THE UNIVERSITY of EDINBURGH
informatics

Applied Machine Learning (AML)

Neural Networks

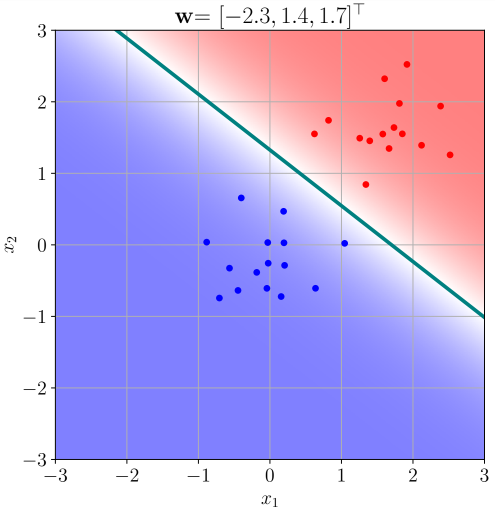
Oisin Mac Aodha • Siddharth N.

Introduction to Neural Networks

Recap - Linear Classifiers

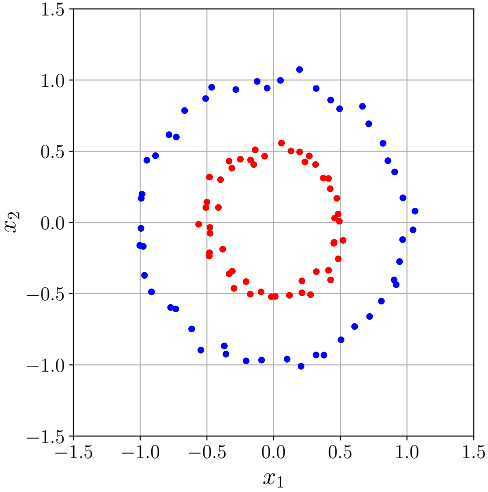
- Logistic regression =
  linear weights + logistic function

$$p(y = 1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$$



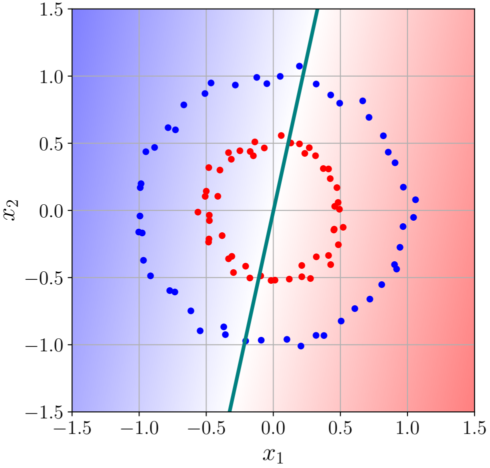
Classifying Non-Linear Data

- There is no linear classifier that will separate the data on the right given these 2D input features.



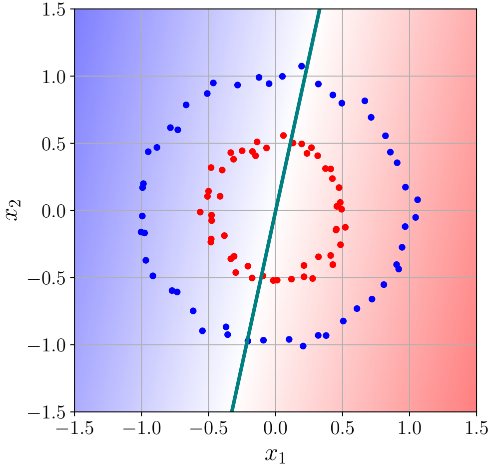
Classifying Non-Linear Data

- There is no linear classifier that will separate the data on the right given these 2D input features.



Classifying Non-Linear Data

- There is no linear classifier that will separate the data on the right given these 2D input features.
- In order to classify it, we can:
  - Use an alternative classifier that can generate *non-linear* decision boundaries.
  - *Transform* our input features so that they are linearly separable.



Neural Networks

- The performance of the classification methods we have explored depend on the input features, i.e. having a good ‘representation‘ of the problem.
- If we do not have good features, many methods will not be effective, e.g. linear classifiers.

Neural Networks

- The performance of the classification methods we have explored depend on the input features, i.e. having a good ‘representation‘ of the problem.
- If we do not have good features, many methods will not be effective, e.g. linear classifiers.

- What if we could *learn* the features from the input data?
- This is what **neural networks** attempt to do.
- We can think of them as a linear method, where we *learn* the features.

Neural Network Intuition

- Neural networks allow us to learn more effective features from the raw input data.
- We can think of them as functions that transform our input into something more useful for our task of interest.

Biological Neurons

- Neural networks are motivated by a *weak* analogy to the human brain, hence the name **artificial neural networks**.

Biological Neurons

- Neural networks are motivated by a *weak* analogy to the human brain, hence the name **artificial neural networks**.
- **Neurons** (also known as nerve cells) are electrically excitable cells in the nervous system that process and transmit information.
- Neurons are the core components of the brain, spinal cord, and nerves of vertebrates.

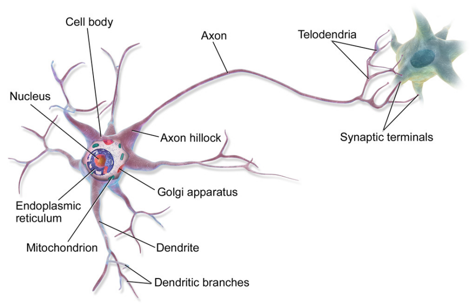


Image by BruceBlaus, CC BY 3.0

Artificial Neurons

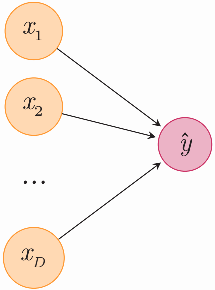
- Each *artificial* neuron is a linear weight vector with a non-linear activation function.
- We compute a neuron's activation as $\hat{y} = g(\boldsymbol{w}^\intercal \boldsymbol{x} + b)$

Artificial Neurons

- Each *artificial* neuron is a linear weight vector with a non-linear activation function.
- We compute a neuron's activation as
  $\hat{y} = g(\boldsymbol{w}^\intercal \boldsymbol{x} + b)$
- Here, $g()$ is a *non-linear activation* function.
- If we used the logistic function this would just be logistic regression.

Artificial Neurons

- Each *artificial* neuron is a linear weight vector with a non-linear activation function.
- We compute a neuron's activation as
  $\hat{y} = g(\boldsymbol{w}^\top \boldsymbol{x} + b)$
- Here, $g()$ is a *non-linear activation* function.
- If we used the logistic function this would just be logistic regression.



Note, we are not depicting the bias term b here.

Artificial Neurons

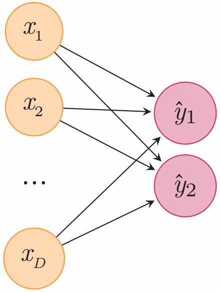
- We can have multiple neurons

$$\hat{y}_1 = g(\boldsymbol{w}_1^\intercal \boldsymbol{x} + b_1)$$
$$\hat{y}_2 = g(\boldsymbol{w}_2^\intercal \boldsymbol{x} + b_2)$$

Artificial Neurons

- We can have multiple neurons

$$\hat{y}_1 = g(\boldsymbol{w}_1^\top \boldsymbol{x} + b_1)$$
$$\hat{y}_2 = g(\boldsymbol{w}_2^\top \boldsymbol{x} + b_2)$$



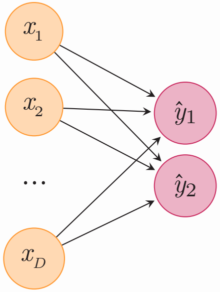
Artificial Neurons

- We can have multiple neurons

$$\hat{y}_1 = g(\boldsymbol{w}_1^\top \boldsymbol{x} + b_1)$$
$$\hat{y}_2 = g(\boldsymbol{w}_2^\top \boldsymbol{x} + b_2)$$

- We can present this using a weight *matrix* $\boldsymbol{W}$ and bias *vector* $\boldsymbol{b}$

$$\hat{\boldsymbol{y}} = g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

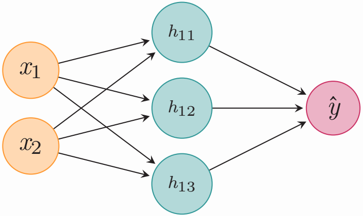


Single Layer Network

- We can connect multiple neurons (i.e. units) together into a directed acyclic graph.
- This results a **feed-forward neural network**.

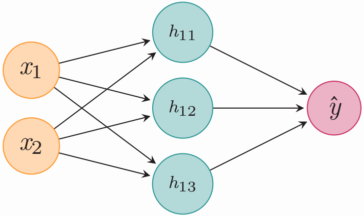
Single Layer Network

- We can connect multiple neurons (i.e. units) together into a directed acyclic graph.
- This results a **feed-forward neural network**.
- One of the simplest neural networks is a **single layer neural network**.



Single Layer Network

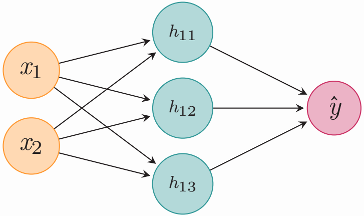
- In a single layer network, we have input units, hidden units, and output units.



Single Layer Network

- In a single layer network, we have input units, hidden units, and output units.
- We can represent this function as

$$\hat{y} = g_2(\boldsymbol{w}_2^\top g_1(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) + b_2)$$



Multilayer Neural Network

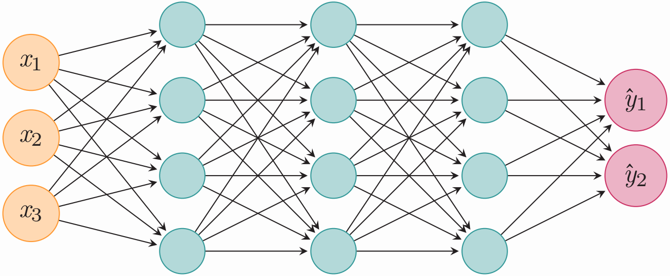
- Individual units in a network are grouped together into **layers**.

Multilayer Neural Network

- Individual units in a network are grouped together into **layers**.
- We can stack multiple layers to form a **multilayer network**, i.e. a multilayer perceptron (MLP).

Multilayer Neural Network

- Individual units in a network are grouped together into **layers**.
- We can stack multiple layers to form a **multilayer network**, i.e. a multilayer perceptron (MLP).
- Here we see a **fully connected network** with three input features, three hidden layers with four hidden units in each, and two output units.



Non-Linear Activation Functions

- Recall our expression for the single layer neural network
  $\hat{y} = g_2(\boldsymbol{w}_2^\top g_1(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + b_2)$

Non-Linear Activation Functions

- Recall our expression for the single layer neural network
  $\hat{y} = g_2(\boldsymbol{w}_2^\top g_1(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) + b_2)$

- One might ask why do we need a non-linear activation function $g()$ in $g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$?

Non-Linear Activation Functions

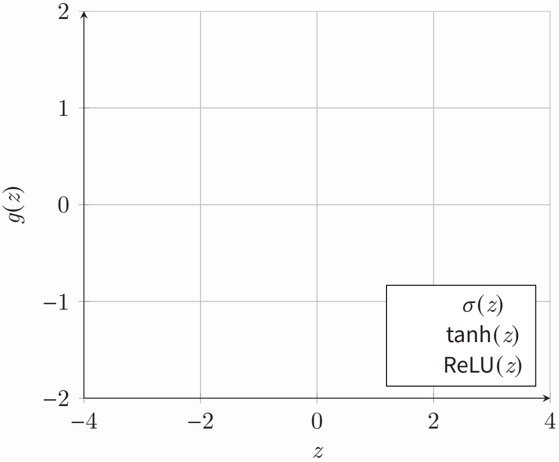
- Recall our expression for the single layer neural network
  $\hat{y} = g_2(\boldsymbol{w}_2^\top g_1(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + b_2)$

- One might ask why do we need a non-linear activation function $g()$ in $g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$?

- Any sequence of linear layers can be equivalently represented with a single linear layer, i.e.

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{W}_1 \boldsymbol{W}_2 \boldsymbol{W}_3 \boldsymbol{x} \\ &\triangleq \boldsymbol{W}' \boldsymbol{x} \end{aligned}$$

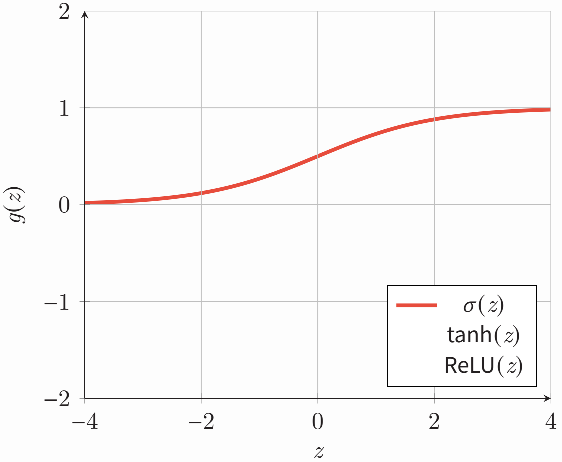
Non-Linear Activation Functions - Choices



Non-Linear Activation Functions - Choices

- Sigmoid i.e. Logistic

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



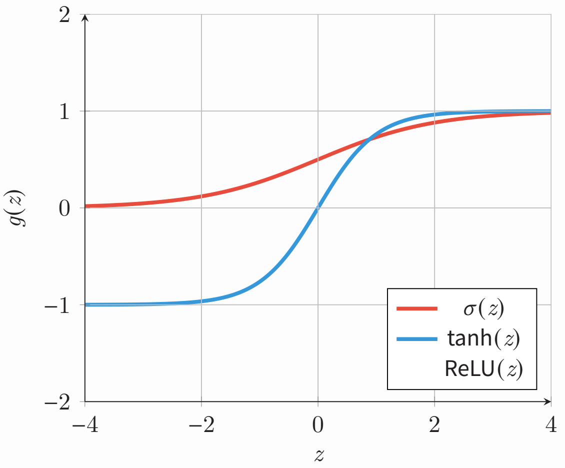
Non-Linear Activation Functions - Choices

- Sigmoid i.e. Logistic

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

- Hyperbolic tangent

$$\tanh(z) = \frac{\exp(2z) - 1}{\exp(2z) + 1}$$



Non-Linear Activation Functions - Choices

- Sigmoid i.e. Logistic

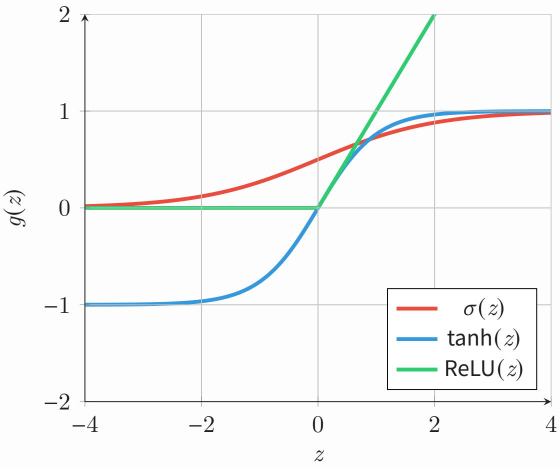
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

- Hyperbolic tangent

$$\tanh(z) = \frac{\exp(2z) - 1}{\exp(2z) + 1}$$

- Rectified linear unit

$$\mathrm{ReLU}(z) = \max(z, 0)$$



Neural Networks - Expressive Power

- Feed-forward neural nets with non-linear activation functions are **universal function approximators**, i.e. they can approximate any function arbitrarily well.

Neural Networks - Expressive Power

- Feed-forward neural nets with non-linear activation functions are **universal function approximators**, i.e. they can approximate any function arbitrarily well.

- In practice, you may need an exponentially large network.
- If you can learn any function, this can just result in overfitting.

Importance of Network Depth

- A fully connected neural network with *one hidden layer* is a universal function approximator.
- This means it can model any sufficiently smooth function given a suitable number of hidden units.

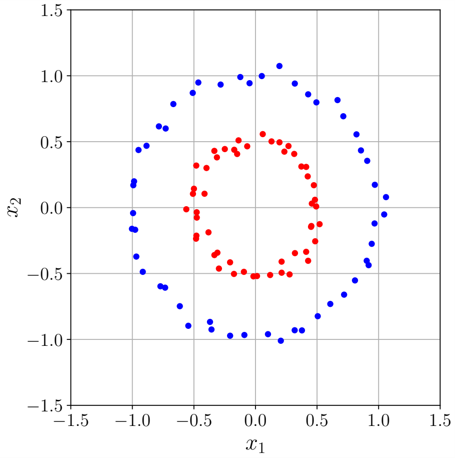
Importance of Network Depth

- A fully connected neural network with *one hidden layer* is a universal function approximator.
- This means it can model any sufficiently smooth function given a suitable number of hidden units.

- However, both experimental and theoretical work have shown that *deeper* neural networks (i.e. ones with more layers) are more effective than *shallow* ones.
- In deeper networks, later layers can leverage the features learned by earlier ones.

Non-Linear Classification with Neural Networks

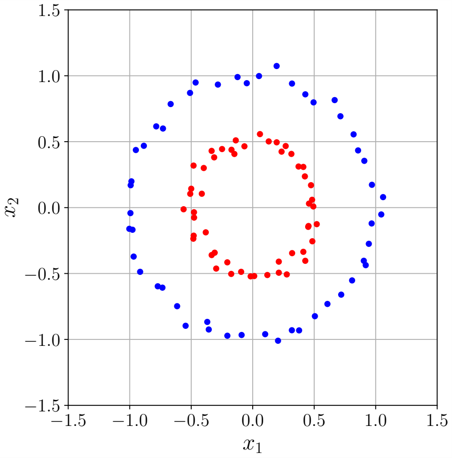
- Here we revisit the non-linear binary classification problem from earlier.



Non-Linear Classification with Neural Networks

- Here we revisit the non-linear binary classification problem from earlier.
- We will use the following neural network with **two** hidden layers:

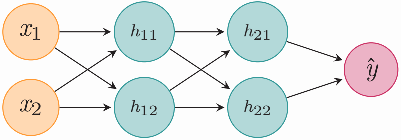
$$\hat{y} = g_3(\boldsymbol{w}_3^\top g_2(\boldsymbol{W}_2 g_1(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2) + b_3)$$

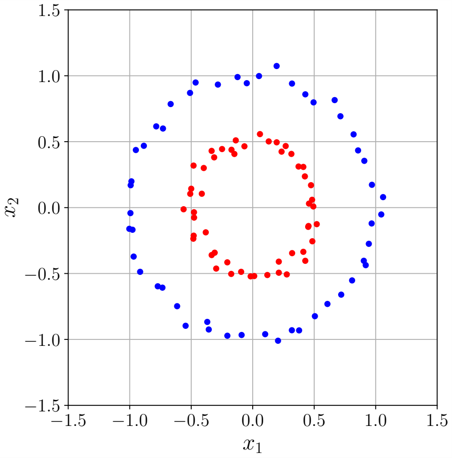


Non-Linear Classification with Neural Networks

- Here we revisit the non-linear binary classification problem from earlier.
- We will use the following neural network with **two** hidden layers:

$$\hat{y} = g_3(\boldsymbol{w}_3^\top g_2(\boldsymbol{W}_2 g_1(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2) + b_3)$$

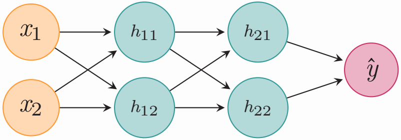


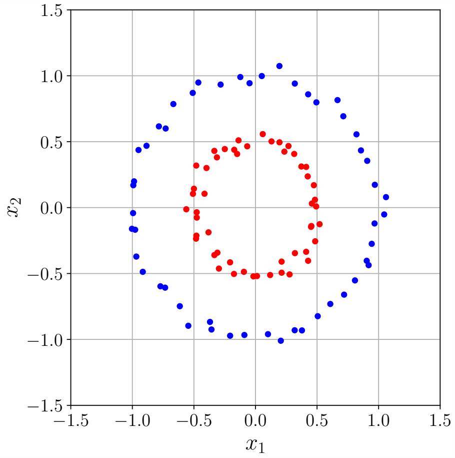


Non-Linear Classification with Neural Networks

- Here we revisit the non-linear binary classification problem from earlier.
- We will use the following neural network with **two** hidden layers:

$$\hat{y} = g_3(\boldsymbol{w}_3^\mathsf{T} g_2(\boldsymbol{W}_2 g_1(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2) + b_3)$$
$$= f(\boldsymbol{x}) = f_3(f_2(f_1(\boldsymbol{x})))$$



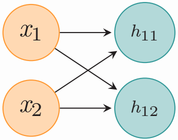


Neural Network Example - Input

- Input x

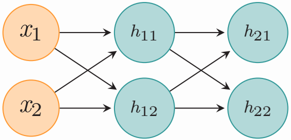
Neural Network Example - First Hidden Layer

- Input $\boldsymbol{x}$
- First hidden layer
  $\boldsymbol{h}_1 = g_1(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)$



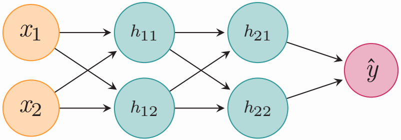
Neural Network Example - Second Hidden Layer

- Input $\boldsymbol{x}$
- First hidden layer
  $\boldsymbol{h}_1 = g_1(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)$
- Second hidden layer
  $\boldsymbol{h}_2 = g_2(\boldsymbol{W}_2 \boldsymbol{h}_1 + \boldsymbol{b}_2)$



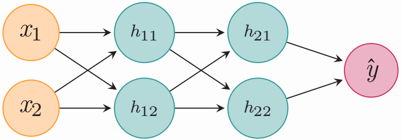
Neural Network Example - Output Layer

- Input $\boldsymbol{x}$
- First hidden layer
  $\boldsymbol{h}_1 = g_1(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)$
- Second hidden layer
  $\boldsymbol{h}_2 = g_2(\boldsymbol{W}_2 \boldsymbol{h}_1 + \boldsymbol{b}_2)$
- Output
  $\hat{y} = g_3(\boldsymbol{w}_3^\top \boldsymbol{h}_2 + b_3)$



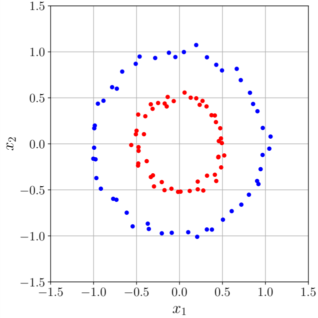
Neural Network Example - Output Layer

- Input $\boldsymbol{x}$
- First hidden layer
  $\boldsymbol{h}_1 = \tanh(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)$
- Second hidden layer
  $\boldsymbol{h}_2 = \tanh(\boldsymbol{W}_2 \boldsymbol{h}_1 + \boldsymbol{b}_2)$
- Output
  $\hat{y} = \sigma(\boldsymbol{w}_3^\intercal \boldsymbol{h}_2 + b_3)$



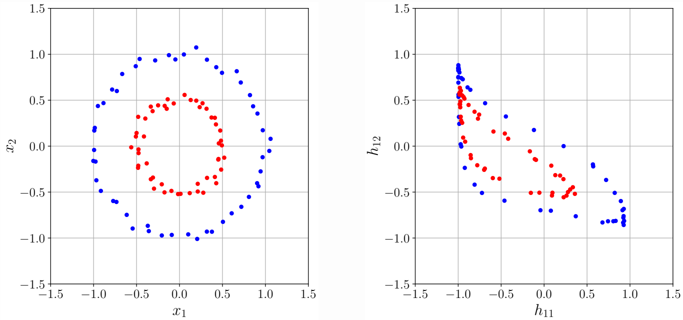
Transforming Features

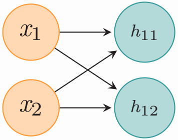
Here we see the outputs from each layer of the network.



Transforming Features

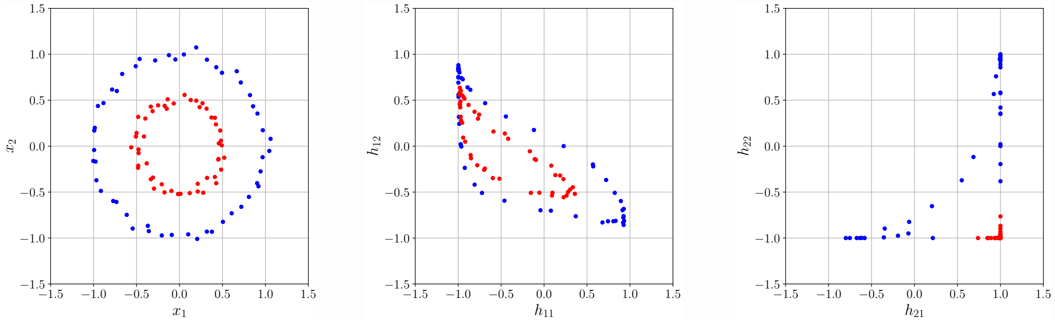
Here we see the outputs from each layer of the network.

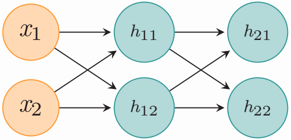




Transforming Features

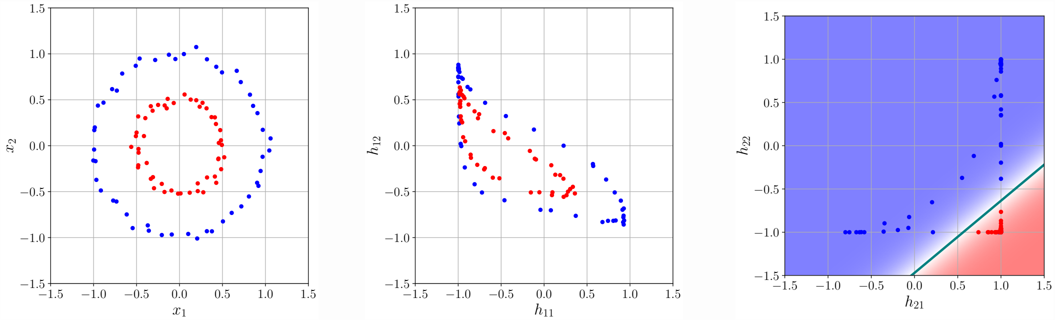
Here we see the outputs from each layer of the network.

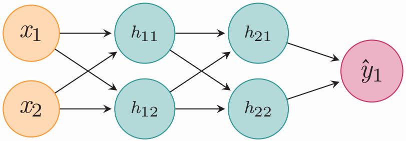




Transforming Features

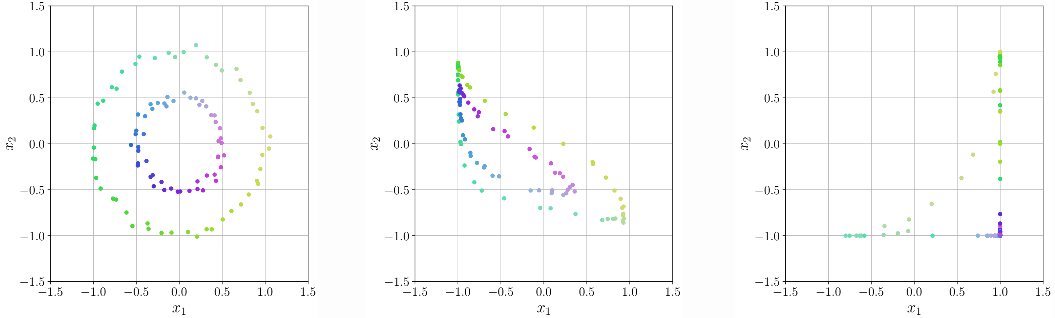
Now we can use a linear classifier on the final hidden features.

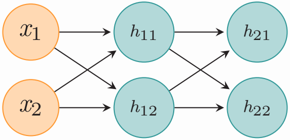




Transforming Features

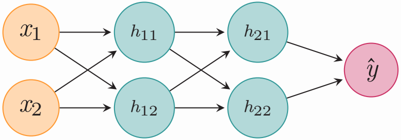
This is the same as previous, but we have colour coded each individual instance.

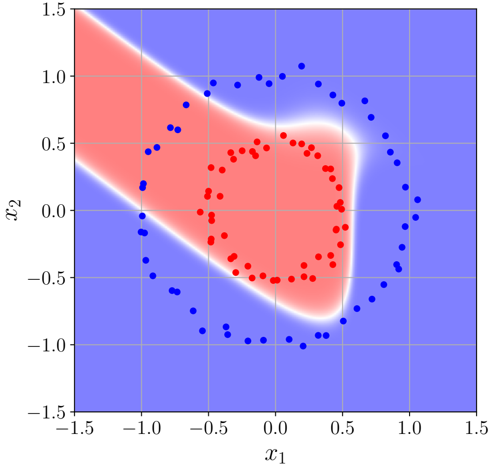




Final Neural Network Predictions

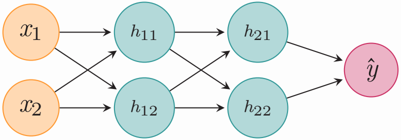
- On the right we see the final network predictions, colour-coded by predicted class.

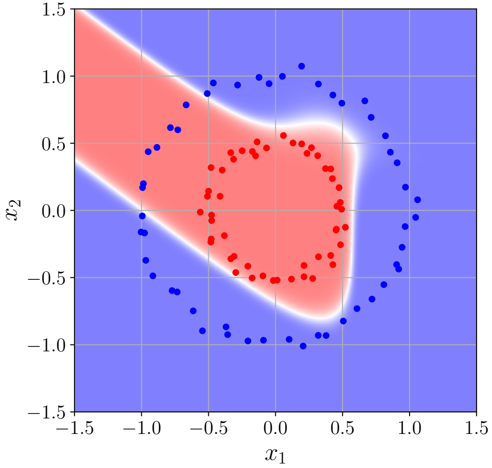




Final Neural Network Predictions

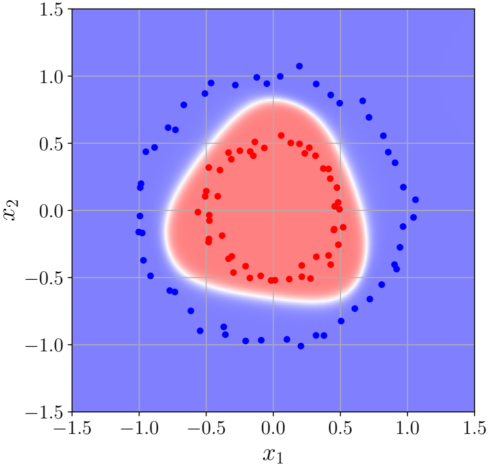
- On the right we see the final network predictions, colour-coded by predicted class.
- Note, in this example we defined a simple, and small, neural network for ease of visualisation.
- Our network did *not* successfully classify *all* the training data.





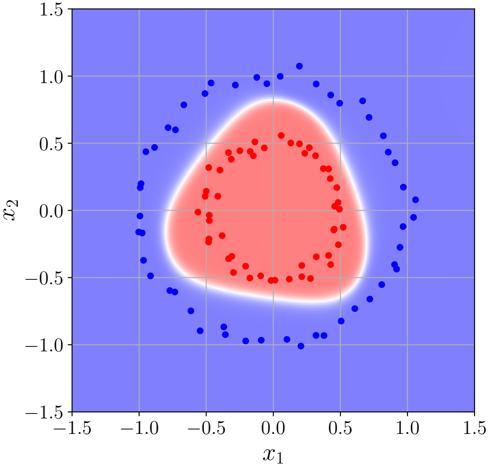
Final Neural Network Predictions

- With a minor change to the structure of the neural network, we can correctly classify the training data.



Final Neural Network Predictions

- With a minor change to the structure of the neural network, we can correctly classify the training data.
- In the example on the right we increased the number of hidden units in the first layer (from two to four).



Training Neural Networks

Neural Network Parameters

- Recall that a multilayer neural network is a nested set of linear functions with non-linear activations.

$$f(\boldsymbol{x}) = f_L(\ldots f_2(f_1(\boldsymbol{x})))$$

Neural Network Parameters

- Recall that a multilayer neural network is a nested set of linear functions with non-linear activations.

$$f(\boldsymbol{x}) = f_L(\ldots f_2(f_1(\boldsymbol{x})))$$

- Each layer f_L has its own weight matrix $\boldsymbol{W}_L$ and bias vector $\boldsymbol{b}_L$.
- The concatenation of these terms form the model weights that need to be learned.

$$\boldsymbol{\theta} = (\,\boldsymbol{W}_1, \boldsymbol{b}_1, \,\boldsymbol{W}_2, \boldsymbol{b}_2, \ldots, \,\boldsymbol{W}_L, \boldsymbol{b}_L)$$

Training Neural Networks

- The task of training a neural network involves finding the best parameters (i.e. weights) for each unit.

Training Neural Networks

- The task of training a neural network involves finding the best parameters (i.e. weights) for each unit.
- We will use a loss function $\mathcal{L}(\boldsymbol{\theta})$ to measure the disagreement between the model prediction $f(\boldsymbol{x})$ and the ground truth target y.

Training Neural Networks

- The task of training a neural network involves finding the best parameters (i.e. weights) for each unit.
- We will use a loss function $\mathcal{L}(\boldsymbol{\theta})$ to measure the disagreement between the model prediction $f(\boldsymbol{x})$ and the ground truth target y.

- During optimisation we will try to find the parameters that minimise the loss.
- We will take the gradient of the loss $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ and use gradient descent to update the parameters.
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}}\mathcal{L}$

Training Multilayer Neural Networks

- Hidden units make optimising the network weights more complicated as we do not have ground truth targets for them.
- Each hidden activity can affect many output units and can therefore have many separate effects on the error.

Backpropagation

- There is a recursive algorithm for computing the derivatives. It uses the **chain rule** by storing some intermediate terms. This is called **backpropagation**.
- We make use of the layered structure of the network to compute the derivatives, heading backwards from the output layer to the inputs.

Backpropagation

- There is a recursive algorithm for computing the derivatives. It uses the **chain rule** by storing some intermediate terms. This is called **backpropagation**.
- We make use of the layered structure of the network to compute the derivatives, heading backwards from the output layer to the inputs.

Backpropagation Algorithm

- Consists of two main steps:
  - A **forward pass**, in which we compute and store the values at all of the hidden units and the network output.

Backpropagation

- There is a recursive algorithm for computing the derivatives. It uses the **chain rule** by storing some intermediate terms. This is called **backpropagation**.
- We make use of the layered structure of the network to compute the derivatives, heading backwards from the output layer to the inputs.

Backpropagation Algorithm

- Consists of two main steps:
  - A **forward pass**, in which we compute and store the values at all of the hidden units and the network output.
  - A **backward pass**, in which we calculate the derivatives of each weight, starting at the end of the network, and reusing the previous computation as we move towards the start.

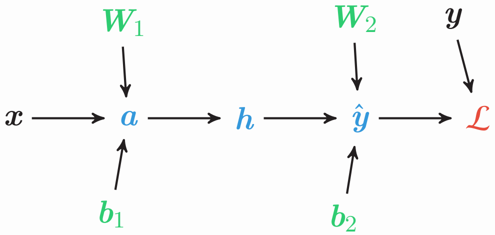
Backpropagation

- We can visualise the computations using a **computation graph**.
- The nodes represent all the inputs and computed quantities, and the edges represent which nodes are computed directly as a function of which other nodes [1].

[1] Example adapted from Ren and MacKay: CSC 411.

Backpropagation

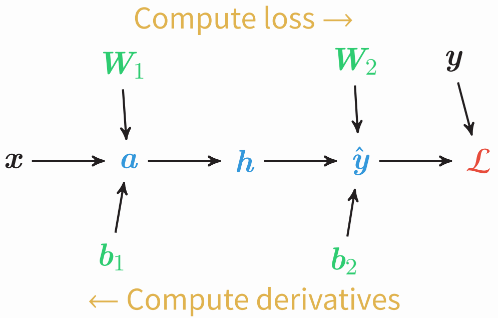
- We can visualise the computations using a **computation graph**.
- The nodes represent all the inputs and computed quantities, and the edges represent which nodes are computed directly as a function of which other nodes [1].



[1] Example adapted from Ren and MacKay: CSC 411.

Backpropagation

- We can visualise the computations using a **computation graph**.
- The nodes represent all the inputs and computed quantities, and the edges represent which nodes are computed directly as a function of which other nodes [1].



[1] Example adapted from Ren and MacKay: CSC 411.

Chain Rule of Calculus

- Suppose we had two functions $u(x)$ and $v(x)$.
- Then $y = u(\,v(x))$ is a function of a function.

Chain Rule of Calculus

- Suppose we had two functions $u(x)$ and $v(x)$.
- Then $y = u(v(x))$ is a function of a function.

- The **chain rule** of calculus gives us a way to expresses the derivative of the composition of two differentiable functions $u(x)$ and $v(x)$ in terms of their derivatives.

Chain Rule of Calculus

- Suppose we had two functions $u(x)$ and $v(x)$.
- Then $y = u(v(x))$ is a function of a function.

- The **chain rule** of calculus gives us a way to expresses the derivative of the composition of two differentiable functions $u(x)$ and $v(x)$ in terms of their derivatives.

- For example, if we substitute $s = v(x)$, thus $y = u(s)$, and

$$\frac{dy}{dx} = \frac{dy}{ds}\frac{ds}{dx}$$

Backpropagation - Forward Pass

x

Backpropagation - Forward Pass

$$\begin{tikzcd} & \boldsymbol{W}_1 \arrow[d] \\ \boldsymbol{x} \arrow[r] & \boldsymbol{a} \\ & \boldsymbol{b}_1 \arrow[u] \end{tikzcd}$$

Forward Pass

$$\boldsymbol{a} = \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1$$

Backpropagation - Forward Pass

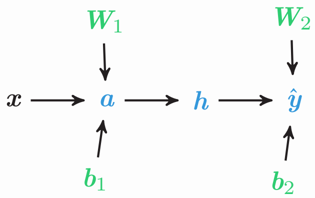
$$
\begin{tikzcd}
 & \boldsymbol{W}_1 \arrow[d] & \\
x \arrow[r] & \boldsymbol{a} \arrow[r] & \boldsymbol{h} \\
 & \boldsymbol{b}_1 \arrow[u] &
\end{tikzcd}
$$

Forward Pass

$$\boldsymbol{a} = \boldsymbol{W}_1 x + \boldsymbol{b}_1$$

$$\boldsymbol{h} = g(\boldsymbol{a})$$

Backpropagation - Forward Pass



Forward Pass

$$
\begin{aligned}
a &= W_1 x + b_1 \\
h &= g(a) \\
\hat{y} &= W_2 h + b_2
\end{aligned}
$$

Backpropagation - Forward Pass

$$\begin{tikzcd}
 & \boldsymbol{W}_1 \arrow[d] & & \boldsymbol{W}_2 \arrow[d] & \boldsymbol{y} \arrow[d] \\
\boldsymbol{x} \arrow[r] & \boldsymbol{a} \arrow[r] & \boldsymbol{h} \arrow[r] & \hat{\boldsymbol{y}} \arrow[r] & \mathcal{L} \\
 & \boldsymbol{b}_1 \arrow[u] & & \boldsymbol{b}_2 \arrow[u] &
\end{tikzcd}$$

Forward Pass

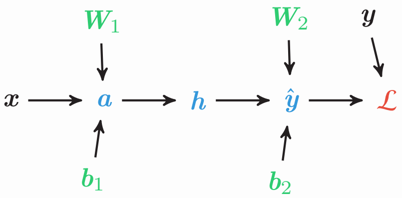
$$\boldsymbol{a} = \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1$$

$$\boldsymbol{h} = g(\boldsymbol{a})$$

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2$$

$$\mathcal{L} = \frac{1}{2} ||\hat{\boldsymbol{y}} - \boldsymbol{y}||^2$$

Backpropagation - Backward Pass



Backward Pass

Forward Pass

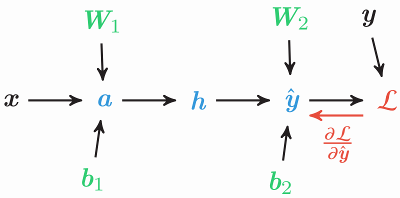
$$\boldsymbol{a} = \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1$$

$$\boldsymbol{h} = g(\boldsymbol{a})$$

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2$$

$$\mathcal{L} = \frac{1}{2} ||\hat{\boldsymbol{y}} - \boldsymbol{y}||^2$$

Backpropagation - Backward Pass



Backward Pass

$$\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} = (\hat{\boldsymbol{y}} - \boldsymbol{y})$$

Forward Pass

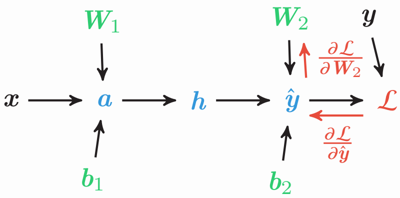
$$\boldsymbol{a} = \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1$$

$$\boldsymbol{h} = g(\boldsymbol{a})$$

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2$$

$$\mathcal{L} = \frac{1}{2} ||\hat{\boldsymbol{y}} - \boldsymbol{y}||^2$$

Backpropagation - Backward Pass



Backward Pass

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = (\hat{y} - y)$$

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W_2}$$

Forward Pass

$$a = W_1 x + b_1$$

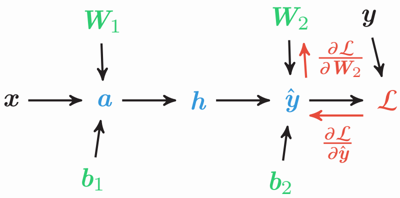
$$h = g(a)$$

$$\hat{y} = W_2 h + b_2$$

$$\mathcal{L} = \frac{1}{2} ||\hat{y} - y||^2$$

THE UNIVERSITY *of* EDINBURGH
informatics

Backpropagation - Backward Pass



Backward Pass

$$\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} = (\hat{\boldsymbol{y}} - \boldsymbol{y})$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{W}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \boldsymbol{h}^\top$$

Forward Pass

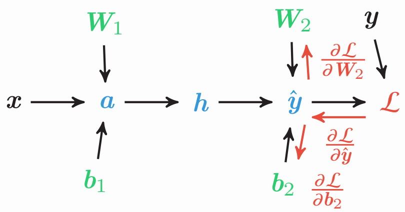
$$\boldsymbol{a} = \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1$$

$$\boldsymbol{h} = g(\boldsymbol{a})$$

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2$$

$$\mathcal{L} = \frac{1}{2} ||\hat{\boldsymbol{y}} - \boldsymbol{y}||^2$$

Backpropagation - Backward Pass



Backward Pass

$$\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} = (\hat{\boldsymbol{y}} - \boldsymbol{y})$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{W}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \boldsymbol{h}^\mathsf{T}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{b}_2}$$

Forward Pass

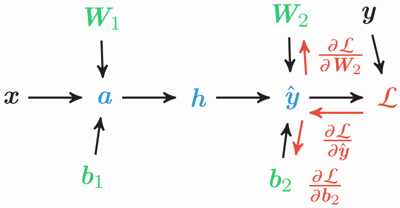
$$\boldsymbol{a} = \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1$$

$$\boldsymbol{h} = g(\boldsymbol{a})$$

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2$$

$$\mathcal{L} = \frac{1}{2} ||\hat{\boldsymbol{y}} - \boldsymbol{y}||^2$$

Backpropagation - Backward Pass



Forward Pass

$$a = W_1 x + b_1$$

$$h = g(a)$$

$$\hat{y} = W_2 h + b_2$$

$$\mathcal{L} = \frac{1}{2}||\hat{y} - y||^2$$

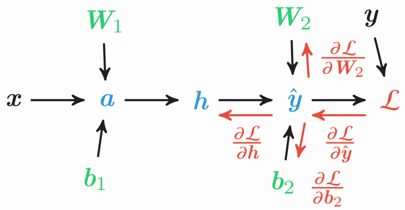
Backward Pass

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = (\hat{y} - y)$$

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} h^{\mathsf{T}}$$

$$\frac{\partial \mathcal{L}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}}$$

Backpropagation - Backward Pass



Forward Pass

$$\boldsymbol{a} = \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1$$

$$\boldsymbol{h} = g(\boldsymbol{a})$$

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2$$

$$\mathcal{L} = \frac{1}{2} ||\hat{\boldsymbol{y}} - \boldsymbol{y}||^2$$

Backward Pass

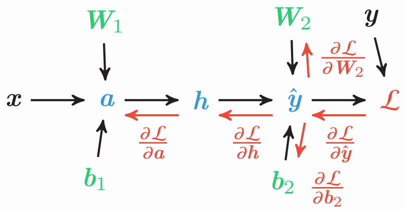
$$\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} = (\hat{\boldsymbol{y}} - \boldsymbol{y})$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{W}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \boldsymbol{h}^\mathsf{T}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{b}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{h}}$$

Backpropagation - Backward Pass



Forward Pass

$$a = W_1 x + b_1$$

$$h = g(a)$$

$$\hat{y} = W_2 h + b_2$$

$$\mathcal{L} = \frac{1}{2} ||\hat{y} - y||^2$$

Backward Pass

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = (\hat{y} - y)$$

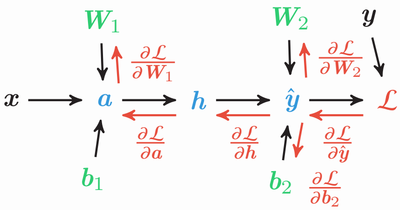
$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} h^\top$$

$$\frac{\partial \mathcal{L}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}}$$

$$\frac{\partial \mathcal{L}}{\partial h} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h}$$

$$\frac{\partial \mathcal{L}}{\partial a} = \frac{\partial \mathcal{L}}{\partial h} \frac{\partial h}{\partial a}$$

Backpropagation - Backward Pass



Forward Pass

$$\boldsymbol{a} = \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1$$

$$\boldsymbol{h} = g(\boldsymbol{a})$$

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2$$

$$\mathcal{L} = \frac{1}{2} ||\hat{\boldsymbol{y}} - \boldsymbol{y}||^2$$

Backward Pass

$$\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} = (\hat{\boldsymbol{y}} - \boldsymbol{y})$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{W}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \boldsymbol{h}^\top$$

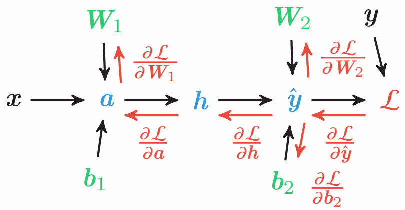
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{b}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{h}}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{a}}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_1} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}_1}$$

Backpropagation - Backward Pass



Forward Pass

$$a = W_1 x + b_1$$

$$h = g(a)$$

$$\hat{y} = W_2 h + b_2$$

$$\mathcal{L} = \frac{1}{2}||\hat{y} - y||^2$$

Backward Pass

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = (\hat{y} - y)$$

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} h^\top$$

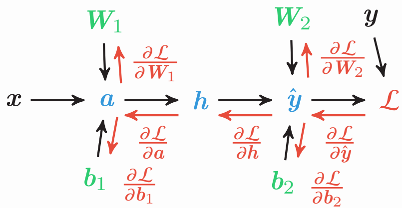
$$\frac{\partial \mathcal{L}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}}$$

$$\frac{\partial \mathcal{L}}{\partial h} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h}$$

$$\frac{\partial \mathcal{L}}{\partial a} = \frac{\partial \mathcal{L}}{\partial h} \frac{\partial h}{\partial a}$$

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial a} \frac{\partial a}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial a} x^\top$$

Backpropagation - Backward Pass



Forward Pass

$$\boldsymbol{a} = \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1$$

$$\boldsymbol{h} = g(\boldsymbol{a})$$

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2$$

$$\mathcal{L} = \frac{1}{2}||\hat{\boldsymbol{y}} - \boldsymbol{y}||^2$$

Backward Pass

$$\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} = (\hat{\boldsymbol{y}} - \boldsymbol{y})$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{W}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \boldsymbol{h}^\top$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{b}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}}$$

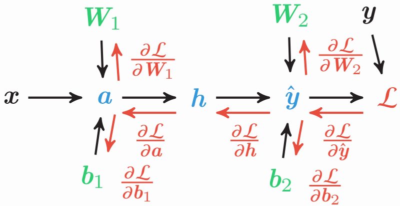
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{h}}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{a}}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_1} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}_1} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}} \boldsymbol{x}^\top$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_1} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{b}_1}$$

Backpropagation - Backward Pass



Forward Pass

$$a = W_1 x + b_1$$

$$h = g(a)$$

$$\hat{y} = W_2 h + b_2$$

$$\mathcal{L} = \frac{1}{2}||\hat{y} - y||^2$$

Backward Pass

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = (\hat{y} - y)$$

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} h^\top$$

$$\frac{\partial \mathcal{L}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}}$$

$$\frac{\partial \mathcal{L}}{\partial h} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h}$$

$$\frac{\partial \mathcal{L}}{\partial a} = \frac{\partial \mathcal{L}}{\partial h} \frac{\partial h}{\partial a}$$

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial a} \frac{\partial a}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial a} x^\top$$

$$\frac{\partial \mathcal{L}}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial a} \frac{\partial a}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial a}$$

Convergence of Neural Networks

- For *logistic regression*, the loss function is conveniently convex. A convex function has just one minimum.

Convergence of Neural Networks

- For *logistic regression*, the loss function is conveniently convex. A convex function has just one minimum.
- Multilayer *neural networks* are **non-convex**, and gradient descent may get stuck in local minima during training and never find the global optimum.
- In practice this is not necessarily an issue and we can still apply gradient-based methods and can obtain good solutions for many practical problems of interest.

Hyperparameters

Network Structure

- There are several elements of the network that you can change e.g.
  - The number of hidden layers.
  - The number of units in each hidden layer.
  - The type of non-linear activation function e.g. ReLU, sigmoid, ...

Hyperparameters

Network Structure

- There are several elements of the network that you can change e.g.
  - The number of hidden layers.
  - The number of units in each hidden layer.
  - The type of non-linear activation function e.g. ReLU, sigmoid, ...

Training Schedule

- There also are several aspects of the training procedure that can be changed e.g.
  - The learning rate.
  - The type of optimiser e.g. standard gradient descent, ...
  - How the weights are initialised.
  - When to stop training.

Automatic Differentiation

- The **backpropagation algorithm**, which can be used to compute the gradient of a loss function applied to the output of the network wrt the parameters in each layer.
- This gradient can then be used with any gradient-based optimisation, e.g. gradient descent.

Automatic Differentiation

- The **backpropagation algorithm**, which can be used to compute the gradient of a loss function applied to the output of the network wrt the parameters in each layer.
- This gradient can then be used with any gradient-based optimisation, e.g. gradient descent.

- Manually computing these gradients for anything but small toy problems is too time consuming.
- Instead, we can make use **automatic differentiation** (or **autodiff**). This is a set of automatic techniques to evaluate the derivative of a function.

Automatic Differentiation

- The **backpropagation algorithm**, which can be used to compute the gradient of a loss function applied to the output of the network wrt the parameters in each layer.
- This gradient can then be used with any gradient-based optimisation, e.g. gradient descent.

- Manually computing these gradients for anything but small toy problems is too time consuming.
- Instead, we can make use **automatic differentiation** (or **autodiff**). This is a set of automatic techniques to evaluate the derivative of a function.

- Many machine learning frameworks have autodiff functionality built in.

Automatic Differentiation Example

The following is an example of using autodiff for binary logistic regression.

```python
import jax.numpy as jnp
from jax import grad, nn

# Define our loss function
def nll_loss(X, y, w):
    pred = nn.sigmoid(X@w)
    loss_pos = (y==1)*jnp.log(pred)
    loss_neg = (y==0)*jnp.log(1.0 - pred)
    loss = -(loss_pos + loss_neg).mean()
    return loss


# Define our dataset, which has 3 instances
# We have already appended a 1.0 to each row of X
X = jnp.array([[1.0, 0.5,-0.35],
               [1.0, -0.1, 0.1],
               [1.0, -1.2, 1.0]])
y = jnp.array([0.0, 0.0, 1.0])

# This is our initial weight vector w
w = jnp.array([0.0, -1.0, 1.0])

```

Automatic Differentiation Example

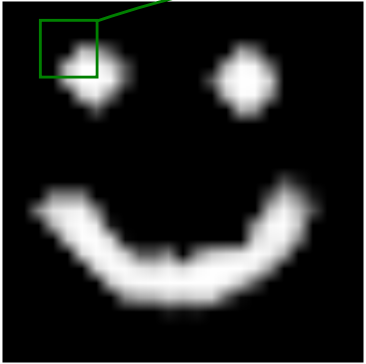
The following is an example of using autodiff for binary logistic regression.

```
import jax.numpy as jnp
from jax import grad, nn

# Define our loss function
def nll_loss(X, y, w):
    pred = nn.sigmoid(X@w)
    loss_pos = (y==1)*jnp.log(pred)
    loss_neg = (y==0)*jnp.log(1.0 - pred)
    loss = -(loss_pos + loss_neg).mean()
    return loss


# Define our dataset, which has 3 instances
# We have already appended a 1.0 to each row of X
X = jnp.array([[1.0, 0.5,-0.35],
               [1.0, -0.1, 0.1],
               [1.0, -1.2, 1.0]])
y = jnp.array([0.0, 0.0, 1.0])

# This is our initial weight vector w
w = jnp.array([0.0, -1.0, 1.0])

# (i) Compute the gradient manually
# Here we use the derived expression
manual_grad = (nn.sigmoid(X@w) - y)@X
manual_grad *= (1.0/X.shape[0])
print(’Manual gradient’, jnp.round(manual_grad, 3))


# (ii) Compute the gradient automatically
# Evaluate the loss and compute the gradient
loss = nll_loss(X, y, w)
w_grad = grad(nll_loss, (2))(X, y, w)
print(’Auto gradient  ’, jnp.round(w_grad, 3))


# We can take one step of gradient descent
learning_rate = 3.0
w_update = w - learning_rate*w_grad
```

Alternative Network Architectures

Images as Tensors

- We can represent images as **matrices**, where each entry stores the intensity value of a given pixel.



0	0	0
0	.7	.7
0	.7	1

Issues with Fully Connected Neural Networks

- Fully connected networks with high-dimensional inputs have a lot of model weights.
- This results in a very large number of model weights that have to be learned.

Issues with Fully Connected Neural Networks

- Fully connected networks with high-dimensional inputs have a lot of model weights.
- This results in a very large number of model weights that have to be learned.

- For example, if our input was an image of size 100×100, this would require $10,000$ weights for *each* hidden unit in the first layer.

Shift Invariance

- Fully connected networks are sensitive to the position of the signal of interest in an input image.

0	1	0	0
1	1	1	0
0	1	0	0
0	0	0	0

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$\mathbf{x}_1$

Shift Invariance

- Fully connected networks are sensitive to the position of the signal of interest in an input image.

$$\begin{array}{|c|c|c|c|}\hline 0 & 1 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array} \quad \sum \underset{\mathbf{x}_1}{\begin{array}{|c|}\hline 0\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 1\\ \hline 1\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline \end{array}} \times \underset{\mathbf{w}}{\begin{array}{|c|}\hline 0\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 1\\ \hline 1\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline \end{array}} = \mathbf{5}$$

Shift Invariance

- Fully connected networks are sensitive to the position of the signal of interest in an input image.

$$
\begin{array}{|c|c|c|c|} \hline 0 & 1 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}
\quad \sum
\underbrace{\begin{array}{|c|} \hline 0\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 1\\ \hline 1\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline \end{array}}_{\mathbf{x}_1}
\times
\underbrace{\begin{array}{|c|} \hline 0\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 1\\ \hline 1\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline \end{array}}_{\mathbf{w}}
= \mathbf{5}
\qquad
\begin{array}{|c|c|c|c|} \hline 0 & 0 & 1 & 0 \\ \hline 0 & 1 & 1 & 1 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}
\quad \sum
\underbrace{\begin{array}{|c|} \hline 0\\ \hline 0\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 1\\ \hline 1\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline \end{array}}_{\mathbf{x}_2}
\times
\underbrace{\begin{array}{|c|} \hline 0\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 1\\ \hline 1\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 1\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline 0\\ \hline \end{array}}_{\mathbf{w}}
= \mathbf{2}
$$

Convolutional Filters

- Constrain each hidden unit to extract features by **sharing** weights across the input.

Convolutional Filters

- Constrain each hidden unit to extract features by **sharing** weights across the input.
- For an image $\boldsymbol{X}$ and $K \times K$ weight matrix $\boldsymbol{W}$ (i.e. a filter) we compute the outputs as

$$h_{ij} = g\left(\sum_{m=1}^{K}\sum_{n=1}^{K} w_{m,n} x_{i+m,j+n} + b\right)$$

Convolutional Filters

- Constrain each hidden unit to extract features by **sharing** weights across the input.
- For an image $\boldsymbol{X}$ and $K \times K$ weight matrix $\boldsymbol{W}$ (i.e. a filter) we compute the outputs as

$$h_{ij} = g\left(\sum_{m=1}^{K}\sum_{n=1}^{K} w_{m,n} x_{i+m,j+n} + b\right)$$

- The output is a **feature map**, where each entry h_{ij} is the local response of the filter convolved with the image at that location.
- Multiple weight matrices can be used to produce multiple feature maps.

Convolution

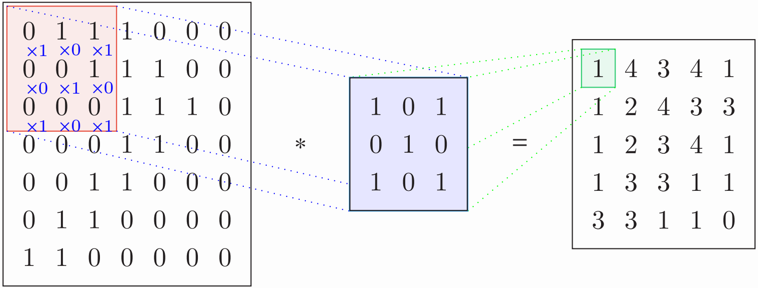
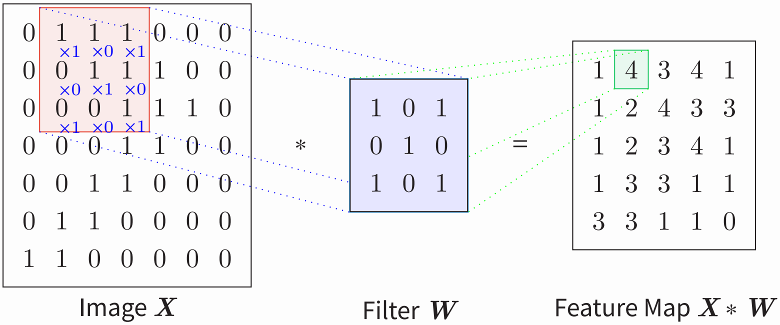
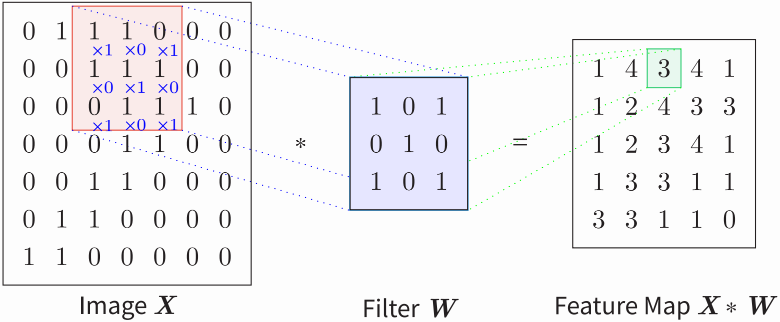


Image $\boldsymbol{X}$ Filter $\boldsymbol{W}$ Feature Map $\boldsymbol{X} * \boldsymbol{W}$

Convolution



Convolution



Convolution

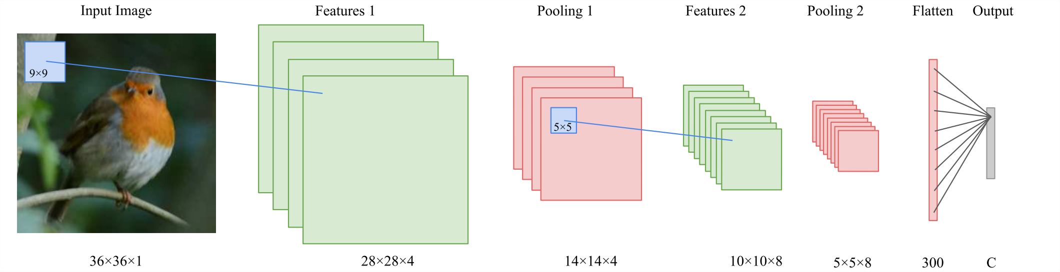
$$\underbrace{\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{Image } \boldsymbol{X}} * \underbrace{\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}}_{\text{Filter } \boldsymbol{W}} = \underbrace{\begin{bmatrix} 1 & 4 & 3 & 4 & 1 \\ 1 & 2 & 4 & 3 & 3 \\ 1 & 2 & 3 & 4 & 1 \\ 1 & 3 & 3 & 1 & 1 \\ 3 & 3 & 1 & 1 & 0 \end{bmatrix}}_{\text{Feature Map } \boldsymbol{X} * \boldsymbol{W}}$$

Convolutional Neural Network - Example

- A Convolutional Neural Network (CNN) consists of *learnable* convolutional filters and *non-learnable* pooling layers.
- The pooling layers reduce the spatial dimensionality of the feature maps.
- For classification, at the output of the network, we have a fully connected layer which predicts one of C classes.

Convolutional Neural Network - Example

- A Convolutional Neural Network (CNN) consists of *learnable* convolutional filters and *non-learnable* pooling layers.
- The pooling layers reduce the spatial dimensionality of the feature maps.
- For classification, at the output of the network, we have a fully connected layer which predicts one of C classes.



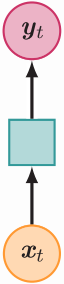
Recurrent Neural Networks

- A model for sequence data (e.g. time series).
- Different network architectures and recurrent units exist, e.g. long short-term-memories (LSTMs).

Recurrent Neural Networks

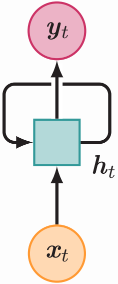
- A model for sequence data (e.g. time series).
- Different network architectures and recurrent units exist, e.g. long short-term-memories (LSTMs).
- In a RNN, each input is processed sequentially, one item at a time.
- Past information is retained through past hidden states.

Recurrent Neural Networks



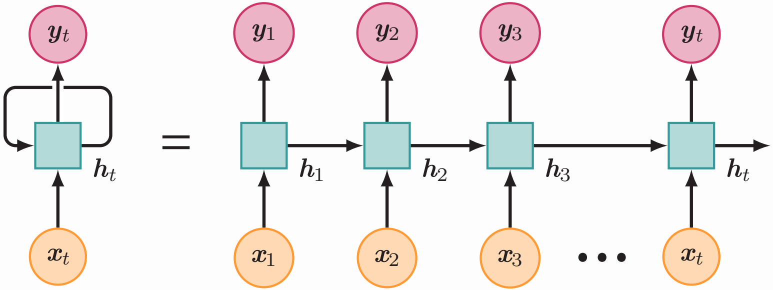
Recurrent Neural Networks

- In RNNs, the outputs $\boldsymbol{y}_t$ are a function of the current input $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{h}_{t-1}$.



Recurrent Neural Networks

- In RNNs, the outputs $\boldsymbol{y}_t$ are a function of the current input $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{h}_{t-1}$.



Transformers

- Alternative, and more recent, approach for modelling sequential data.
- Unlike RNNs, Transformers process the entire input all at once.
  - Thus training can be performed in parallel.
  - They are also less susceptible to 'forgetting' information from the past, i.e. better suited to capture *long-range* dependencies.

Transformers

- Alternative, and more recent, approach for modelling sequential data.
- Unlike RNNs, Transformers process the entire input all at once.
    - Thus training can be performed in parallel.
    - They are also less susceptible to ‘forgetting’ information from the past, i.e. better suited to capture *long-range* dependencies.

- Transformers have a special type of unit called a **self-attention** unit. This is used to compute similarity scores between inputs in the input sequence.

Transformers

- Alternative, and more recent, approach for modelling sequential data.
- Unlike RNNs, Transformers process the entire input all at once.
  - Thus training can be performed in parallel.
  - They are also less susceptible to 'forgetting' information from the past, i.e. better suited to capture *long-range* dependencies.

- Transformers have a special type of unit called a **self-attention** unit. This is used to compute similarity scores between inputs in the input sequence.

- They can also be applied to other data types, e.g. images.

Summary

- Artificial neural networks are a powerful non-linear modelling tool for classification and regression.
- They are *not* biologically plausible models.

Summary

- Artificial neural networks are a powerful non-linear modelling tool for classification and regression.
- They are *not* biologically plausible models.
- The output of the hidden units are a new representation of the original input data. This can be interpreted as learned features.
- Training makes use of the backpropagation algorithm to compute derivatives.

Summary

- Artificial neural networks are a powerful non-linear modelling tool for classification and regression.
- They are *not* biologically plausible models.
- The output of the hidden units are a new representation of the original input data. This can be interpreted as learned features.
- Training makes use of the backpropagation algorithm to compute derivatives.
- Beyond standard fully connected networks, alternative architectures exist for learning from structured input data (e.g. images, audio, text, ...).