Coen's AI Notes

diverse

2025-03-26

Table of contents

1 Journey into Artificial Intelligence **1**
1.1 Learning Path 1
1.2 Some background how I started 1

2 The AI Landscape: Understanding the Big Picture **3**
2.1 Navigating the World of AI Technologies 3
2.2 Key Concepts to Take Away 3

I Perceptron Fundamentals **5**

3 Understanding the Perceptron **7**
3.1 The Biological Inspiration: From Brain Neurons to Artificial Intelligence 7
3.2 From Biology to Machine: Implementing a Perceptron 8

4 Practical Applications of the Perceptron **9**
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 First Implementation of Perceptron algorithm 10
4.5 Reference 11

5 Decision Making with Perceptrons **13**
5.1 Making Decisions 13
5.2 Understanding the Decision Boundary 13
5.3 Applications 14

6 Teaching a Perceptron: The Learning Process **15**
6.1 Introduction to Perceptron Learning 15
6.2 The Learning Algorithm 15
6.2.1 Mathematical Foundation 15
6.2.2 Training Process 16

6.3 Visualizing the Learning Process 16
6.4 Practical Considerations 16

7 Understanding Perceptron Limitations 17
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

II Neural Networks 23

8 Introduction to Neural Networks 25
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

9 Practical Example: Classifying Iris Flowers 29
9.1 A Real-World Machine Learning Challenge 29
9.2 The Dataset 30
9.3 Key Learning Points 30

10 The Mathematics Behind Neural Networks 31
10.1 Understanding the Magic 31
10.2 The Building Blocks 31
10.2.1 1. Neurons and Weights 31
10.2.2 2. Activation Functions 31
10.3 The Learning Process 32
10.3.1 1. Forward Propagation 32
10.3.2 2. Loss Calculation 32

11 Exploring Neural Network Architectures 33
11.1 The Rich Landscape of Neural Networks 33
11.2 Feedforward Neural Networks (FNN) 33
11.3 Convolutional Neural Networks (CNN) 33
11.4 Recurrent Neural Networks (RNN) 33
11.5 Long Short-Term Memory (LSTM) 34
11.6 Autoencoders 34
11.7 Generative Adversarial Networks (GAN) 34
11.8 Choosing the Right Architecture 34
11.9 Future Directions 34

III Next Steps 35

12 Next Steps in Your AI Journey 37
12.1 Congratulations on Your Progress! . . . 37
12.2 Expanding Your Knowledge . . . 37
12.2.1 1. Advanced Topics . . . 37
12.2.2 2. Practical Skills . . . 37
12.3 Real-World Applications . . . 38
12.3.1 1. Industry Applications . . . 38
12.3.2 2. Research Areas . . . 38
12.4 Building Your Portfolio . . . 38
12.5 Community Engagement . . . 39
12.5.1 1. Online Communities . . . 39
12.5.2 2. Local Groups . . . 39
12.6 Continuous Learning . . . 39
12.6.1 1. Advanced Courses . . . 39
12.6.2 2. Reading Materials . . . 39
12.7 Career Paths . . . 39
12.8 Best Practices Moving Forward . . . 40
12.9 Final Thoughts . . . 40

13 Python for Neural Networks 41
13.1 Why Python for Neural Networks? . . . 41
13.2 Next Steps . . . 41

14 Finding and Preparing Data 43
14.1 The Importance of Data . . . 43
14.2 Popular Data Sources . . . 43
14.2.1 1. Public Datasets . . . 43

15 Essential Resources and References 45
15.1 Core Learning Resources . . . 45
15.1.1 Coen's Links . . . 45
15.1.2 Books . . . 45
15.2 Video Courses and Tutorials . . . 46
15.2.1 1. Foundational Series . . . 46
15.2.2 2. Programming Tutorials . . . 46
15.2.3 3. Advanced Topics . . . 46
15.3 Online Platforms . . . 46
15.3.1 1. Interactive Learning . . . 46
15.3.2 2. Research Papers . . . 47
15.3.3 3. Code Repositories . . . 47
15.4 Community Resources . . . 47
15.4.1 1. Forums and Discussion . . . 47
15.4.2 2. Blogs and Newsletters . . . 47
15.4.3 3. Tools and Libraries . . . 47

15.5 Academic Papers 47
15.5.1 1. Foundational Papers 47
15.5.2 2. Modern Breakthroughs 48

Chapter 1

Journey into Artificial Intelligence

Welcome! This is a Work-in-Progress, a collection of notes on AI I collected and use in my workshops.

A newer version of this pdf may be found here

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 how I started

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 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.

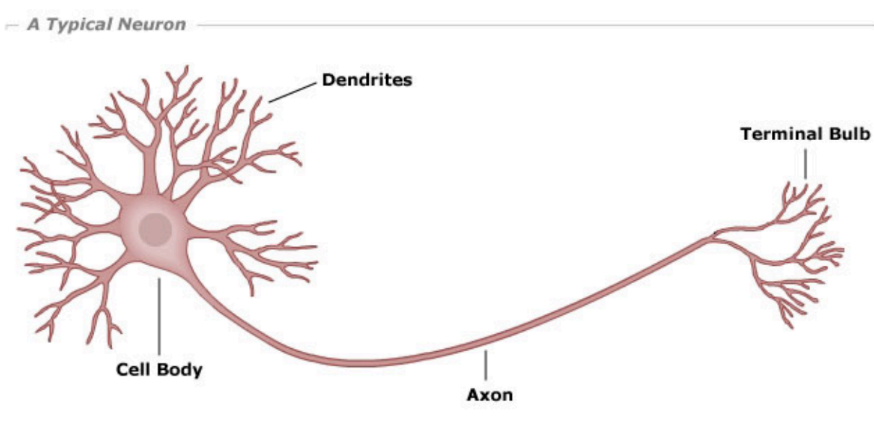

Figure 3.1: A typical biological neuron structure

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.

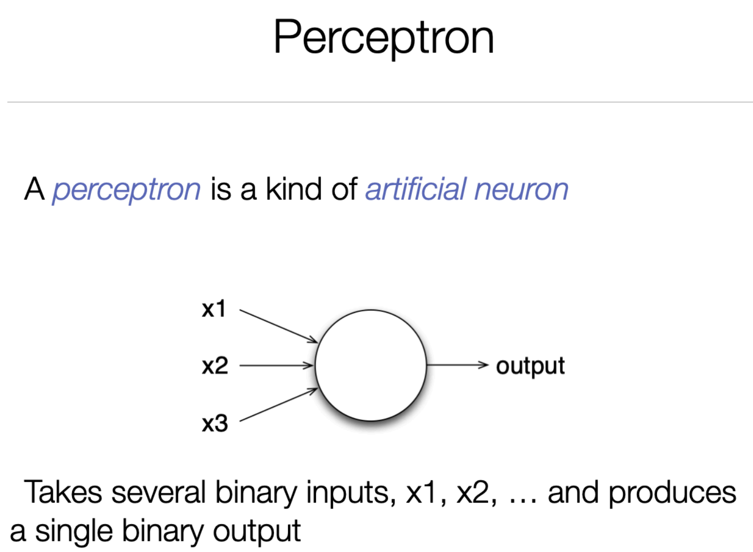

Figure 3.2: Perceptron's architectural diagram

Let's explore a practical example with three inputs. We'll call our input values `x1`, `x2`, and `x3`, with their corresponding weights `w1`, `w2`, and `w3`. The Perceptron processes these inputs in two steps:

1. First, it calculates a `weighted sum` and adds the bias: `z := w1*x1 + w2*x2 + w3*x3 + bias`
2. Then, it applies what we call an **activation function** to produce the final output: let's use a very simpel 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: `w1 = 1`, `w2 = 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 prefered 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 First Implementation of Perceptron algorithm

According to Wikipedia:

> The artificial neuron network was invented in 1943 by Warren McCulloch and Walter Pitts in 'A logical calculus of the ideas immanent in nervous activity'. the Perceptron Machine was first implemented in hardware in the Mark I, which was demonstrated in 1960.
>
> It was connected to a camera with 20×20 cadmium sulfide photocells to make a 400-pixel image. The main visible feature is the sensory-

to-association plugboard, which sets different combinations of input features. To the right are arrays of potentiometers that implemented the adaptive weights.

4.5 Reference

- wikipedia: perceptron

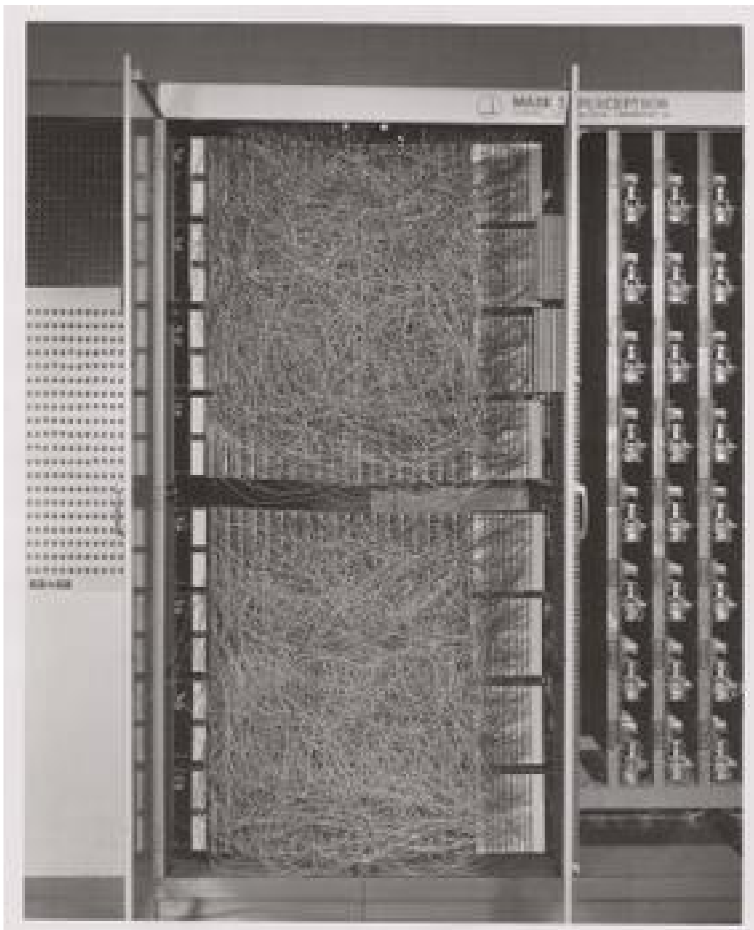

Figure 4.1: The Mark I Perceptron machine, the first implementation of the perceptron algorithm (source: wikipedia)

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 `x1 = 1` and `x2 = 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

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

6.2.2 Training Process

To train the Perceptron, we have to have `labeled data` (ie. input data combined with the desired output for those values)

So for training AND gate behavior we have to list all combinations of 2 bits that are possible as input, and also the desired output value:

```
| Input  | Desired Output |
|--------|----------------|
| (0, 0) | 0              |
| (0, 1) | 0              |
| (1, 0) | 0              |
| (1, 1) | 1              |
```

and training (1 epoc) means calling the train function with each of these examples:

```
foreach dataItem in trainingData do:
    inputs := dataItem[0]
    desiredOutput := dataItem[1]
    learningPerceptron train(inputs, desiredOutput)
```

6.3 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

6.4 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) $\rightarrow$ Output: 0 - Input (0,1) $\rightarrow$ Output: 1 - Input (1,0) $\rightarrow$ Output: 1 - Input (1,1) $\rightarrow$ Output: 0

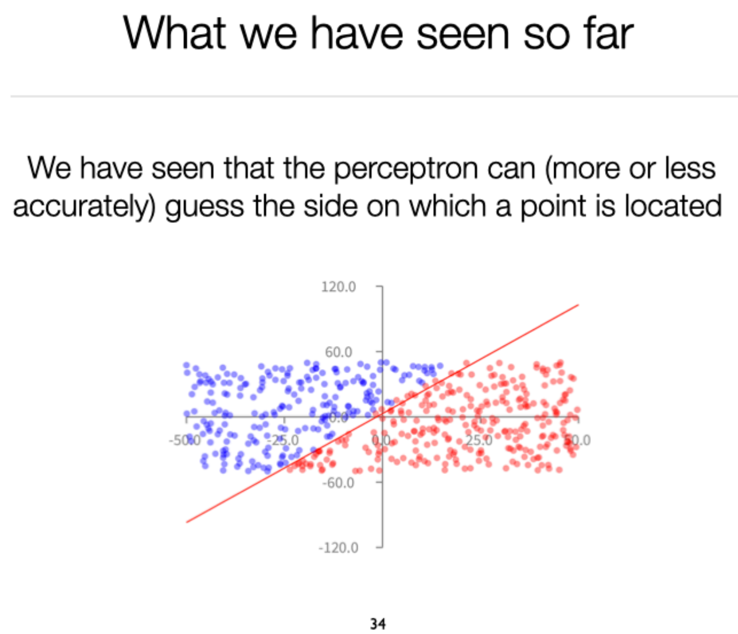

Figure 7.1: Visual representation of XOR problem

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.

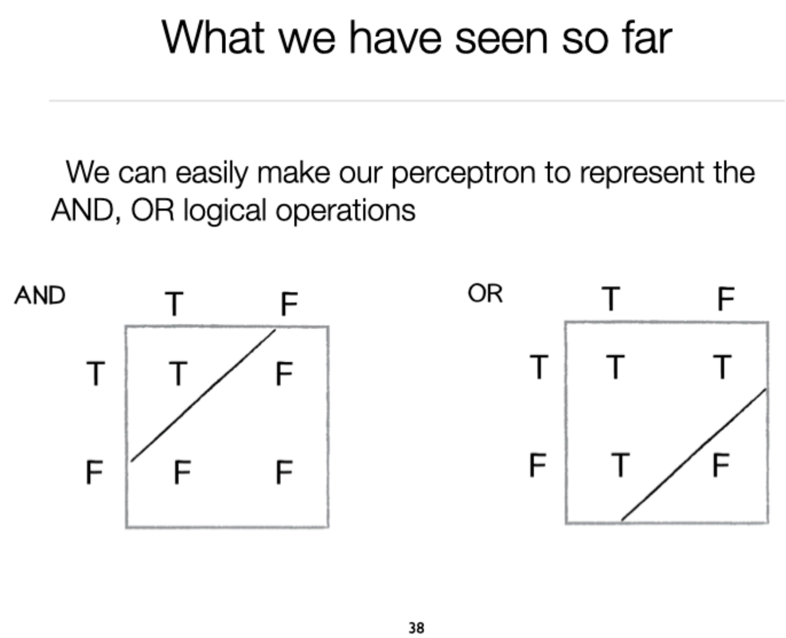

Figure 7.2: Attempted linear separation of XOR

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!

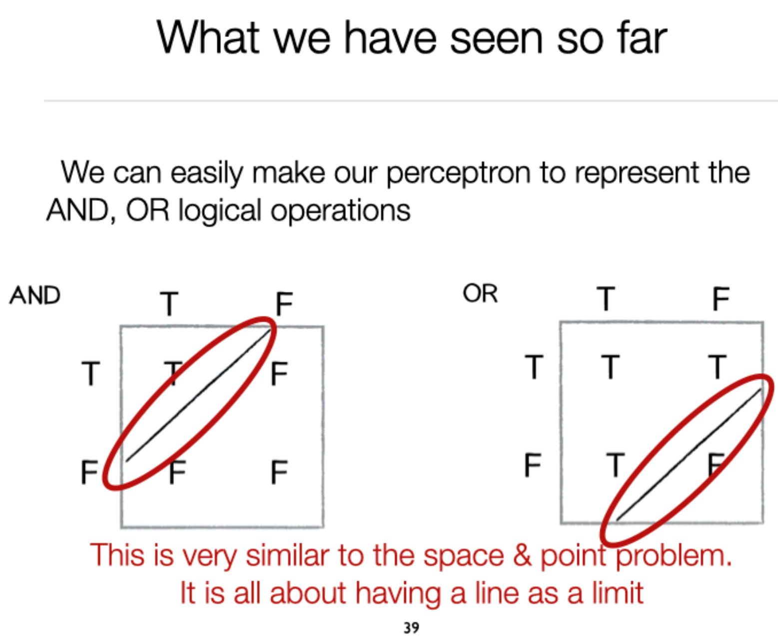

Figure 7.3: Multi-layer solution

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

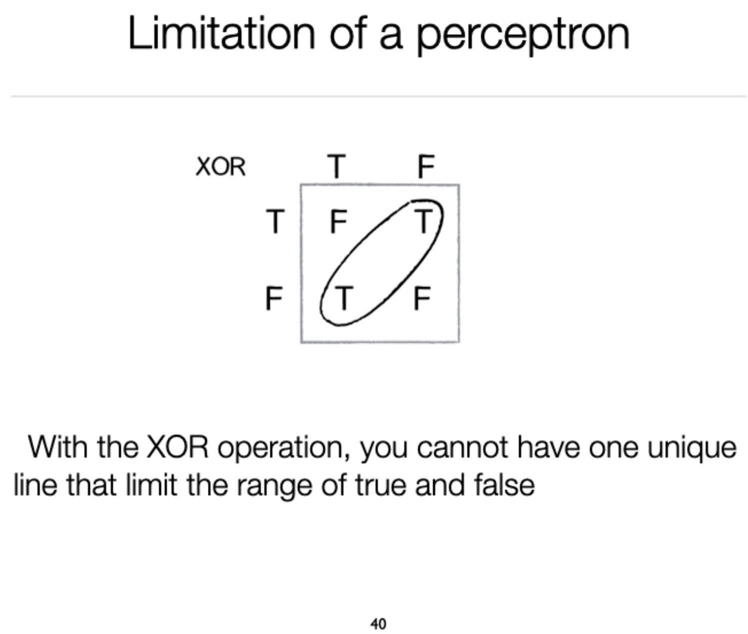

Figure 7.4: Complete neural network solution

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.

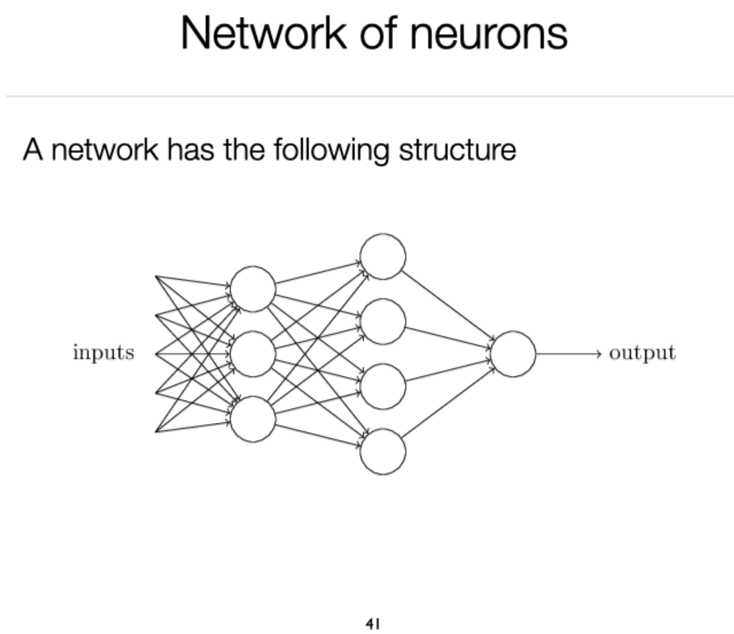

Figure 8.1: Basic neural network architecture

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.

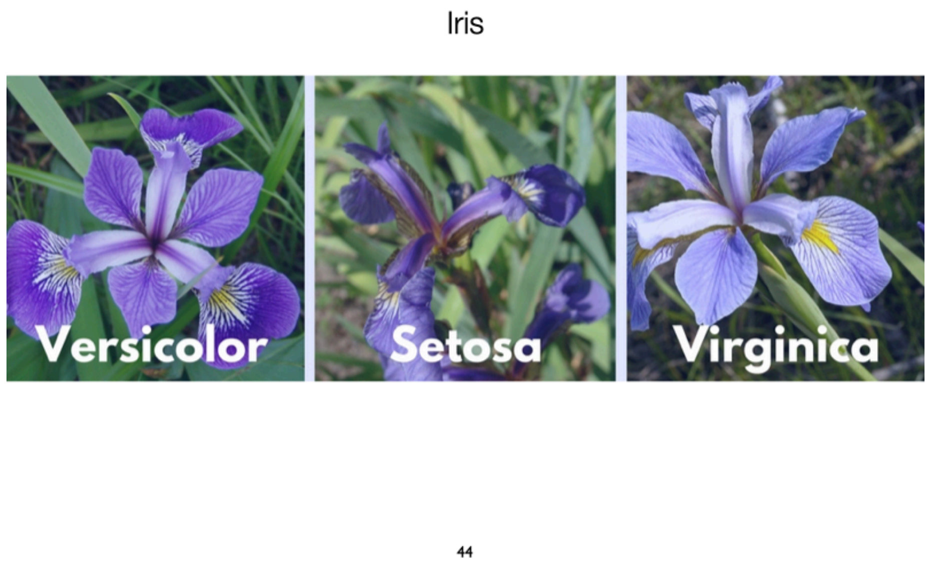

Figure 9.1: Different types of Iris flowers

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

Building a network that can do this is really outside of the scope of these notes, but a lot of info can be found on the internet on `Iris Classification`.

9.3 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 (a bit of) how they actually work under the hood.

10.2 The Building Blocks

10.2.1 1. Neurons and Weights

To be formally correct we should say `artificial neuron` to distinguish them from `biological neurons` like we have in our brain. A neuron normally has inputs: 1, or 2, or $\cdots$

Each neuron performs two key operations: 1. Weighted sum of inputs. 2. Activation function: $a = f(z)$

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.

10.3.2 2. Loss Calculation

Measure the network's error and Backpropagation

- what is the output?
- What would be my desired output?

The smaller the difference between the output I got and the output I desired, the better the output of my model is. This difference is calculated with a so-called Loss function. Backpropagation is an algorithm that helps make that difference small. When backpropagation is performed we call that Training the AI model.

Chapter 11

Exploring Neural Network Architectures

11.1 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.

11.2 Feedforward Neural Networks (FNN)

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

11.3 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

11.4 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

11.5 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

11.6 Autoencoders

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

11.7 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

11.8 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

11.9 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 12

Next Steps in Your AI Journey

12.1 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.

12.2 Expanding Your Knowledge

12.2.1 1. Advanced Topics

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

12.2.2 2. Practical Skills

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

12.3 Real-World Applications

12.3.1 1. Industry Applications

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

12.3.2 2. Research Areas

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

12.4 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

12.5 Community Engagement

12.5.1 1. Online Communities

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

12.5.2 2. Local Groups

- Meetups
- Hackathons
- Workshops
- Study groups

12.6 Continuous Learning

12.6.1 1. Advanced Courses

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

12.6.2 2. Reading Materials

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

12.7 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

12.8 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

12.9 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 13

Python for Neural Networks

13.1 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

13.2 Next Steps

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

Chapter 14

Finding and Preparing Data

14.1 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.

14.2 Popular Data Sources

14.2.1 1. Public Datasets

- Kaggle
  - Competitions and datasets
  - Active community
  - Detailed documentation
- Eindhoven open data
  - lots of data about Eindhoven

Chapter 15

Essential Resources and References

15.1 Core Learning Resources

15.1.1 Coen's Links

- Jessy: Het belang van duidelijke AI-prompts
- Journalists on Hugging Face

Perplexity is often a great start for finding things (with references): https://www.perplexity.ai/ To understand about Transformers this is a very nice start: https://ig.ft.com/generative-ai/ 'Our own' page about (Gen)AI: https://stasemsoft.github.io/FontysICT-sem1/docs/artificial-intelligence/ai.html To dive further into how Transformers works: https://www.deeplearning.ai/short-courses/how-transformer-llms-work/ and also to other short courses on deeplearning.ai The development I showed was https://www.cursor.com/ you have like only 500 requests for free… after that you could choose to pay 20 euro a Month (yes, that can be a lot for students, I know), or look for alternatives, 2 of which I tried a bit (you can use local LLM's with them, which basically makes them free): AIDER: https://aider.chat/.
Avante: https://github.com/yetone/avante.nvim (but then you need to learn about 'vi': https://neovim.io/ which is a hurdle).

https://www.prompthub.us/blog/how-polite-should-we-be-when-prompting-llms

15.1.2 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

15.2 Video Courses and Tutorials

15.2.1 1. Foundational Series

- 3Blue1Brown Neural Networks
  - Visual explanations
  - Mathematical intuition
  - Clear animations

15.2.2 2. Programming Tutorials

- Fast.ai Deep Learning Course
  - Practical approach
  - Top-down learning
  - Real-world applications

15.2.3 3. Advanced Topics

- Stanford CS231n
  - Computer Vision
  - Deep Learning
  - State-of-the-art techniques

15.3 Online Platforms

15.3.1 1. Interactive Learning

- Kaggle Learn
  - Hands-on exercises
  - Real datasets
  - Community support

15.3.2 2. Research Papers

- arXiv Machine Learning
  - Latest research
  - Open access
  - Preprint server

15.3.3 3. Code Repositories

- Papers With Code
  - Implementations of papers
  - Benchmarks
  - State-of-the-art tracking

15.4 Community Resources

15.4.1 1. Forums and Discussion

- r/MachineLearning
- Cross Validated
- AI Stack Exchange

15.4.2 2. Blogs and Newsletters

- Distill.pub
  - Interactive explanations
  - Visual learning
  - Deep insights

15.4.3 3. Tools and Libraries

- TensorFlow
- PyTorch
- Scikit-learn

15.5 Academic Papers

15.5.1 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)

15.5.2 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)