CO70050 Introduction to Machine Learning

Lectured by Josiah Wang

Typed by Aris Zhu Yi Qing

November 8, 2021

Contents

1 Definitions

1. **Artificial Intelligence**: Techniques that enable computers to mimic human behaviour and intelligence. It could be using logic, if-then rules, machine learning, etc.
2. **Machine Learning(ML)**: Subset of AI techniques using statistical methods that enable the systems to learn and improve with experience.
   - More data means more accurate predictions.
   - Select/Extract good features for predictions. more feature $\not\Rightarrow$ better prediction (curse of dimensionality: increased computational complexity, data sparsity, overfitting)
   - Pipeline: feature encoding, ML algorithm, and evaluation.
3. **Deep Learning**: Subset of machine learning techniques using multilayer Artificial Neural Networks(ANN) and vast amounts of data for learning.
4. **Supervised learning**: Take input variables and correct output labels as inputs, feed them into a supervised learning algorithm to generate a model which can be used to estimate labels of other input variables.
   - **Semi-supervised learning**: Some data have labels, some do not.
   - **Weakly-supervised learning**: Inexact output labels.
5. **Unsupervised learning**: Take input variables only, feed them into an unsupervised learning algorithm to generate a model which can be used to estimate labels of other input variables.
   - discover hidden/latent structure within the data ("lossy data compression")
6. **Reinforcement learning**: Largely the same as unsupervised learning, except that the estimated labels at the end "interact with an environment" and send reward signal back to the reinforcement learning algorithm such that the algorithm will take the reward signal into consideration when learning the model next time.
   - find which action an agent should take, depending on its current state, to maximise the received rewards (Policy search)
7. **Classification**: The task of approximating a mapping function from input variables to discrete output variables.
   - Binary classification: only two possible cases.
   - Multi-class classification: more than one possible class to choose from, but every input belongs to exactly one class.
   - Multi-label classification: Each input can belong to more than one class.
8. **Regression**: The task of approximating a mapping function from input variables to continuous output variables.
9. **Lazy Learner**: Stores the training examples and postpones generalising beyond these data until an explicit request is made at test time.
10. **Eager Learner**: Constructs a general, explicit description of the target function based on the provided training examples.
11. **Non-parametric model**: Assume that data distribution cannot be defined in terms of a finite set of parameters. The distribution depends on the data themselves.
12. **Underfitting/high bias**: a lot of errors, oversimplified assumptions.
13. **Overfitting/high variance**: fits "perfectly" the training data, and may not fit the test data well.

2 Classification

2.1 Instance-based Learning

2.1.1 k Nearest Neighbours (k-NN) classifier

1. Non-parametric model
2. Lazy learner
3. Procedure: Obtain k nearest data, classify the instance under the class which the most number of neighbours belong to.

4. k is usually an odd number.
5. Increasing k will make the classifier
   - have a smoother decision boundary (higher bias)
   - less sensitive to training data (lower variance)
6. Various distance metrics, such as:
   - Manhattan distance (L^1-norm):
   $$d(x^{(i)}, x^{(q)}) = \sum_{k=1}^{K} \left| x_k^{(i)} - x_k^{(q)} \right|,$$
   - Euclidean distance (L^2-norm):
   $$d(x^{(i)}, x^{(q)}) = \sqrt{\sum_{k=1}^{K} \left(x_k^{(i)} - x_k^{(q)} \right)^2},$$
   - Chebyshev distance (L^∞-norm):
   $$d(x^{(i)}, x^{(q)}) = \max_{k=1}^{K} \left| x_k^{(i)} - x_k^{(q)} \right|.$$
7. the curse of dimensionality
   - k-NN relies on distance metrics, which may not work well if using all features in high dimensional space.
   - If many features are irrelevant, instances belonging to the same class may be far from each other.
   - Solution: Weight each feature differently, or perform feature selection/extraction.

2.1.2 Distance-weighted k-NN

1. Any measure favouring the votes of nearby neighbours will work, such as:
   - Inverse of distance: $w^{(i)} = \frac{1}{d(x^{(i)}, x^{(q)})}$,
   - Gaussian distribution: $w^{(i)} = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{d(x^{(i)}, x^{(q)})^2}{2} \right)$.
2. k is not so important in distance-weighted k-NN. Distant data will have small weights and won't greatly affect classification.
3. if $k = N$ (size of the training set): global method. Otherwise, it is a local method.
4. Robust to noisy training data — the impact of isolated noise is smoothened out.
5. simple yet powerful, but might be slow for large datasets

2.1.3 k-NN regression (quick intro)

1. compute the (weighted) mean value across k nearest neighbours.

2.2 Decision Trees

2.2.1 Intuitions and Introduction

1. Eager learner
2. Decision Tree learning (or construction/induction) is a method for approximating discrete classification functions by means of a tree-based representation.
3. A decision tree can be represented as a set of if-then rules.
4. Decision Tree learning algo employ top-down greedy search through the space of possible solutions.
5. General Algorithm:
   (a) Search for an 'optimal' splitting rule on training data.
   (b) Split your dataset according to the chosen splitting rule
   (c) Repeat 5a and 5b on each new splitted subset unless the subset contains data of only one class.

2.2.2 Selecting the optimial splitting rule

1. several metrics:
   - Information gain: Used in ID3, C4.5
     - quantifies the reduction of information entropy
   - Gini impurity: Used in CART
   - Variance reduction: Used in CART
2. information entropy:
   - **Information Content** I is a quantity derived from the probability of an event occurring from a random variable X as
     $$I(x) = \log_2(\frac{1}{P(x)}) = -\log_2(P(x)),$$
     which could be interpreted as the amount of information required to fully determine the state of a random variable, and the definition should satisfy several conditions as specified here.
   - **Information Entropy** H of a discrete random variable X with its p.m.f. being $P(X)$ is defined as the expected amount of information content as following:
     $$H(X) = E[I(X)] = -\sum_{k=1}^{n} P(x_k)\log_2(P(x_k)),$$
     where n is the number of possible discrete values of X. For a p.d.f. $f(X)$, we can define the **continuous entropy** as
     $$H(X) = -\int_x f(x)\log_2(f(x))\mathrm{d}x.$$
     - The analogy in continuous case is imperfect (it can have negative values) but is still often used in deep learning.
     - The p.d.f. is often unknown, but can be approximated with density estimation algorithms for instance.
3. Use Information Entropy to select the 'optimal' split rule.
   - **Information Gain (IG)** is the difference between the initial entropy and the (weighted) average entropy of the produced subsets.
   - Mathematically,
     $$\mathrm{IG}(D,S) = H(D) - \sum_{s\in S}\frac{|s|}{|D|}H(s),$$
     where D stands for the dataset, S stands for the subsets after splitting, and $|\cdot|$ is the cardinality operator.

2.2.3 Comments

1. prevent overfitting
   - Early stopping, e.g. max depth, min examples.
   - Pruning
     (i) Go through nodes which are connected only to leaf nodes.
     (ii) Turn each into a leaf node (with majority class label).
     (iii) Evaluate pruned tree on validation set. Prune if accuracy higher than unpruned.
     (iv) Repeat until all such nodes have been tested.
2. Random Forests
   - Many decision trees voting on the class label.
   - Each tree generated with random samples of training set (bagging) and random subset of features.
3. Regression Trees
   - Instead of a class label, each leaf node now predicts an $x \in \mathbb{R}$.
   - Use a different metric for splitting, e.g. variance reduction.

3 ML Evaluation

3.1 Common Pipeline

1. shuffle the initial dataset to get rid of potential implicit ordering

2. split shuffled dataset to larger training dataset and smaller test dataset
3. use training dataset to train the model
4. feed test dataset into the trained model to evaluate the model's performance

3.2 Hyperparameter Tuning

- Hyperparameter is defined as the model parameter that are chosen before the training, such as the k of the k-NN algorithm.
- Overall objective is to find the hyperparameter values that lead to the best performance
- The correct approach is to split the dataset into 3: training/validation/test, with common splits between 6:2:2 and 8:1:1.
- It's possible to merge validation set into training set to train the model again with more data after identifying the best hyperparameter using the validation set. This can improve performance.

3.3 Corss Validation

1. used when dataset is small
2. divide dataset into k (usually 10) equal folds/splits: use $k-1$ folds for training and 1 for testing
3. iterate k times, each time test on different portion of data
4. performance on all k held-out test sets can be average:

$$\text{Global error estimate} = \frac{1}{N}\sum_{i=1}^{N} e_i$$

5. Caution: The estimate is for *algorithm* accuracy, not *model* accuracy.
6. While tuning hyperparameter, doing the same thing as with large datasets at each iteration.

3.4 Evaluation Metrics

- Confusion Matrix:

$$\begin{pmatrix} \text{True Positive} & \text{False Negative} \\ \text{False Positive} & \text{True Negative} \end{pmatrix} \quad \text{OR} \quad \begin{pmatrix} \text{TP} & \text{FN} \\ \text{FP} & \text{TN} \end{pmatrix}$$

$$\text{OR} \begin{pmatrix} \text{TP} & \text{FN} & \text{FN} & \text{FN} \\ \text{FP} & \text{TN} & ? & ? \\ \text{FP} & ? & \text{TN} & ? \\ \text{FP} & ? & ? & \text{TN} \end{pmatrix} \text{ for Class 1 in multiple classes}$$

- Metrics:

$$\text{Accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FN}+\text{FP}}$$

$$\text{Classification error} = 1 - \text{accuracy}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}$$

$$\text{Macro-averaged } \langle\text{metric}\rangle = \frac{1}{N}\sum_{i=1}^{N} \langle\text{metric}\rangle_i$$

$$\text{Micro-averaged precision} = \frac{\sum_{i=1}^{N} \text{TP}_i}{\sum_{i_1}^{N} \text{TP}_i + \text{FP}_i}, \quad \text{same for other metrics}$$

$$F_1 = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

$$F_\beta = (1+\beta^2) * \frac{\text{precision} * \text{recall}}{(\beta^2 * \text{precision}) + \text{recall}}$$

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N} \left(Y_i - \hat{Y}_i\right)^2$$

$$\text{Root MSE} = \sqrt{\text{MSE}}$$

- Precision v.s. Recall:
  - high recall, low precision: Most of the positive examples are identified correctly (low FN) but there are many FPs.

  - high precision, low recall: We miss a lot of positive examples (high FN) but those we predict as positive are really positive (low FP).

3.5 Imbalanced Data Distribution

Problem

- Imbalanced dataset: classes are not equally represented
- accuracy is misleading, affected a lot by the majority class
- Macro-averaged recall can help detect if a class is misclassified, but no info about FP. Similarly for precision.
- we should look at several matrices, along with the confusion matrix

Solution

- normalized matrix: divide each row by the total number of samples in that class
- downsample the majority class/upsample the minority class
- Look at differet metrics and choose ones that reflect the intended behaviour of the model.

3.6 Overfitting

- Overfitting: good performance on training data, poor generalization to other data, can occur when:
  - the model we use is too complex (use right level of complexity)
  - the examples in the training set are not representative of all possible situations (get more data)
  - learning is performed for too long (stop training earlier)
- underfitting: poor performance on the training data and poor generalization to other data

3.7 Confidence Intervals

- True error of model h is the probability that it misclassifies a randomly drawn example x from distribution D:

$$\epsilon_D(h) := P[f(x) \neq h(x)].$$

- Sample error (classification error) of the model h based on a data sample S:

$$\epsilon_S(h) := \frac{1}{N} \sum_{x \in S} \delta(f(x), h(x))$$

- Confidence interval is defined as, when given a sample S with the number of test samples $n \geq 30$, we can say that with $N\%$ confidence, the true error lies in the interval:

$$\epsilon_S(h) \pm Z_N \sqrt{\frac{\epsilon_S(h) * (1 - \epsilon_S(h))}{n}}$$

where

$$\Phi(S' \leq |Z_N|) = N\%$$

and S' is the normalized sample of S.

3.8 Testing for Statistical Significance

- The statistical tests tell us if there is indeed a difference between the two distributions, such as Randomisation test, T-test, Wilcoxon rank-sum test, etc.
- The null hypothesis H_0 states that the two algorithms/models perform the same and the performance differences are only due to sampling error.
- The statistical tests return a p-value: the probability of obtaining observed performance differences (or more), assuming that H_0 is correct.
- small p-value means we can be more confident that one system is actually different from another.
- performance difference is statistically significant if $p < 0.05$.

- $p > 0.05$ *does not mean* that the two algorithms are similar. E.g. collecting more data can change the p-value in certain direction.
- P-hacking is the misuse of data analysis to find patterns in data that can be presented as statistically significant when in fact there is no underlying effect.
  - This dramatically increases and understates the risk of FPs, i.e. reject the H_0 where in fact we shouldn't, i.e. we thought there is difference in distribution, where in fact there isn't.
  - Fight against P-hacking: adaptive p-value
    1. rank p-values from M experiments:
    $$p_1 \leq p_2 \leq p_3 \leq \ldots \leq p_M$$
    2. Calculate the Benjamini-Hochberg critical value for each experiment:
    $$z_i = 0.05\frac{i}{M}$$
    3. significant results are the ones where the p-value is smaller than the critical value

4 Artificial Neural Networks (ANN)

- ANN: a class of ML algorithms. Architectures of connected neurons, usually optimized with gradient descent
- Deep learning: using ANN with multiple hidden layers. Complex models trained with large datasets.

4.1 Linear Regression

- Supervised learning
- the desired labels are continuous (instead of discrete)
- represented as $y = ax + b$
- Loss function:
  - "sum-of-squares", $E = \frac{1}{2}\sum_{i=1}^{N}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$.
  - smaller values of E means more accurate predictions

4.2 Neurons

- Given input X, parameter θ, and **activation function** $g(z)$, we can model the **artificial neuron** as
$$\hat{y} = g(X\theta).$$
- logistic activation function(logistic function):
$$g(z) = \frac{1}{1+e^{-z}}.$$
- Perceptron: uses the threshold function as the activation function
$$h(x) = f(W^T x) = \begin{cases} 1 & \text{if sum}(W^T x) > 0 \\ 0 & \text{otherwise,} \end{cases}$$
with the learning rule
$$\theta_i := \theta_i + \alpha(y - h(x))x_i,$$
where y is the desired output and $h(x)$ is the prediction.
- connecting multiple neurons in a manner of multiple layers creates a multi-layer network.

4.3 Activation functions

- Sigmoid activation:
$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}.$$
- Tanh activation:
$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$
- ReLU activation:
$$f(x) = \text{ReLU}(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x \geq 0. \end{cases}$$

- Softmax activation:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_k e^{z_k}}.$$

- Most activation functions are applied element-wise. Softmax is an exception.
- ReLU is commonly used for very deep networks. Tanh and sigmoid also work well and can be more robust.
- activation of output layer should depends on the task.
  - classifying into two classes → sigmoid or tanh
  - predicting an unbounded score → linear
  - predicting a probability distribution → softmax

4.4 Loss function

- lower loss means better task performance
- MSE (quadratic loss, or L2 loss):

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

- maximize the likelihood of the network assigning the correct labels to all inputs in our dataset:

$$\prod_{i=1}^{N} p\left(y^{(i)} \mid x^{(i)}; \theta\right),$$

  assuming examples are i.i.d. s.t. $p(A \wedge B) = p(A)p(B)$.
- For binary classification, the probability of success output given input:

$$\prod_{i=1}^{N} \left(\hat{y}^{(i)}\right)^{y^{(i)}} \left(1 - \hat{y}^{(i)}\right)^{1-y^{(i)}}.$$

Maximizing the above expression is equivalent to maximizing

$$\sum_{i=1}^{N} y^{(i)} \log\left(\hat{y}^{(i)}\right) + (1 - \hat{y}^{(i)}) \log\left(1 - \hat{y}^{(i)}\right).$$

Turning this into a loss we can minimise the **binary cross-entropy**:

$$L = -\frac{1}{N}\sum_{i=1}^{N} y^{(i)} \log\left(\hat{y}^{(i)}\right) + (1 - \hat{y}^{(i)}) \log\left(1 - \hat{y}^{(i)}\right)$$

which can be generalized to **categorical cross-entropy** for multi-class classification:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_c^{(i)} \log\left(\hat{y}_c^{(i)}\right),$$

where C is the set of possible classes and $\hat{y}_c^{(i)}$ is the predicted probability of class c for datapoint i.

4.5 Backward Propagation

Assume that there are

- N samples,
- M output nodes,
- $D^{(i)}$ nodes at layer i, and let the first layer be at index 0,
- $t + 1$ number of layers in total.

and let

- $L \in \mathbb{R}^{N \times M}$ be the loss of the ANN,
- $W^{(i)} \in \mathbb{R}^{D^{(i-1)} \times D^{(i)}}$ be the weights at layer i,
- $X^{(i)} \in \mathbb{R}^{N \times D^{(i)}}$ be the activated node values at layer i.
- $Z^{(i)}$ be the linear-combination values at layer i,
- $g(Z)$ be the activation function of Z,

- $X := X^{(0)}$ to be the input values
- $\hat{Y} := X^{(t)}$ to be the output values.

Thus

$$\begin{aligned}\Delta^{(i)} := \frac{\partial L}{\partial W^{(i)}} &= \frac{\partial Z^{(i)}}{\partial W^{(i)}}\frac{\partial L}{\partial Z^{(i)}} \\ &= \frac{\partial\left(X^{(i-1)}W^{(i)}\right)}{\partial W^{(i)}}\delta^{(i)} \\ &= X^{(i-1)}\delta^{(i)},\end{aligned}$$

where for $0 < i < t$,

$$\begin{aligned}\delta^{(i)} := \frac{\partial L}{\partial Z^{(i)}} &= \frac{\partial Z^{(i+1)}}{\partial Z^{(i)}}\frac{\partial L}{\partial Z^{(i+1)}} \\ &= \frac{\partial\left(W^{(i+1)} * g(Z^{(i)})\right)}{\partial Z^{(i)}}\delta^{(i+1)} \\ &= \delta^{(i+1)}W^{(i+1)}g'(Z^{(i)}),\end{aligned}$$

and for $i = t$,

$$\begin{aligned}\delta^{(t)} &= \frac{\partial L}{\partial Z^{(t)}} \\ &= \frac{\partial \hat{Y}}{\partial Z^{(t)}}\frac{\partial L}{\partial \hat{Y}} \\ &= \frac{\partial g(Z^{(t)})}{\partial Z^{(t)}}\frac{\partial L}{\partial \hat{Y}} \\ &= g'(Z^{(t)})\frac{\partial L}{\partial \hat{Y}},\end{aligned}$$

where the value of $\frac{\partial L}{\partial \hat{Y}}$ depends on the chosen loss function, and the value of $g'(Z^{(i)})$ depends on the chosen activation function.

4.6 Gradient Descent

- Definition: Repeatedly update a parameter W by taking small steps in the negative direction of the partial derivative, as follow

$$W := W - \alpha\frac{\partial L}{\partial W}$$

where α is a hyperparameter for the learning rate/step size.

- vector notation: for a function $f : \mathbb{R}^K \mapsto \mathbb{R}$, the gradient is

$$\nabla_\theta f(\theta) = \begin{pmatrix}\frac{\partial f(\theta)}{\partial \theta_1} \\ \frac{\partial f(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\theta)}{\partial \theta_K}\end{pmatrix}.$$

In the case of $L(W)$ being MSE, we can find the value of W such that the value of $L(W)$ is minimized by setting partial derivative to 0. We thus have

$$\nabla_W E(W) = X^T(XW - y) = 0 \quad \Longrightarrow \quad W^* = \left(X^T X\right)^{-1}X^T y,$$

where

$$X = \begin{pmatrix}x_1^{(1)} & \dots & x_K^{(1)} & 1.0 \\ x_1^{(2)} & \dots & x_K^{(2)} & 1.0 \\ \vdots & \ddots & \vdots & \vdots \\ x_1^{(N)} & \dots & x_K^{(N)} & 1.0\end{pmatrix}, \qquad y = \begin{pmatrix}y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)}\end{pmatrix}, \qquad \theta = \begin{pmatrix}a_1 \\ \vdots \\ a_K \\ b\end{pmatrix}.$$

- Comparisons among various gradient descent methods:
  - gradient descent: Compute gradient based on the whole dataset.
  - stochastic gradient descent: Loop over each datapoint, compute gradient based on the data point and update weights. This can be very noisy.
  - mini-batched gradient descent: Loop over batches of datapoints, compute gradient based on the batch and update weights. This is what we mostly use in practice.
- Adaptive learning rates: Take smaller steps as we get closer to the minimum. Strategies for updating the learning rate include:
  - every epoch

  - after a certain number of epochs
  - when performance on the validation set hasn't improved for several epochs.

- Weight initialization:
  - zeros
  - normal: draw randomly from a normal distribution $\sim N(0,1)$ or $N(0,0.1)$
  - Xavier Glorot:

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}\right],$$

    where U is the uniform distribution, n_j is the number of neurons in the previous layer, n_{j+1} is the number of neurons in the next layer.
  - Random initialization lead to different results, sometimes even with the same seed! So embrace randomness, run with different random seeds and report the average.
    * GPU threads finish in a random order, leading to randomness.
    * Small rounding errors add up.

- Data normalization:
  - min-max normalization: scaling the smallest value to a and largest value to b,

$$X' = a + \frac{X - X_{\min}}{X_{\max} - X_{\min}}(b-a)$$

  - standardization (z-normalization): scaling the data to follow standard normal distribution,

$$X' = \frac{X-\mu}{\sigma}$$

  - Normalization helps because $\Delta^{(i)} \propto X^{(i)}$,

$$\Delta^{(i)} = \frac{\partial L}{\partial W^{(i)}} = X^{(i-1)}\frac{\partial L}{\partial Z^{(i)}}.$$

  - the scaling parameters, like a and b in min-max normalization, need to be calculated based on the *training set only.*

- Gradient checking: By calculating

$$W_t^{(i)} = W_{t-1}^{(i)} - \alpha\frac{\partial L(W^{(i)})}{\partial W^{(i)}} \quad \Longrightarrow \quad \frac{\partial L(W^{(i)})}{\partial W^{(i)}} = \frac{W_{t-1}^{(i)} - W_t^{(i)}}{\alpha},$$

  where t indicates the number of iterations, and

$$\frac{\partial L(W^{(i)})}{W^{(i)}} \approx \frac{L(W^{(i)}+\epsilon) - L(W^{(i)}-\epsilon)}{2\epsilon},$$

  where $\epsilon > 0$ is very small, both values should be pretty similar.

4.7 Overfitting in ANN

- Very important to use held-out validation and test sets.
- If ANN is underfitting on the training data, we can try increasing the number of neurons/layers.
- If ANN is overfitting on the training data too fast, we can try reducing its capacity by lowering the number of neurons/layers.
- To tackle the problem of overfitting:
  - early stopping: We can use the validation data to choose when to stop (before the model keeps improving until it overfits).
  - regularization: Adding some info/constraints to stop the model from overfitting.
    * L2 regularization:

$$L_2 = L + \lambda\sum_w w^2 \quad \Longrightarrow \quad W := W - \alpha\left(\frac{\partial L}{\partial W} + 2\lambda w\right).$$

    * L1 regularization:

$$L_1 = L + \lambda\sum_w |w| \implies W := W - \alpha\left(\frac{\partial L}{\partial W} + \lambda\,\mathrm{sign}(w)\right).$$

  - dropout: During training, randomly set some (typically 50%) neural activations to 0. During testing, use all the neurons, but scale the activations. This prevents the ANN from relying on any one node.