Technische Hochschule Rosenheim

Modul - Introduction to AI - part II (AI2)

Bachelor Programme AAI

07 - Advanced Neural Networks

Prof. Dr. Marcel Tilly

Faculty of Computer Science, Cloud Computing

Agenda

Technische Hochschule Rosenheim

On the menu for today:

- Neural Networks: Hyperparameter
  - Learning Rate
  - Activation Functions
  - Model Topology

Hyperparameter

Technische Hochschule Rosenheim

Hyperparameter are parameters the ANN cannot learn!

Which hyperparamter do you know?

Hyperparameter

Technische Hochschule Rosenheim

Hyperparameter are parameters the ANN cannot learn!

Which hyperparamter do you know?

- Learning rate
- Activation function
- ANN topology: number of layers, neurons per layer

Learning Rate

- Remember, in deep learning, our goal is to *minimize a loss function*.
- If the learning rate is too high, our loss will start jumping all over the place and never converge.
- And if the learning rate is too small, the model will take way too long to converge.

Grid Search

Technische Hochschule Rosenheim

This is the simplest possible way to get good hyperparameters. It's literally just brute force.

The Algorithm

- Try out a bunch of LRs from a given set of LRs, and see what works best.
  - **The Pros**: It's easy enough for a fifth grader to implement. Can be easily parallelized.
  - **The Cons**: As you probably guessed, it's insanely computationally expensive.

Random Search

Technische Hochschule Rosenheim

Random search searches **Randomly**.

The Algorithm

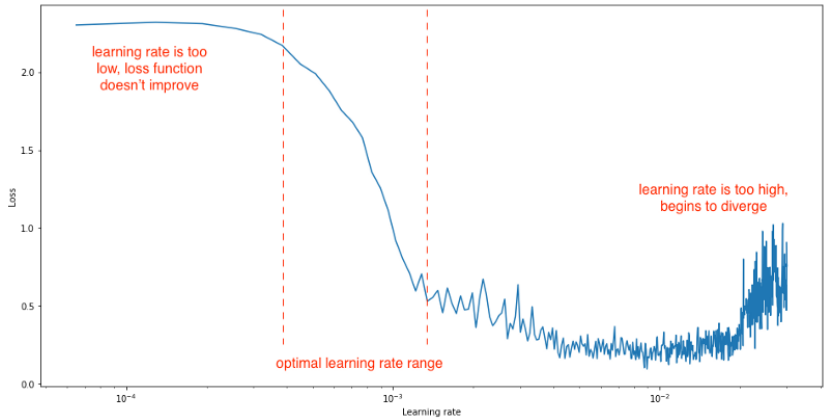
- Try out a bunch of random hyperparameters from a uniform distribution over some hyperparameter space, and see what works best.
  - **The Pros**: Can be easily parallelized. Just as simple as grid search, but a bit better performance, as illustrated
  - **The Cons**: While it gives better performance than grid search, it is still just as computationally intensive

Cyclical Learning Rates

Technische Hochschule Rosenheim

Paper from Leslie N. Smith, "Cyclical Learning Rates for Training Neural Networks"

- Method for scheduling (changing) the learning rate over time.
- Use a *learning rate schedule* that varies the learning rate from a minimum to maximum
  - linearly increasing the learning rate after each iteration e.g. from 1e-7 to 1e-1
  - evaluate the loss at each iteration
  - plot the loss (or test error or accuracy) against the learning rate on a log scale

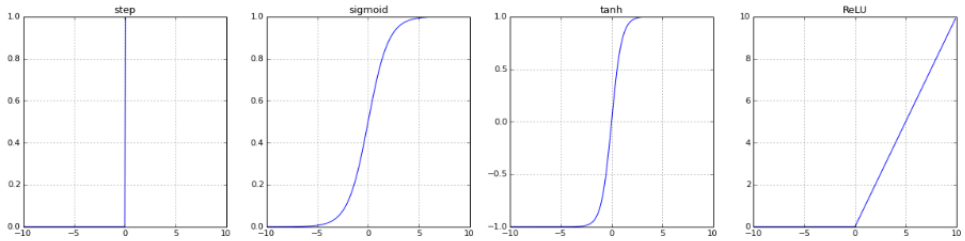


Activation Function (AF)

- What is an activation function and what does it do in a network?
- Why is there a need for it and why not use a linear function instead?
- What are the desirable features in an activation function?
- How (and which) to use them in deep neural networks?

$$f(x,\theta) = \begin{cases} 1, \text{ wenn x } \geq \theta \\ 0, \text{ sonst} \end{cases}$$

$$f(x,\theta) = \tanh(x-\theta)$$



$$f(x,\theta) = \frac{1}{1+e^{-\frac{x-\theta}{T}}} = sigmoid(x,\theta)$$

$$f(x,\theta) = \begin{cases} x, \text{ wenn x } \geq \theta \\ 0, \text{ x } < \theta \end{cases}$$

Activation Function

Technische Hochschule Rosenheim

- An activation function is a function that is added into an artificial neural network in order to help the network learn complex patterns in the data.
- When comparing with a neuron-based model that is in our brains, the activation function is at the end deciding what is to be fired to the next neuron.
  - That is exactly what an activation function does in an ANN as well.
- It takes in the output signal from the previous cell and converts it into some form that can be taken as input to the next cell.

AF- Why is there a need for it?

Technische Hochschule Rosenheim

1. **Limitation**
   - The activation function is $W * x + b$ where W is the weights of the cell and the x is the inputs and then there is the bias b added to that.
   - This value if not restricted to a certain limit can go very high in magnitude especially in case of very deep neural networks that have millions of parameters.
   - This might lead to computational issues.
2. **Non-linearity**
   - The most important feature in an activation function is its ability to add non-linearity into a neural network.

AF- Features of an activation function

Technische Hochschule Rosenheim

1. **Avoid exploding gradient problem**: If the value of `AF()` >> 1, the activations might explode.
2. **Avoid vanishing gradient problem**: If the value of `AF()` is between 0 and 1, then several such values will get multiplied to calculate the gradient of the initial layers
3. **Zero-Centered**: Output of the activation function should be symmetrical at zero so that the gradients do not shift to a particular direction.
4. **Computational Expense**: Activation functions are applied after every layer and need to be calculated millions of times in deep networks. Hence, they should be computationally inexpensive to calculate.
5. **Differentiable**: As mentioned, neural networks are trained using the gradient descent process, hence the layers in the model need to differentiable or at least differentiable in parts.

AF- Extensions

Technische Hochschule Rosenheim

- **Sigmoid**: It is computationally expensive, causes vanishing gradient problem and not zero-centred. This method is generally used for binary classification problems.
- **Softmax**: The softmax is a more generalised form of the sigmoid. It is used in multi-class classification problems Similar to sigmoid, it produces values in the range of 0–1 therefore it is used as the final layer in classification models
- **TanH**: If you compare it to sigmoid, it solves just one problem of being zero-centred.
- **ReLU**: It is easy to compute. Does not cause the *Vanishing Gradient Problem*. It has just one issue of not being zero centred. It suffers from "dying ReLU" problem. Since the output is zero for all negative inputs. It causes some nodes to completely die and not learn anything.
  - Leaky ReLU and Parametric ReLU: It is defined as $f(x) = \max(\alpha x, x)$
  - ReLU6: It is basically ReLU restricted on the positive side and it is defined as $f(x) = \min(\max(0,x),6)$
- **Swish**: It is defined as f(x) = x*sigmoid(x) (by Ramachandran et.al, 2017)

AF- What to use?

Technische Hochschule Rosenheim

Actication functions in in deep neural networks:

- **Tanh** and **sigmoid** cause huge vanishing gradient problems. Hence, they should not be used.
- Start with ReLU in your network.
- If you think the model has stopped learning, then you can replace it with a LeakyReLU to avoid the Dying ReLU problem. However, the Leaky ReLU will increase the computation time a little bit.
- Activation functions work best in their default hyperparameters that are used in popular frameworks such as Tensorflow and Pytorch. However, one can fiddle with the negative slope in LeakyReLU and set it to 0.02 to expedite learning.

Neural Networks revisited

Technische Hochschule Rosenheim

Artificial neural networks have two main hyperparameters that control the architecture or topology of the network

1. the number of layers
2. the number of nodes in each hidden layer

Questions:

- What is the number of hidden layers to use?
- How many hidden neurons in each hidden layer?
- What is the purpose of using hidden layers/neurons?
- Is increasing the number of hidden layers/neurons always gives better results?

Why do we have multiple layers?

Technische Hochschule Rosenheim

- A single-layer neural network can only be used to represent *linearly separable* functions.
  - This means very simple problems where, say, the two classes in a classification problem can be neatly separated by a line.
- A Multilayer Perceptron can be used to represent convex regions.
  - This means that in effect, they can learn to draw shapes around examples in some high-dimensional space that can separate and classify them, overcoming the limitation of linear separability.

Theoretical finding by Lippmann (1987) *"An introduction to computing with neural nets"* that shows that an MLP with two hidden layers is sufficient for creating classification regions of any desired shape.

Input - ?? - Output

Technische Hochschule Rosenheim

Specifically, the universal approximation theorem states that a feedforward network with a linear output layer and at least one hidden layer with any “squashing” activation function (such as the logistic sigmoid activation function) can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units."

Goodfellow et. al., "Deep Learning"; 2016

- How many layer?
- Is one layer sufficient?
- How many neurons?

How many layers and nodes to use?

- **Experimantation**: Trial and error
- **Intuition**: You have experimence and *boroow* ideas/results from others
- **Literature**: Check literature and theoretical work
- **Tools**: Use hyperparameter testing tools for brute-force
  - AutoML
  -

Naive Guidelines for ANNs

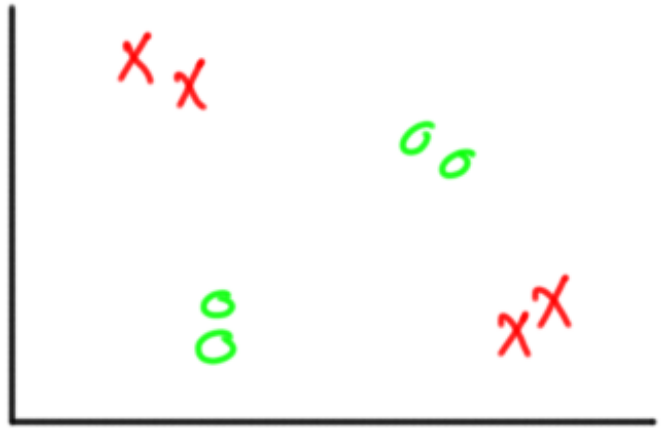
Technische Hochschule Rosenheim

Here are some guidelines to know the number of hidden layers and neurons per each hidden layer in a classification problem:

1. Based on the data, draw an expected decision boundary to separate the classes.
2. Express the decision boundary as a set of lines. Note that the combination of such lines must yield to the decision boundary.
3. The number of selected lines represents the number of hidden neurons in the first hidden layer.
4. To connect the lines created by the previous layer, a new hidden layer is added. Note that a new hidden layer is added each time you need to create connections among the lines in the previous hidden layer.
5. The number of hidden neurons in each new hidden layer equals the number of connections to be made.

Example

XOR-Problem: A classification problem with two classes

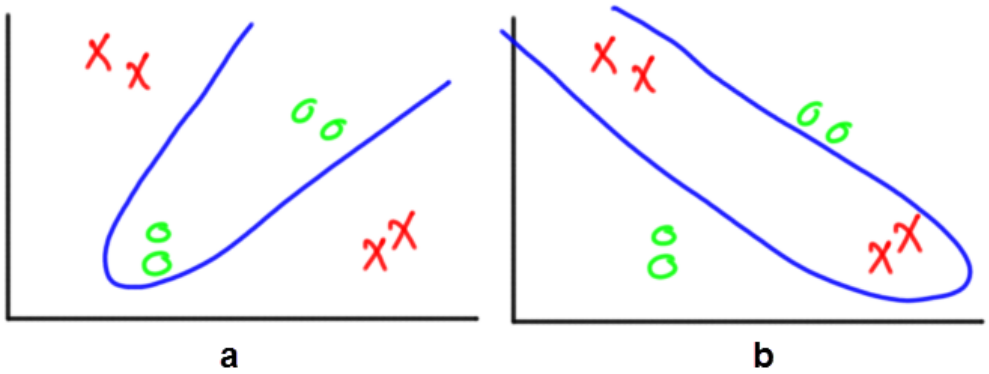


- The first question to answer is whether hidden layers are required or not?

In artificial neural networks, hidden layers are required if and only if the data must be separated non-linearly.

Example

- It seems that the classes must be non-linearly separated. A single line will not work. As a result, we must use hidden layers in order to get the best decision boundary.



In order to add hidden layers, we need to answer these following two questions:

- What is the required number of hidden layers?
- What is the number of the hidden neurons across each hidden layer?

Example

Technische Hochschule Rosenheim

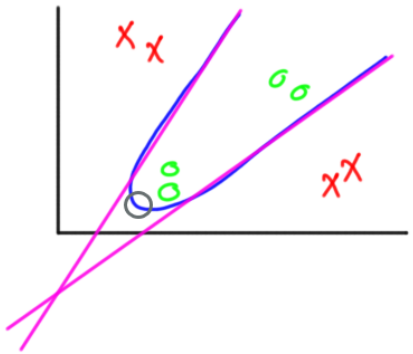
- The idea of representing the decision boundary using a set of lines comes from the fact that any ANN is built using the single layer perceptron as a building block.
- The single layer perceptron is a linear classifier which separates the classes using a line created according to the following equation:

$$y = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_i \cdot x_i + b$$

The ANN is built using multiple perceptron networks. This results in a network that creates multiple lines.

Example

- The decision boundary is replaced by a set of lines.
- The lines start from the points at which the boundary curve changes direction. At such point, two lines are placed, each in a different direction.
- There is just one point at which the boundary curve changes direction. Thus, there will be just two lines required.
- In other words, there are two single layer perceptron networks. Each perceptron produces a line.



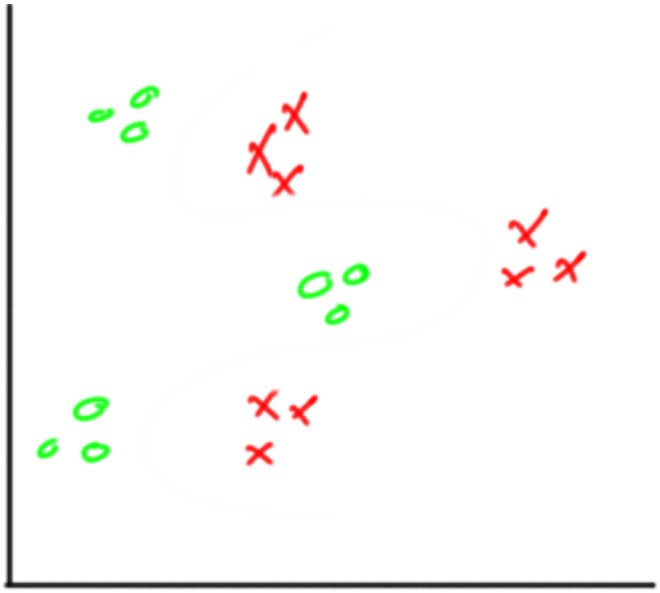
Example

Technische Hochschule Rosenheim

- Up to this point, we have a single hidden layer with two hidden neurons.
- Each hidden neuron could be regarded as a linear classifier that is represented as a line.
- There will be two outputs, one from each classifier (i.e. hidden neuron).
- But we are to build a single classifier with one output representing the class label, not two classifiers.
- As a result, the outputs of the two hidden neurons are to be merged into a single output.
- In other words, the two lines are to be connected by another neuron

Task

How could the ANN look like or the following dataset?



Summary

Technische Hochschule Rosenheim

Lessons learned today:

- Hyperparameter settings of Neural Networks
  - Activation Functions
  - Learning Rate
  - Model topology