

## Coen's AI Notes

diverse



# Table of contents

<b>1</b>	<b>Journey into Artificial Intelligence</b>	<b>1</b>
1.1	Learning Path . . . . .	1
1.2	Some background . . . . .	1
<b>2</b>	<b>The AI Landscape: Understanding the Big Picture</b>	<b>3</b>
2.1	Navigating the World of AI Technologies . . . . .	3
2.2	Key Concepts to Take Away . . . . .	3
<b>I</b>	<b>Perceptron Fundamentals</b>	<b>5</b>
<b>3</b>	<b>Understanding the Perceptron</b>	<b>7</b>
3.1	The Biological Inspiration: From Brain Neurons to Artificial Intelligence . . . . .	7
3.2	From Biology to Machine: Implementing a Perceptron . . . . .	8
<b>4</b>	<b>Practical Applications of the Perceptron</b>	<b>9</b>
4.1	Building Logic Gates with Perceptrons . . . . .	9
4.1.1	Creating an AND Gate . . . . .	9
4.2	Beyond AND Gates . . . . .	10
4.3	Network . . . . .	10
4.4	Reference . . . . .	10
<b>5</b>	<b>Decision Making with Perceptrons</b>	<b>13</b>
5.1	Making Decisions . . . . .	13
5.2	Understanding the Decision Boundary . . . . .	13
5.3	Applications . . . . .	14
<b>6</b>	<b>Teaching a Perceptron: The Learning Process</b>	<b>15</b>
6.1	Introduction to Perceptron Learning . . . . .	15
6.2	The Learning Algorithm . . . . .	15
6.2.1	Mathematical Foundation . . . . .	15
6.3	Implementing Learning . . . . .	16
6.3.1	Training Process . . . . .	16

6.4	Visualizing the Learning Process . . . . .	16
6.5	Practical Considerations . . . . .	16
<b>7</b>	<b>Understanding Perceptron Limitations</b>	<b>17</b>
7.1	The XOR Problem: A Classic Challenge . . . . .	17
7.1.1	What is XOR? . . . . .	17
7.1.2	Why Can't a Single Perceptron Solve XOR? . . . . .	18
7.2	The Solution: Multiple Layers . . . . .	19
7.3	Key Takeaways . . . . .	21
<b>II</b>	<b>Neural Networks</b>	<b>23</b>
<b>8</b>	<b>Introduction to Neural Networks</b>	<b>25</b>
8.1	Beyond Single Perceptrons: Building Neural Networks . . . . .	25
8.2	Understanding Network Architecture . . . . .	26
8.2.1	Key Components . . . . .	26
8.3	How Information Flows . . . . .	26
8.4	Creating a Simple Network . . . . .	27
8.5	Training the Network . . . . .	27
8.6	Advantages of Neural Networks . . . . .	27
<b>9</b>	<b>Practical Example: Classifying Iris Flowers</b>	<b>29</b>
9.1	A Real-World Machine Learning Challenge . . . . .	29
9.2	The Dataset . . . . .	30
9.3	Building the Neural Network . . . . .	30
9.4	Preparing the Data . . . . .	31
9.5	Training Process . . . . .	31
9.6	Making Predictions . . . . .	31
9.7	Evaluating Performance . . . . .	31
9.8	Key Learning Points . . . . .	31
<b>10</b>	<b>The Mathematics Behind Neural Networks</b>	<b>33</b>
10.1	Understanding the Magic . . . . .	33
10.2	The Building Blocks . . . . .	33
10.2.1	1. Neurons and Weights . . . . .	33
10.2.2	2. Activation Functions . . . . .	33
10.3	The Learning Process . . . . .	34
10.3.1	1. Forward Propagation . . . . .	34
10.3.2	2. Loss Calculation . . . . .	34
10.3.3	3. Backpropagation . . . . .	34
10.4	Gradient Descent Visualization . . . . .	34
10.5	Practical Implementation . . . . .	34
10.6	Key Insights . . . . .	34
10.7	Beyond the Basics . . . . .	35

<b>11 Advanced Training Techniques</b>	<b>37</b>
11.1 Beyond Basic Training . . . . .	37
11.2 The Challenge of Overfitting . . . . .	37
11.2.1 Understanding Overfitting . . . . .	37
11.2.2 Solutions to Overfitting . . . . .	37
11.3 Data Augmentation . . . . .	38
11.4 Batch Processing . . . . .	38
11.4.1 Mini-batch Training . . . . .	38
11.5 Learning Rate Scheduling . . . . .	38
11.6 Transfer Learning . . . . .	39
11.7 Monitoring and Visualization . . . . .	39
11.8 Best Practices . . . . .	39
11.9 Next Steps . . . . .	40
<b>12 Exploring Neural Network Architectures</b>	<b>41</b>
12.1 The Rich Landscape of Neural Networks . . . . .	41
12.2 Feedforward Neural Networks (FNN) . . . . .	41
12.3 Convolutional Neural Networks (CNN) . . . . .	41
12.4 Recurrent Neural Networks (RNN) . . . . .	41
12.5 Long Short-Term Memory (LSTM) . . . . .	42
12.6 Autoencoders . . . . .	42
12.7 Generative Adversarial Networks (GAN) . . . . .	42
12.8 Choosing the Right Architecture . . . . .	42
12.9 Future Directions . . . . .	42
<b>III Next Steps</b>	<b>43</b>
<b>13 Next Steps in Your AI Journey</b>	<b>45</b>
13.1 Congratulations on Your Progress! . . . . .	45
13.2 Expanding Your Knowledge . . . . .	45
13.2.1 1. Advanced Topics . . . . .	45
13.2.2 2. Practical Skills . . . . .	45
13.3 Real-World Applications . . . . .	46
13.3.1 1. Industry Applications . . . . .	46
13.3.2 2. Research Areas . . . . .	46
13.4 Building Your Portfolio . . . . .	46
13.5 Community Engagement . . . . .	47
13.5.1 1. Online Communities . . . . .	47
13.5.2 2. Local Groups . . . . .	47
13.6 Continuous Learning . . . . .	47
13.6.1 1. Advanced Courses . . . . .	47
13.6.2 2. Reading Materials . . . . .	47
13.7 Career Paths . . . . .	47
13.8 Best Practices Moving Forward . . . . .	48
13.9 Final Thoughts . . . . .	48

<b>14 Python for Neural Networks</b>	<b>49</b>
14.1 Why Python for Neural Networks?	49
14.2 Essential Python Libraries	49
14.2.1 1. NumPy	49
14.2.2 2. TensorFlow/Keras	49
14.2.3 3. PyTorch	50
14.3 Getting Started	50
14.4 From Pharo to Python	50
14.4.1 Key Differences	50
14.5 Resources for Learning	50
14.6 Best Practices	51
14.7 Next Steps	51
<b>15 Finding and Preparing Data for Neural Networks</b>	<b>53</b>
15.1 The Importance of Data	53
15.2 Popular Data Sources	53
15.2.1 1. Public Datasets	53
15.2.2 2. Domain-Specific Sources	53
15.3 Data Preparation Steps	54
15.4 Best Practices	54
15.4.1 1. Data Quality	54
15.4.2 2. Data Split	54
15.4.3 3. Data Augmentation	55
15.5 Common Challenges	55
15.6 Tools and Libraries	55
15.7 Next Steps	55
<b>16 Essential Resources and References</b>	<b>57</b>
16.1 Core Learning Resources	57
16.1.1 Coen's Links	57
16.1.2 Books	57
16.2 Video Courses and Tutorials	58
16.2.1 1. Foundational Series	58
16.2.2 2. Programming Tutorials	58
16.2.3 3. Advanced Topics	58
16.3 Online Platforms	58
16.3.1 1. Interactive Learning	58
16.3.2 2. Research Papers	58
16.3.3 3. Code Repositories	58
16.4 Community Resources	59
16.4.1 1. Forums and Discussion	59
16.4.2 2. Blogs and Newsletters	59
16.4.3 3. Tools and Libraries	59
16.5 Academic Papers	59
16.5.1 1. Foundational Papers	59
16.5.2 2. Modern Breakthroughs	59

# Chapter 1

## Journey into Artificial Intelligence

Welcome! This guide will always be a Work-in-Progress which can take you from the fundamental building blocks of AI onto neural networks.

### 1.1 Learning Path

1. **AI Overview.**
2. **Perceptron Fundamentals:** The basic building block of neural networks - the Perceptron.
3. **Neural Networks:** How will multiple Perceptrons combine to create neural networks capable of solving complex problems.
4. **Possible next Steps.**

### 1.2 Some background

1. It started out as visualizations of Perceptrons and Neural Networks in the Glamorous Toolkit, which helped me give students insights in Neural Networks.
2. Get Hands-on: start using and trying out AI-tools you encounter.
3. When using online AI tools, please keep the privacy in mind when using personal data!!
4. Please also keep the Societal impact in mind! We can use AI to help us all, but there is of course also a dark side! When concentrating on efficiency only that could mean (and often does!) people getting fired.





## Chapter 2

# The AI Landscape: Understanding the Big Picture

### 2.1 Navigating the World of AI Technologies

In today's rapidly evolving technological landscape, terms like *AI*, *Machine Learning*, *Deep Learning*, and *Generative AI* are frequently used, but how do they relate to each other? Let's explore these interconnected concepts through an engaging and informative video presentation.

The video "AI, Machine Learning, Deep Learning and Generative AI Explained" provides an excellent 10-minute overview that will help you understand how these different technologies fit together in the broader AI ecosystem. You can watch it here:

[AI, Machine Learning, Deep Learning and Generative AI Explained](#)

### 2.2 Key Concepts to Take Away

After watching the video, you'll understand: - Where Machine Learning and Deep Learning fit within the AI landscape.



## Part I

# Perceptron Fundamentals



## Chapter 3

# Understanding the Perceptron

### 3.1 The Biological Inspiration: From Brain Neurons to Artificial Intelligence

The Perceptron represents one of the most fundamental concepts in artificial intelligence, drawing its inspiration directly from the human brain's neural structure. This groundbreaking idea was first introduced in 1943 by Warren S. McCulloch and Walter Pitts in their seminal paper 'A Logical Calculus of the Ideas Immanent in Nervous Activity', where they proposed a mathematical model of biological neurons.

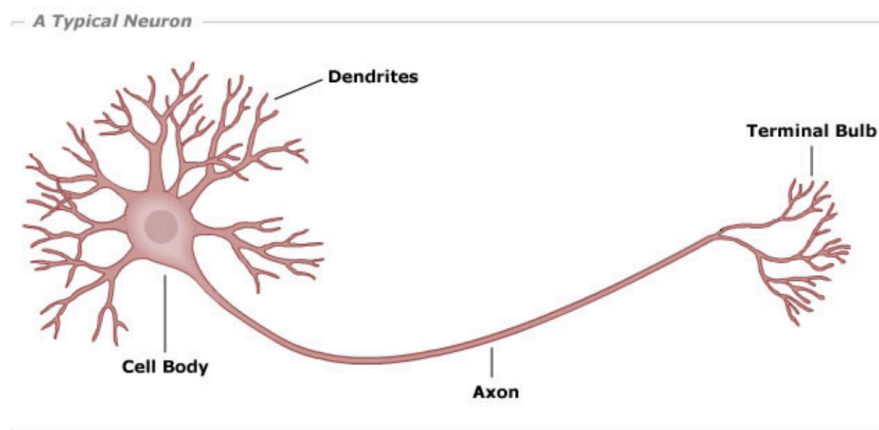


Figure 3.1: A typical biological neuron structure

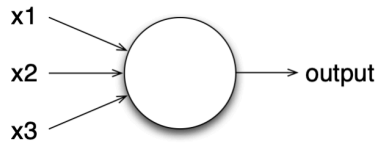
## 3.2 From Biology to Machine: Implementing a Perceptron

A Perceptron's architecture elegantly mirrors its biological counterpart through three key components: **inputs**, **weights**, and a **bias**. Each input connection has an associated weight that determines its relative importance, while the bias helps adjust the Perceptron's overall sensitivity to activation.

### Perceptron

---

A *perceptron* is a kind of *artificial neuron*



Takes several binary inputs,  $x_1, x_2, \dots$  and produces a single binary output

Figure 3.2: Perceptron's architectural diagram

Let's explore a practical example with three inputs. We'll call our input values  $x_1, x_2$ , and  $x_3$ , with their corresponding weights  $w_1, w_2$ , and  $w_3$ . The Perceptron processes these inputs in two steps:

1. First, it calculates a **weighted sum** and adds the bias:  $z := w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + \text{bias}$
2. Then, it applies what we call an **activation function** to produce the final output: let's use a very simple one, called a Step function:

$$\begin{cases} \text{Output is 1 if } z > 0 \\ \text{Output is 0 if } z \leq 0 \end{cases}$$

which determines the final output.

Keep in mind that the number of inputs for a Perceptron can vary.

## Chapter 4

# Practical Applications of the Perceptron

### 4.1 Building Logic Gates with Perceptrons

Let's look at an example of how Perceptrons can be used?

#### 4.1.1 Creating an AND Gate

You may know the concept of an AND gate: given two inputs (both can be 0 or 1) the AND gate will output a 1 if both inputs are 1, and 0 in all other cases.

Consider a Perceptron with the following configuration: - Weights:  $w_1 = 1$ ,  $w_2 = 1$  - Bias:  $-1.5$

Here's the truth table for an AND gate:

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

One example: when inputs are 1 and 1 the weighted sum is  $(1 * 1 + 1 * 1)$ , adding the bias gives:  $0.5$ , which is greater than 0, so the activation function will give 1.

## 4.2 Beyond AND Gates

By changing the bias to  $-0.5$  this Perceptron turns into an OR gate (which returns 1 if at least one of the inputs is 1)

Changing the values can give you a NOR gate and a NOT gate, which would be nice to figure out yourself (or use your preferred search engine).

## 4.3 Network

By combining several Perceptrons (sending the output of one to the input of another) you can probably imagine that it is possible to create Networks of Perceptrons. By changing the values of weights and biases of the connected Perceptrons it is possible to build complex electronic circuits.

So far, we've only looked at binary circuits where inputs and outputs are restricted to 0 and 1. However, when we generalize this concept to allow larger positive values, negative values, floating-point numbers, and different activation functions, the Perceptron becomes an incredibly versatile tool. This generalization opens up possibilities for pattern recognition, classification tasks, regression problems, and complex decision-making systems. This is where the true power of neural networks begins to emerge, as they can learn to handle continuous data and make sophisticated decisions based on multiple inputs.

In relatively simple cases it can be used as sort of a decision machine. More complex applications of this technology can help recognizing objects in the real world.

Up until now we didn't look at how a perceptron can learn and become smarter. That will be subject of next chapter chapters. The concept of a Perceptron was generalized to what we now call an (artificial) Neuron.

When combining Artificial Perceptrons/Neurons to Networks they are referred to as **Multi Layered Perceptron (MLP)** or **(Artificial) Neural Network (ANN)**.

## 4.4 Reference

- [wikipedia: perceptron](#)



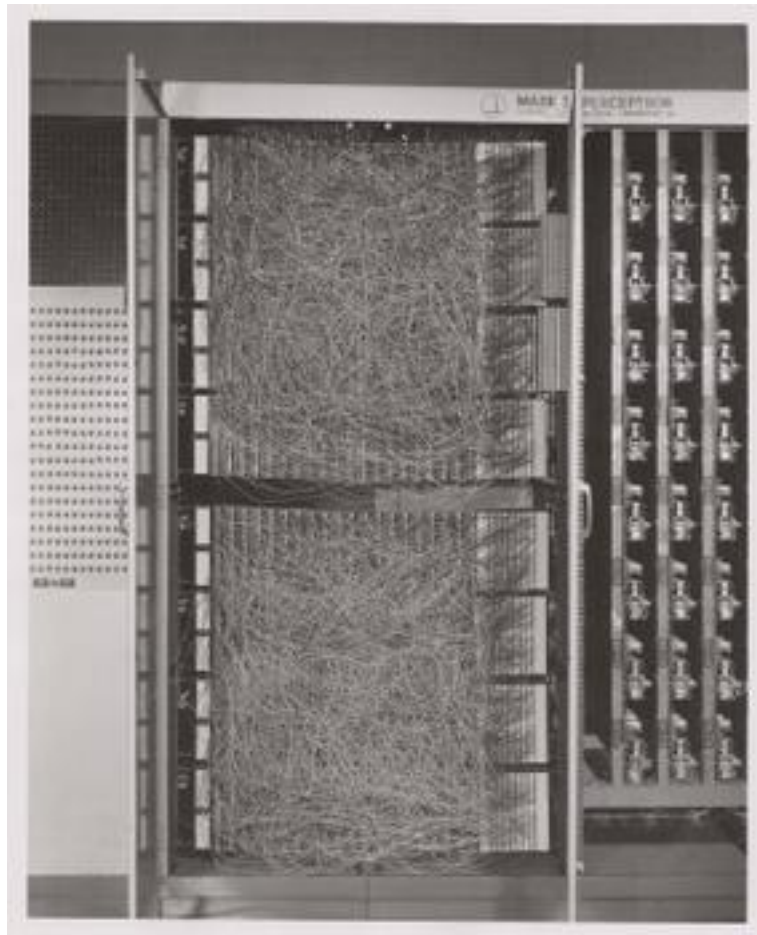


Figure 4.1: The Mark I Perceptron machine, the first implementation of the perceptron algorithm



## Chapter 5

# Decision Making with Perceptrons

### 5.1 Making Decisions

A Perceptron can be used to make decisions based on multiple inputs. Let's look at a practical example, where we take a perceptron with weights 0.5 and  $-0.8$  and bias 0.3

This Perceptron takes two inputs and makes a decision based on their values. The weights and bias determine how the Perceptron interprets the inputs.

For example, if we have inputs  $x_1 = 1$  and  $x_2 = 0.5$ , the Perceptron will: 1. Calculate the weighted sum:  $0.5 \cdot 1 + (-0.8) \cdot 0.5 = 0.1$  2. Add the bias:  $0.1 + 0.3 = 0.4$  3. Apply the step function: since  $0.4 > 0$ , output will be 1

This means the Perceptron has decided “yes” for these input values.

### 5.2 Understanding the Decision Boundary

The weights and bias create a decision boundary in the input space. Any point above this boundary will result in an output of 1, while points below will result in 0.

For our example: - Weight 1 (0.5) determines how much we value the first input - Weight 2 (-0.8) determines how much we value the second input - The bias (0.3) shifts the decision boundary

This creates a line where:  $0.5x_1 - 0.8x_2 + 0.3 = 0$

Points above this line will result in a “yes” decision, while points below will result in a “no” decision.

## 5.3 Applications

This decision-making capability can be used for:

- Classification problems
- Pattern recognition
- Simple rule-based systems
- Binary decisions based on multiple factors

The beauty of this approach is that by adjusting the weights and bias, we can create different decision boundaries for different types of problems.

## Chapter 6

# Teaching a Perceptron: The Learning Process

### 6.1 Introduction to Perceptron Learning

One of the most fascinating aspects of Perceptrons is their ability to learn from examples. Instead of manually setting weights and bias, we can train a Perceptron to discover the optimal parameters through a process called supervised learning.

### 6.2 The Learning Algorithm

The learning process follows these key steps:

1. Start with random weights and bias
2. Present a training example
3. Compare the Perceptron's output with the desired output
4. Adjust the weights and bias based on the error
5. Repeat with more examples until performance is satisfactory

#### 6.2.1 Mathematical Foundation

The weight update rule is elegantly simple:

```
new_weight = current_weight + learning_rate * error * input
```

Where: - **learning\_rate** is a small number (like 0.1) that controls how big each adjustment is - **error** is the difference between desired and actual output (1 or -1) - **input** is the input value for that weight

## 6.3 Implementing Learning

Here's how we create a learning Perceptron:

```
learningPerceptron := Neuron new
  step;
  learningRate: 0.1;
  initialize. "Sets random initial weights"
```

### 6.3.1 Training Process

To train the Perceptron, we present examples with their desired outputs:

```
"Training for AND gate behavior"
trainingData := #(
  ((0 0) 0)
  ((0 1) 0)
  ((1 0) 0)
  ((1 1) 1)
).

trainingData do: [:example |
  inputs := example first.
  desiredOutput := example second.
  learningPerceptron train: inputs desiredOutput: desiredOutput
].
```

## 6.4 Visualizing the Learning Process

As the Perceptron learns, its decision boundary gradually moves to the correct position. You can monitor this progress by:

1. Tracking the error rate over time
2. Visualizing the decision boundary's movement
3. Testing the Perceptron with new examples

## 6.5 Practical Considerations

For successful learning: - Ensure your training data is representative - Consider using multiple training epochs (complete passes through the data) - Monitor for convergence (when the weights stabilize) - Be aware that not all problems are linearly separable

In the next chapter, we'll explore the limitations of what a single Perceptron can learn, which will lead us naturally to the need for more complex neural networks.

## Chapter 7

# Understanding Perceptron Limitations

### 7.1 The XOR Problem: A Classic Challenge

While Perceptrons are powerful tools for many classification tasks, they face a fundamental limitation: they can only solve linearly separable problems. The classic example of this limitation is the XOR (exclusive OR) function.

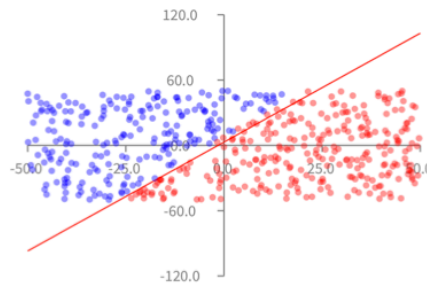
#### 7.1.1 What is XOR?

The XOR function outputs 1 only when exactly one of its inputs is 1: - Input (0,0) → Output: 0 - Input (0,1) → Output: 1 - Input (1,0) → Output: 1 - Input (1,1) → Output: 0

## What we have seen so far

---

We have seen that the perceptron can (more or less accurately) guess the side on which a point is located



34

Figure 7.1: Visual representation of XOR problem

### 7.1.2 Why Can't a Single Perceptron Solve XOR?

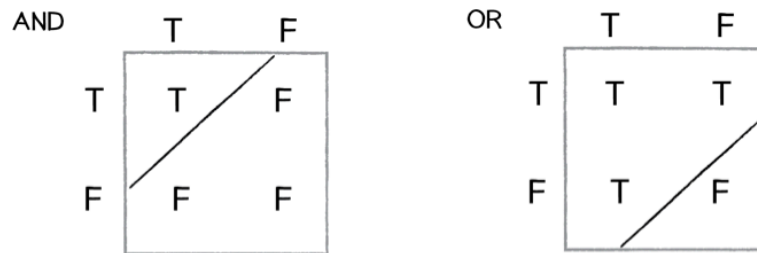
A Perceptron creates a single straight line (or hyperplane in higher dimensions) to separate its outputs. However, the XOR problem requires two separate lines to correctly classify all points.



## What we have seen so far

---

We can easily make our perceptron to represent the AND, OR logical operations



38

Figure 7.2: Attempted linear separation of XOR

As you can see, no single straight line can separate the points where output should be 1 (blue) from points where output should be 0 (red).

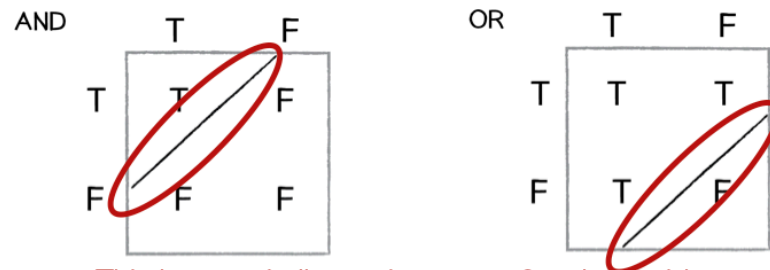
## 7.2 The Solution: Multiple Layers

To solve the XOR problem, we need to combine multiple Perceptrons in layers. This is our first glimpse at why we need neural networks!

## What we have seen so far

---

We can easily make our perceptron to represent the AND, OR logical operations



This is very similar to the space & point problem.  
It is all about having a line as a limit

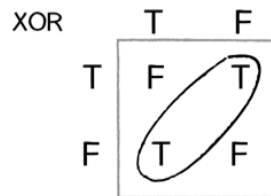
39

Figure 7.3: Multi-layer solution

By using multiple Perceptrons, we can: 1. First create separate regions with individual Perceptrons 2. Then combine these regions to form more complex decision boundaries

## Limitation of a perceptron

---



With the XOR operation, you cannot have one unique line that limit the range of true and false

40

Figure 7.4: Complete neural network solution

## 7.3 Key Takeaways

1. Single Perceptrons can only solve linearly separable problems
2. Many real-world problems (like XOR) are not linearly separable
3. Combining Perceptrons into networks overcomes this limitation
4. This limitation led to the development of multi-layer neural networks

In the next section, we'll explore how to build and train these more powerful multi-layer networks.



# Part II

## Neural Networks



## Chapter 8

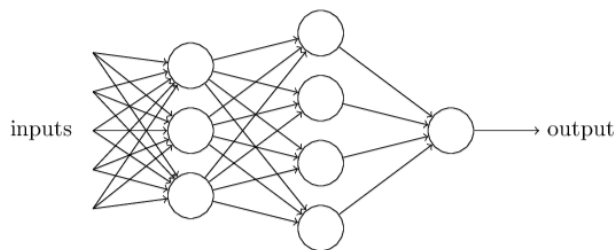
# Introduction to Neural Networks

### 8.1 Beyond Single Perceptrons: Building Neural Networks

Having seen the limitations of single Perceptrons, we now venture into the fascinating world of neural networks. These powerful structures combine multiple Perceptrons in layers to solve complex problems that single Perceptrons cannot handle.

## Network of neurons

A network has the following structure



41

Figure 8.1: Basic neural network architecture

## 8.2 Understanding Network Architecture

A typical neural network consists of three main components:

1. **Input Layer:** Receives the raw data
2. **Hidden Layer(s):** Processes the information through multiple Perceptrons
3. **Output Layer:** Produces the final result

### 8.2.1 Key Components

Each connection in the network has:

- A weight that determines its strength
- A direction of information flow (forward only)
- An associated neuron that processes the incoming signals

## 8.3 How Information Flows

The network processes information in these steps:

1. Input values are presented to the input layer
2. Each neuron in subsequent layers:



- Receives weighted inputs from the previous layer
  - Applies its activation function
  - Passes the result to the next layer
3. The output layer produces the final result

## 8.4 Creating a Simple Network

Here's how to create a basic neural network:

```
network := NeuralNetwork new
  inputSize: 2;
  addHiddenLayer: 3;
  outputSize: 1;
  initialize.
```

This creates a network with: - 2 input neurons - 3 neurons in one hidden layer  
- 1 output neuron

## 8.5 Training the Network

Unlike single Perceptrons, neural networks use more sophisticated training algorithms:

```
"Training data for XOR problem"
trainingData := #(
  ((0 0) 0)
  ((0 1) 1)
  ((1 0) 1)
  ((1 1) 0)
).

"Train the network"
1000 timesRepeat: [
  trainingData do: [:example |
    inputs := example first.
    desiredOutput := example second.
    network trainOn: inputs expecting: desiredOutput
  ]
].
```

## 8.6 Advantages of Neural Networks

1. Can solve non-linearly separable problems
2. Handle complex pattern recognition
3. Learn hierarchical features automatically

4. Scale well to large problems

In the next sections, we'll explore practical applications and see how to train networks on real-world data.

## Chapter 9

# Practical Example: Classifying Iris Flowers

### 9.1 A Real-World Machine Learning Challenge

The Iris flower classification problem is a classic example in machine learning. It involves predicting the species of an Iris flower based on measurements of its physical characteristics. This problem perfectly illustrates how neural networks can solve real-world classification tasks.



Figure 9.1: Different types of Iris flowers

## 9.2 The Dataset

The Iris dataset includes measurements of three different Iris species: - Iris Setosa - Iris Versicolor - Iris Virginica

For each flower, we have four measurements: 1. Sepal length 2. Sepal width 3. Petal length 4. Petal width

## 9.3 Building the Neural Network

Let's create a network to classify Iris flowers:

```
irisNetwork := NeuralNetwork new
  inputSize: 4;           "Four measurements"
  addHiddenLayer: 5;      "Hidden layer with 5 neurons"
  outputSize: 3;          "Three possible species"
  initialize.
```

## 9.4 Preparing the Data

We need to format our data appropriately:

```
"Example of one flower's data"
measurements := #(5.1 3.5 1.4 0.2). "Setosa"
expectedOutput := #(1 0 0). "One-hot encoding for Setosa"
```

## 9.5 Training Process

```
"Training with multiple examples"
trainingData do: [:example |
    measurements := example measurements.
    species := example species.
    irisNetwork trainOn: measurements expecting: species
].
```

## 9.6 Making Predictions

After training, we can use the network to classify new flowers:

```
newFlower := #(6.3 2.9 5.6 1.8).
prediction := irisNetwork predict: newFlower.
```

## 9.7 Evaluating Performance

To assess our network's accuracy:

1. Split data into training and testing sets
2. Train on the training set
3. Evaluate on the testing set
4. Calculate accuracy metrics

## 9.8 Key Learning Points

1. Neural networks can handle multi-class classification
2. Real-world data often needs preprocessing
3. We can measure success with accuracy metrics
4. The same principles apply to many similar problems

This practical example demonstrates how neural networks can solve real classification problems. In the next section, we'll explore the mathematics behind how these networks learn.



## Chapter 10

# The Mathematics Behind Neural Networks

### 10.1 Understanding the Magic

While neural networks might seem magical, they're built on solid mathematical foundations. Let's demystify how they actually work under the hood.

### 10.2 The Building Blocks

#### 10.2.1 1. Neurons and Weights

Each neuron performs two key operations: 1. Weighted sum of inputs:  $z = \sum_{i=1}^n w_i x_i + b$  2. Activation function:  $a = f(z)$

Where: -  $w_i$  are the weights -  $x_i$  are the inputs -  $b$  is the bias -  $f$  is the activation function

#### 10.2.2 2. Activation Functions

Common activation functions include:

1. Sigmoid:  $f(x) = \frac{1}{1+e^{-x}}$ 
  - Outputs between 0 and 1
  - Useful for probability predictions
2. ReLU (Rectified Linear Unit):  $f(x) = \max(0, x)$ 
  - Simple and efficient
  - Helps prevent vanishing gradients
3. Tanh:  $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ 
  - Outputs between -1 and 1

- Often better than sigmoid for hidden layers

## 10.3 The Learning Process

### 10.3.1 1. Forward Propagation

Information flows through the network:

```
"Example of forward propagation"
layer1Activation := (weights1 dot: inputs) + bias1.
layer1Output := activationFunction value: layer1Activation.
```

### 10.3.2 2. Loss Calculation

Measure the network's error:  $E = \frac{1}{2} \sum (target - output)^2$

### 10.3.3 3. Backpropagation

Update weights to minimize error:  $\Delta w = -\eta \frac{\partial E}{\partial w}$

Where  $\eta$  is the learning rate.

## 10.4 Gradient Descent Visualization

The network learns by descending the error surface: 1. Calculate error gradient  
2. Take small steps in the opposite direction 3. Repeat until reaching a minimum

## 10.5 Practical Implementation

In Pharo, we can implement these concepts:

```
NeuralNetwork >> updateWeights: inputs error: error
    "Update weights using gradient descent"
    learningRate := 0.1.
    delta := error * self derivativeActivation: self lastOutput.
    weights := weights + (learningRate * (inputs * delta))
```

## 10.6 Key Insights

1. Neural networks learn through iterative optimization
2. The choice of activation function matters
3. Learning rate affects training stability
4. Gradient descent finds local minima



## 10.7 Beyond the Basics

Advanced concepts build on these foundations: - Momentum for faster convergence - Regularization to prevent overfitting - Batch normalization for stability - Advanced optimizers like Adam

Understanding these mathematical principles helps us: 1. Debug network issues 2. Choose appropriate architectures 3. Optimize performance 4. Innovate new solutions



# Chapter 11

## Advanced Training Techniques

### 11.1 Beyond Basic Training

While we've covered the fundamentals of neural network training, there are many advanced techniques that can significantly improve performance and efficiency.

### 11.2 The Challenge of Overfitting

#### 11.2.1 Understanding Overfitting

Overfitting occurs when a model learns the training data too well: - Memorizes training examples instead of learning patterns - Performs poorly on new, unseen data - Shows high training accuracy but low test accuracy

#### 11.2.2 Solutions to Overfitting

1. **Regularization**

```
"L2 Regularization example"  
regularizedError := error + (lambda * weights squared sum)
```

2. **Dropout**

- Randomly deactivate neurons during training
- Forces the network to be more robust
- Typically 20-50% dropout rate

3. **Early Stopping**

- Monitor validation performance

- Stop when performance starts degrading
- Save best performing model

## 11.3 Data Augmentation

Increase training data variety:

```
"Image augmentation example"
augmentedImage := originalImage
  rotate: (Random new nextFloat * 15);
  scale: (0.9 to: 1.1);
  addNoise: 0.05
```

Common augmentation techniques: 1. Rotation and scaling 2. Adding noise 3. Color adjustments 4. Random cropping

## 11.4 Batch Processing

### 11.4.1 Mini-batch Training

Benefits of batch processing: - Faster convergence - Better generalization - More stable gradients

```
"Mini-batch training example"
batchSize := 32.
batches := trainingData batchesOf: batchSize.
batches do: [:batch |
  gradients := self computeGradients: batch.
  self updateWeights: gradients
]
```

## 11.5 Learning Rate Scheduling

Adaptive learning rates improve training:

### 1. Step Decay

```
"Step decay example"
learningRate := initialRate * (decayFactor raisedTo: epochNumber // stepSize)
```

### 2. Exponential Decay

```
"Exponential decay"
learningRate := initialRate * (decayBase raisedTo: epochNumber)
```

### 3. Cosine Annealing

- Cyclical learning rates

- Helps escape local minima
- Enables better exploration

## 11.6 Transfer Learning

Leverage pre-trained models:

1. Feature Extraction
  - Use pre-trained network as feature extractor
  - Add custom layers for your task
  - Freeze pre-trained weights
2. Fine-tuning

```
"Fine-tuning example"
pretrainedNetwork
  freezeLayersUpTo: -2;
  addLayer: (Dense neurons: outputSize);
  trainOn: newData
```

## 11.7 Monitoring and Visualization

Track training progress:

1. Loss Curves
  - Plot training and validation loss
  - Identify overfitting early
  - Guide hyperparameter tuning
2. Confusion Matrix

```
"Generate confusion matrix"
confusionMatrix := predictions zip: actualLabels collect: [:pred :actual |
  (pred = actual) asBit
] groupedBy: #yourself
```

## 11.8 Best Practices

1. Data Preparation
  - Normalize inputs
  - Handle missing values
  - Balance classes
2. Model Architecture
  - Start simple
  - Gradually add complexity
  - Use proven architectures

### 3. Training Process

- Monitor key metrics
- Save checkpoints
- Use cross-validation

## 11.9 Next Steps

Advanced techniques to explore: 1. Ensemble methods 2. Hyperparameter optimization 3. Advanced architectures 4. Custom loss functions

Remember: These techniques are tools in your toolkit. Choose them based on your specific problem and requirements.

## Chapter 12

# Exploring Neural Network Architectures

### 12.1 The Rich Landscape of Neural Networks

Neural networks come in many shapes and sizes, each designed to excel at specific types of tasks. Let's explore some of the most important architectures and their applications.

### 12.2 Feedforward Neural Networks (FNN)

The classic architecture we've been working with so far. Information flows in one direction: - Input layer  $\rightarrow$  Hidden layer(s)  $\rightarrow$  Output layer - Perfect for classification and regression tasks - Examples: Our Iris classifier, handwriting recognition

### 12.3 Convolutional Neural Networks (CNN)

Inspired by the human visual cortex: - Specialized for processing grid-like data (images, video) - Uses convolution operations to detect patterns - Excellent at feature extraction - Applications: Image recognition, computer vision, medical imaging

### 12.4 Recurrent Neural Networks (RNN)

Networks with memory: - Can process sequences of data - Information cycles through the network - Great for time-series data and natural language - Applications: Language translation, speech recognition, stock prediction

## 12.5 Long Short-Term Memory (LSTM)

A sophisticated type of RNN: - Better at remembering long-term dependencies  
- Controls information flow with gates - Solves the vanishing gradient problem -  
Applications: Text generation, music composition

## 12.6 Autoencoders

Self-learning networks: - Learn to compress and reconstruct data - Useful for  
dimensionality reduction - Can detect anomalies - Applications: Data compression,  
noise reduction, feature learning

## 12.7 Generative Adversarial Networks (GAN)

Two networks competing with each other: - Generator creates fake data - Discriminator  
tries to spot fakes - Through competition, both improve - Applications: Creating realistic  
images, style transfer, data augmentation

## 12.8 Choosing the Right Architecture

The choice of architecture depends on: 1. Type of data (images, text, time-series)  
2. Task requirements (classification, generation, prediction) 3. Available computational  
resources 4. Need for real-time processing

## 12.9 Future Directions

Neural network architectures continue to evolve: - Hybrid architectures combining  
multiple types - More efficient training methods - Better handling of uncertainty -  
Integration with other AI techniques

In the next section, we'll dive deeper into training these networks effectively.



## Part III

# Next Steps



## Chapter 13

# Next Steps in Your AI Journey

### 13.1 Congratulations on Your Progress!

You've come a long way in understanding neural networks and their applications. Now, let's explore where to go from here.

### 13.2 Expanding Your Knowledge

#### 13.2.1 1. Advanced Topics

- Deep Learning architectures
- Reinforcement Learning
- Natural Language Processing
- Computer Vision
- Generative AI

#### 13.2.2 2. Practical Skills

- Model deployment
- Cloud computing
- Version control
- Data engineering
- DevOps for AI

## 13.3 Real-World Applications

### 13.3.1 1. Industry Applications

- Healthcare diagnostics
- Financial forecasting
- Autonomous systems
- Robotics
- Smart cities

### 13.3.2 2. Research Areas

- Explainable AI
- Ethical AI
- Federated Learning
- Few-shot Learning
- Neural Architecture Search

## 13.4 Building Your Portfolio

### 1. Personal Projects

```
"Example project structure"
AIProject new
  title: 'Image Classification';
  description: 'Classifying plant species';
  technologies: #('CNN' 'Transfer Learning');
  dataset: 'PlantNet';
  initialize
```

### 2. Documentation

- Clear README files
- Architecture diagrams
- Performance metrics
- Deployment instructions

### 3. Code Quality

- Clean code principles
- Unit tests
- Performance optimization
- Error handling

## 13.5 Community Engagement

### 13.5.1 1. Online Communities

- AI/ML forums
- GitHub discussions
- Stack Overflow
- Research paper discussions

### 13.5.2 2. Local Groups

- Meetups
- Hackathons
- Workshops
- Study groups

## 13.6 Continuous Learning

### 13.6.1 1. Advanced Courses

- Deep Learning specializations
- MLOps certifications
- Domain-specific training
- Research methodologies

### 13.6.2 2. Reading Materials

- Research papers
- Technical blogs
- Industry reports
- Case studies

## 13.7 Career Paths

### 1. Industry Roles

- Machine Learning Engineer
- AI Researcher
- Data Scientist
- MLOps Engineer

### 2. Research Paths

- PhD programs
- Research labs
- Academic positions
- Industry research

## 13.8 Best Practices Moving Forward

### 1. Stay Current

- Follow AI news
- Read research papers
- Experiment with new tools
- Join discussions

### 2. Build Network

- Connect with experts
- Share knowledge
- Collaborate on projects
- Mentor others

### 3. Maintain Balance

- Theory and practice
- Breadth and depth
- Learning and applying
- Teaching and learning

## 13.9 Final Thoughts

Remember: 1. AI is a rapidly evolving field 2. Focus on fundamentals 3. Practice regularly 4. Share your knowledge 5. Stay curious and experimental

Your journey in AI is just beginning. Keep learning, experimenting, and growing!

## Chapter 14

# Python for Neural Networks

### 14.1 Why Python for Neural Networks?

Python has become the de facto language for machine learning and neural networks, thanks to its: - Rich ecosystem of libraries - Easy-to-read syntax - Extensive community support - Powerful numerical computing capabilities

### 14.2 Essential Python Libraries

#### 14.2.1 1. NumPy

```
import numpy as np

# Create input data
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0]) # XOR function
```

#### 14.2.2 2. TensorFlow/Keras

```
from tensorflow import keras

model = keras.Sequential([
    keras.layers.Dense(4, activation='relu', input_shape=(2,)),
    keras.layers.Dense(1, activation='sigmoid')
])
```

### 14.2.3 3. PyTorch

```
import torch
import torch.nn as nn

class XORNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Linear(2, 4)
        self.layer2 = nn.Linear(4, 1)
```

## 14.3 Getting Started

1. Install Python from [python.org](https://python.org)
2. Set up a virtual environment:

```
python -m venv myenv
source myenv/bin/activate # On Unix
myenv\Scripts\activate   # On Windows
```

3. Install required packages:

```
pip install numpy tensorflow torch scikit-learn
```

## 14.4 From Pharo to Python

### 14.4.1 Key Differences

1. Syntax:

```
"Pharo"
network := NNetwork new.
network configure: 4 hidden: 6 nbOfOutputs: 3.
```

```
# Python
network = NeuralNetwork()
network.configure(4, hidden=6, nb_outputs=3)
```

2. Libraries:

- Pharo: Built-in neural network implementation
- Python: Multiple mature frameworks available

## 14.5 Resources for Learning

1. Online Courses:



- Coursera's Deep Learning Specialization
  - Fast.ai's Practical Deep Learning
  - Google's Machine Learning Crash Course
2. Documentation:
    - [TensorFlow Docs](#)
    - [PyTorch Tutorials](#)
    - [Scikit-learn Guide](#)
  3. Practice Projects:
    - MNIST digit classification
    - Image recognition
    - Natural language processing

## 14.6 Best Practices

1. Code Organization:
  - Use clear variable names
  - Document your code
  - Follow PEP 8 style guide
2. Development Environment:
  - Use Jupyter notebooks for experimentation
  - Version control with Git
  - Regular code backups
3. Performance:
  - Vectorize operations with NumPy
  - Use GPU acceleration when available
  - Profile code for bottlenecks

## 14.7 Next Steps

1. Choose a framework (TensorFlow or PyTorch)
2. Complete online tutorials
3. Build simple projects
4. Join the Python ML community

Remember: The concepts you learned in Pharo translate well to Python - focus on understanding the principles rather than just the syntax.



## Chapter 15

# Finding and Preparing Data for Neural Networks

### 15.1 The Importance of Data

Data is the foundation of any machine learning project. The quality and quantity of your data often matter more than the sophistication of your model.

### 15.2 Popular Data Sources

#### 15.2.1 1. Public Datasets

- [Kaggle](#)
  - Competitions and datasets
  - Active community
  - Detailed documentation
- [UCI Machine Learning Repository](#)
  - Academic datasets
  - Well-documented
  - Quality-controlled
- [Google Dataset Search](#)
  - Comprehensive search engine
  - Various domains
  - Multiple formats

#### 15.2.2 2. Domain-Specific Sources

1. Images
  - ImageNet

- CIFAR-10/100
- MS COCO
- 2. **Text**
  - Wikipedia dumps
  - Project Gutenberg
  - Common Crawl
- 3. **Specialized**
  - Medical: MIMIC
  - Financial: Yahoo Finance
  - Scientific: NASA Earth Data

## 15.3 Data Preparation Steps

### 1. Collection

```
"Example: Download from URL"
data := ZnClient new
  url: 'https://example.com/dataset.csv';
  get
```

### 2. Cleaning

```
"Remove missing values"
cleanData := data reject: [:row |
  row includesAny: #(nil '' 'N/A')
]
```

### 3. Preprocessing

```
"Normalize numerical values"
normalized := data collect: [:value |
  (value - mean) / standardDeviation
]
```

## 15.4 Best Practices

### 15.4.1 1. Data Quality

- Check for missing values
- Remove duplicates
- Handle outliers
- Validate data types

### 15.4.2 2. Data Split

```
"Split into training and testing sets"
splitRatio := 0.8.
```

```
splitIndex := (data size * splitRatio) asInteger.  
trainingSet := data first: splitIndex.  
testingSet := data allButFirst: splitIndex.
```

### 15.4.3 3. Data Augmentation

- Increase dataset size
- Improve model robustness
- Balance classes

## 15.5 Common Challenges

1. **Insufficient Data**
  - Use data augmentation
  - Transfer learning
  - Synthetic data generation
2. **Imbalanced Classes**
  - Oversampling
  - Undersampling
  - SMOTE technique
3. **Noisy Data**
  - Data cleaning
  - Outlier detection
  - Robust preprocessing

## 15.6 Tools and Libraries

1. **Data Processing**
  - Pandas (Python)
  - NumPy (Python)
  - Pharo Data Frame
2. **Visualization**
  - Matplotlib
  - Seaborn
  - Roassal (Pharo)

## 15.7 Next Steps

1. Choose appropriate datasets
2. Implement robust preprocessing
3. Validate data quality
4. Document your process

Remember: Good data preparation is crucial for successful machine learning projects.

## Chapter 16

# Essential Resources and References

### 16.1 Core Learning Resources

#### 16.1.1 Coen's Links

- [Jessy: Het belang van duidelijke AI-prompts](#)
- [Journalists on Hugging Face](#)

#### 16.1.2 Books

##### 1. Neural Networks and Deep Learning

- Author: Michael Nielsen
- [Free Online Book](#)
- Perfect for beginners and intermediate learners
- Clear explanations with interactive examples

##### 2. Deep Learning

- Authors: Ian Goodfellow, Yoshua Bengio, Aaron Courville
- [Available Online](#)
- Comprehensive coverage of deep learning
- Industry standard reference

##### 3. Agile AI in Pharo

- Author: Alexandre Bergel
- Practical implementation in Pharo
- Hands-on examples and exercises
- [Book Link](#)

## 16.2 Video Courses and Tutorials

### 16.2.1 1. Foundational Series

- [3Blue1Brown Neural Networks](#)
  - Visual explanations
  - Mathematical intuition
  - Clear animations

### 16.2.2 2. Programming Tutorials

- [Fast.ai Deep Learning Course](#)
  - Practical approach
  - Top-down learning
  - Real-world applications

### 16.2.3 3. Advanced Topics

- [Stanford CS231n](#)
  - Computer Vision
  - Deep Learning
  - State-of-the-art techniques

## 16.3 Online Platforms

### 16.3.1 1. Interactive Learning

- [Kaggle Learn](#)
  - Hands-on exercises
  - Real datasets
  - Community support

### 16.3.2 2. Research Papers

- [arXiv Machine Learning](#)
  - Latest research
  - Open access
  - Preprint server

### 16.3.3 3. Code Repositories

- [Papers With Code](#)
  - Implementations of papers
  - Benchmarks
  - State-of-the-art tracking



## 16.4 Community Resources

### 16.4.1 1. Forums and Discussion

- [r/MachineLearning](#)
- [Cross Validated](#)
- [AI Stack Exchange](#)

### 16.4.2 2. Blogs and Newsletters

- [Distill.pub](#)
  - Interactive explanations
  - Visual learning
  - Deep insights

### 16.4.3 3. Tools and Libraries

- [TensorFlow](#)
- [PyTorch](#)
- [Scikit-learn](#)

## 16.5 Academic Papers

### 16.5.1 1. Foundational Papers

- “A Logical Calculus of Ideas Immanent in Nervous Activity” (McCulloch & Pitts, 1943)
- “Learning Internal Representations by Error Propagation” (Rumelhart et al., 1986)
- “Gradient-Based Learning Applied to Document Recognition” (LeCun et al., 1998)

### 16.5.2 2. Modern Breakthroughs

- “Deep Residual Learning for Image Recognition” (He et al., 2015)
- “Attention Is All You Need” (Vaswani et al., 2017)
- “Language Models are Few-Shot Learners” (Brown et al., 2020)

