Ensemble/
Optimise your Neural Network

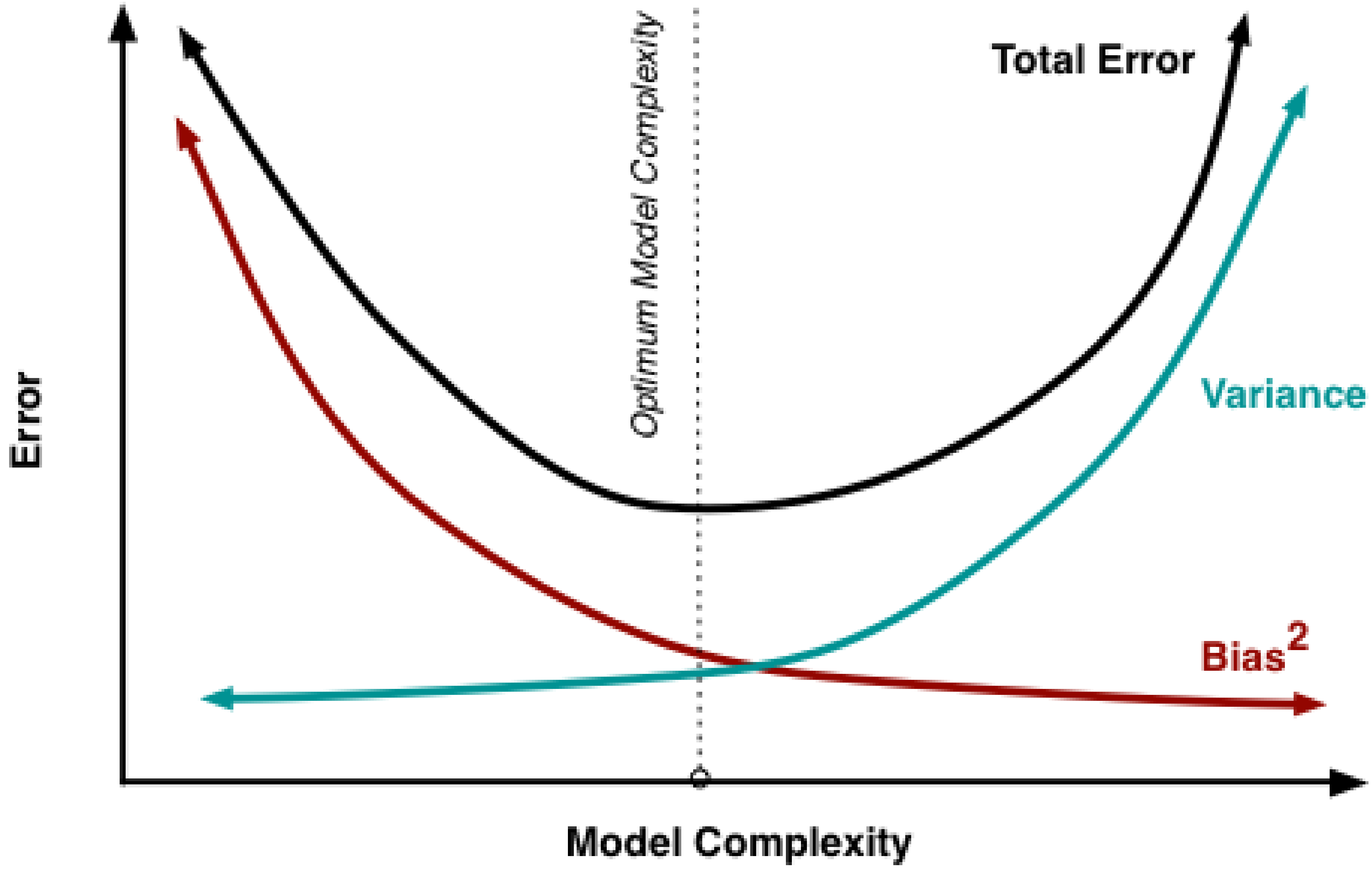
Tianchu.Zhao@uts.edu.au

Bias and Variance

- bias: measures the difference between model prediction and real world value
- variance: measures how much a model's performance will be affected when change happens in the data

Types of ensemble

- Bagging
  - Stands for bootstrap aggregation
  - decrease variance through generating data from original data
  - the data is generated through combination with repetition to produce multiple sets of data that’s the same size of your original data
- Boosting
  - first produce multiple models from subsets of the original data
  - then combine the prediction output from models (e.g. majority vote)
- Stacking
  - similar to boosting but your predicted results are then added back to the original data for further training/prediction (blending).



Total Error
Variance
Bias^2
Optimum Model Complexity
Error
Model Complexity

Why ensemble works

- Ensemble averages bias -> Unlikely to overfit
- Ensemble reduce variance

How to optimise your neural network

- Data split
- Regularization
- Gradient Descent
- Hyperparameter

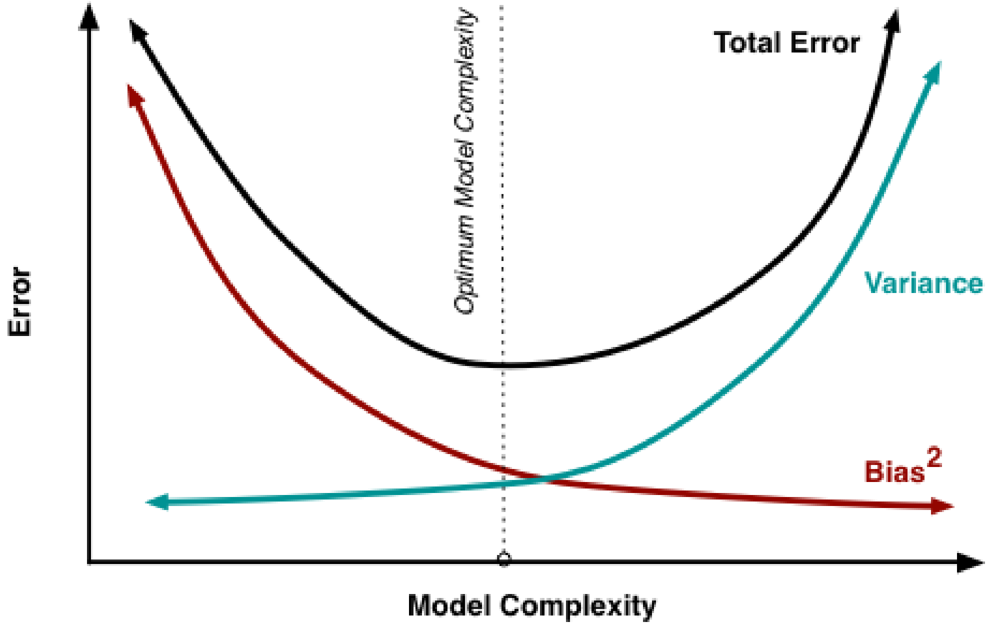
Data Split

- Traditionally what we learn
  - Data are split into training, development(validation), testing set
  - in a 60%/20%/20% configuration

- In Big Data world, due to the size of data
  - a 98%/1%/1% configuration is more appropriate

Model fit

- underfitting will result in high bias -> unable to well calssify the data
- overfitting will result in high variance -> unable to adapt to new data
- we need to find the optimal point between the two:



Find out the problem

- training set low error, validation set big error
  - -> high variance, overfitting
- training set and development set have similar error, not low
  - -> high bias, underfitting
- training set high error, development set even higher
  - -> high variance, high bias, bad model
- training set low error, development set low error, low difference between two
  - -> low variance and bias, good model

Solve the problem

- high variance:
  - bigger network, more layers, more neuron in layers
  - suitable network architecture, hyperparameters
  - longer training time, better optimization algorithms
- high bias:
  - more data
  - regularization
  - suitable network architecture

Regularization 1 – regularization term

- penalise model complexity by adding regularization to the Cost function

- originally we have cost function

$$C = \frac{1}{m}\sum_{j=1}^{m}(a_j^L - y_j)$$

- now we add a regularization term

$$C = \frac{1}{m}\sum_{j=1}^{m}(a_j^L - y_j) + \lambda$$

- The regularization is defined as
- for L2

$$L_2 : \lambda = \frac{\lambda}{2m} w^2 = \frac{\lambda}{2m} \sum_{j=1} w_j^2 = \frac{\lambda}{2m} w^T w$$

- for L1

$$L_1 : \lambda = \frac{\lambda}{2m} |w| = \frac{\lambda}{2m} \sum_{j=1} |w_j|$$

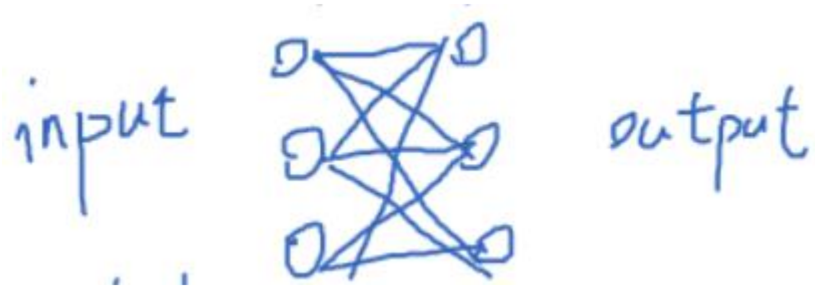
- Regularization penalise for large weight
  - eg the regularization term will have a large value if the weight is large thus increase the cost
- This reduces the impact of individual weight
- Makes the sigmoid function closer to linear

$$z^L = w^L a^{L-1} + b^L$$

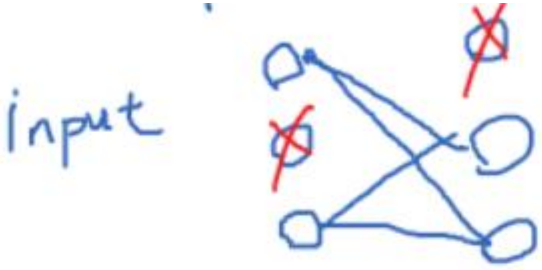
$$a^L = \sigma(z^L)$$

Regularization 2 – Dropout

- initialise mask with probability that a neuron will be 0
- without dropout



- with dropout with 0.6 probability of keeping weights of neuron at a layer

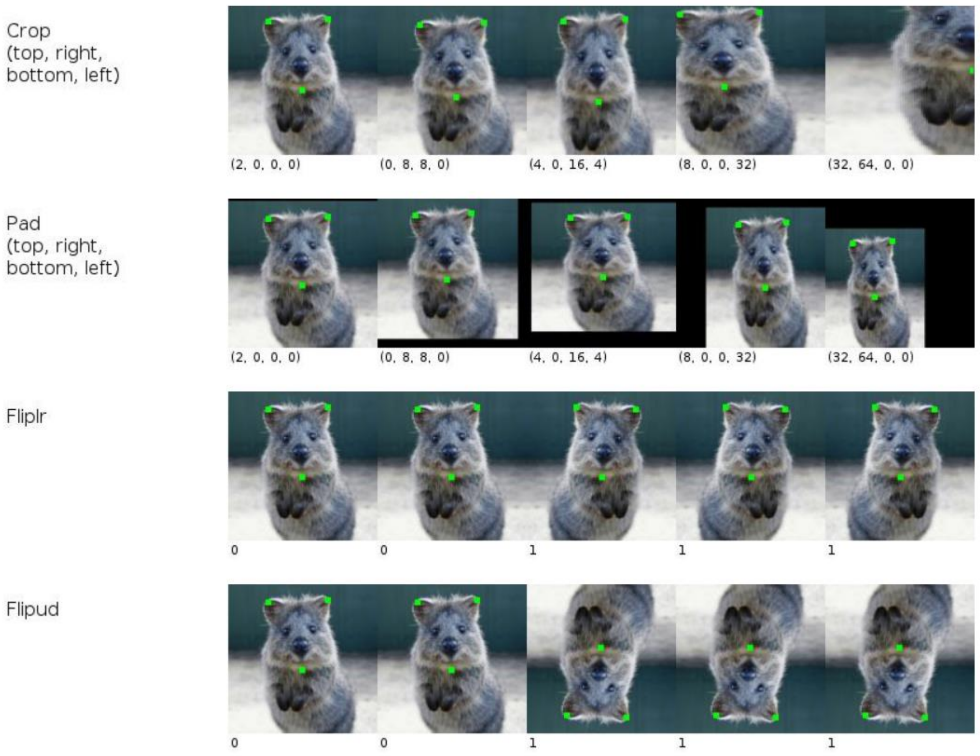


- Intuition
  - when the weight in the neural network converges, the neural network is not replying on any single weight

- In prediction
  - the mask is replace by expectation instead of binary (eg 0.6 in this case)

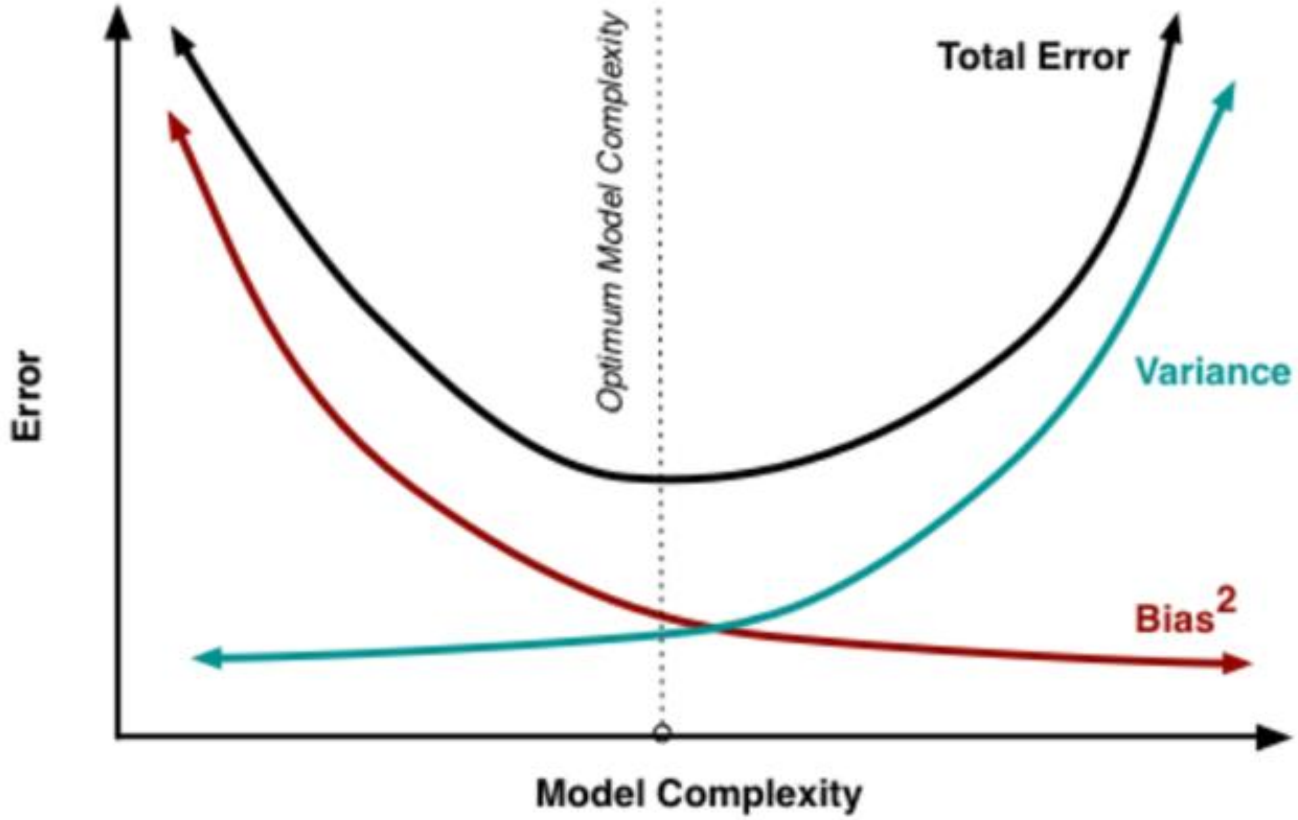
Regularization 3 – Data Augmentation

- Introduces variance in to the data, makes it better adapt to future changes
- https://github.com/aleju/imgaug



Regularization 4 – Early Stopping

- stop training when validation error start to diverge from training error



Gradient Descent

- Gradient Descent Strategy
  - Types
  - Normalisation
  - Momentum
  - RMSProp
  - Adam
  - Weight Decay

- Types
  - Batch
    - run through the whole training data, then take a step
    - this can be very slow if you have a large amount of data
  - minibatch
    - take a step gradient descent after calculating through a portion of the whole data
    - the unit of each run through of the whole data is epoch
    - within 1 epoch we can have multiple mini batches descent
  - stochastic
    - take a step gradient descent for every training data point

- batch:
  - every descent takes through the whole data, takes a long time, slow
  - unaffected by the noise within data, step is bigger
  - lost always face the lowest direction
- stochastic:
  - every descent takes through 1 data sample,
  - lots of noise, smaller learning rate is more suitable
  - lost overall face lowest direction
- minibatch takes the benefit of the two


batch
minibatch
stochastic

Normalisation

- data normalisation
- batch normalisaiton

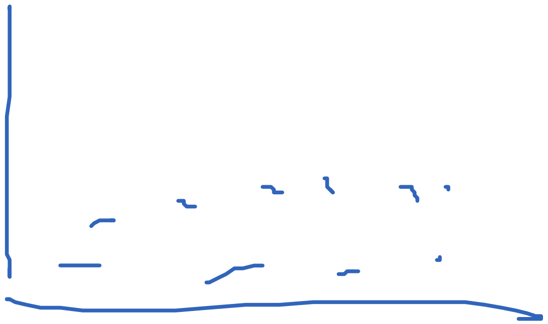
- data normalization

$$x = \frac{x-\mu}{\sigma}$$

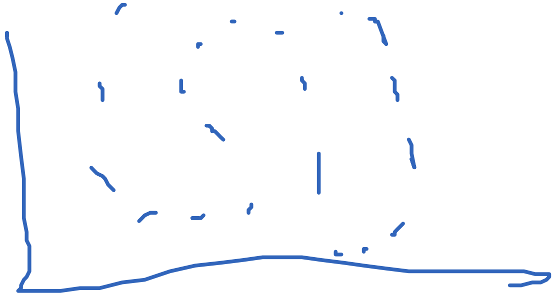
$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{i}$$

$$\sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m} (x^{i}-\mu)^{2}}$$

from

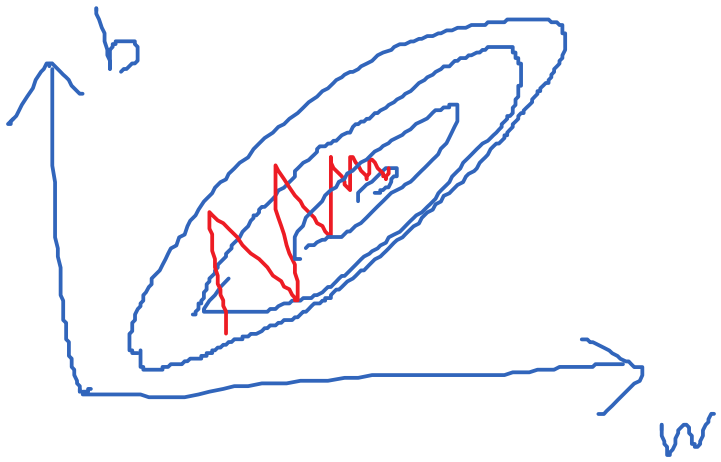


to



- normalization can perform with larger learning rate requires less iteration to reach minimum

without normalisation



- batch normalization

$$\mu = \frac{1}{m} \sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z_i - \mu)^2$$

$$z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

- normalisation stablise gradient direction in Cost function
- the weight change from the previous layer has less effect to the current layer, more stablise network
  - eg it introduce noise because the normalisation and make later layer less dependeng on the current layer

Exponentially weighted averages

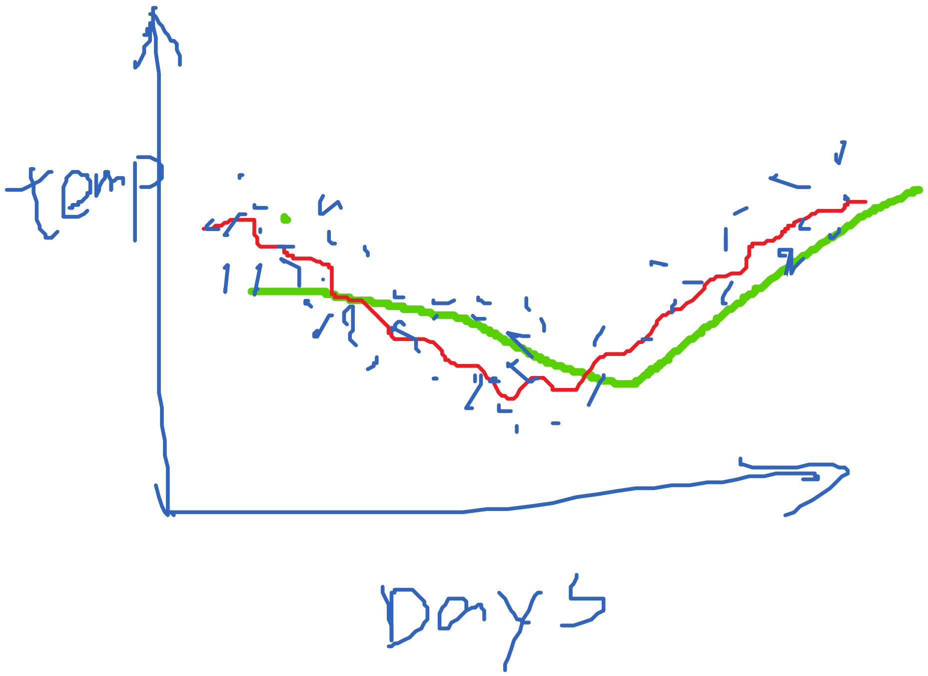
$\theta_1 = 30\,°C$

$\theta_2 = 31\,°C$

$\theta_2 = 29\,°C$

$\vdots$

$\theta_{365} = 30\,°C$



- compute the trend (moving average)

$$V_0 = 0$$

$$V_1 = 0.9 V_0 + 0.1 \theta_1$$

$$V_2 = 0.9 V_1 + 0.1 \theta_2$$

$$\vdots$$

$$V_t = 0.9 V_{t-1} + 0.1 \theta_t$$

- general form

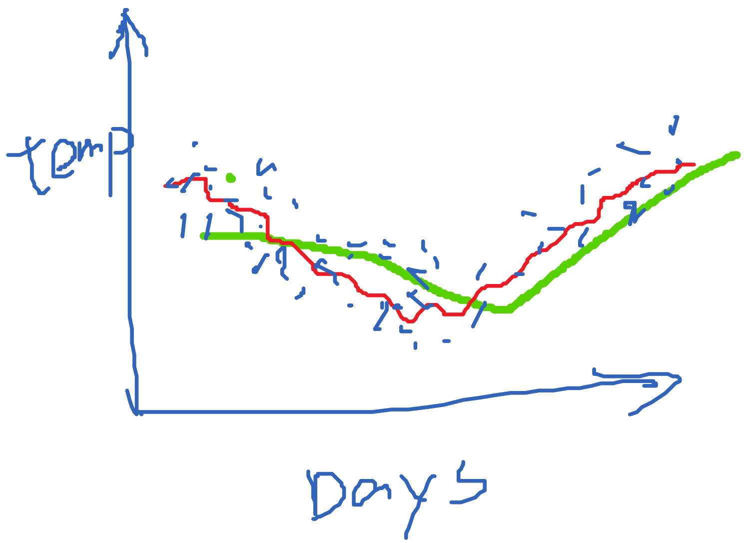
$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

- Vt is the approximately average over

$$\approx \frac{1}{1-\beta} \text{ days}$$

$\beta = 0.9 : \approx 10$ days

$\beta = 0.98 : \approx 50$ days



- Use bias correction to correct to fix the shifted line problem

$$\frac{V_t}{1-B^t}$$

$t=2$: $1-B^t = 1-(0.98)^2 = 0.0396$

$t=20$: $= 0.3324$

$t=200$: $= 0.9824$

Momentum

for l = 1, .. , L:

$$v_{dW^{[l]}} = \beta v_{dW^{[l]}} + (1-\beta)dW^{[l]}$$

$$v_{db^{[l]}} = \beta v_{db^{[l]}} + (1-\beta)db^{[l]}$$

$$W^{[l]} := W^{[l]} - \alpha v_{dW^{[l]}}$$

$$b^{[l]} := b^{[l]} - \alpha v_{db^{[l]}}$$

RMSProp (Root mean square propagation)

- e is usually 10^-8, it is use for preventing divide by 0 error
- when db/dw is large,
- (db)^2/(dw)^2 is large,
- Sdw/Sdb is large,
- dw/sqrt(Sdw+e) is small
- db/sqrt(Sdb+e) is small

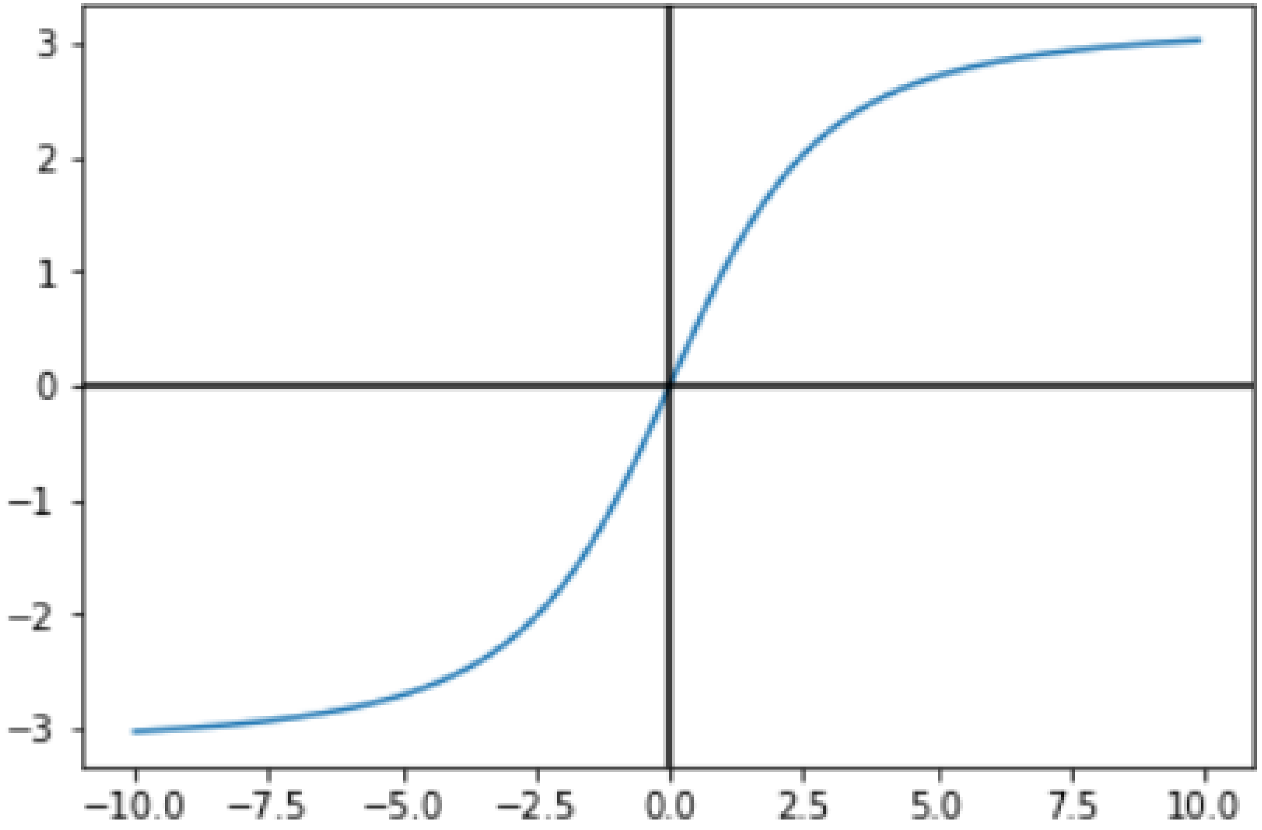
$$s_{dw} = \beta s_{dw} + (1-\beta)(dw)^2$$

$$s_{db} = \beta s_{db} + (1-\beta)(db)^2$$

$$w := w - \alpha \frac{dw}{\sqrt{s_{dw} + \epsilon}}$$

$$b := b - \alpha \frac{db}{\sqrt{s_{db} + \epsilon}}$$

- drawing dw holding Sdw constant



Adam (momentum + RMSprop)

- in RMSprop replace db with Vdb

Learning rate decay

- if using a fix learning rate, at near minimum, due to the noise in batch, it won't converge accurately, and will bounce within a large range of cost values
- a strategy to overcom this is leanring rate decay,
- such that we use a larger learning rate at the beginning for faster descent,
- and use a smaller learning rate as the training time increase
- common learning rate decay formula

$$\alpha = \frac{1}{1 + decay\_rate * epoch\_num} * \alpha_0$$

Hyperparameters

Parameters (the information that the model will figure out itself)

- w: weight
- b: bias

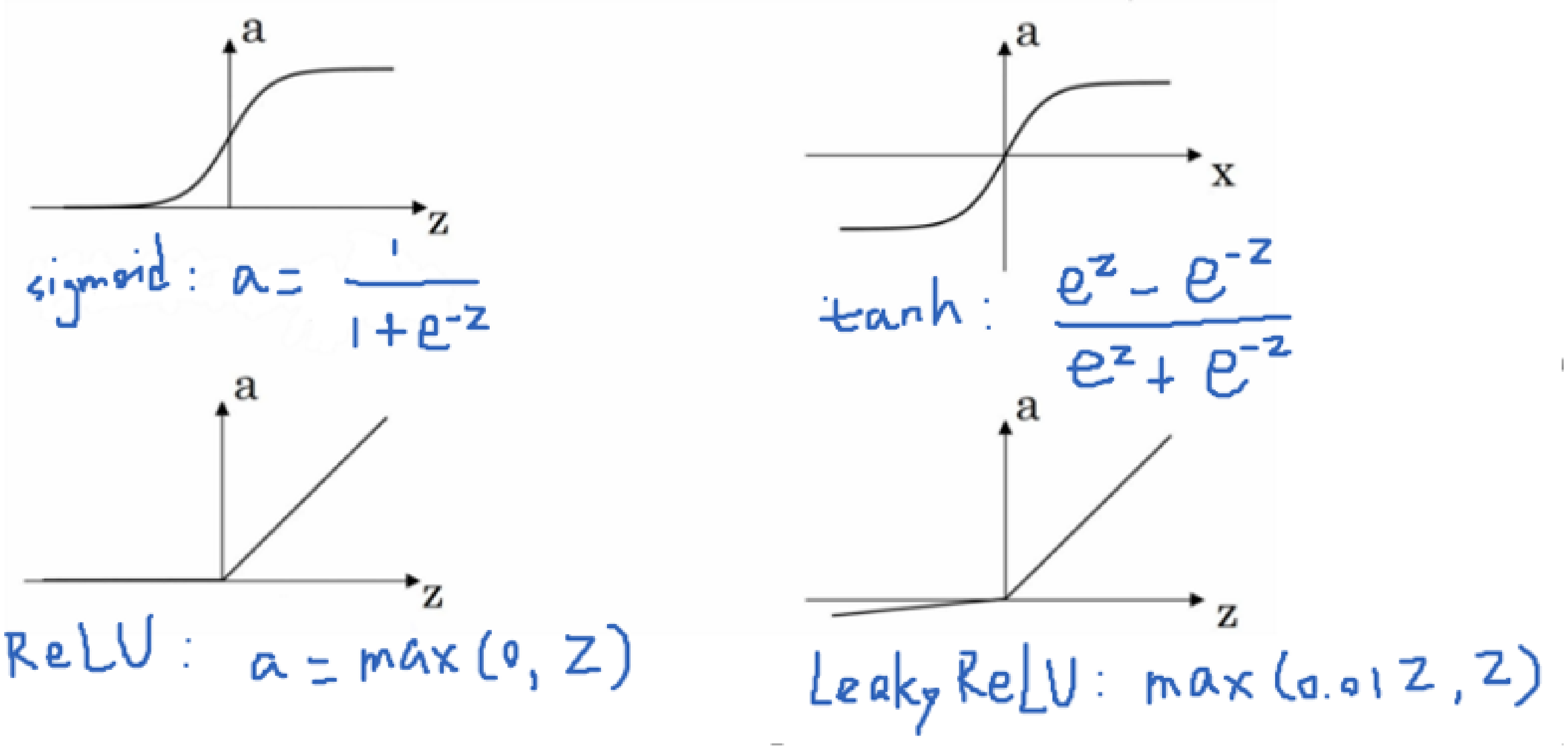
Hyper parameters (tunable)

- a: learning speed
- N: number of iteration
-
- size of minibatch
- B: momentum (set)
- (noted when B closer to 1, even a small change will result in large sensitivity change, consider (1/(1-B))
- B1, B2, e: Adam parameter: 0.9, 0.999, 10^-8
- decay_rate
- droppout
- adam parameters (set)
-
- L: number of layer
- n: number of neurons within each layer
- a/g(z): activation function

Activation functions

- this introduces the nonlinearity within the model,
- from the lecture, if we don't have actication function, the neural network is simply multiple matrices chaining together, this is no different to a linear model.

- Four kinds of common activation functions



- Tanh almost always better than sigmoid
- but both tanh and sigmoid have problem when z is very large or very small
- when z is very large or very small, the gradient is almost 0 and this slows down the gradient calculation

- That's why we hae ReLU, ReLU stands for rectified linear unit
- when $z>0$, the gradient is always 1, thus dramatically speeds up the computation.
- although when $z<0$ the gradient is 0, this doesn't have huge impact to the model

Grand Plan (besides the lecture material)

- (✓) Neural Network Foundation
- (✓) Ensemble/Optimise your neural network
- (☐) Convolution Neural Network
- (☐) Recurrent Neural Network
- (☐) Generative Adversarial Network (+ unsupervised + symmetric nn)
- (☐) Reinforcement Learning
- (☐) Big data
- (☐) Timeseries/Natural Language Processing