Lecture 6. Artificial Neural Networks

From Perceptron to MLP

Zhonghai Lu

KTH Royal Institute of Technology

April 15, 2024

Agenda

What is ANN?

- The inventor of the first neurocomputer, Dr. Robert Hecht-Nielsen, defines a neural network as ”...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.”
- The idea of ANNs is based on the belief that working principle of human brain by making the right connections, can be imitated using silicon and wires as living neurons and dendrites.

ANN and brain

- An ANN is based on a collection of connected units or nodes called *artificial neurons* which loosely model the neurons in a biological brain.
- Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another.
- An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it.

From biological to artificial neuron

An artificial neuron mimics the working of a biophysical neuron, but is not a biological neuron model.

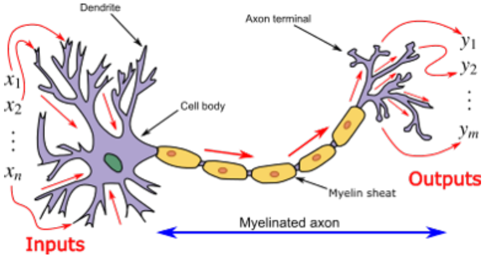


Figure: A biological neuron

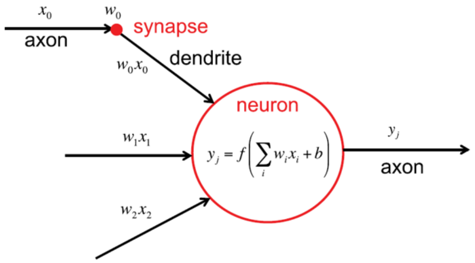


Figure: An artificial neuron

Source: Wikipedia

Source: Sze et al. Proc. of the IEEE, 105(12), Dec. 2017

Perceptron (1962)

Perceptron is a mathematical or conceptual model of a biological neuron.

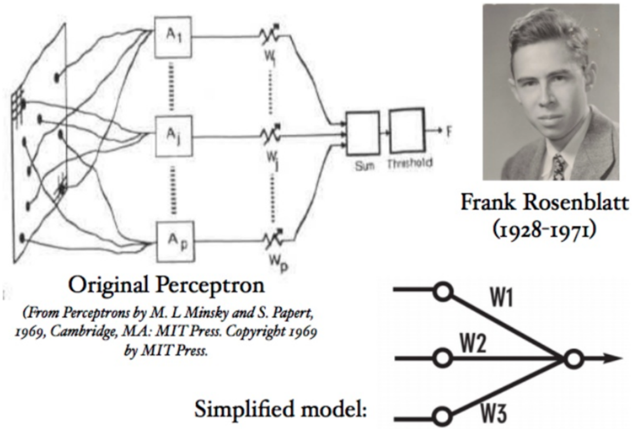

A1
Aj
Ap
W1
Wj
Wp
Sum
Threshold
F
Frank Rosenblatt
(1928-1971)
Original Perceptron
(From Perceptrons by M. L Minsky and S. Papert, 1969, Cambridge, MA: MIT Press. Copyright 1969 by MIT Press.
Simplified model:
W1
W2
W3

Figure: The perceptron (Soucre: Roelof Pieters 2015)

Perceptron (1962)

A nonlinear function is essential to model arbitrarily complex functionality which is not achievable by linear combinations of any depth.

The nonlinear activation function $g(\cdot)$ was given by a step function.

$$y = g(\sum_{i=1}^{n} x_i \cdot w_i + b)$$

$$g(a) = \begin{cases} +1 & \text{if } a \geq \theta \\ \text{-}1 & \text{if } a < \theta \end{cases}$$

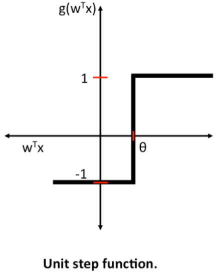


Figure: A unit step function

Why step function not in favor any more?

- Since the learning strategy requires computation of the gradient of the error function at each iteration step, we must guarantee the continuity and differentiability of the error function.
- We have to use a kind of activation function other than the step function used in perceptrons, because the composite function produced by interconnected perceptrons is discontinuous, and therefore the error function too.
- One of the more popular activation functions for backpropagation networks is the sigmoid.

Sigmoid function

- The term “Sigmoid” means S shaped. It is called a ‘squashing function’ as it maps the whole real axis into a finite interval. – Tanh() is also S shaped.
- Logistic Sigmoid function: The constant c can be selected arbitrarily and its reciprocal 1/c is called the temperature parameter in stochastic neural networks.

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

- Higher values of c bring the shape of the sigmoid closer to that of the step function and in the limit $c \to \infty$, it converges to a step function at the origin.

Sigmoid function

The derivative of $s(x)$ has the form.

$$\frac{d}{dx}s(x) = \frac{e^{-x}}{(1+e^{-x})^2} = s(x)(1-s(x))$$

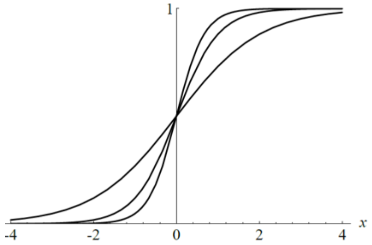
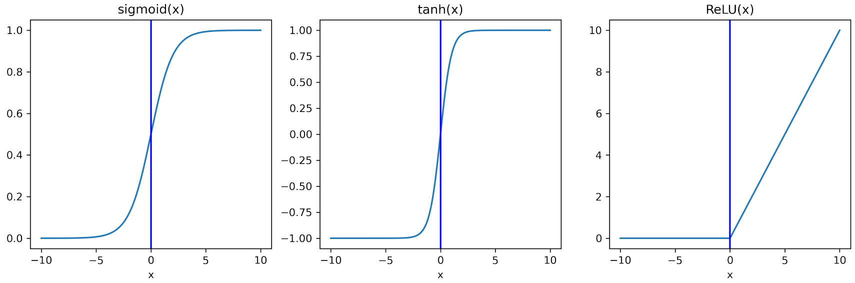


Figure: Three sigmoids, c=1, 2, 3

Common activation functions



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

$$tanh(x) = 2s(2x) - 1$$

Hyperbolic tangent.

$$f(x) = max(0, x)$$

ReLU: Rectified Linear Unit

ANN

- An artificial neuron mimics the working of a biophysical neuron, but is not a biological neuron model.
- An artificial neural network is a composition of simple elements called artificial neurons, which receive input, perform simple calculation, and produce output.

MLP

- A Multi-Layer Perceptron (MLP) contains one or more hidden layers, apart from one input and one output layer.
- The input layer has no weights associated. It should not be considered as a network layer. For convenience, many people still call it input layer, though.
- An MLP is a feed-forward network without feedback connections.

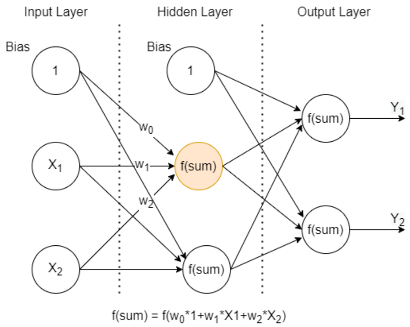


Figure: A simple MLP

MLP representations

MLP can be represented in different forms: graph, math, software.

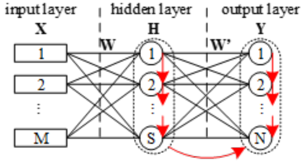
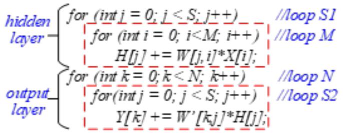


Figure: Graphical representation

$$\vec{H} = f(W_{S\times M} \times \vec{X} + \vec{b})$$
$$\vec{Y} = f(W'_{N\times S} \times \vec{H} + \vec{b'})$$

Figure: Mathematical representation


hidden layer
output layer

```
for (int j = 0; j < S; j++)              //loop S1
    for (int i = 0; i<M; i++)            //loop M
        H[j] += W[j, i]*X[i];
for (int k = 0; k < N; k++)              //loop N
    for(int j = 0; j < S; j++)           //loop S2
        Y[k] += W'[k,j]*H[j];
```

Figure: Software representation

Power of MLP

Neural networks are said to be **universal function approximators**.

- MLPs are universal function approximators as shown by Cybenko's theorem*.
- For example, a two-layer network with linear outputs can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy, provided the network has a sufficiently large number of hidden units.

*Cybenko, G. "Approximation by superpositions of a sigmoidal function". Mathematics of Control, Signals, and Systems, 2(4), 303–314, 1989.

Universal Approximator Theorem

- By the theorem, MLP with one hidden layer is enough to represent (not learn) an approximation of any function to an arbitrary degree of accuracy.
- Even if MLP is able to represent an arbitrary function, learning can fail for two reasons.
  - The optimization algorithm may fail to find the value of the parameters for the desired function.
  - The training algorithm might choose the wrong function as a result of overfitting.

So, why go deeper, to Deep Neural Networks (DNN)?

- Shallow net may need (exponentially) larger width
- Shallow net may overfit more
- On the other hand, going deeper gives more opportunities.

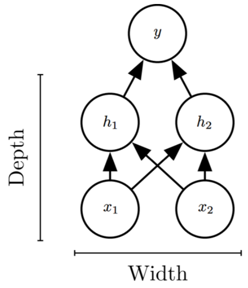


Figure: Width and depth of MLP

Training overview

- Why training? Purpose and nature
- What is error? Point error vs. Total error. Error as a function of weights
- How to calculate errors for weights in different layers? Back propagation
- How to minimize error? Stochastic Gradient Descent (SGD) and others. Local optimum vs. Global optimum
- How to select an optimization algorithm?
- How to avoid under-fitting, overfitting? under-fitting vs. over-fitting.

Neural network training and inference

- Artificial Neural Network (ANN) training is a supervised learning.
- Given a set of features $X = (X_1, X_2, \dots)$ and a target Y, an ANN can learn the relationship between the features and the target, for either classification or regression.

- Training: Given an ANN architecture, the goal of training or learning is to assign 'correct' weights to the connections between layers so that the error is minimized.
- Inference: Once the training terminates, the "learned" ANN is ready for inference or prediction, i.e., Given an input vector, these weights determine what the output vector of the ANN is.

The training process

ANN training is typically an error-based supervised learning – ”learning from mistakes”.

- Initialization: Initially all the connection weights of the ANN are randomly assigned.
    - Forward pass: For every input in the training dataset, the ANN is activated and its output is computed.
    - Backward pass: This output is compared with the desired output that we already know, and the error or loss is “propagated” back to the previous layer.
      – Error back propagation
    - Weight update: This error is noted and the weights are “adjusted” accordingly.
      – Optimization
- Iteration: This process is repeated until the output error is below a predetermined threshold or a stopping criterion reached.

The training process

The learning is iterative.

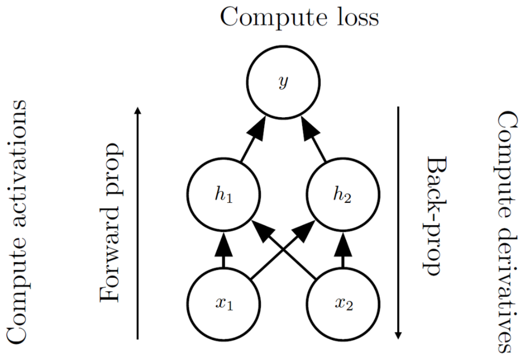


Figure: Iterative forward computation, backward error propagation, and weight tuning

Error or loss function

For regression problems, a standard error function is the sum-of-squares error function.

For each training sample n, the error is defined as

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$

where y_k is the forward calculation result, t_k is the target result.

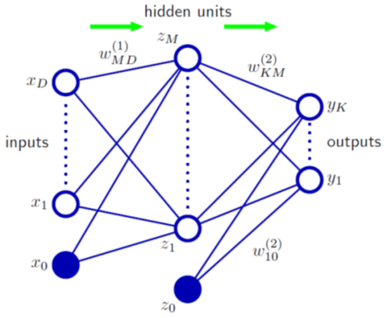


Figure: An MLP

Weight optimization

The combination of weights which minimizes the error function is considered to be a solution of the learning problem.

- Learning objective:
  Minimize the error (E) as a function of all weights w_i
- Learning strategy: Gradient descent

$$w_i = w_i - \eta \frac{\partial E}{\partial w_i}$$

where $\eta > 0$ is the learning rate.
Note that the error function is defined with respect to *a training set*, and so each step requires that the entire training set be processed in order to evaluate the error gradient.

Stochastic Gradient Descent (SGD)

- Error functions for a set of independent observations comprise a sum of terms, one for each data point, *n*.

$$E(w) = \sum_{n=1}^{N} E_n(w)$$

- Stochastic gradient descent (SGD), known as sequential or on-line gradient descent, updates the weight vector based on one data point at a time,

$$w^{t+1} = w^t - \eta \cdot E_n(w^t)$$

  This update is repeated by cycling through the data either *in sequence* or by *selecting points at random with replacement*.

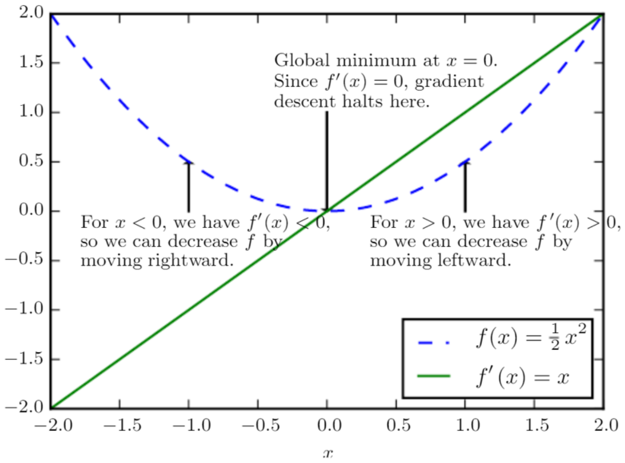
- It handles redundancy in the data much more efficiently.

Back propagation and SGD

- Error Back-propagation
  - An efficient method of computing gradients in directed graphs of computations, such as neural networks.
  - It is a simple implementation of chain rule of derivatives, which allows to compute all partial derivatives in linear time in terms of the graph size.
- SGD
  - It is an optimization method using the gradient information to update weights.
  - It is one of many known optimization techniques which use gradient information such as RMSProp (Root Mean Square Propagation) and Adam (Adaptive Moment Estimation) etc.

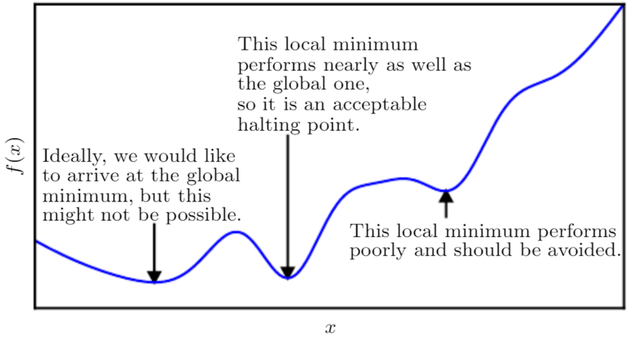
Why gradient descent?

An illustration of how the gradient descent algorithm uses the derivatives of a function to follow the function downhill to a minimum.



Approximate minimization: Local vs Global optimum

Optimization algorithms may fail to find a global minimum when there are multiple local minima or plateaus present.



In the context of deep learning, we generally accept such solutions even though they are not truly minimal.

Optimization algorithms

- Stochastic Gradient Descent (SGD)
  - Can be slow
- SGD with moment
  - Introduce a variable v as velocity (direction and speed) of moving through parameter space
  - Nesterov moment (Nesterov's accelerated gradient, NAG)
- Algorithms with adaptive learning rates
  - RMSProp (Root Mean Square Propagation)
  - RMSProp with moment
  - Adam: Adaptive moments
- Others

Which one to choose?

- Research show that, while the family of optimization algorithms with adaptive learning rates (represented by RMSprop and AdaDelta) performed fairly robust, no single best algorithm emerged.
- The choice of which optimization algorithm to use, at this point, seems to depend largely on the user's familiarity with the algorithm.

An example of polynomial curve fitting

A polynomial function

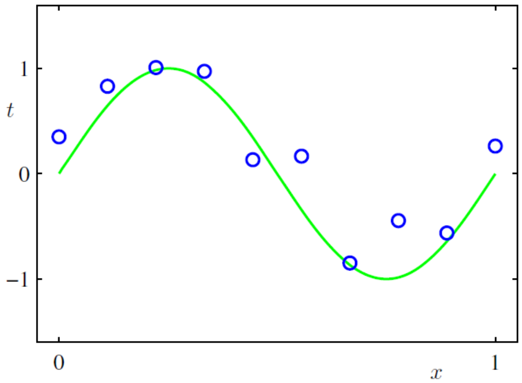
$$t = f(x) = w_0 + w_1 \cdot x + w_2 \cdot x^2 + \ldots + w_M \cdot x^M + \epsilon$$

where M is the order of the polynomial, ϵ a stochastic variable.
Choosing the order of M is the question of model selection.

- Now we try to fit the polynomial model to a *sin*() function.
- We will see how different models lead to different fitness, in particular, under-fitting and over-fitting.

An example of polynomial curve fitting

- A plot was generated by choosing values of x_n, for $n = 1, \cdots, N$, spaced uniformly in range [0, 1], and the target data set t was obtained by first computing $t = sin(2x)$ and then adding a small level of random noise having a Gaussian distribution to each such point in corresponding values of the function.
- Our goal is to predict the value of t for some new value of x, without knowledge of the green curve.



Plot of a training data set of N = 10 points, shown as blue circles, each comprising an observation of the input variable x along with the corresponding target variable t.
* The green curve shows the function sin(2x) used to generate the data.

Fitting four curves to the data set

- The constant (M = 0) and first order (M = 1) polynomials give rather poor fits to the data and consequently rather poor representations of the function sin(2x), so called **under-fitting**.

  Usually the model is too simple to capture the subtleties of data.
- The third order (M = 3) polynomial seems to give the best fit to the function sin(2x).
- When M = 9, we obtain an excellent fit to the training data. The polynomial passes exactly through each data point and Error(w) = 0. However, the fitted curve oscillates wildly and gives a very poor representation of the function sin(2x). This latter behaviour is known as **over-fitting**.

  The model memorizes the specifics of the data set, but becomes difficult to generalize to new data, so called generalization error.

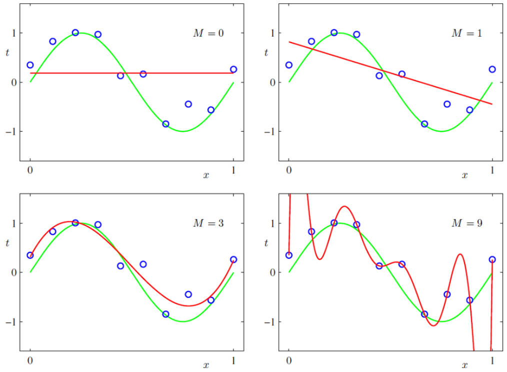
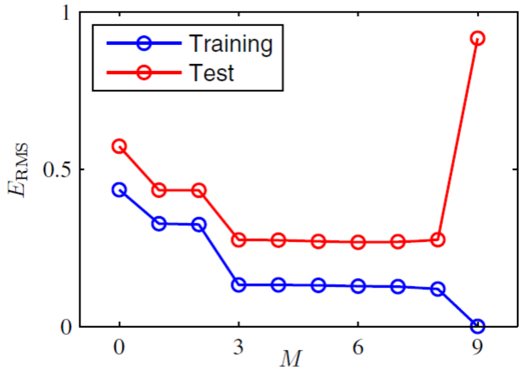


Figure: Plots of polynomials having various orders M, shown as red curves, fitted to the data set

Underfitting and overfitting

A data set is usually split into a training set and a test set.

- Underfitting: Both training set error and test set error are high.
  Usually the model is too simple to capture the subtleties of data.
- Overfitting: Training set error is low, even zero, but the test set error goes high.
  The model memorizes the specifics of the data set, but becomes difficult to generalize to new data, so called **generalization error**.

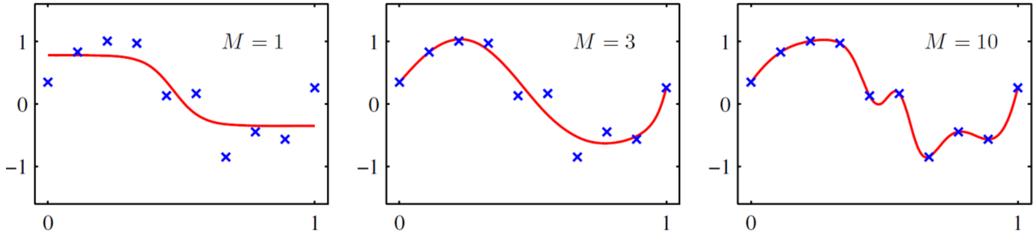


M indicates model complexity. For M = 9, the training set error goes to zero. But the test set error becomes very large.

Overfitting example

Alternatively we can fit a nueral network to the sine dataset.

- The number of input and output units in a neural network is generally determined by the dimensionality of the data set, whereas the number M of hidden units is a free parameter that can be adjusted to give the best predictive performance.



- Examples of two-layer networks trained on 10 data points drawn from the sinusoidal data set. The graphs show the result of fitting networks having M = 1, 3 and 10 hidden units, respectively, by minimizing a sum-of-squares error function.
- Note that the generalization error is not a simple function of M. Given the same M, different weights can lead to different errors.

Combat overfitting

- Data augmentation
  - Not only adding data points (including adding noises)
  - But also transform data (shift, scale, slight rotate etc.), particularly useful for image recognition.
- Regularization via parameter norm penalty (E.g. L2 weight decay, L1 sparse weights)
- Early stopping
- Drop out
- Others

Regularization

- Regularization is the idea of adding a regularization term to an error function in order to control over-fitting, so that the total error function to be minimized takes the form:

$$E(W) = E_D(W) + \lambda E_W(W)$$

  $E_D(W)$ is data-dependent error and $E_W(W)$ the regularization term. λ is the regularization coefficient.
- Regularized least squares

$$E(W) = \frac{1}{2}\sum_{n=1}^{N}(y_n - t_n)^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|W_j|^q$$

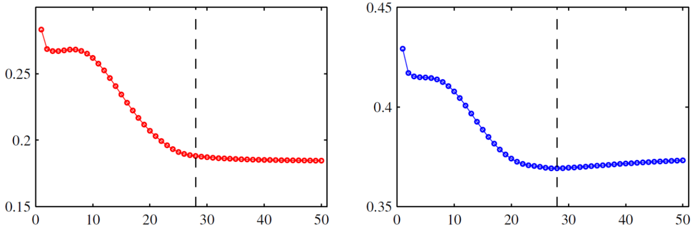
  $q = 1$: L1 regularization/L1 norm (Lasso regression), driving some weights to zero (sparse model). Lasso: Least absolute shrinkage and selection operator.
  $q = 2$: L2 regularization/L2 norm is a quadratic regularizer (Ridge regression, Weight decay).

Early stopping

- An alternative to regularization as a way of controlling the effective complexity of a network.
- The training of nonlinear network models corresponds to an iterative reduction of the error function defined with respect to a set of training data.
    - For many of the optimization algorithms for network training, the error is a non-increasing function of the iteration index.
    - However, the error measured with respect to independent data, generally called a validation set, often shows a decrease at first, followed by an increase as the network starts to over-fit.

Early stopping

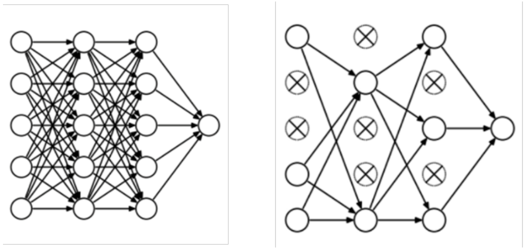
- An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step, for the sinusoidal data set.

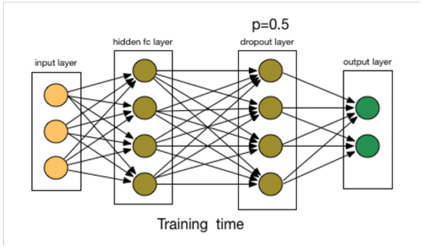


- The goal of achieving the best generalization performance suggests that training should be stopped at the point shown by the vertical dashed lines, corresponding to the minimum of the validation set error.

Drop out

Drop-out is an ensemble learning technique.





- At every iteration, it randomly selects some nodes and removes them along with all of their incoming and outgoing connections.
- So each iteration has a different set of nodes and this results in a different set of outputs.

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

Transfer learning

Learning by random initialization of weights can result in uncertainty.

- Final outcome is sensitive to the initially chosen weights.
- Training can take too long or too many epochs to converge.

Transfer learning means that the actual learning can use an already proven network layers with set of weights as the starting point for training the neural network.

- Re-use proven network layers and set of weights
- Transfer learning is very commonly used, for example, in the feature extraction process of CNNs.

Modeling for time-series prediction

- The key is to re-organize the data set to suit the need for supervised training of neural networks.
- Training set in sequence, not randomly pick up, to maintain the structure, order of the sequence.

Data re-organization

For example, we are going to predict the next two values for a sequence [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. We can re-organize the sequence as a data set suitable for supervised learning.

- The output vector has a size of 2.
- Assume that the input vector has a size of three. This is a design decision.

Then we can create a training and a test data set as follows.

Training set	input vector	output vector
	[1, 2, 3]	[4, 5]
	[2, 3, 4]	[5, 6]
	[3, 4, 5]	[6, 7]
	[4, 5, 6]	[7, 8]
	[5, 6, 7]	[8, 9]
	[6, 7, 8]	[9, 10]
Testing set	**input vector**	**output vector**
	[7, 8, 9]	[10th, 11th]
	[8, 9, 10]	[11th, 12th]

Table: Transform a series into an input-output pair for supervised learning

Example: Predicting Fibonacci series

- The Fibonacci series is a series of integer numbers (except first 0) in the following sequence.
  $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots\ldots..$
- The sequence F_n of Fibonacci numbers is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2}$$

  with seed values $F_0 = 0$ and $F_1 = 1$.
- Given some Fibonacci numbers, predict what the next number is or numbers are?

Ipython demo. `http://localhost:8888/notebooks/IL2233VT22/Lec6_ann/sine-fibonacci-sequence-prediction-mlp-lstm.ipynb`

Example: Implementation steps

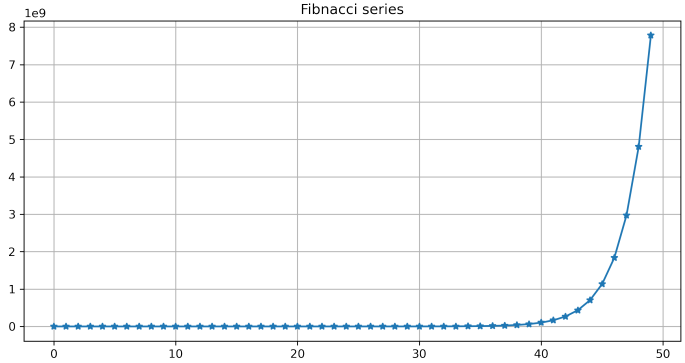
Let's generate 50 numbers in the sequence.

- Visualize the data set.
- Data pre-processing
  - Organize the data into input-output pairs for supervised learning.
  - Split the data set into a training set and a test set, say 70% : 30%.
- Define an MLP model.
  - The input vector size determines the number of neurons of the input layer.
  - The output vector size determines the number of neurons of the output layer.
  - How many hidden layers and how many neurons each hidden layer is up to the problem and your experience.
- Use the training set to train the MLP.
- Check the prediction accuracy with the test set.
- Visualize the test data and predicted results.

Example

Generate the first 50 numbers.

[0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986 102334155 165580141 267914296 433494437 701408733 1134903170 1836311903 2971215073 4807526976 7778742049]



Q: Is the series stationary? If not, can we use differencing to make it stationary?

Example: Data preprocessing and model construction

We may use 3 current values to predict the next value. Then the first few entries of the training data would be as follows. The test set uses X to represent numbers, because the numbers are too large.

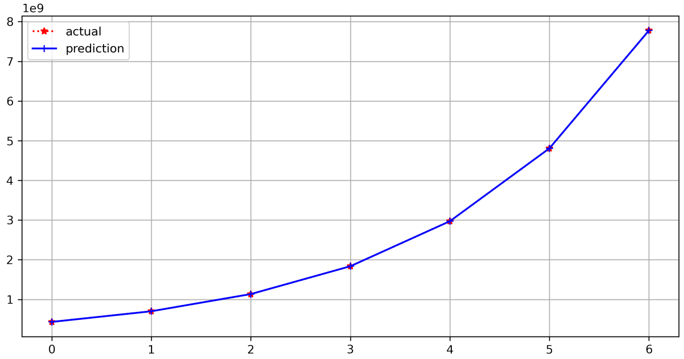
Training set	input vector	output vector
	[0, 1, 1]	[2]
	[1, 1, 2]	[3]
	[1, 2, 3]	[5]
	[2, 3, 5]	[8]
	[3, 5, 8]	[13]
	[5, 8, 13]	[21]
Testing set	**input vector**	**output vector**
	[x1, x2, x3]	[x4]
	[x2, x3, x4]	[x5]

Table: Transform a series into input-output pairs for supervised learning

From this data re-organization, we can infer that the input layer of the MLP has 3 neurons, and the output layer 1 neuron.

Example: Prediction and validation

- After training with sufficient epoches, we can report MSE, MAE, MAPE etc.
- We can also visualize the prediction results against the test set.



Summary

- ANN is a structured composition of artificial neurons.
- A nonlinear transformation following a weighted sum of input stimulus allows us to model complex non-linear behaviors.
- ANN training is an error-based learning process to tune weights for connections among neurons via error back-propagation and stochastic gradient optimization.
- Underfitting can often be handled by using more powerful models, but risk is overfitting, too good for the training data but bad for generalization to the test data.
- Time-series prediction with ANN needs to re-organize the data into input-output pairs to enable supervised learning.

References

- Ian Goodfellow, Yoshua Bengio, and Aaron Courville, “Deep Learning”. MIT press, 2016.
  `https://www.deeplearningbook.org/`
- Christopher. M. Bishop. ”Pattern Recognition and Machine Learning”. Springer 2006.
  `https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/`
- Trevor Hastie, Robert Tibshirani, and Jerome Friedman. ”The Elements of Statistical Learning: Data Mining, Inference, and Prediction”. Second Edition. Springer Series in Statistics, February 2009.
  `https://web.stanford.edu/~hastie/ElemStatLearn/`