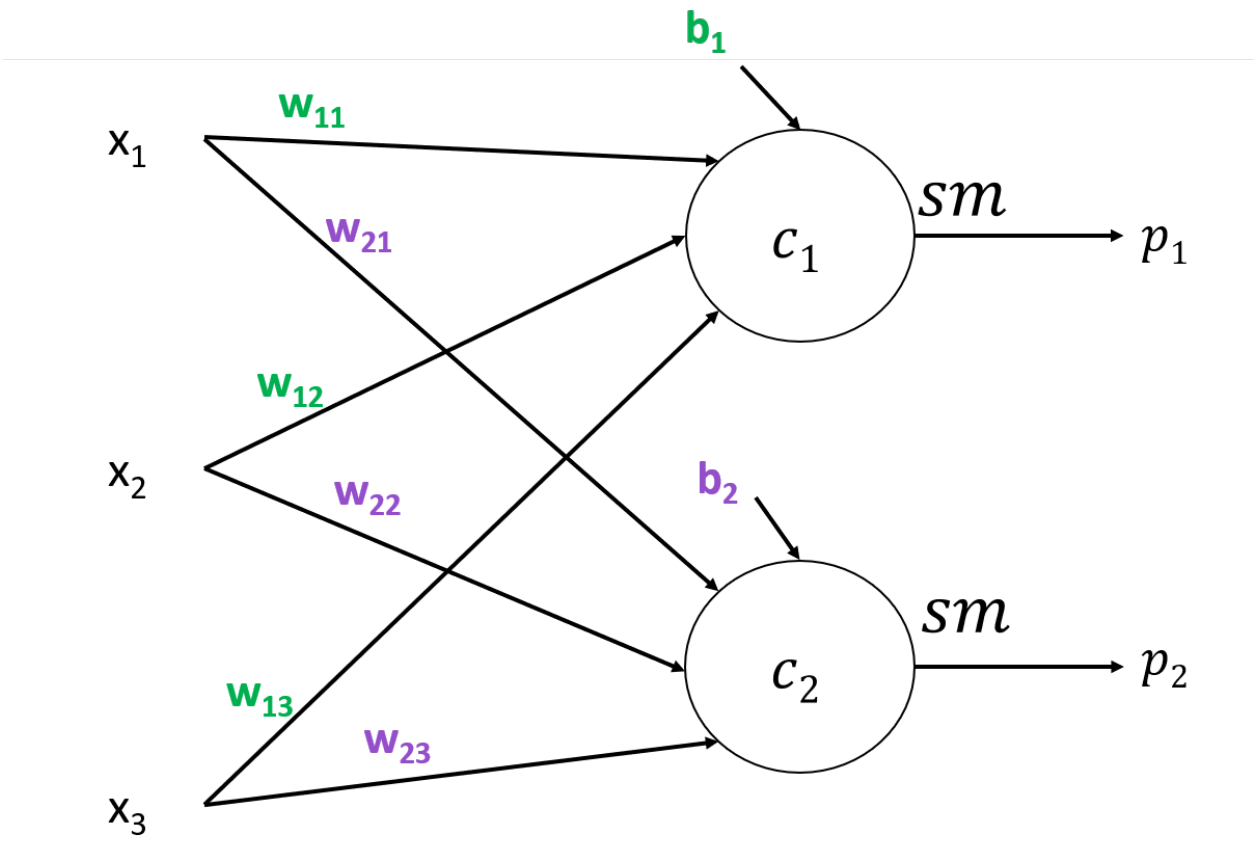
Different concepts:

	Classic logistic regression	Neural network
Algorithm	Newton-Raphson (NR)	(Stochastic) Gradient Descent
Batch	-	Random subset of data
Iteration	One parameter update	One batch used (one update!)
Epoch	-	All samples used once
Learning rate	Determined by NR	Hyper-parameter
Aim	Parameter estimation	Optimization

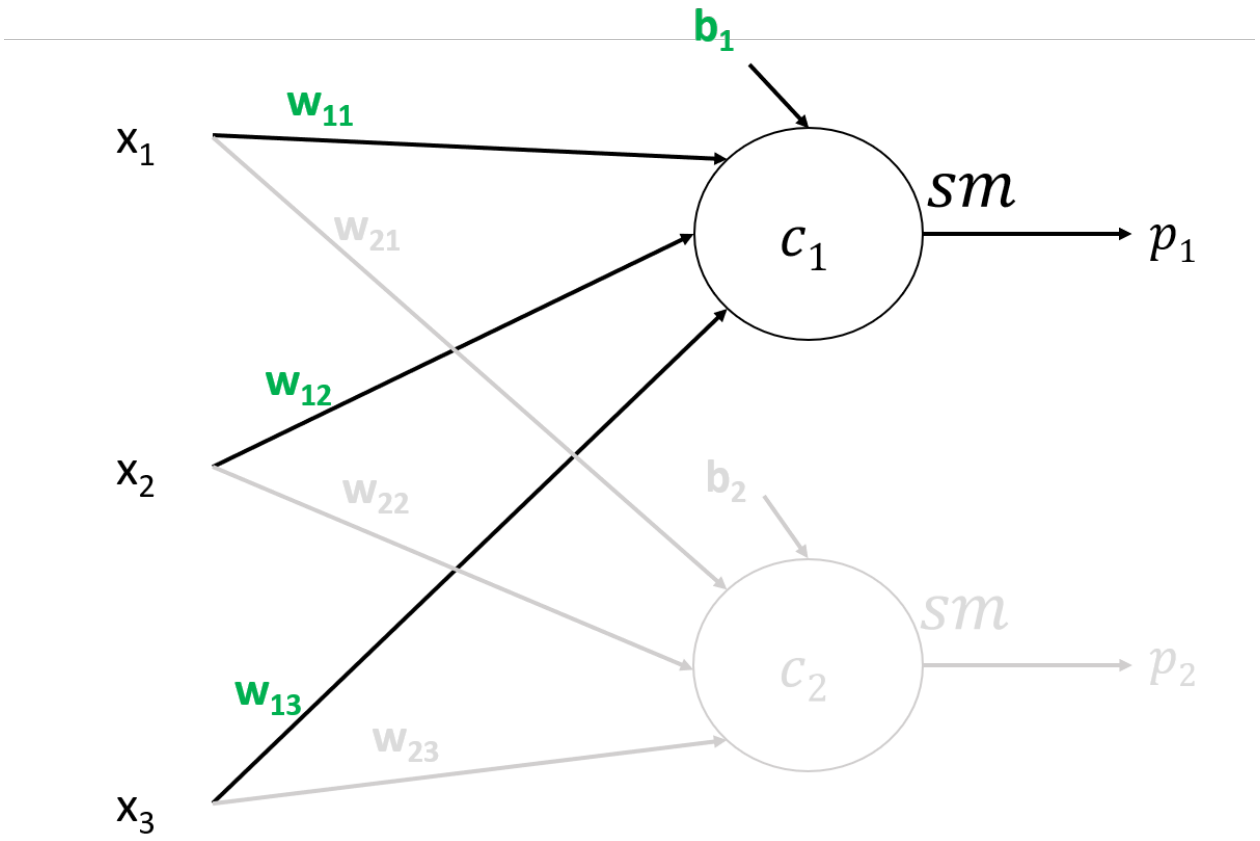
Table of Contents

Multiclass problems

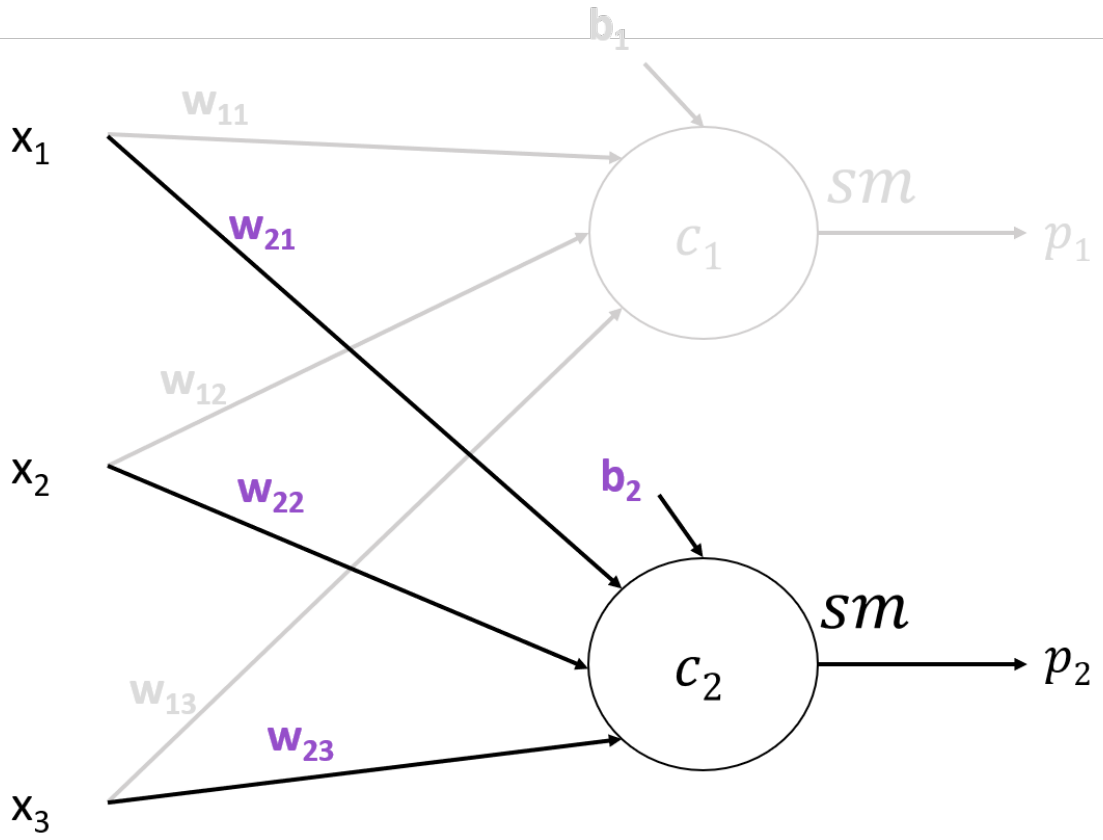
More often a classification task involves classifying objects into a set of $K > 2$ categories. [Illustration shows K=2 for simplicity]



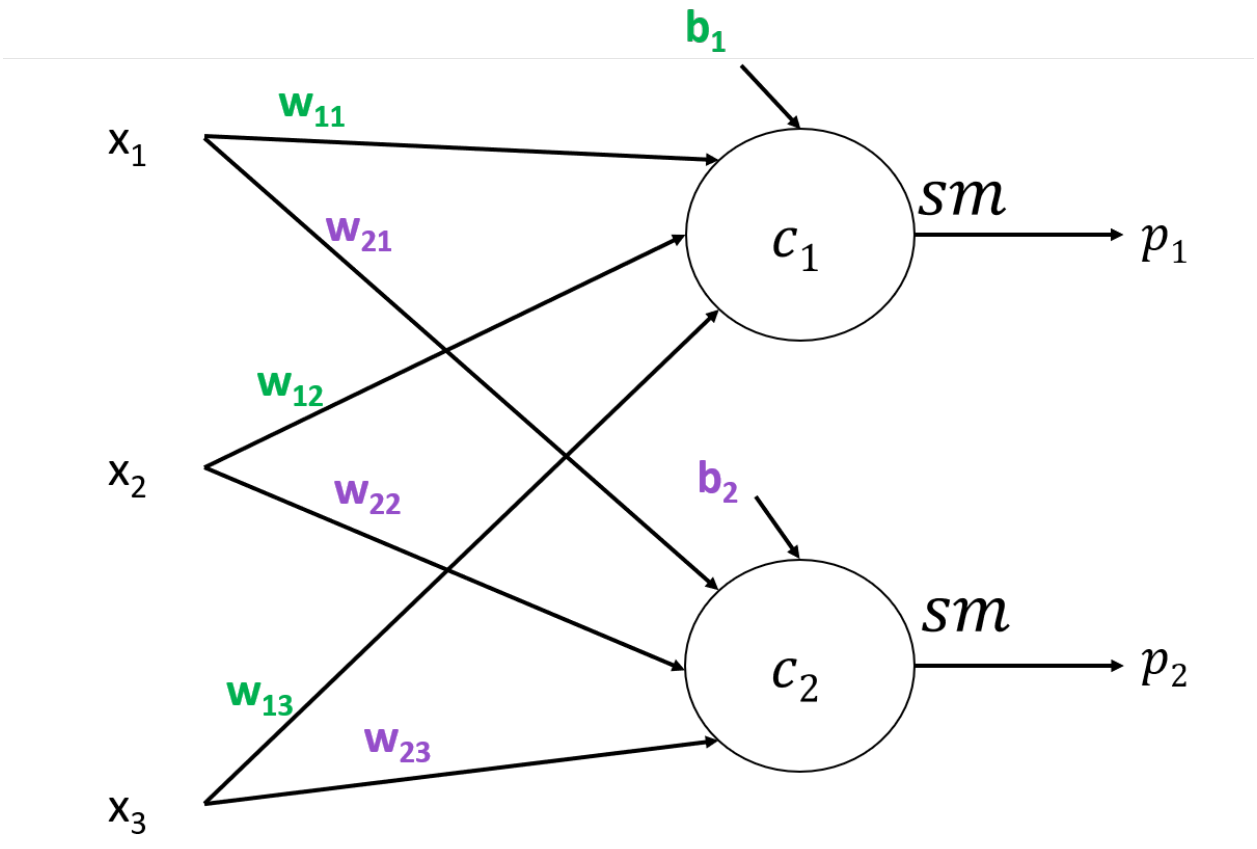
Multiclass problems



Multiclass problems



Multiclass problems



Multiclass problems: the softmax function

- We can't use the σ activation function now that there are two confidence values (i.e. two linear predictors).
- Instead we use the *softmax* function
- Here, this is:

$$p_1 = \frac{exp(c_1)}{exp(c_1) + exp(c_2)}$$

$$p_2 = \frac{exp(c_2)}{exp(c_1) + exp(c_2)}$$

- This function results in the end set of class probabilities forming a probability distribution
- An instance (sampling unit) is classified as the class with the highest probability

Multiclass problems: the loss function

- We still minimise the negative log likelihood, but taking into account the fact we have multiple outcome categories
- This is an example of *cross-entropy loss*
- This is equivalent to the multinomial regression negative log likelihood

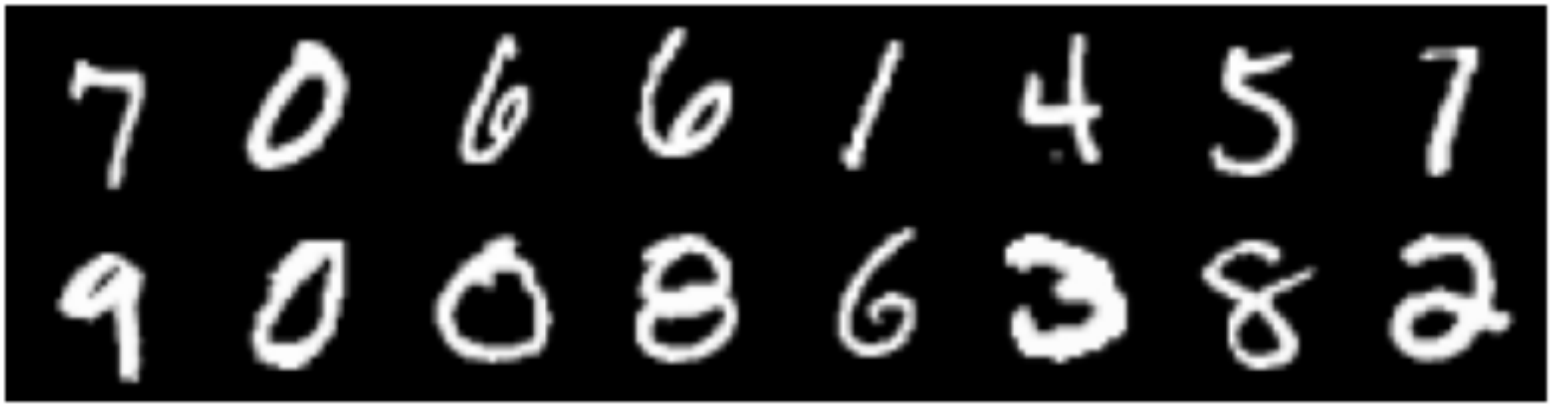
Table of Contents

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

Using our network for multiclass problems, we have:

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

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

Training loss

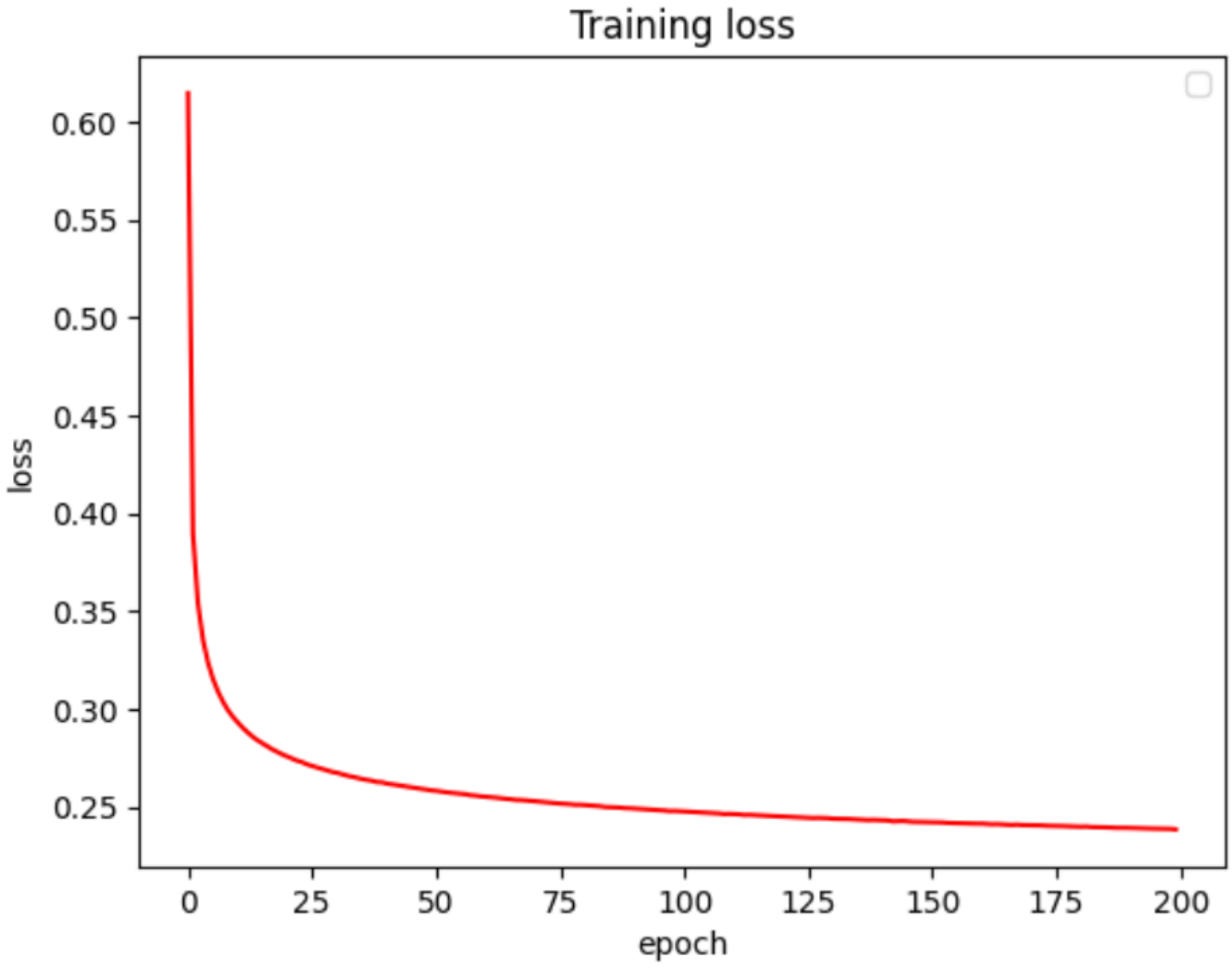


Table of Contents

Summary

- We have re-written standard logistic regression as a simple neural network
- We have extended this to classification tasks involving multiple outcome classes
- Next steps: We will see how this naturally extends to incorporate additional layers (Multi Layer Perceptrons)