CompSci 367, tutorial 10

Neural networks

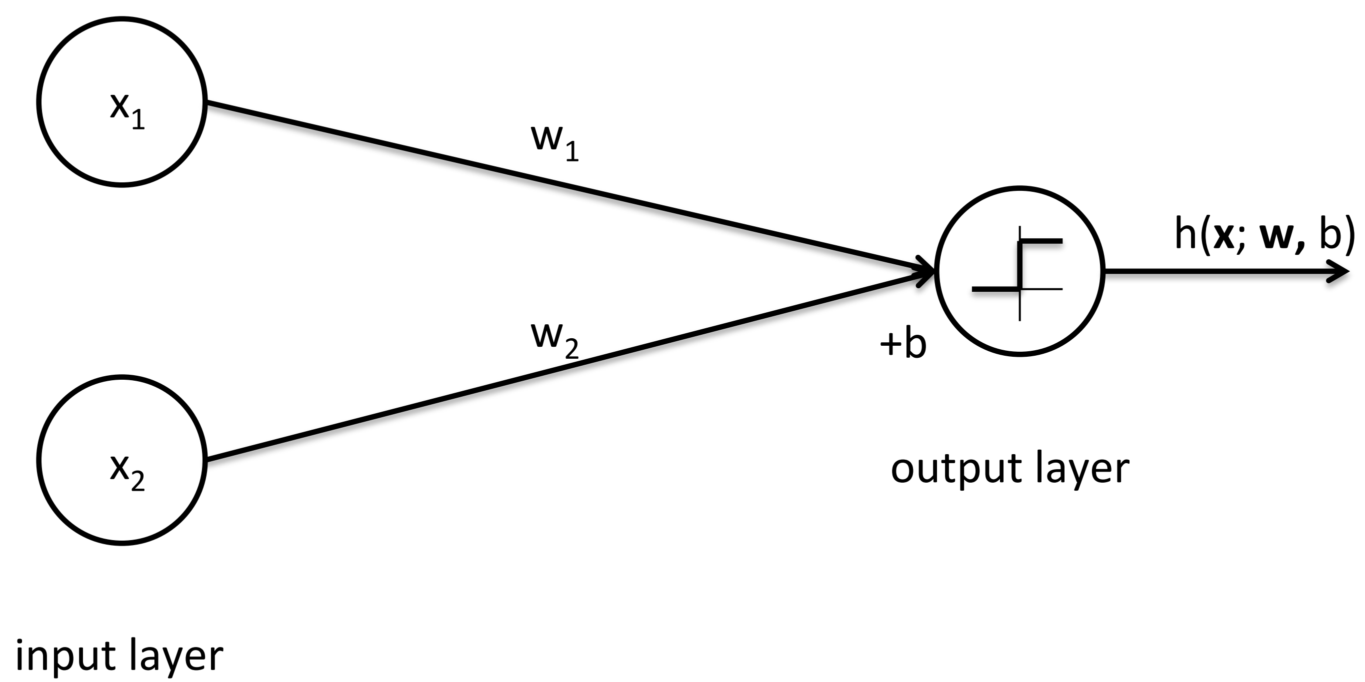
Alexander Donaldson

NN outline

- We will just talk about *feedforward neural networks* (probably the most common and useful)
- Take an input, map to an output
  - i.e. compute a mathematical function of the input
- Used for classification (output is in $[0,1]^n$) and regression (output is in $\mathbf{R}^n$)
- Uses a network of computational units (aka neurons) to build up a function

History: The 4 evolutions of NNs

#1: Perceptron (late 50s)



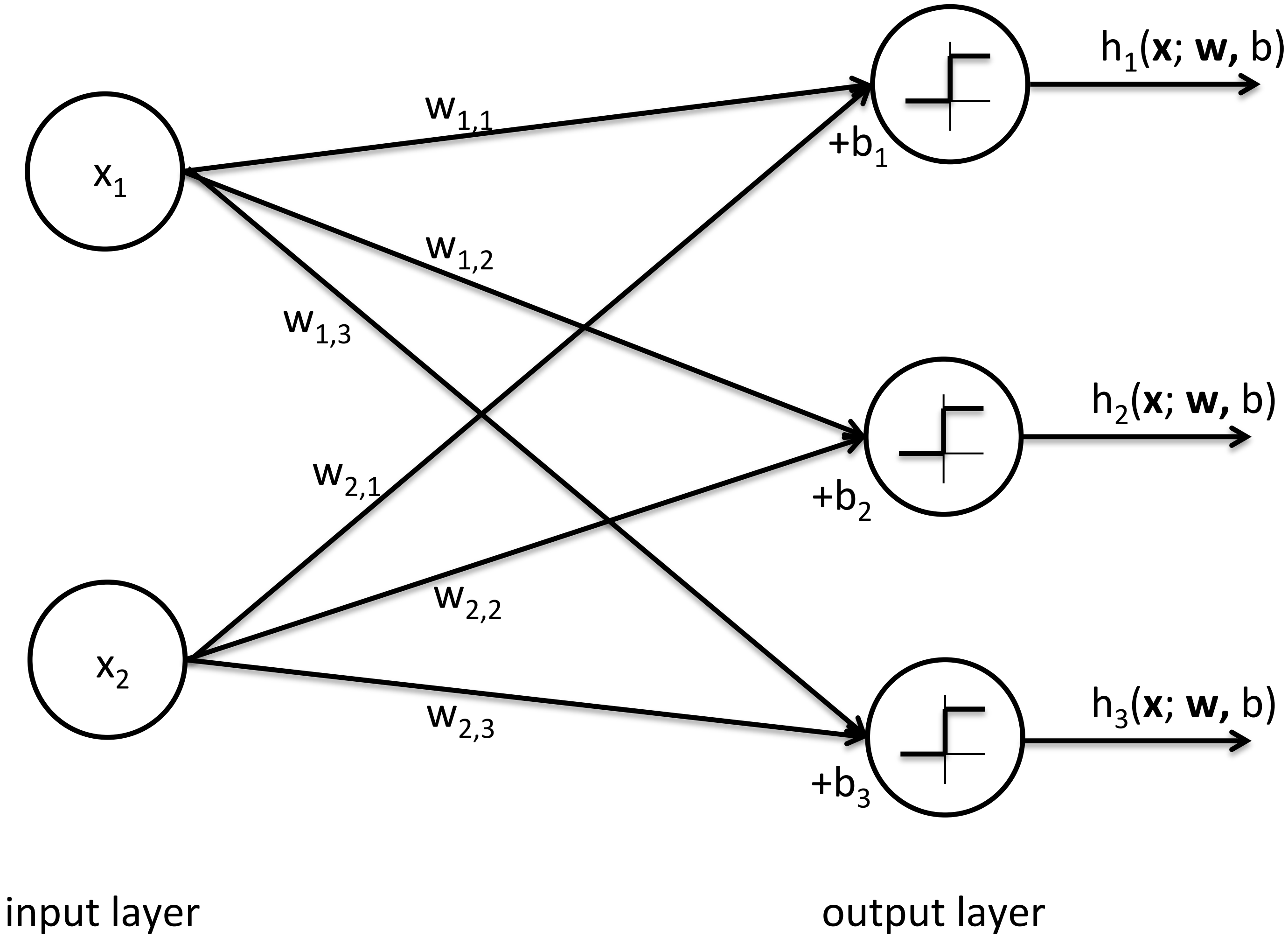
Perceptron hypothesis function

- Formula for our network:
  $h(\boldsymbol{x};\boldsymbol{w}, b)=H(w_1 x_1 + w_2 x_2 + b)$
- $H(z)= \begin{cases} 1, & z\geq 0 \\ 0, & z<0 \end{cases}$
- Want h(**x**) to be close to the target function y(**x**) – y(**x**) is the function we are trying to approximate
- Bias *b* sometimes called w_0

Hypothesis space

- Hypothesis space is the parameter space – has dimension equal to total number of weights and biases
- Space of real numbers
- E.g. one location in this space is:
  w = [1.5, -0.5], b = [1.0]

Multiple output units

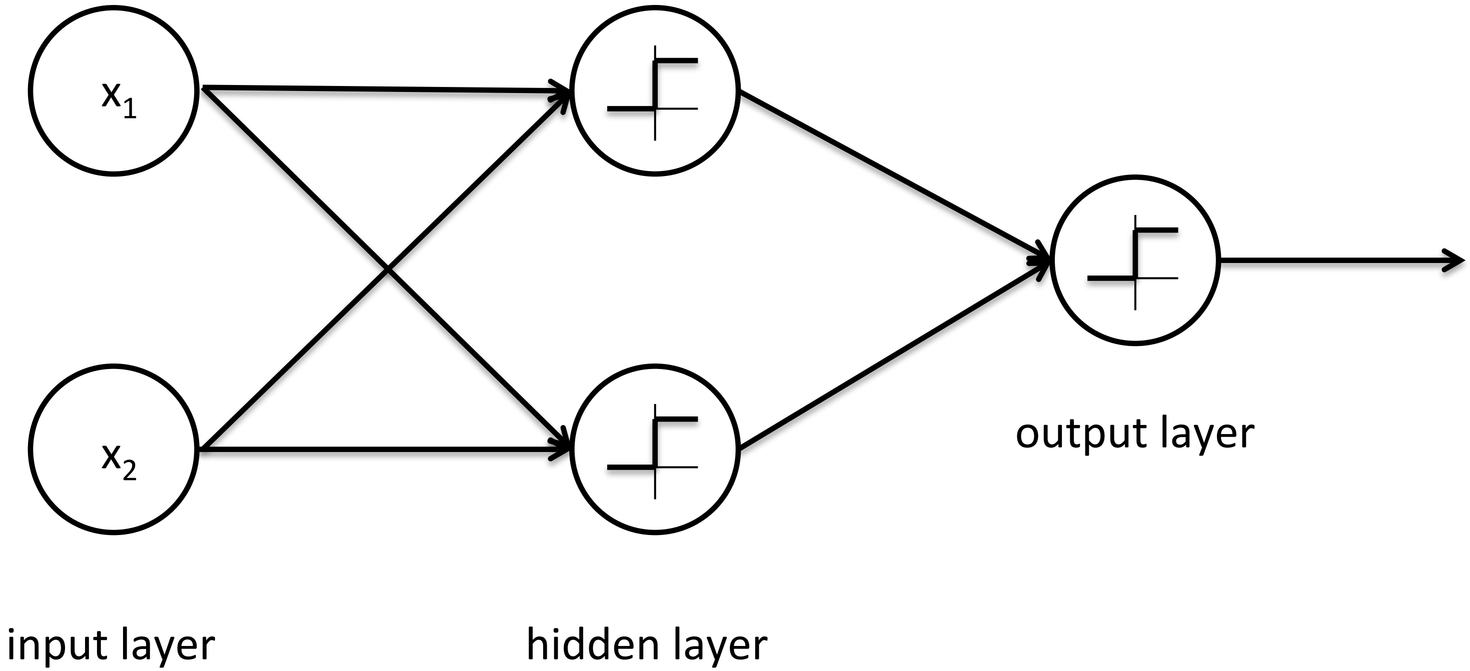


Perceptron hypothesis function

- General formula:
  $h_j(\boldsymbol{x};\boldsymbol{W},\boldsymbol{b})=H(\sum_i w_{ij} x_i +b_j)$
- Can also use theta to specify the set of all parameters (weights and biases):
  $h_j(\boldsymbol{x};\theta)$

Problem

- Can't model XOR, a very simple function
- Solution: add a *hidden layer* between the input and output layer – multi-layer perceptron

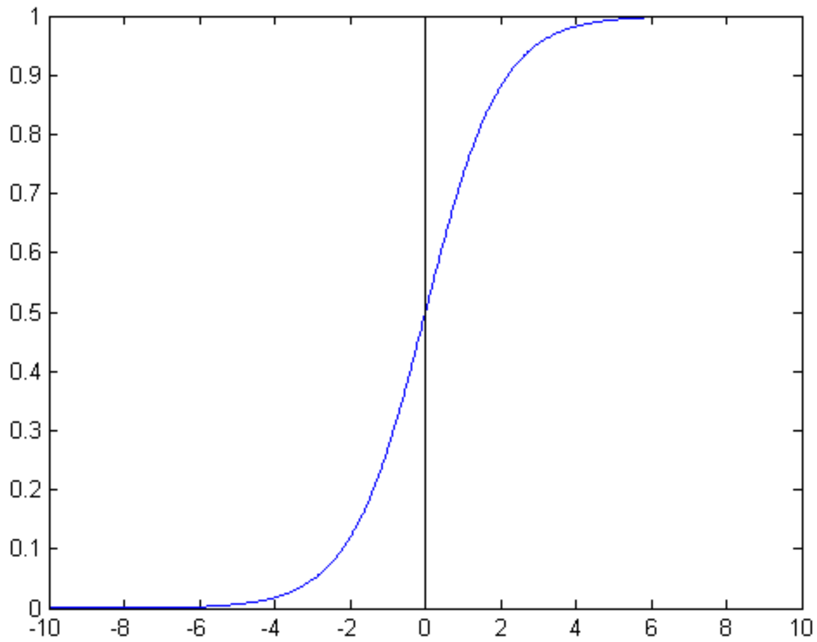


Another problem

- No fast way to choose weights to make h(**x**) close to y(**x**)
- People lost interest in perceptrons for many years

#2: Sigmoid units and gradient descent (70s/80s)

- Change the activation function from step function to logistic function (commonly referred to as sigmoid) –> gradient descent!



Gradient descent using backpropagation

- Would like to do gradient descent, i.e. change parameters to minimise some *loss function*
- The smaller the loss function, the closer h(**x**) is to y(**x**)
- Need to know how changing parameters affect h(**x**)
- Since we now have a smooth activation function – the gradient gives us meaningful information - we can *backpropagate* this gradient through the network, and find out the gradient of the loss function with respect to each parameter

Gradient descent formulae

- Loss function for regression is least squares: $l(\theta)=\sum_{k} (h(\boldsymbol{x}_k;\theta)-y_k)^2$
- For classification we use a different loss function, but it is the same idea

Gradient descent

- Remember in gradient descent we change each parameter depending on the magnitude and direction of the loss functions gradient with respect to that parameter
- e.g. if dl/dw_1 is negative and large, it means that the loss will decrease a lot if we increase w_1, so we increase w_1 a lot
- We find dl/dw_1 using backpropagation

Limits of modelling power

- To model an arbitrary function, need exponentially many units in the hidden layer
- Idea: add more layers – but now gradient descent gets stuck/is too slow (don't know which one)
- Another loss of interest in NNs

In the second NN winter…

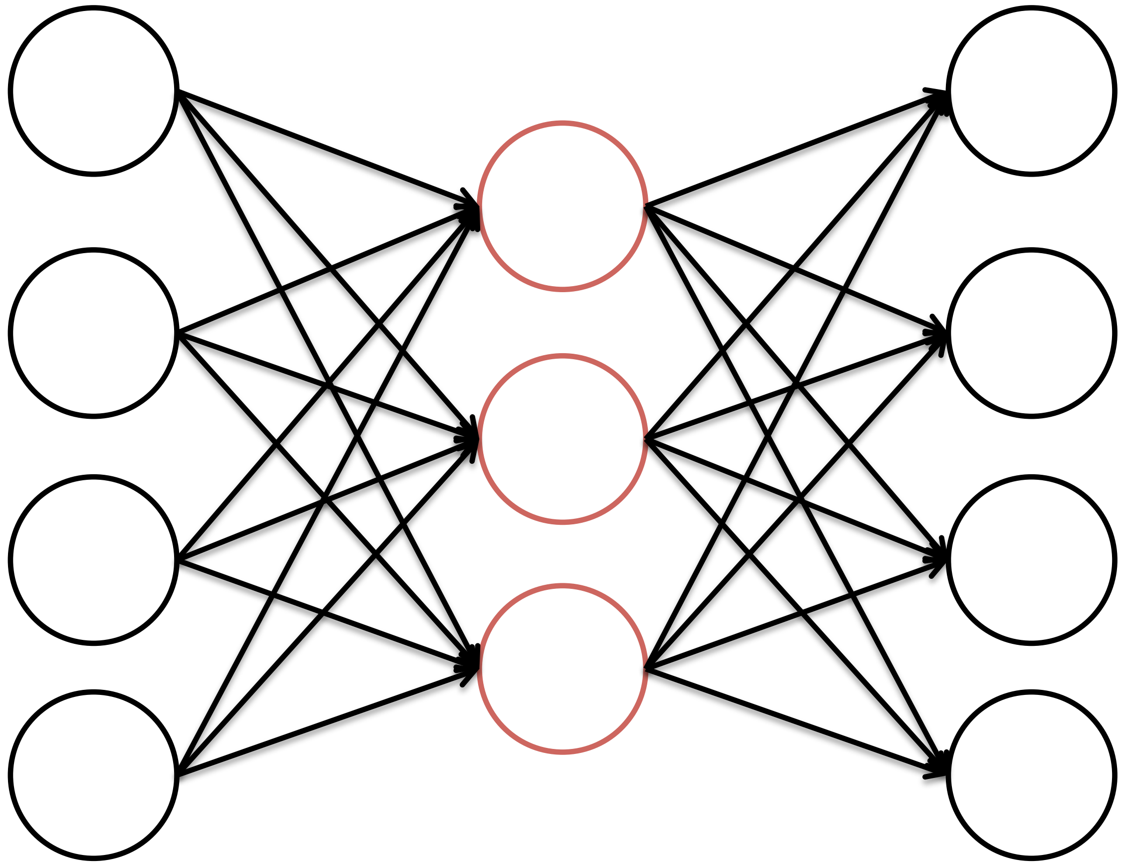
- Some researchers carry the flame through dark times
- Yann LeCun invents convolutional neural networks, which take into account spatial invariance of images (and invariances in other media types)

#3: Deep learning 1.0 (2006)

- Hinton and Salakhutdinov publish a paper showing that you can train *deep* neural networks (many hidden layers) by *unsupervised pre-training* using *autoencoders*
- (They used a generative autoencoder, but feedfoward autoencoders do the same thing…)

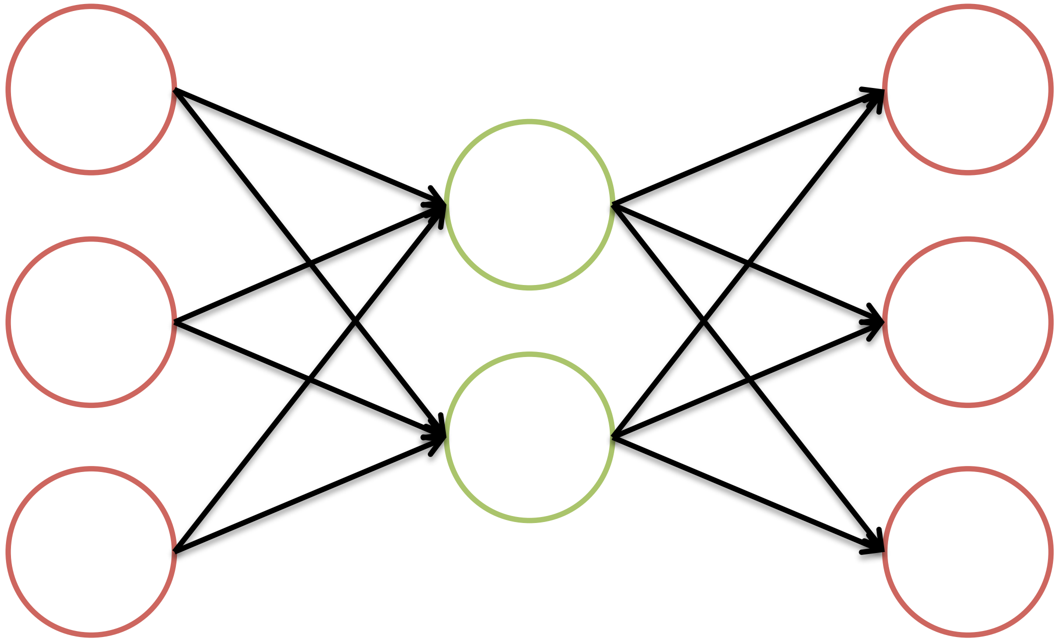
Feedforward autoencoder

- y(**x**) = **x**. Just trying to output the input.
- But less hidden units than input units, so hidden units are learning *features* of input



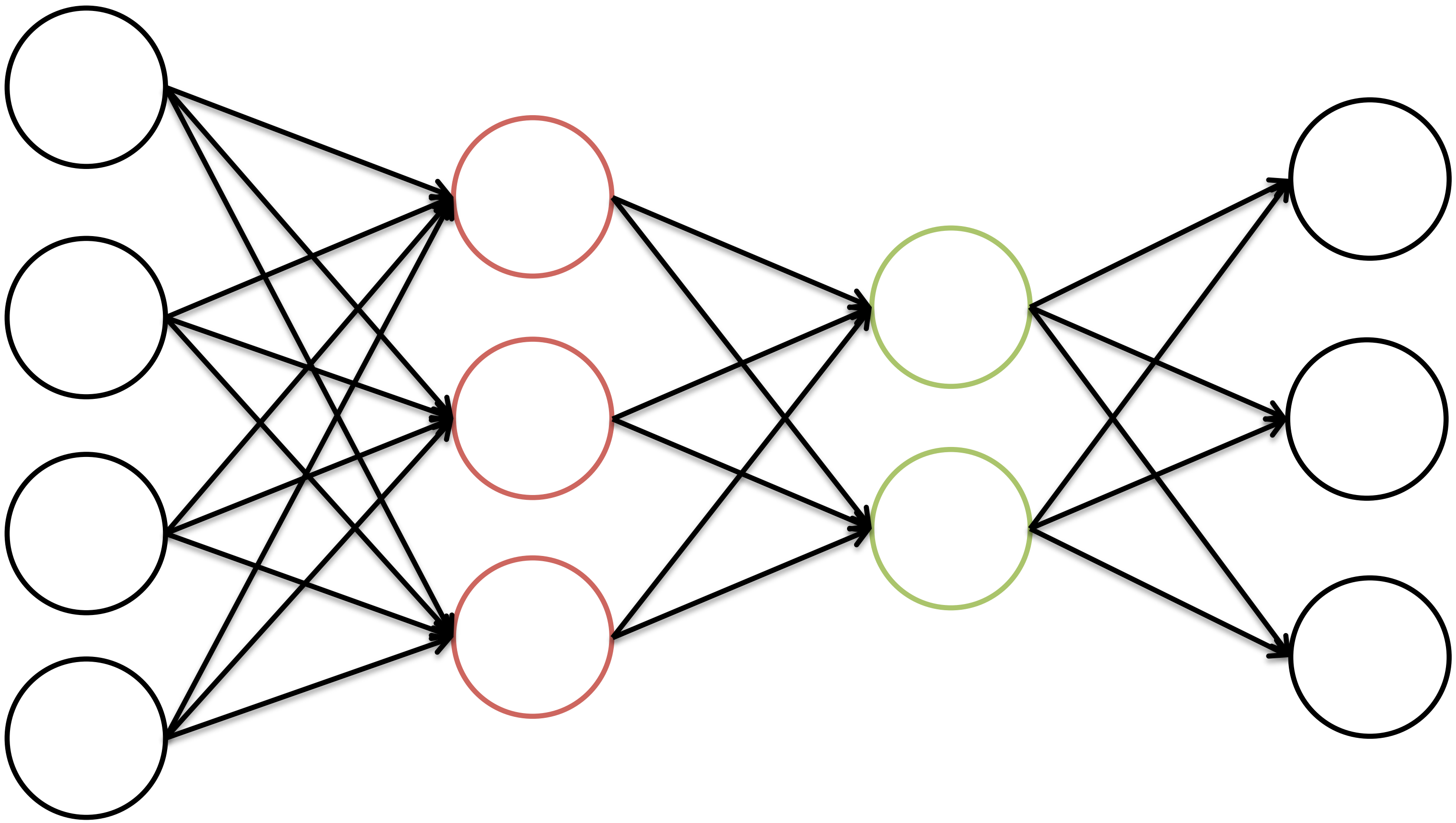
Stacked autoencoders

- Use hidden units of previous autoencoder as input to next autoencoder



Fine-tuning

- Finally, use weights from autoencoders to initialise a feedfoward network, then do supervised training using the labelled data

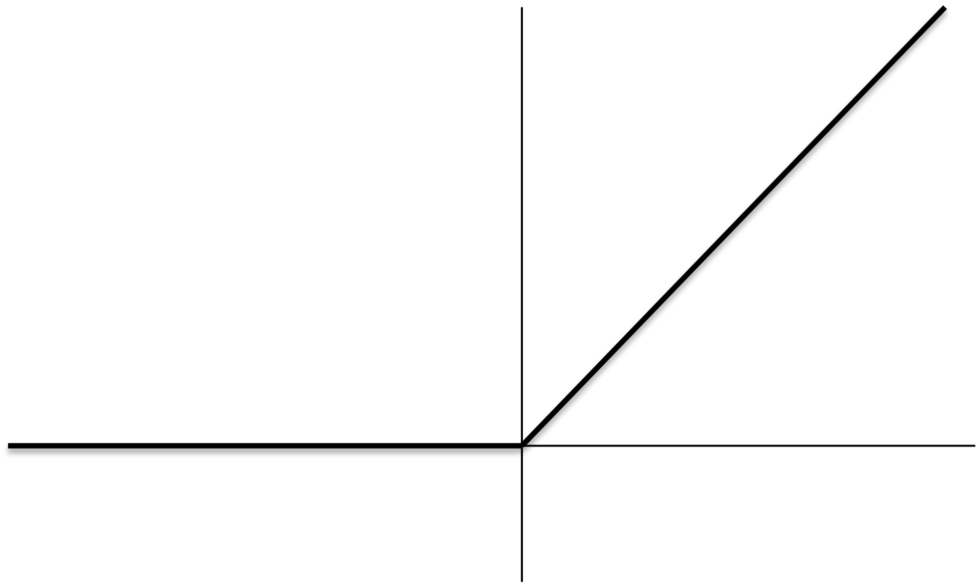


Advantages of pre-training

- Can use unlabelled data, which is often more abundant than labelled data
- Deep feedforward network can be initialised in a good starting position in parameter space – allows you to train deep networks

#4: Deep learning 2.0 (~2010)

- Problem with sigmoid is that the gradients get very small at either end, so gradient descent becomes slow – *vanishing gradients problem*
- Enter *rectified linear*:



Rectified linear activation function

- No vanishing gradients on right hand side
- Allows deep networks to be trained with gradient descent without using pre-training

Dropout

- Randomly remove units when training
- Acts as a *regulariser* – gives better performance on validation and test
- Why does it work? Maybe something to do with units not relying on other units, so learning more robust features

Practical aspects of training a deep neural network

Processor

- Within each layer, lots of calculations can be done independently (one for each unit in the next layer)
- Lends itself to parallelisation…
- Use GPUs!
- Coming soon, NPUs! Watch this space.

Validation set to avoid overfitting

- Split data into training, validation and test
- Train network on training set
- Monitor error on validation set. If it starts increasing, then stop training, because we are overfitting to the training set
- However, need to let it run for a bit because validation error can go upwards in the short-term but trend downwards in the long-term

Choosing hyperparameters

- Hyperparameters include initial learning rate, momentum, weight decay…
- Can also use validation set to choose these hyperparameters
- Try different values and look at which gives best performance on validation set after training, while also using the validation set for early stopping

Multiple runs

- Parameters are randomly initialised
- This means that the parameters can end up in a different location in parameter space
- If you are doing comparison between networks, it is good practice to do multiple runs to capture the variance in the final test error
- (However in practice, large networks take so long to train that they only get trained once)

Data augmentation

- More training data = more accurate classifier
- We know we can do certain transformations to examples and retain the same class e.g. a handwritten digit can be skewed slightly and still be the same digit
- So can artificially generate more training examples. This is called *data augmentation.*

The unreasonable effectiveness of deep learning

The power of deep learning

- Broke records by a long way on many image and speech datasets (still hold records)
- Hand-engineered features which took decades to develop have been made redundant
- All this from simply increasing the depth of the network
- Has led to some people labelling deep learning “unreasonably” effective

Re-using representation

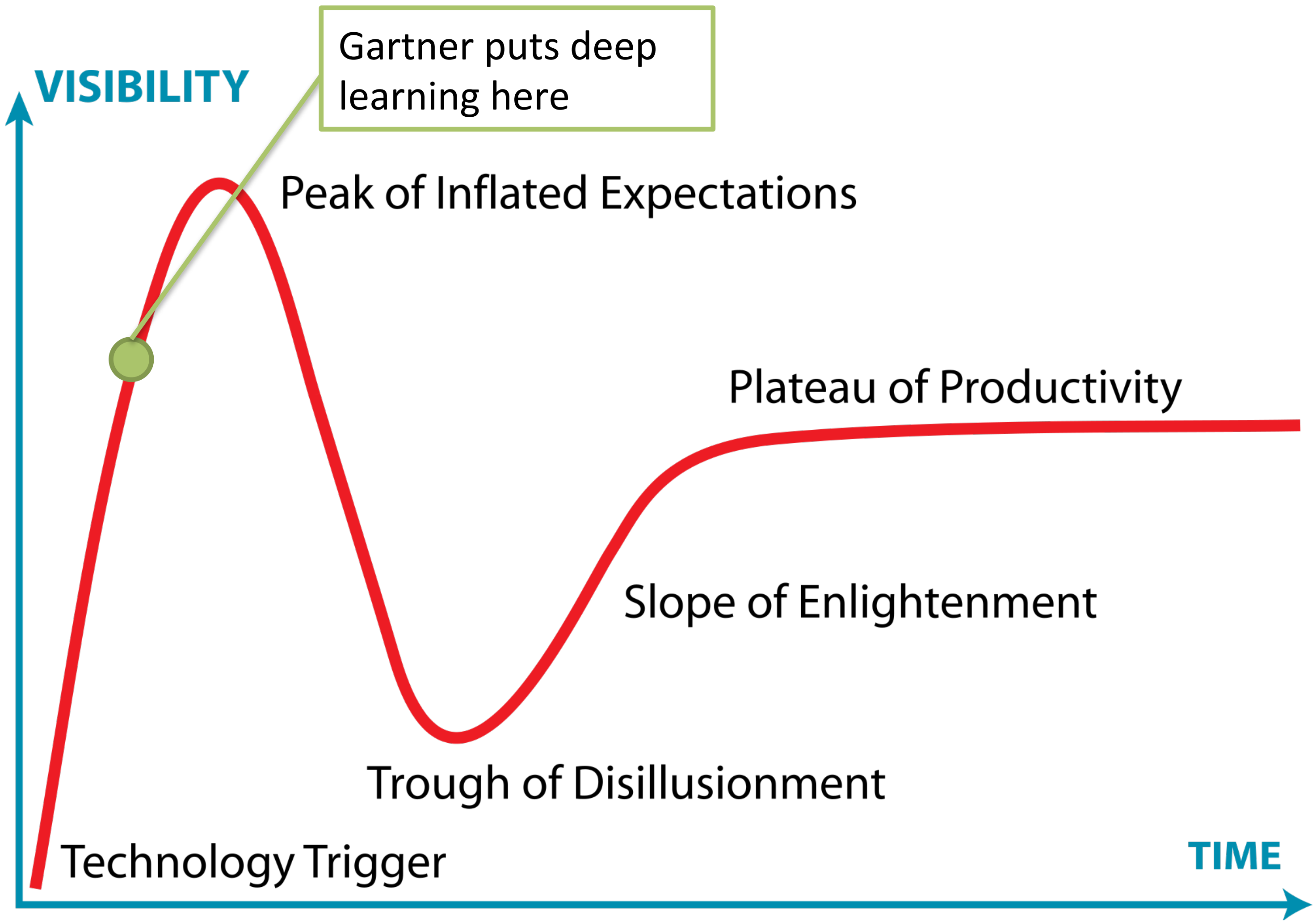
- Why so effective? Because complex features tend to share component features
- So multiple features can use the same feature from the previous layer
- Some theoretical results indicate that some functions that require an exponential number of units in a single-hidden-layer network only require a polynomial number of units in a two-hidden-layer network

Over-hype

Mainstream interest

- Deep learning has received high-profile mainstream press coverage
- Often hailed as a promising step towards strong AI
- Lots of attention from big tech companies - Google, Facebook and Baidu have all hired top experts from academia

Hype cycle



Gartner puts deep learning here
VISIBILITY
Peak of Inflated Expectations
Plateau of Productivity
Slope of Enlightenment
Trough of Disillusionment
Technology Trigger
TIME

Reality

- Over-hype is dangerous – has killed AI research in the past many times
- Is deep learning a silver bullet? No. Not for AI, not even for classification.
- We have a long, long way to go before we can achieve strong AI
- But deep learning has proved itself to be a highly effective method, so it is probably a step in the right direction

Post-graduate study

Honours

- Get an extra edge over the competition
- Multiple 700 level papers in AI
- Talk to Pat, Pat or Mike if interested

Neural networks

- Lots of open problems
- I’m happy to talk!
- Talk to Pat Riddle if you want to do research in this area