COMPSC 361
Advanced Neural Networks

Neural Networks III

Instructor : Thomas Lacombe

Week 11

Disclaimer/Acknowledgment:

The following slides are reusing some of the content created by Andrew Ng in his book “Machine Learning Yearning” and his Coursera course about Improving DNN.

https://github.com/daiwk/ml-yearning/blob/master/Ng-MLY01-13.pdf

https://www.deeplearning.ai/courses/deep-learning-specialization/

Outline

Introduction

Artificial Neural Networks (ANN)

- Single Unit: Architecture of Perceptron (NN1)
- Connection to Shallow Machine Learning (NN1)
- Multi-Layer Feed-Forward Neural Network (NN2)

Design Issues (NN3)

Deep Learning / Large Language Models (NN4)

Empirical and iterative process

Neural network design:

Lots of choices to make!

- Evaluation:

Cost function

Evaluation strategy

- NN hyperparameters:

Number of layers

Number of neurons per layer

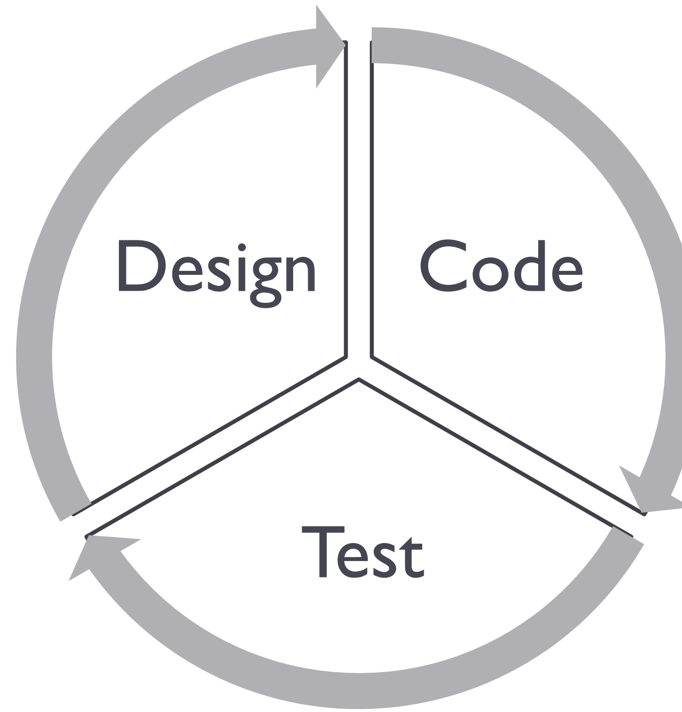
Activation functions

Learning rate

Weight initialisation

Mini-batch size

…



Design issues: Evaluation strategy

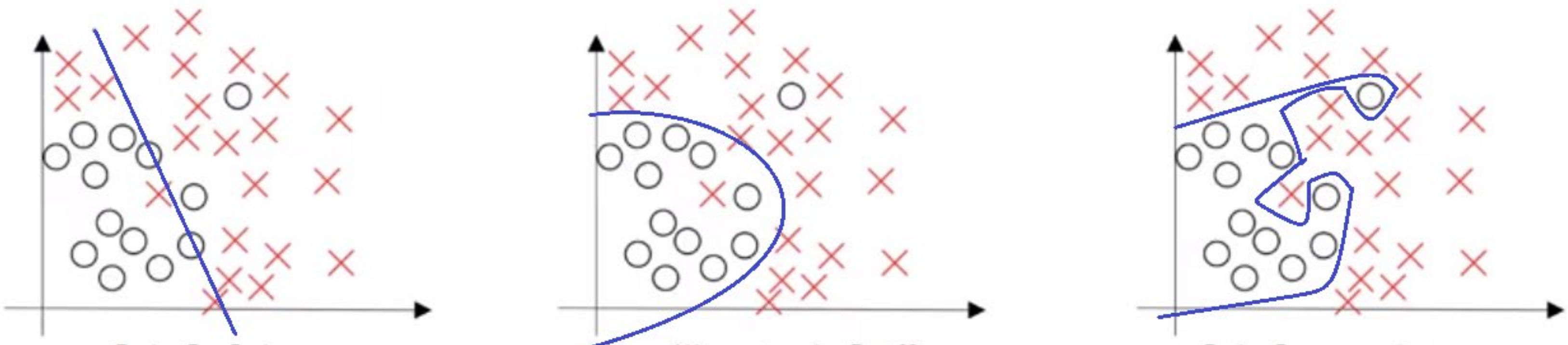
- Train/dev/test sets with deep networks

- If small dataset (100, 1000, 10 000 samples)
  - 60%/20%/20%
- If large dataset (> 1 000 000)
  - 98%/1%/1%

- Training set and dev/test set usually need to come from same distribution (but it is ok if it varies a bit when gathering a lot of training data).
- Make sure dev and test sets come from the same distribution.

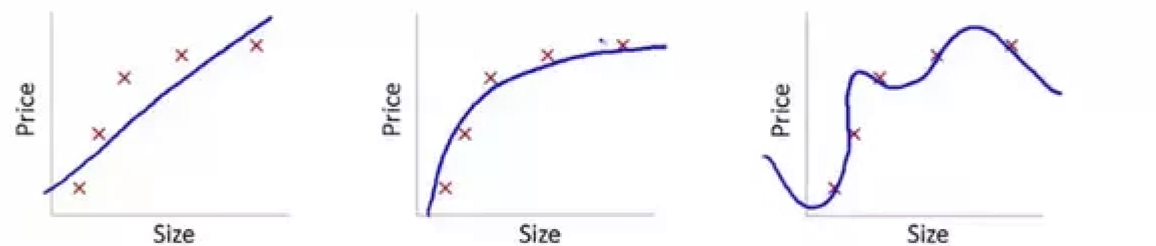
How does your model do?

- Bias vs variance / underfitting vs overfitting



high bias | "just right" | high variance

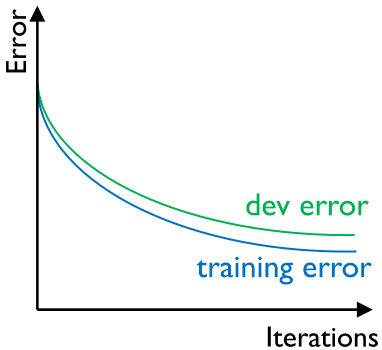
Underfit | | Overfit



$\theta_0 + \theta_1 x$ | $\theta_0 + \theta_1 x + \theta_2 x^2$ | $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

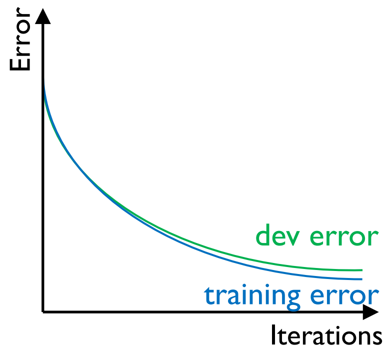
High bias (underfit) | "Just right" | High variance (overfit)

How does your model do?



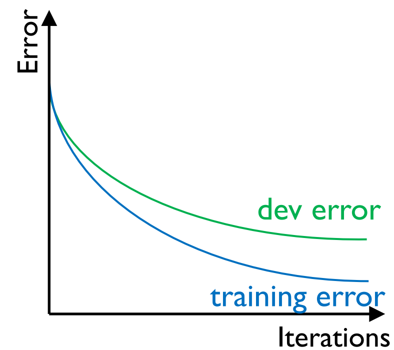
training error = 15%
dev error = 16%

High bias
Underfitting



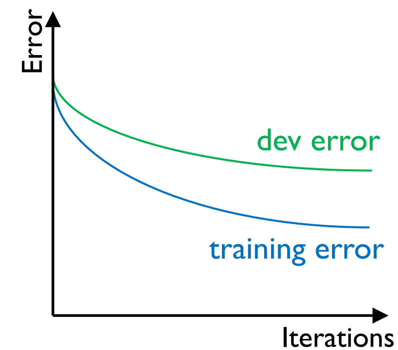
training error = 3%
dev error = 4%

Low bias
Low variance
Good performance



training error = 3%
dev error = 15%

High variance
Overfitting



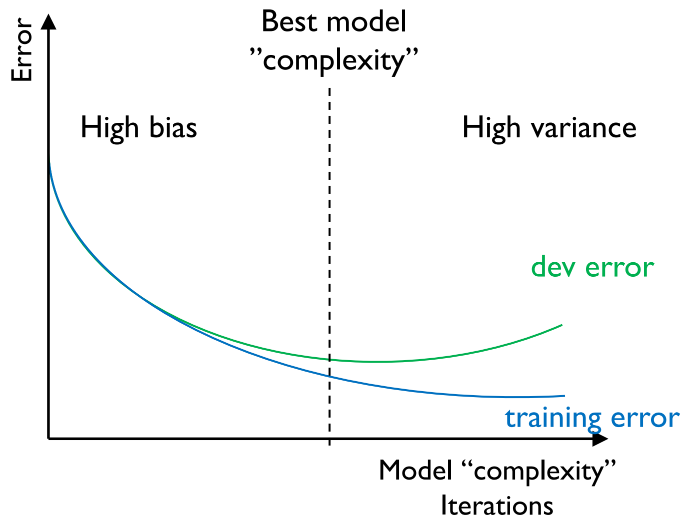
training error = 15%
dev error = 35%

High bias
High variance

https://towardsdatascience.com/how-to-efficiently-design-machine-learning-system-caef9e05d5f

How to improve learning? Overfitting

- Very common problem with Deep Learning: overfitting



- Regularisation = discouraging learning a more complex model
- Reduces the variance, but increases the bias

Design issues: Regularisation

"Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error."

Deep learning, 5.2.2, p.117

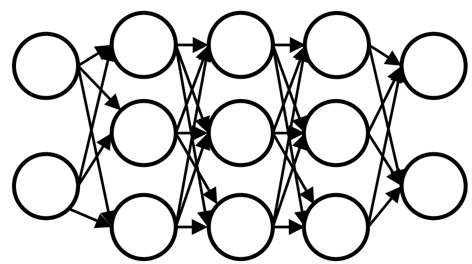
- Different technics:
  - L1 & L2 regularisation
  - Dropout
  - Early stopping
  - Data augmentation

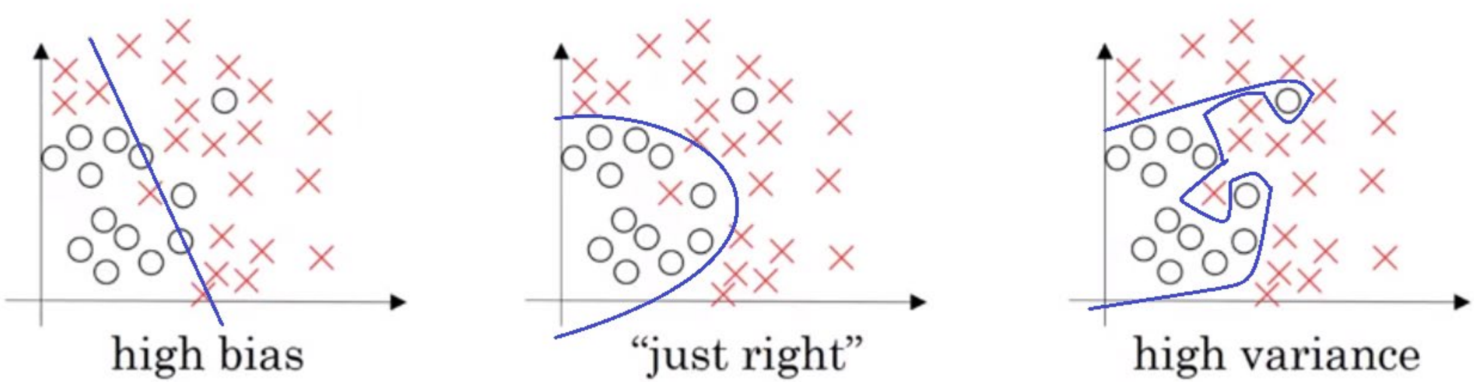
https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

How does regularisation help to avoid overfitting?

First intuition:

- E.g., L1 & L2 regularisation penalises large weights values.
- Keeping weights close to zero for some neurons.

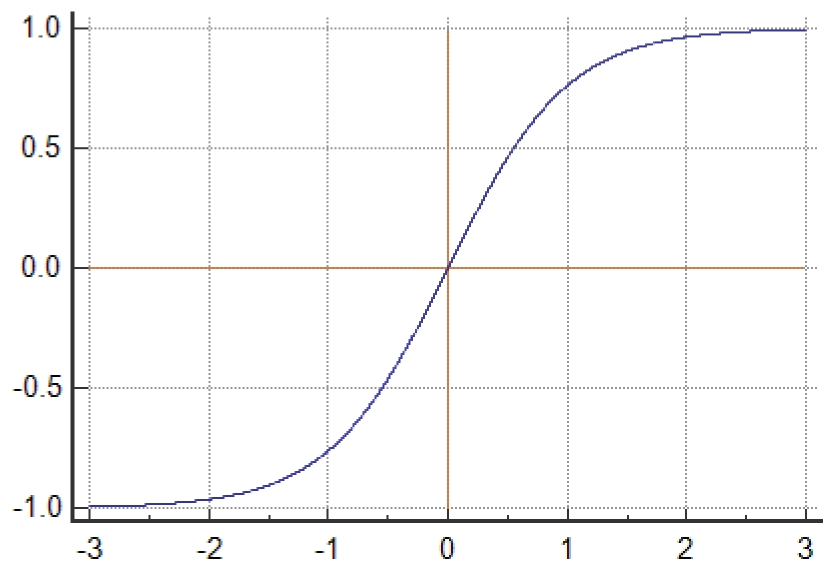




How does regularisation help to avoid overfitting?

Second intuition:

- Limiting the weights values will bring the output of a neuron in the linear zone of activation function (e.g. tanh).
- It will limit the NN power to model non-linearities.



L1 and L2 regularisation

- Why use it?

Large weights:

- are characteristic of more complex models (higher learning time).
- can be a sign of an over-specialized network (overfitting).
- make the network unstable (sensitive to noise).

Penalises/constrains the weight values towards 0.

- A "weight shrinkage" or a "penalty against complexity"
- Encourages simpler models.

L1 and L2 regularization

L1 and L2 norms

$$||\mathbf{w}||_1 = |w_1| + |w_2| + ... + |w_N|$$
$$\|\mathbf{w}\|_2 = \left(|w_1|^2 + |w_2|^2 + \ldots + |w_N|^2\right)^{\frac{1}{2}}$$

- How does it work?

1. Calculate the weights size
   - Sum of the absolute values of the weights → L1. $\sum_{i=1}^{N} |w_i|$
   - Sum of the squared values of the weights → L2. $\sum_{i=1}^{N} w_i^2$

2. Apply regularisation to the weight update
   - L1 regularisation : $Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$
   - L2 regularisation: $Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$

λ controls the penalty

$$0 < \lambda < 1$$

L1 vs L2 regularisation

- Weight update: $w \leftarrow w - \frac{dL}{dw}$
- L1: $$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$
- L2: $$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

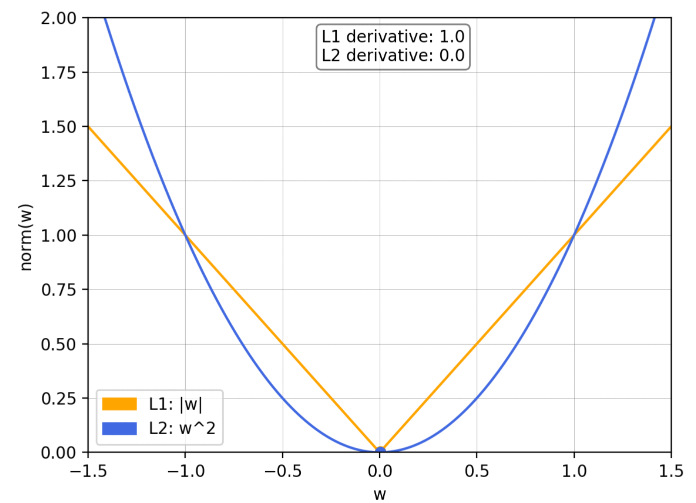


Image source: https://towardsdatascience.com/visualizing-regularization-and-the-l1-and-l2-norms-d962aa769932

→ L2 penalises more large weights and less small weights than L1.

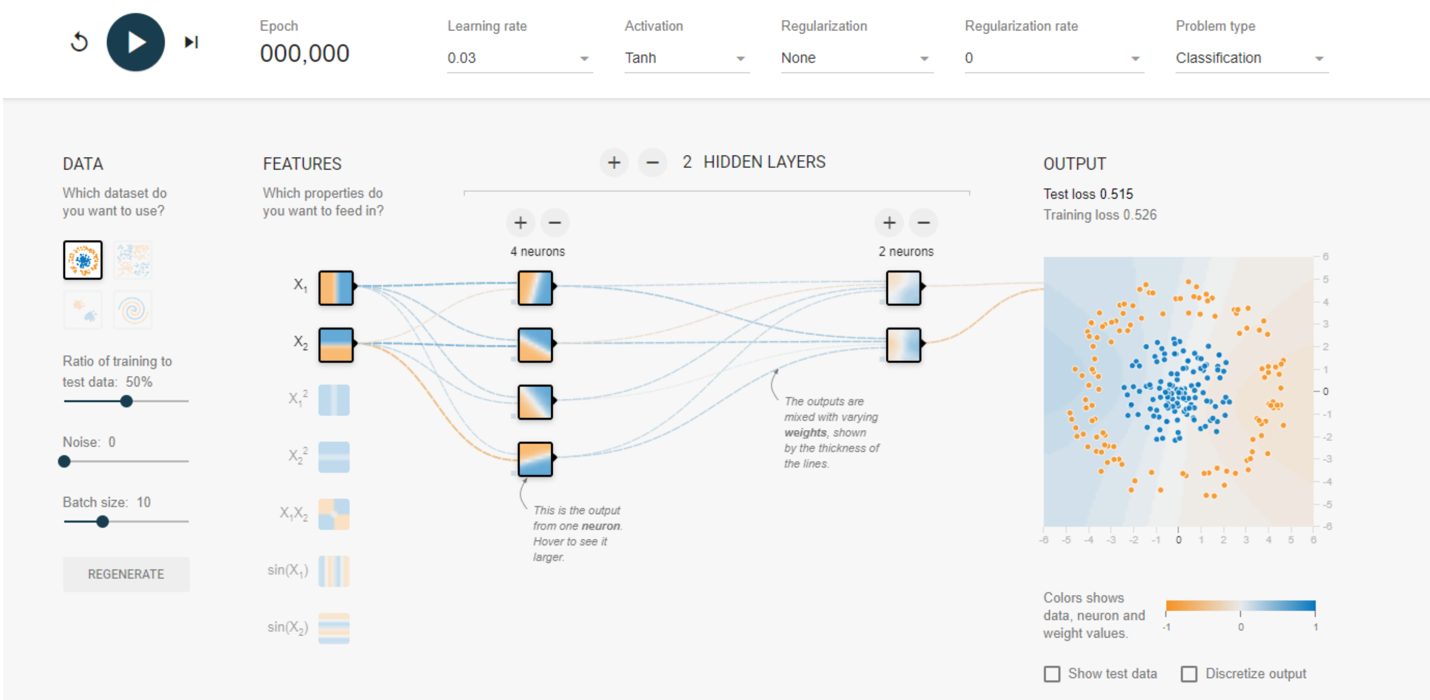
→ L1 shrinks weights to 0 while L2 shrinks weights evenly.

L1 vs L2 regularisation

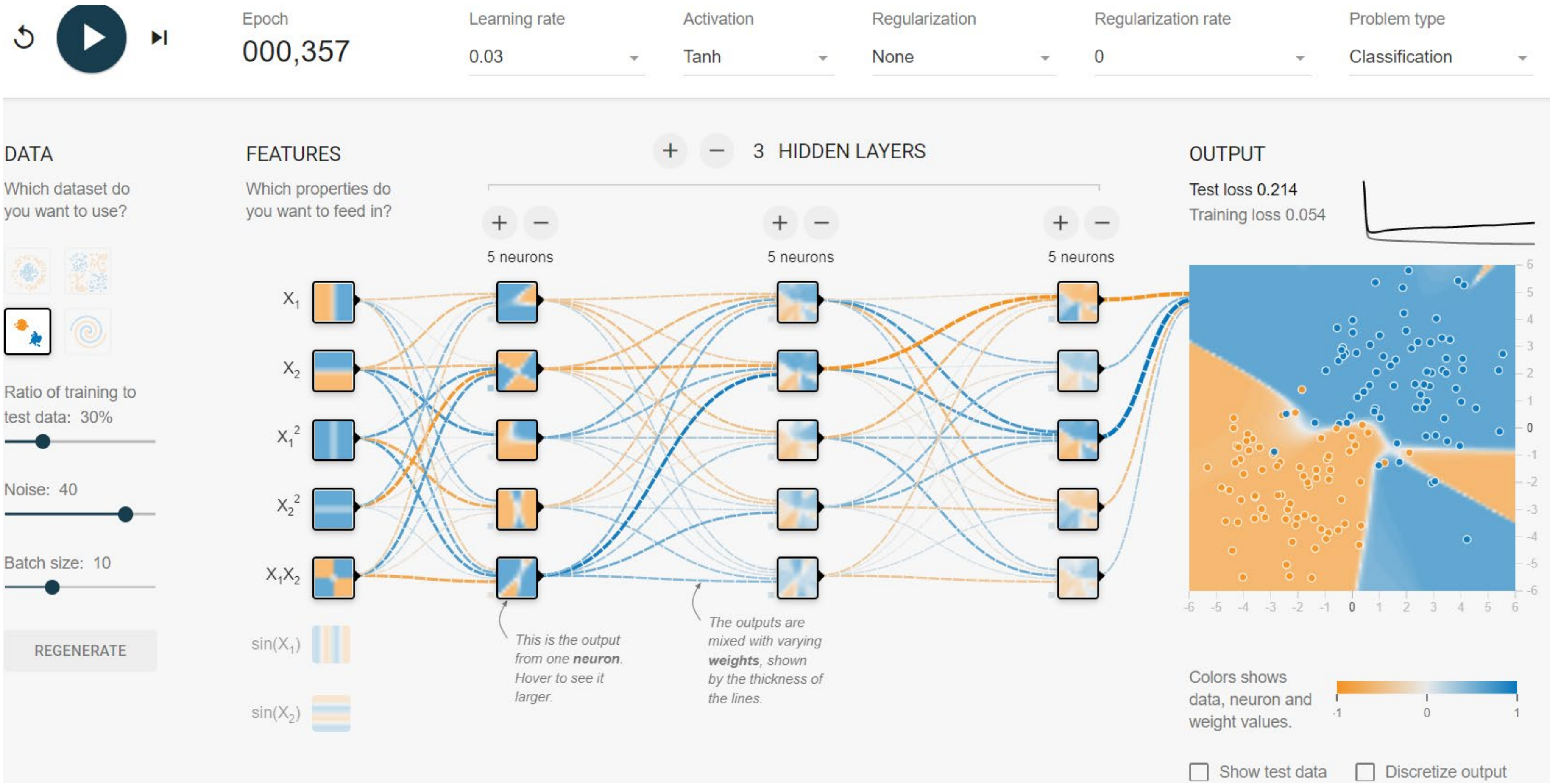
- L1 regularisation results in a sparse weight matrix (a lot of weight values to 0).
- L1 regularisation is acting as feature selection, dropping irrelevant features.
- L2 regularisation results in less sparse weight matrix than L1, and it will reduce the effect of collinear features.
- Penalising the weights forces the NN to “focus” more on simpler features that explain most of the variance, than on complex ones.

Tensorflow playground platform

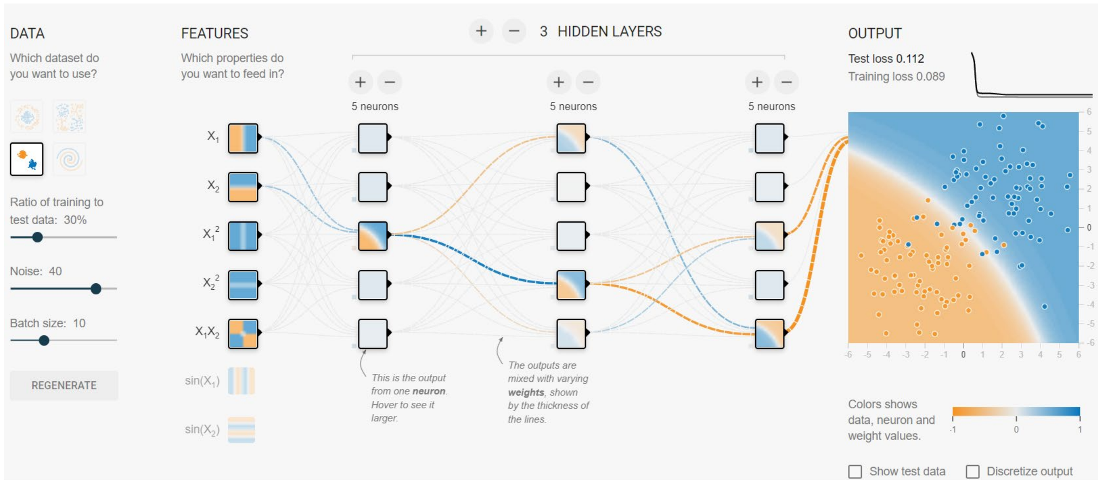
- Platform to test and visualise the effects of varying hyperparameters: https://playground.tensorflow.org/



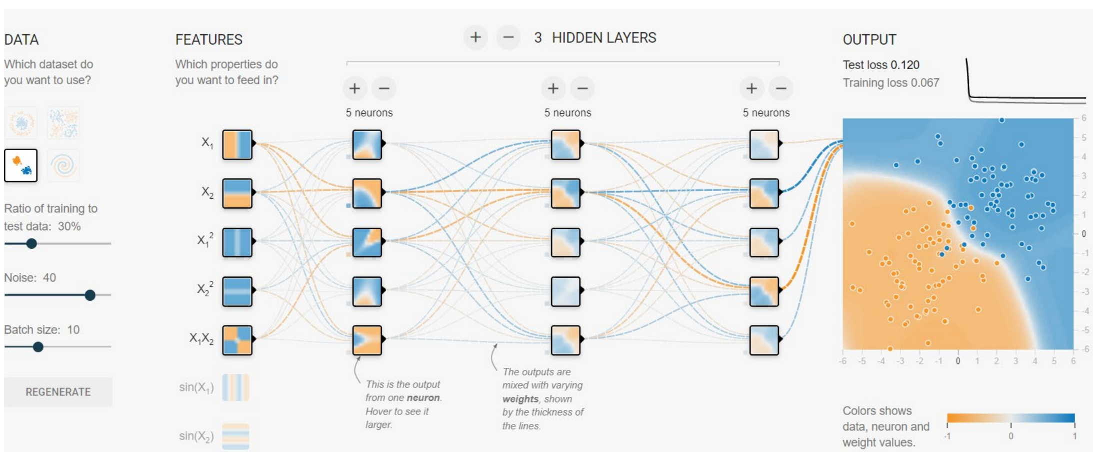
Without regularisation



With regularisation



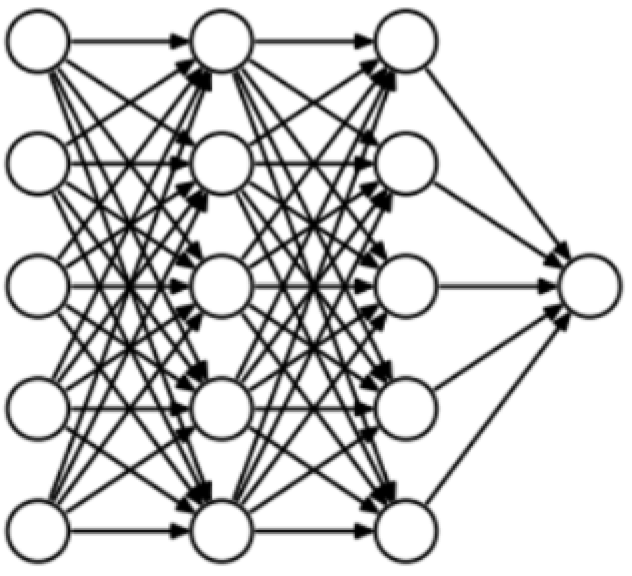
L1



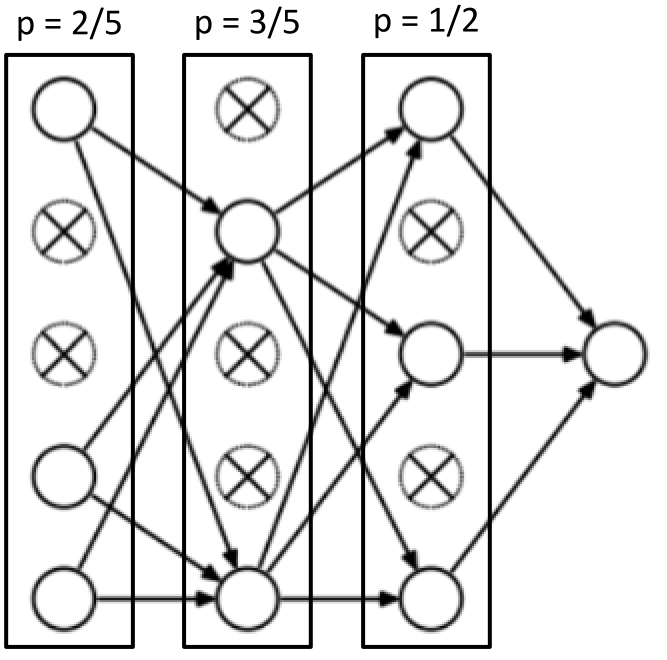
L2

Dropout

- How does it work?



Probability for hidden layers: p = 0.5
Probability for input layer: 0.5 < p < 1.0



https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/
https://wandb.ai/authors/ayusht/reports/Dropout-in-PyTorch-An-Example--VmlldzoxNTgwOTE
Srivastava, N. et al. (2014). Dropout: a simple way to prevent neural networks from overfitting.

Dropout

- Why does it work?

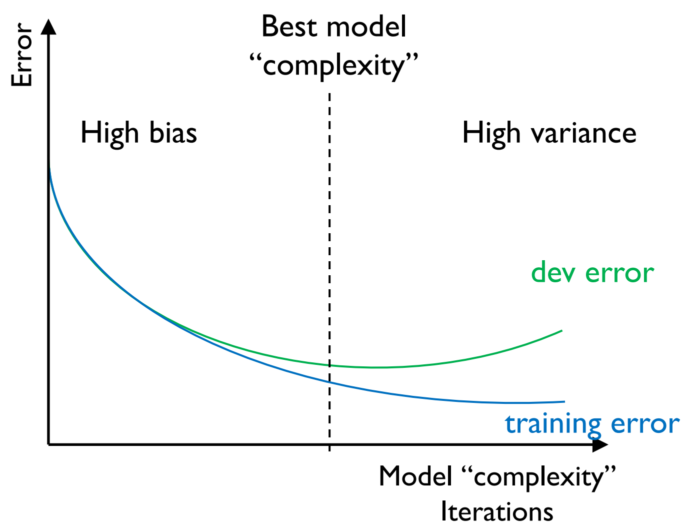
The nodes cannot rely on any single previous node (feature).

- Prevents too large weights.
- Encourages spreading out the weights.
- Forces the nodes to be more generally useful.

Can be combined with other forms of regularisation.

Early stopping

- Simple and popular regularisation technic.



- Learn enough, but not too much!
  - Avoid to end up in the high variance zone.

Early stopping

- When to stop?

1. Monitor the performance
   - Loss on the dev dataset.
   - Additional metrics (e.g., precision, recall, etc).

2. Trigger the early stopping
   - Simplest trigger: increase of the loss compared to the last iterations.
   - More elaborated ones: no change over several epochs, absolute change in a metric, average change in a metric over several epochs, reaching a specific level of performance, etc.

3. Choose the model to keep
   - Usually, keep the model from the epoch before the increase in loss.

[Prechelt L. (1998) Early Stopping - But When?]

Regularisation

- Data augmentation

- Overfitting can happen if you do not have enough data to train all parameters.

- Pre-processing technic → does not modify explicitly the learning algorithm.
  - Increases the training data set size.
  - Increases the diversity of the data.
  - Especially used with images.
  - Includes operations like rotating the image, flipping, scaling, adding noise, etc.

⚠ Can lead to underfitting if generated data not relevant to the task.

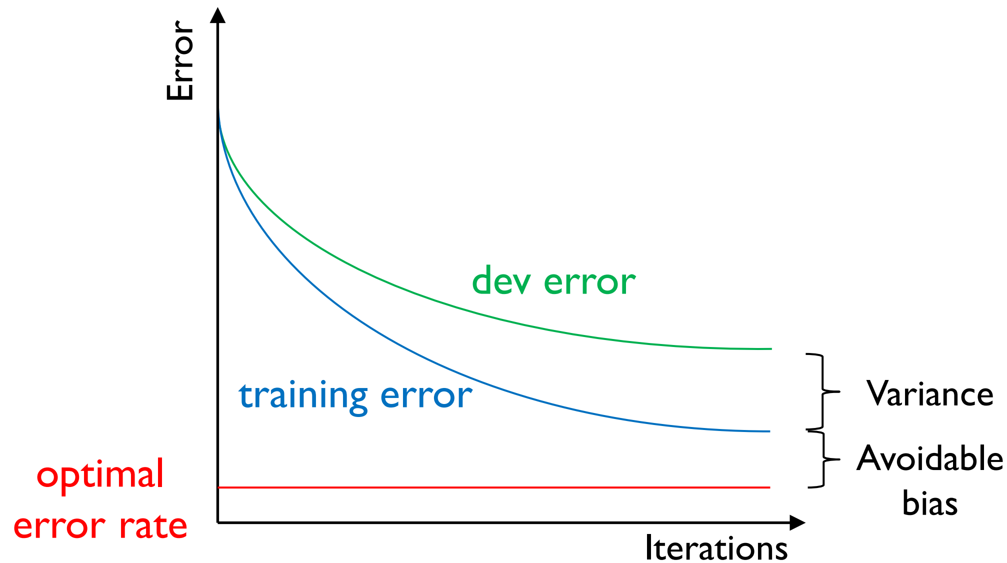
[Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning.]

Optimal error rate / avoidable bias

- Optimal error rate
  - Error rate of an optimal classifier (e.g., human performance)
  - Can be hard to estimate

- Avoidable bias
  - Training error – optimal error rate

- Variance
  - Dev error – training error



Andrew Ng, "Machine Learning Yearning", Chap. 22.

Simplest formula to address variance/bias issues

- High avoidable bias
  - Intuition: model not complex enough to map inputs and outputs.
  - Simple fix: Increase model size (e.g., increase layers or neurons per layer).
  - Might increase variance and risk of overfitting (if no regularisation).
  - Will slow the learning.

- High variance
  - Intuition: training data not sufficient to generalise on dev data.
  - Simple fix: Add data to the training set.
  - More data might not be available.
  - Try data augmentation.

Andrew Ng, "Machine Learning Yearning", Section 23.

Bias vs variance tradeoff

- Some choices reduce bias but increase variance.
  - E.g., increasing size of the network.

- Some choices reduce variance but increase bias.
  - E.g., adding regularisation (early stopping might stop learning before reaching low bias, penalizing high weights might prevent the model to reach low bias, etc).
  - Effect of regularisation on bias can be reduced with a good hyperparameter tuning.
  - Data augmentation does not increase bias if relevant augmentation.

- More useful advice in Sections 25 to 27 of Andrew Ng's "Machine Learning Yearning" book, to reduce variance and bias.

Andrew Ng, "Machine Learning Yearning", Sections 24 to 27.

Design issues: Initialisation

Initialise the weights and biases in the network

- Random: E.g., weights are initialized randomly from Uniform[−0.1, 0.1]. Biases are initialized to 0s.
- Zero: All weights are initialized to 0.

- With deep networks, always initialise weight randomly (e.g. standard normal distribution) to break the « symmetry ».

- Additional tip: Also good to normalize inputs to mean zero and use random weight initialization with avg. weight centered at zero.

Design issues: learning rate

Gradient descent is an optimization algorithm that finds the local minimum of a function by taking "steps" in the direction of the negative of the gradient.

- What will happen if we use a learning rate that is too small or too large?
- Learning efficiency, optimization accuracy
- E.g., learning steps that were taken to find the local minimum of a function

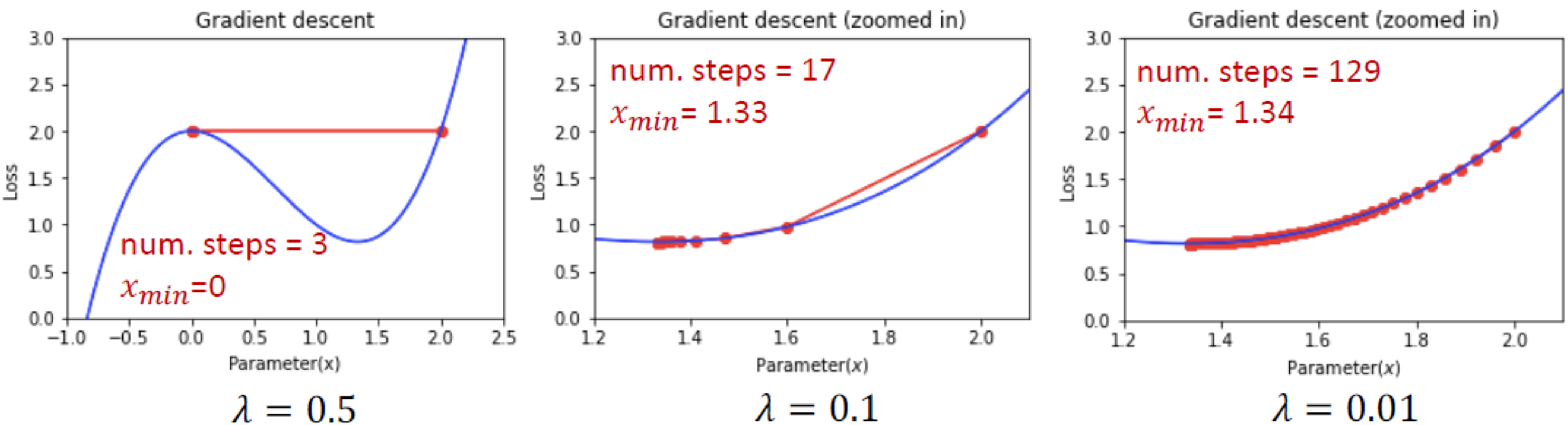


Image source: Meng-Fen Chiang

Design issues: learning rate

- **Learning rate decay schedule**
  - Common practice: decrease learning rate over time (learning rate decay).
  - E.g., linear decay.
  - Linear decay for set number of iterations and then constant.

- **Adaptive learning rates strategies**
  - Monitors the model’s performance and adapt the learning rate in response.
  - Reduces learning rate when performance plateaus.
  - Increases learning rate when performance does not improve for a number of iterations.

Design issues :Exploding and vanishing gradients

- Gradients are calculated in the backpropagation process to update the weights in the desired direction.

- **Vanishing gradients:**
  - Gradients become smaller and smaller and can become close to 0.
  - Can slow down or stop the learning process (very small weights' update).

$out\_z_j = g(\Sigma_i\ w_{i,j}\ x_i + b_j)$

$out\_z_j' = g'(\Sigma_i\ w_{i,j}\ x_i + b_j) * x_i$ (chain rule)

If g'() is close to 0, then the value of the gradient becomes smaller and smaller as backpropagation processes back to the initial layers (significant for large NN).

Design issues :Exploding and vanishing gradients

- Gradients are calculated in the backpropagation process to update the weights in the desired direction.

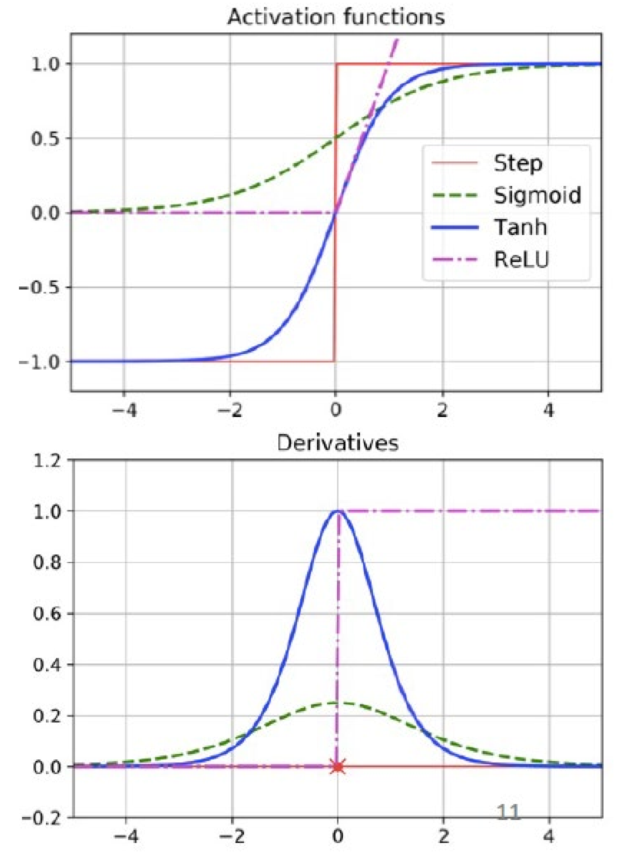
- **Vanishing gradients:**
  - Gradients become smaller and smaller and can become close to 0.
  - Can slow down or stop the learning process (very small weights' update).

- **Exploding gradients:**
  - Gradients become larger and larger as backpropagation progresses.
  - Learning can become unstable (large weights update) and diverge.

Design issues : Vanishing gradients and activation functions

As the number of layers goes up, the gradient is more likely to vanish during backpropagation.

- Using the ReLU activation function instead of tanh or sigmoid units can reduce this problem since its gradient does not go to zero as the input goes to zero.
- The Sigmoid and Tanh functions saturates at 0 or 1 when inputs become small or large.



Source: https://towardsdatascience.com/why-rectified-linear-unit-relu-in-deep-learning-and-the-bestpractice-to-use-it-with-tensorflow-e9880933b7ef

Existing activation functions

Summary of existing activation functions:

https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

Jupyter Notebook

Neural Network Design Issues
Coding Example

Advantages v.s. Disadvantages

Disadvantages

- Long training time
- Require to empirically determine, e.g., the network topology or "structure."
- Difficult to interpret the symbolic meaning behind the learnable weights and hidden units in the network

Advantages

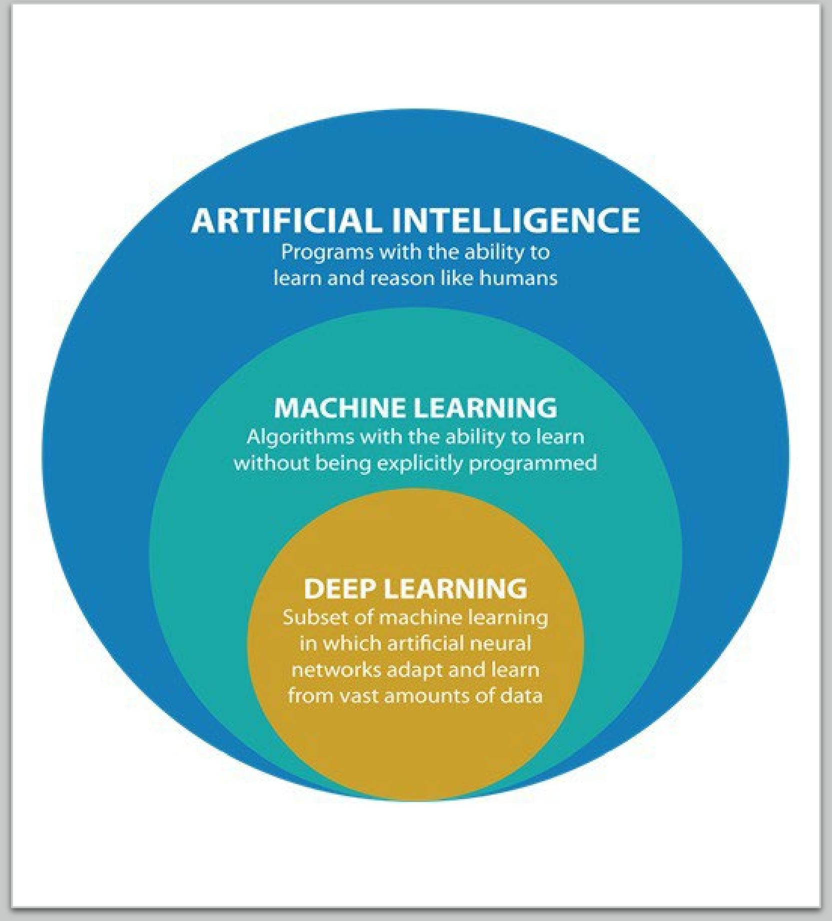
- High tolerance to noisy data
- Widely and empirically successful on real- world data, e.g., hand-written letters
- Algorithms are inherently parallel
- Techniques have recently been developed for the extraction of rules from trained neural networks
- Deep neural networks are powerful

Sebastian Raschka, Vahid Mirjalili.
Python Machine Learning

SUMMARY

- Neural Nets
  - Multilayer Perceptron Architecture
  - Nonlinear Activation Functions
  - Training: Backpropagation algorithm

- Design Issues
  - Evaluation
  - Regularization techniques
  - Learning Rate
  - Initialization
  - Vanishing Gradient Problem
  - Tips …



Resources

- Coding Libraries/Practice
  - Python Machine Learning (3rd Edition) by Sebastian Raschka at https://github.com/rasbt/python- machine-learning-book-3rd-edition
  - https://playground.tensorflow.org/

- Book Chapters
  - Chapter 6.7, 6.8 Introduction to Data Mining by Kumar et al.
  - https://www.deeplearningbook.org/ Part II Deep Networks, chap. 8 and 11.

- Others
  - https://github.com/daiwk/ml-yearning/blob/master/Ng-MLY01-13.pdf
  - http://karpathy.github.io/2019/04/25/recipe/