UNIVERSITY OF EDINBURGH
Business School

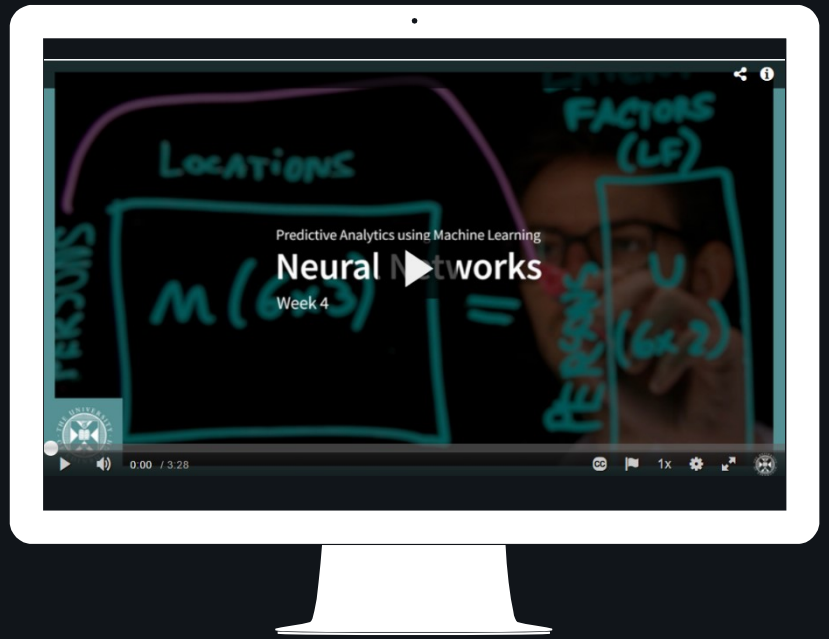
Predictive Analytics and Modelling of Data

CMSE11428 (2020-2021)

Dr Xuefei Lu
The University of Edinburgh Business School

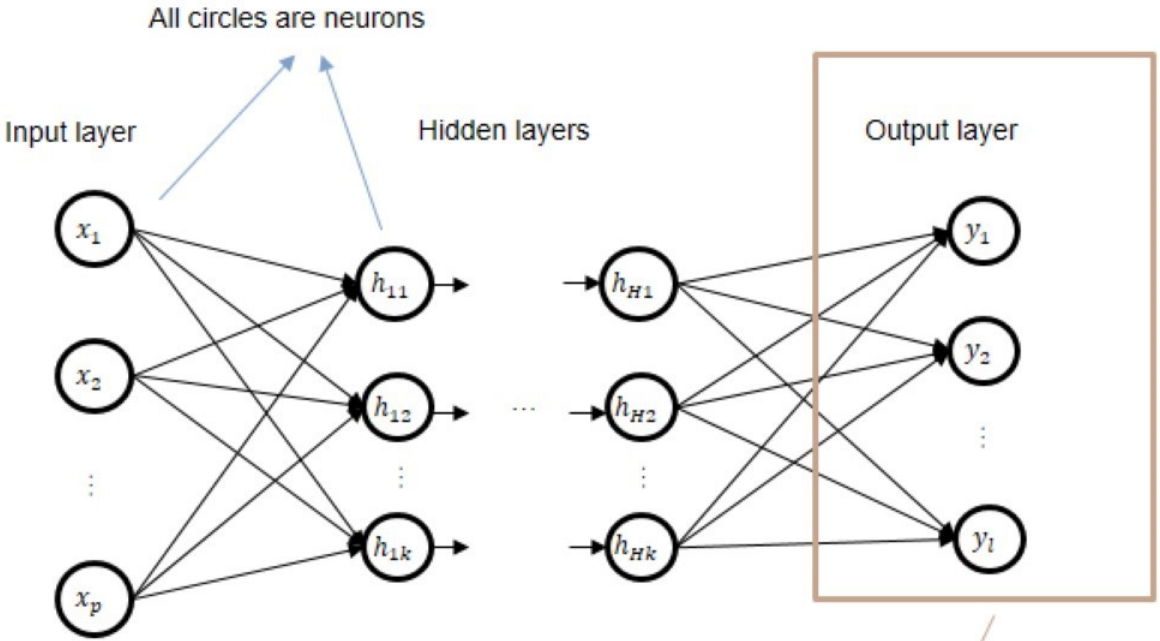
Artificial Neural Networks (ANNs)

Neural Networks



- Please watch the following video:
- https://media.ed.ac.uk/media/Neural+Networks/0_owjlgx0o/141757871

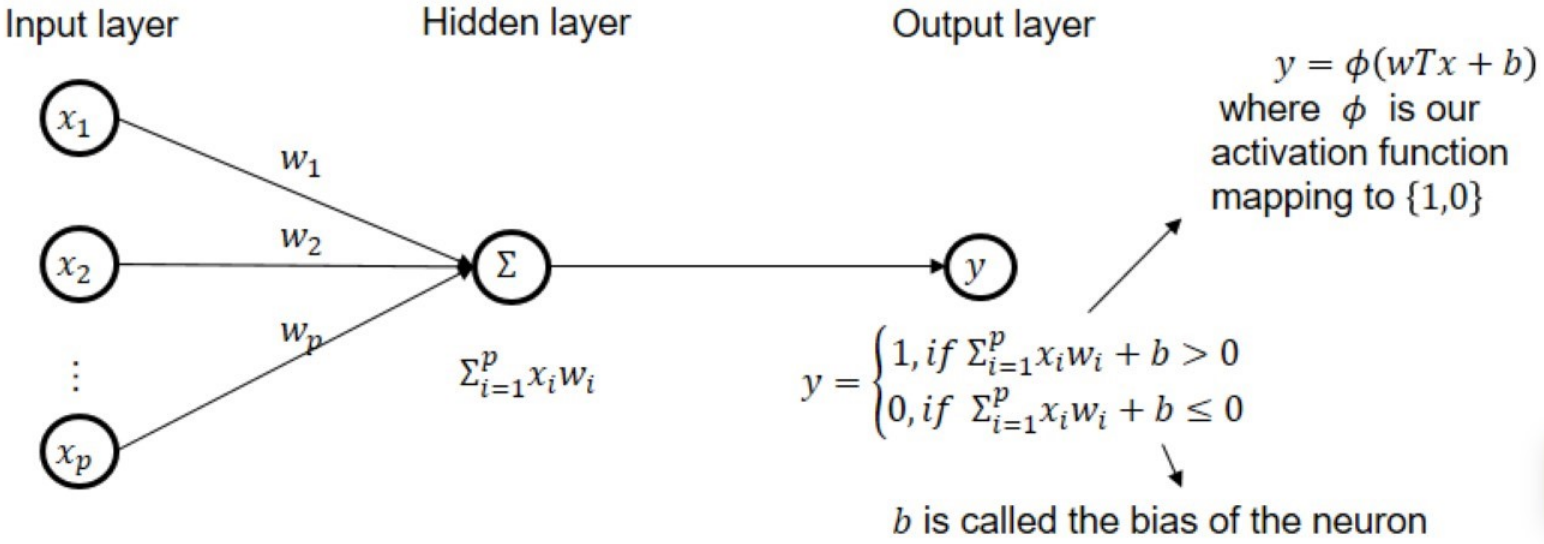
Neural network topology



A classification NN has 1 output neuron for each class
A regression NN has 1 output neuron

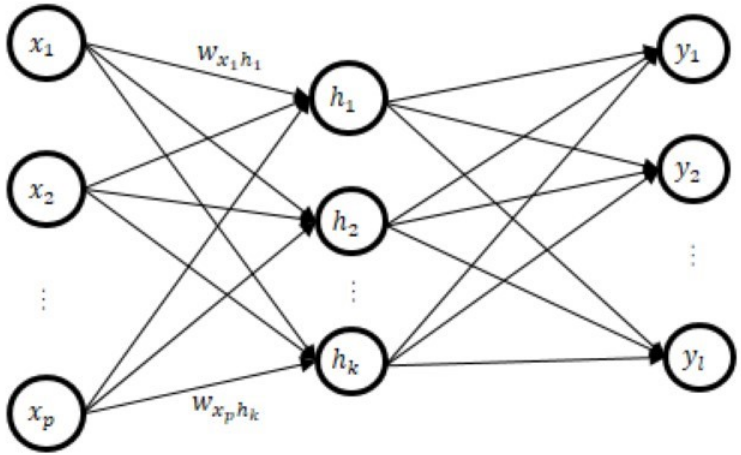
A perceptron

- 1 neuron consisting of a weighted combinations of inputs
- Notice how the output is a (0/1) outcome based on the weighted inputs



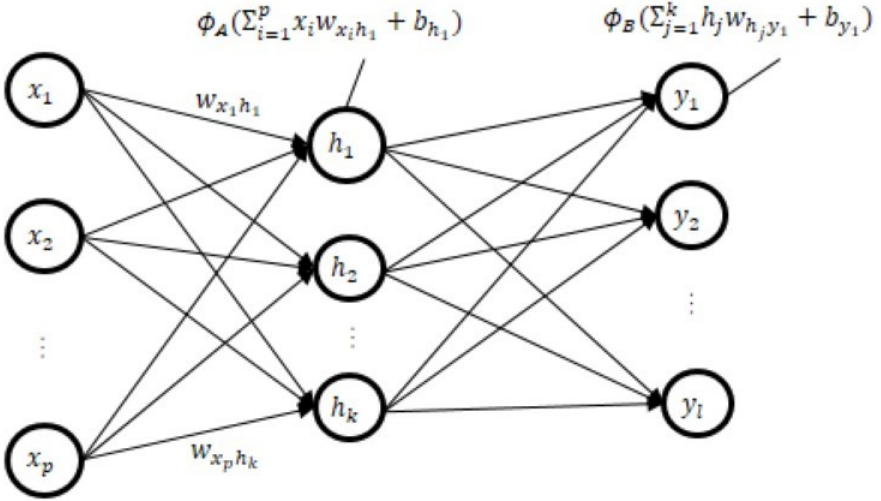
ANN with 1 hidden layer

- p input variables, 1 hidden layer with k neurons, and l output classes.
- Notice the weights between the neurons
- Each neuron transforms its weighted input using an activation function



Multilayer perceptron (MLP)

- Network consisting of many connected perceptrons
- Feed-forward: there are no connections to previous layers/no cycles
- Activation function present in each neuron ϕ



Notice how activations stack:

$$y_1 = \phi_B\left(\Sigma_{j=1}^{k}\phi_A\left(\Sigma_{i=1}^{p} x_i w_{x_i h_j} + b_{h_j}\right) + b_{y_1}\right)$$

Artificial Neural Networks (ANNs)

- Structured like human brains, connecting neurons to each other to form a network
- Contain 1 or more 'hidden' layers of neurons, between the input layer (the independent variables), and output layer (dependent variable)
  - The neurons transform their input before sending it on to the next layer

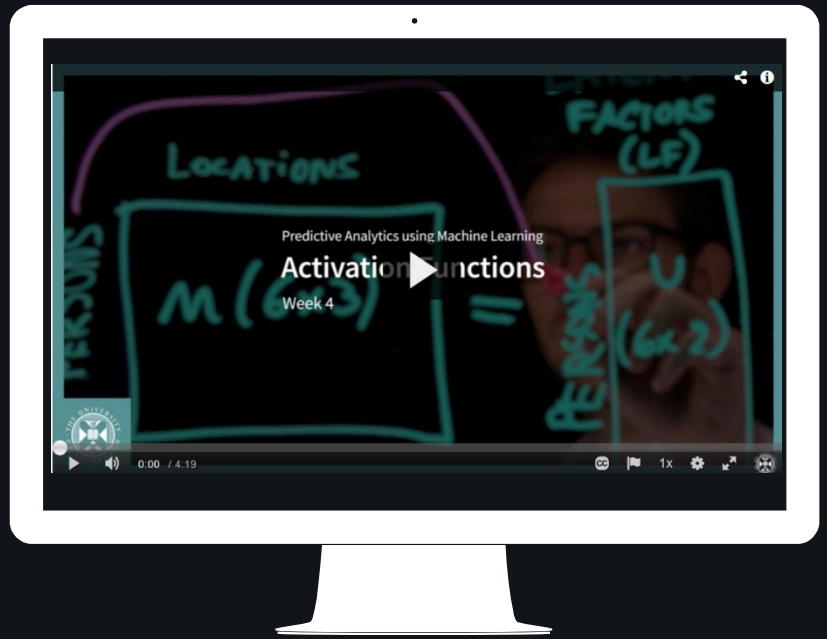
- Universal approximation theorems: they can be made of any number of layers, neurons, transformations, allowing for modelling the most complex relations between input and output

Activation functions

Activation functions

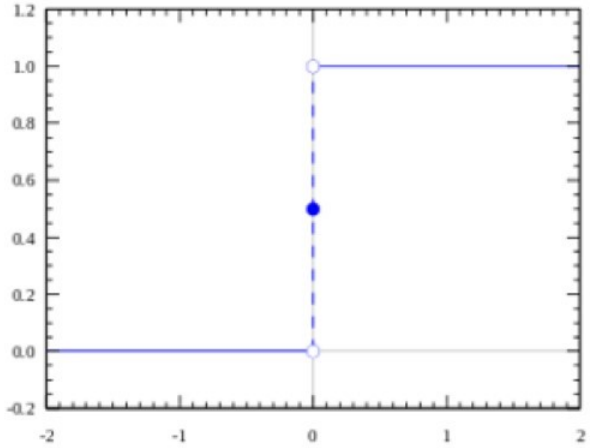
- Like with SVMs, there exist many activation functions
- Key characteristic: having a non-constant derivative
  - Later on, we use gradient descent to check the best direction to improve our weights
  - If the derivative is constant/not changing a lot, there is no information on how to further improve our parameters

Activation Functions



- Please watch the following video:
- https://media.ed.ac.uk/media/Neural+Networks/0_owjlgx0o/141757871

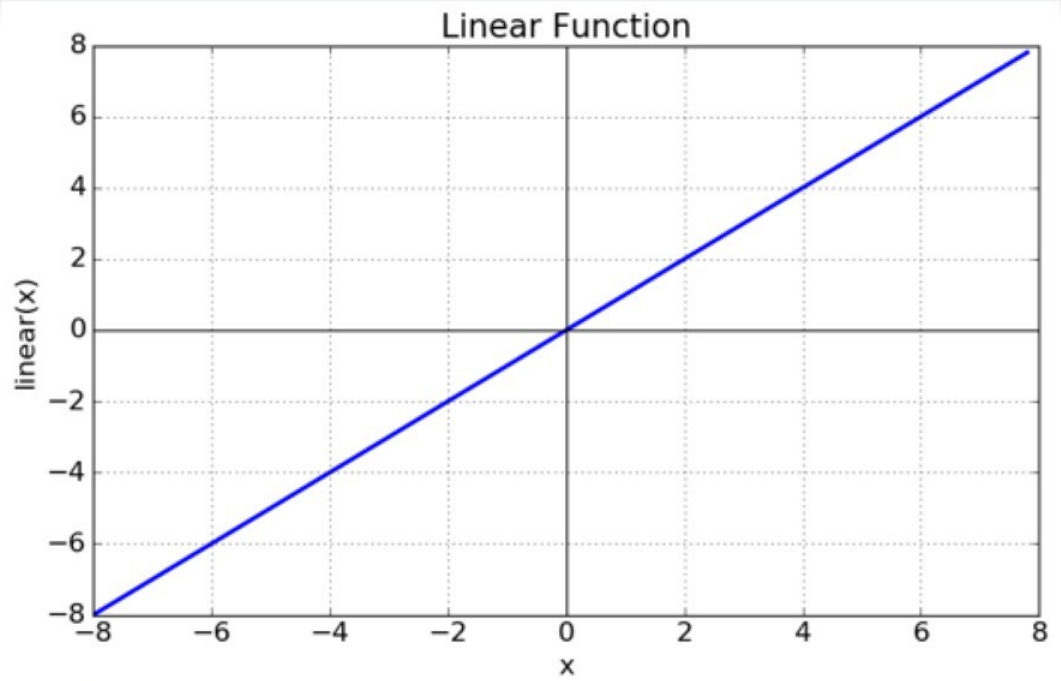
Unit step function



$$f(x) = \begin{cases} 1 \; if \; x > 0 \\ 0 \; if \; x \leq 0 \end{cases}$$

- Derivate 0 everywhere except at the ‘step’, not very useful
- Very rough

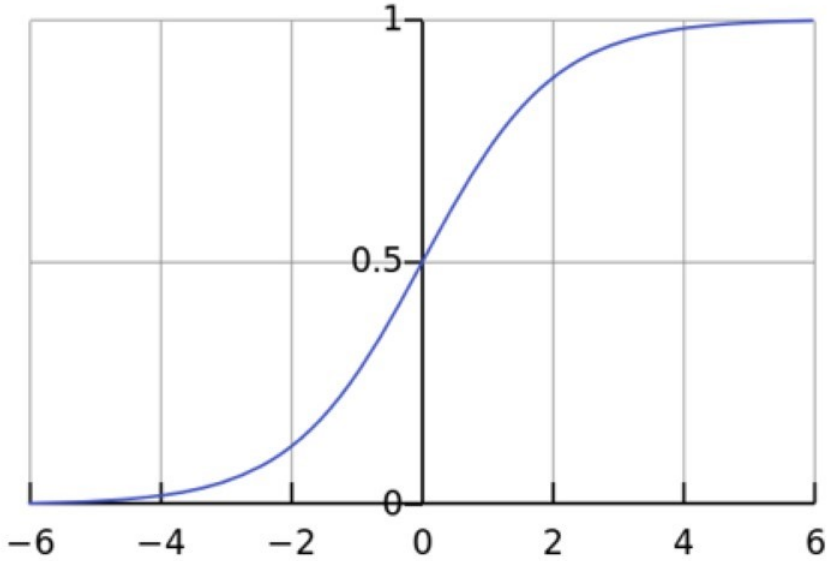
Linear function



$$f(x) = ax$$

- Constant derivative
- Multiple layers not useful: linear combinations of linear combinations are linear combinations

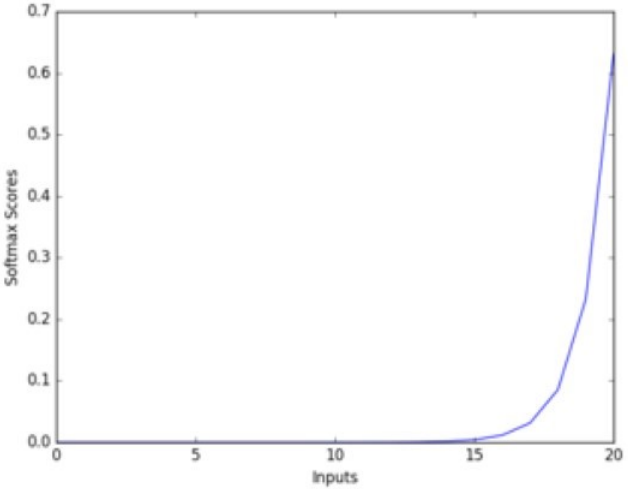
Sigmoid function



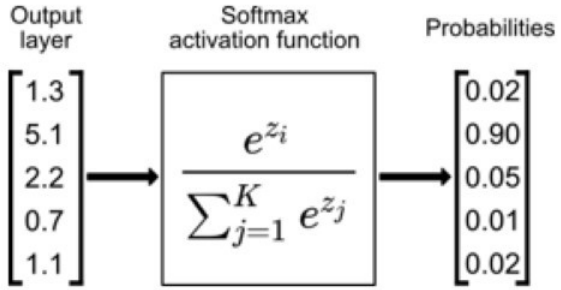
$$f(x) = \frac{1}{1 + e^{-x}}$$

- Not a constant derivative, but little happening at the edges
- Food for thought: a perceptron with this activation function is a logistic regression

Softmax



$$f(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \, , i = 1 \ldots K$$



- Not a constant derivative
- Often used in the output layer to transform the output scores into probabilities

  -> Class with highest probability is the predicted class

https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60

Try it yourself (optional)

You may try to build your NN on the Tensorflow website and play with different archetectures.

- https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.09198&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

- 8 - Building an MLP in Keras.ipynb + churn_ibm.csv
- 9 - Building an MLP with different kernels.ipynb

Activity: How to build an MLP in Keras

Parameters and cost function

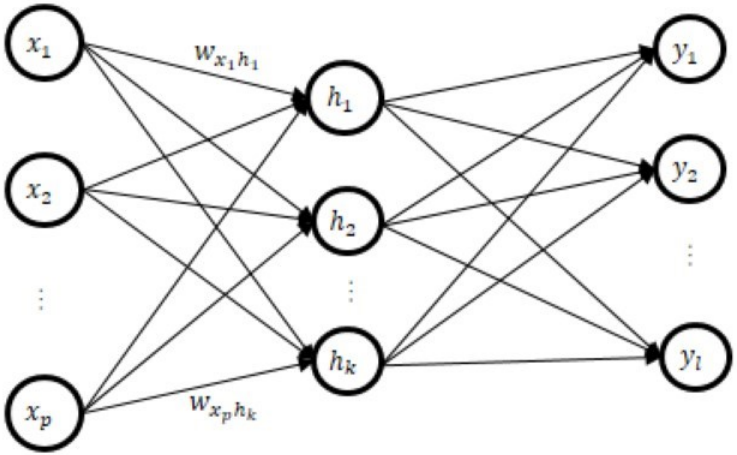
Parameters

- The elements we try to optimise to find the best network are the parameters $\boldsymbol{w}$ in each neuron
- Neural networks have many other parameters to finetune it as well, these are called hyperparameters:
  - Number of layers
  - Number of neurons in the layers
  - Activation function(s)
  - Learning rate
  - Regularisation rate
  - Dropout rate
  - Batch size
  - Number of epochs
  - …

Covered later

Cost/loss functions

- The network's output layer allows to calculate the cost/evaluate
- Many cost functions exist, most widely used:
  - Regression: quadratic loss: $E = \Sigma_1^l \frac{1}{2}(y_i - \hat{y}_i)^2$ can also be used for classification
  - Classification: cross-entropy: $E = -\sum_i^l y_i \log(\hat{y}_l)$, for l classes, where $\hat{y}_i$ is the softmax probability for the i-th class



Optimisation

Optimisation with gradient descent

- It is typically not possible (computationally) to optimise all the parameters in a neural network
- Approximation can be achieved through a forward-backward propagation to iteratively change the weights
  - Forward pass: assign random values to the weights
  - Backward pass:
    - Start in the final layer, calculate for each weight the direction (change in value) which gives you the 'steepest decline' (gradient) that will decrease your error/cost the most
    - Keep propagating these weights backwards until the first layer is reached
    - Keep iterating while weights are still changing significantly

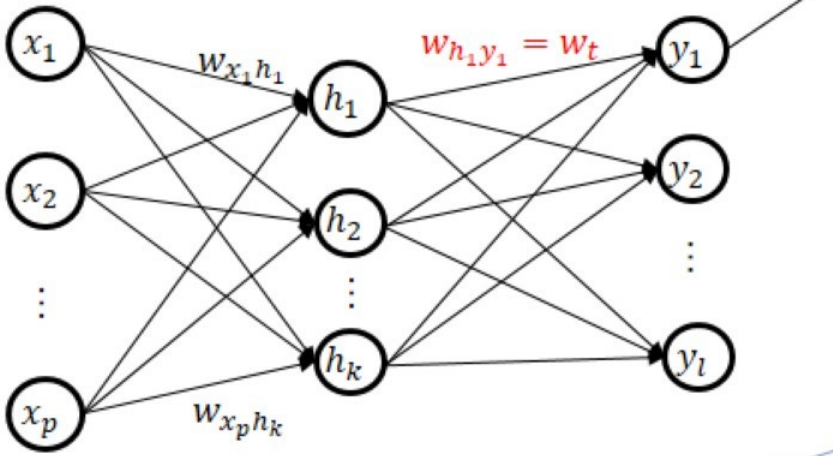
Backpropagation

Cost function: $E = \Sigma_1^l \frac{1}{2}(y_i - \hat{y}_i)^2$

Activation function: $\phi_B = \frac{1}{1+e^{-y_1'}}$

$\hat{y}_1 = \phi_B(\Sigma_{j=1}^k h_j w_{h_j y_1} + b_{y_1})$

$y_1' = h_j w_{h_j y_1} + b_{y_1}$



How to best update the weights in this neuron?
Use the chain rule:

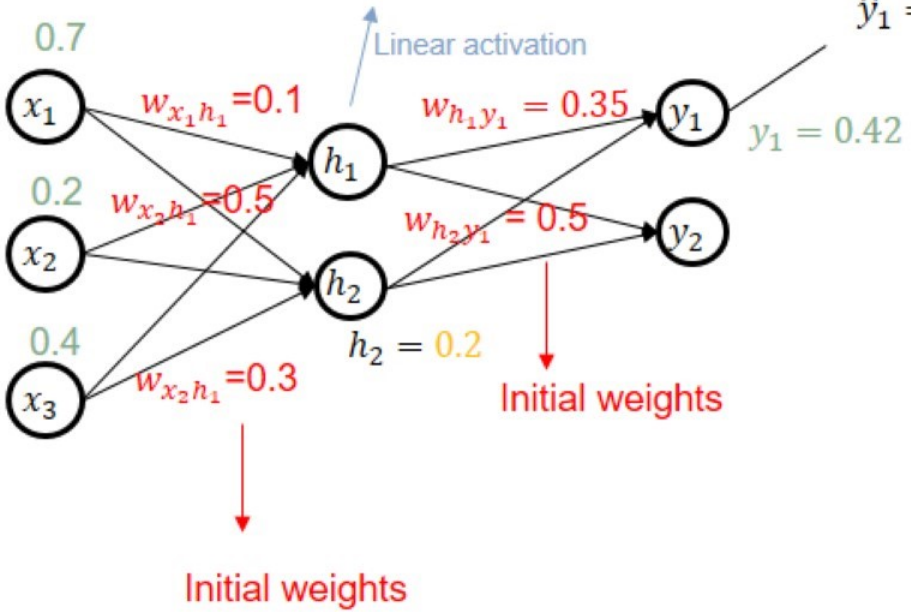
$$\frac{\partial_E}{\partial_{w_t}} = \frac{\partial_E}{\partial_{w_{h_1 y_1}}} = \frac{\partial_E}{\partial_{\hat{y}_1}} \cdot \frac{\partial_{\hat{y}_1}}{\partial_{y_1'}} \cdot \frac{\partial_{y_1'}}{\partial_{w_{h_1 y_1}}}$$

This one is not trivial

$$\frac{\partial_E}{\partial_{\hat{y}_1}} = 2 \cdot \frac{1}{2} \cdot (y_1 - \hat{y}_1)^{2-1} \cdot -1 = -(y_1 - \hat{y}_1) \qquad \frac{\partial_{\hat{y}_1}}{\partial_{y_1'}} = \hat{y}_1(1 - \hat{y}_1) \qquad \frac{\partial_{y_1'}}{\partial_{w_{h_1 y_1}}} = h_1 + 0 = h_1$$

$$\boxed{\frac{\partial_E}{\partial_{w_t}} = -(y_1 - \hat{y}_1) \cdot \hat{y}_1(1 - \hat{y}_1) \cdot h_1}$$

Example (no bias)



Cost function: $E = \Sigma_1^l \frac{1}{2}(y_i - \hat{y}_i)^2$

Activation function: $\phi_B = \frac{1}{1+e^{-y_1'}}$

$$\hat{y}_1 = \phi_B\left(\Sigma_{j=1}^k h_j w_{h_j y_1} + b_{y_1}\right) = \frac{1}{1+e^{-0.202}} = 0.55$$

$$y_1' = 0.29 \cdot 0.35 + 0.2 \cdot 0.5 = 0.202$$

Use the chain rule:

$$\frac{\partial_E}{\partial_{w_{h_1y_1}}} = \frac{\partial_E}{\partial_{\hat{y}_1}} \cdot \frac{\partial_{\hat{y}_1}}{\partial_{y_1'}} \cdot \frac{\partial_{y_1'}}{\partial_{w_{h_1y_1}}}$$

$$\frac{\partial_E}{\partial_{w_{h_1y_1}}} = -(y_1 - \hat{y}_1) \cdot \hat{y}_1(1 - \hat{y}_1) \cdot h_1$$

$$= -(0.42 - 0.55) \cdot 0.55 \cdot (1 - 0.55) \cdot 0.29 = 0.009$$

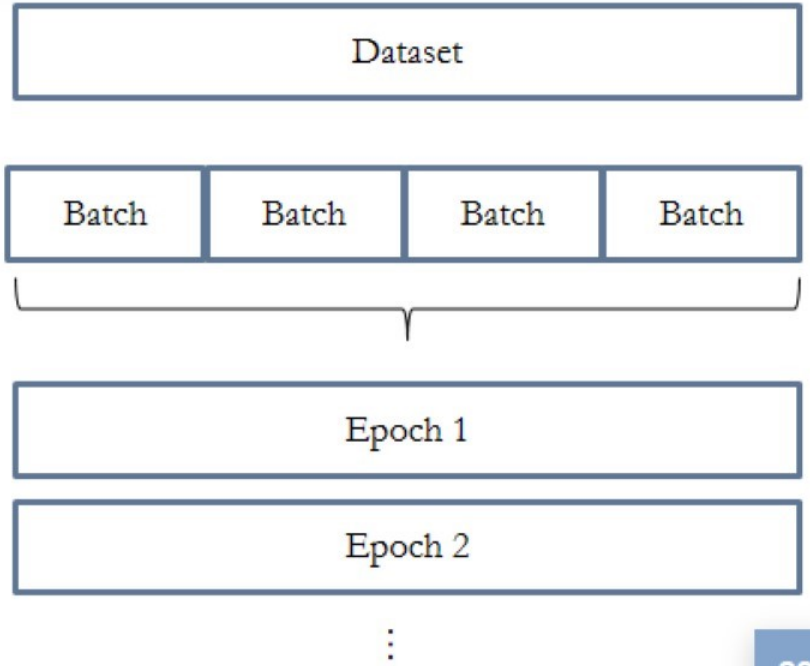
Points in green are the input data

Optimisation

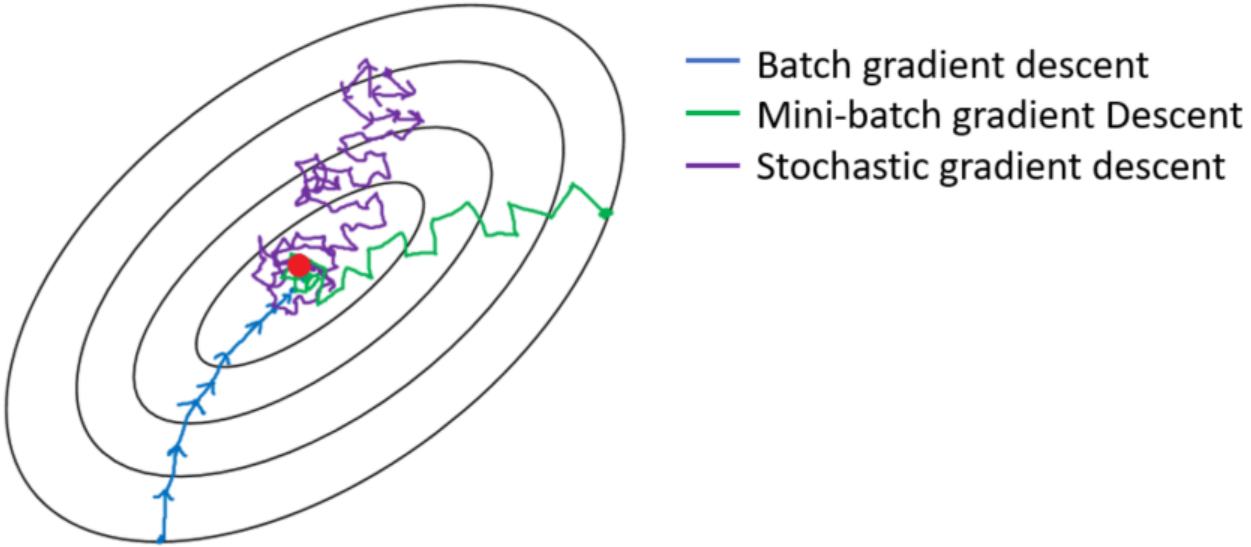
- Once we obtain the gradient, we can use it to update the weights:
  - $w_t^+ = w_t - \alpha \cdot \frac{\partial_E}{\partial_{w_t}}$
  - α is the learning rate:
    - Allows us to control how much we update the weight in that direction
    - High learning rate: we might try to converge too soon, getting stuck in local optimum
    - Low learning rate: we might not converge enough (or at all), resulting in underfitting
- Hence the importance of having activation functions with non-constant derivates!

Batches and epochs

- Sometimes we need more than 1 iteration in order for stochastic gradient descent to converge
  - Epoch: 1 full run of all data points over the network
  - We might require a high number of epochs
- Calculating gradient descent one-by-one is time consuming
  - Use batches instead: subset of data points
  - Average new outcomes of weights per batch
  - Batch size influence:
    - Small batches: converge faster, but might not be representative
    - Large batches: might take a while to converge again

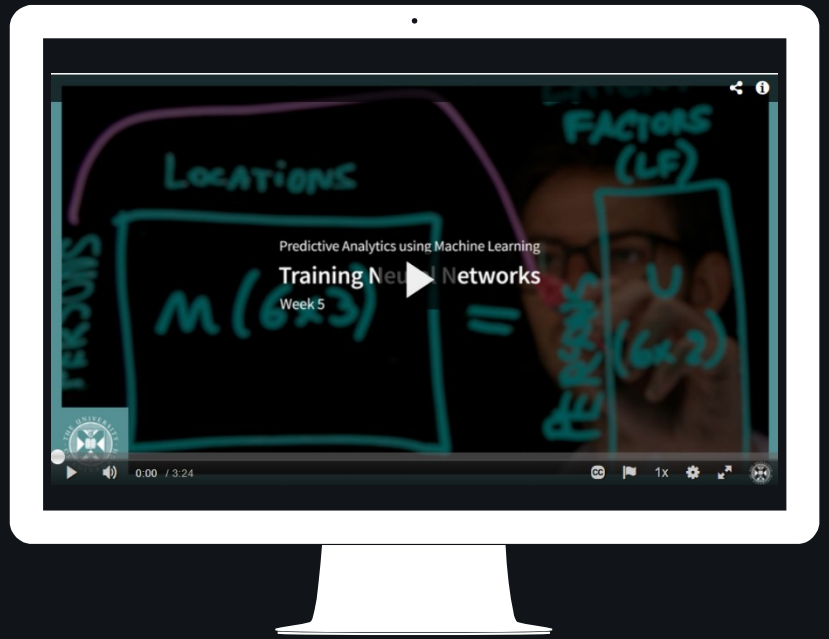


Influence on gradient descent



https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

Training Neural Networks



- Please watch the following video:
- https://media.ed.ac.uk/media/Training+Neural+Networks/1_3sos4inp/141757871

- 10 - Building neural networks for regression.ipynb

Activity: How to build neural networks for regression

Avoiding overfitting

Dealing with overfitting

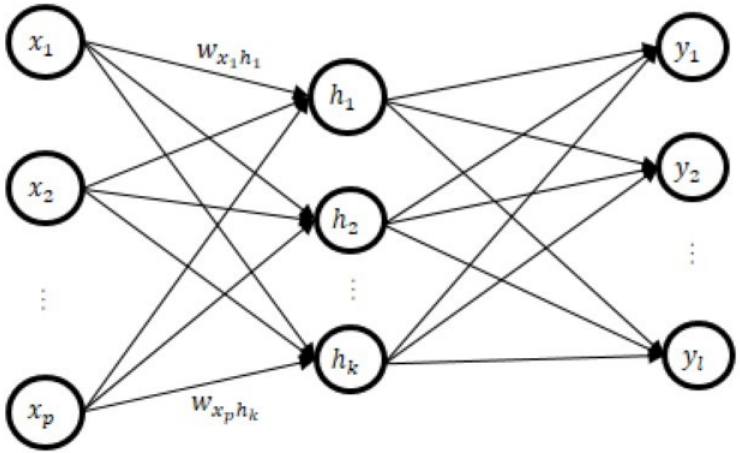
- Please watch the following video:
- https://media.ed.ac.uk/media/Overfitting/1_89ctex9d/141757871

Avoiding overfitting

- We can avoid the network becoming too complex, often a sign of overfitting
- Two main techniques:
  - Regularisation: penalize for having high sum of weights (and hence including many neurons)
  - Dropout: don't use all neurons for training in every iteration
  - Early stopping: use training and validation sets, stop when error starts to increase on validation set

Regularisation

- Two types or regularization:
  - L1: weights can become 0
  - L2: weight decay, weights driven towards 0, very prone to outliers (due to squaring)
  - λ is yet another hyperparameter to set the severeness of punishing for higher weights



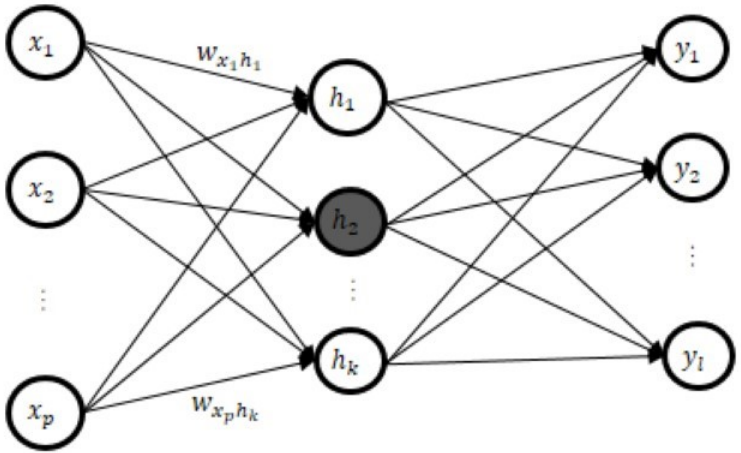
Loss function $L(y, \hat{y}) = E = \Sigma_1^l \frac{1}{2}(y_i - \hat{y}_i)^2$

L1 regularisation: $Cost\ function = L(y, \hat{y}) + \lambda \sum_{n \in neurons} |w_n|$

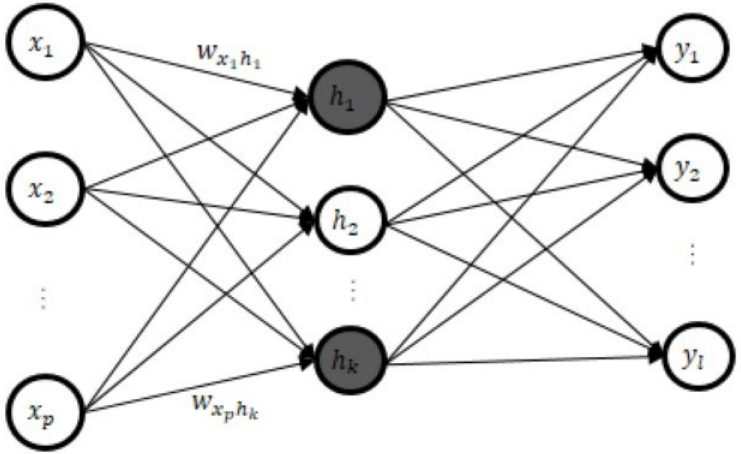
L2 regularisation: $Cost\ function = L(y, \hat{y}) + \lambda \Sigma_{n \in neurons} w_n^2$

Dropout

Iteration 1

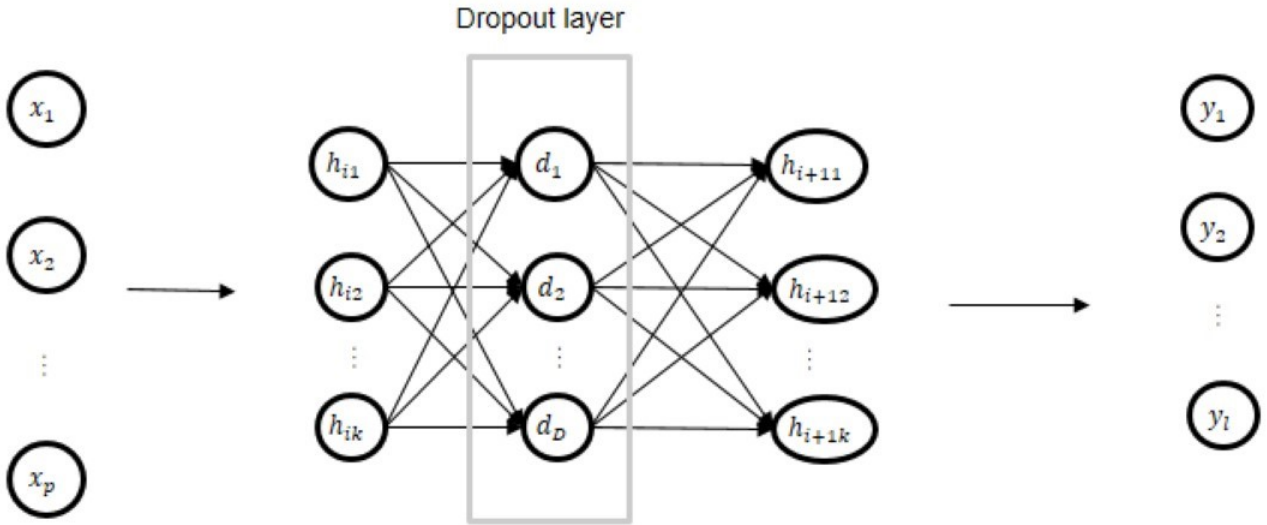


Iteration 2



Dropout

- To obtain this, extra dropout layers are added (set weights to 0)
  - Dropout rate: proportion of neuros that are dropped



Neural networks wrap-up

- Benefits:
  - Incredibly powerful model
  - Can handle very high-dimensional input
  - Is a generalization of basically any other model
- Downsides:
  - Overfit easily
  - Take a long time to optimize
  - An incredibly big set of hyperparameters to tune

- You can find many examples of (deep) neural network architectures for applications such as image classification, text generation, and so on:
  - https://keras.io/optimizers/
  - https://keras.io/regularizers/
  - https://keras.io/examples/lstm_text_generation/

- 11 - Hyperparameter exploration.ipynb

Activity: Looking at different hyperparameters

UNIVERSITY OF EDINBURGH
Business School