COMS30035, Machine learning:

Classification and Neural Networks

Edwin Simpson (adapted from slides by Rui Ponte Costa)

Department of Computer Science, SCEEM
University of Bristol

October 3, 2023

Textbooks

We will follow parts of the Chapter 4 and 5 of the Bishop book:

- Bishop, C. M., Pattern recognition and machine learning (2006). Available for free here.

Agenda

- Discriminant functions
- Logistic regression
- Perceptron
- Neural networks (multi-layer perceptron)
  - Architecture
  - The backpropagation algorithm
  - Gradient descent

See: [Chapter 5, Bishop]

Classification

- It is the classical example of **supervised learning**
- Goal: Classify input data into one of K classes

Classification

- It is the classical example of **supervised learning**
- Goal: Classify input data into one of K classes
- Model: *Discriminant function*:
  - A function that takes an input vector x and assigns it to class C_k. For simplicity we will focus on $K = 2$ and will first study linear functions (see Bishop for the general cases).

Linear discriminant function

- The simplest linear discriminant (LD) is $\boxed{y(x) = w_0 + \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}}$
  - where y is used to predicted class C_k, x is the input vector (feature values)
  - w_0 is a scalar, which we call *bias*
  - w_T is our vector of parameters, which we call *weights*

[1] See Bishop p184 and p190.

Linear discriminant function

- The simplest linear discriminant (LD) is $\boxed{y(x) = w_0 + \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}}$
  - where y is used to predicted class C_k, x is the input vector (feature values)
  - w_0 is a scalar, which we call *bias*
  - w_T is our vector of parameters, which we call *weights*
- This looks like linear regression! Except the next step...

[1] See Bishop p184 and p190.

Linear discriminant function

- The simplest linear discriminant (LD) is $\boxed{y(x) = w_0 + \boldsymbol{w^T x}}$
  - where y is used to predicted class C_k, x is the input vector (feature values)
  - w_0 is a scalar, which we call *bias*
  - w_T is our vector of parameters, which we call *weights*
- This looks like linear regression! Except the next step...
- For $K = 2$: An input vector x is assigned to class C_1 if $y(x) \geq 0$ and to class C_2 otherwise.

[1] See Bishop p184 and p190.

Linear discriminant function

- The simplest linear discriminant (LD) is $\boxed{y(x) = w_0 + \boldsymbol{w}^T \boldsymbol{x}}$
  - where y is used to predicted class C_k, x is the input vector (feature values)
  - w_0 is a scalar, which we call *bias*
  - w_T is our vector of parameters, which we call *weights*
- This looks like linear regression! Except the next step...
- For $K = 2$: An input vector x is assigned to class C_1 if $y(x) \geq 0$ and to class C_2 otherwise.
- Optimisation: least-squares (as for regression) [1], where we want to minimise the cost or error function:

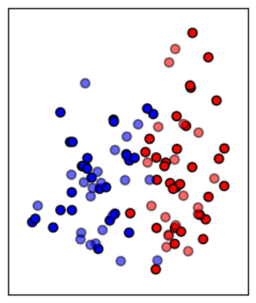
$$\boxed{E = \frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{w}^T\boldsymbol{x}_n + w_0 - t_n)^2}$$ where t_n are the targets/labels (e.g. t_1 = C_1).

[1] See Bishop p184 and p190.

LD and linear separability

Example

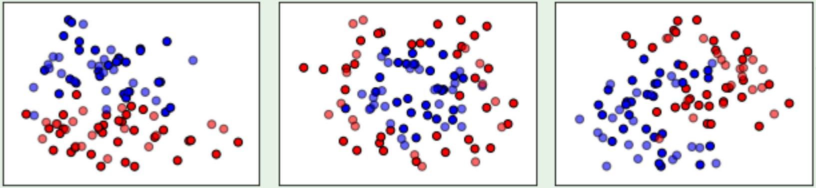
Linear separability is when two sets of points are separable by a line. We generated two sets of points using two Gaussians to illustrate this point, which can easily be fit by a LD. A *decision boundary* is the boundary that separates the two given classes, which our models will try to find.



Linear separability vs nonlinear separability

Example

Which datasets **are** and **are not** linearly separable[2]?

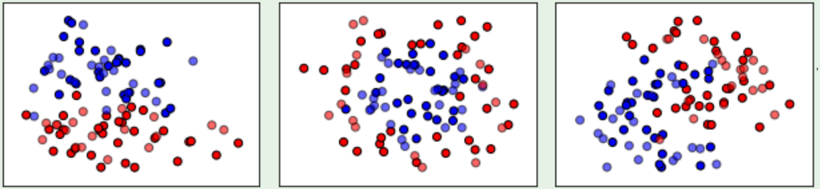


[2] Example from Sklearn here.

Linear separability vs nonlinear separability

Example

Which datasets **are** and **are not** linearly separable[2]?



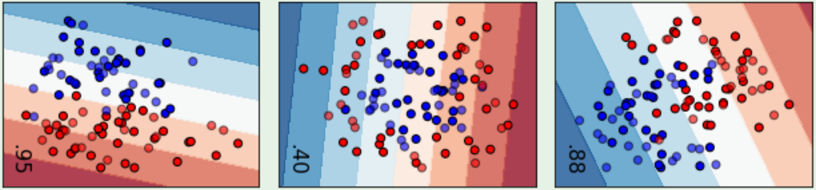
Only the first dataset is linearly separable!

[2] Example from Sklearn here.

Linear discriminant

Example

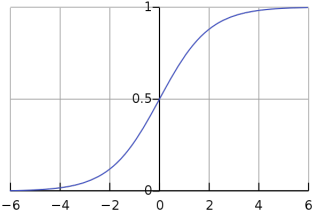
Using sklearn we fitted a LD to the data:



As expected, the LD model only does a good job in finding a good separation in the first dataset.

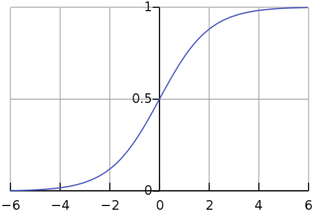
Logistic regression

- We use a logistic function to obtain the probability of class C_k: $\boxed{y(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})}$ where σ denotes the logistic sigmoid function (s-shaped), for example:



Logistic regression

- We use a logistic function to obtain the probability of class C_k: $\boxed{y(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})}$ where σ denotes the logistic sigmoid function (s-shaped), for example:



- such that when $y \to 0$ we choose class 2 and $y \to 1$ class 1.
- Taking a probabilistic view: $p(C_1|\boldsymbol{x}) = y(\boldsymbol{x})$, and $p(C_2|\boldsymbol{x}) = 1 - p(C_1|\boldsymbol{x})$.

Logistic regression – maximum likelihood estimation

Follow MLE recipe:

1. Define likelihood: For a dataset $\{x_n, t_n\}$, where the targets $t_n \in \{0, 1\}$ we have $p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}) = \prod_{n=1}^{N} y_n^{t_n}(1 - y_n)^{1-t_n}$ where $y_n = p(C_1|x_n)$. [3]

[3] The exponent selects the probability of the target class (i.e. if $t_n = 1$ we get y_n; if $t_n = 0$ we get $1 - y_n$).

[4] Note that we used the logarithm product and power rule.

[5] This solution makes sense since we want to optimise the difference between the model output y and the desired targets t.

Logistic regression – maximum likelihood estimation

Follow MLE recipe:

1. Define likelihood: For a dataset $\{x_n, t_n\}$, where the targets $t_n \in \{0, 1\}$ we have $p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}) = \prod_{n=1}^{N} y_n^{t_n}(1 - y_n)^{1-t_n}$ where $y_n = p(C_1|x_n)$. [3]
2. Take negative logarithm of the likelihood [4]:
$$-ln\, p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}) = -\sum_{n=1}^{N}\{t_n ln\, y_n + (1 - t_n) ln(1 - y_n)\}$$

---

[3] The exponent selects the probability of the target class (i.e. if $t_n = 1$ we get y_n; if $t_n = 0$ we get $1 - y_n$).

[4] Note that we used the logarithm product and power rule.

[5] This solution makes sense since we want to optimise the difference between the model output y and the desired targets t.

Logistic regression – maximum likelihood estimation

Follow MLE recipe:

1. Define likelihood: For a dataset $\{x_n, t_n\}$, where the targets $t_n \in \{0, 1\}$ we have $p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}) = \prod_{n=1}^{N} y_n^{t_n}(1 - y_n)^{1-t_n}$ where $y_n = p(C_1|x_n)$. [3]
2. Take negative logarithm of the likelihood [4]:
$$-ln\, p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}) = -\sum_{n=1}^{N}\{t_n ln\, y_n + (1 - t_n) ln(1 - y_n)\}$$
3. Calculate the derivative w.r.t. the parameters $\boldsymbol{w}$:[5]
$$\frac{d\, ln\, p(\boldsymbol{t}|\boldsymbol{x},\boldsymbol{w})}{d\boldsymbol{w}} = \sum_{n=1}^{N}(y_n - t_n)x_n$$

---

[3] The exponent selects the probability of the target class (i.e. if $t_n = 1$ we get y_n; if $t_n = 0$ we get $1 - y_n$).

[4] Note that we used the logarithm product and power rule.

[5] This solution makes sense since we want to optimise the difference between the model output y and the desired targets t.

Logistic regression – maximum likelihood estimation

Follow MLE recipe:

1. Define likelihood: For a dataset $\{x_n, t_n\}$, where the targets $t_n \in \{0, 1\}$ we have $p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}) = \prod_{n=1}^{N} y_n^{t_n}(1 - y_n)^{1-t_n}$ where $y_n = p(C_1|x_n)$. [3]
2. Take negative logarithm of the likelihood [4]:
$$-ln\, p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}) = -\sum_{n=1}^{N}\{t_n ln\, y_n + (1 - t_n) ln(1 - y_n)\}$$
3. Calculate the derivative w.r.t. the parameters $\boldsymbol{w}$:[5]
$$\frac{d\, ln\, p(\boldsymbol{t}|\boldsymbol{x},\boldsymbol{w})}{d\boldsymbol{w}} = \sum_{n=1}^{N}(y_n - t_n)x_n$$
4. Now we can use Eq. above to directly update $\boldsymbol{w}$ using the data x.

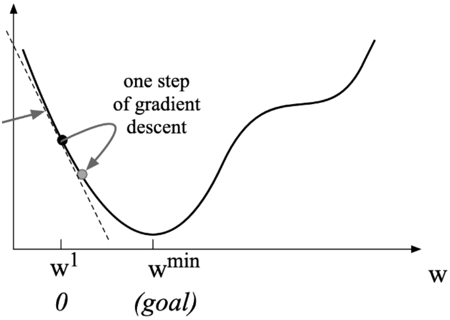
---

[3] The exponent selects the probability of the target class (i.e. if $t_n = 1$ we get y_n; if $t_n = 0$ we get $1 - y_n$).

[4] Note that we used the logarithm product and power rule.

[5] This solution makes sense since we want to optimise the difference between the model output y and the desired targets t.

MLE using Gradient Descent



- Start with random weight values
- We want to adjust each weight w to minimise negative log likelihood: move downhill to the minimum
- The derivative represents the slope: $\frac{d \ln p(\mathbf{t}|\mathbf{x},\mathbf{w})}{d\mathbf{w}} = \sum_{n=1}^{N} (y_n - t_n) x_n$
- Increase or decrease w by a small amount in the downward direction

Logistic regression – maximum likelihood estimation

More details on calculating the derivative:

1. From here $-ln\, p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}) = -\sum_{n=1}^{N}\{t_n ln\, y_n + (1 - t_n)ln(1 - y_n)\}$

[6] We used the chain rule and $d\ ln(x) = 1/x$. We also used the derivative of the sigmoid $dy_n = y(1 - y_n)$.

[7] You can find the full derivation here.

Logistic regression – maximum likelihood estimation

More details on calculating the derivative:

1. From here $-ln\, p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}) = -\sum_{n=1}^{N}\{t_n ln\, y_n + (1-t_n) ln(1-y_n)\}$
2. We get $\sum_{n=1}^{N}\{-\frac{t_n}{y_n} + \frac{(1-t_n)}{1-y_n}\}\{y_n(1-y_n)\}x_n$ [6]

[6] We used the chain rule and $d\, ln(x) = 1/x$. We also used the derivative of the sigmoid $dy_n = y(1-y_n)$.

[7] You can find the full derivation here.

Logistic regression – maximum likelihood estimation

More details on calculating the derivative:

1. From here $-ln\, p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}) = -\sum_{n=1}^{N}\{t_n ln\, y_n + (1 - t_n) ln(1 - y_n)\}$
2. We get $\sum_{n=1}^{N}\{-\frac{t_n}{y_n} + \frac{(1-t_n)}{1-y_n}\}\{y_n(1 - y_n)\}x_n$ [6]
3. The above simplifies to $\sum_{n=1}^{N}\{-t_n(1 - y_n) + (1 - t_n)y_n\}x_n$

---

[6] We used the chain rule and $d\, ln(x) = 1/x$. We also used the derivative of the sigmoid $dy_n = y(1 - y_n)$.

[7] You can find the full derivation here.

Logistic regression – maximum likelihood estimation

More details on calculating the derivative:

1. From here $-ln\, p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}) = -\sum_{n=1}^{N}\{t_n ln\, y_n + (1-t_n) ln(1-y_n)\}$
2. We get $\sum_{n=1}^{N}\{-\frac{t_n}{y_n} + \frac{(1-t_n)}{1-y_n}\}\{y_n(1-y_n)\}x_n$ [6]
3. The above simplifies to $\sum_{n=1}^{N}\{-t_n(1-y_n) + (1-t_n)y_n\}x_n$
4. And in turn to $\sum_{n=1}^{N}\{y_n - t_n\}x_n$ [7]

---

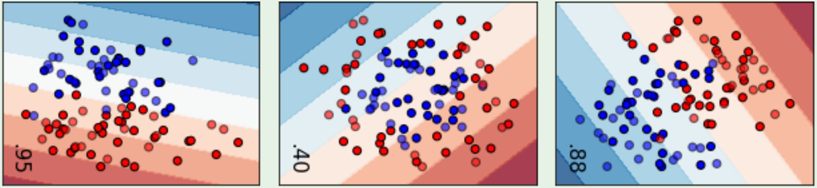
[6] We used the chain rule and $d\, ln(x) = 1/x$. We also used the derivative of the sigmoid $dy_n = y(1-y_n)$.

[7] You can find the full derivation here.

Logistic regression

Example

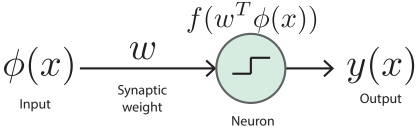
Using sklearn we fitted a logistic regression classifier to the data:



As you can see the results are very similar to LD, but because of probabilistic formulation we have an explicit probability of belonging to one or the other class (not shown); this can be very useful in real-world applications (e.g. self-driving cars or cancer detection).

Perceptron – a simplified neural network

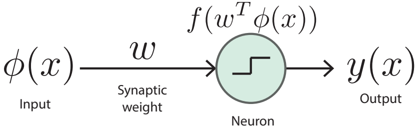
- It is the very beginning of neural network models in ML!
- It is directly inspired by how neurons process information:



[8] Intuitively we want to improve our chances of having $t_n = y_n = -1$ or $t_n = y_n = 1$, which will both decrease our error function.

Perceptron – a simplified neural network

- It is the very beginning of neural network models in ML!
- It is directly inspired by how neurons process information:



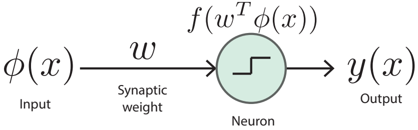
- It is an example of a linear discriminant model given by $y(\boldsymbol{x}) = f(\boldsymbol{w}^T\phi(\boldsymbol{x}))$

  with a nonlinear *activation function* $f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$

[8] Intuitively we want to improve our chances of having $t_n = y_n = -1$ or $t_n = y_n = 1$, which will both decrease our error function.

Perceptron – a simplified neural network

- It is the very beginning of neural network models in ML!
- It is directly inspired by how neurons process information:

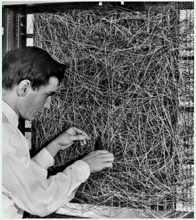


- It is an example of a linear discriminant model given by $y(\boldsymbol{x}) = f(\boldsymbol{w}^T\phi(\boldsymbol{x}))$
  with a nonlinear *activation function* $f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$
- Here the target $t = \{+1, -1\}$.
- And we aim to mimimise the following error $-\sum_{n=1}^{N} \boldsymbol{w}^T\phi_n t_n$ [8]

[8] Intuitively we want to improve our chances of having $t_n = y_n = -1$ or $t_n = y_n = 1$, which will both decrease our error function.

Perceptron – a simplified neural network

Example



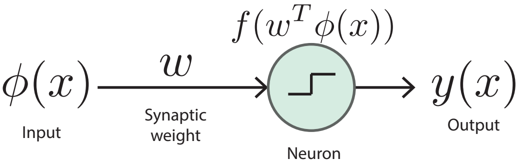
The Perceptron of Rosenblatt (1962)

Perceptrons started the journey to the current *deep learning* revolution!
Frank Rosenblatt used IBM and special-purpose hardware for a parallel implementation of perceptron learning.
Marvin Minksy, showed that such models could only learn *linearly separable problems*.
However, this limitation is only true in the case of single layers!

source: Bishop p193.

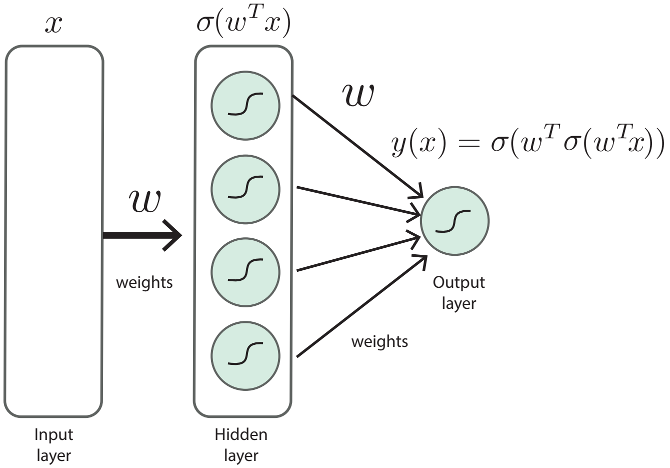
Neural networks

From a single layer perceptron:



Neural networks

To a Multiple Layer Perceptron (MLP) [9]:



[9] Although, we call it perceptron, it typically uses logistic sigmoid activation functions (continous nonlinearities), instead of step-wise discontinous nonlinearities.

Neural networks

- Neural networks are at heart composite functions of linear-nonlinear functions.
- **Deep learning**[10] refers to neural networks (or MLPs) with more than 1 hidden layer
- They can be applied in any form of learning, but we will focus on supervised learning and classification in particular

[10] If you would like to learn more take our Applied Deep Learning unit in your 4th year.

[11] Here we focus on simple feedforward nnets but the recipe is the same for any neural network.

[12] Note that this makes them parametric models.

Neural networks

- Neural networks are at heart composite functions of linear-nonlinear functions.
- **Deep learning**[10] refers to neural networks (or MLPs) with more than 1 hidden layer
- They can be applied in any form of learning, but we will focus on supervised learning and classification in particular
- MLP recipe [11]:
  - Define architecture (e.g. how many hidden layers and neurons) [12]
  - Define cost function (e.g. mean squared error)
  - Optimise network using backprop:
    1. Forward pass – calculate activations; generate y_k
    2. Calculate error/cost function
    3. Backward pass – use backprop to update parameters

---

[10] If you would like to learn more take our Applied Deep Learning unit in your 4th year.

[11] Here we focus on simple feedforward nnets but the recipe is the same for any neural network.

[12] Note that this makes them parametric models.

Neural networks – forward pass step-by-step

1. Calculate activations of the hidden layer h: $a_j = \sum_{i=1}^{D} w_{ji}^{(h)} x_i + w_{j0}^{(h)}$ [linear]
2. Pass it through a nonlinear function: $z_j = \sigma(a_j)$ [nonlinear[13]]

---

[13] In MLP we typically use sigmoid functions.

[14] For classification problems we use a sigmoid at the output, where each output neuron codes for one class.

Neural networks – forward pass step-by-step

1. Calculate activations of the hidden layer h: $a_j = \sum_{i=1}^{D} w_{ji}^{(h)} x_i + w_{j0}^{(h)}$ [linear]
2. Pass it through a nonlinear function: $z_j = \sigma(a_j)$ [nonlinear[13]]
3. Calculate activations of the output layer o: $a_k = \sum_{j=1}^{hiddensize} w_{kj}^{(o)} z_j + w_{k0}^{(o)}$ [linear]
4. Compute predictions using a sigmoid: $y_k = \sigma(a_k)$ [nonlinear[14]]

[13] In MLP we typically use sigmoid functions.

[14] For classification problems we use a sigmoid at the output, where each output neuron codes for one class.

Neural networks – forward pass step-by-step

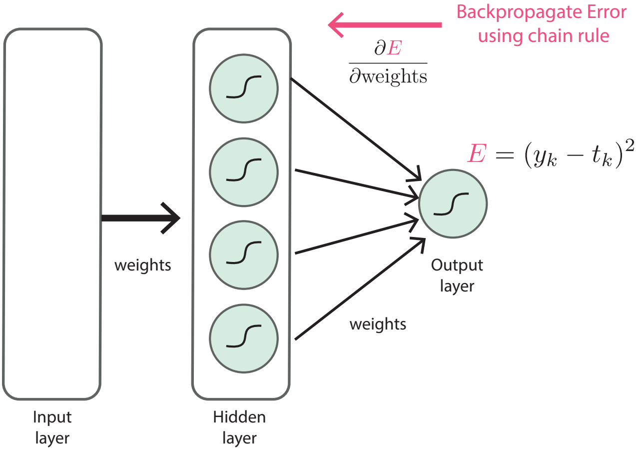
1. Calculate activations of the hidden layer h: $a_j = \sum_{i=1}^{D} w_{ji}^{(h)} x_i + w_{j0}^{(h)}$ [linear]
2. Pass it through a nonlinear function: $z_j = \sigma(a_j)$ [nonlinear[13]]
3. Calculate activations of the output layer o: $a_k = \sum_{j=1}^{hiddensize} w_{kj}^{(o)} z_j + w_{k0}^{(o)}$ [linear]
4. Compute predictions using a sigmoid: $y_k = \sigma(a_k)$ [nonlinear[14]]
5. All together: $y_k = \sigma\left(\sum_{i=1}^{D} w_{kj}\, \sigma\left(\sum_{i=1}^{D} w_{ji} x_i^{(h)} + w_{j0}^{(h)}\right) + w_{k0}^{(o)}\right)$

[13] In MLP we typically use sigmoid functions.

[14] For classification problems we use a sigmoid at the output, where each output neuron codes for one class.

Neural networks – backward pass

We now need to optimise our weights, and as before we use derivatives to find a solution. Effectively backpropagating the output error signal across the network – backpropagation algorithm.



Neural networks – backward pass

We now need to optimise our weights, and as before we use derivatives to find a solution. Effectively backpropagating the output error signal across the network – backpropagation algorithm.

1. Compute the error (or cost) function: e.g.: $E = \frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{y}(\boldsymbol{x_n}, \boldsymbol{w}) - \boldsymbol{t_n})^2$

[15] σ' denotes the derivative of the sigmoid activation function.

[16] Note that the updates for the bias terms w_0 do not depend on the activity of the previous layer z_j and x_i.

Neural networks – backward pass

We now need to optimise our weights, and as before we use derivatives to find a solution. Effectively backpropagating the output error signal across the network – backpropagation algorithm.

1. Compute the error (or cost) function: e.g.: $E = \frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{y}(\boldsymbol{x_n}, \boldsymbol{w}) - \boldsymbol{t_n})^2$
2. Use the *chain rule* to compute the gradients w.r.t. $\boldsymbol{w}$, $\frac{dE}{d\boldsymbol{w}}$
3. For the output weights w_{kj} we get:
$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}} = \sigma'(y_n - t_n)z_j$$ [15]

[15] σ' denotes the derivative of the sigmoid activation function.

[16] Note that the updates for the bias terms w_0 do not depend on the activity of the previous layer z_j and x_i.

Neural networks – backward pass

We now need to optimise our weights, and as before we use derivatives to find a solution. Effectively backpropagating the output error signal across the network – backpropagation algorithm.

1. Compute the error (or cost) function: e.g.: $E = \frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{y}(\boldsymbol{x_n}, \boldsymbol{w}) - \boldsymbol{t_n})^2$
2. Use the *chain rule* to compute the gradients w.r.t. $\boldsymbol{w}$, $\frac{dE}{d\boldsymbol{w}}$
3. For the output weights w_{kj} we get:
   $$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}} = \sigma'(y_n - t_n)z_j$$ [15]
4. Whereas for the input weights w_{ji} we get:
   $$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial a_k}\frac{\partial a_k}{\partial z_j}\frac{\partial z_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \sigma'(y_n - t_n)w_{kj}^T\sigma' x_i$$ [16]

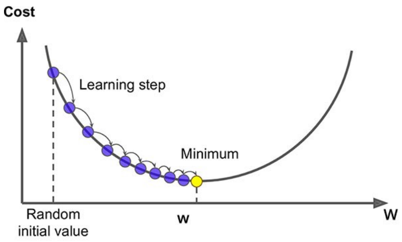
---

[15] σ' denotes the derivative of the sigmoid activation function.

[16] Note that the updates for the bias terms w_0 do not depend on the activity of the previous layer z_j and x_i.

Neural networks – gradient descent [18]

In many ML methods is common to iteratively update the parameters by descending the gradient.



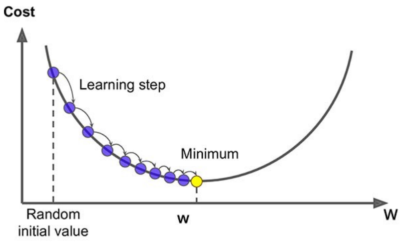
17

[17] Figure from https://mc.ai/an-introduction-to-gradient-descent-2/

[18] Its called descent because we are minimising the cost function, so descending on the function landscape, which can be quite hilly!

Neural networks – gradient descent [18]

In many ML methods is common to iteratively update the parameters by descending the gradient.



[17]

In our neural network this means to update the weights using:

- $w_{ji} = w_{ji} - \Delta w_{ji}$, where $\Delta w_{ji} = \sigma'(y_n - t_n) w_{kj}^T \sigma' x_i$
- $w_{kj} = w_{kj} - \Delta w_{kj}$, where $\Delta w_{kj} = \sigma'(y_n - t_n) z_j$
- This is often done in mini-batches – using a small number of samples to compute Δw.

---

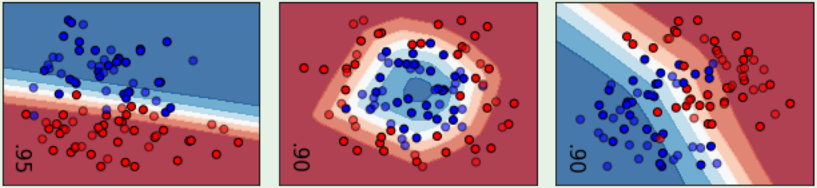
17 Figure from `https://mc.ai/an-introduction-to-gradient-descent-2/`

18 Its called descent because we are minimising the cost function, so descending on the function landscape, which can be quite hilly!

Neural networks

Example

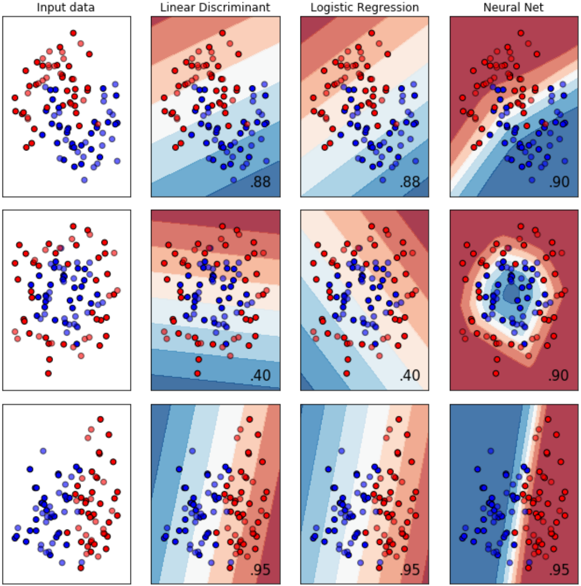
Using sklearn we fitted a MLP classifier to the data:



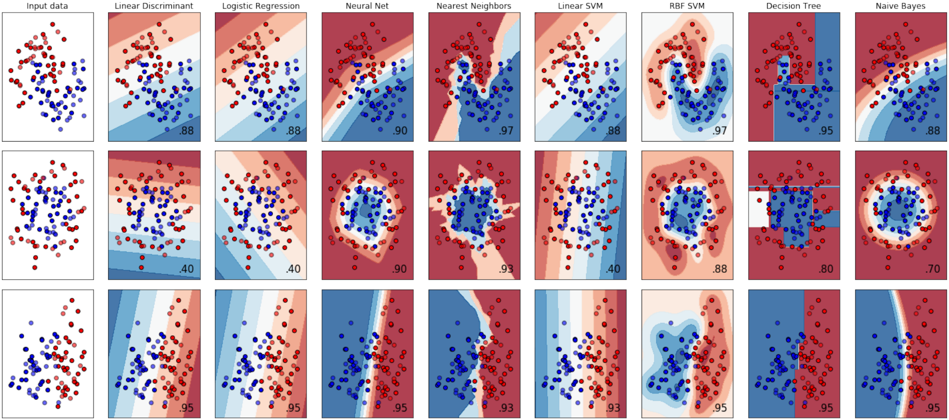
An MLP with one hidden layer can perform well in nonlinear classification problems. However, because MLPs are highly flexible they can easily *overfit*.

Solutions: *early stopping* (stop when test performance starts decreasing) and *regularisation* methods such as *dropout* (randomly turn off units during training).

Classification methods – overall comparison



Classification methods – overall comparison [including methods from the upcoming lectures]



Tasks

- Post questions Teams > QA channel or bring them to the next lecture

Tasks

- Post questions Teams > QA channel or bring them to the next lecture
- Next lab (Week 2): Neural nets and SVMs
    1. See link to lab 2 on BB

Quiz and video time!

Watch this very cool video about the perceptron [19].

Go to Blackboard unit page » Quizzes » Week 1 » Classification and neural networks

[Should take you less than 5 minutes]

[19] Note the comment at the end – it underlies all the recent successes using deep learning!