Lecture week 2

Regression

THE UNIVERSITY of ADELAIDE

Lecture Summary

Regression

- Generalised Additive Model
- Generalised Linear Model

Neural Networks

- Basics: review
- Gradient Descent
- Activation Functions

Regularisation

- Ridge Regularisation
- Lasso Regularisation

Linear regression and its variants

Linear regression

Basic equation

$$y = w_o + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_n x_{in} + \epsilon$$
$$Y = W \times X + \epsilon$$

ϵ is an error, and is assumed to follow Gaussian distribution, therefore weight estimation can follow confidence intervals.

- We are predicting the mean

What is the main problem with linear regression?

Two well-known in Data Science variation of linear regression

1. Generalised Additive Model (GAM)
2. Generalised Linear Model (GLM)

More on Linear regression

- Common metric for LR is R^2 (R-squared, R2, or coefficient of determination)

$$R^2 = 1 - \frac{\sum_{i=1}^{n}\left(y_i - y_i^p\right)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = 1 - \frac{ResidualSumSquares}{TotalSumSquares}$$

- n is the number of instances, $\bar{y}$ is mean of y, y_i^p is predicted y_i value
- What if number of predictors increases?
- To fix this, use adjusted R^2.

$$R_{adj}^2 = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

Where n is the number of instances, p is number of predictors.

This way increasing $(1 - R^2)$ with increased p is offset by $\frac{n-1}{n-p-1}$

Generalised Linear Model (GLM)

When is this used?

Relationship X=>Y is not linear

Variance of Y is not constant (i.e., not normally distributed)

What is the difference?

In linear regression the underlying assumption is that the y_i values are coming from a normal distribution.

In GLM, the y_i values come from any exponential family distribution.

GLM has three components

Link Function Linear Predictor

$$\text{link}(E_Y(y|x)) = b_0 + b_1 x_i + \cdots + b_p x_p$$

Therefore, $E_Y(y|x) = \mu = link^{-1}(\boldsymbol{B}^{\boldsymbol{T}}\boldsymbol{X})$

$P(y|x) =$ Probability distribution from exp family

Regression: GLM

If we try to model something like this:

- Relation X =>Y is not linear
- Variance of Y is not constant

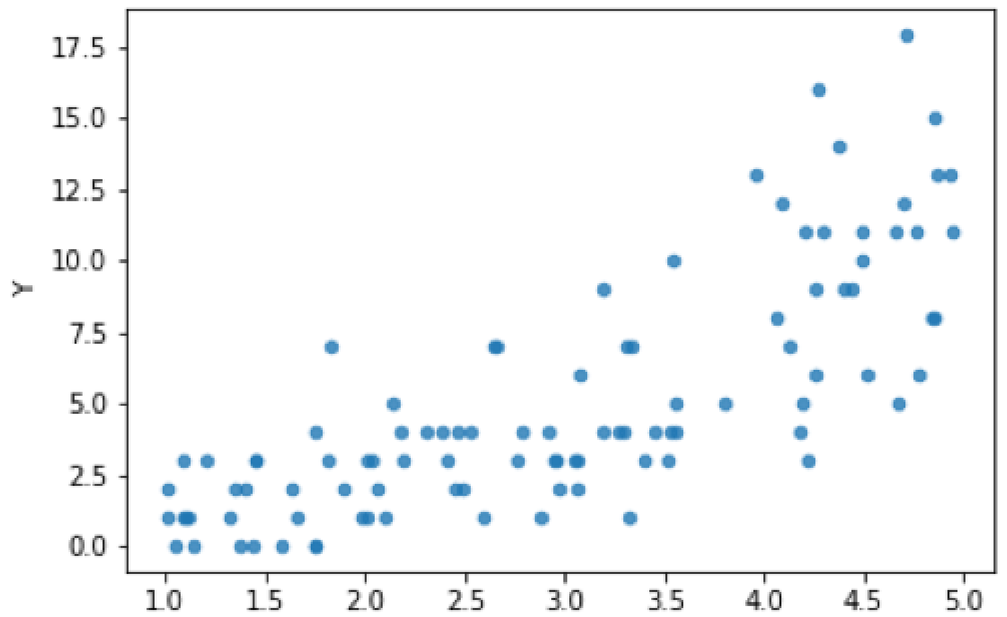
GLM allows to link linear regression with the mean

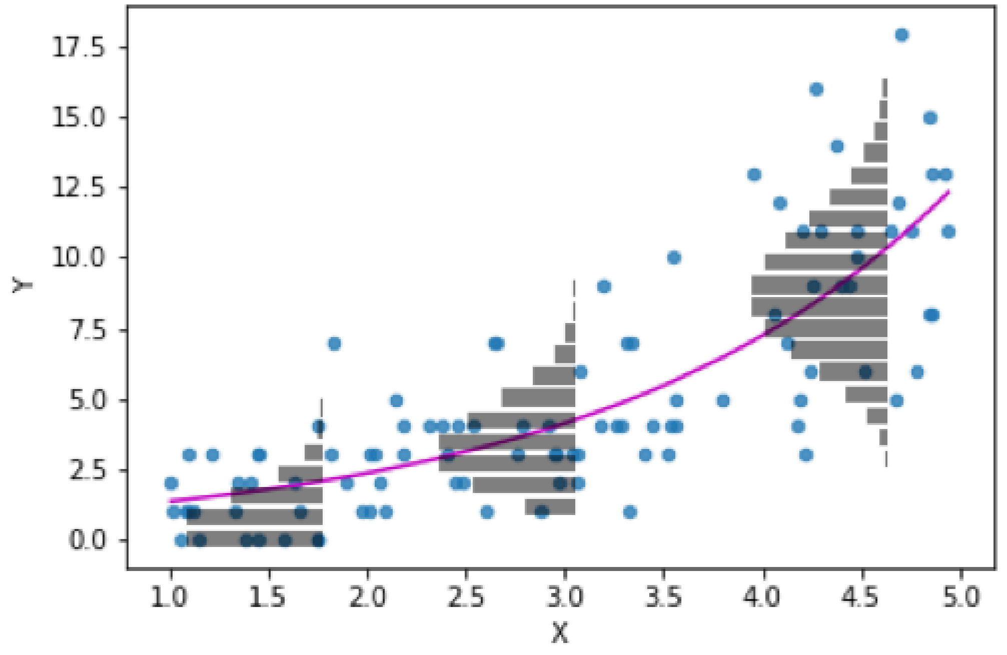
Link function Linear predictor

$$\underbrace{\ln}_{\text{Link function}} \lambda_i = \underbrace{b_0 + b_1 x_i}_{\text{Linear predictor}}$$

$$y_i \sim \underbrace{\text{Poisson}}_{\text{Probability distribution}}(\lambda_i)$$

Probability distribution





- Logistic regression is kind of GLM.

https://towardsdatascience.com/generalized-linear-models-9cbf848bb8ab

How does this work?

For each distribution, there is a corresponding canonical link function.

So, if you know the behaviour of your output (in terms of distribution), you can pick that distribution and then find the corresponding link function.

The rule is that

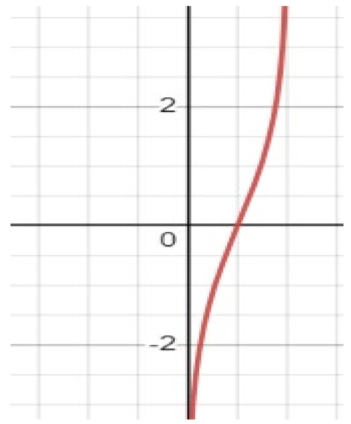
$$\mathrm{link}(E_Y(y|x)) = b_0 + b_1 x_i + \cdots + b_p x_p = \eta$$

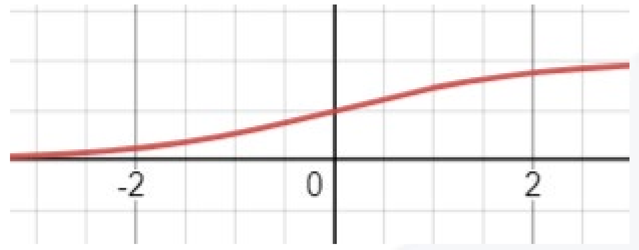
Link function transforms mean of the data distribution into linear predictor η

Distribution	Link Function
Normal	**Identity** $\boldsymbol{B^T X = \mu}$
Binomial (Logistic Regression)	**Logit** $\boldsymbol{B^T X = ln \frac{\mu}{1-\mu}}$
Poisson	**Log** $\boldsymbol{B^T X = ln(\mu)}$
Gamma	**Inverse=** $\boldsymbol{B^T X = -\frac{1}{\mu}}$

Looking at some link functions

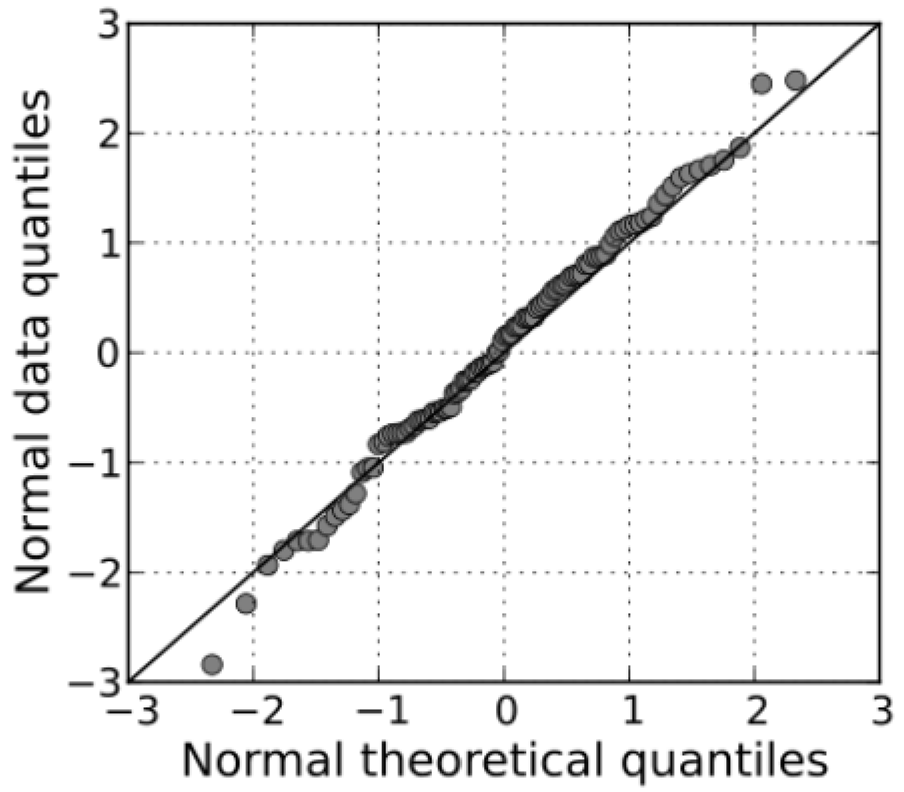
- Linear regression: residual error follows normal distribution
  - We are predicting mean for given data
  - The transformation from mean μ to linear predictor $\boldsymbol{B}^T\boldsymbol{X}$ is just identity
- Bernoulli distribution: outcome is 0 or 1.
  - We are predicting $P(Y = 1|X)$, what is the link function?
  - $p(Y=1)=p^Y(1-p)^{1-Y}$
  - Link function for Bernoulli is $\boldsymbol{B}^T\boldsymbol{X} = \boldsymbol{ln}\frac{\mu}{\boldsymbol{1-\mu}}$, $link^{-1} = \frac{1}{1+\exp(-\boldsymbol{B}^T\boldsymbol{X})}$
  - How to justify this mathematically?

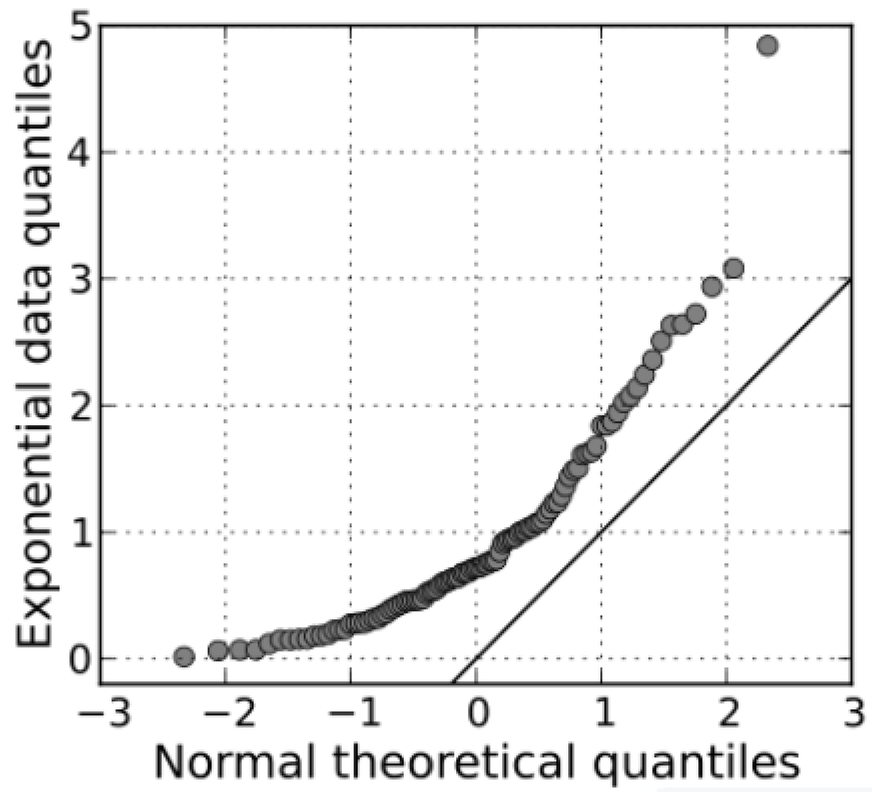




How to choose link function?

- Try to predict using linear regression (assuming normal distribution of residual)
- Make QQ plots of residuals (Quantile-Quantile plot). Y axis: your sample percentile, X is normal distribution (or another distribution). Straight line: agreement of X and Y distr.

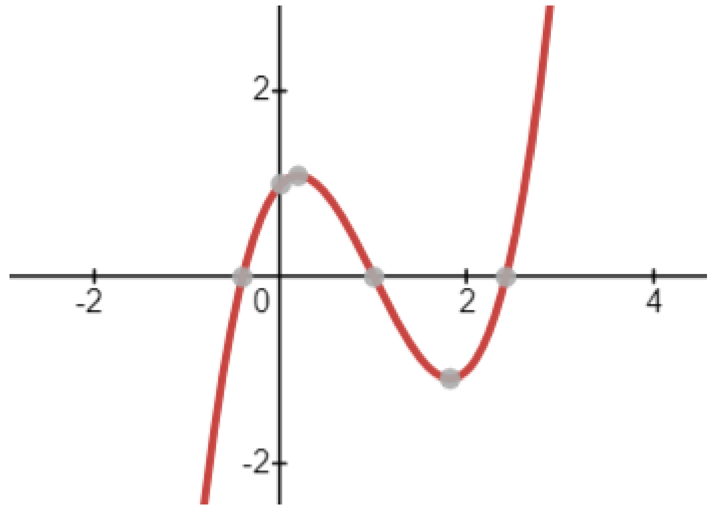




https://statisticsbyjim.com/graphs/qq-plot/

Looking at some link functions

- But what if your data shows something like this:



- General Additive Model (GAM)

Generalised Additive Model (GAM)

Basic equation:

$$y = \beta_0 + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p) + \epsilon.$$

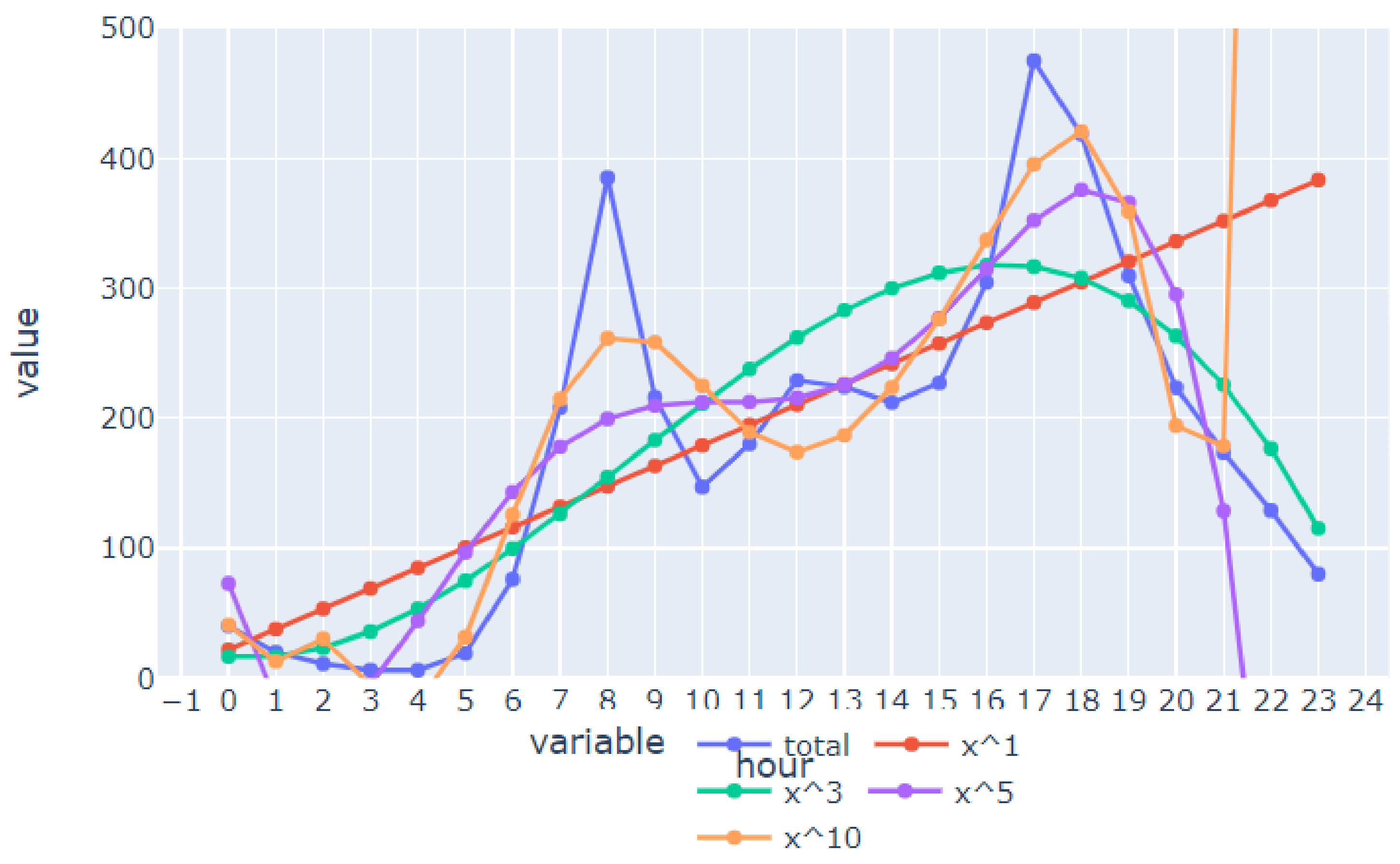
where *f1,* … are called smoothing functions. GAM allows nonlinearity into linear regression.

So the difference with GLM is allowing nonlinearity into linear predictor

Polynomial regression is a special case.

Can be combined with GLM

Polynomial Regression on Median Bike Rentals Per Hour - Hours 22, 23 Predicted



https://bookdown.org/ssjackson300/Machine-Learning-Lecture-Notes/generalised-additive-models.html
https://towardsdatascience.com/generalised-additive-models-6dfbedf1350a

GLM vs GAM research results

GLM

```
 library(caret)
  tmk_model<-train(as.factor(gen_onset_n)~.,
                        method="glm",
                        trControl=fitControl.2,
                        family="binomial",
                        metric="ROC",
                        data=training)
probs <- predict(tmk_model, newdata=test,
type="prob")
```

GAM

```
library(caret)
gam_formula<-formula(gen_onset_n~
ccode+centralasia+TMKiGPdum+ …)
tmk_model <- gam(gam_formula,
                    family=binomial(link="logit"),
                    data=df_train,select=TRUE)
probs <- predict(tmk_model, newdata=test,
type="response")
```

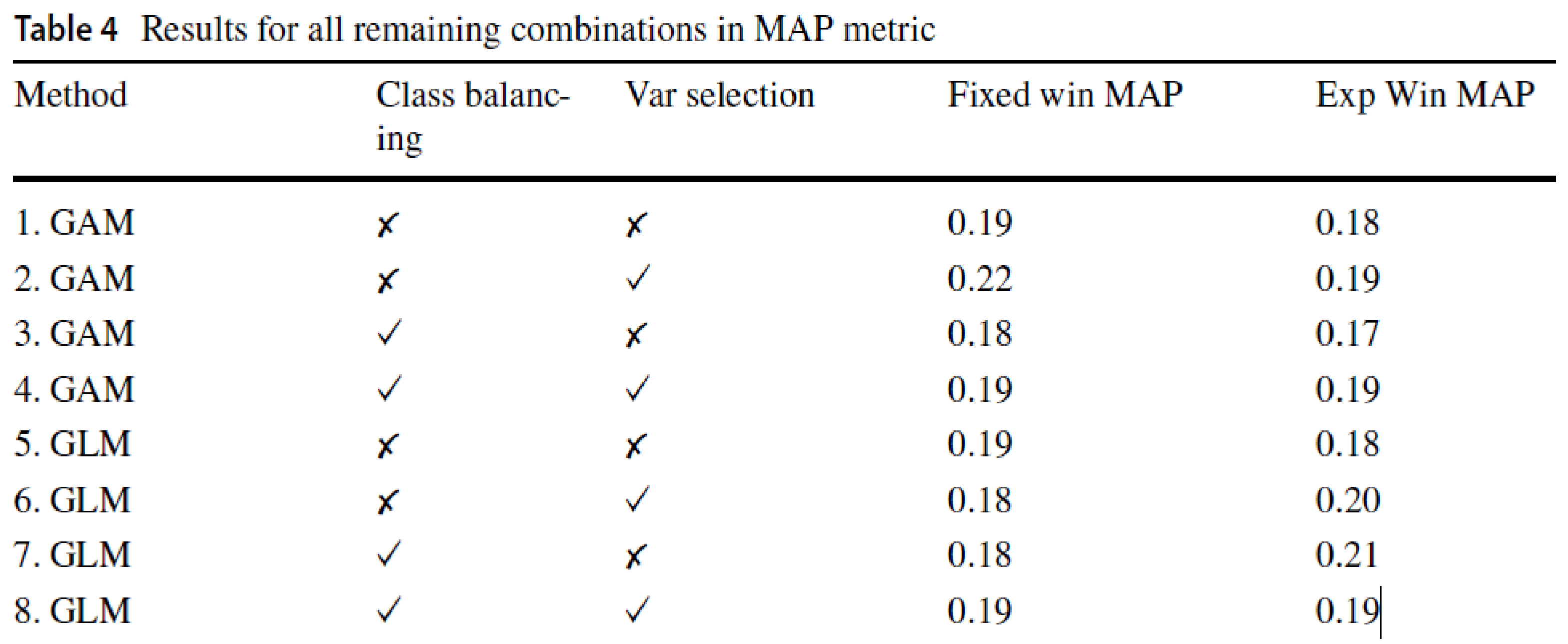


Table 4 Results for all remaining combinations in MAP metric

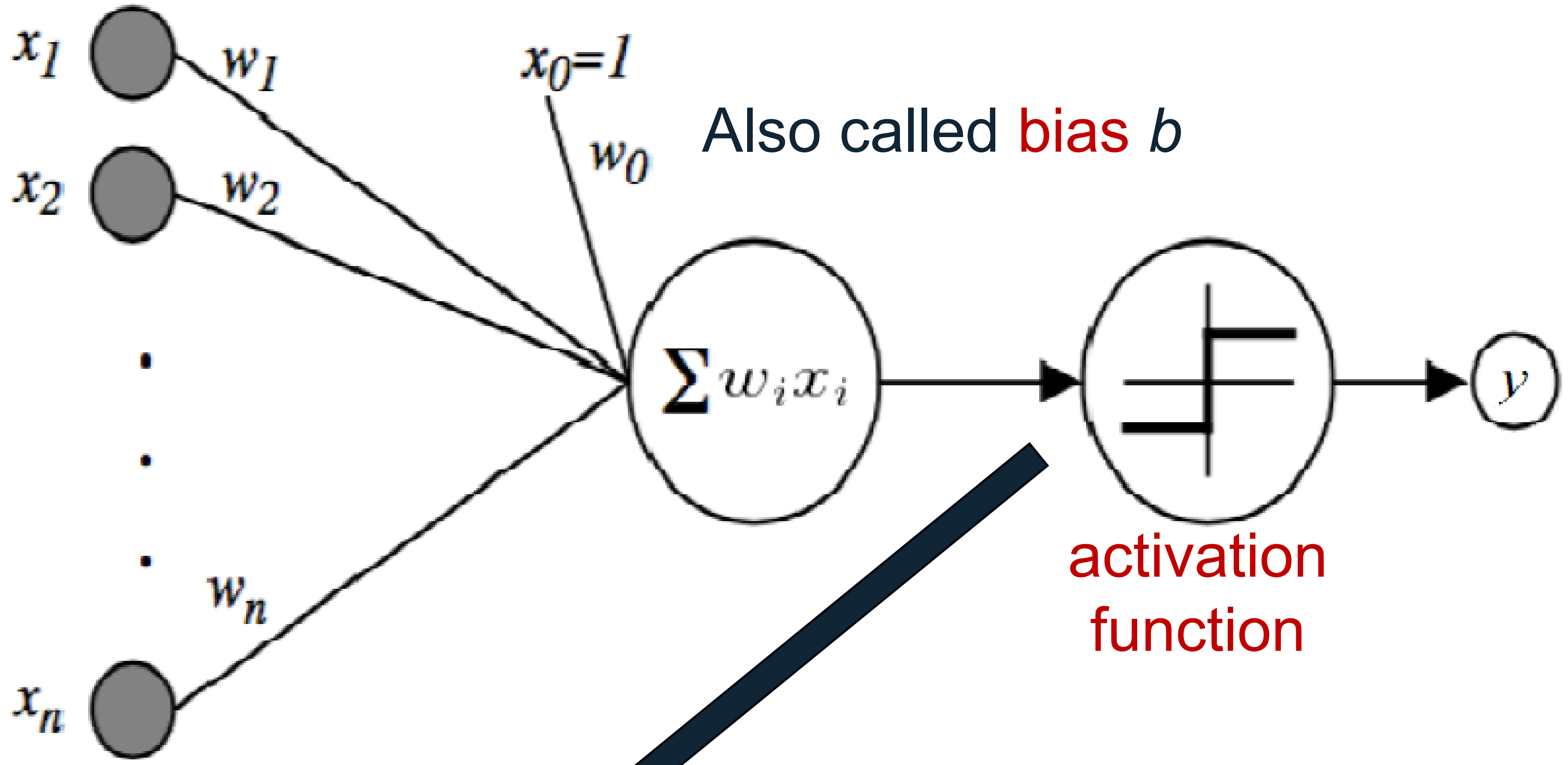
Method	Class balancing	Var selection	Fixed win MAP	Exp Win MAP
1. GAM	✗	✗	0.19	0.18
2. GAM	✗	✓	0.22	0.19
3. GAM	✓	✗	0.18	0.17
4. GAM	✓	✓	0.19	0.19
5. GLM	✗	✗	0.19	0.18
6. GLM	✗	✓	0.18	0.20
7. GLM	✓	✗	0.18	0.21
8. GLM	✓	✓	0.19	0.19

Krzywicki, A., Muchlinski, D., Goldsmith, B.E. and Sowmya, A., 2022. From academia to policy makers: a methodology for real-time forecasting of infrequent events. *Journal of Computational Social Science*, pp.1-22.

THE UNIVERSITY *of* ADELAIDE

Neural Networks (revision from online)

From perceptron to Neural Networks



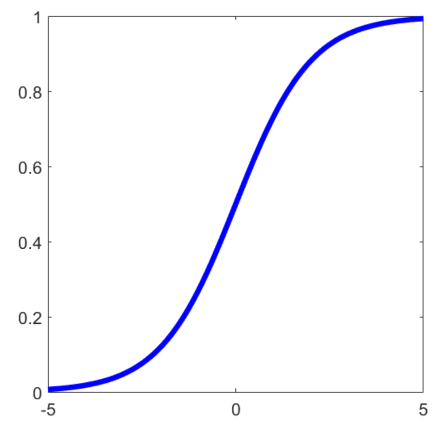
Also called bias *b*

activation function

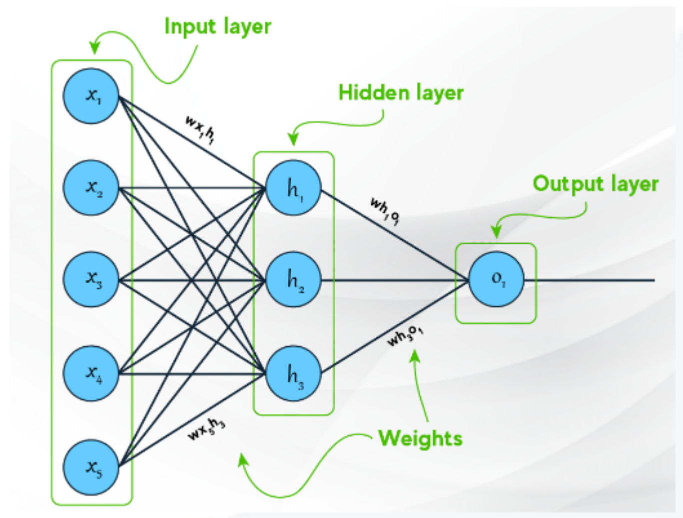
$$z(\boldsymbol{x}) = \sum_{i=0}^{n} x_i w_i$$

$$z(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w}$$

$$y = f(z(x)) = \begin{cases} 1 & if \;\; z(\boldsymbol{x}) \geq 0 \\ -1 & if \;\; z(\boldsymbol{x}) < 0 \end{cases}$$

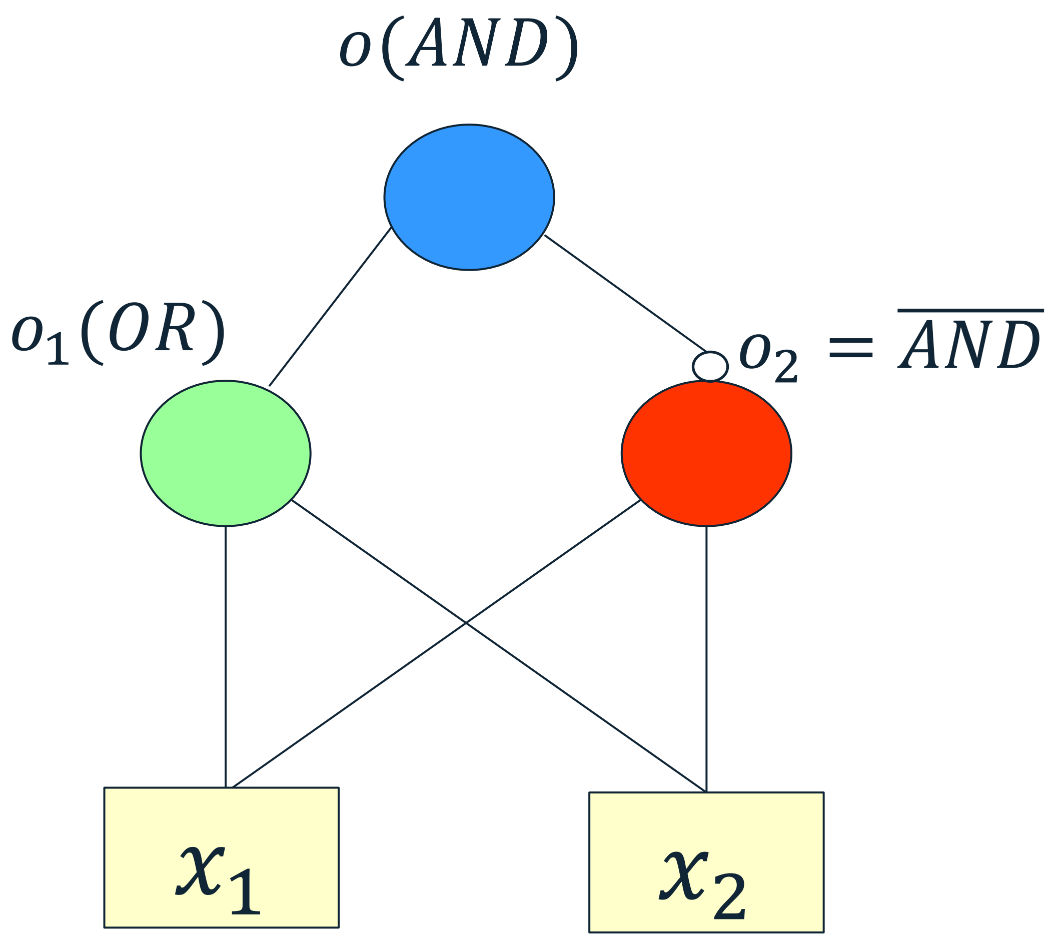


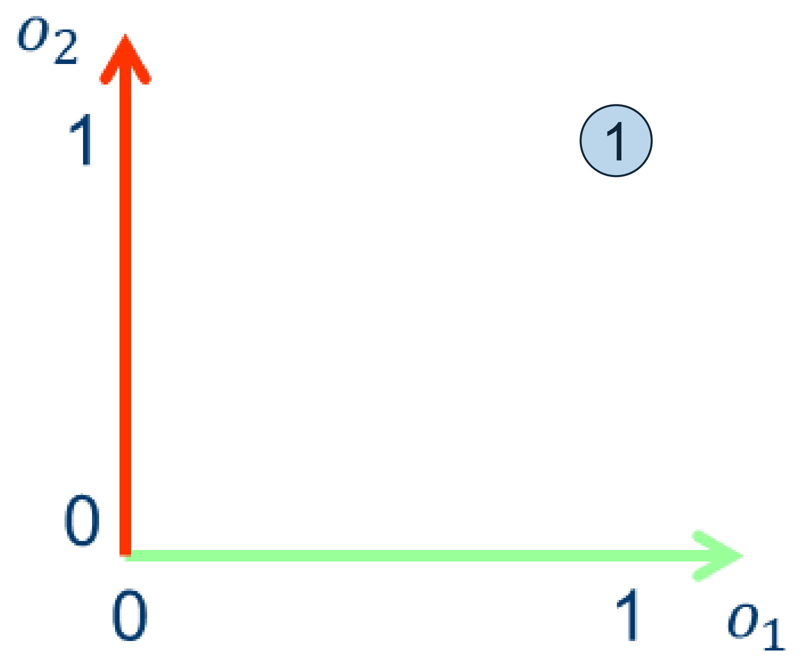
$$f(x) = \frac{1}{1 + e^{-z}}$$



Limitation of Perceptron not in multilayer NN

- XOR can be represented with additional layer of perceptrons
- This is one of the reasons why multilayer NN work





Simple Example: Learning AND

Training Data: {((0,0),0), ((0,1),0), ((1,0),0), ((1,1), 1)}

Learning rate $\eta = 1$, we also assume [0,1] activation function (not [-1,1])

Initialization: $w_1 = w_2 = b = 0$

	$\boldsymbol{x}$	t	y	Δw_1	Δw_2	Δb	w_1^{new}	w_2^{new}	b^{new}
							0	0	0
Epoch 1	0 0	0	1	0	0	-1	0	0	-1
	0 1	0	0	0	0	0	0	0	-1
	1 0	0	0	0	0	0	0	0	-1
	1 1	1	0	1	1	1	1	1	0

$$y = f(a(x)) = \begin{cases} \mathbf{1} & \boldsymbol{if} \;\; \boldsymbol{a}(\boldsymbol{x}) \geq \mathbf{0} \\ \mathbf{0} & \boldsymbol{if} \;\; \boldsymbol{a}(\boldsymbol{x}) < \mathbf{0} \end{cases}$$

Epoch: one cycle through the training data, *t* is target, *y* is actual output

What would be y if we make the first step for Epoch 2?

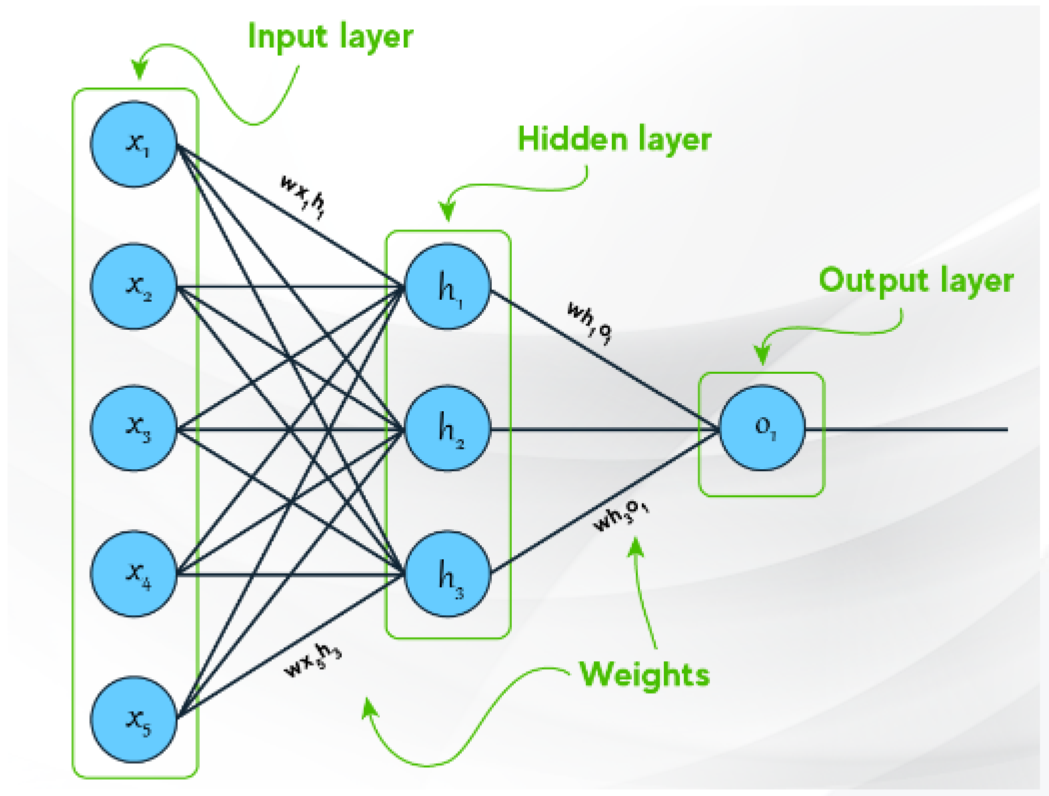
BackProp and Gradient Descent

- Sigmoid function as the activation function

$$z(\boldsymbol{x}) = \sum_{i=0}^{n} x_i w_i$$
$$z(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w}$$
$$f(x) = \frac{1}{1 + e^{-z}}$$

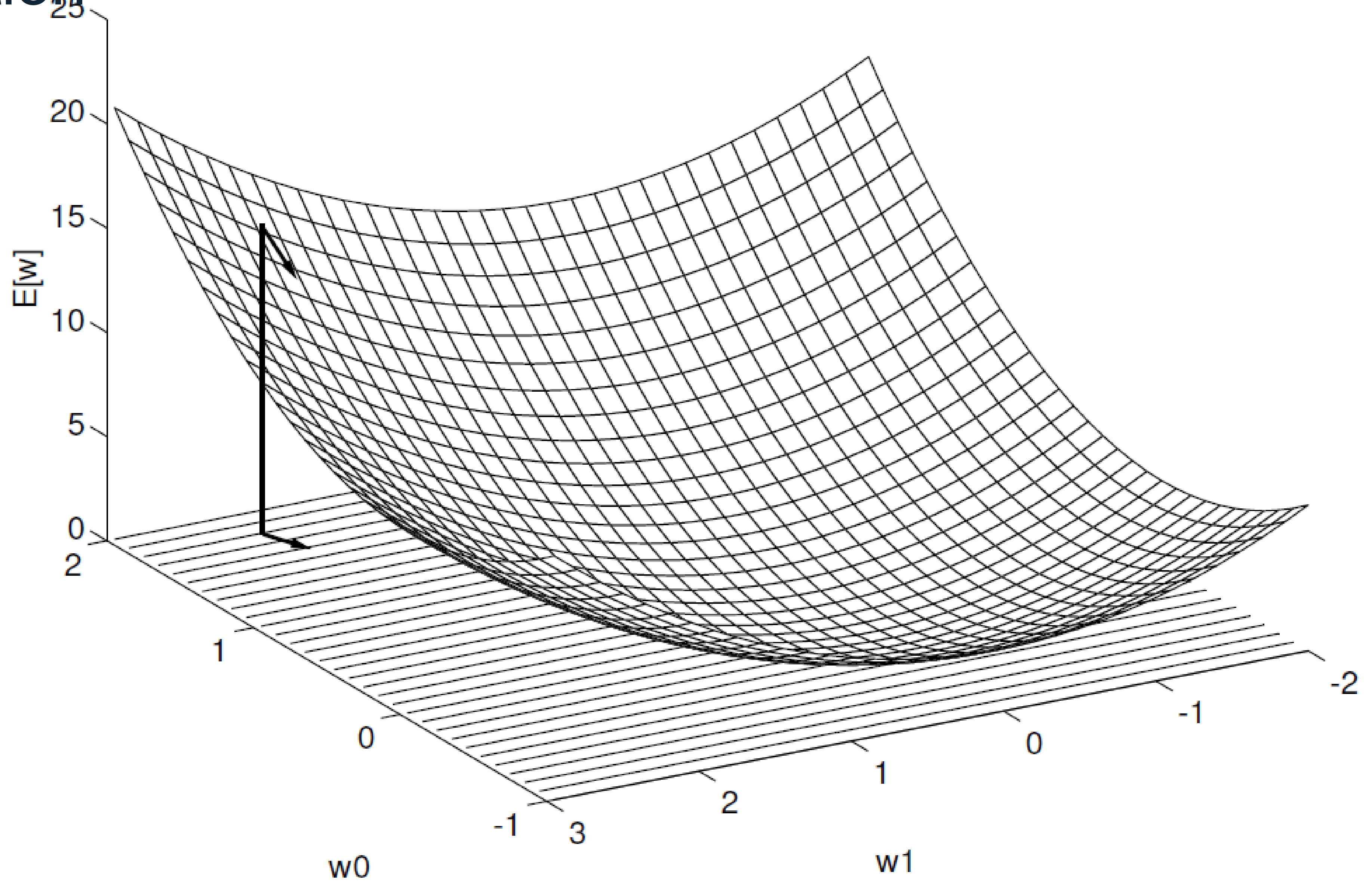
- NN with hidden layers

- Update weights for all units using BP algorithm (BP) calculated with gradient descent and chain rule
- BP: forward pass: calculate outputs
- BP backward pass: update weights using square error (or different loss function) for each unit



Gradient Descent

Weights need to change to get the min error on an error surface, which is unknown function of weights. Do it in small steps following Expectation-Maximisation



Offline and Online GD

Offline Training: Weight update after **all** training patterns

- correct
- computationally expensive and slow
- works with reasonably large learning rates (fewer updates!)

- $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_D[\mathbf{w}]$

Online/Stochastic Training: Weight update after **each** training pattern/each batch of random instances

- $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_d[\mathbf{w}]$

- approximation (can in theory run into oscillations)
- faster (fewer epochs!)
- smaller learning rates necessary

Batch Training: Weight update after a batch of training patterns

- A compromise between the two

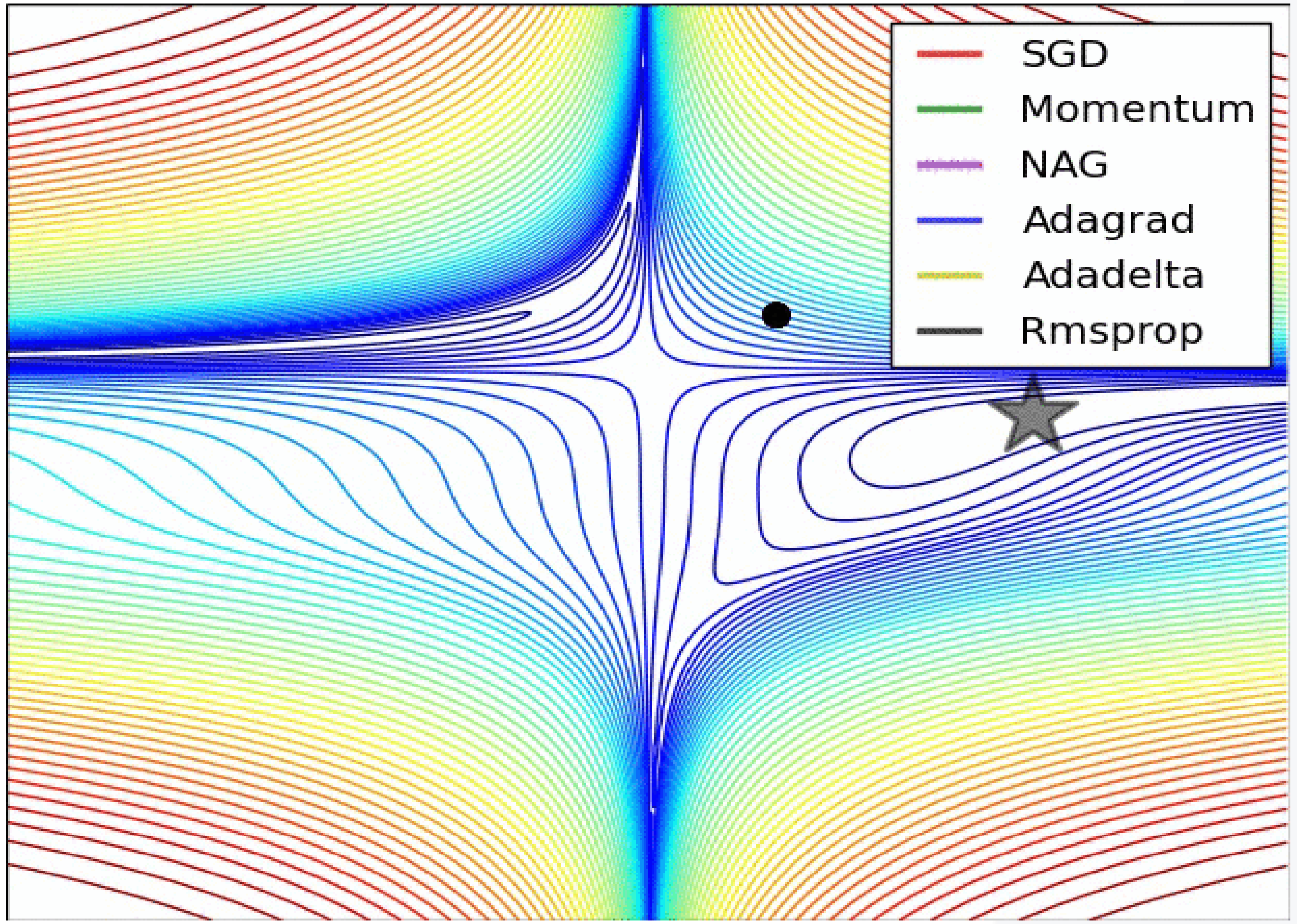
THE UNIVERSITY *of* ADELAIDE

Gradient Descent in Action

There are many different choices for gradient descent:

- SGD
- Momentum
- NAG
- Adagrad
- Adadelta
- RMSprop

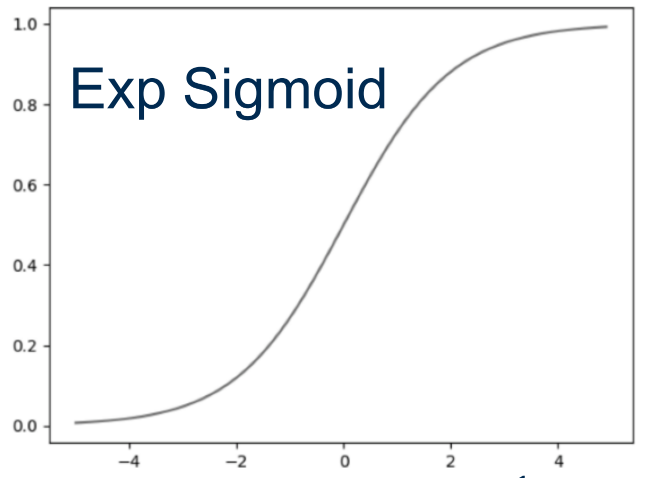
Which one of these works best?



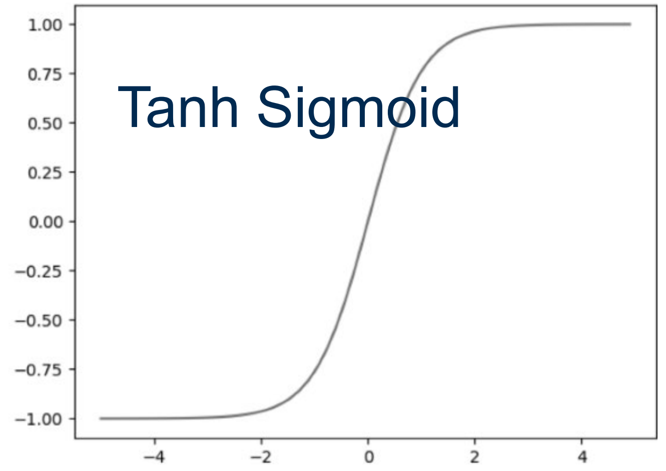
Excellent review of GD methods can be found here:
https://ruder.io/optimizing-gradient-descent/

Activation functions

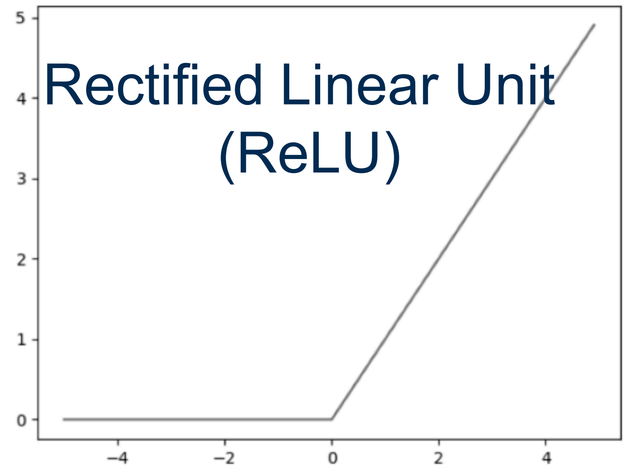
The main change in 1986 was to use BP with sigmoid activation function



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



$$f(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

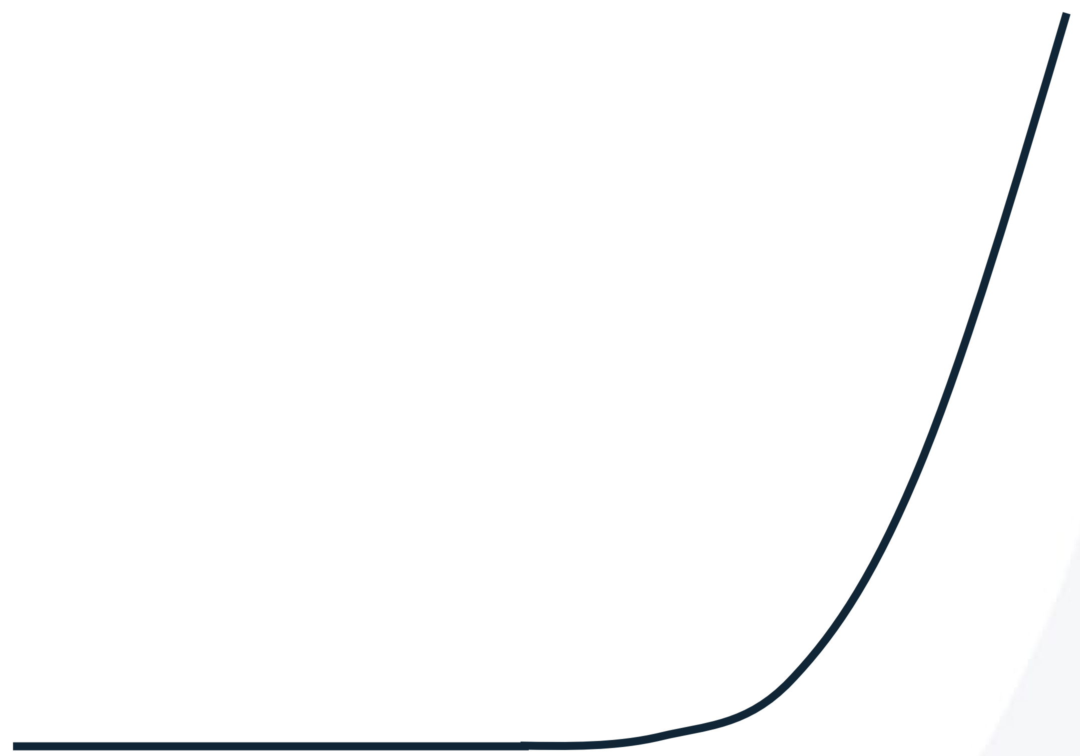


$$f(x) = max\{0, x\}$$

The main reason for sigmoid: it is a differentiable function and can be used to calculate gradient for BP

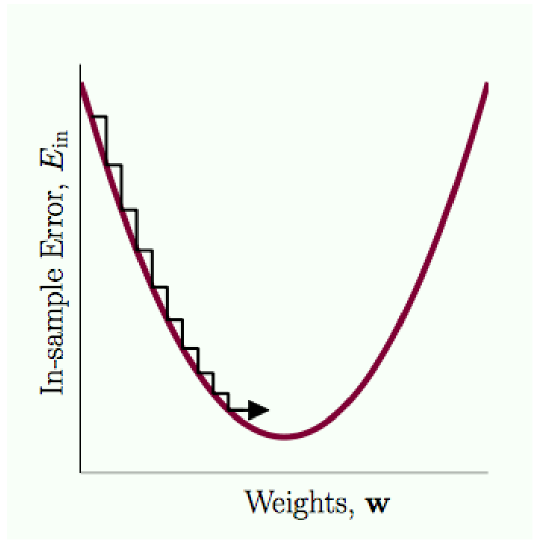
Activation Functions (what makes it good)

- Is the calculation easy?
- Is the function usefully differentiable (non-zero slope)?
- Is the function and its derivative continuous (no discontinuities)?
- Is the slope large or does it disappear to zero?

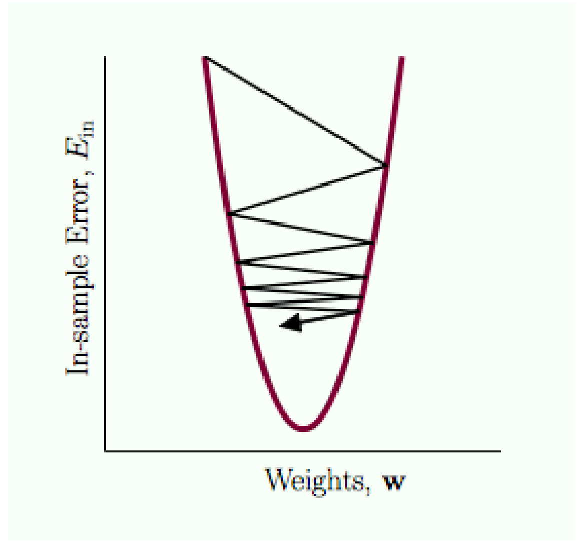


Improving BP

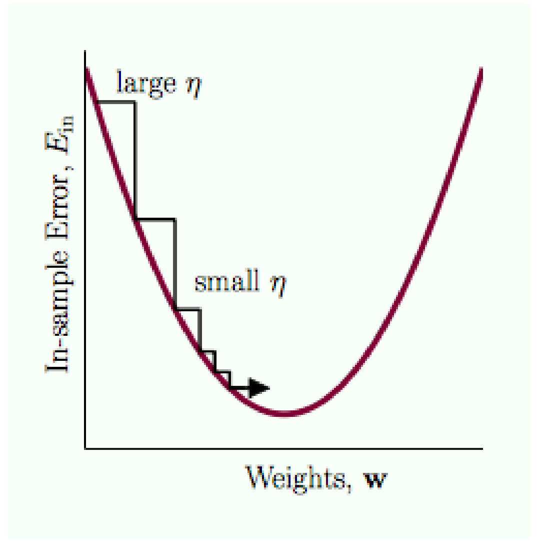
η too small



η too large



η just right



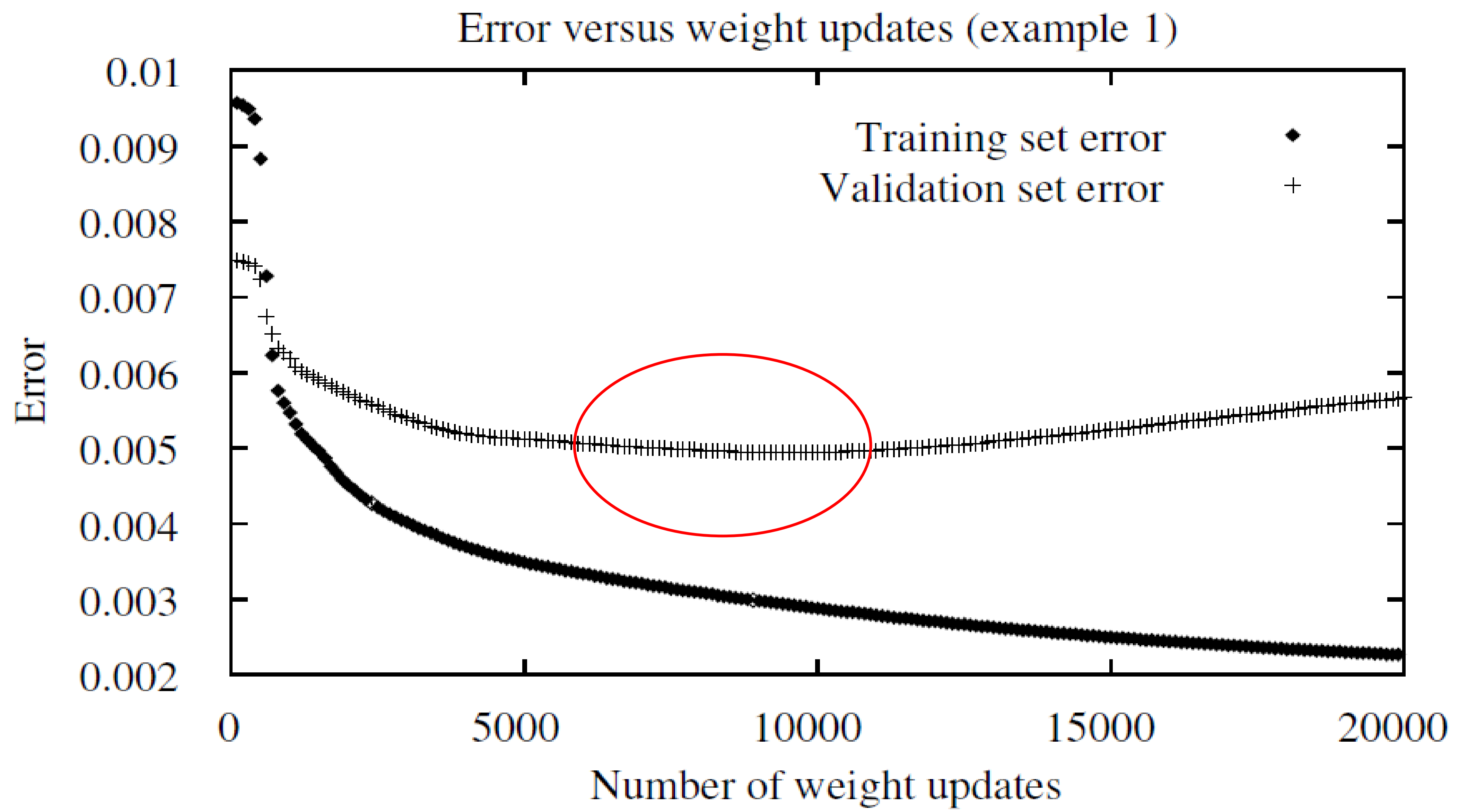
Can be selected using cross-validation, usually <0.5

Use momentum for smoother descent (add moving average of weight)

$$\Delta w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \Delta w_{ji}(n-1)$$

Select initial random weights few times, sometimes the start point may be closer to better (global) minimum.

How much training is needed?



Stop when validation error gets to min

Recap

- What are the similarities and differences between logistic regression and neural networks?

- There is a Neural Network Playground., where you can play with NN parameters while training a network.
  Have you tried it?
  What have you learned, what was interesting? (direct link)

https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261

THE UNIVERSITY of ADELAIDE

NN parallel processing

It’s a Big Data thing!

If we have many, many samples AND we have many, many parameters to calculate gradients for AND we want to do many epochs…

This will balloon out to a huge number!

Instead of calculating all of the required partial derivatives to determine the overall slope…

We take 1 (or a random sample – in practice) to determine the weights and slopes

Mini-Batch Training: Weight update after a batch of training patterns

- Can use Map-Reduce
  - Give a mini-batch to a Map function with current weights **w**
  - If there is no error, Map does not produce key-value pair
  - Else, key-value is produced with the weight update
  - In Reduce stage, all weight updates are summed up.
  - Process another batch

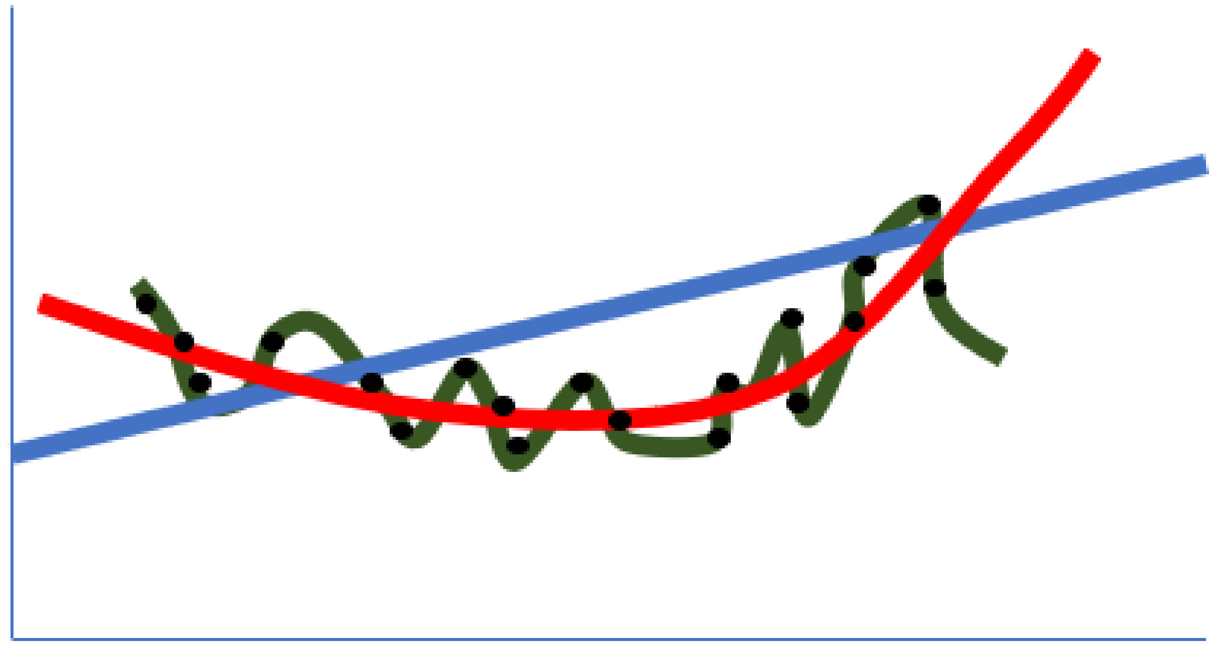
THE UNIVERSITY *of* ADELAIDE

Regularisation

(different types of)

Optimizing a loss function to learn parameters

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Fitting to data}} + \underbrace{\lambda R(W)}_{\text{Choose the simplest model}}$$



Too Simple

Too Complex

Just Right

Why do we need regularisation?

What is loss function L?

How to choose lambda parameter?

Regularisation

$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W), \lambda \geq 0$

We pick **λ** to determine how ‘regular’ we want the data to be. High **λ** will simplify the equation (perhaps to a straight line).

- We choose **λ** with devset or cross-validation.

Ridge (L2 Regularisation)

- $R_{L_2}(w) \triangleq \|W\|_2^2$

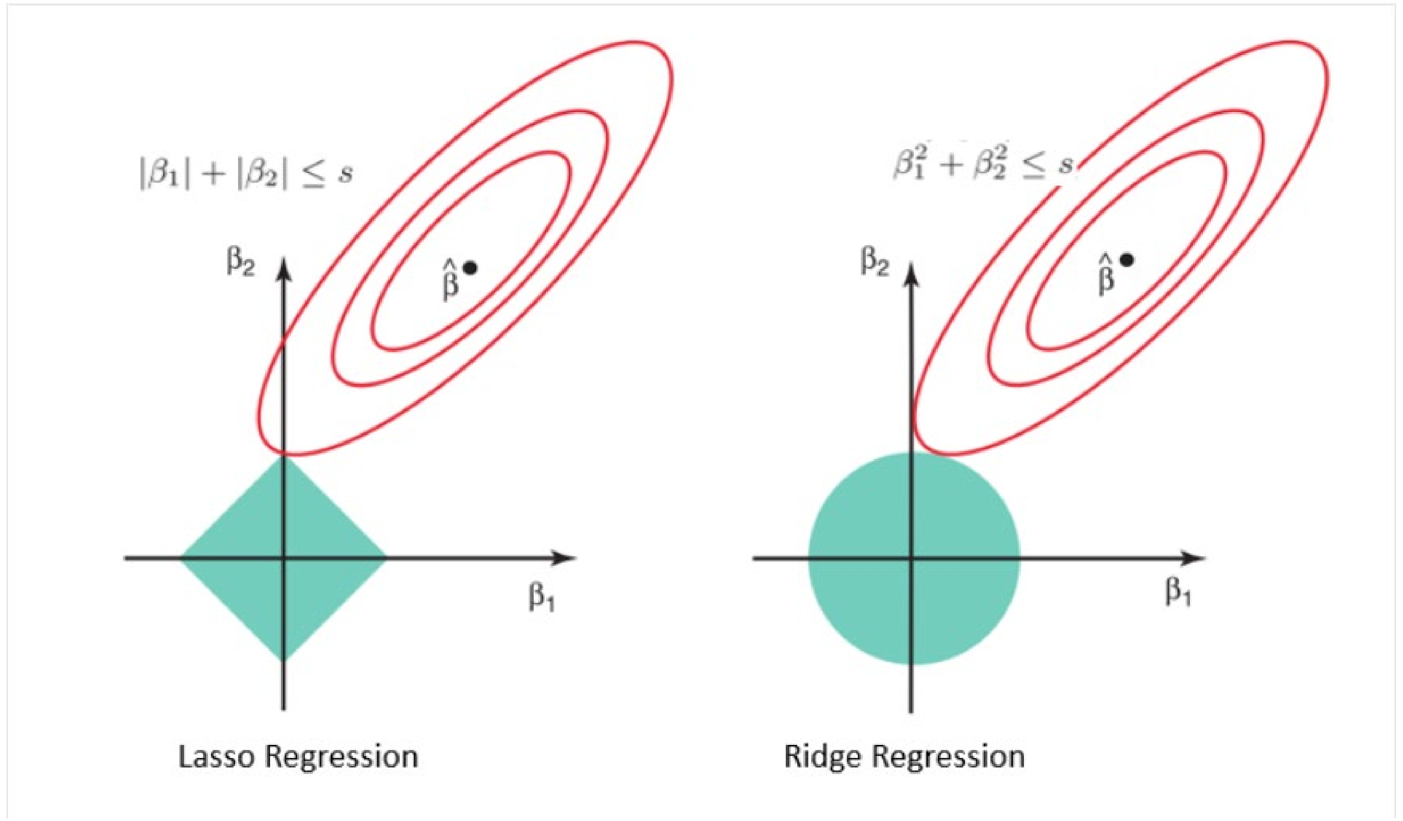
Lasso (L1 Regularisation)

- $R_{L_1}(w) \triangleq \sum_{k=1}^{Q} \|W\|_1$

Ridge vs Lasso

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W),$$
$$\lambda \geq 0$$

$|W|$ vs W^2



Ridge

Increasing **λ** will never result in a straight line (constant) result (it will asymptote it to it.

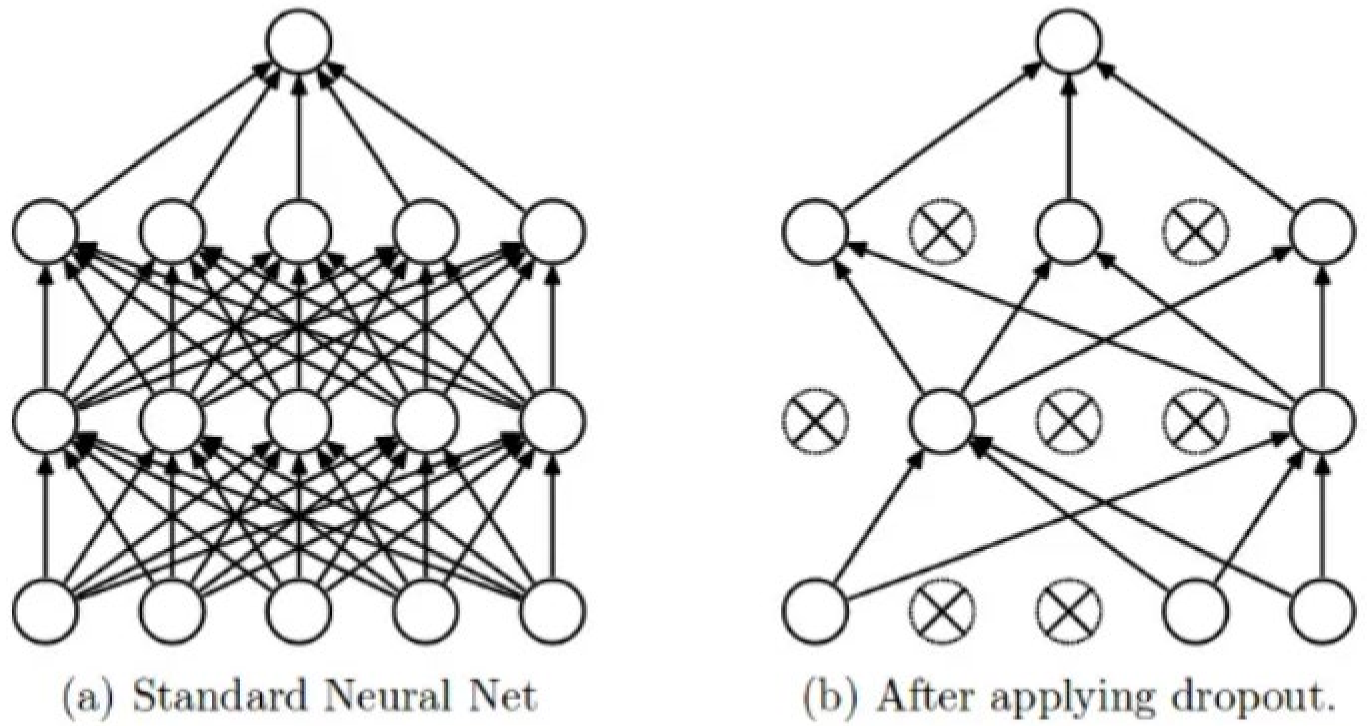
Lasso

Increasing **λ** can result in a straight line (constant), meaning it can completely negate irrelevant inputs!

Dropout Layers in NN

Why dropout?

- The big problem is over-fitting
- The logic of drop out is that because not every node is present for every training example:
  Individual pathways through the network can't be too specialised and the network needs to have 'back-up plans'



Recap

1. How does regularisation reduce overfitting? Explain this given the cost function with regularisation.
2. What is w gradient of L1 regularisation part of loss function $\lambda|w|$

$$L_1 = (wx + b - y)^2 + \lambda|w|$$

$$L_2 = (wx + b - y)^2 + \lambda w^2$$

https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261

SUB CRUCE LUMEN

THE UNIVERSITY of ADELAIDE