Feed-forward Neural Network

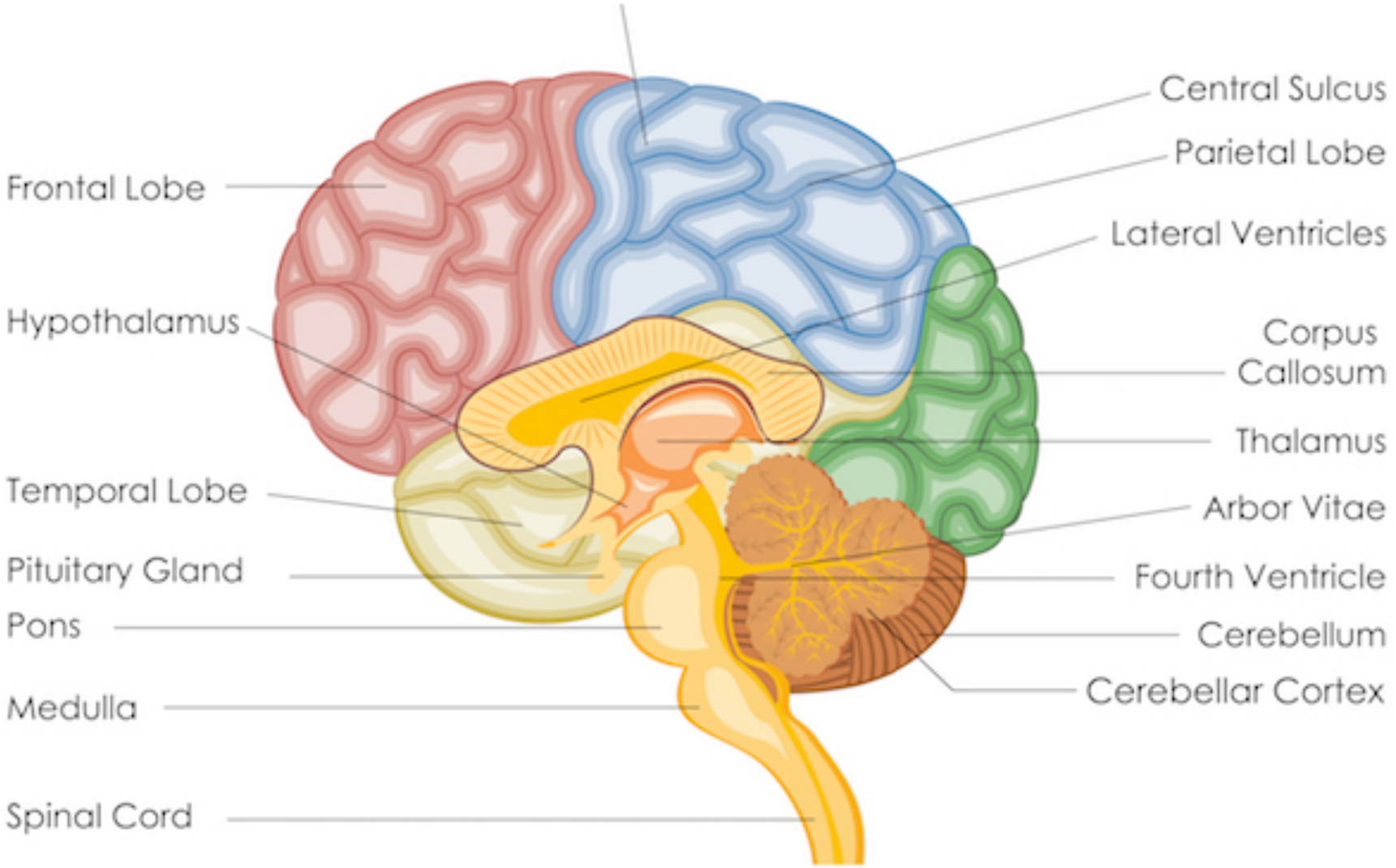
COMP3608 - Intelligent Systems
Inzamam Rahaman

Looking to nature

- Recall that when we studied meta-heuristics that we said that nature can be good source of inspiration in trying to achieve rational behaviour

- Looked at implicit forms of intelligence
  - Natural Selection - GA and GP
  - Flocking Behaviour - PSO
- What about explicit forms of intelligence
  - What about…

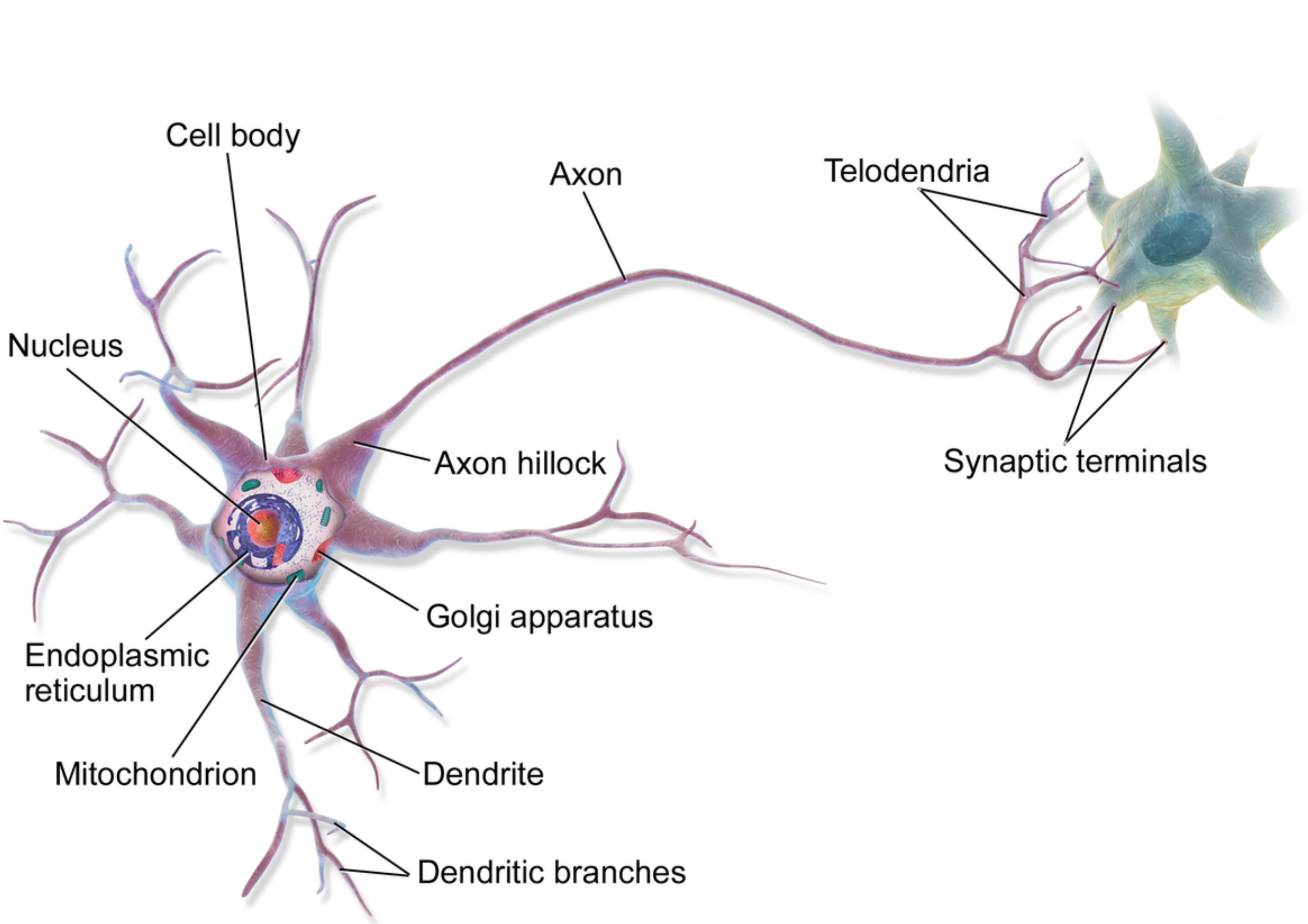
The Brain!



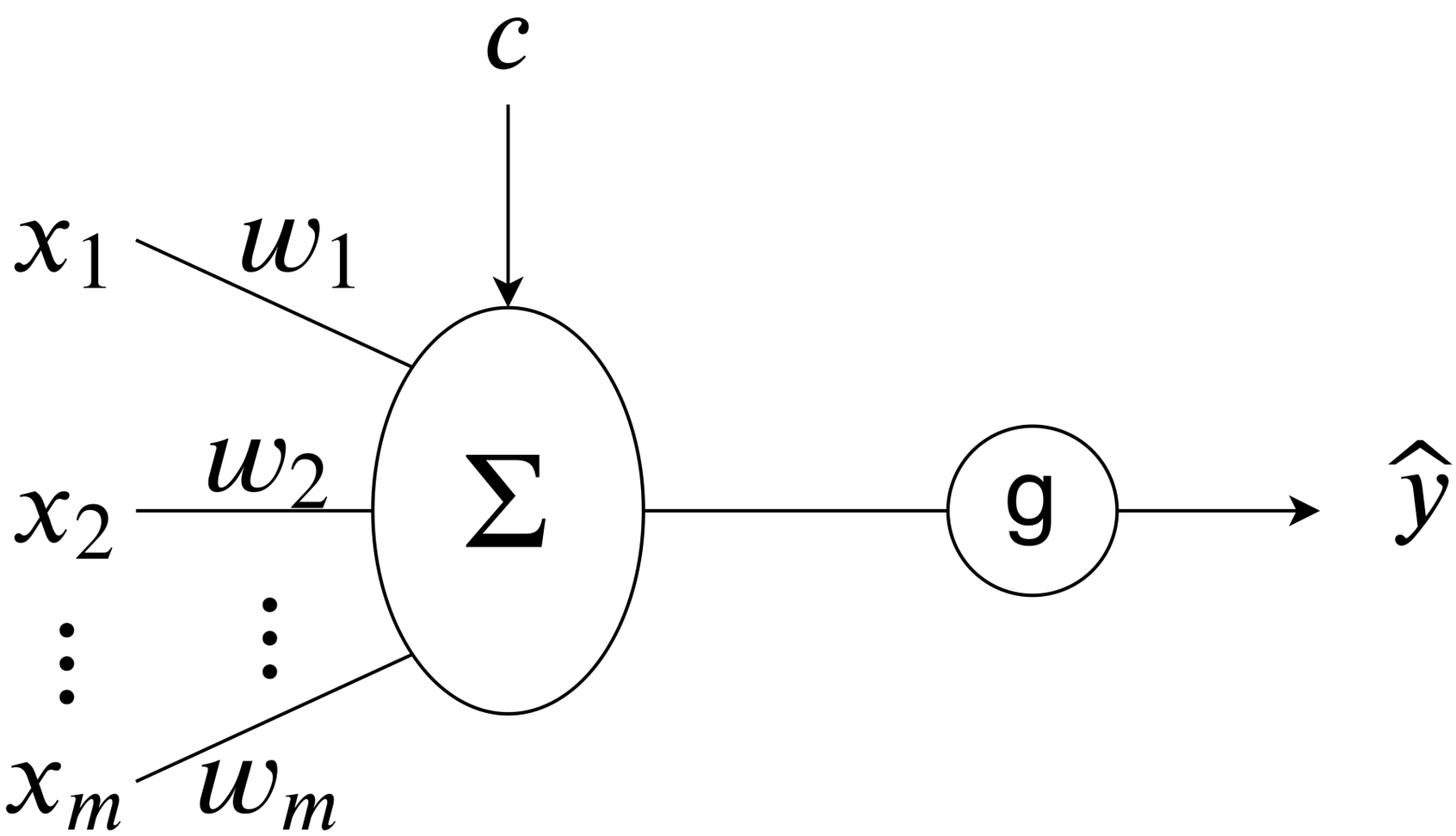
Not that one though - this one!

Brain

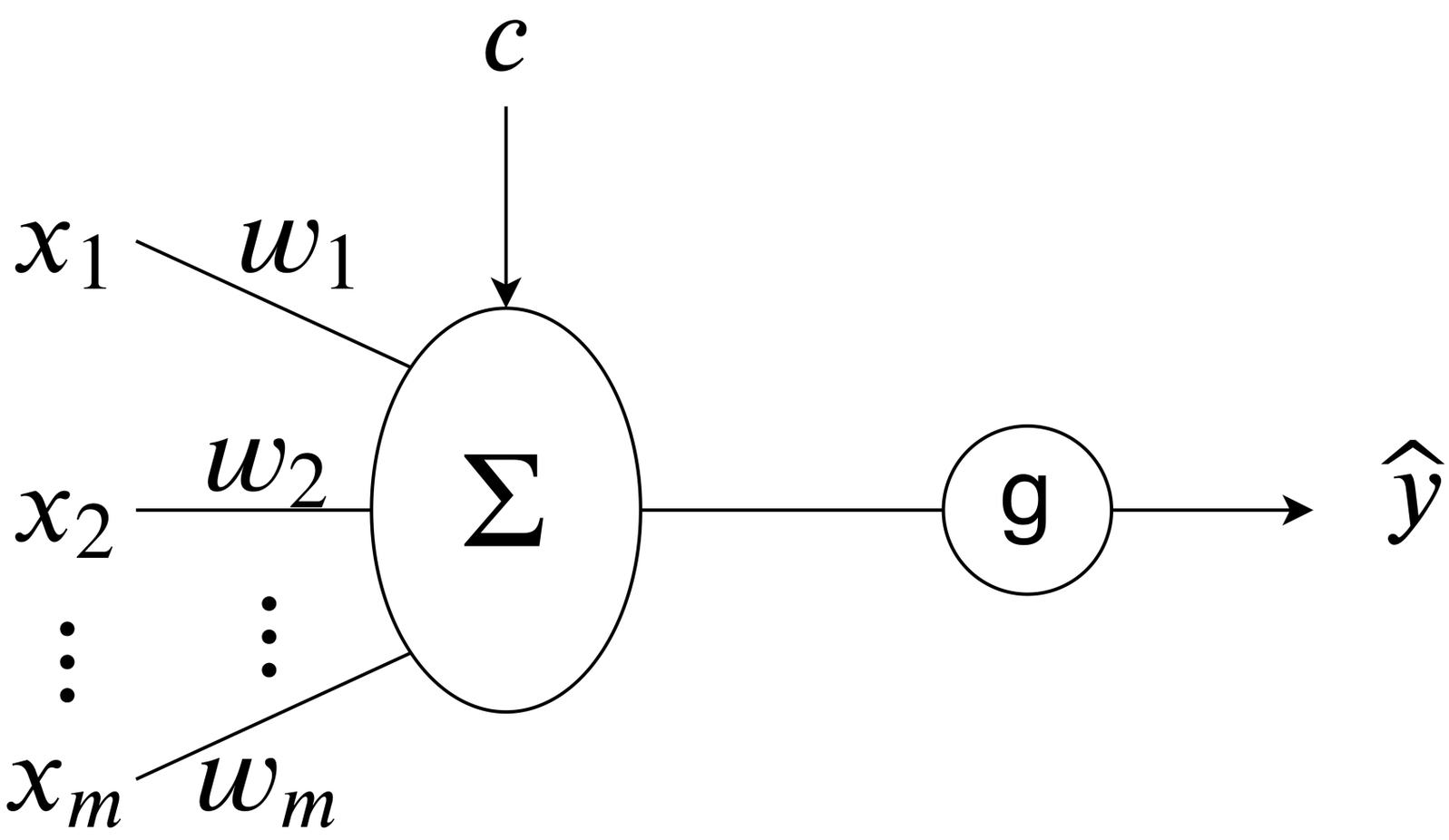
- Brain considered most adaptable form of intelligence known
- Don’t understand a lot about the brain
- Probably understand the motions of celestial bodies better than we understand the brain
- Nevertheless, we can draw upon our crude understanding of the brain to achieve intelligence


Cell body
Axon
Telodendria
Nucleus
Axon hillock
Synaptic terminals
Golgi apparatus
Endoplasmic
reticulum
Mitochondrion
Dendrite
Dendritic branches

The Perceptron



The Perceptron



$w_1, w_2, \ldots, w_m$ are the weights for features

$x_1, x_2, \ldots, x_m$ respectively; c is called the bias term;

g is called our activation function

Perceptron

- If we try to write out the perceptron mathematically we get

$$\hat{y}(x) = g(w^T x + c)$$

This should look sort of familiar

Activation Functions

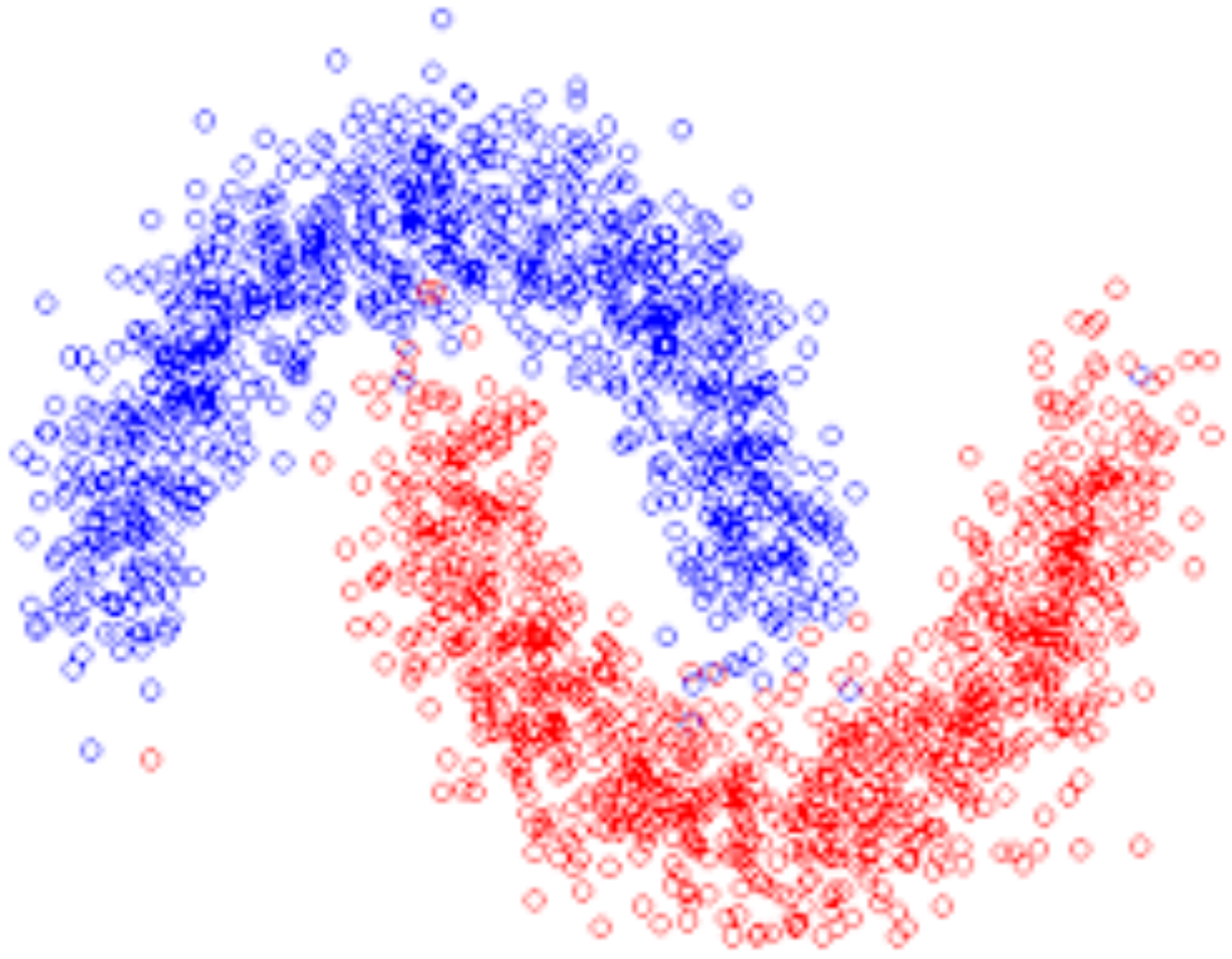
- Several possible choices for g, most non-linear in nature to better capture geometry of problem
- Examples:
  - $\sigma(x) = \frac{1}{1 + e^{-x}}$
  - $\mathrm{ReLU}(x) = \max(0,x)$
  - $\mathrm{id}(x) = x$
  - $\mathrm{softplus}(x) = \log(1 + e^{x})$
  - many others (research still ongoing)

Perceptron

- The perceptron is a clearly a linear model!
- Sigmoid and Linear Regression can be considered special cases!
- Is this sufficient?

Linear Models

- Linear Models cannot adequately model all phenomena
- E.g. the adjacent classification problem
- How do we handle such cases?



Non-linear cases

- One method is to use a non-linear basis transformation.

- Suppose each input vector is of size m, we define p non-linear transformations $\phi_1, \phi_2, \ldots \phi_p : \mathbb{R}^m \rightarrow \mathbb{R}$ that extract p new features that can then be fed into a linear model

  - Sometimes, we say that we have a transformation $\Phi : \mathbb{R}^m \rightarrow \mathbb{R}^p$ such that

$$\Phi(x) = \left[\phi_1(x), \phi_2(x), \ldots, \phi_p(x)\right]^T$$

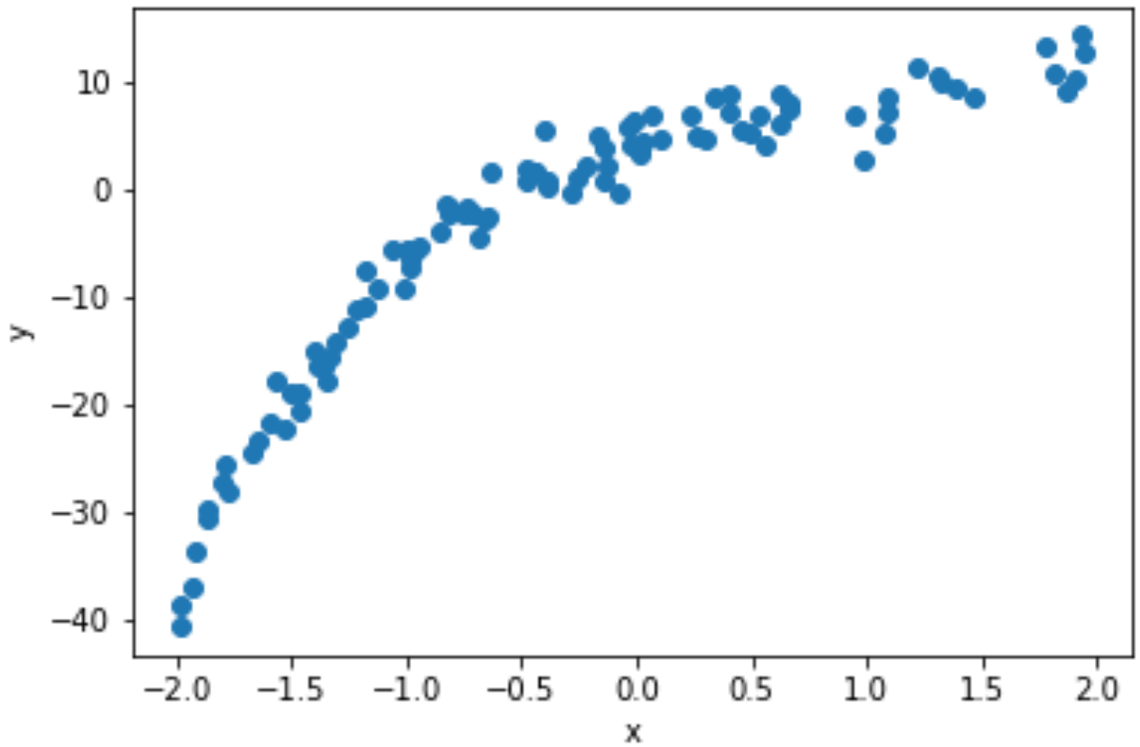
- Consider the case of fitting a polynomial of degree 3

Non-linear cases

- We can simply define 3 transformations $\phi_k(x) = x^k$ where k is the index of the transformation (i.e. taking on values of 1, 2, and 3)

- Using our transformations, we can then train a linear regression model that allows us to solve the problem well enough

- Similar processes can be used for other cases



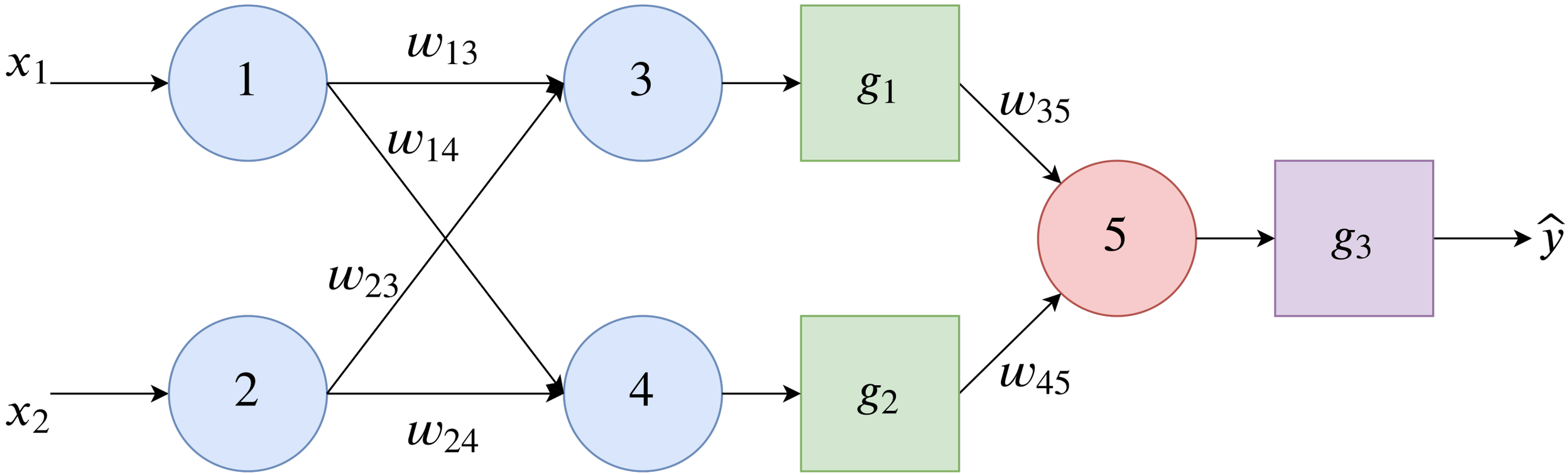
Learning Transformations

- Engineering these feature transformations is difficult and time consuming
- Can we automate this process?

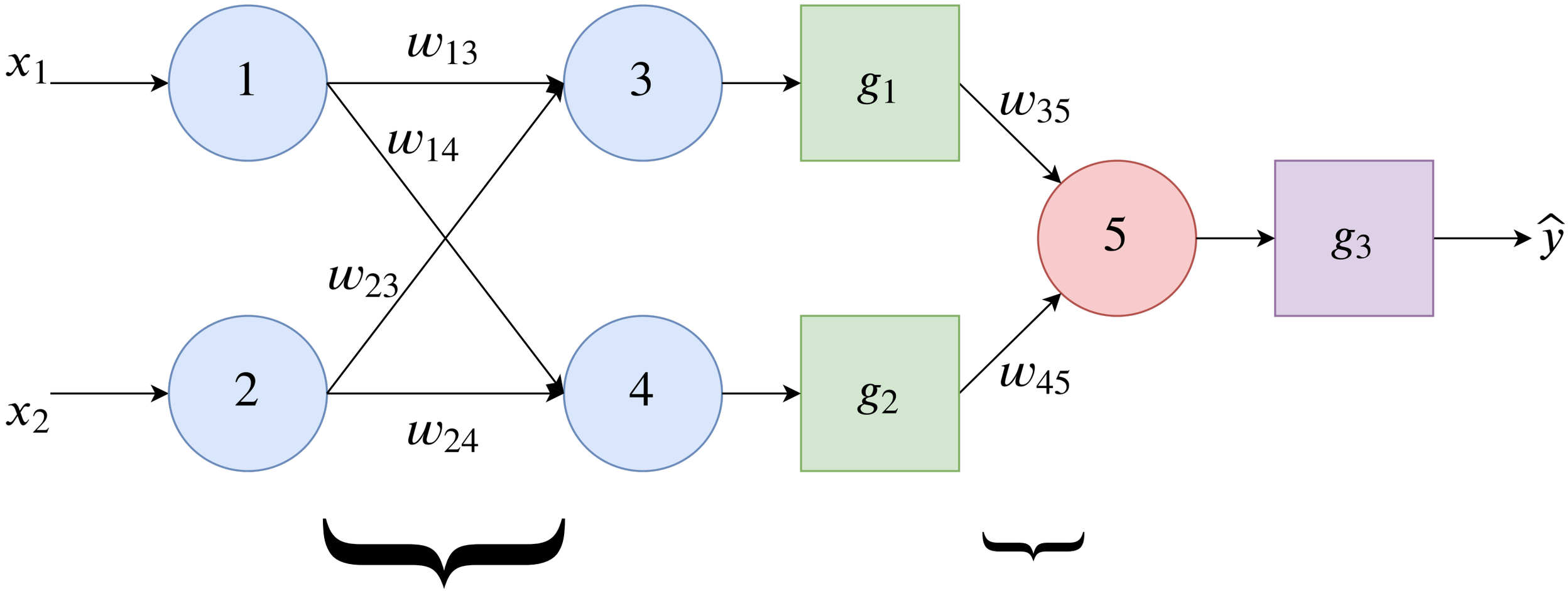
From Perceptrons to Brains

- A brain is made up of more than one perceptron
- Furthermore, it has layers of these perceptrons that feed data into each other
- What if we chain layers of perceptrons together
- Tune them together as part of learning process
- Earlier layers (called hidden layers) encode (non-linear) transformations
  - Hidden layers **always** use a non-linear activation function!
- Last layer (called the output layer) essentially performs logistic regression or linear regression on the data transformed by the hidden layers

Anatomy a Simple Feed-forward Neural Network



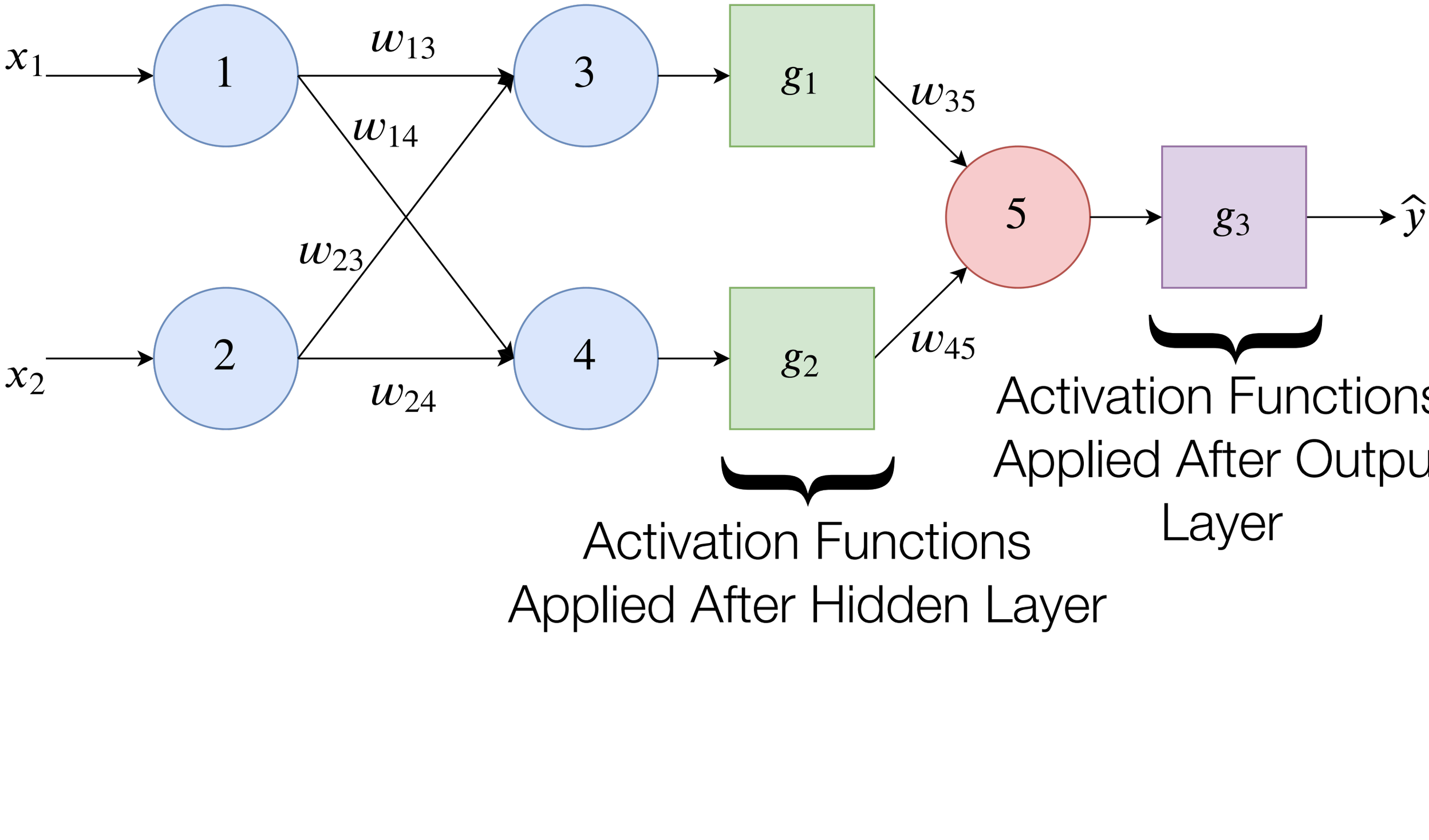
Anatomy a Simple Feed-forward Neural Network



Weights of hidden layer

Weights of output layer

Anatomy a Simple Feed-forward Neural Network

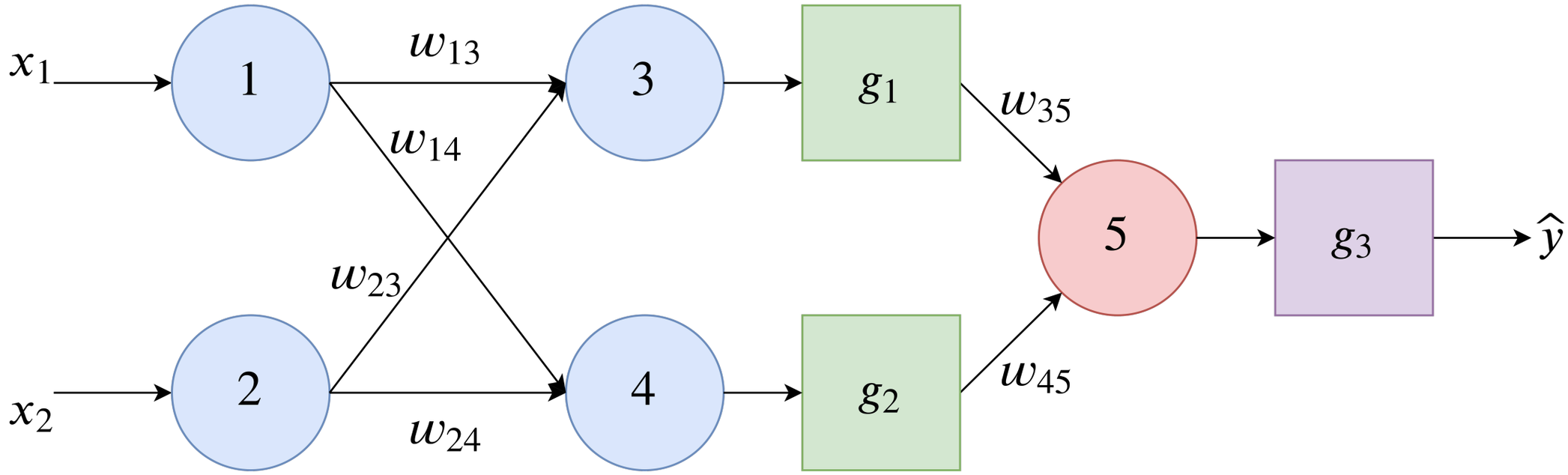


Feed-Foward Neural Networks

- NNs are represented as a sequence of matrix operations
- Just like with Linear Regression and Logistic Regression, NNs weights are randomly initialised
- Passing data through the neural network is called making the forward pass
- Loss is calculated using same procedure like linear models
    - Get predictions and compare predictions with actual

Forward Pass Example

- Suppose that we randomly initialised the weights of our network
- We want to train it to solve a regression problem
  - We use MSE Loss
- Let's consider a single data point for now
- $x = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$
- $y = 1$



$$w_{13} = 0.1, w_{23} = 0.2, w_{14} = 0.15, w_{24} = 0.1, w_{35} = 0.5, w_{45} = 0.5,$$

$$g_1, g_2, g_3 = \sigma(x)$$

$$o_3 = x_1 w_{13} + x_2 w_{23} = 2(0.1) + 3(0.2) = 0.8$$

$$o_4 = x_1 w_{14} + x_2 w_{24} = 2(0.15) + 3(0.1) = 0.6$$

$$z_3 = \sigma(0.8) = 0.69$$

$$z_4 = \sigma(0.6) = 0.65$$

$$o_5 = z_3 w_{35} + z_4 w_{45} = 0.69(0.5) + 0.65(0.5) = 0.67$$

$$z_5 = \hat{y} = \sigma(0.67) = 0.66$$

Forward Pass

- We predicted 0.67 and expected 1.0
- MSE Loss is $(1.0 - 0.67)^2 = 0.11$
- So we made a prediction and we were able to assess how badly we were off
- How to adjust weights?

Backward Pass - Problems

- Recall that to use gradient descent, we need to use the gradient wrt to the loss

- For most parameters in the network, we don’t have access to that gradient directly!

- How can we compute it?

Chain Rule

- When the output of one function becomes the input of another, this is an example of function composition

- When two functions are composed, we can can calculate the gradient wrt to the initial input using the chain rule

$$\frac{d}{dx}\left[f\left(g\left(x\right)\right)\right] = f'(g(x))g'(x)$$

Chain Rule - Example

$$f(w) = \sigma(w^2)$$

$$f'(w) = 2w\sigma'(w^2)$$

$$f'(w) = 2w\sigma(w^2)(1 - \sigma(w^2))$$

Chain Rule - Example

$$f(w) = \sigma(w^2)$$

$$f'(w) = 2w\sigma'(w^2)$$

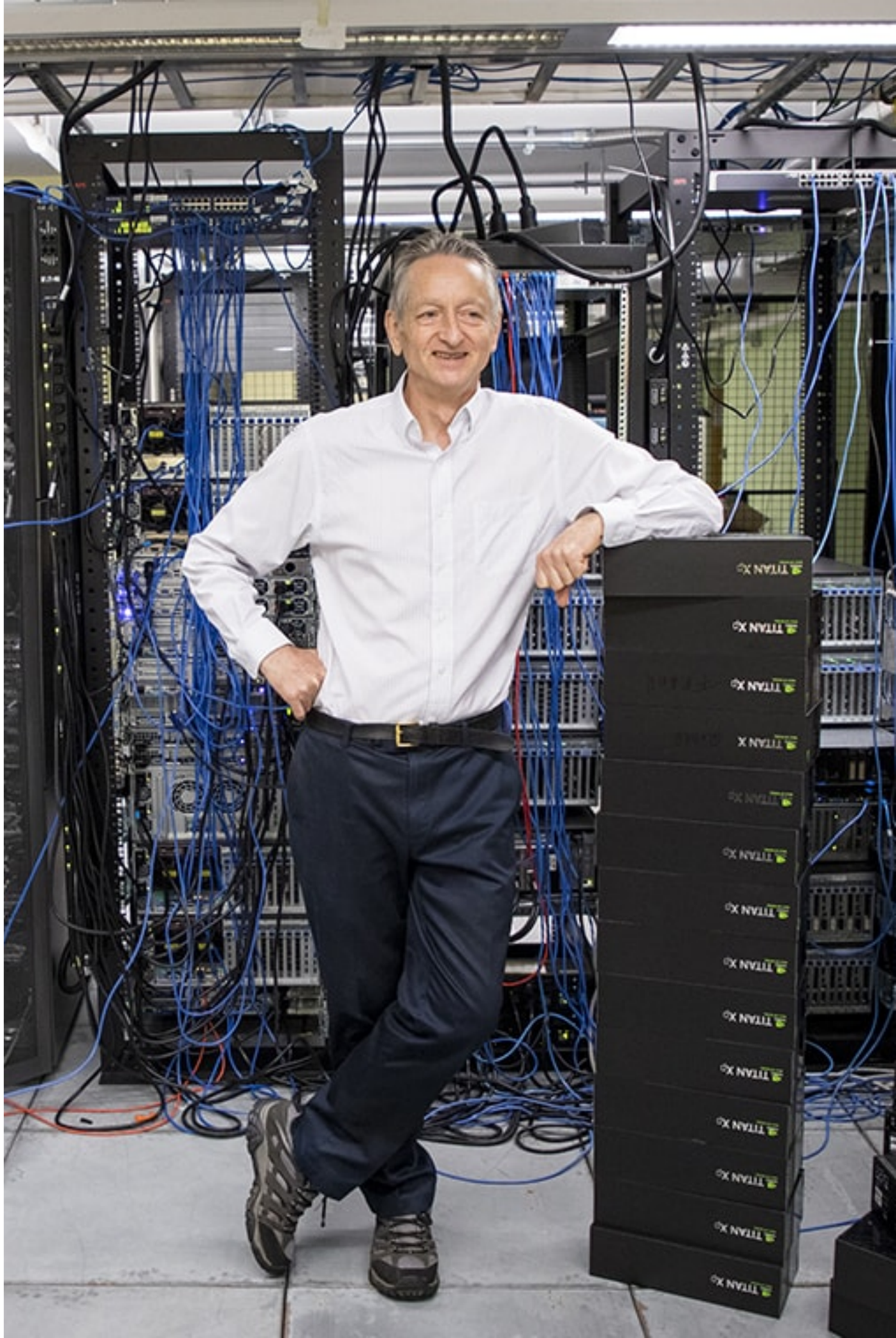
$$f'(w) = 2w\sigma(w^2)\sigma(1 - w^2)$$

Use similar process to compute gradients wrt to loss

Me and the boys about to receive the Turing award

Backpropogation

- Main method of training neural networks
- Exploits the chain rule repeatedly
- gradients flow back and accumulate
- A form of dynamic programming

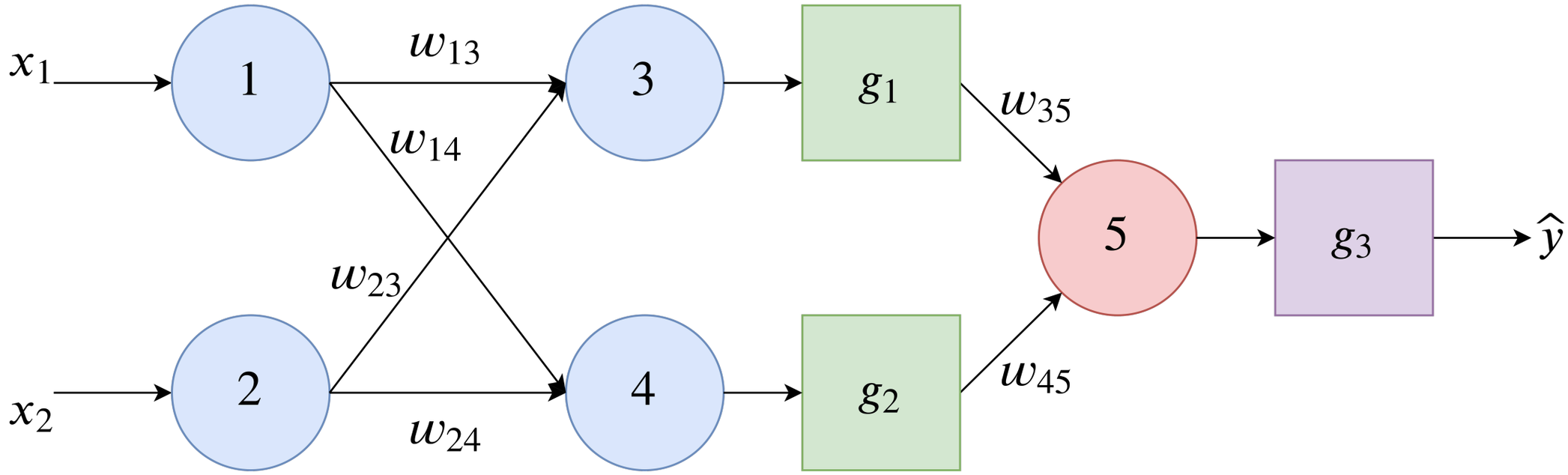


Backpropogation

1. Compute gradients wrt to inputs

2. Use chain rule to get gradients of parameters wrt to loss, going back through edges as needed to add gradients together

3. Use these gradients in gradient descent



$$w_{13} = 0.1, w_{23} = 0.2, w_{14} = 0.15, w_{24} = 0.1, w_{35} = 0.5, w_{45} = 0.5,$$

$$g_1, g_2, g_3 = \sigma(x)$$

$$o_3 = x_1 w_{13} + x_2 w_{23} = 2(0.1) + 3(0.2) = 0.8$$

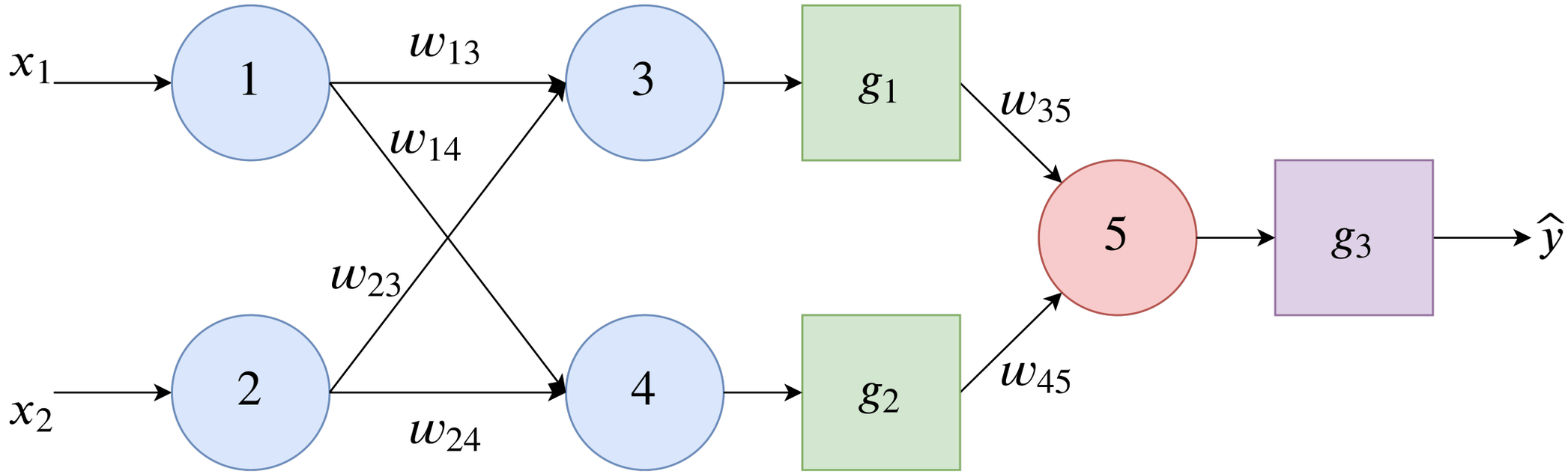
$$o_4 = x_1 w_{14} + x_2 w_{24} = 2(0.15) + 3(0.1) = 0.6$$

$$z_3 = \sigma(0.8) = 0.69$$

$$z_4 = \sigma(0.6) = 0.65$$

$$o_5 = z_3 w_{35} + z_4 w_{45} = 0.69(0.5) + 0.65(0.5) = 0.67$$

$$z_5 = \hat{y} = \sigma(0.67) = 0.66$$

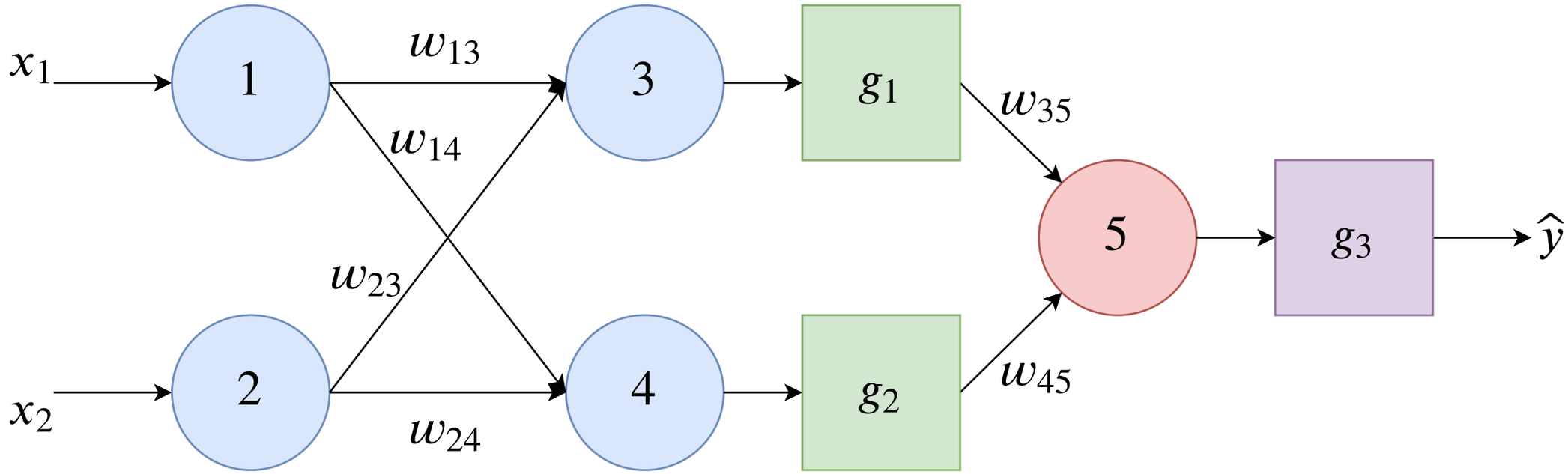


$$\frac{do_3}{dw_{13}} = x_1 = 2 \qquad \frac{do_3}{dw_{23}} = x_2 = 3$$

$$\frac{do_4}{dw_{14}} = x_1 = 2 \qquad \frac{do_4}{dw_{24}} = x_2 = 3$$

$$\frac{dz_3}{do_3} = \sigma'(o_3) = 0.65(1 - 0.65) = 0.23$$

$$\frac{dz_4}{do_4} = \sigma'(o_4) = 0.69(1 - 0.69) = 0.21$$



$$\frac{do_5}{dw_{35}} = z_3 = 0.65 \qquad \frac{do_5}{dw_{45}} = z_4 = 0.69$$

$$\frac{do_5}{dz_3} = w_{35} = 0.5 \qquad \frac{do_5}{dz_4} = w_{45} = 0.5$$

$$\frac{d\hat{y}}{o_5} = \sigma'(0.67) = 0.22$$

$$\frac{dL}{d\hat{y}} = -2(y - \hat{y}) = -2(1 - 0.66) = -0.68$$

$$\frac{dL}{d\hat{y}} = -0.68$$

$$\frac{dL}{do_5} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{do_5} = -0.68(0.22) = -0.15$$

$$\frac{dL}{dz_3} = \frac{dL}{do_5}\frac{do_5}{dz_3} = -0.15(0.5) = -0.08$$

$$\frac{dL}{dz_4} = \frac{dL}{do_5}\frac{do_5}{dz_4} = -0.15(0.5) = -0.08$$

$$\frac{dL}{dw_{35}} = \frac{dL}{do_5}\frac{do_5}{dw_{35}} = -0.15(0.65) = -0.10$$

$$\frac{dL}{dw_{45}} = \frac{dL}{do_5}\frac{do_5}{dw_{45}} = -0.15(0.69) = -0.10$$

$$\frac{dL}{do_4} = \frac{dL}{dz_4}\frac{dz_4}{do_4} = (-0.08)(0.21) = -0.02$$

$$\frac{dL}{do_3} = \frac{dL}{dz_3}\frac{dz_3}{do_3} = (-0.08)(0.23) = -0.02$$

$$\frac{dL}{dw_{13}} = \frac{dL}{do_3}\frac{do_3}{dw_{13}} = (-0.02)(2) = -0.04$$

$$\frac{dL}{dw_{23}} = \frac{dL}{do_3}\frac{do_3}{dw_{23}} = (-0.02)(3) = -0.06$$

$$\frac{dL}{dw_{14}} = \frac{dL}{do_4}\frac{do_4}{dw_{14}} = (-0.02)(2) = -0.04$$

$$\frac{dL}{dw_{24}} = \frac{dL}{do_4}\frac{do_4}{dw_{24}} = (-0.02)(3) = -0.06$$

Update Step

- Now that we have the gradients, we can now apply the standard update step from gradient descent

- For our example, let $\alpha = 1.0$

  - In most cases, this is very high, we are only using it to illustrate the training process

$$w_{13} = w_{13} - \alpha \frac{dL}{dw_{13}}$$

$$w_{14} = w_{14} - \alpha \frac{dL}{dw_{14}}$$

$$w_{23} = w_{23} - \alpha \frac{dL}{dw_{23}}$$

$$w_{24} = w_{24} - \alpha \frac{dL}{dw_{24}}$$

$$w_{35} = w_{35} - \alpha \frac{dL}{dw_{35}}$$

$$w_{45} = w_{45} - \alpha \frac{dL}{dw_{45}}$$

Theoretical Properties

- Universal Approximation Theorem: a neural network with a single hidden layer can approximate any continuous function on a hidden compact subset of $\mathbb{R}$.

- Basically, nearly any function can, in principle, be learnt by a neural network

- In practice, doesn’t work out
  - No guarantee that we know the architecture
  - No guarantee we could ever find the weights needed

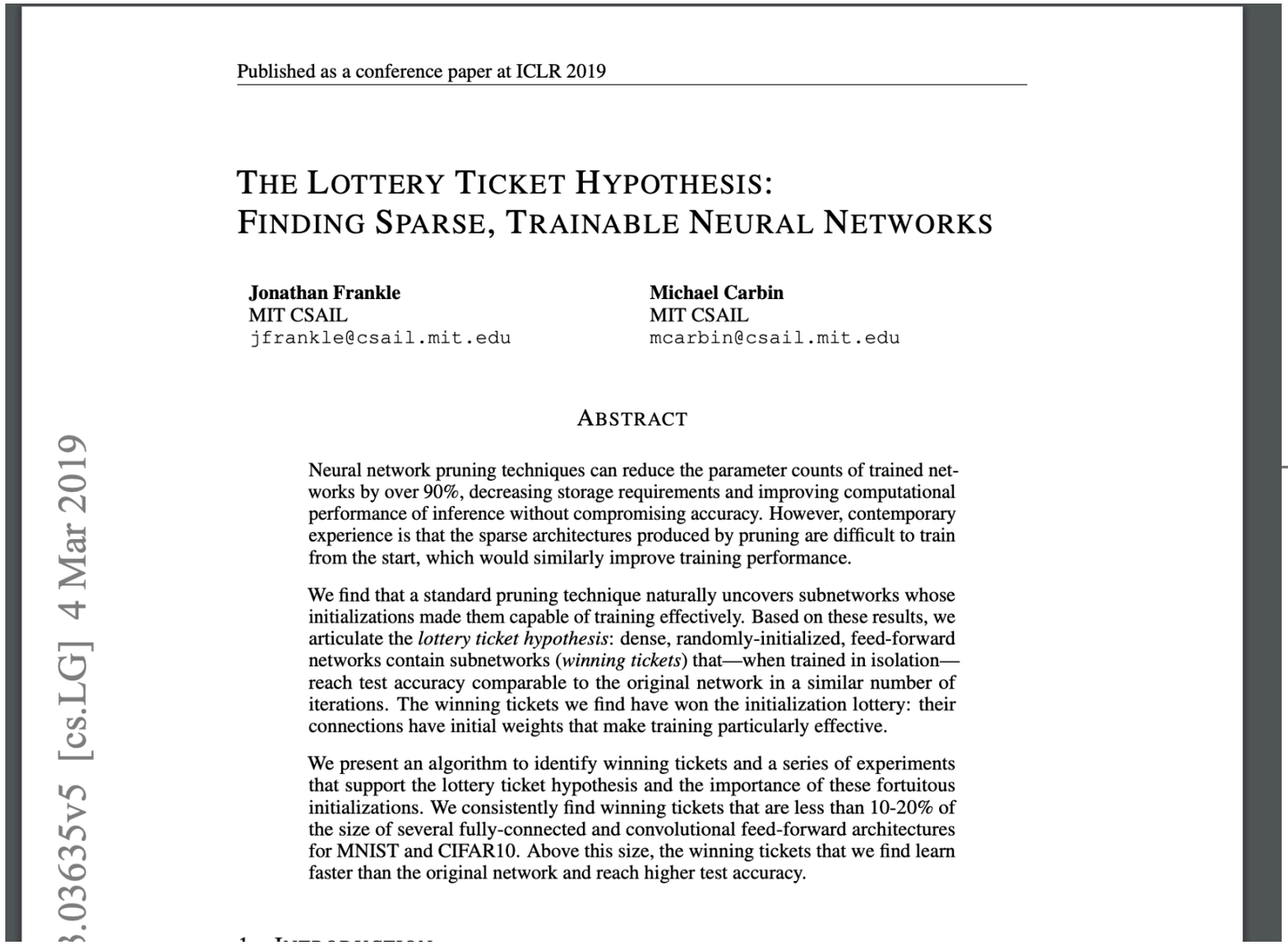
Computer Science > Machine Learning

Deep Learning and the Information Bottleneck Principle

Naftali Tishby, Noga Zaslavsky

(Submitted on 9 Mar 2015)

Deep Neural Networks (DNNs) are analyzed via the theoretical framework of the information bottleneck (IB) principle. We first show that any DNN can be quantified by the mutual information between the layers and the input and output variables. Using this representation we can calculate the optimal information theoretic limits of the DNN and obtain finite sample generalization bounds. The advantage of getting closer to the theoretical limit is quantifiable both by the generalization bound and by the network's simplicity. We argue that both the optimal architecture, number of layers and features/connections at each layer, are related to the bifurcation points of the information bottleneck tradeoff, namely, relevant compression of the input layer with respect to the output layer. The hierarchical representations at the layered network naturally correspond to the structural phase transitions along the information curve. We believe that this new insight can lead to new optimality bounds and deep learning algorithms.



Published as a conference paper at ICLR 2019

THE LOTTERY TICKET HYPOTHESIS: FINDING SPARSE, TRAINABLE NEURAL NETWORKS

Jonathan Frankle
MIT CSAIL
jfrankle@csail.mit.edu

Michael Carbin
MIT CSAIL
mcarbin@csail.mit.edu

ABSTRACT

Neural network pruning techniques can reduce the parameter counts of trained networks by over 90%, decreasing storage requirements and improving computational performance of inference without compromising accuracy. However, contemporary experience is that the sparse architectures produced by pruning are difficult to train from the start, which would similarly improve training performance.

We find that a standard pruning technique naturally uncovers subnetworks whose initializations made them capable of training effectively. Based on these results, we articulate the *lottery ticket hypothesis*: dense, randomly-initialized, feed-forward networks contain subnetworks (*winning tickets*) that—when trained in isolation—reach test accuracy comparable to the original network in a similar number of iterations. The winning tickets we find have won the initialization lottery: their connections have initial weights that make training particularly effective.

We present an algorithm to identify winning tickets and a series of experiments that support the lottery ticket hypothesis and the importance of these fortuitous initializations. We consistently find winning tickets that are less than 10-20% of the size of several fully-connected and convolutional feed-forward architectures for MNIST and CIFAR10. Above this size, the winning tickets that we find learn faster than the original network and reach higher test accuracy.

Professor: You should read more Theoretical ML Papers - they're not that hard.

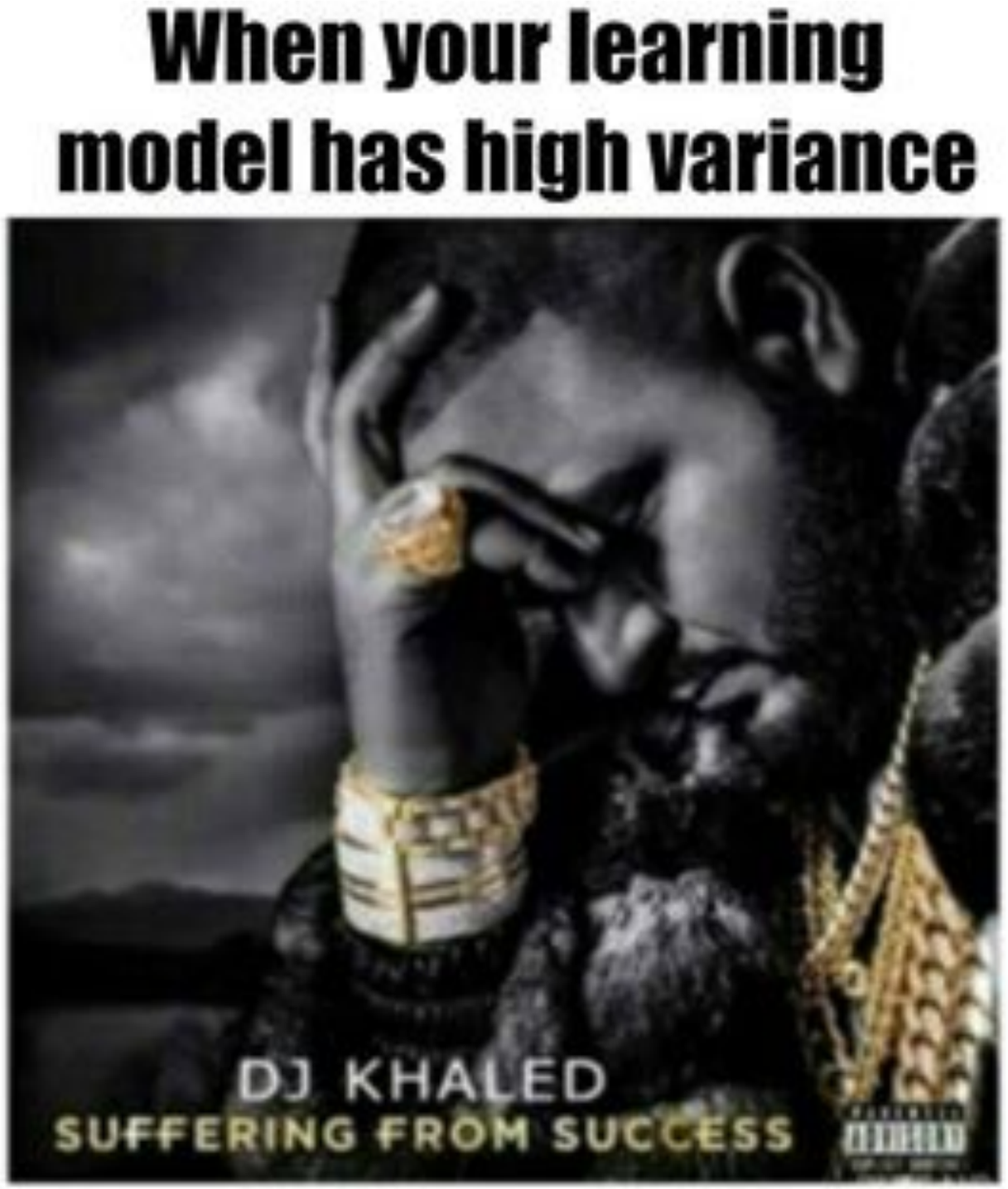
Theoretical ML Papers:

Practical Problems

- Neural Networks rely on large matrices
- Uses a lot of Linear Algebra
- Might be expensive computationally
- GPUs and related co-processors can help

Practical Problems

- Neural Networks are also prime examples of high variance models
- Keeping architectures small is vital
- But deciding architectures is still difficult
  - AutoML helps to some degree with very well-defined tasks



Practical Problems

- These types of networks are good for tabular data
- Many interesting types of data are not tabular
    - E.g. images
- How can we solve this?
- Will answer this next week :D