

Demystifying Artificial Intelligence Sorcery

(Part 3: Deep Learning)^a

Abdelbacet Mhamdi
abdelbacet.mhamdi@bizerte.r-iset.tn

Dr.-Ing. in Electrical Engineering
Senior Lecturer at ISET Bizerte

^aAvailable @ <https://github.com/a-mhamdi/jlai/>



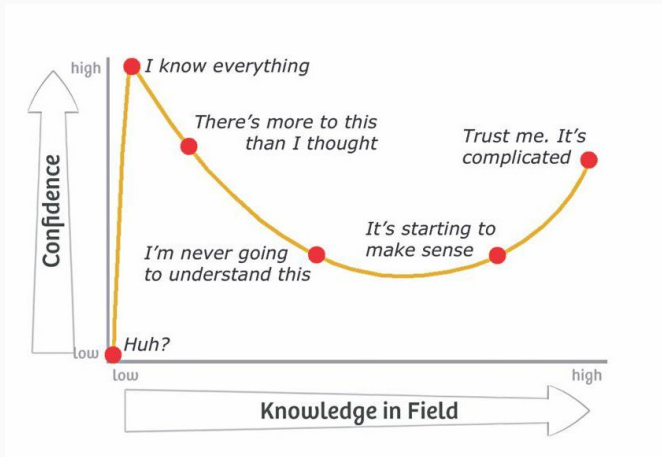
Disclaimer

This document features some materials gathered from multiple online sources.

Please note no copyright infringement is intended, and I do not own nor claim to own any of the original materials. They are used for educational purposes only.

I have included links solely as a convenience to the reader. Some links within these slides may lead to other websites, including those operated and maintained by third parties. The presence of such a link does not imply a responsibility for the linked site or an endorsement of the linked site, its operator, or its contents.

DUNNING-KRUGER EFFECT

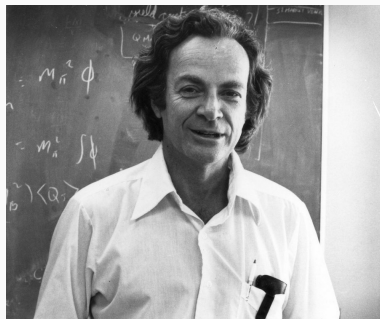


Kruger, J. and Dunning, D. (1999) *Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments*. **J Pers Soc Psychol**. 77(6) pp. 1121–1134.

 10.1037/0022-3514.77.6.1121

“Knowledge isn’t free. You have to pay attention.”

Richard P. Feynman




1. Natural Language Processing
2. Computer Vision
3. Generative AI
4. Reinforcement Learning
5. Quizzes

REMINDER



julialang.org/

A screenshot of a terminal window titled "~julia/julia". The prompt is "+ julia". The user has entered "julia" and the REPL has responded with a stylized ASCII art logo of the word "julia" in green. To the right of the logo, the text "Documentation: https://docs.julialang.org" is displayed. Below the logo, the text "Type '?' for help, ']' for pkg help." and "Version 1.6.3 (2021-09-23)" are shown. The user then enters "julia> println(\"Hello, World!\")" and the REPL outputs "Hello, World!". The prompt "julia>" is shown again.

DEVELOPMENT ENVIRONMENTS



Pluto.jl



▲ \$ docker compose up

▼ \$ docker compose down



- ▲ **Fast:** native code for multiple platforms via LLVM;
- ▲ **Dynamic:** good support for interactive use (*like a scripting language*);
- ▲ **Reproducible:** environment recreation across platforms, with pre-built binaries;
- ▲ **Composable:** multiple dispatch as a paradigm (*oop & functional programming*);
- ▲ **General:** asynchronous I/O, metaprogramming, debugging, logging; profiling, pkg, ...
- ▲ **Open Source:** GitHub repository at <https://github.com/JuliaLang/julia>.



JULIA MICRO-BENCHMARKS (1/2)



<https://julialang.org/benchmarks>



JULIA MICRO-BENCHMARKS (2/2)

Geometric Means¹ of Micro-Benchmarks by Language

1	C	1.0
2	Julia	1.17006
3	LuaJIT	1.02931
4	Rust	1.0999
5	Go	1.49917
6	Fortran	1.67022
7	Java	3.46773
8	JavaScript	4.79602
9	Matlab	9.57235
10	Mathematica	14.6387
11	Python	16.9262
12	R	48.5796
13	Octave	338.704



¹Measure of central tendency expressed as $(x_1 \times x_2 \times \dots \times x_n)^{1/n}$

SOURCE CONTROL MANAGEMENT (SCM)



The screenshot shows the GitHub repository page for 'a-mhamdi/jlai'. The repository is public and has 2 stars and 3 forks. The main branch is 'main'. The repository contains a file tree with the following files and folders:

- .github/workflows: Update docker-image.yml (2 weeks ago)
- Codes: vgg and resnet transfer learning (yesterday)
- Docker: rm Docker cheat sheet (3 days ago)
- Exams: exam w/ answers (4 days ago)
- Slides-Labs: change colors (yesterday)
- .gitignore: change colors (yesterday)
- LICENSE: Initial commit (4 months ago)
- README.md: update Docker README file (2 weeks ago)

The README.md file is selected, showing the title 'Fuzzy Logic, Machine Learning and Deep Learning with Julia'. The 'About' section on the right provides more details about the repository, including the title 'An Introduction to Artificial Intelligence with Julia', the license 'MIT license', and the number of stars (2) and forks (3). The 'Languages' section shows the following language distribution:

Language	Percentage
Julia	94.3%
Dockerfile	3.4%
Batchfile	2.1%
TeX	0.2%

<https://github.com/a-mhamdi/jlai>



The screenshot shows a web browser window displaying the Docker Hub page for the image `abmhamdi/jlai-p3`. The page has a dark theme. At the top, the browser tab is titled "abmhamdi/jlai-p3 - Docker Image | Docker Hub - Brave". The address bar shows the URL `hub.docker.com/r/abmhamdi/jlai-p3`. The Docker Hub navigation bar is visible with links for Explore, Repositories, Organizations, and Usage. A search bar and various utility icons are also present. The main content area features the image icon (a cube with three vertical bars), the name `abmhamdi/jlai-p3`, and the text "By [abmhamdi](#) · Updated 1 minute ago". Below this, it says "Artificial Intelligence Labs - Part 3 @ ISETBZ" and "IMAGE". There are also tags for "DATA SCIENCE", "LANGUAGES & FRAMEWORKS", and "MACHINE LEARNING & AI". The page shows 0 stars and 11 downloads. A "Manage Repository" button is in the top right. At the bottom, there are two sections: "Overview" and "Tags". The "Overview" section has a heading "Deep Learning with Julia" and a description: "This repository contains slides, labs and code examples for using Julia to implement some artificial intelligence related algorithms. Codes run on top of a Docker image, ensuring a consistent and reproducible environment." The "Tags" section shows the "Docker Pull Command" as `docker pull abmhamdi/jlai-p3` with a "Copy" button.

<https://hub.docker.com/r/abmhamdi/jlai-p3>

GENERAL OVERVIEW

NLP is important for tasks such as language translation, text classification, and language generation because it allows computers to process and understand human language.

CNNs are used for image classification and other computer vision tasks because they are able to automatically learn features from raw data. This is useful for tasks where manual feature engineering is difficult or impractical.

Transfer Learning allows models pre-trained on large datasets to be fine-tuned for specific tasks with limited data. This is valuable for domains where labeled data is scarce or expensive to obtain, enabling faster training and better performance.

GANs are used for tasks such as image generation and data augmentation because they are able to generate new data samples that are similar to a given dataset.

VAEs are used for tasks such as image generation and anomaly detection because they are able to learn a compact representation of a dataset and generate new samples from this representation.

Reinforcement Learning is used for sequential decision-making tasks such as game playing, robotics, and autonomous systems because it enables agents to learn optimal behaviors through trial and error by maximizing cumulative rewards from interactions with an environment.

Natural Language Processing

PURPOSE

- ▶ **Natural Language Processing (NLP)** is a field at the intersection of artificial intelligence, computer science, and linguistics that enables computers to understand, interpret, and generate human language.
- ▶ **NLP** encompasses the development of algorithms, statistical models, and neural networks that process both written and spoken language, capturing syntax, semantics, and pragmatic meaning.
- ▶ **NLP** powers diverse applications including machine translation, question answering, text summarization, sentiment analysis, named entity recognition, chatbots, and large language models like those used in conversational AI systems.

USE CASES

Part-of-speech tagging

Named entity recognition

Sentiment analysis

Machine translation

Text summarization

Part-of-speech tagging Identifying the parts of speech (*e.g., noun, verb, adjective*) in a sentence

“Apple released a new iPhone in California”

Apple (*noun*), released (*verb*), a (*determiner*), new (*adjective*),
iPhone (*noun*), in (*preposition*), California (*noun*)

Named entity recognition Identifying and labeling named entities (*e.g., people, organizations, locations*) in a text

“Apple released a new iPhone in California”

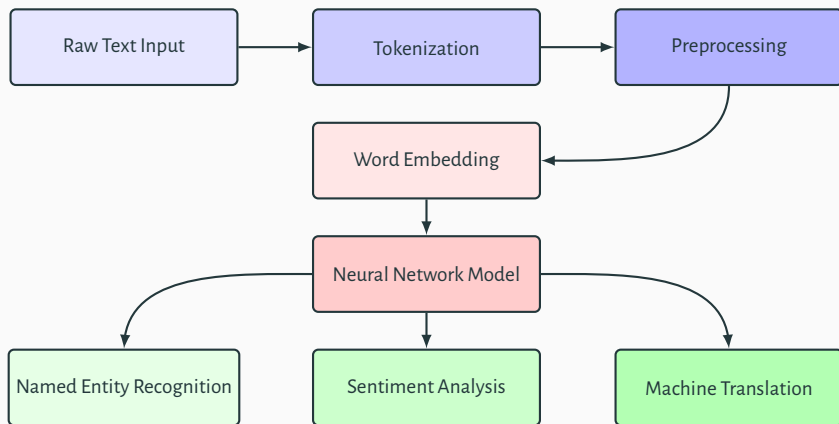
Apple (*Organization*), iPhone (*Product*), California (*Location*)

Sentiment analysis Determining the sentiment (*e.g., positive, neutral, negative*) of a piece of text

Machine translation Translating text from one language to another

Text summarization Generating a concise summary of a longer piece of text

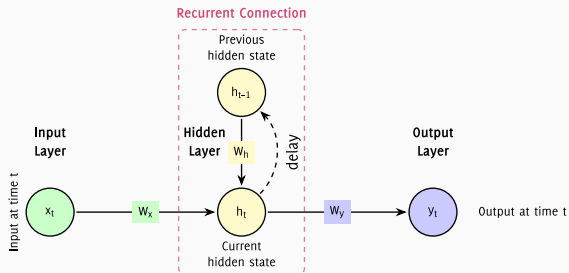
PIPELINE



GENERAL PROCESS IN JULIA

1. Preprocess the text data by tokenizing into words or subwords, optionally normalizing text (*e.g.*, *lowercasing*, *removing special characters*), and handling language-specific features.
2. Build a vocabulary from the most frequent tokens in the training data, including special tokens (*e.g.*, *[PAD]*, *[UNK]*, *[CLS]*, *[SEP]*) for model requirements.
3. Encode the text data as sequences of integer indices using the vocabulary, mapping out-of-vocabulary tokens to a designated unknown token.
4. Pad or truncate sequences to a uniform length to create batches suitable for efficient model training and inference.
5. Define the **NLP** model architecture (*e.g.*, *RNN*, *LSTM*, *Transformer*) using a deep learning library such as `Flux.jl` or `Knet.jl`.
6. Train the model using an optimization algorithm (*e.g.*, *Adam*, *SGD*) with an appropriate loss function (*e.g.*, *cross-entropy for classification*, *perplexity for language modeling*).
7. Evaluate the trained model on validation data, tune hyperparameters as needed, and use it to make predictions on new, unseen data.

RNN


Legend:

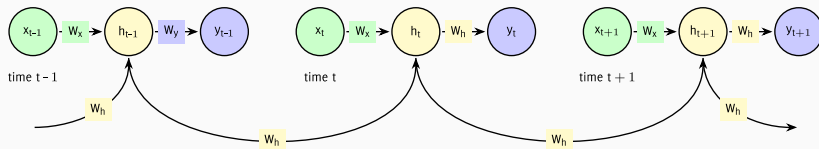
- Input Nodes
- Hidden State
- Output Nodes

RNN Equations:

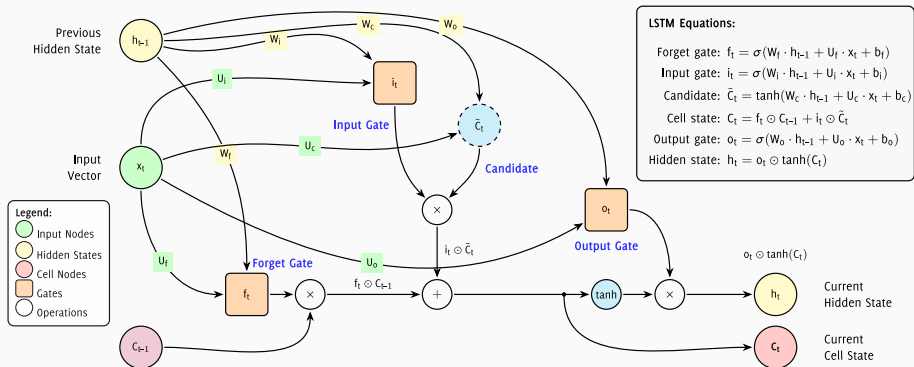
Hidden state: $h_t = f(W_x \cdot x_t + W_h \cdot h_{t-1} + b_h)$

Output: $y_t = f(W_y \cdot h_t + b_y)$

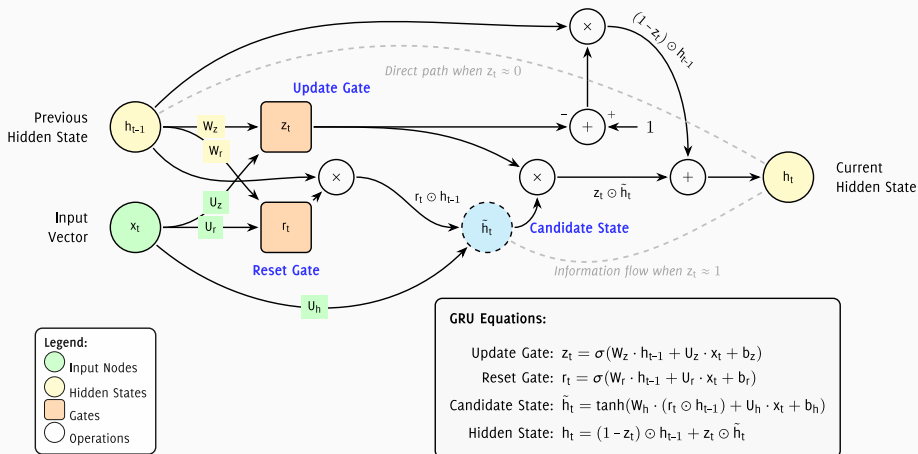
- *tanh* or *ReLU* for hidden layer,
- *softmax/sigmoid* for output

Unfolded RNN (Through Time)


LSTM



GRU





The code is available @ github.com/a-mhamdi/jlai → *Codes* → *Julia* → *Part-3*

→ *nlp* → *nlp.jl*

Pluto.jl 

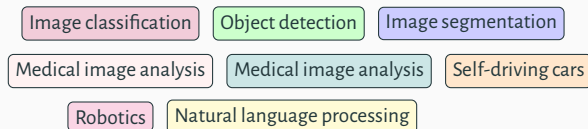
→ *nlp* → *nlp.ipynb*



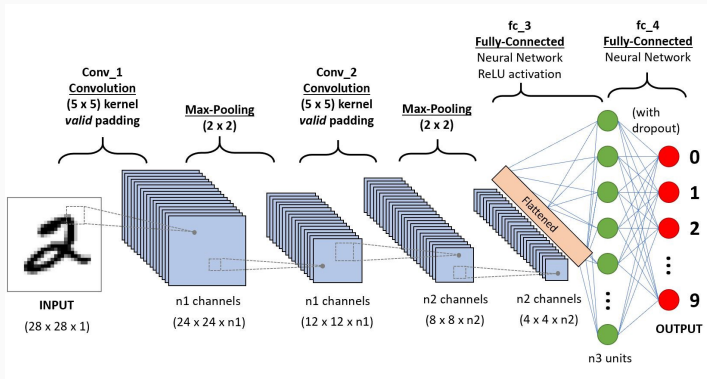
Computer Vision

MOTIVATING FACTORS & USE CASES

- ▶ A **Convolutional Neural Network (CNN)** is a type of neural network that is particularly well-suited for image classification and object recognition tasks. It is designed to process data with a grid-like topology, such as an image.
- ▶ **CNNs** are composed of several types of layers, including convolutional layers, pooling layers, and fully connected layers:
 - ❶ The **convolutional layers** apply filters to the input data, which are used to detect patterns and features in the data.
 - ❷ The **pooling layers** reduce the spatial dimensions of the data, which helps to reduce the complexity of the model and make it more robust to small translations of the input data.
 - ❸ The **fully connected layers** combine the features learned by the convolutional and pooling layers to make a prediction.



ARCHITECTURE



► Source

DIMENSIONALITY OPERATIONS AND TECHNIQUES

- ▶ **Input Channels:** Number of channels in input (e.g., 3 for RGB, 1 for grayscale)
- ▶ **Output Channels:** Number of filters/kernels applied; determines feature map depth
- ▶ **Feature Maps:** Output of convolutional layers
- ▶ **Dropout:** Randomly deactivate neurons during training to prevent overfitting
- ▶ **Batch Normalization:** Normalize layer inputs across mini-batch
- ▶ **Padding:** Adds zeros around input borders
- ▶ **Stride:** Step size of filter movement
- ▶ **Pooling:** Downsample spatial dimensions (Max/Average pooling)
- ▶ **Flatten:** Convert multi-dimensional feature maps to 1-D vector for fully connected layers

PIPELINE

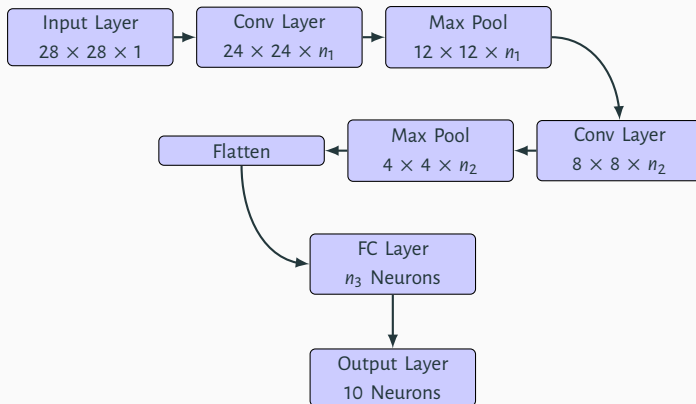


IMAGE KERNELS

By [Victor Powell](#)

An image kernel is a small matrix used to apply effects like the ones you might find in Photoshop or Gimp, such as blurring, sharpening, outlining or embossing. They're also used in machine learning for 'feature extraction', a technique for determining the most important portions of an image. In this context the process is referred to more generally as "convolution" (see: [convolutional neural networks](#).)

To see how they work, let's start by inspecting a black and white image. The matrix on the left contains numbers, between 0 and 255, which each correspond to the brightness of one pixel in a picture of a face. The large, granulated picture has been blown up to make it easier to see; the last image is the "real" size.

<https://setosa.io/ev/image-kernels/>

WHAT IS PADDING

- ▶ involves adding extra pixels around the border of an image;
- ▶ prevents the shrinking of the input image;
- ▶ preserves information on the border.

$$\text{output_shape} = \left\lceil \frac{\text{input_shape} + 2 \times \overbrace{\text{padding}}^p - \overbrace{\text{filter_size}}^k}{\underbrace{\text{stride}}_s} \right\rceil + 1$$

Let's consider $s = 1$, which means that the filter moves one pixel at a time:

valid: ($p = 0$) no padding at all

$$\text{output_shape} = \text{input_shape} - k + 1$$

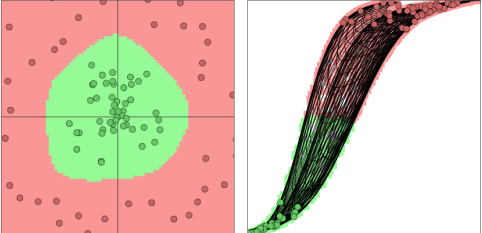
same: $\left(p = \frac{(k-1)}{2} \text{ \& } k \text{ is odd} \right)$ the output is the same dimension as the input

$$\text{output_shape} = \text{input_shape}$$

CONVNETJS DEMO

Feel free to change this, the text area above gets eval()'d when you hit the button and the network gets reloaded. Every 10th of a second, all points are fed to the network multiple times through the trainer class to train the network. The resulting predictions of the network are then "painted" under the data points to show you the generalization.

On the right we visualize the transformed representation of all grid points in the original space and the data, for a given layer and only for 2 neurons at a time. The number in the bracket shows the total number of neurons at that level of representation. If the number is more than 2, you will only see the two visualized but you can cycle through all of them with the cycle button.



simple data circle data spiral data
random data

Controls:
CLICK: Add red data point
SHIFT+CLICK: Add green data point
CTRL+CLICK: Remove closest data point

Go [back to ConvNetJS](https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html)

drawing neurons 0 and 1 of layer with index 4 (tanh)

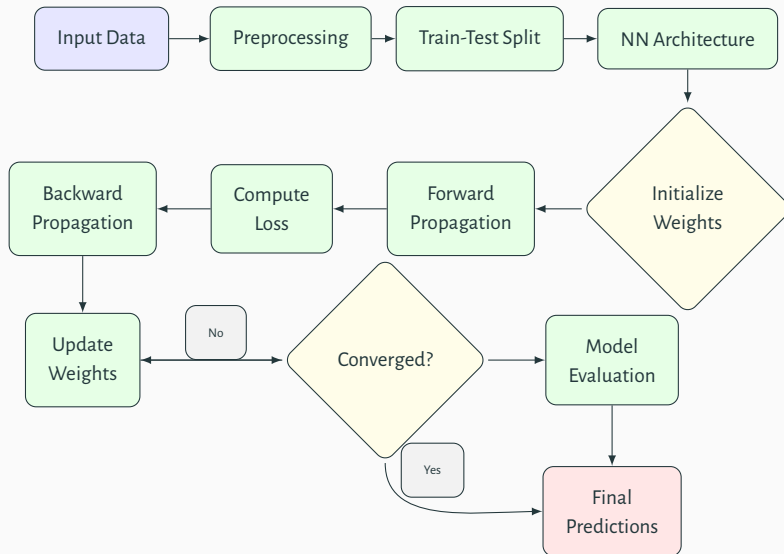
fc(6) tanh(6) fc(2) **tanh(2)**

fc(2)

cycle through visualized neurons at selected layer (if more than 2)

<https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>

NEURAL NETWORK LEARNING ALGORITHM



CNN EXPLAINER



<https://poloclub.github.io/cnn-explainer/>



The code is available @ github.com/a-mhamdi/jlai → *Codes* → *Julia* → *Part-3*

→ *cnn* → *cnn.jl*

Pluto.jl 

→ *cnn* → *cnn.ipynb*



DRIVING FORCES & USE CASES

Transfer Learning is a machine learning technique where a model trained on one task is adapted for a second, related task. It leverages knowledge from the source task to improve performance, reduce training time, and decrease data requirements for the target task.

Common approach:

Fine-tuning pre-trained models on new datasets

- ▶ Example: A model pre-trained on ImageNet (1M+ images, 1000 classes) can be fine-tuned for specialized tasks like medical image diagnosis or autonomous vehicle perception
- ▶ Typical strategy: Freeze early layers (*general feature extractors*), retrain later layers (*task-specific*)
- ▶ Results in better performance than training from scratch, especially with limited data

Key advantages:

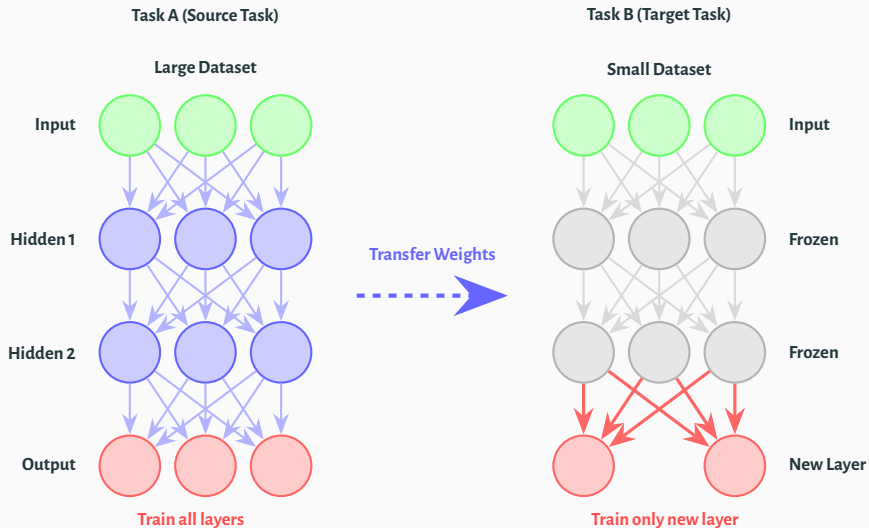
- ▶ Reduces training time and computational costs significantly
- ▶ Enables learning with smaller datasets (*hundreds vs. millions of examples*)
- ▶ Captures generalizable features (*edges, textures, shapes*) from large-scale data
- ▶ Widely used in computer vision (YOLO, ResNet) and **NLP** (BERT, GPT)

Image classification

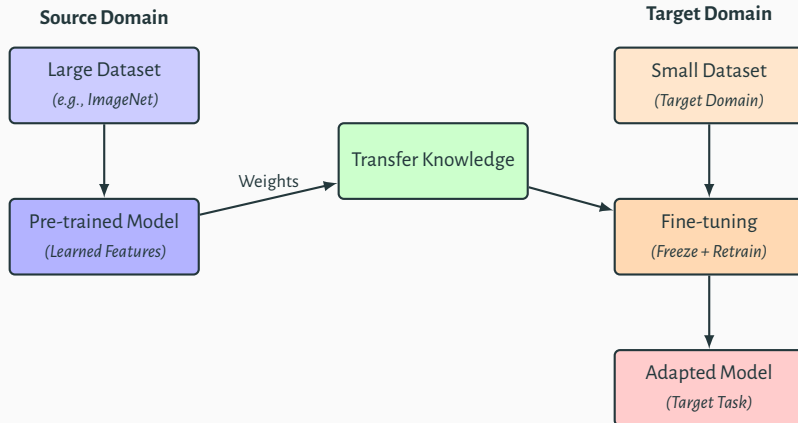
Computer vision

Natural language processing

Robotics



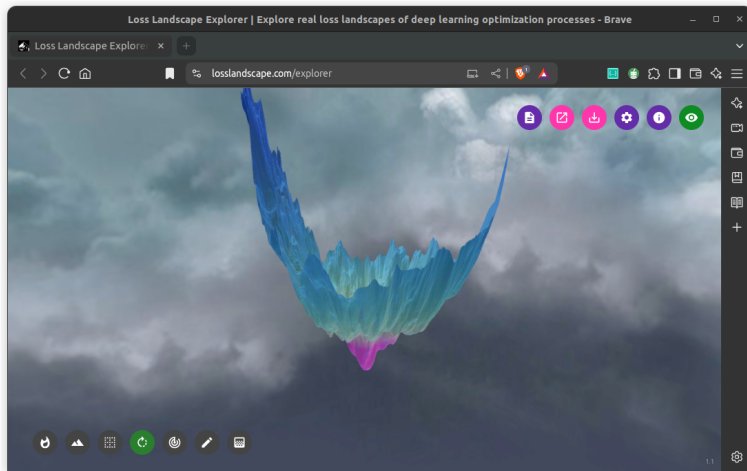
PIPELINE



CLASSIC CNN ARCHITECTURES FOR TRANSFER LEARNING

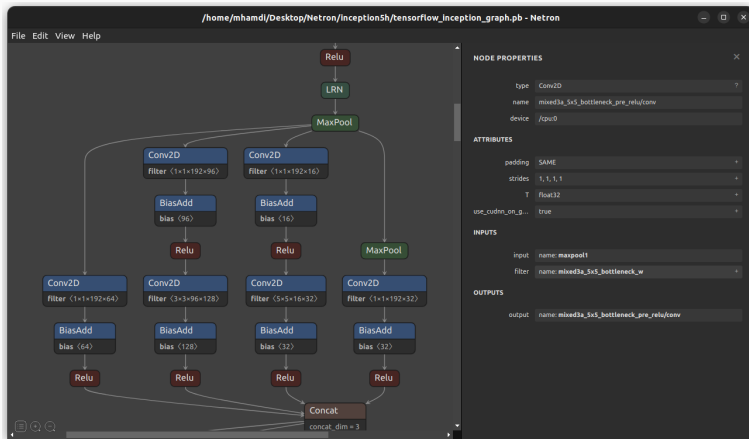
- ▶ **AlexNet (2012)** - ImageNet winner
 - 8 layers (5 conv + 3 FC), 60M parameters
 - Introduced ReLU, dropout, data augmentation
 - First successful deep CNN on ImageNet
- ▶ **VGGNet (2014)** - VGG16/VGG19
 - 16-19 layers, 138M parameters
 - Simple architecture: 3×3 conv filters throughout
 - Demonstrated that depth improves performance
- ▶ **GoogLeNet/Inception (2014)**
 - 22 layers, 6.8M parameters (efficient!)
 - Inception modules: parallel conv operations
 - Won ImageNet 2014
- ▶ **ResNet (2015)** - Revolutionary
 - 50-152 layers, skip connections (*residual learning*)
 - Solved vanishing gradient problem
 - Most widely used for transfer learning
- ▶ **Modern variants:** EfficientNet, MobileNet, DenseNet

WHY SKIP CONNECTION



<https://losslandscape.com/explorer>

NETRON



<https://github.com/lutzroeder/netron>

GENERAL PROCESS IN JULIA

1. Load the pre-trained model (*e.g., a convolutional neural network trained on ImageNet*).
2. Replace the final layer (or layers) of the pre-trained model with a new, untrained layer (or layers) that is suitable for your target task.
3. Freeze the weights of the pre-trained layers to prevent them from being updated during training.
4. Load your dataset and split it into training and validation sets.
5. Use the training set to fine-tune the weights of the new layer (or layers) using gradient descent and a suitable loss function.
6. Monitor the performance of the model on the validation set and adjust the hyperparameters (*e.g., learning rate*) as needed.
7. When you're satisfied with the performance of the model on the validation set, you can use it to make predictions on the test set or on new data.

OBJECT DETECTION: EVOLUTION TIMELINE

► Classical Era (2001-2013)

- Viola-Jones (2001): Face detection with Haar cascades
- HOG (*Histogram of Oriented Gradients*) + SVM (2005): Pedestrian detection
- DPM (2008): Deformable Part Models
- Selective Search (2013): Region proposals

► Two-Stage Detectors (2014-2015)

- R-CNN, Fast R-CNN, Faster R-CNN: Accurate but slow

► One-Stage Era: YOLO (2015-2024)

- YOLOv1-v3 (2015-2018): Real-time detection
- YOLOv4-v8 (2020-2023): Current state-of-the-art
- [Still dominant for real-time applications](#)

► Modern Era (2020-present)

- DETR (2020): Transformer-based detection
- Vision Transformers: ViT-based detectors
- Foundation Models: SAM (*Segment Anything Model*), DINO, Grounding DINO



The code is available @ github.com/a-mhamdi/jlai → Codes → Julia → Part-3

→ *transfer-learning* → *transfer-learning.jl*

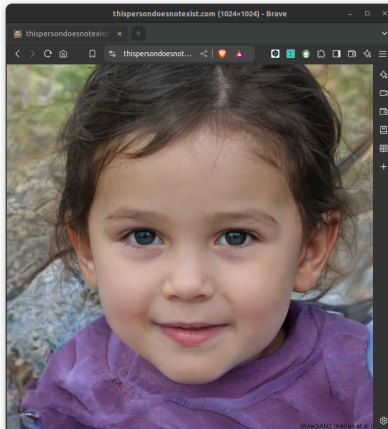
Pluto.jl

→ *transfer-learning* → *transfer-learning.ipynb*

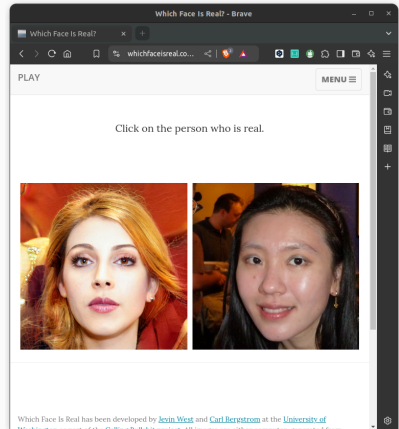


Generative AI

DEMOS



<https://thispersondoesnotexist.com/>



<https://www.whichfaceisreal.com>

AN OVERVIEW

A **Generative Adversarial Network (GAN)** is a deep learning framework for generating realistic synthetic data through adversarial training. It consists of two competing neural networks:

Generator G: Creates synthetic samples from random noise

Discriminator D: Classifies samples as real or fake

Training process (*adversarial game*):

- ▶ **Generator's objective:** Produce samples that fool the discriminator

$$\max_G \ln D(G(z))$$

- ▶ **Discriminator's objective:** Correctly distinguish real from fake samples

$$\max_D [\ln D(x) + \ln(1 - D(G(z)))]$$

- ▶ Networks trained alternately: *D* learns to detect fakes, *G* learns to create better fakes
- ▶ Training continues until *G* produces samples indistinguishable from real data
- ▶ Unsupervised learning approach (*no labeled synthetic data needed*)
- ▶ Learns the underlying data distribution
- ▶ Can generate novel, realistic samples never seen during training

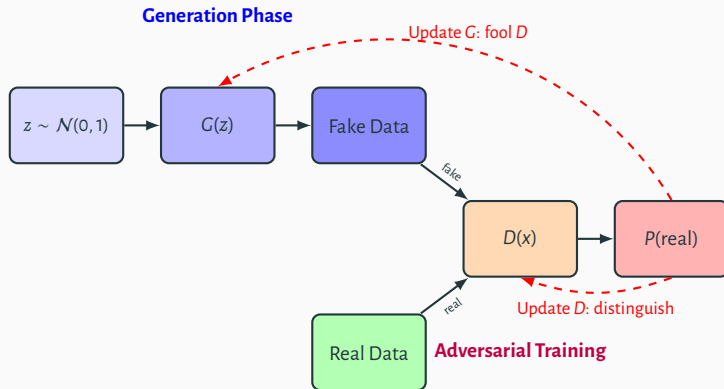
Image generation

Image style transfer

Super-resolution

Data augmentation

ARCHITECTURE & USE CASES



Task

Setup: You have a simple 1D **GAN** trying to generate data that matches a target distribution;

Target Distribution: Real data points are $\{2, 3, 4\}$.

Generator (G)

Takes noise z and produces fake data:

$$G(z) = 2z + 1, \quad \text{where } z \sim \mathcal{N}(0, 1)$$

Discriminator (D)

Binary classifier that outputs probability $[0, 1]$ that input is real:

$$D(x) = \sigma(wx + b) \tag{1}$$

where σ is the sigmoid function:

$$\sigma(a) = \frac{1}{1 + e^{-a}} \tag{2}$$

Initial weights: $w = 0.5, b = -1$

1. Forward Pass

- (a) Generate two fake samples using $z = 0$ and $z = 1$. What are the generated values?
- (b) For the real sample $x = 3$, compute $D(3)$.
- (c) For the fake sample $G(0)$, compute $D(G(0))$

2. Discriminator Loss

Compute \mathcal{L}_D using $x_{\text{real}} = 3$ and $x_{\text{fake}} = G(0)$. The discriminator tries to output 1 for real data and 0 for fake data. Binary cross-entropy loss for one real and one fake sample:

$$\mathcal{L}_D = - [\ln(D(x_{\text{real}})) + \ln(1 - D(x_{\text{fake}}))] \quad (3)$$

3. Generator Loss

Compute \mathcal{L}_G for $x_{\text{fake}} = G(1)$. The generator tries to fool the discriminator:

$$\mathcal{L}_G = -\ln(D(x_{\text{fake}})) \quad (4)$$

4. Gradient Direction (Conceptual)

For the discriminator loss you computed, should w increase or decrease to better classify $x = 3$ as real (output closer to 1)? What about to classify $x_{\text{fake}} = 1$ as fake (output closer to 0)?

1. Forward Pass

$$(a) G(0) = 2(0) + 1 = 1, \quad G(1) = 2(1) + 1 = 3$$

$$(b) D(3) = \sigma(0.5 \cdot 3 - 1) = \sigma(0.5) \approx 0.62$$

$$(c) D(G(0)) = D(1) = \sigma(0.5 \cdot 1 - 1) = \sigma(-0.5) \approx 0.38$$

2. Discriminator Loss

$$\mathcal{L}_D = -[\ln(D(3)) + \ln(1 - D(1))] \approx 0.96$$

3. Generator Loss

$$\mathcal{L}_G = -\ln(D(G(1))) = -\ln(D(3)) \approx 0.48$$

4. Gradient Direction

- ▶ To classify $x = 3$ as real: w should **increase** (make $D(3) \rightarrow 1$)
- ▶ To classify $x = 1$ as fake: w should **decrease** (make $D(1) \rightarrow 0$)
- ▶ These conflict! This is the adversarial game between G and D .



The code is available @ github.com/a-mhamdi/jlai → *Codes* → *Julia* → *Part-3*

→ *gan* → *gan.jl*

Pluto.jl

→ *gan* → *gan.ipynb*



SELF-INFORMATION, ENTROPY, AND CROSS-ENTROPY

Self-Information (Surprisal) (*surprise of event x*)

$$I(x) = -\log_2 p(x)$$

Example

Fair coin, $p(\text{heads}) = 0.5$: $I(\text{heads}) = -\log_2 0.5 = 1$ bit.

Entropy (*avg. uncertainty*)

$$H(P) = \mathbb{E}[I(X)] = -\sum_x p(x) \log_2 p(x)$$

Example

Fair coin: $H(P) = 1$ bit; Biased ($p = 0.9$): $H(P) \approx 0.469$ bits.

Cross-Entropy (# bits to encode P using Q 's code)

$$H(P, Q) = -\sum_x p(x) \log_2 q(x)$$

Example

P fair, Q biased ($q = 0.9$): $H(P, Q) \approx 1.085$ bits.

KL DIVERGENCE AND EXPECTATION

- Non-symmetric measure of difference between distributions P and Q ; quantifies expected info loss approximating P with Q .

$$D_{\mathcal{KL}}(P\|Q) = \sum_x P(x) \ln \frac{P(x)}{Q(x)}$$

(Non-negative; zero iff $P = Q$; asymmetric, not a true metric.)

- Always ≥ 0 ; used in ML (model fit), info theory (compression), stats (inference), steganography.

Expectation form (under P)

$$D_{\mathcal{KL}}(P\|Q) = \mathbb{E}_{X \sim P} \left[\ln \frac{P(X)}{Q(X)} \right]$$

Task:

Consider the following four discrete probability distributions over the support $\{1, 2, 3\}$:

P_1 skewed toward lower values

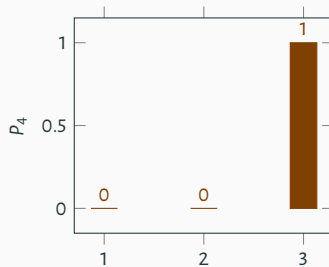
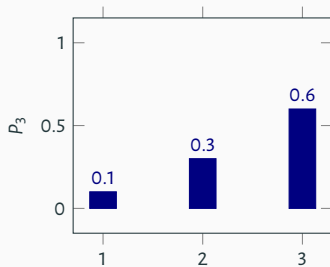
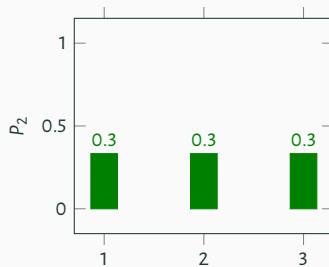
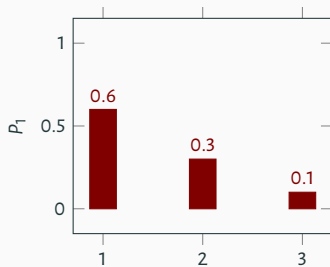
P_2 uniform

P_3 skewed toward higher values

P_4 degenerate, concentrated on one outcome

Compute the Kullback-Leibler divergence $D_{\mathcal{KL}}(P_1 \| P_2)$ using the natural logarithm. The $D_{\mathcal{KL}}$ divergence is given by:

$$D_{\mathcal{KL}}(P_1 \| P_2) = \sum_{x=1}^3 P_1(x) \ln \left(\frac{P_1(x)}{P_2(x)} \right)$$



From the bar charts, extract the probabilities:

- ✓ Compute the ratios $\frac{P_1(x)}{P_2(x)}$ for each x :
 - ▶ For $x = 1$: $\frac{0.6}{1/3} = 1.8$
 - ▶ For $x = 2$: $\frac{0.3}{1/3} = 0.9$
 - ▶ For $x = 3$: $\frac{0.1}{1/3} = 0.3$
- ✓ Take the natural logarithms:
 - ▶ $\ln(1.8) \approx 0.588$
 - ▶ $\ln(0.9) \approx -0.105$
 - ▶ $\ln(0.3) \approx -1.204$
- ✓ Multiply by $P_1(x)$:
 - ▶ $0.6 \times 0.588 \approx 0.353$
 - ▶ $0.3 \times (-0.105) \approx -0.032$
 - ▶ $0.1 \times (-1.204) \approx -0.12$
- ✓ Sum the values: $0.353 + (-0.032) + (-0.12) \approx 0.201$ Thus, $D_{\mathcal{KL}}(P_1 \| P_2) \approx 0.201$ nats.

Task:

Using the same distributions P_1 and P_2 from the bar charts, compute the Jensen-Shannon divergence $\mathcal{JS}(P_1, P_2)$, defined as:

$$\mathcal{JS}(P_1, P_2) = \frac{1}{2} D_{\mathcal{KL}}(P_1 \| M) + \frac{1}{2} D_{\mathcal{KL}}(P_2 \| M)$$

where $M = \frac{P_1 + P_2}{2}$ is the average distribution.

Let's compute the average distribution M :

$$\blacktriangleright M(1) = \frac{0.6 + 1/3}{2} \approx \frac{0.6 + 0.333}{2} = 0.467$$

$$\blacktriangleright M(2) = \frac{0.3 + 1/3}{2} \approx \frac{0.3 + 0.333}{2} = 0.317$$

$$\blacktriangleright M(3) = \frac{0.1 + 1/3}{2} \approx \frac{0.1 + 0.333}{2} = 0.217$$

$$D_{\mathcal{KL}}(P_1 \| M) \approx 0.056$$

$$D_{\mathcal{KL}}(P_2 \| M) \approx 0.047$$

Finally:

$$\mathcal{JS}(P_1, P_2) = \frac{1}{2}(0.058) + \frac{1}{2}(0.047) \approx 0.053$$

Thus, $\mathcal{JS}(P_1, P_2) \approx 0.053$ nats.

MOTIVATING FACTORS & USE CASES

A **Variational Autoencoder (VAE)** is a probabilistic generative model that learns a continuous latent representation of data. Unlike standard autoencoders, **VAEs** learn a **probability distribution** over the latent space, enabling controlled generation of new samples.

- ▶ **Encoder $q_{W_e}(z|x)$** : Maps input x to latent distribution parameters (μ, σ)
- ▶ **Latent space z** : Samples drawn from $\mathcal{N}(\mu, \sigma^2)$ (reparameterization trick)
- ▶ **Decoder $p_{W_d}(\hat{x}|z)$** : Reconstructs data from latent representation

Training objective (*Evidence Lower Bound - ELBO*):

- ▶ **Reconstruction loss**: Ensures decoded output matches input
- ▶ **KL divergence**: Regularizes latent space to follow prior distribution $\mathcal{N}(0, I)$

$$\text{ELBO} = \mathbb{E}_{q_{W_e}(z|x)} \left[\ln p_{W_d}(\hat{x}|z) \right] - D_{\mathcal{KL}}(q_{W_e}(z|x) || p(z))$$

- ▶ Balances faithful reconstruction with smooth, structured latent space

Generative modeling

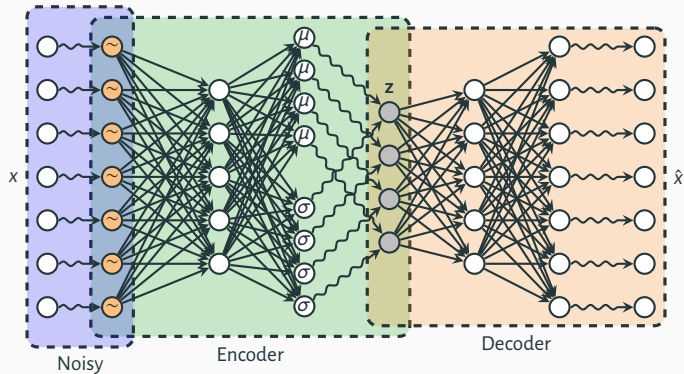
Anomaly detection

Data compression

Representation learning

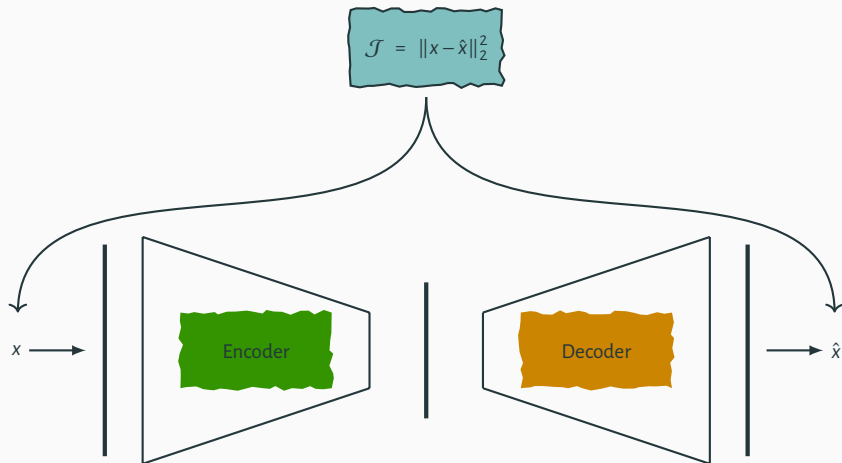
Semi-supervised learning

ARCHITECTURE OF VARIATIONAL AUTOENCODER



LOSS OF VANILLA AUTOENCODER

MINIMIZE SQUARED ERROR LOSS

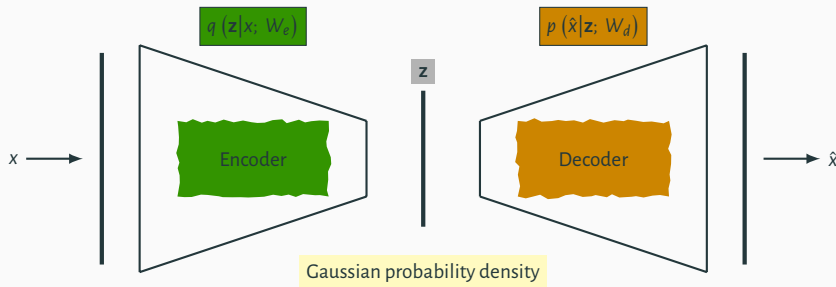


LOSS OF VARIATIONAL AUTOENCODER

$$\mathcal{J} = - \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}^{(i)}; W_e)} \left[\ln p(\hat{\mathbf{x}}^{(i)}|\mathbf{z}; W_d) \right] + D_{\mathcal{KL}} \left(q(\mathbf{z}|\mathbf{x}^{(i)}; W_e) \parallel p(\mathbf{z}) \right)$$

Expected negative log likelihood term wrt to encoder distribution

Kullback-Leibler divergence term
where $p(\mathbf{z}) \sim \mathcal{N}(\mu = 0, \sigma^2 = 1)$



$D_{\mathcal{KL}}$ LOSS DERIVATION

In a VAE, the latent vector \mathbf{z} is calculated by:

$$\mathbf{z} = \mu + \sigma \odot \epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(\mathbf{0}_l, \mathbb{1}_{l \times l})$$

μ and σ denote respectively the mean and variances for the latent vector \mathbf{z} . The encoder learns to output the two vectors $\mu \in \mathbb{R}^l$, and $\sigma \in \mathbb{R}^l$. The encoder distribution is

$$q(\mathbf{z} | \mathbf{x}^{(i)}) = \mathcal{N}(\mathbf{z} | \mu(\mathbf{x}^{(i)}), \Sigma(\mathbf{x}^{(i)})) \quad \text{where} \quad \Sigma = \begin{bmatrix} \sigma_1^2 & 0 & 0 & \cdots \\ 0 & \sigma_2^2 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_l^2 \end{bmatrix}$$

The latent prior is given by

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{0}_l, \mathbb{1}_{l \times l})$$

$$D_{\mathcal{KL}}(q(\mathbf{z} | \mathbf{x}^{(i)}; \text{We}) \parallel p(\mathbf{z})) = \frac{1}{2} \left[-\sum_{j=1}^l (\ln \sigma_j^2 + 1) + \sum_{j=1}^l \sigma_j^2 + \sum_{j=1}^l \mu_j^2 \right]$$



The code is available @ github.com/a-mhamdi/jlai → *Codes* → *Julia* → *Part-3*

→ *vae* → *vae.jl*

Pluto.jl

→ *vae* → *vae.ipynb*

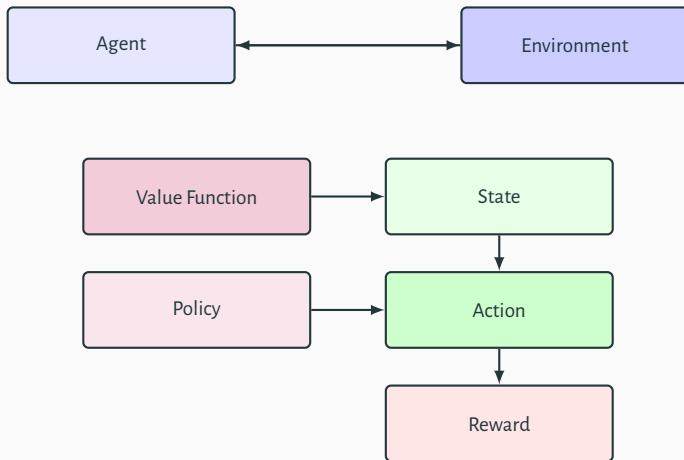


Reinforcement Learning

SYNOPSIS

- ▶ **Reinforcement Learning (RL)** is a machine learning paradigm in which an **agent** learns to make decisions by interacting with an **environment**. The agent learns a **policy**—a strategy mapping **states** to **actions**—that maximizes cumulative **reward** over time through trial and error.
- ▶ Learning occurs via feedback: the agent receives rewards (or penalties) for its actions and adjusts its behavior to improve future outcomes. This process inherently involves balancing **exploration** of new actions with **exploitation** of known rewarding behaviors.
- ▶ **RL** is applied in diverse domains such as control systems, game playing (*e.g., chess and Go*), robotics, and natural language processing, where it has achieved notable success in complex decision-making tasks.

PIPELINE



MARKOV PROPERTY

Definition

A state s_t satisfies the **Markov property** if:

$$P(s_{t+1} | s_t, s_{t-1}, \dots, s_0) = P(s_{t+1} | s_t)$$

→ *The future is independent of the past given the present. The current state captures all relevant information from history.*

Why Important for RL?

- ▶ Enables tractable decision-making
- ▶ State representation must capture sufficient information
- ▶ Foundation for Markov Decision Processes (MDPs)

MARKOV CHAIN & GRIDWORLD EXAMPLE

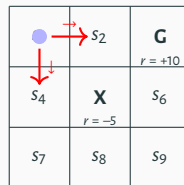
Markov Chain

A sequence of states where:

$$P(s_{t+1} = s' | s_t = s) = \mathcal{P}_{ss'}$$

Connection to RL:

- Models environment
- Add actions \rightarrow MDP
- Policy creates chain



G: Goal, **X:** Obstacle

MARKOV DECISION PROCESS (MDP)

Definition

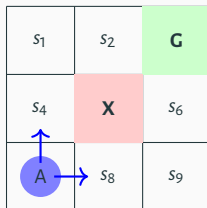
An MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where:

- ▶ \mathcal{S} : Set of states
- ▶ \mathcal{A} : Set of actions
- ▶ \mathcal{P} : State transition probability $\mathcal{P}(s' | s, a) = P(s_{t+1} = s' | s_t = s, a_t = a)$
- ▶ \mathcal{R} : Reward function $\mathcal{R}(s, a, s')$ or $\mathcal{R}(s, a)$
- ▶ $\gamma \in [0, 1]$: Discount factor

Dynamics: At each time step t :

1. Agent observes state $s_t \in \mathcal{S}$
2. Agent takes action $a_t \in \mathcal{A}$
3. Environment transitions to $s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)$
4. Agent receives reward $r_t = \mathcal{R}(s_t, a_t, s_{t+1})$

MDP: GRIDWORLD EXAMPLE



Components:

- ▶ $\mathcal{S} = \{s_1, \dots, s_9\}$
- ▶ $\mathcal{A} = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
- ▶ $\mathcal{R}(s, a, s')$:
 - Goal (G): +10
 - Obstacle (X): -5
 - Others: -1
- ▶ $\mathcal{P}(s' | s, a)$:
 - Intended: 0.8
 - Perpendicular: 0.1 each

Goal: Find policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ to maximize expected cumulative reward

ACTION SPACE: DISCRETE VS CONTINUOUS

Discrete Action Space

Finite set of actions: $\mathcal{A} = \{a_1, a_2, \dots, a_n\}$

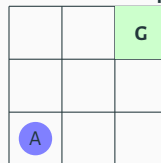
- ▶ Count actions: $|\mathcal{A}| < \infty$
- ▶ *GridWorld movements, game buttons*

Continuous Action Space

Infinite actions in range: $\mathcal{A} \subseteq \mathbb{R}^m$

- ▶ Real-valued vectors
- ▶ *Robot joint torques, steering angle*

3x3 GridWorld Examples



Discrete:

$$\mathcal{A} = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$$

Action: Move one cell

Continuous:

$$\mathcal{A} = \{(\Delta x, \Delta y) : \Delta x, \Delta y \in [-1, 1]\}$$

Example: $a = (0.7, 0.3)$

Move 0.7 right, 0.3 up

POLICY: DETERMINISTIC VS STOCHASTIC

A **policy** π maps states to actions, guiding the agent's behavior.

Deterministic Policy

$$a = \pi(s)$$

- Outputs single action
- Always same action in same state
- Example: $\pi(s_7) = \uparrow$

Stochastic Policy

$$a \sim \pi(a|s)$$

- Probability distribution over actions
- Randomized action selection
- Example: $\pi(\uparrow | s_7) = 0.6$,
 $\pi(\rightarrow | s_7) = 0.4$

3×3 GridWorld Example from s_7

Deterministic: $\pi(s_7) = \rightarrow$ (always move right)

Stochastic: $\pi(\uparrow | s_7) = 0.5$, $\pi(\rightarrow | s_7) = 0.3$, $\pi(\downarrow | s_7) = 0.1$, $\pi(\leftarrow | s_7) = 0.1$

EPISODES AND HORIZONS

A sequence of interactions: $s_0, a_0, r_1, s_1, a_1, r_2, \dots$ from initial state to terminal state.

Episodic Tasks

- ▶ Have terminal states
- ▶ Finite length episodes
- ▶ Agent resets after termination
- ▶ *GridWorld (reach goal), game rounds*

Continuing Tasks

- ▶ No terminal states
- ▶ Infinite interaction
- ▶ No natural endpoint
- ▶ *Stock trading, process control*

Horizon

Finite Horizon: Fixed time limit T . Return: $G_t = \sum_{k=0}^{T-t-1} r_{t+k+1}$

Infinite Horizon: Unlimited timesteps. Return: $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ (requires $\gamma < 1$)

VALUE FUNCTIONS

State-Value Function $V^\pi(s)$

Expected return starting from state s following policy π :

$$V^\pi(s) = \mathbb{E}_\pi [G_t \mid s_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right]$$

Action-Value Function $Q^\pi(s, a)$

Expected return starting from state s , taking action a , then following policy π :

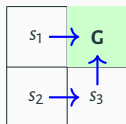
$$Q^\pi(s, a) = \mathbb{E}_\pi [G_t \mid s_t = s, a_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right]$$

Relationship:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q^\pi(s, a)$$

→ Value functions measure "how good" it is to be in a state (or take an action).

VALUE FUNCTION



Policy π :

- ▶ s_1 : right
- ▶ s_2 : right
- ▶ s_3 : up

Setup:

- ▶ Reward: -1 per step, $+10$ at goal
- ▶ Deterministic transitions
- ▶ Discount: $\gamma = 0.9$
- ▶ Goal (G) is terminal

Compute $V^\pi(s_1)$:

$$\begin{aligned} V^\pi(s_1) &= r + \gamma V^\pi(\text{Goal}) \\ &= -1 + 0.9 \times 10 \\ &= 8 \end{aligned}$$

Compute $V^\pi(s_2)$:

$$\begin{aligned} V^\pi(s_2) &= -1 + \gamma V^\pi(s_3) \\ V^\pi(s_3) &= -1 + \gamma V^\pi(\text{Goal}) \\ &= 8 \\ V^\pi(s_2) &= -1 + 0.9 \times 8 = \mathbf{6.2} \end{aligned}$$

MODEL-BASED VS MODEL-FREE RL

Model-Based RL

Agent has access to (or learns) a model of the environment: $\mathcal{P}(s'|s, a)$ and $\mathcal{R}(s, a)$

- ▲ Can plan ahead
 - ▲ Sample efficient
 - ▲ Simulate trajectories
-
- ▶ *Dynamic Programming*
 - ▶ *AlphaZero (learns model)*

Model-Free RL

Agent learns directly from experience without modeling environment dynamics

- ▲ No model bias
 - ▲ Simpler implementation
 - ▲ Works in complex environments
-
- ▶ *Q-Learning*
 - ▶ *Policy Gradient methods*
 - ▶ *Actor-Critic*

→ *Model-based uses \mathcal{P} and \mathcal{R} explicitly; model-free learns value/policy from transitions (s, a, r, s') directly.*

ENVIRONMENT PROPERTIES

DETERMINISTIC VS STOCHASTIC ENVIRONMENT

Deterministic: Next state fully determined by current state and action

$$s_{t+1} = f(s_t, a_t) \quad (\text{no randomness})$$

Chess, deterministic GridWorld

Stochastic: Next state sampled from probability distribution

$$s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t) \quad (\text{randomness in transitions})$$

Dice games, robot control with sensor noise, slippery GridWorld (0.8 intended, 0.1 each perpendicular)

ENVIRONMENT PROPERTIES

EPISODIC VS NON-EPISODIC (CONTINUING)

Episodic: Task has terminal states, natural endpoints, episodes reset

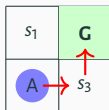
Game rounds, robot reaching target

Non-Episodic (Continuing): No terminal states, infinite interaction. Requires discount factor $\gamma < 1$ for bounded returns

Stock trading, server load balancing, process control

(2x2) GRIDWORLD EXAMPLE

DETERMINISTIC ENVIRONMENT + DETERMINISTIC POLICY



Environment: Deterministic

- ▶ Transitions: 100% in intended direction
- ▶ $\mathcal{P}(s_3 | s_2, \rightarrow) = 1.0$

Policy: Deterministic π

- ▶ $\pi(s_2) = \rightarrow$
- ▶ $\pi(s_3) = \uparrow$

Rewards & Discount:

- ▶ Step: -1, Goal: +10, $\gamma = 0.9$

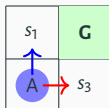
Trajectory from s_2 : $s_2 \xrightarrow{\rightarrow} s_3 \xrightarrow{\uparrow} G$

Return: $G = -1 + 0.9(-1) + 0.9^2(10) = -1 - 0.9 + 8.1 = 6.2$

Value: $V^\pi(s_2) = 6.2$ (deterministic, so expected = actual)

(2x2) GRIDWORLD EXAMPLE

DETERMINISTIC ENVIRONMENT + STOCHASTIC POLICY



Environment: Deterministic

- ▶ Transitions: 100% success

Policy: Stochastic π

- ▶ $\pi(\rightarrow | s_2) = 0.7$
- ▶ $\pi(\uparrow | s_2) = 0.3$

Rewards & Discount:

- ▶ Step: -1, Goal: +10, $\gamma = 0.9$

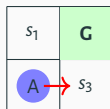
Possible Trajectories from s_2 :

- ▶ Path 1 (prob 0.7): $s_2 \xrightarrow{\rightarrow} s_3 \xrightarrow{\uparrow} G \rightarrow \text{Return: } -1 + 0.9(-1) + 0.9^2(10) = 6.2$
- ▶ Path 2 (prob 0.3): $s_2 \xrightarrow{\uparrow} s_1 \xrightarrow{\rightarrow} G \rightarrow \text{Return: } -1 + 0.9(-1) + 0.9^2(10) = 6.2$

Value: $V^\pi(s_2) = 0.7(6.2) + 0.3(6.2) = 6.2$ (same path length!)

(2x2) GRIDWORLD EXAMPLE

STOCHASTIC ENVIRONMENT + DETERMINISTIC POLICY



Environment: Stochastic (slippery)

- ▶ Intended: 0.8
- ▶ Perpendicular: 0.1 each
- ▶ $\mathcal{P}(s_3 | s_2, \rightarrow) = 0.8$
- ▶ $\mathcal{P}(s_1 | s_2, \rightarrow) = 0.1$
- ▶ $\mathcal{P}(s_2 | s_2, \rightarrow) = 0.1$ (hit wall)

Policy: Deterministic

- ▶ $\pi(s_2) = \rightarrow$

Rewards: Step: -1, Goal: +10, $\gamma = 0.9$

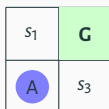
Expected outcomes from s_2 taking \rightarrow :

- ▶ 0.8: reach $s_3 \rightarrow V^\pi(s_3) = 8$
- ▶ 0.1: reach $s_1 \rightarrow V^\pi(s_1) = 8$
- ▶ 0.1: stay $s_2 \rightarrow V^\pi(s_2)$ (recursive)

Value: $V^\pi(s_2) = -1 + 0.9 [0.8(8) + 0.1(8) + 0.1V^\pi(s_2)] \rightarrow V^\pi(s_2) = 6.02$

(2x2) GRIDWORLD EXAMPLE

STOCHASTIC ENVIRONMENT + STOCHASTIC POLICY



Environment: Stochastic

- ▶ Intended: 0.8, Perpendicular: 0.1 each

Policy: Stochastic π

- ▶ $\pi(\rightarrow | s_2) = 0.6$
- ▶ $\pi(\uparrow | s_2) = 0.4$

Rewards: Step: -1, Goal: +10, $\gamma = 0.9$

Computing $V^\pi(s_2)$:

When action \rightarrow (prob 0.6):

- ▶ $\mathcal{P}(s_3 | s_2, \rightarrow) = 0.8, \mathcal{P}(s_1 | s_2, \rightarrow) = 0.1, \mathcal{P}(s_2 | s_2, \rightarrow) = 0.1$

When action \uparrow (prob 0.4):

- ▶ $\mathcal{P}(s_1 | s_2, \uparrow) = 0.8, \mathcal{P}(s_3 | s_2, \uparrow) = 0.1, \mathcal{P}(s_2 | s_2, \uparrow) = 0.1$

Value: $V^\pi(s_2) = 5.48 + 0.09V^\pi(s_2) \rightarrow V^\pi(s_2) = 6.02$

GRIDWORLD SETUP (3×3)



Problem Specification:

- **States:** $S = \{(r, c) : r, c \in \{1, 2, 3\}\}$ (9 states)
- **Actions:** $\mathcal{A} = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
- **Terminal:** (3, 3) gives $R = +10$, episode ends
- **Step Cost:** $R = -1$ for all non-terminal transitions
- **Discount:** $\gamma = 0.9$
- **Walls:** Hitting boundary \rightarrow stay in place, pay -1

Value Functions:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t \mid s_0 = s \right]$$

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t \mid s_0 = s, a_0 = a \right]$$

Relationship: $V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s, a)$

CASE 1: DETERMINISTIC ENVIRONMENT & POLICY

Policy: $\pi(s)$ = Right if possible, else Down (deterministic)

Environment: 100% intended movement (deterministic transitions)

(1, 1) 4.58 →	(1, 2) 6.2 →	(1, 3) 8.0 ↓
(2, 1) 6.2 →	(2, 2) 8.0 →	(2, 3) 10.0 ↓
(3, 1) 8.0 →	(3, 2) 10.0 →	G

Values: $V^\pi(s)$, Arrows: $\pi(s)$

Bellman Equation (Deterministic):

$$V^\pi(s) = R(s, \pi(s)) + \gamma V^\pi(s')$$

$$Q^\pi(s, a) = R(s, a) + \gamma V^\pi(s')$$

Value Iteration (Backward from Goal):

$$V(3, 3) = 0 \quad (\text{terminal})$$

$$V(3, 2) = +10 + 0.9(0) = 10.0$$

$$V(2, 3) = +10 + 0.9(0) = 10.0$$

$$V(3, 1) = -1 + 0.9(10) = -1 + 9 = 8.0$$

$$V(2, 2) = -1 + 0.9(10) = -1 + 9 = 8.0$$

$$V(1, 3) = -1 + 0.9(10) = -1 + 9 = 8.0$$

$$V(2, 1) = -1 + 0.9(8) = -1 + 7.2 = 6.2$$

$$V(1, 2) = -1 + 0.9(8) = -1 + 7.2 = 6.2$$

$$V(1, 1) = -1 + 0.9(6.2) = -1 + 5.58 = 4.58$$

Q-Values at Start State (1, 1):

$$Q(\rightarrow) = -1 + 0.9(6.2) = 4.58$$

$$Q(\downarrow) = -1 + 0.9(6.2) = 4.58$$

$$Q(\leftarrow) = -1 + 0.9(4.58) = 3.122$$

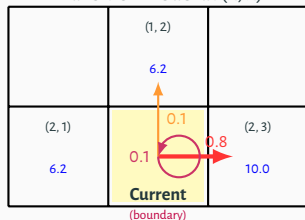
$$Q(\uparrow) = -1 + 0.9(4.58) = 3.122$$

CASE 2: STOCHASTIC ENVIRONMENT (SLIPPERY GRID)

Policy: Deterministic (Right if possible, else Down)

Environment: 80% intended, 10% perpendicular slip each side

Transition Model at (2, 2)



Note: "Perpendicular slip" means if action is horizontal (R/L), slip is vertical (U/D); if action is vertical, slip is horizontal.

Stochastic Bellman Equation:

$$V^{\pi}(s) = \sum_{s'} P(s' | s, \pi(s)) [R + \gamma V^{\pi}(s')]$$

Computing $V(2, 2)$ with action Right:

$$V(2, 2) =$$

$$\begin{aligned} &0.8[-1 + 0.9V(2, 3)] \quad (\text{intended}) \\ &+ 0.1[-1 + 0.9V(1, 2)] \quad (\text{slip up}) \\ &+ 0.1[-1 + 0.9V(2, 2)] \quad (\text{slip down/stay}) \end{aligned}$$

(using deterministic values as approximation):

$$\begin{aligned} V(2, 2) &= 0.8[-1 + 0.9(10)] \\ &+ 0.1[-1 + 0.9(6.2)] \\ &+ 0.1[-1 + 0.9V(2, 2)] \end{aligned}$$

$$0.91V(2, 2) = 6.758$$

$$V(2, 2) = \boxed{7.426}$$

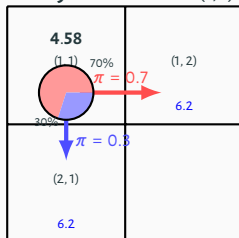
- Deterministic: $V(2, 2) = 8.0$
- Stochastic: $V(2, 2) = 7.426$
- **Penalty: -0.574** (due to slipping risk)

CASE 3: DETERMINISTIC ENVIRONMENT & STOCHASTIC POLICY

Environment: 100% intended movement (deterministic)

Policy at (1, 1): $\pi(\rightarrow | 1, 1) = 0.7$, $\pi(\downarrow | 1, 1) = 0.3$ (stochastic)

Policy Distribution at (1, 1)



Q-Function Table at (1, 1):

Action	$Q^\pi(1, 1, a)$
\rightarrow	4.58
\downarrow	4.58
\leftarrow	3.122
\uparrow	3.122

Bellman Expectation over Actions:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \cdot Q^\pi(s, a)$$

$$Q^\pi(s, a) = R(s, a) + \gamma V^\pi(s')$$

Step 1: Compute Q-values

$$Q(\rightarrow) = -1 + 0.9(6.2) = 4.58$$

$$Q(\downarrow) = -1 + 0.9(6.2) = 4.58$$

$$Q(\leftarrow) = -1 + 0.9(4.58) = 3.122$$

$$Q(\uparrow) = -1 + 0.9(4.58) = 3.122$$

Step 2: Expectation over stochastic policy

$$\begin{aligned}
 V^\pi(1, 1) &= \pi(\rightarrow) \cdot Q(\rightarrow) + \pi(\downarrow) \cdot Q(\downarrow) \\
 &\quad + \pi(\leftarrow) \cdot Q(\leftarrow) + \pi(\uparrow) \cdot Q(\uparrow) \\
 &= 0.7(4.58) + 0.3(4.58) \\
 &\quad + 0(3.122) + 0(3.122) \\
 &= 4.58(0.7 + 0.3) \\
 &= \boxed{4.58}
 \end{aligned}$$



The code is available @ github.com/a-mhamdi/jlai → Codes → Julia → Part-3

→ *reinforcement-learning* → *reinforcement-learning.jl*

Pluto.jl

→ *reinforcement-learning* → *reinforcement-learning.ipynb*



Quizzes

KNOWLEDGE CHECK



1

Go to wooclap.com

2

Enter the event code in the top banner

Event code
JLAI3<https://app.wooclap.com/JLAI3>

FURTHER READING (1/2)

References

- [Alz+21] L. Alzubaidi et al. “**Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions**”. In: *Journal of Big Data* 8.1 (2021). DOI: 10.1186/s40537-021-00444-8.
- [Azu21] P. Azunre. ***Transfer Learning for Natural Language Processing***. Manning Publications Co. LLC, 2021, p. 272.
- [Goo+17] I. Goodfellow et al. ***Deep Learning***. MIT Press, 2017.
- [HZM16] X. Hao, G. Zhang, and S. Ma. “**Deep Learning**”. In: *International Journal of Semantic Computing* 10.03 (2016), pp. 417–439. DOI: 10.1142/s1793351x16500045.
- [KW13] D. P. Kingma and M. Welling. “**Auto-Encoding Variational Bayes**”. In: (Dec. 2013). arXiv: 1312.6114 [stat.ML].
- [LBH15] Y. LeCun, Y. Bengio, and G. Hinton. “**Deep learning**”. In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539.
- [LD19] B. Lauwens and A. B. Downey. ***Think Julia : How to Think Like a Computer Scientist. How to Think Like a Computer Scientist***. O'Reilly Media, 2019, p. 298.

FURTHER READING (2/2)

- [PWP20] D. Phil Winder Ph. ***Reinforcement Learning Industrial Applications of Intelligent Agents. Industrial Applications of Intelligent Agents.*** O'Reilly Media, Incorporated, 2020.
- [Sar21] I. H. Sarker. “**Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions**”. In: *SN Computer Science* 2.6 (2021). DOI: 10.1007/s42979-021-00815-1.
- [SB] R. S. Sutton and A. G. Barto. ***Reinforcement Learning An Introduction. An Introduction.*** A Bradford Book, p. 552.
- [SC21] F. F. L. da Silva and A. H. R. A. H. R. Costa. ***Transfer Learning for Multiagent Reinforcement Learning Systems.*** Springer International Publishing AG, 2021.
- [Ser21] L. Serrano. ***Grokking Machine Learning.*** Manning Publications Co. LLC, 2021, p. 498.
- [SKP] M. Sewak, M. R. Karim, and P. Pujari. ***Practical Convolutional Neural Networks: Implement advanced deep learning models using Python.*** Packt Publishing - ebooks Account, p. 218.
- [SR] J. Silge and D. Robinson. ***Text Mining with R: A Tidy Approach.*** O'Reilly Media, p. 194.