AMME4710: COMPUTER VISION AND IMAGE PROCESSING
WEEK 8

Dr. Mitch Bryson

School of Aerospace, Mechanical and Mechatronic Engineering, University of Sydney

Last Week

- Introduction to Machine Learning and Image Classification
  - Data-driven and Supervised Machine Learning approaches to Classification
  - Image Feature Extraction
  - Classification Algorithms: k-NN, Decision Trees, SVMs
  - Classifier Performance Metrics and Validation

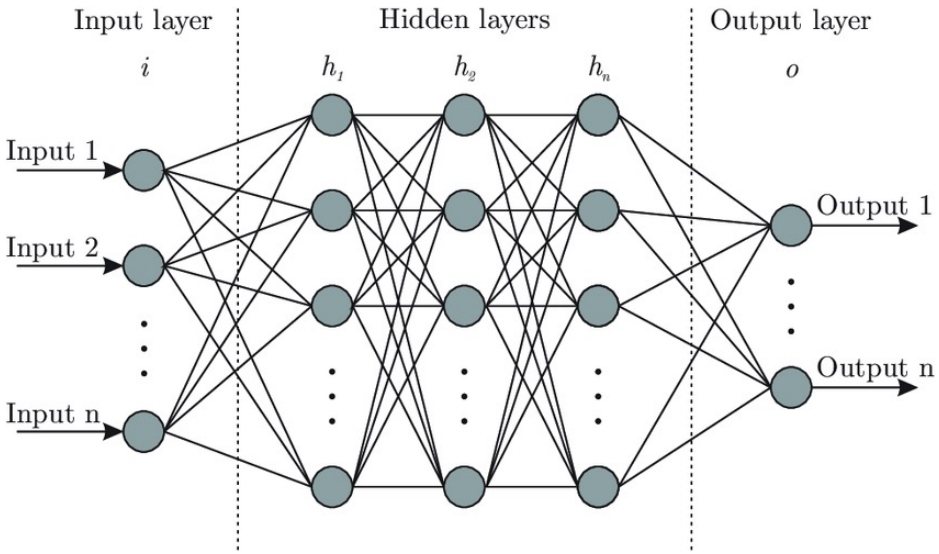
This Week's Lecture

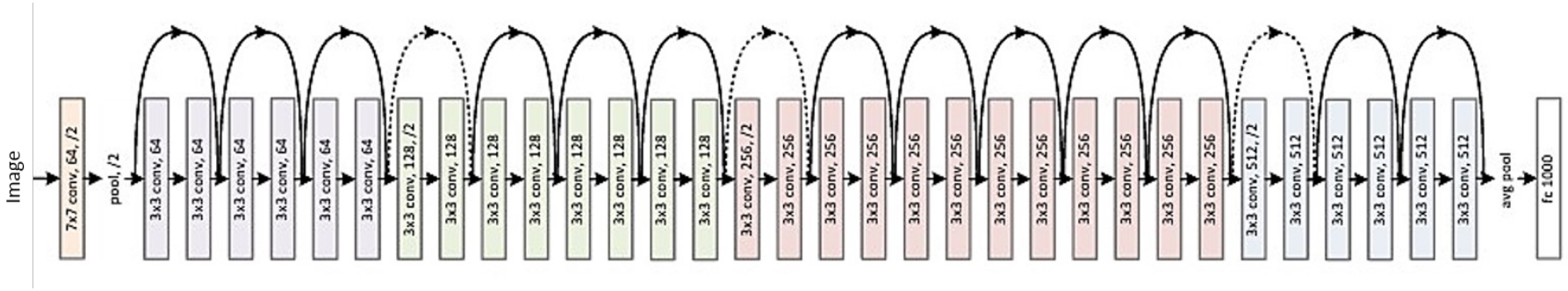
- Introduction to Deep Learning and Image Classification

- Learning Objectives:
  - To gain an understanding of deep learning techniques based on neural networks for image classification

Deep Learning

- "Deep Learning" is a form of machine learning that involves task-performing computational models that are built from multiple "layers" of processing
- Encompasses several different techniques and algorithms, but has become synonymous with the use of Neural Networks

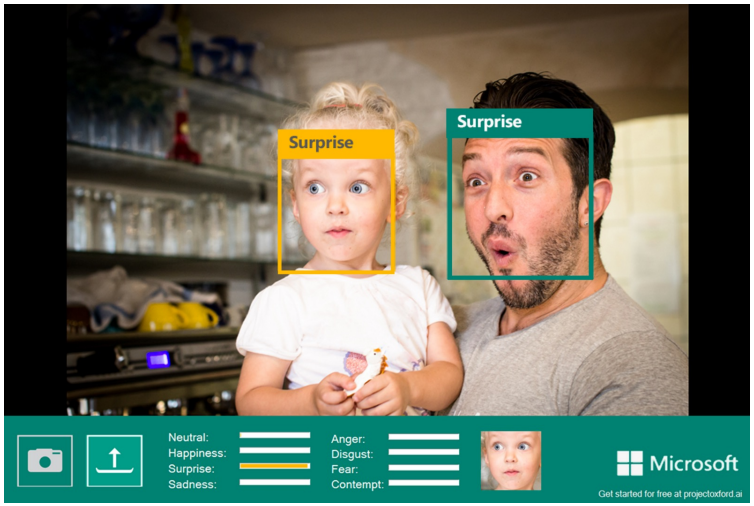




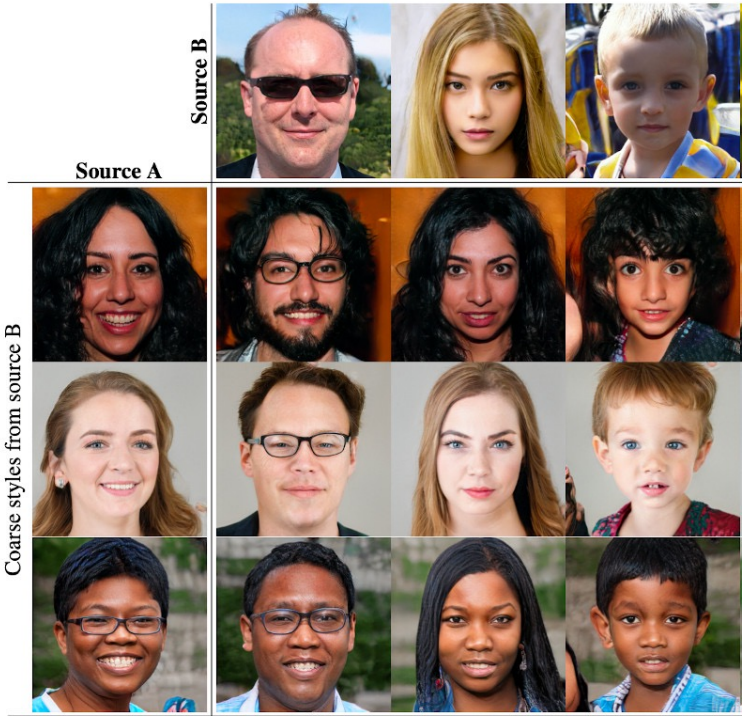
ResNet Architecture: He et. al., "Deep Residual Learning for Image Recognition", CVPR, 2015.

Applications

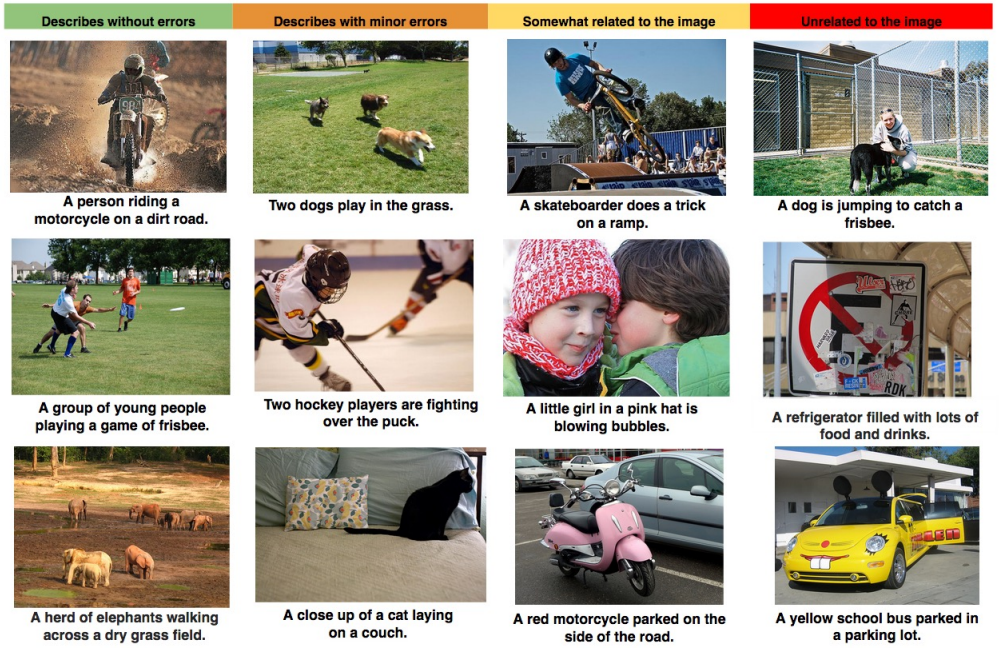
https://deepart.io



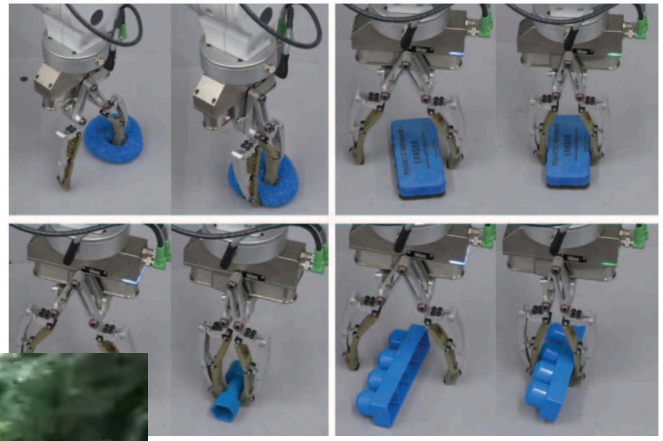
https://azure.microsoft.com/en-au/services/cognitive-services/face/

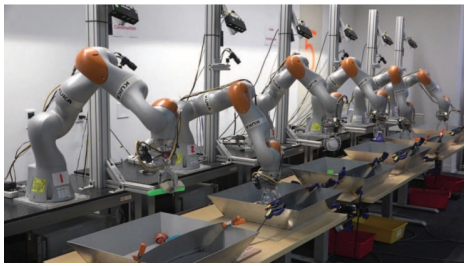


T. Karras et. al., "A Style-Based Generator Architecture for Generative Adversarial Networks", CVPR, 2019.



End-to-end architectures for robotic control using vision



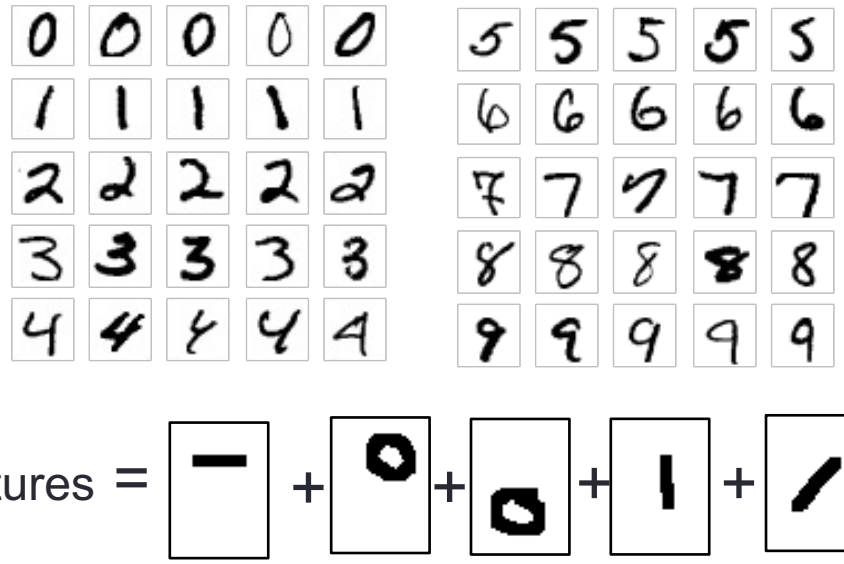


S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection", IJRR, 2017

N. Smolyanskiy, A. Kamenev, J. Smith, S. Birchfield, "Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness", IROS, 2017

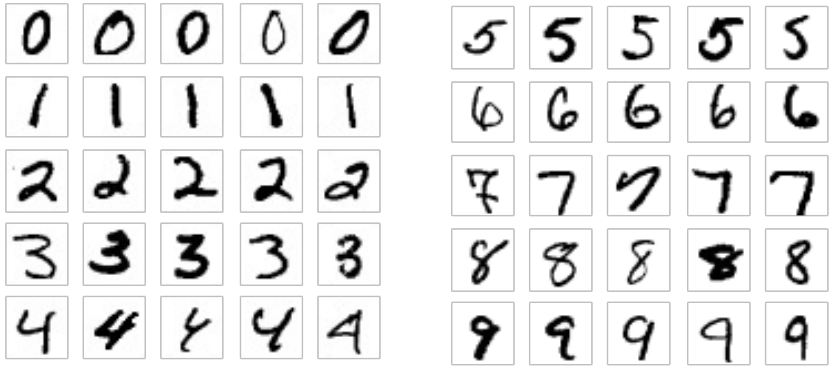
Deep Learning vs. Traditional Learning in Computer Vision

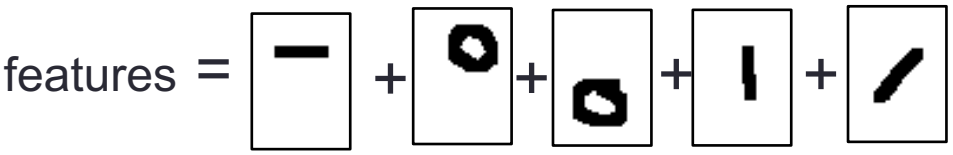
- Although there is no strict distinction between supervised machine learning algorithms explored in Week 7 and Deep Learning, there is an important difference in thinking: the role of features
- Defining and specifying ”designed” features can result in robust model performance for simple problems, but becomes more difficult for complex problems



Deep Learning vs. Traditional Learning in Computer Vision

- Although there is no strict distinction between supervised machine learning algorithms explored in Week 7 and Deep Learning, there is an important difference in thinking: the role of features
- Defining and specifying ”designed” features can result in robust model performance for simple problems, but becomes more difficult for complex problems

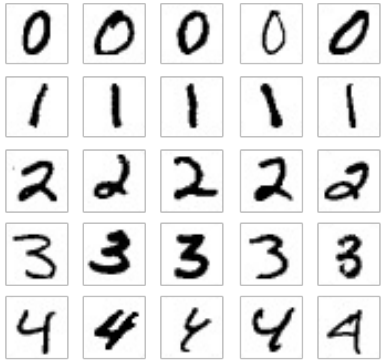


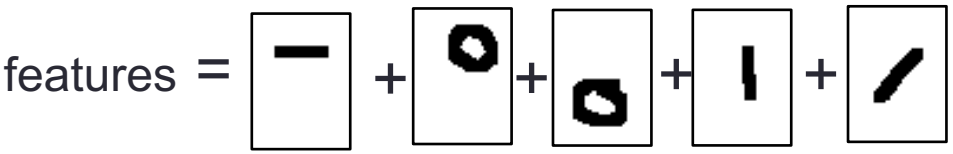


= ???

Deep Learning vs. Traditional Learning in Computer Vision

- Although there is no strict distinction between supervised machine learning algorithms explored in Week 7 and Deep Learning, there is an important difference in thinking: the role of features
- Defining and specifying ”designed” features can result in robust model performance for simple problems, but becomes more difficult for complex problems
- What if we let the machine learnt algorithm also learn the best features and feature space too? …. We are going to need a lot of data

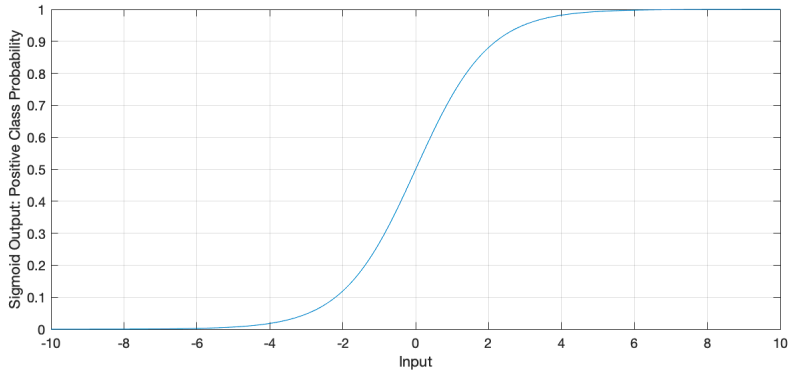




= ???

Logistic Regression

Sigmoid function: $y = \frac{1}{1 + e^{-x}}$

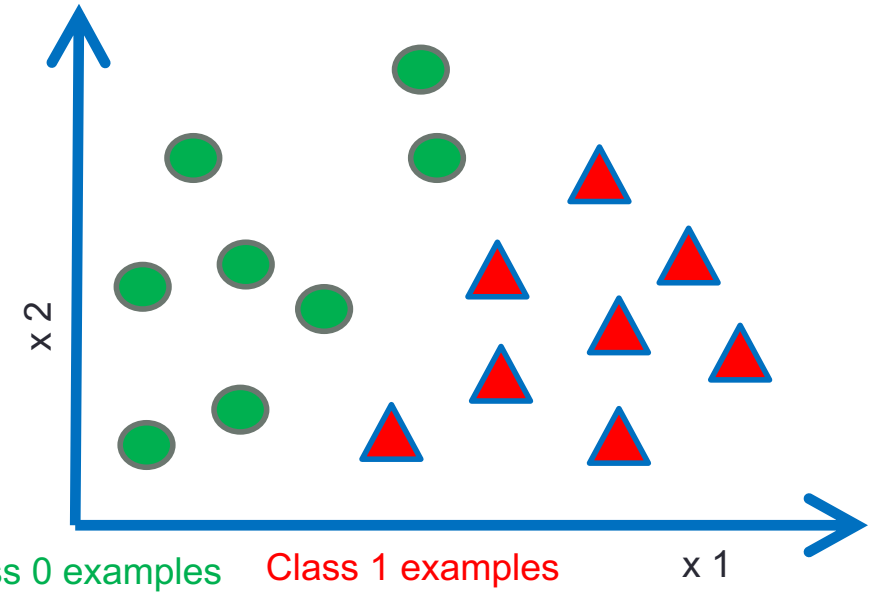


- Logistic regression is a process for fitting a model that predicts binary discrete output as a function of continuous input variables
- Logistic regression computes a linear combination of the input **x** through the parameters and passes this scalar through the sigmoid function
- This effectively splits the data using a hyperplane in N dimensions and maps the distance of a point to the plane to [0,1]

$$\mathbf{x} = [1, x_1, x_2, \ldots, x_N]^T$$

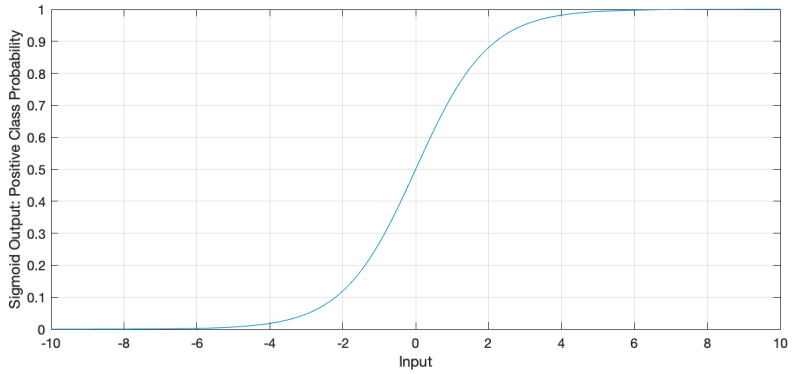
$$y = h(x) = \frac{1}{1+e^{-\theta^T x}} \qquad \theta = [\theta_0, \theta_1, \theta_2, \ldots, \theta_N]^T$$

$$\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_N x_N$$



Logistic Regression

Sigmoid function: $y = \frac{1}{1 + e^{-x}}$

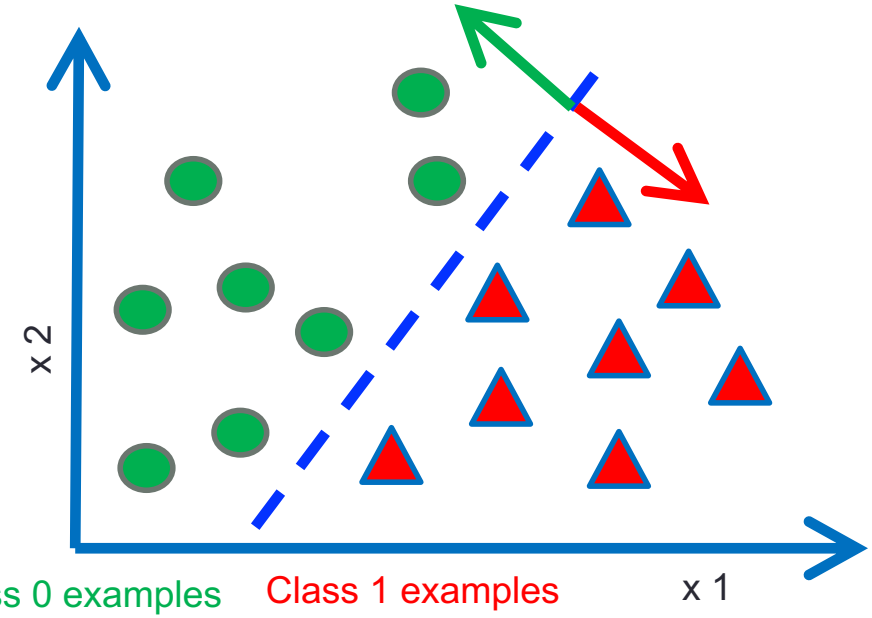


- Logistic regression is a process for fitting a model that predicts binary discrete output as a function of continuous input variables
- Logistic regression computes a linear combination of the input **x** through the parameters and passes this scalar through the sigmoid function
- This effectively splits the data using a hyperplane in N dimensions and maps the distance of a point to the plane to [0,1]

$$\mathbf{x} = [1, x_1, x_2, \ldots, x_N]^T$$

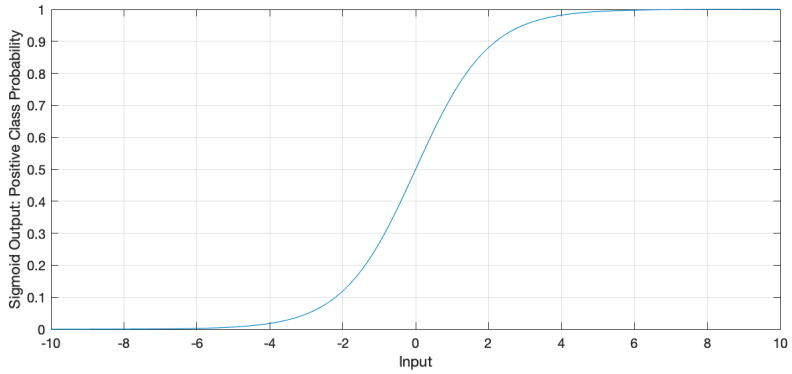
$$y = h(x) = \frac{1}{1+e^{-\theta^T x}} \qquad \theta = [\theta_0, \theta_1, \theta_2, \ldots, \theta_N]^T$$

$$\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_N x_N$$



Logistic Regression

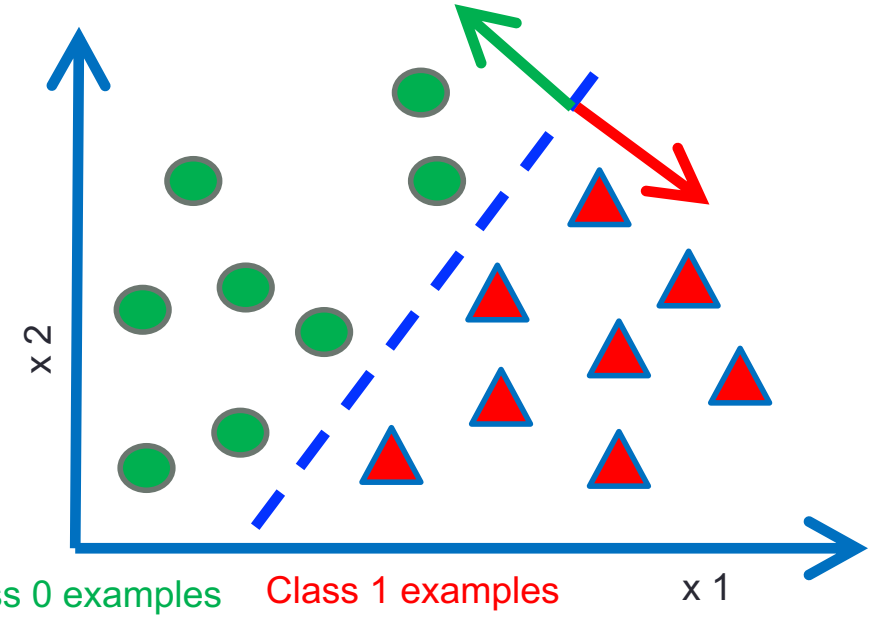
Sigmoid function: $y = \frac{1}{1+e^{-x}}$



- Logistic regression is a process for fitting a model that predicts binary discrete output as a function of continuous input variables
- Logistic regression computes a linear combination of the input **x** through the parameters and passes this scalar through the sigmoid function
- This effectively splits the data using a hyperplane in N dimensions and maps the distance of a point to the plane to [0,1]

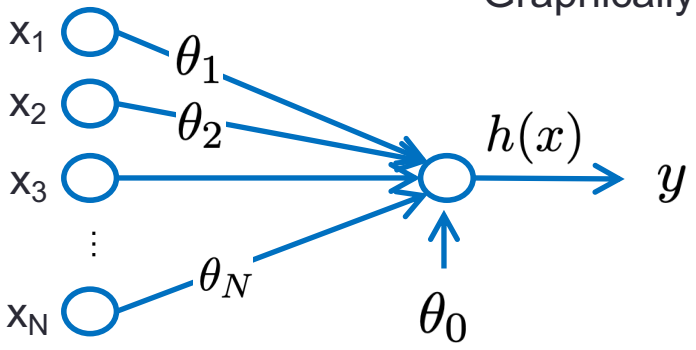
$$\mathbf{x} = [1, x_1, x_2, \ldots, x_N]^T$$

$$y = h(x) = \frac{1}{1+e^{-\theta^T x}} \qquad \theta = [\theta_0, \theta_1, \theta_2, \ldots, \theta_N]^T$$



$$\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_N x_N$$

- Graphically:



Cost Function

- Training a logistic regression model involves computing the parameters $\theta_0, \theta_1, \theta_2, \ldots, \theta_N$ that best optimize the performance of the classifier on training data
- Intuitively, training sample "cost" for a given set of parameters should be high if the predicted output does not match with the true label

Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

Cost Function

- Training a logistic regression model involves computing the parameters $\theta_0, \theta_1, \theta_2, \ldots, \theta_N$ that best optimize the performance of the classifier on training data
- Intuitively, training sample "cost" for a given set of parameters should be high if the predicted output does not match with the true label

Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

Optimal classifier parameters

Cost Function

- Training a logistic regression model involves computing the parameters $\theta_0, \theta_1, \theta_2, \ldots, \theta_N$ that best optimize the performance of the classifier on training data
- Intuitively, training sample "cost" for a given set of parameters should be high if the predicted output does not match with the true label

Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

Total optimization cost

Cost Function

- Training a logistic regression model involves computing the parameters $\theta_0, \theta_1, \theta_2, \ldots, \theta_N$ that best optimize the performance of the classifier on training data
- Intuitively, training sample "cost" for a given set of parameters should be high if the predicted output does not match with the true label

Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

Cost for training sample i

Cost Function

- Training a logistic regression model involves computing the parameters $\theta_0, \theta_1, \theta_2, \ldots, \theta_N$ that best optimize the performance of the classifier on training data
- Intuitively, training sample "cost" for a given set of parameters should be high if the predicted output does not match with the true label
- One way to map this cost is via the binary cross entropy cost function:

Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

$$\begin{aligned} J(\theta)^i &= -log(h(x^i)) && \text{if } y^i = 1 \\ &= -log(1 - h(x^i)) && \text{if } y^i = 0 \end{aligned}$$

Cost Function

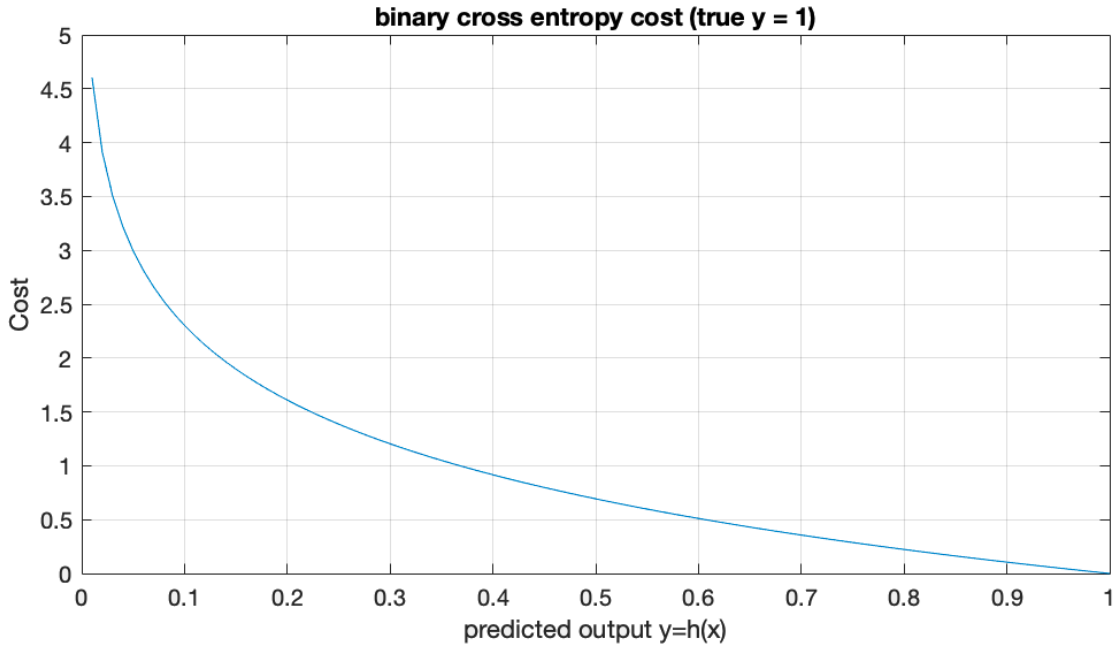
- Training a logistic regression model involves computing the parameters $\theta_0, \theta_1, \theta_2, \ldots, \theta_N$ that best optimize the performance of the classifier on training data
- Intuitively, training sample "cost" for a given set of parameters should be high if the predicted output does not match with the true label
- One way to map this cost is via the binary cross entropy cost function:

Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

$$J(\theta)^i = -log(h(x^i)) \quad \text{if } y^i = 1$$
$$= -log(1 - h(x^i)) \quad \text{if } y^i = 0$$



Cost Function

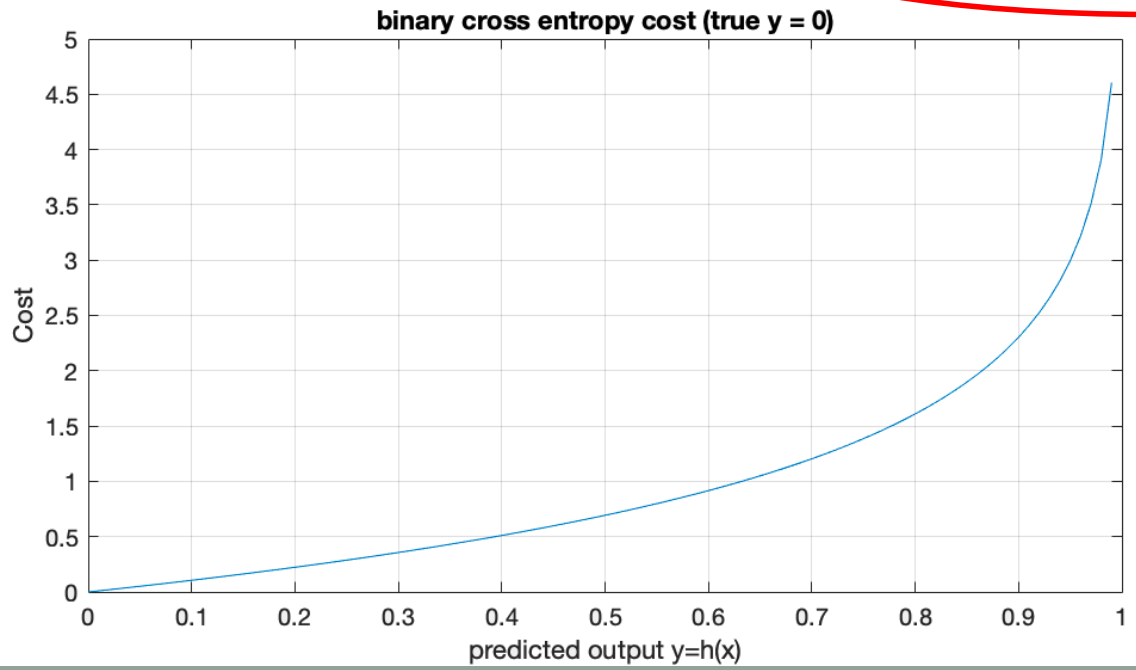
- Training a logistic regression model involves computing the parameters $\theta_0, \theta_1, \theta_2, \ldots, \theta_N$ that best optimize the performance of the classifier on training data
- Intuitively, training sample "cost" for a given set of parameters should be high if the predicted output does not match with the true label
- One way to map this cost is via the binary cross entropy cost function:

Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

$$J(\theta)^i = -log(h(x^i)) \quad \text{if } y^i = 1$$
$$= -log(1 - h(x^i)) \text{ if } y^i = 0$$



Cost Function

Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

- Training a logistic regression model involves computing the parameters $\theta_0, \theta_1, \theta_2, \ldots, \theta_N$ that best optimize the performance of the classifier on training data

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

- Intuitively, training sample "cost" for a given set of parameters should be high if the predicted output does not match with the true label
- One way to map this cost is via the binary cross entropy cost function:

$$\begin{aligned} J(\theta)^i &= -log(h(x^i)) \qquad \text{if } y^i = 1 \\ &= -log(1 - h(x^i)) \quad \text{if } y^i = 0 \end{aligned}$$

- Which is equivalent to:

$$J(\theta)^i = -[y^i log(h(x^i)) + (1 - y^i) log(1 - h(x^i))]$$

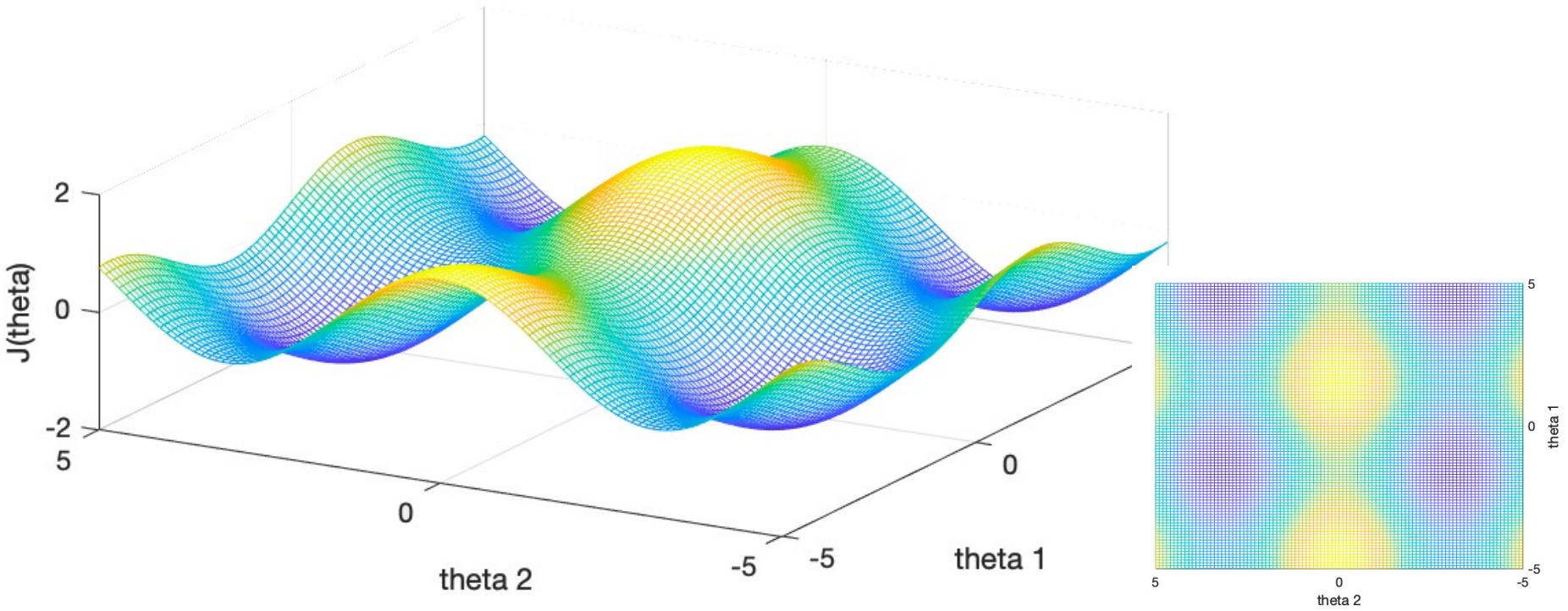
Optimisation

- The resulting cost function is non-linear and requires a general optimization method to find a minima

Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$



Optimisation

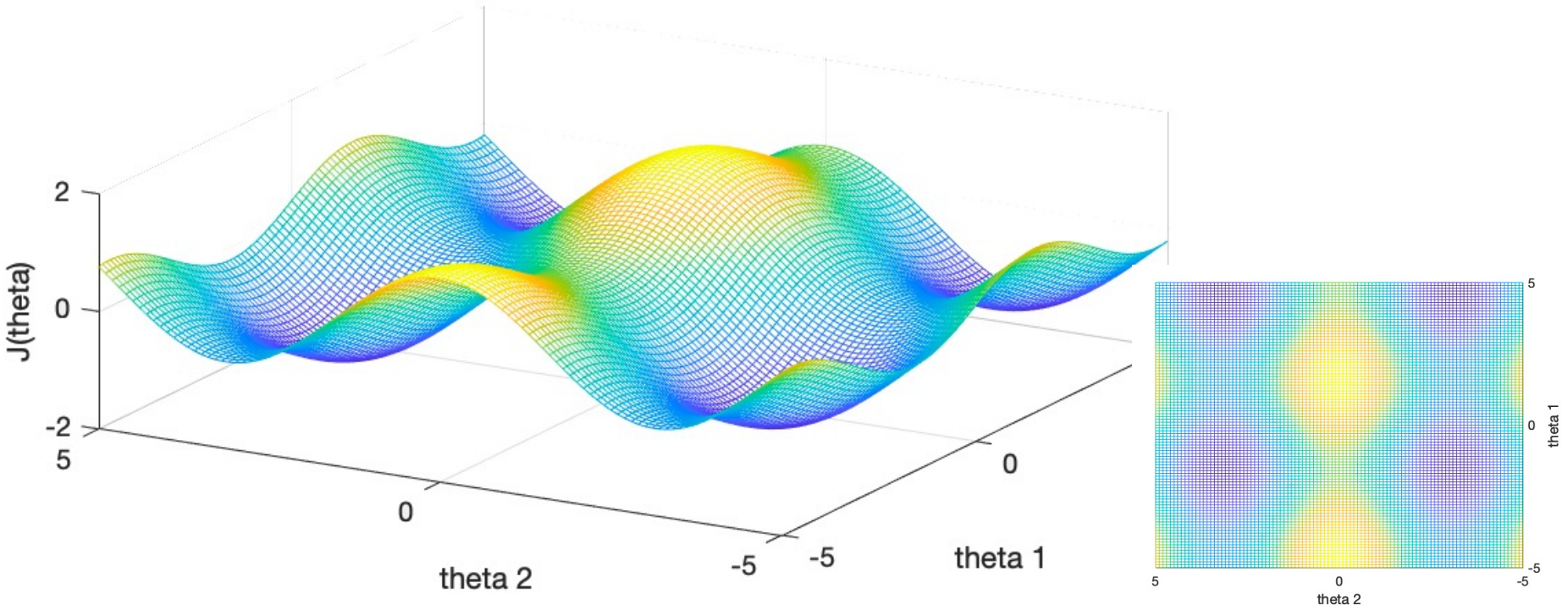
- The resulting cost function is non-linear and requires a general optimization method to find a minima
- **Gradient Descent** is an optimization strategy that recursively moves the parameter vector along the negative gradient direction of the cost function J() until convergence

Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

$$\theta^{n+1} = \theta^n - \alpha \nabla J(\theta^n)$$



Optimisation

- The resulting cost function is non-linear and requires a general optimization method to find a minima
- **Gradient Descent** is an optimization strategy that recursively moves the parameter vector along the negative gradient direction of the cost function J() until convergence

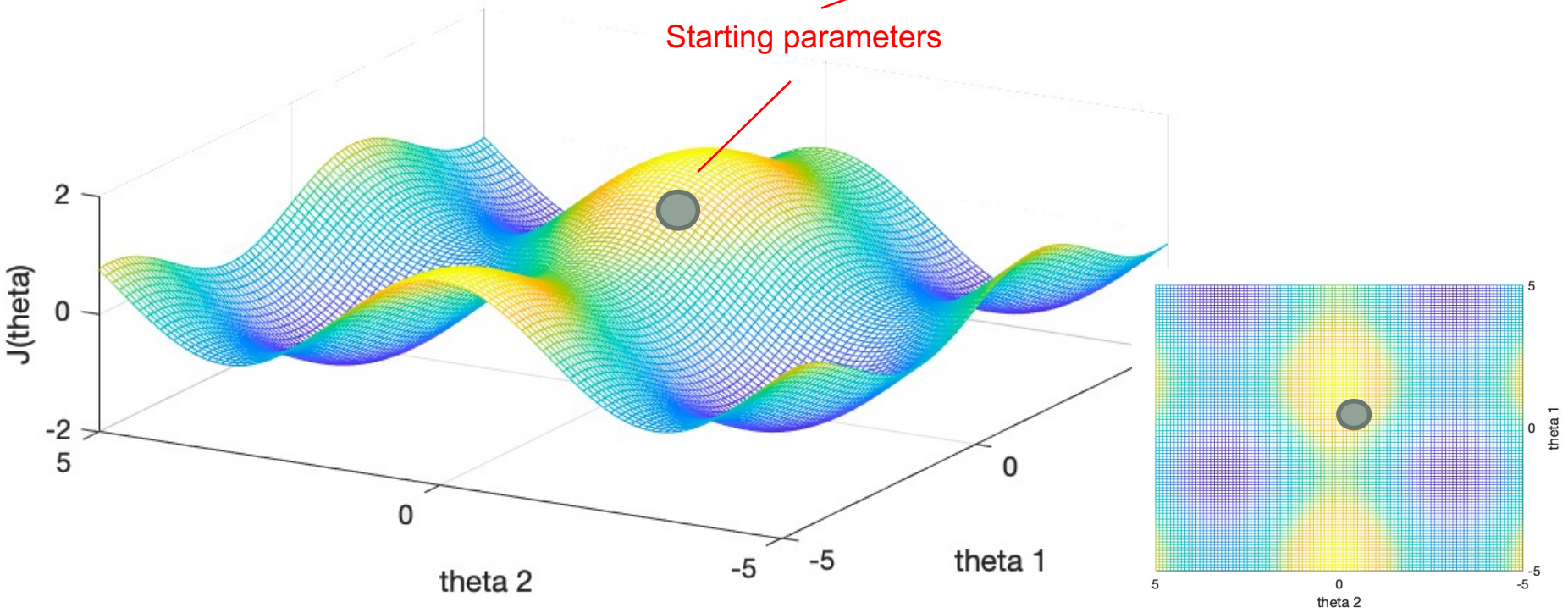
Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

$$\theta^{n+1} = \theta^n - \alpha \nabla J(\theta^n)$$

Starting parameters



Optimisation

- The resulting cost function is non-linear and requires a general optimization method to find a minima
- **Gradient Descent** is an optimization strategy that recursively moves the parameter vector along the negative gradient direction of the cost function J() until convergence

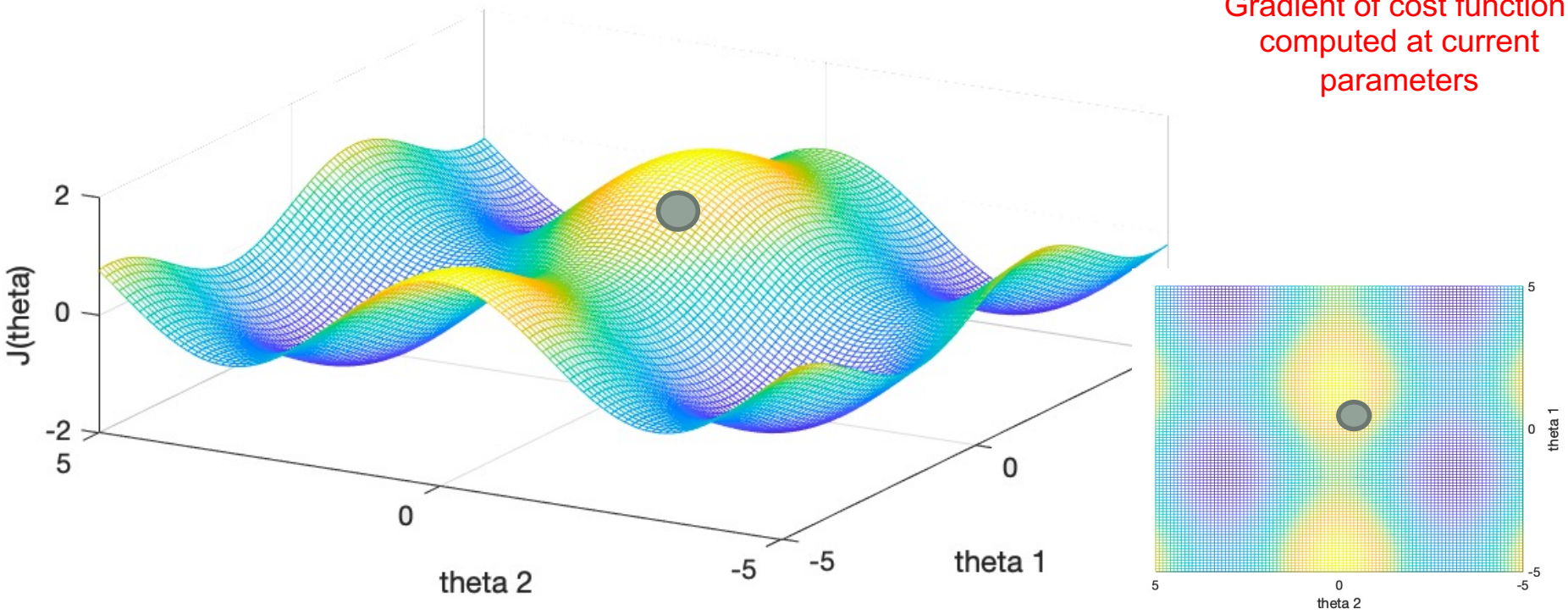
Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

$$\theta^{n+1} = \theta^n - \alpha \nabla J(\theta^n)$$

Gradient of cost function, computed at current parameters



Optimisation

- The resulting cost function is non-linear and requires a general optimization method to find a minima
- **Gradient Descent** is an optimization strategy that recursively moves the parameter vector along the negative gradient direction of the cost function J() until convergence

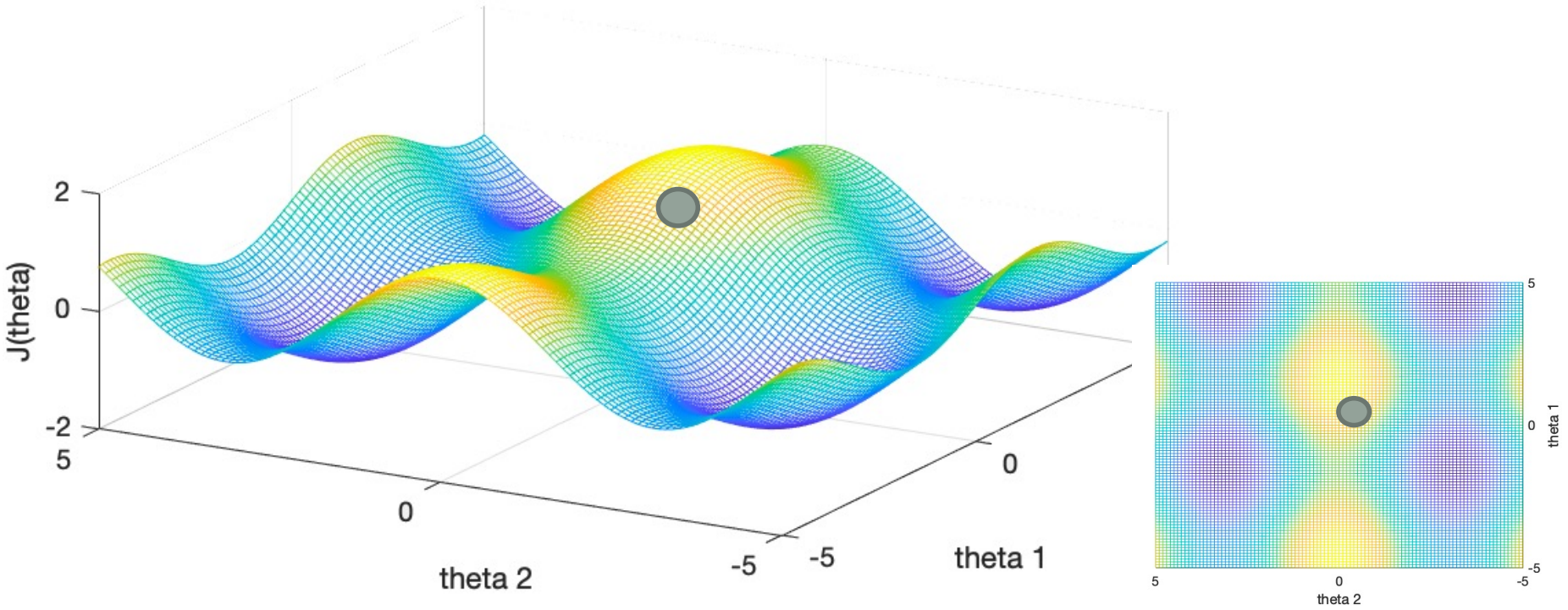
Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

$$\theta^{n+1} = \theta^n - \alpha \nabla J(\theta^n)$$

Step size



Optimisation

Training data pairs

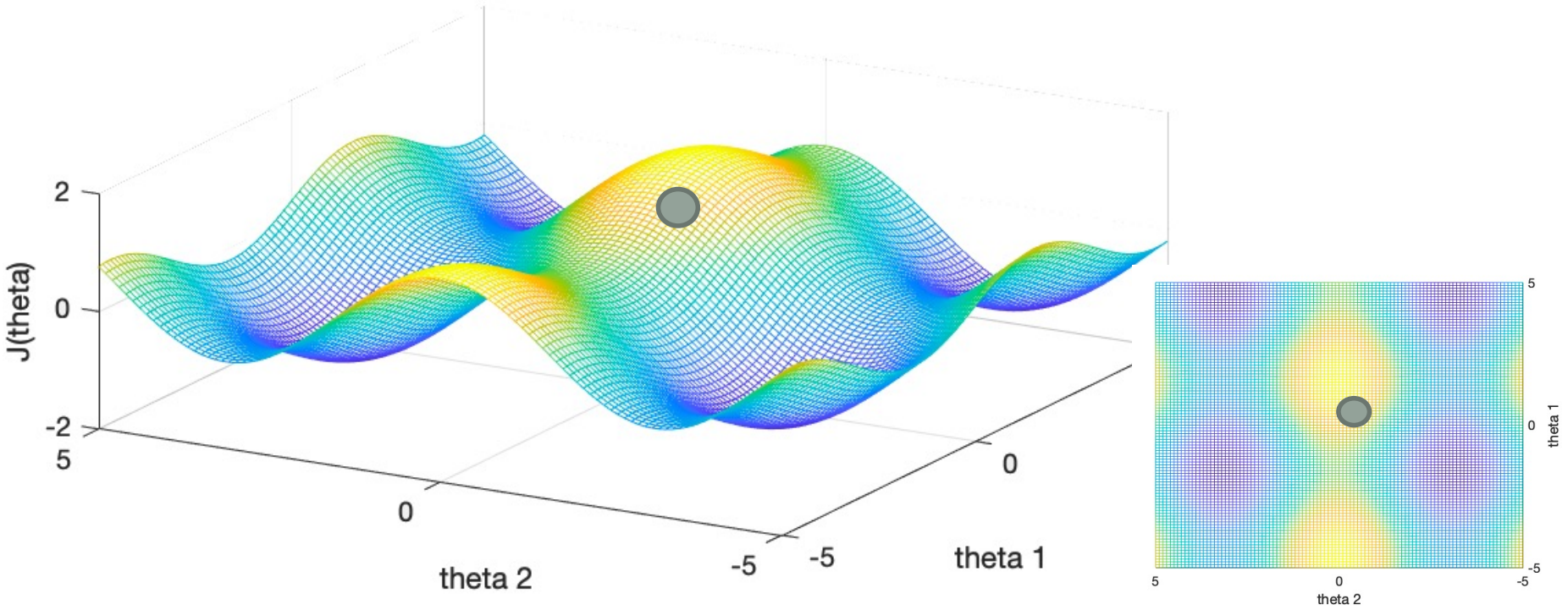
$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

- The resulting cost function is non-linear and requires a general optimization method to find a minima
- **Gradient Descent** is an optimization strategy that recursively moves the parameter vector along the negative gradient direction of the cost function J() until convergence

Updated parameters

$$\theta^{n+1} = \theta^n - \alpha \nabla J(\theta^n)$$



Optimisation

Training data pairs

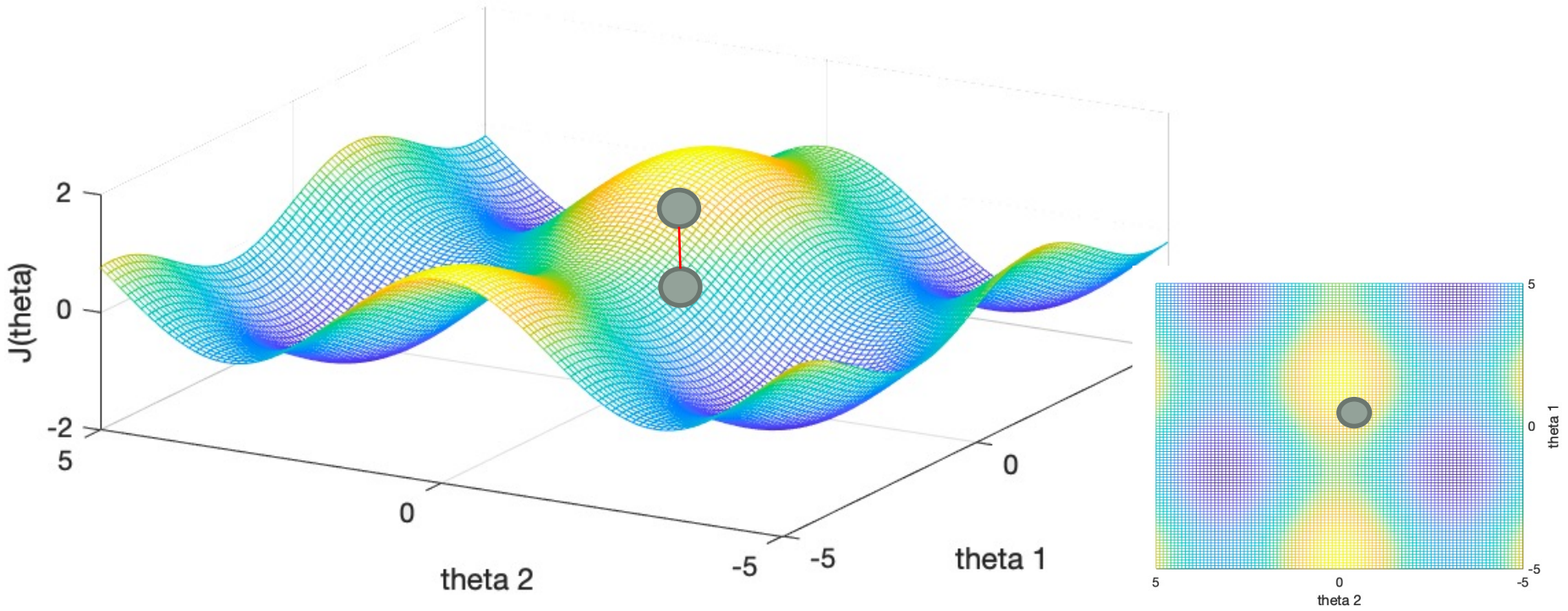
$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

- The resulting cost function is non-linear and requires a general optimization method to find a minima
- **Gradient Descent** is an optimization strategy that recursively moves the parameter vector along the negative gradient direction of the cost function J() until convergence

Updated parameters

$$\theta^{n+1} = \theta^n - \alpha \nabla J(\theta^n)$$



Optimisation

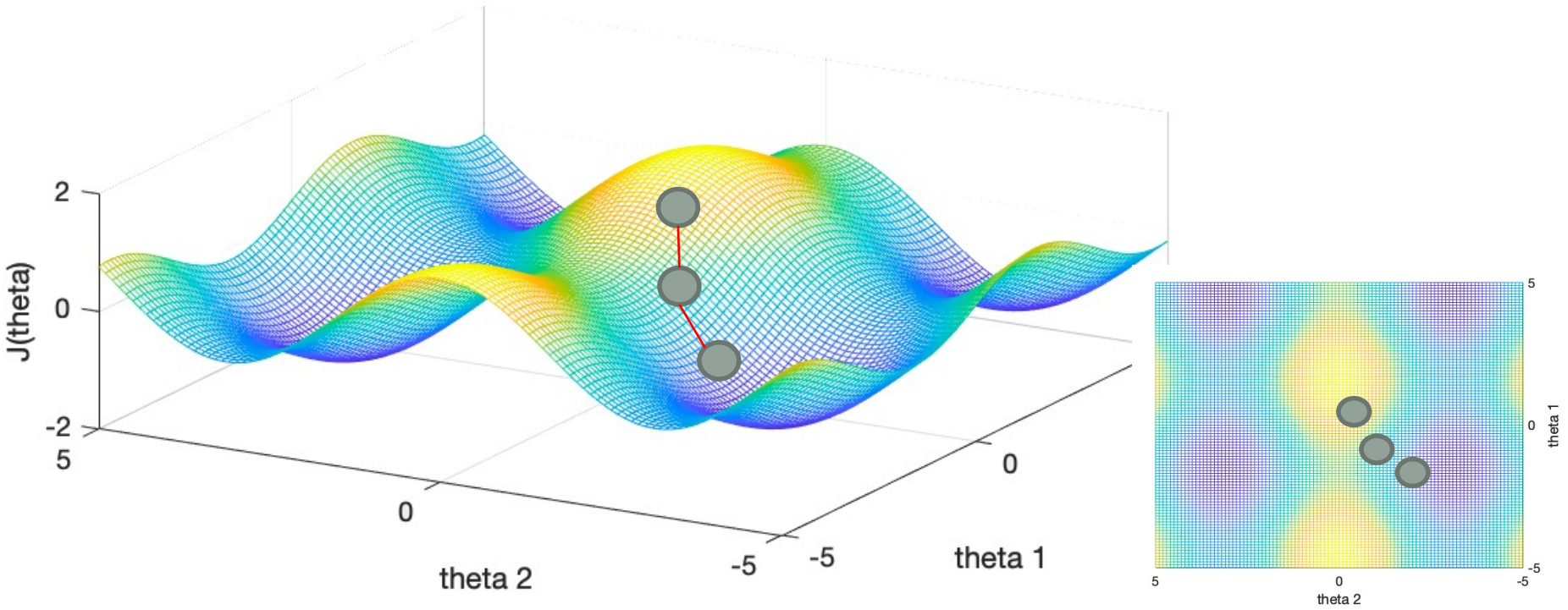
- The resulting cost function is non-linear and requires a general optimization method to find a minima
- **Gradient Descent** is an optimization strategy that recursively moves the parameter vector along the negative gradient direction of the cost function J() until convergence

Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

$$\theta^{n+1} = \theta^n - \alpha \nabla J(\theta^n)$$



Optimisation

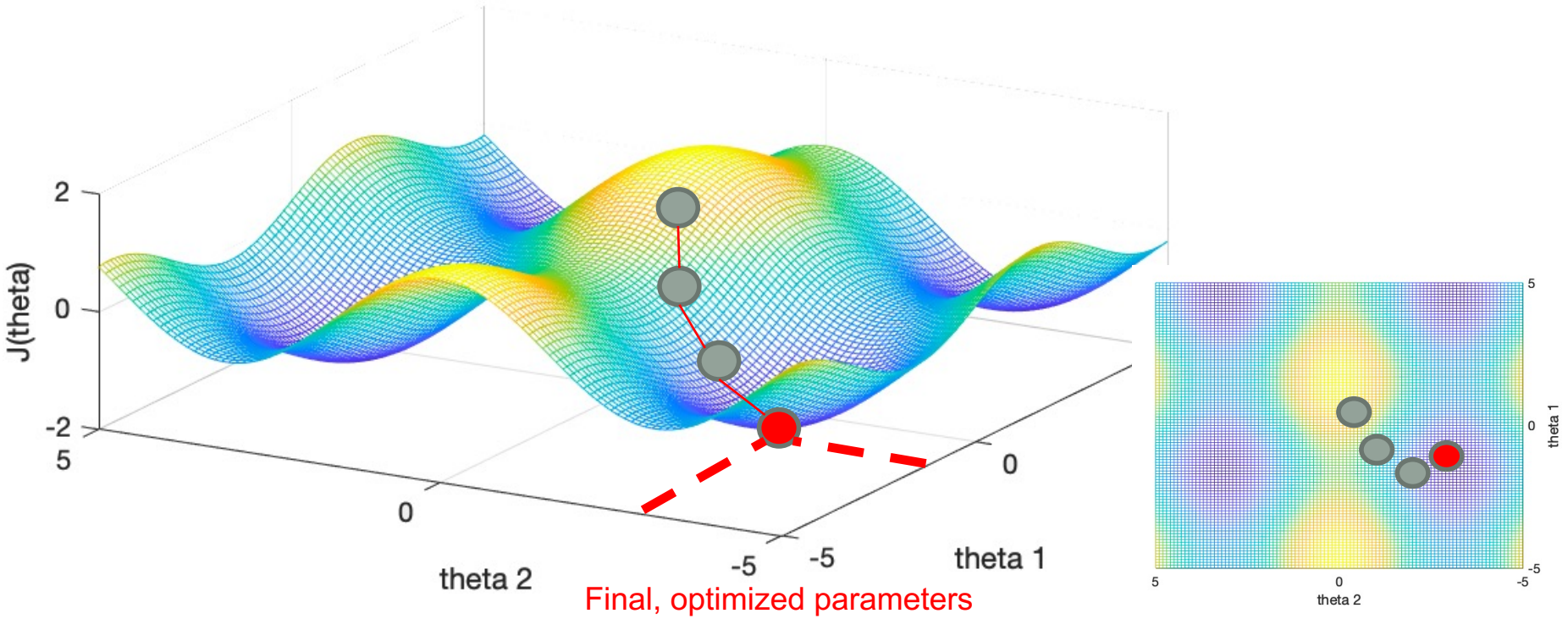
- The resulting cost function is non-linear and requires a general optimization method to find a minima
- **Gradient Descent** is an optimization strategy that recursively moves the parameter vector along the negative gradient direction of the cost function J() until convergence

Training data pairs

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

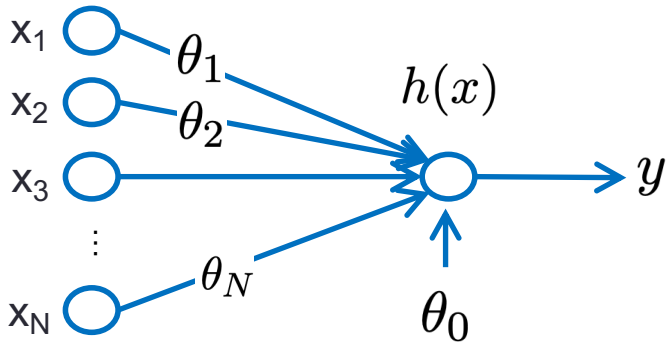
$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

$$\theta^{n+1} = \theta^n - \alpha \nabla J(\theta^n)$$



Problems with Logistic Regression

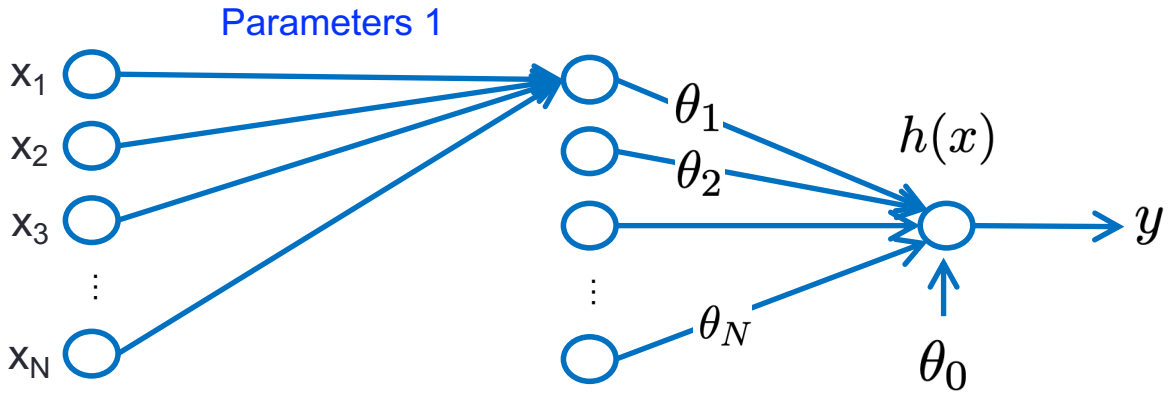
- Logistic regression can only model situations in which the input space can be separated using linear functions (i.e. a hyperplane)
- An effective model for dealing with complex problems must be able to model non-linear functions with arbitrary levels of complexity



Problems with Logistic Regression

- Logistic regression can only model situations in which the input space can be separated using linear functions (i.e. a hyperplane)
- An effective model for dealing with complex problems must be able to model non-linear functions with arbitrary levels of complexity

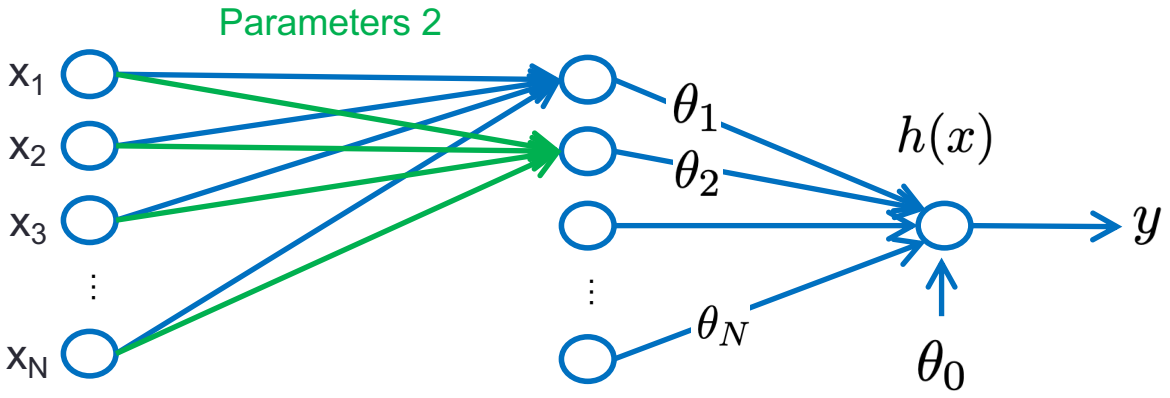
- One way to approach this is by "stacking" multiple logistic regression models together



Problems with Logistic Regression

- Logistic regression can only model situations in which the input space can be separated using linear functions (i.e. a hyperplane)
- An effective model for dealing with complex problems must be able to model non-linear functions with arbitrary levels of complexity

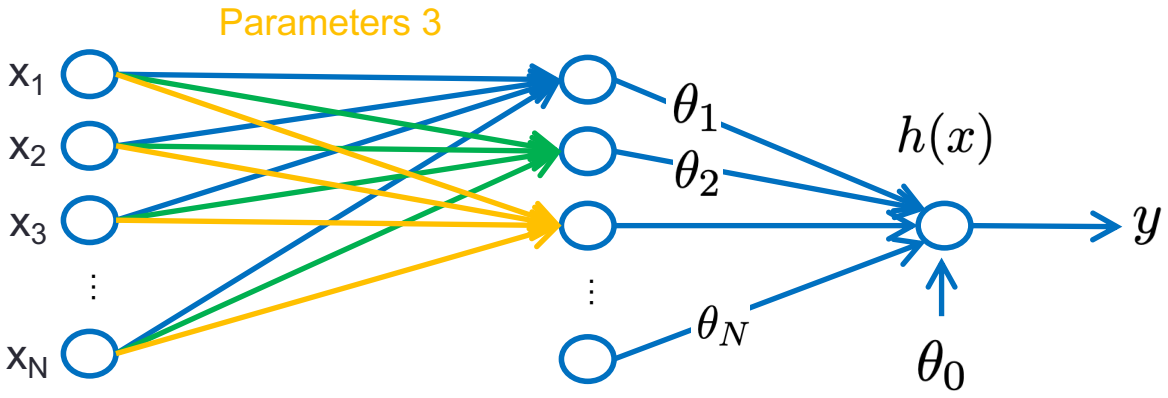
- One way to approach this is by "stacking" multiple logistic regression models together



Problems with Logistic Regression

- Logistic regression can only model situations in which the input space can be separated using linear functions (i.e. a hyperplane)
- An effective model for dealing with complex problems must be able to model non-linear functions with arbitrary levels of complexity

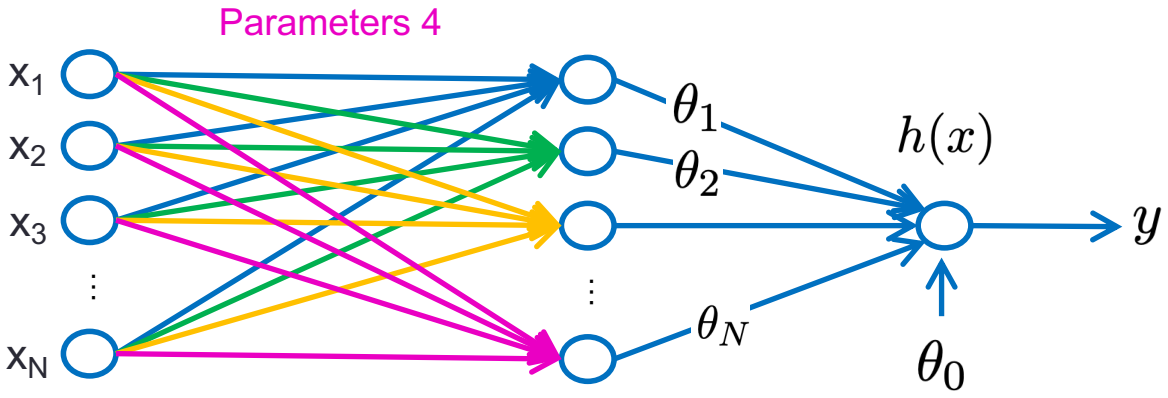
- One way to approach this is by "stacking" multiple logistic regression models together



Problems with Logistic Regression

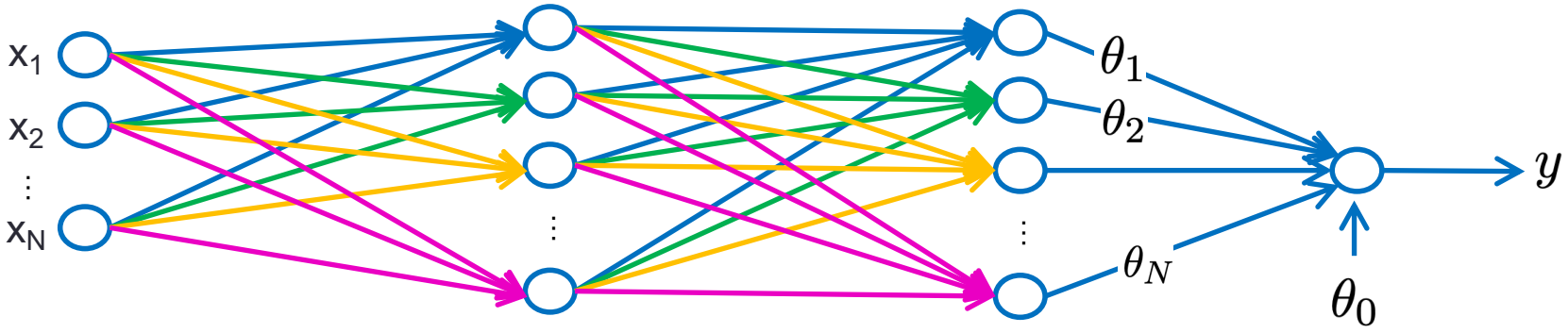
- Logistic regression can only model situations in which the input space can be separated using linear functions (i.e. a hyperplane)
- An effective model for dealing with complex problems must be able to model non-linear functions with arbitrary levels of complexity

- One way to approach this is by "stacking" multiple logistic regression models together
- At each input node, we now transform the outputs from the previous stage to a new linear combination of outputs, with sigmoid



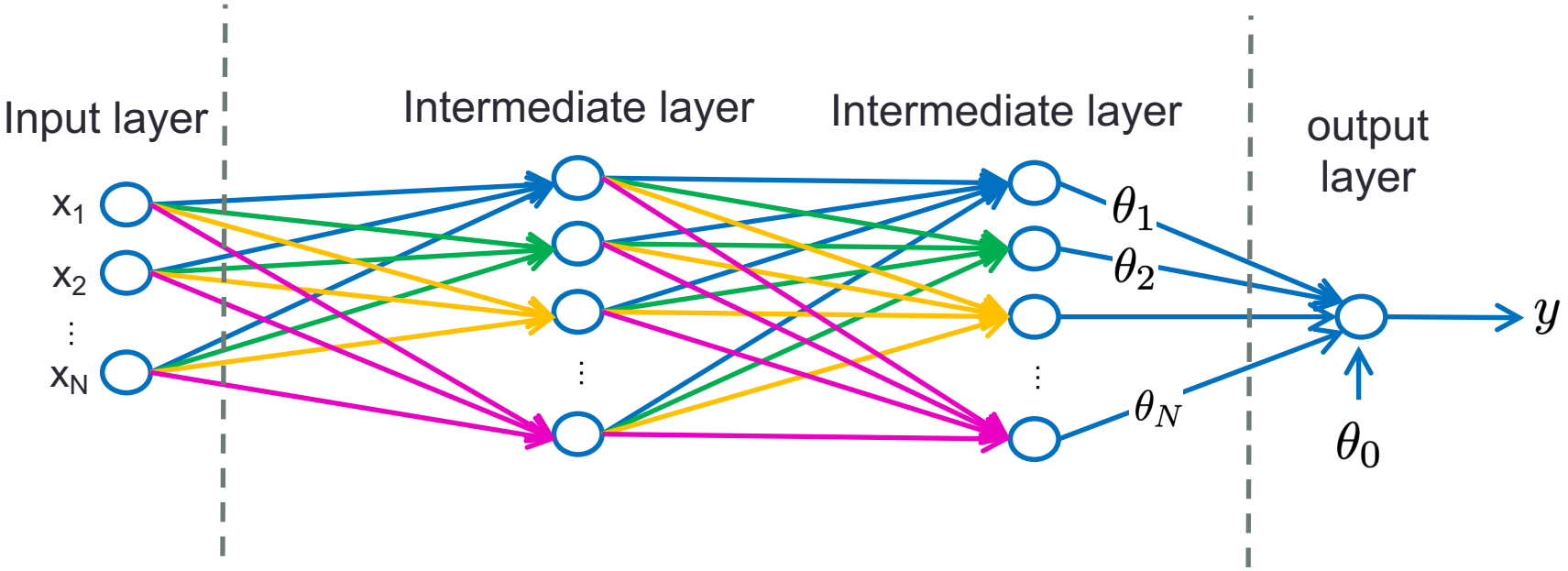
A Neural Network

- Repeating this process across multiple layers produces a **neural network** which becomes capable of modelling more arbitrarily complex functions with each layer



A Neural Network

- Repeating this process across multiple layers produces a **neural network** which becomes capable of modelling more arbitrarily complex functions with each layer
- We can vary the number of nodes in each intermediate layer as well as the values of the parameters connecting them (also referred to as weights)
- This architecture is sometime referred to as a Multi-Layered Perceptron (MLP) or a fully-connected network



A Neural Network

- Repeating this process across multiple layers produces a **neural network** which becomes capable of modelling more arbitrarily complex functions with each layer
- We can vary the number of nodes in each intermediate layer as well as the values of the parameters connecting them (also referred to as weights)
- This architecture is sometime referred to as a Multi-Layered Perceptron (MLP) or a fully-connected network

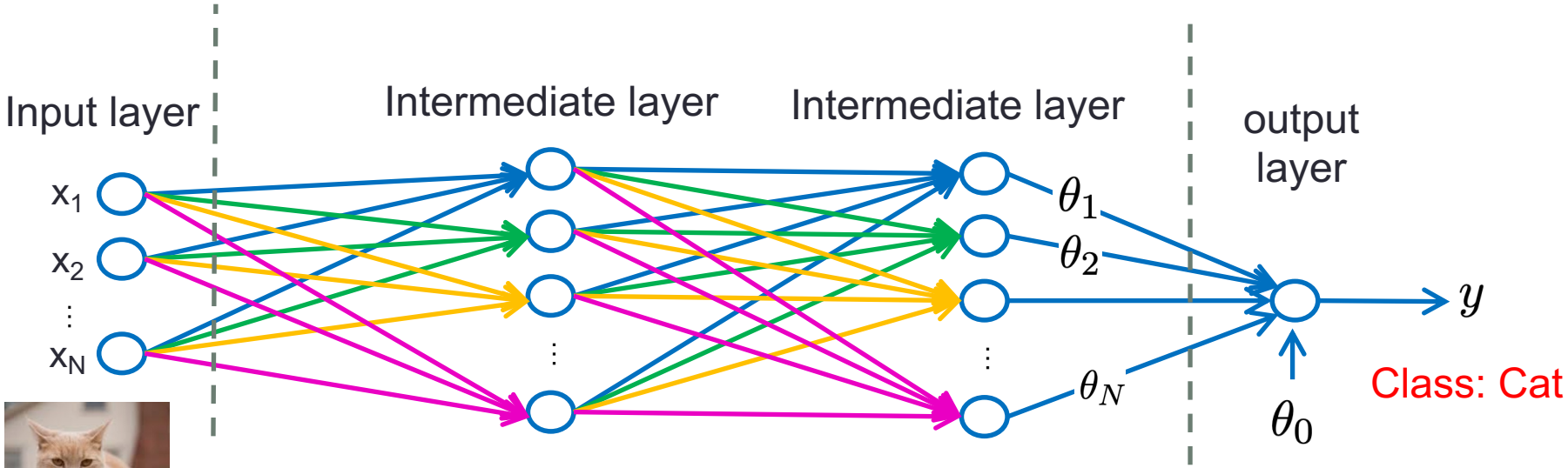
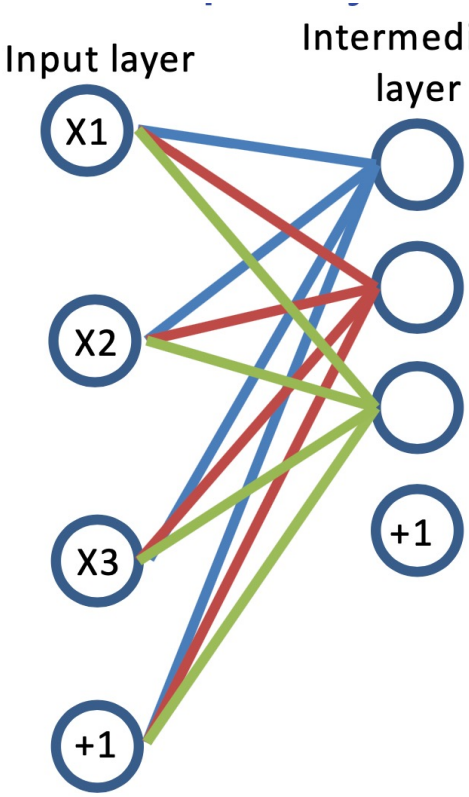


Image Classifier: input layer same dimension as pixels in image

Neural Networks: Matrix Notation

- For each intermediate layer node, we have the value of the node equal to linear combination of input layer's outputs via the parameter/weights
- To account for the constant offsets (θ_{x0}) we add a "bias" node to each layer for convenience in representation
- If we concatenate the outputs of the intermediate nodes, we can represent the operation of transforming one layer's output to the next layer's input using matrix notation:



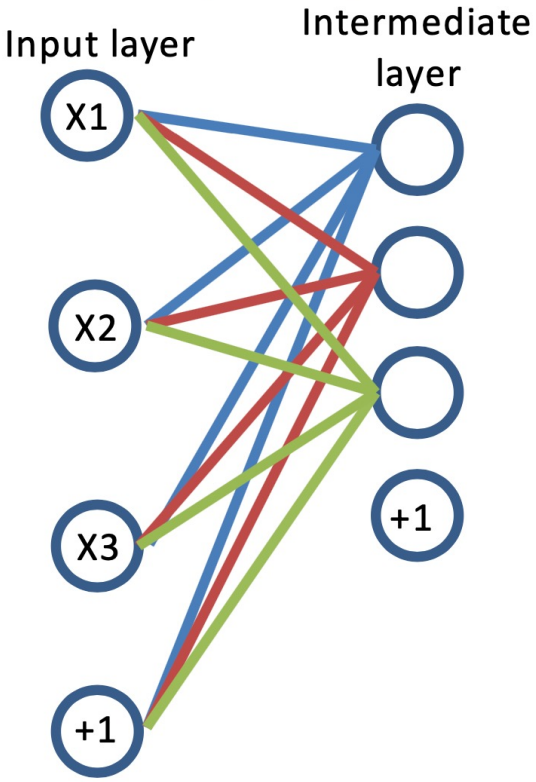
$$h(\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2 + \theta_{13}x_3)$$

$$h(\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2 + \theta_{23}x_3)$$

$$h(\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2 + \theta_{33}x_3)$$

Neural Networks: Matrix Notation

- For each intermediate layer node, we have the value of the node equal to linear combination of input layer's outputs via the parameter/weights
- To account for the constant offsets (θ_{x0}) we add a "bias" node to each layer for convenience in representation
- If we concatenate the outputs of the intermediate nodes, we can represent the operation of transforming one layer's output to the next layer's input using matrix notation:

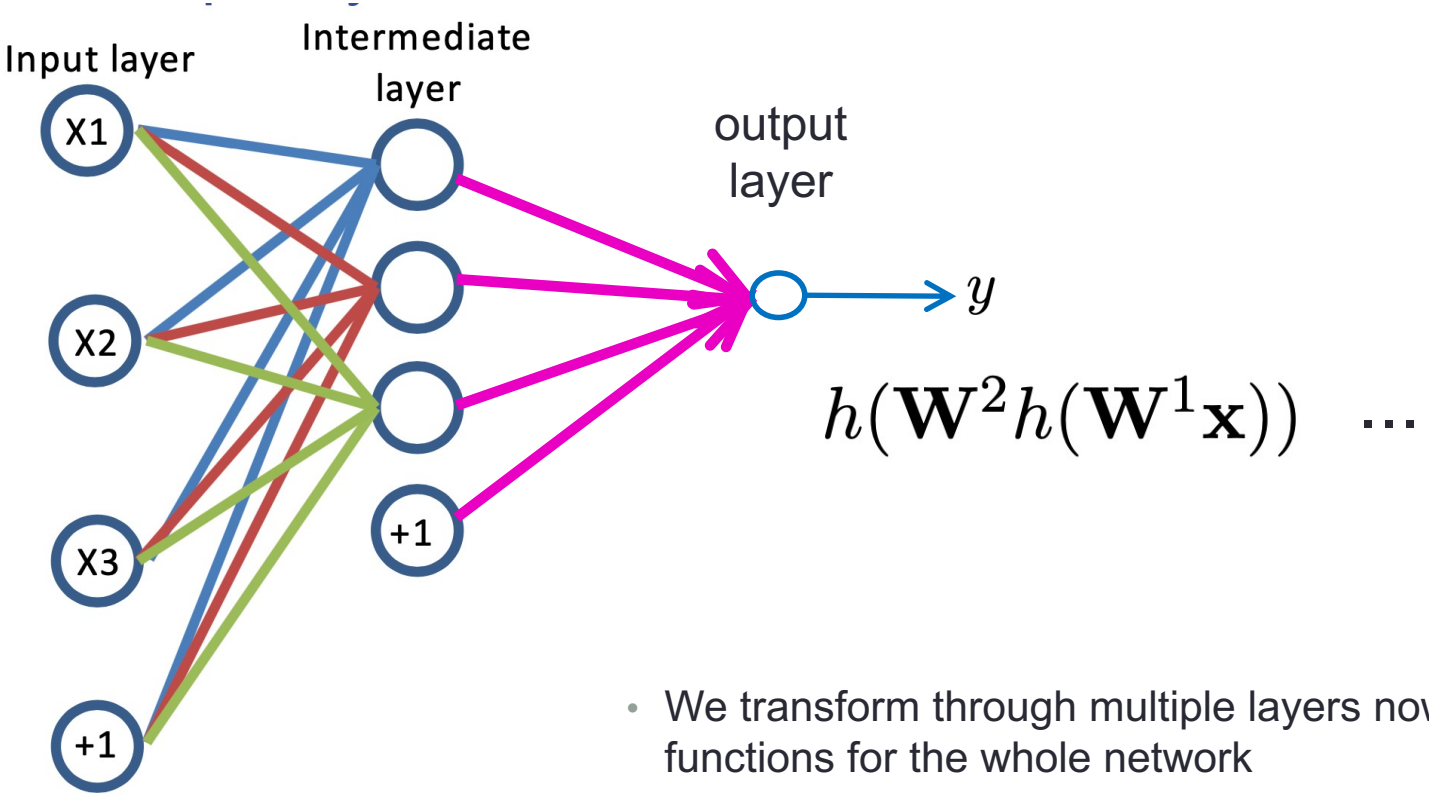


$$h\left(\begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} & \theta_{10} \\ \theta_{21} & \theta_{22} & \theta_{23} & \theta_{20} \\ \theta_{31} & \theta_{32} & \theta_{33} & \theta_{30} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{bmatrix}\right)$$

$$h(\mathbf{W}^1\mathbf{x})$$

Neural Networks: Matrix Notation

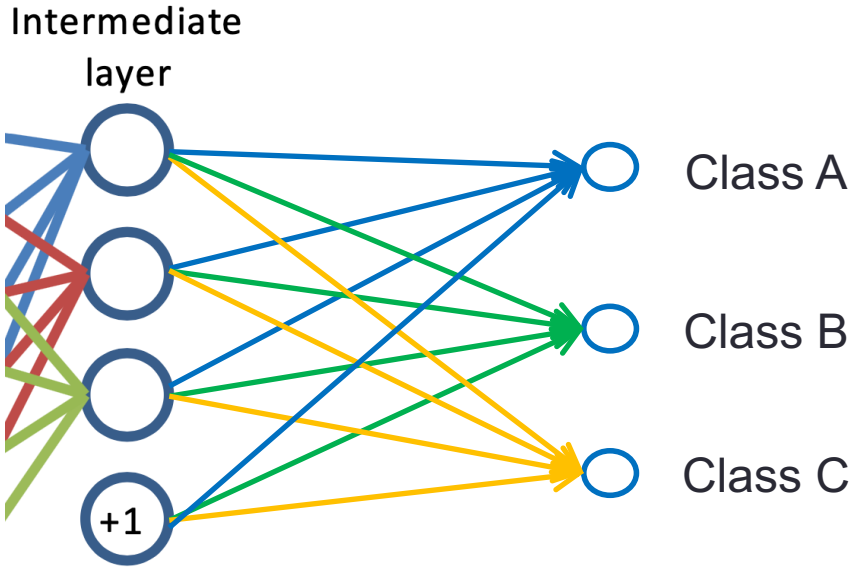
- For each intermediate layer node, we have the value of the node equal to linear combination of input layer's outputs via the parameter/weights
- To account for the constant offsets (θ_{x0}) we add a "bias" node to each layer for convenience in representation
- If we concatenate the outputs of the intermediate nodes, we can represent the operation of transforming one layer's output to the next layer's input using matrix notation:



- We transform through multiple layers now as functions of functions for the whole network

Neural Network: Output Layer

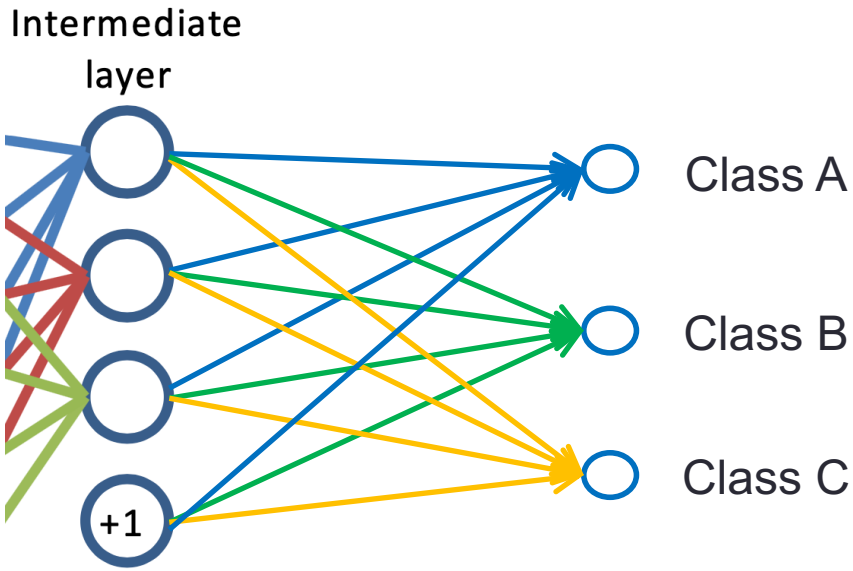
- The network's output layer may also contain multiple nodes, for example, to facilitate multi-class classification
- In multi-class classification we often assign one output node per class



Neural Network: Output Layer

- The network's output layer may also contain multiple nodes, for example, to facilitate multi-class classification
- In multi-class classification we often assign one output node per class
- The **Softmax** output is one way we can map the output layer to "probabilities" corresponding to multiple different classes

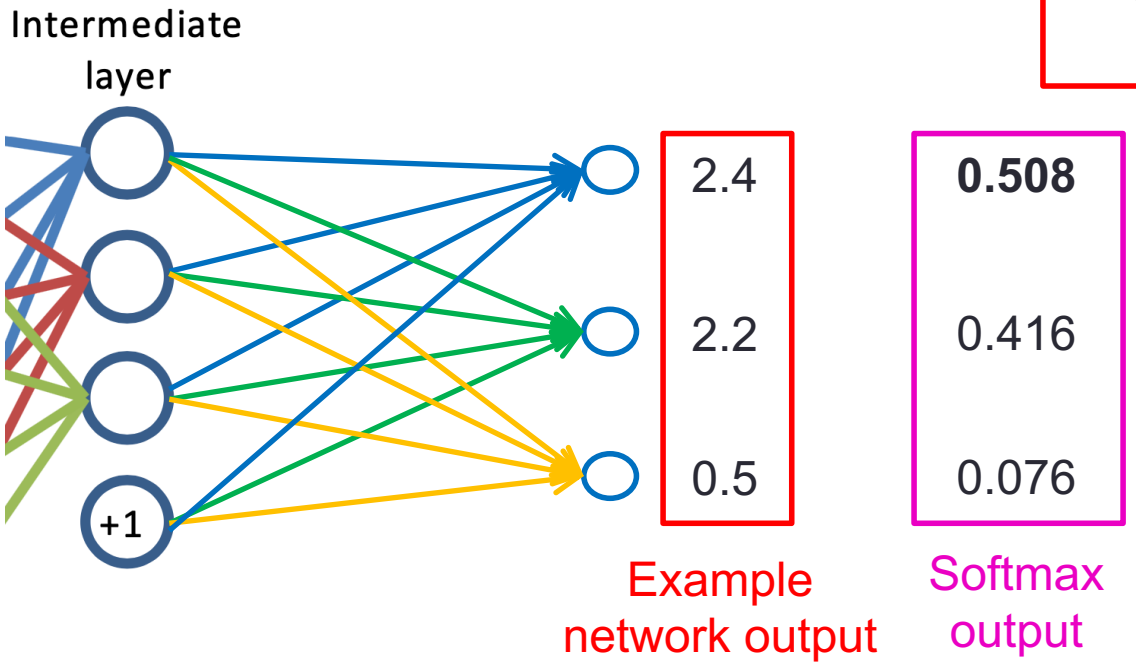
$$\sigma(x_j) = \frac{e^{x_j}}{\sum_{i=1}^{N} e^{x_i}}$$



Neural Network: Output Layer

- The network's output layer may also contain multiple nodes, for example, to facilitate multi-class classification
- In multi-class classification we often assign one output node per class
- The **Softmax** output is one way we can map the output layer to "probabilities" corresponding to multiple different classes

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_{i=1}^{N} e^{x_i}}$$



- Softmax corresponds to a pseudo-probability and we can take the maximum probability to determine the class output

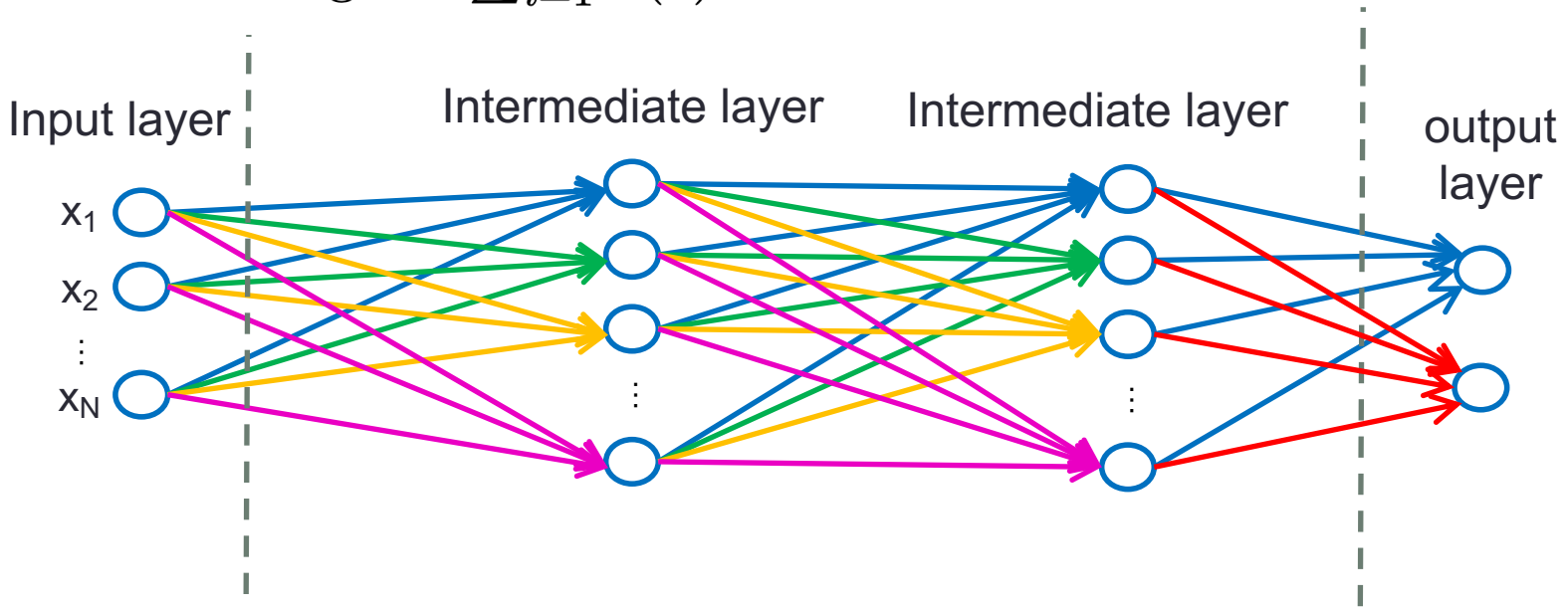
Neural Network Training and Optimisation

- Training consists of specifying the parameters inside W^1, W^2, ... , W^L, of the L layers that minimize a cost function computed over the training data
- A multi-layer network may have thousands to millions of parameters
- The same optimization procedure used for logistic regression (cross entropy cost, gradient descent optimisation) can still be used here:

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)$$

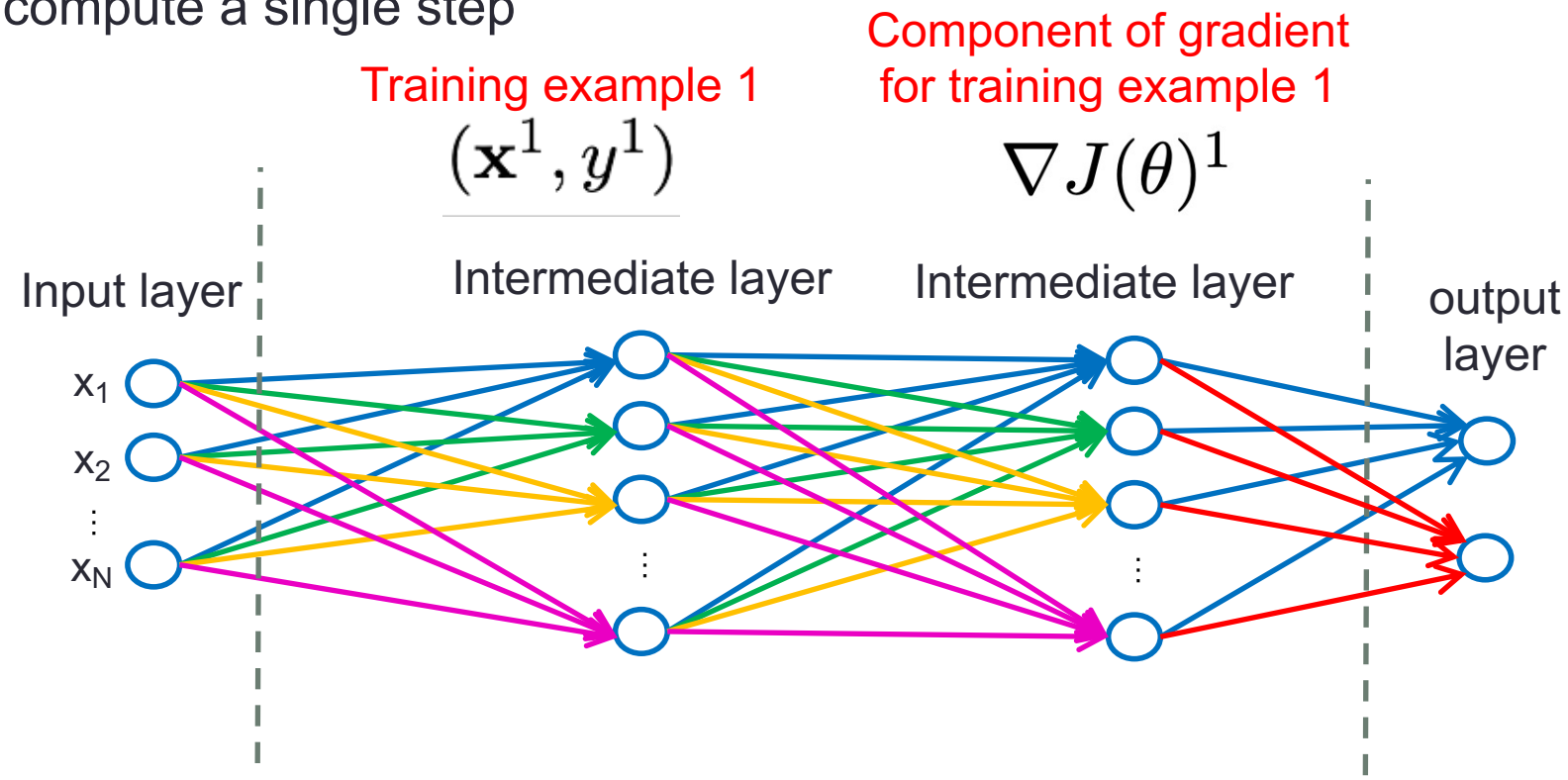
$$\theta^* = \arg\min \sum_{i=1}^{M} J(\theta)^i$$

$$\theta^{n+1} = \theta^n - \alpha \nabla J(\theta^n)$$



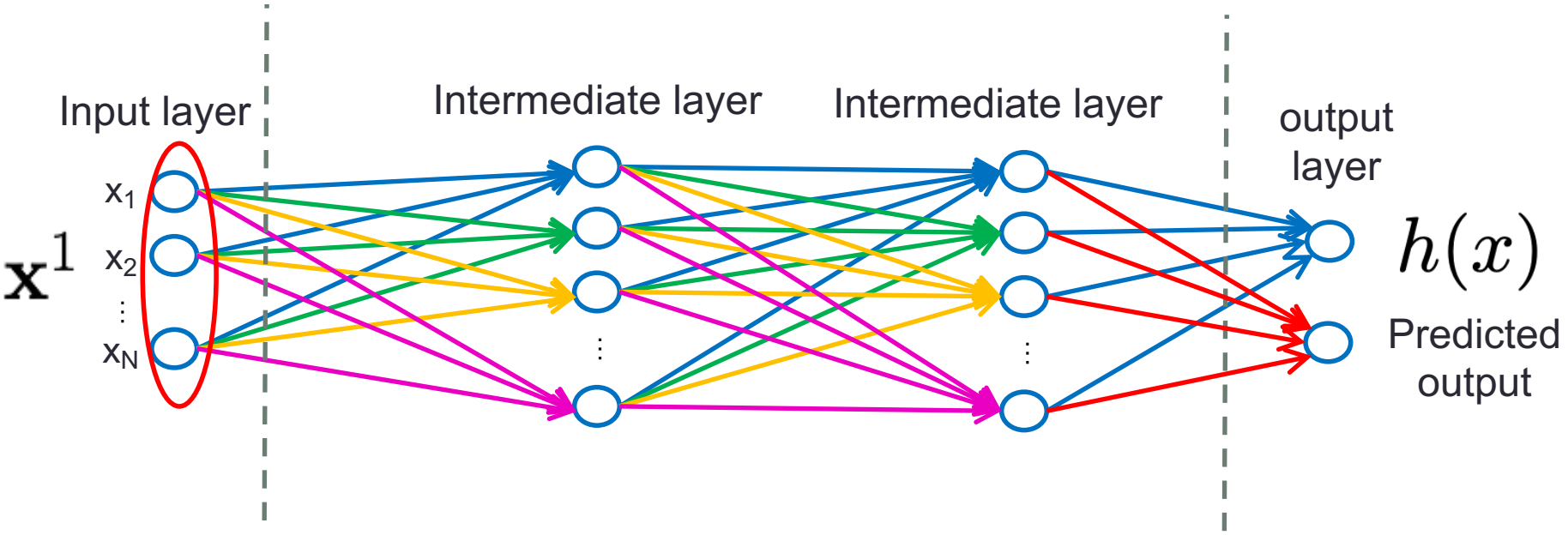
Neural Network Training and Optimisation

- The main computation required is the calculation of the gradient vector at each iteration: tackled through a process known as **back-propagation**
- The total cost function gradient for all examples is the sum of the gradients for each training example i: back-propagation computes each change in the gradient for each example and sums them together to compute a single step



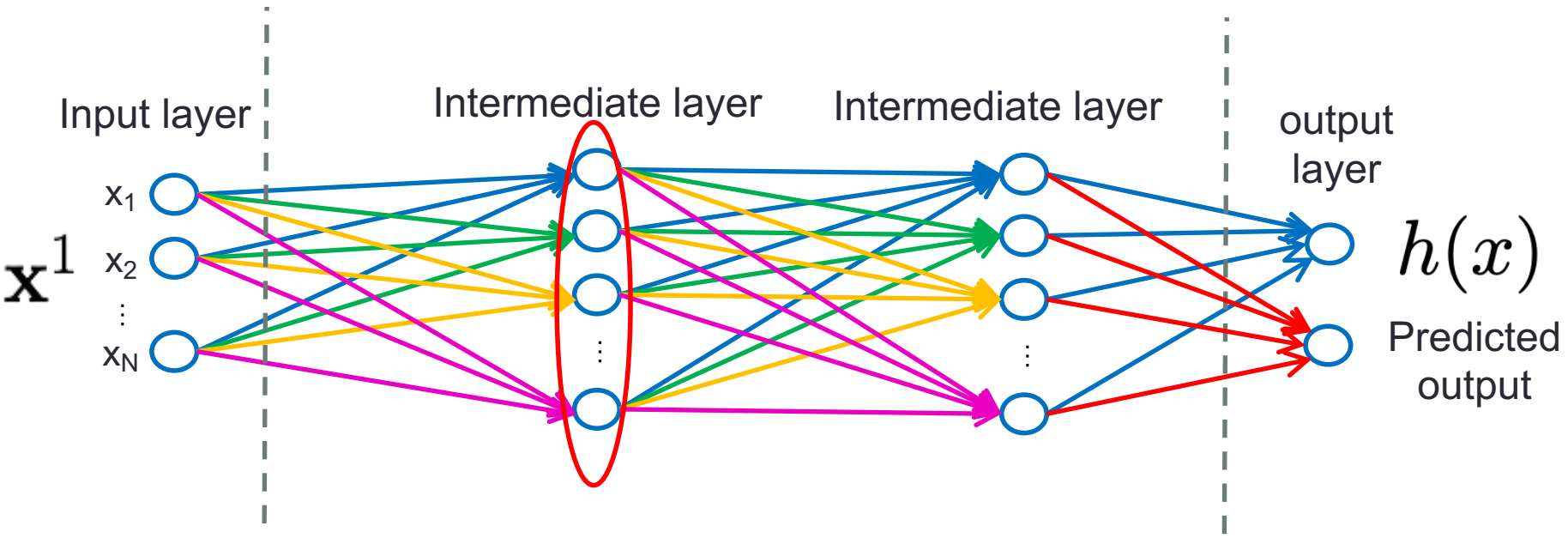
Neural Network Training and Optimisation

- The process begins with passing an example through the network's current parameters (referred to as "Feed-Forward") and comparing to the output value
- Gradients for the last layer are then calculated based on the difference between expected and predicted output
- Gradients are then calculated each layer back one-by-one based on those of the preceding layer based on the chain rule of differentiation



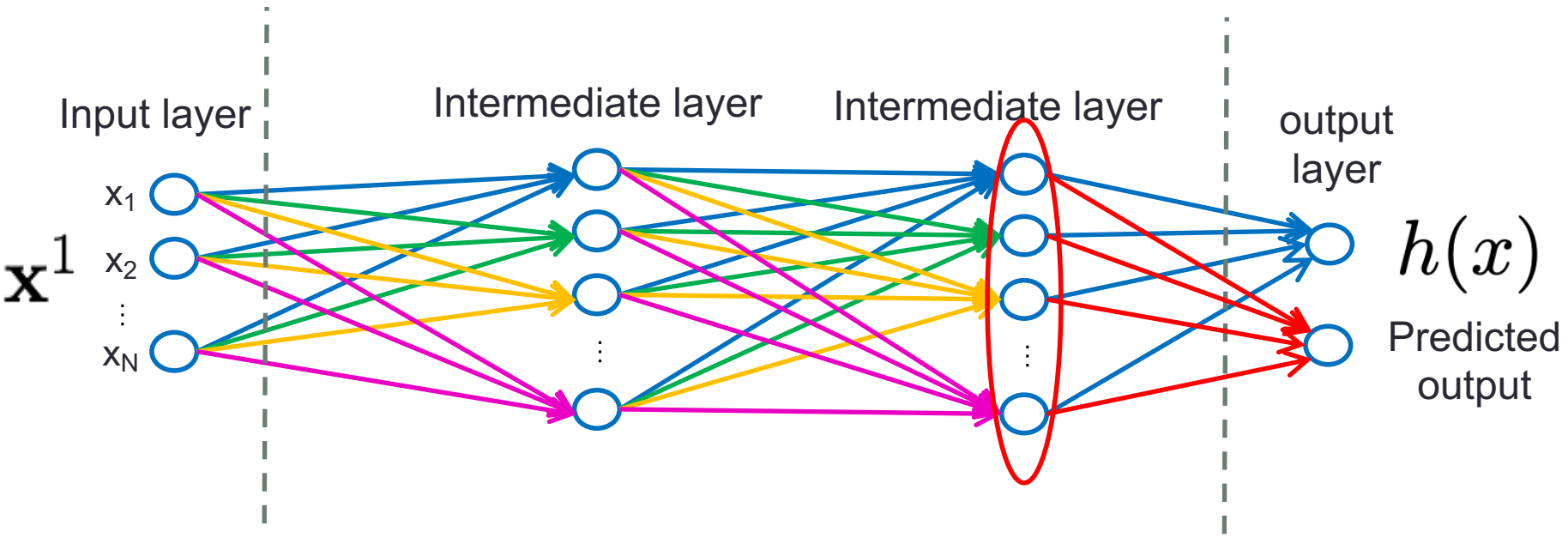
Neural Network Training and Optimisation

- The process begins with passing an example through the network's current parameters (referred to as "Feed-Forward") and comparing to the output value
- Gradients for the last layer are then calculated based on the difference between expected and predicted output
- Gradients are then calculated each layer back one-by-one based on those of the preceding layer based on the chain rule of differentiation



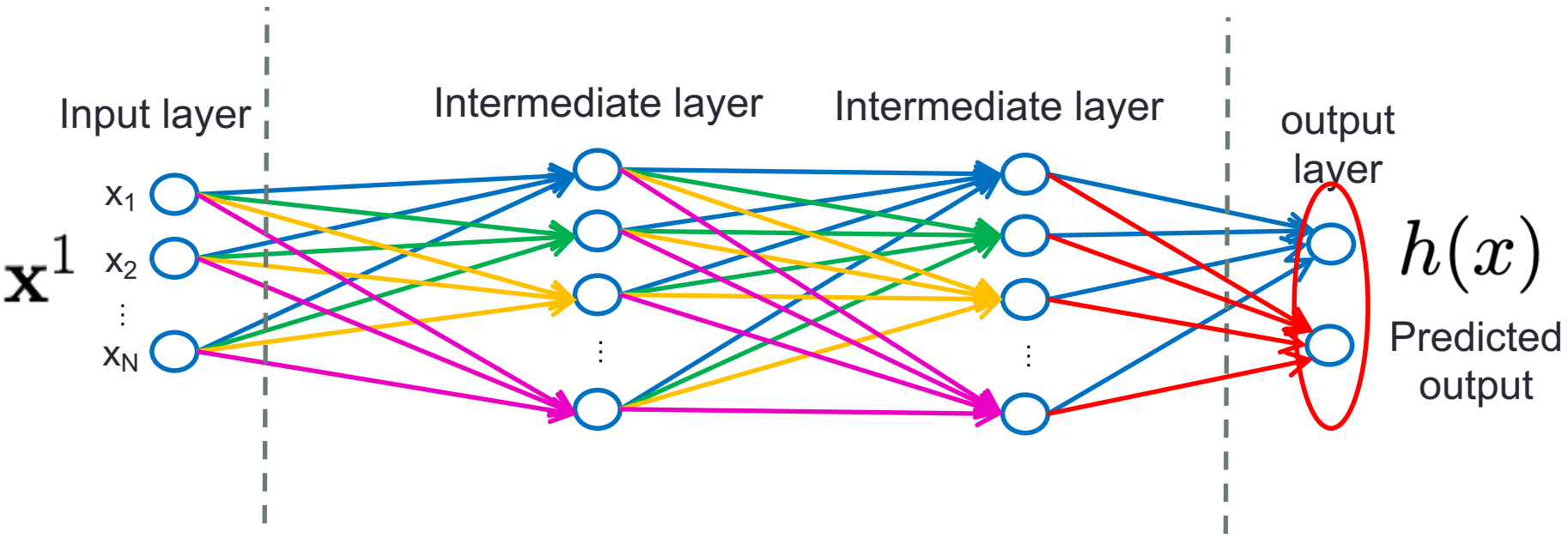
Neural Network Training and Optimisation

- The process begins with passing an example through the network's current parameters (referred to as "Feed-Forward") and comparing to the output value
- Gradients for the last layer are then calculated based on the difference between expected and predicted output
- Gradients are then calculated each layer back one-by-one based on those of the preceding layer based on the chain rule of differentiation



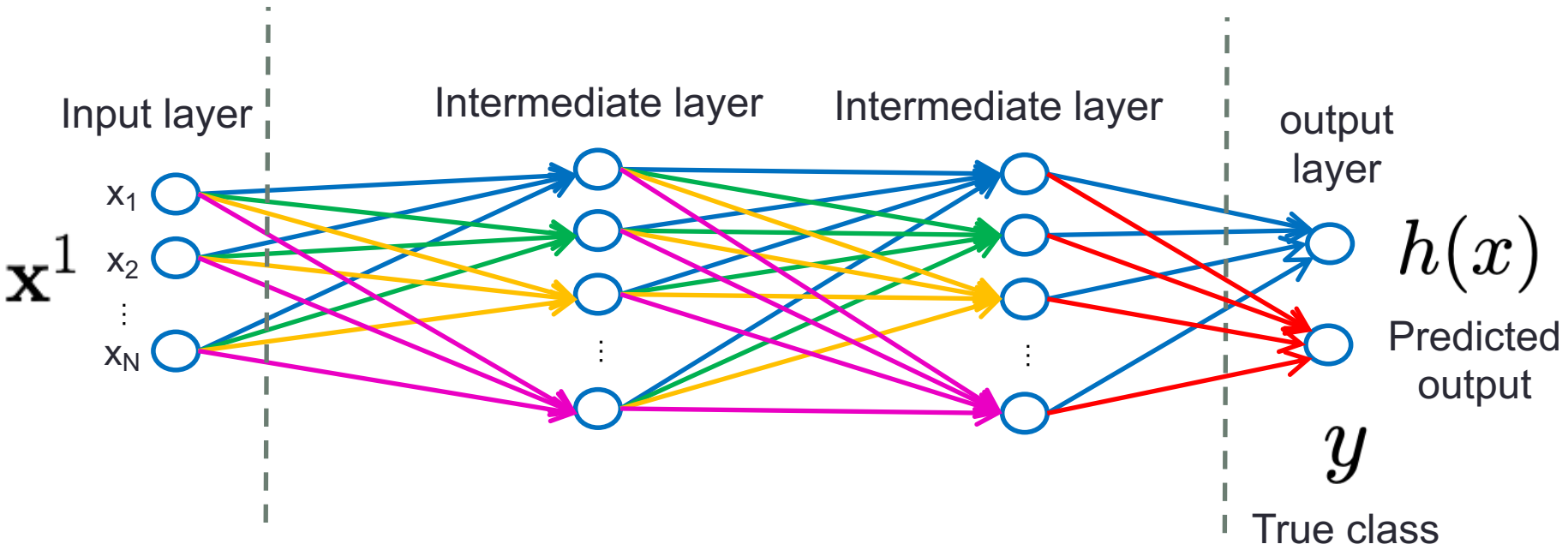
Neural Network Training and Optimisation

- The process begins with passing an example through the network's current parameters (referred to as "Feed-Forward") and comparing to the output value
- Gradients for the last layer are then calculated based on the difference between expected and predicted output
- Gradients are then calculated each layer back one-by-one based on those of the preceding layer based on the chain rule of differentiation



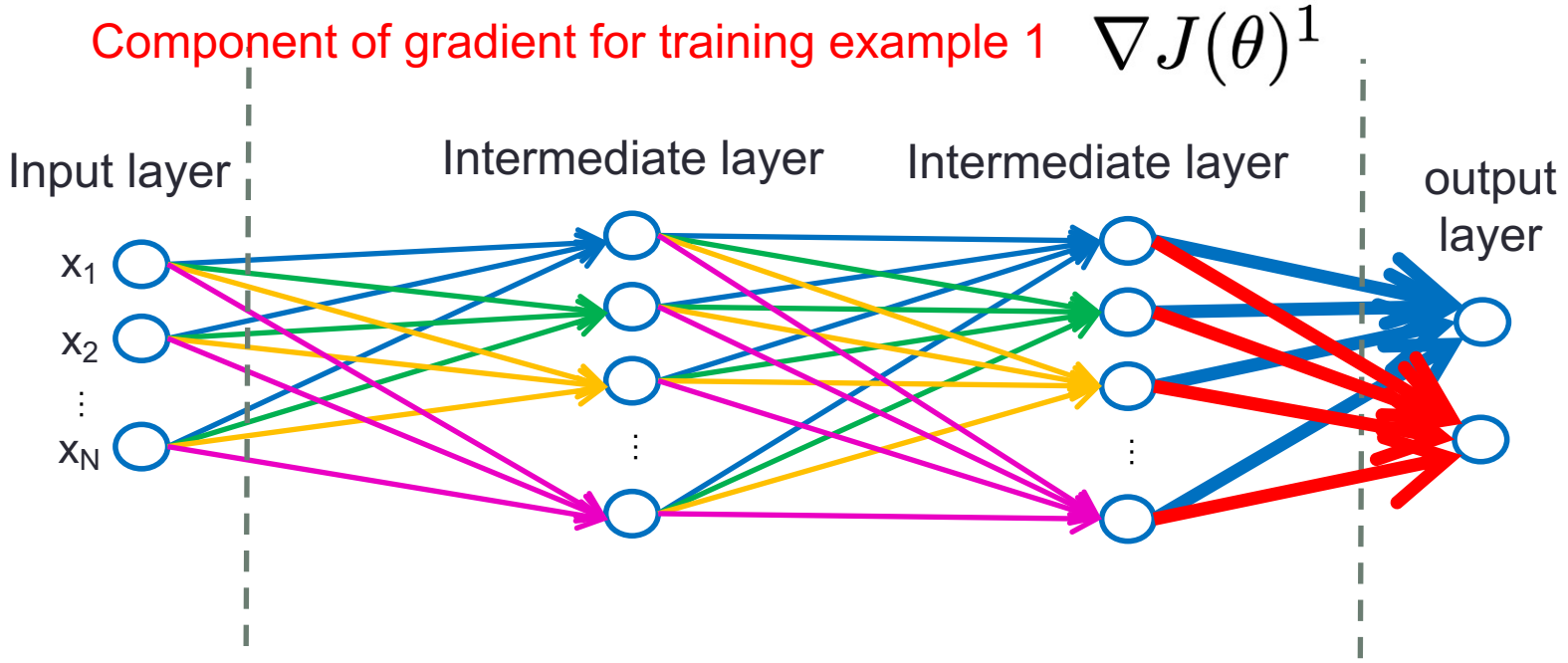
Neural Network Training and Optimisation

- The process begins with passing an example through the network's current parameters (referred to as "Feed-Forward") and comparing to the output value
- Gradients for the last layer are then calculated based on the difference between expected and predicted output
- Gradients are then calculated each layer back one-by-one based on those of the preceding layer based on the chain rule of differentiation



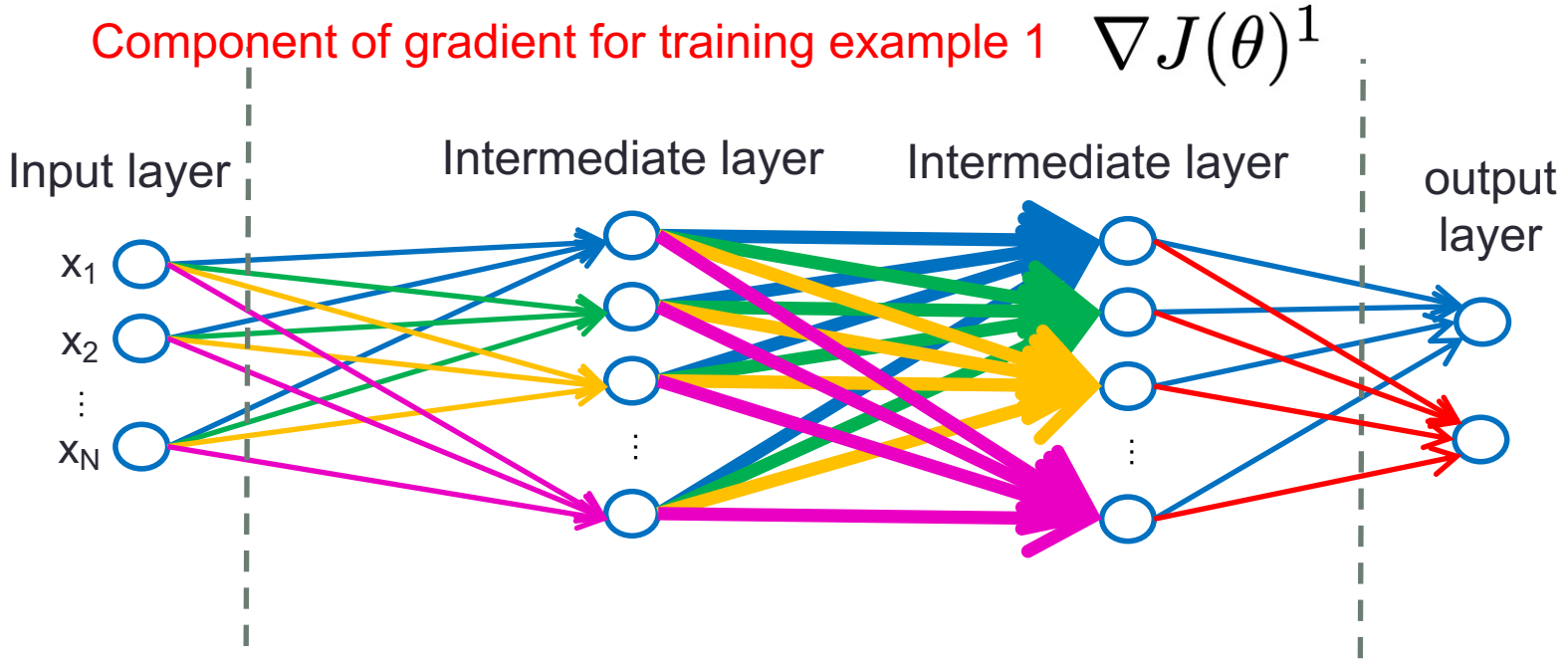
Neural Network Training and Optimisation

- The process begins with passing an example through the network's current parameters (referred to as "Feed-Forward") and comparing to the output value
- Gradients for the last layer are then calculated based on the difference between expected and predicted output
- Gradients are then calculated each layer back one-by-one based on those of the preceding layer based on the chain rule of differentiation



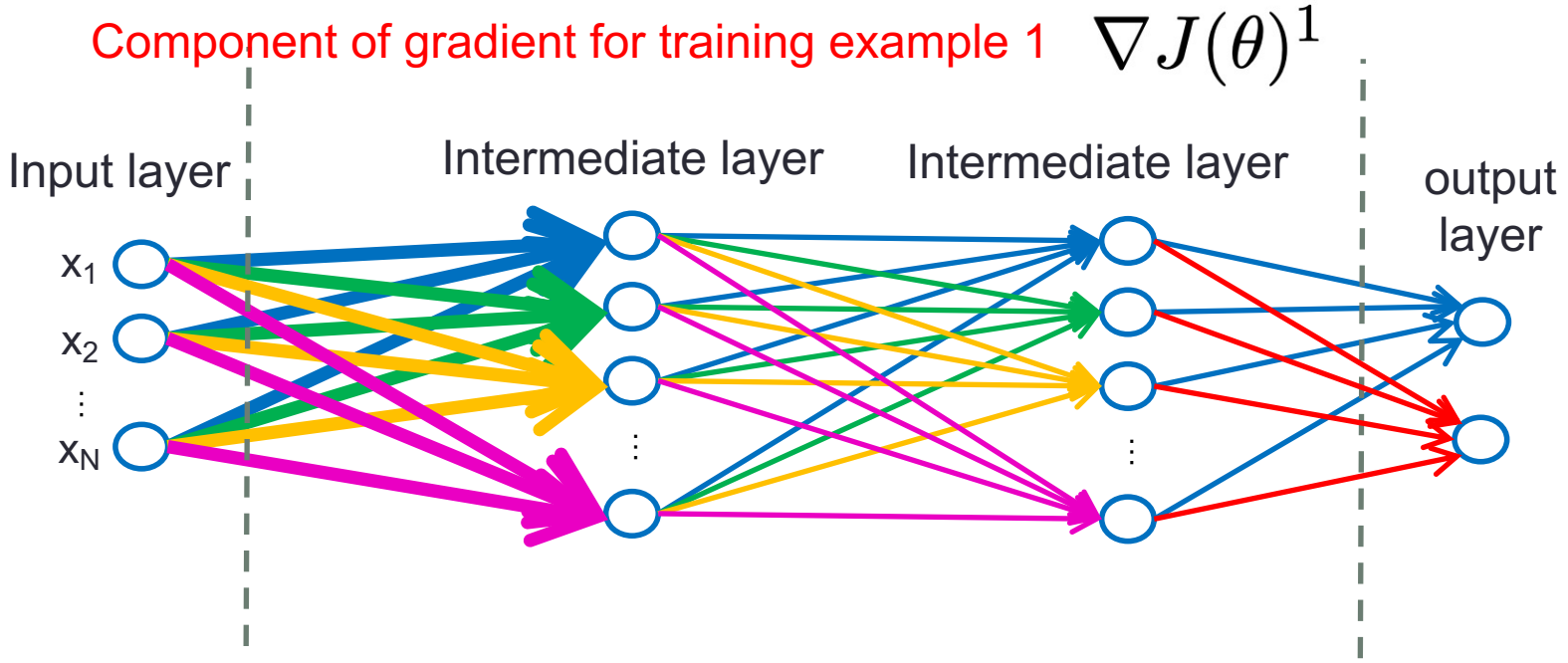
Neural Network Training and Optimisation

- The process begins with passing an example through the network's current parameters (referred to as "Feed-Forward") and comparing to the output value
- Gradients for the last layer are then calculated based on the difference between expected and predicted output
- Gradients are then calculated each layer back one-by-one based on those of the preceding layer based on the chain rule of differentiation



Neural Network Training and Optimisation

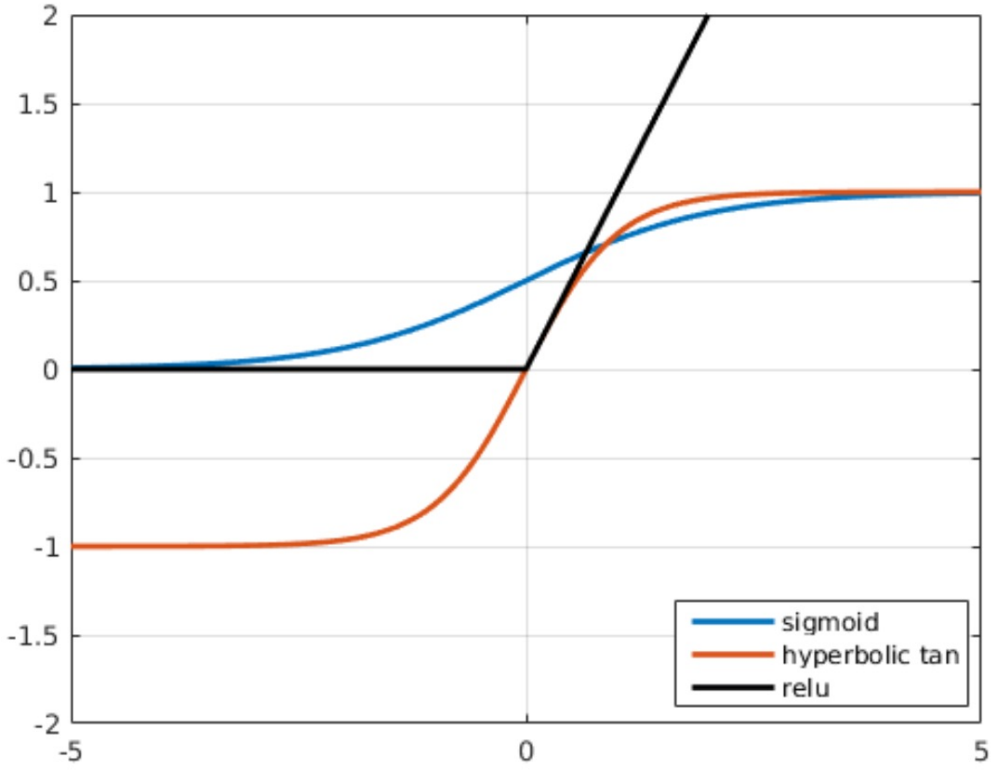
- The process begins with passing an example through the network's current parameters (referred to as "Feed-Forward") and comparing to the output value
- Gradients for the last layer are then calculated based on the difference between expected and predicted output
- Gradients are then calculated each layer back one-by-one based on those of the preceding layer based on the chain rule of differentiation



Activation Functions

- The output sigmoid function applied at each node (referred to as the activation function) can be replaced by other functions that may make training easier

- Sigmoid: “sigm”
- Hyperbolic Tangent ”tanh”
- Rectified Linear Units “relu”

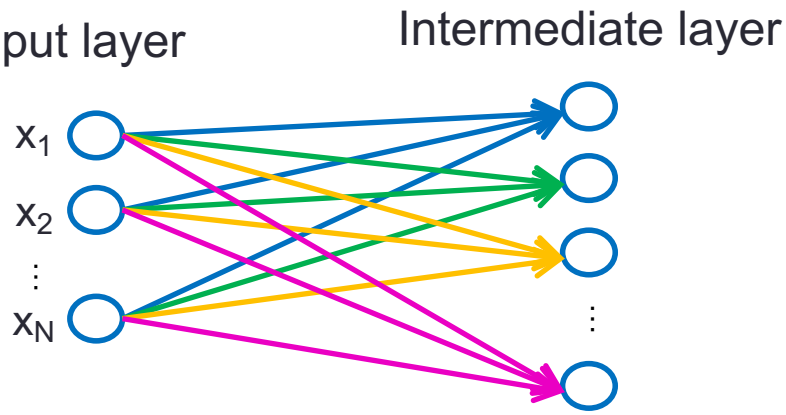


5 minute break

Convolutional Neural Networks

- The MLP connects every single pixel/input in the image to a large number of nodes: number of parameters is very high, and therefore can be difficult to train effectively for high-resolution images
- Convolutional Neural Networks (CNNs) simplify this complexity by modelling a layer that applies a convolution across the image data using a set of filters
- The filters themselves become the parameters of the layer, resulting in lower complexity but with enough depth to effectively model image interpretation

200x200 pixels = 40,000 input nodes



Even just 100 intermediate nodes = 4,000,000 parameters to train (just this layer)

Convolutional Neural Networks

- The MLP connects every single pixel/input in the image to a large number of nodes: number of parameters is very high, and therefore can be difficult to train effectively for high-resolution images
- Convolutional Neural Networks (CNNs) simplify this complexity by modelling a layer that applies a convolution across the image data using a set of filters
- The filters themselves become the parameters of the layer, resulting in lower complexity but with enough depth to effectively model image interpretation

Input image

Input image

2	2	2	1	1
1	4	4	1	3
2	3	4	3	2
3	3	2	4	1
2	4	3	4	2

Filter

-1	-2	-1
0	0	0
1	2	1

Response

4	7	7
-2	-2	-2
1	0	1

4	7	7
0	0	0
1	0	1

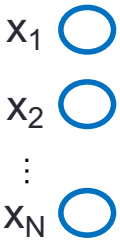
Response (after activation, relu)

Response Image

Convolutional Neural Networks

- The MLP connects every single pixel/input in the image to a large number of nodes: number of parameters is very high, and therefore can be difficult to train effectively for high-resolution images
- Convolutional Neural Networks (CNNs) simplify this complexity by modelling a layer that applies a convolution across the image data using a set of filters
- The filters themselves become the parameters of the layer, resulting in lower complexity but with enough depth to effectively model image interpretation

200x200 pixels = 40,000 input nodes



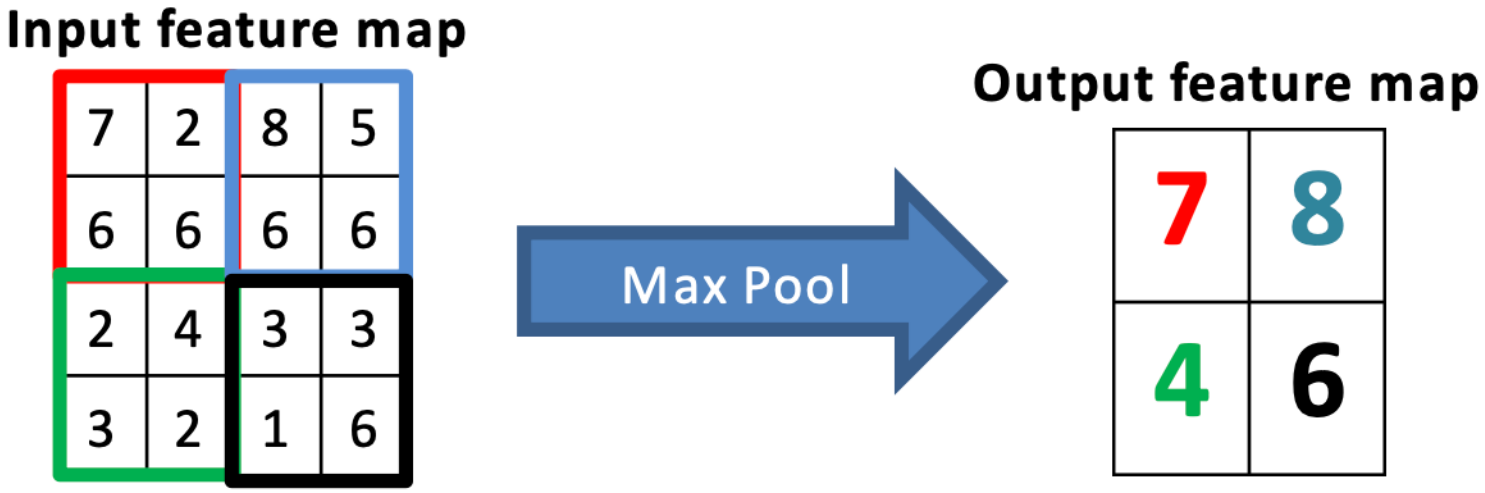
Convolutional Layer

Output: 128 response images

Example
128 5x5 filters = 3,200 parameters

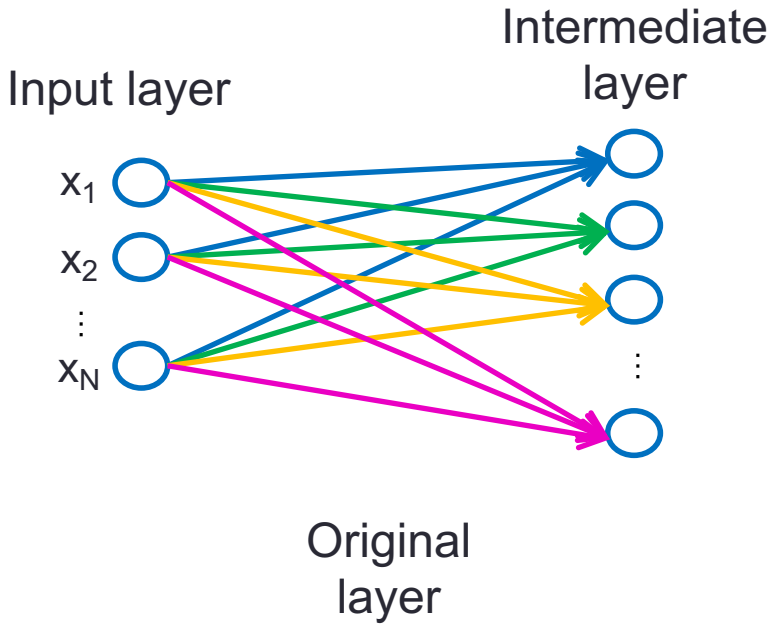
Other Network Layers: Pooling

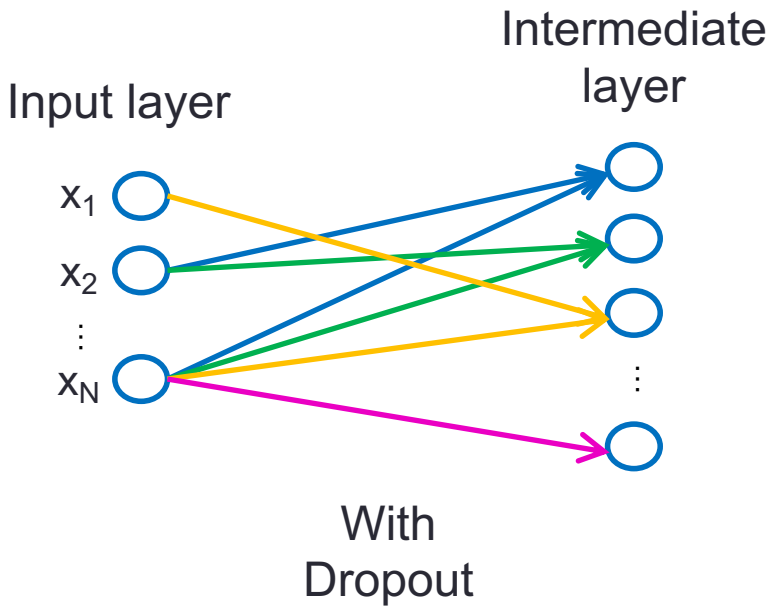
- Pooling is a dimensionality reduction technique that down-samples the input image from one layer to the next
- Pooling can be based on the max or mean



Other Network Layers: Dropout

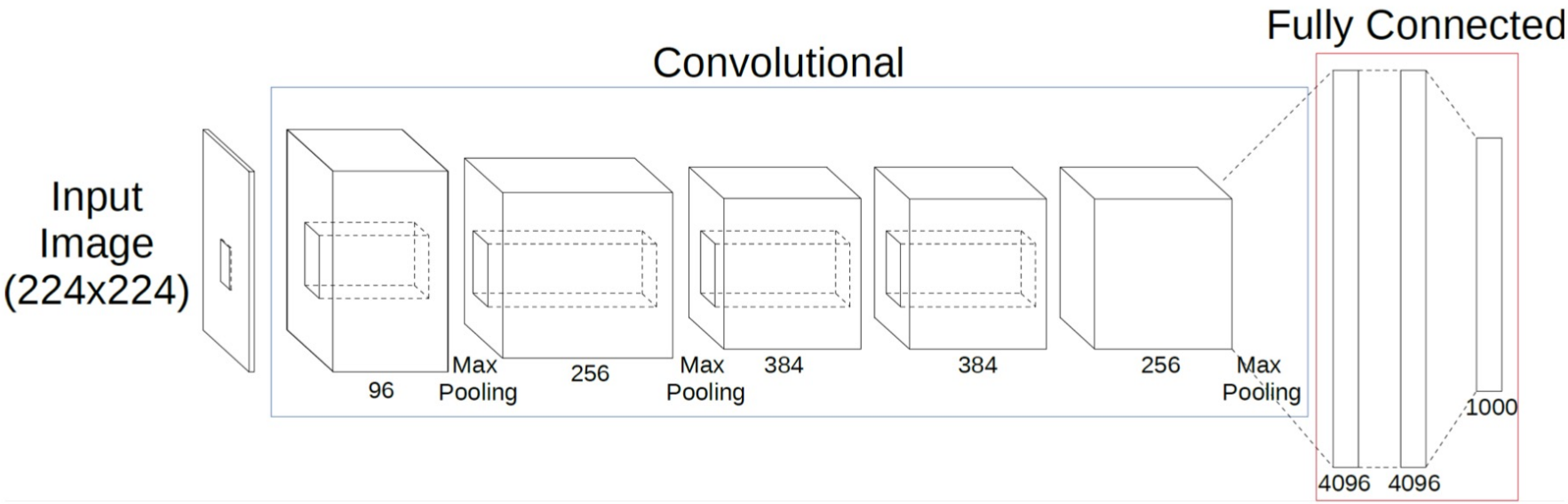
- Dropout is a technique for regularization that artificially sets random weights in the layer to zero
- Can be used to prevent over-fitting and increase robustness





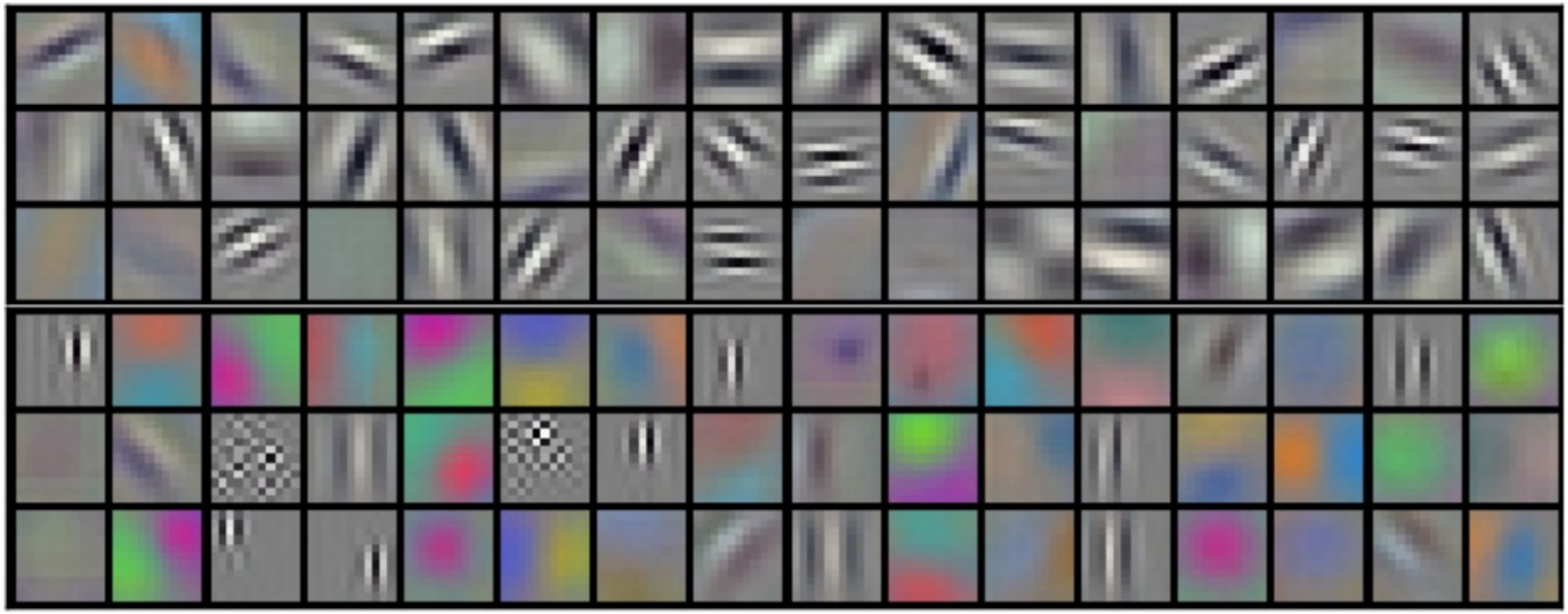
AlexNet (2012 ILSRVC Winner)

- AlexNet was the winning entry to the 2012 Imagenet Large Scale Visual Recognition Challenge
- It incorporated combinations of convolutional layers, fully connected layers, pooling and drop-out to train an 8-layer network for 1000 class recognition
- It contains ~60 million parameters and was trained on 1.2 million images and took six days to train on 2 GPUs



AlexNet (2012 ILSRVC Winner)

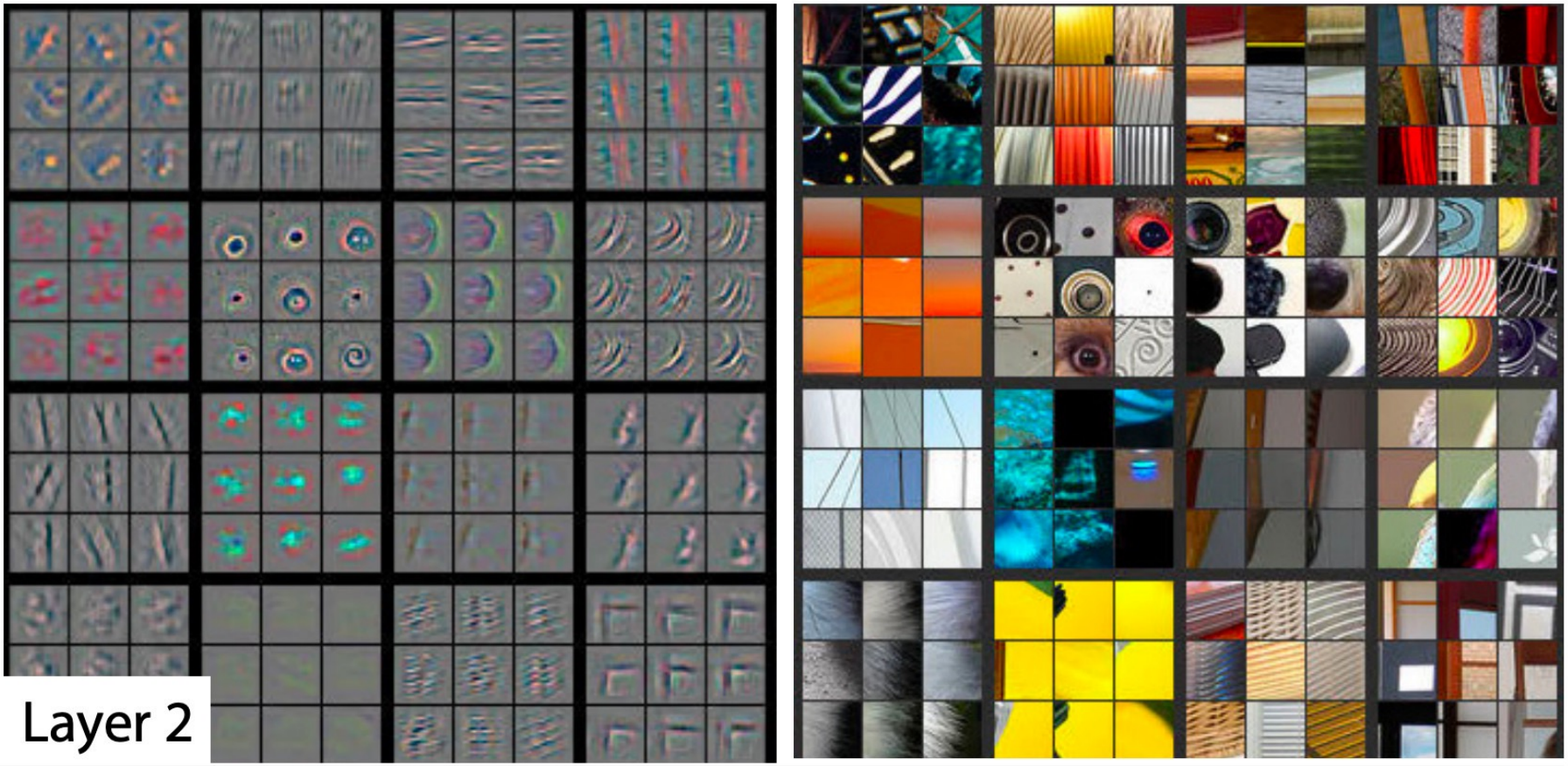
- AlexNet's convolutional layers end up learning filters of increasing levels of abstraction



First Convolutional Layer Filters
(11x11x3)

AlexNet (2012 ILSRVC Winner)

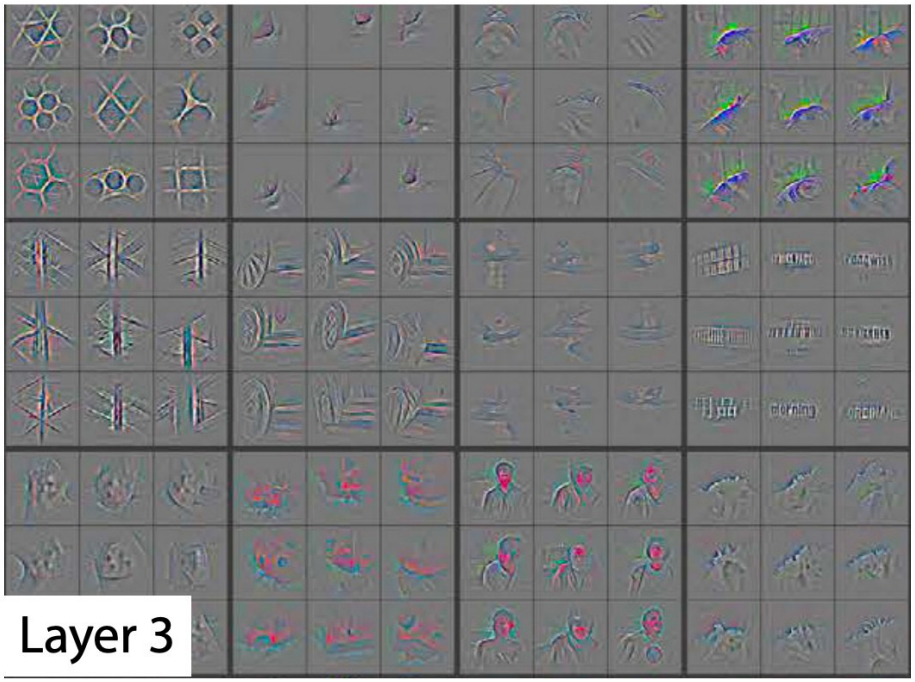
- AlexNet's convolutional layers end up learning filters of increasing levels of abstraction



Second layer responses

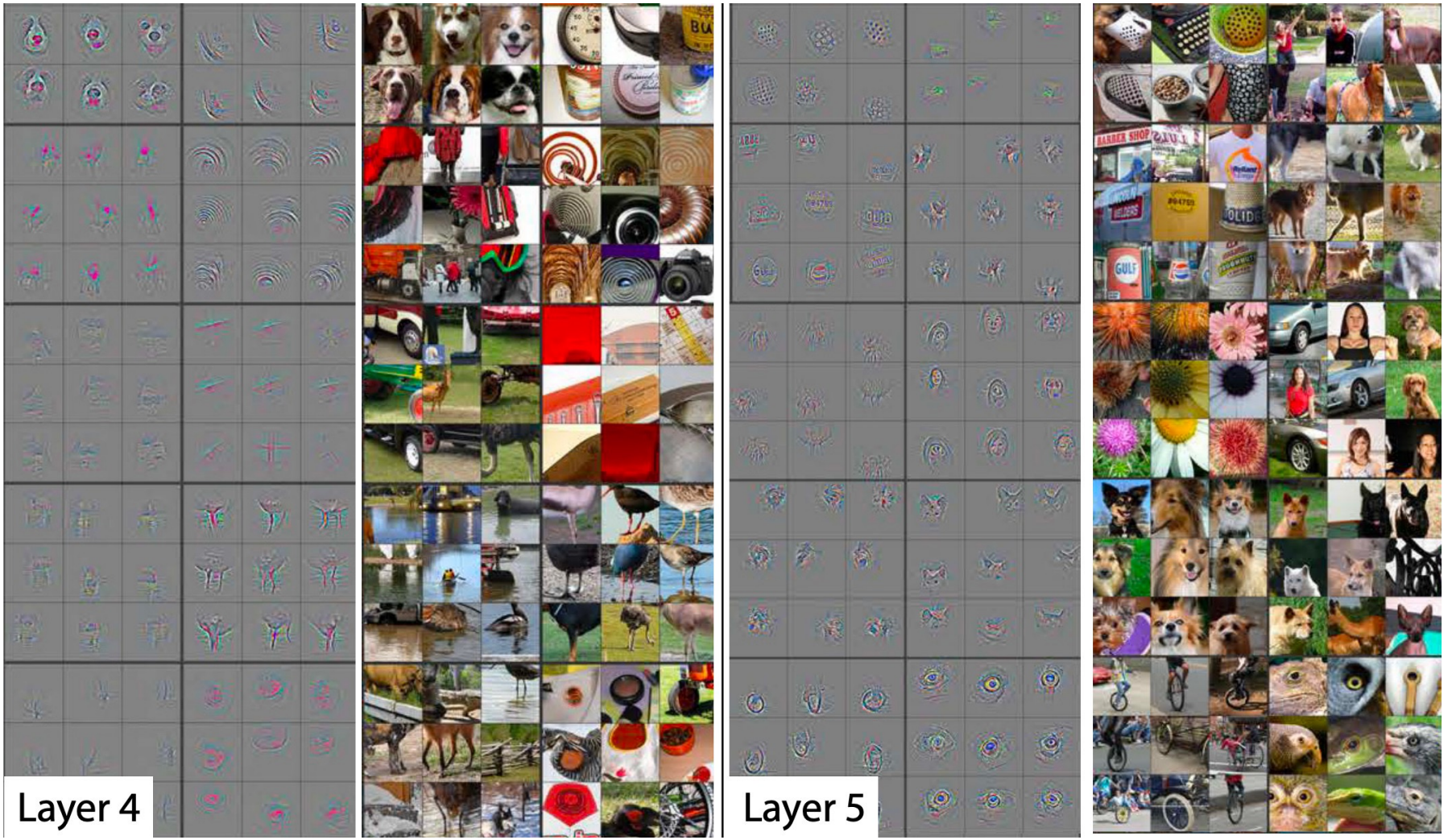
AlexNet (2012 ILSRVC Winner)

- AlexNet's convolutional layers end up learning filters of increasing levels of abstraction



Third layer responses

AlexNet (2012 ILSRVC Winner)



Fourth and Fifth layer responses

Further Reading and Next Week

- References:
  - Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning”, Nature 521, 2015.
  - R. Szeliski, “Computer Vision: Algorithms and Applications”, Springer, 2010

- Next Week:
  - Introduction to Group Major Projects
  - Further Deep Learning in Computer Vision
  - Introduction to OpenCV and Keras