

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/>



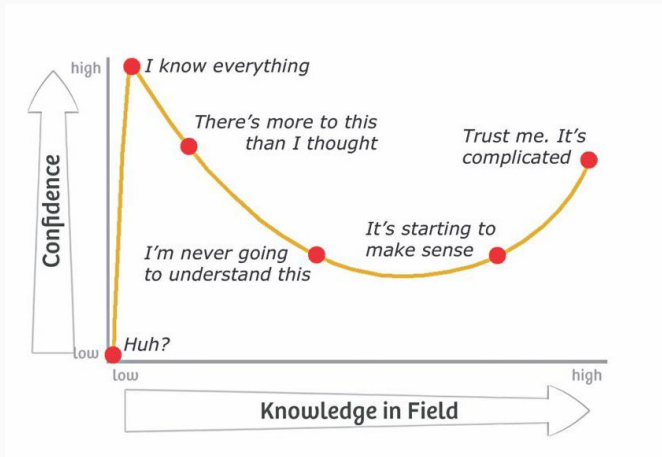
Disclaimer

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.

doi 10.1037/0022-3514.77.6.1121

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

Richard P. Feynman




1. CNN, VAE, GAN & NLP
2. Transfer Learning
3. Reinforcement Learning
4. 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 ASCII art logo of the word "julia" made of small circles, followed by the text "Documentation: https://docs.julialang.org", "Type '?' for help, ']' for pkg help.", and "Version 1.6.3 (2021-09-23)". 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

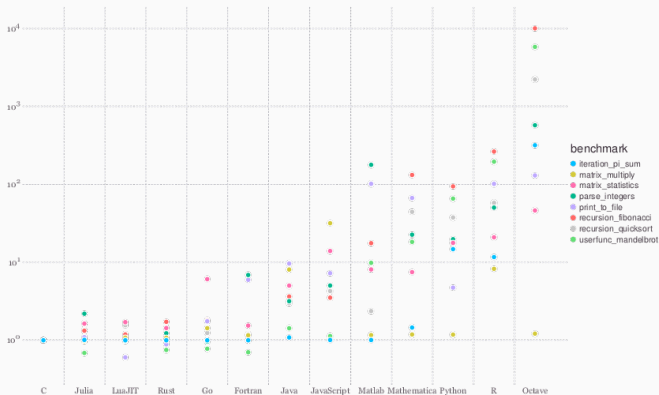
▼ \$ docker compose down



- ▲ **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 displays the GitHub interface for the repository `a-mhamdi/jlai`. The repository is public and has 2 unwatchers, 3 forks, and 2 stars. The main content area shows the file structure of the repository, with the `README.md` file selected. The README content is titled "Fuzzy Logic, Machine Learning and Deep Learning with Julia". The right sidebar provides additional information, including the repository's description, "An Introduction to Artificial Intelligence with Julia", and a list of languages used in the repository: Julia (94.3%), Dockerfile (3.4%), Batchfile (2.1%), and TeX (0.2%).

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

Languages

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 is currently empty. On the right side of the "Overview" section, there is a "Docker Pull Command" section with the command `docker pull abmhamdi/jlai-p3` and a "Copy" button.

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

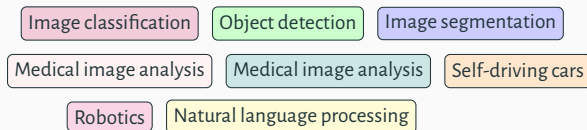
CNN, VAE, GAN & NLP

- CNNs** (*Convolutional Neural Networks*) 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.
- VAEs** (*Variational Autoencoders*) 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.
- GANs** (*Generative Adversarial Networks*) 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.
- NLP** (*Natural Language Processing*) is important for tasks such as language translation, text classification, and language generation because it allows computers to process and understand human language.

CNN

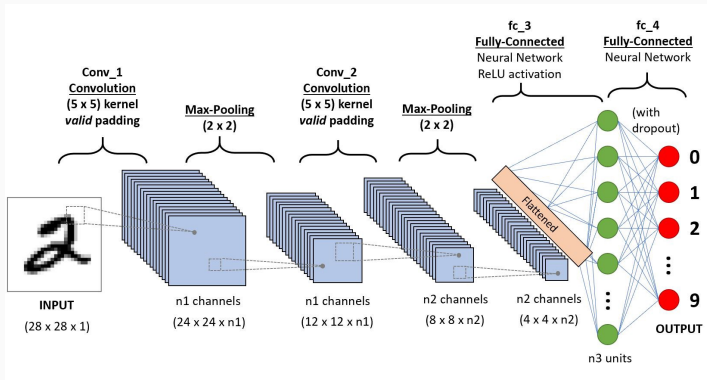
MOTIVATING FACTORS & USE CASES

- ▶ 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.



CNN

ARCHITECTURE



► Source

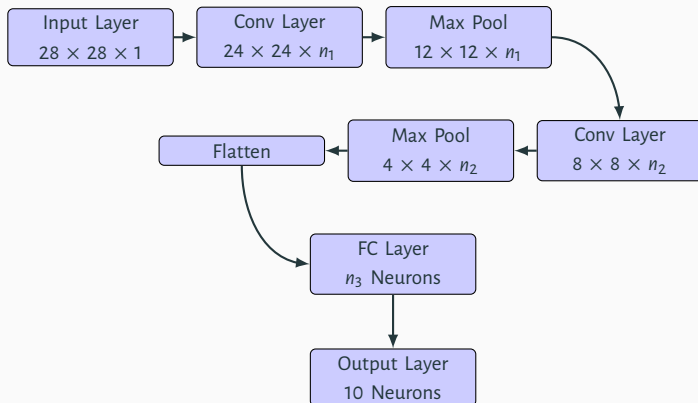
CNN

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

CNN

PIPELINE



CNN

IMAGE KERNELS

Image Kernels explained · x +

setosa.io/ev/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/>

CNN

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.

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

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

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

CNN

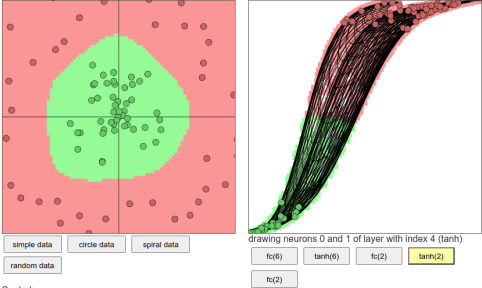
CONVNETJS DEMO

ConvNetJS demo: Classify

cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

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/)

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

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

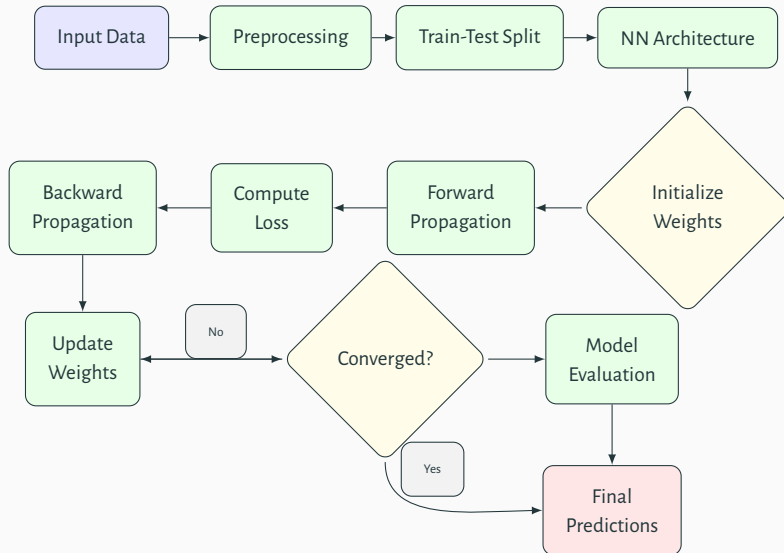
fc(2)

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

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

CNN

NEURAL NETWORK LEARNING ALGORITHM



CNN

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*



VAE

MOTIVATING FACTORS & USE CASES

- ▶ A **Variational Autoencoder (VAE)** is a type of deep learning model that is used to learn latent representations of data. It is a generative model, which means that it can generate new samples of data that are similar to the training data.
- ▶ **VAEs** are trained to encode the data into a low-dimensional latent space and then decode the latent representation back into the original data space. During training, the **VAE** learns to reconstruct the input data, while also trying to enforce a constraint on the latent space that encourages it to represent the data in a meaningful way.
- ▶ The constraint that is used in a **VAE** is called the variational lower bound. This lower bound is maximized during training, which encourages the latent space to be structured in a way that is useful for generating samples that are similar to the training data.

Generative modeling

Anomaly detection

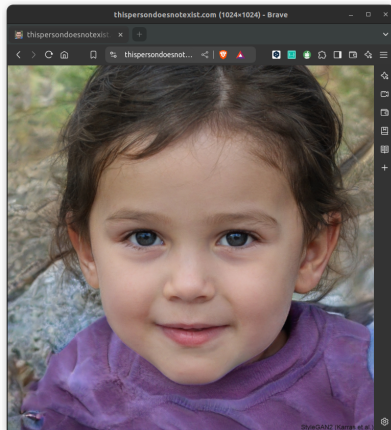
Data compression

Representation learning

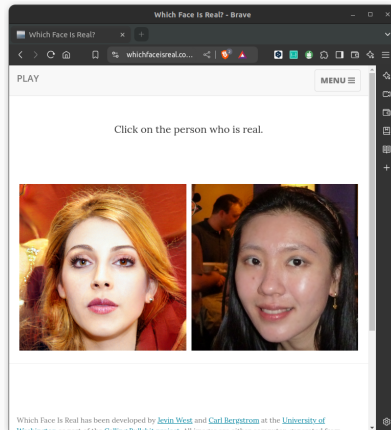
Semi-supervised learning

VAE

DEMOS



<https://thispersondoesnotexist.com/>

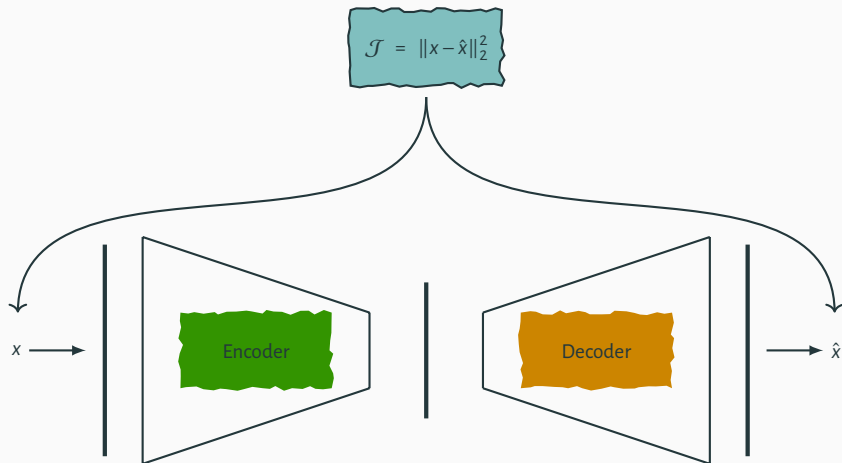


<https://www.whichfaceisreal.com>

VAE

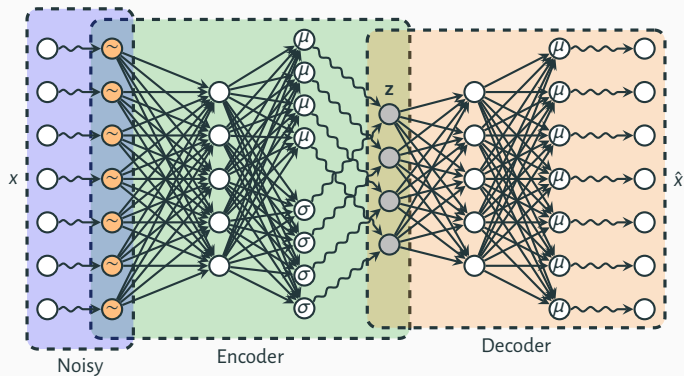
LOSS OF VANILLA AUTOENCODER

MINIMIZE SQUARED ERROR LOSS



VAE

ARCHITECTURE OF VARIATIONAL AUTOENCODER



VAE

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.

VAE

KL DIVERGENCE AND EXPECTATION

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

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

(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)

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

VAE

Task:

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

P_1 skewed toward lower values

P_2 uniform

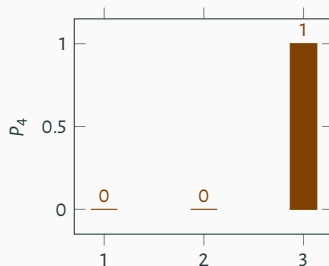
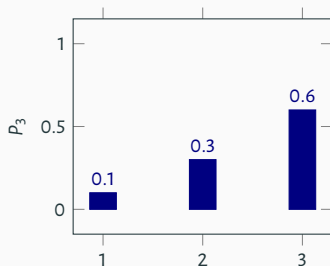
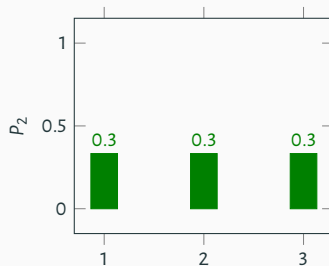
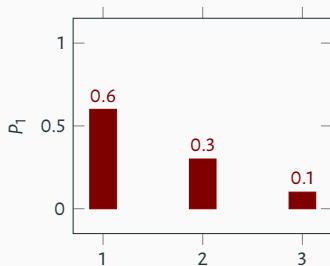
P_3 skewed toward higher values

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:

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

VAE



VAE

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, $D_{KL}(P_1 \| P_2) \approx 0.201$ nats.

VAE

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.

VAE

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.

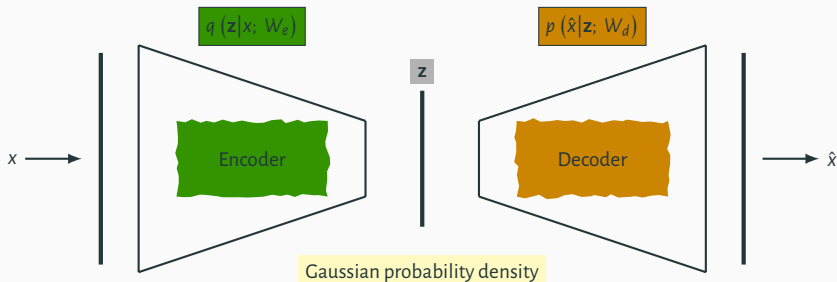
VAE

LOSS OF VARIATIONAL AUTOENCODER

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

Expected negative log likelihood term wrt to encoder distribution

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



VAE

 D_{KL} LOSS DERIVATION

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

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

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

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

The latent prior is given by

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

$$D_{KL}(q(\mathbf{z} | \mathbf{x}^{(i)}; \boldsymbol{\mu}, \boldsymbol{\sigma}) \| p(\mathbf{z})) = \frac{1}{2} \left[-\sum_i (\log \sigma_i^2 + 1) + \sum_i \sigma_i^2 + \sum_i \mu_i^2 \right]$$

VAE



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

→ *vae* → *vae.jl*

Pluto.jl

→ *vae* → *vae.ipynb*



GAN

AN OVERVIEW

- ▶ A **Generative Adversarial Network (GAN)** is a type of deep learning model designed to generate new, synthetic samples of data. It consists of two networks: a **generator network** and a **discriminator network**. The **generator network** generates synthetic samples, while the **discriminator network** tries to distinguish between the synthetic samples and real samples of data.
- ▶ During training, the **generator and discriminator networks** are trained concurrently, with the **generator** trying to generate synthetic samples that are indistinguishable from real samples, and the **discriminator** trying to correctly classify the samples as either real or synthetic. The **generator** is trained to improve its synthetic samples based on the feedback from the **discriminator**, and the **discriminator** is trained to become more sensitive to synthetic samples.
- ▶ The goal of a **GAN** is to learn a generative model that can produce synthetic samples that are similar to the training data.

Image generation

Image style transfer

Text-to-image synthesis

Video prediction

Text-to-speech synthesis

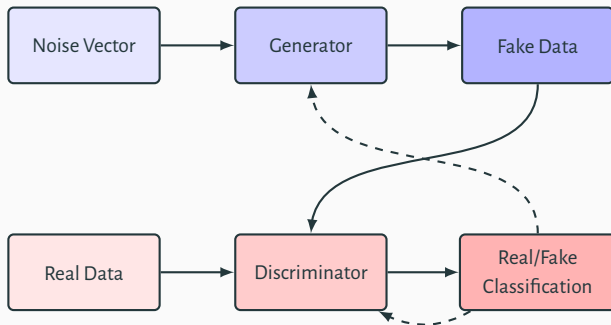
Super-resolution

Data augmentation

Domain adaptation

GAN

ARCHITECTURE & USE CASES



GAN



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

→ *gan* → *gan.jl*

Pluto.jl

→ *gan* → *gan.ipynb*



NLP

PURPOSE & USE CASES

- ▶ **Natural Language Processing (NLP)** is a field of artificial intelligence and computer science that focuses on the interaction between computers and humans using natural language.
- ▶ **NLP** involves the development of algorithms and models that can understand, interpret, and generate human language.
- ▶ **NLP** is used in a wide range of applications, including machine translation, question answering, text summarization, text classification, and sentiment analysis.

Part-of-speech tagging

Named entity recognition

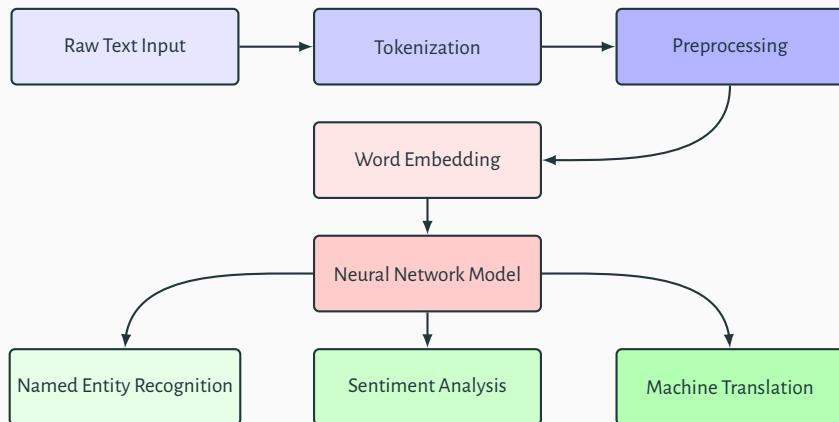
Sentiment analysis

Machine translation

Text summarization

NLP

PIPELINE



NLP

GENERAL PROCESS IN JULIA

1. Preprocess the text data by lowercasing, removing punctuation, and splitting the text into individual tokens (*e.g.*, words or subwords).
2. Build a vocabulary of the most common tokens in the text data.
3. Encode the text data as a sequence of integers using the vocabulary.
4. Pad the encoded sequences to the same length to make them suitable for input to a model.
5. Define the **NLP** model using a library such as `Flux.jl` or `Knet.jl`.
6. Train the model using gradient descent and a suitable loss function.
7. Use the trained model to make predictions on new data.

NLP



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

→ *nlp* → *nlp.jl*

Pluto.jl

→ *nlp* → *nlp.ipynb*



Transfer Learning

TRANSFER LEARNING

DRIVING FORCES & USE CASES

- ▶ **Transfer Learning** is a machine learning technique in which a model that has been trained on one task is re-purposed on a second related task. **Transfer Learning** can be used to improve the performance of the second task by leveraging the knowledge learned from the first task.
- ▶ One common use of **Transfer Learning** is to fine-tune a pre-trained model on a new dataset. For example, a pre-trained image classification model that has been trained on a large dataset such as ImageNet can be fine-tuned on a smaller dataset of a different but related task, such as detecting objects in medical images. Fine-tuning the pre-trained model on the new dataset can lead to improved performance compared to training a model from scratch on the smaller dataset.
- ▶ **Transfer Learning** is useful because it allows a machine learning model to learn from a large amount of data, even if the data is not directly related to the task at hand. It can also be used to speed up the training process, since the model does not need to be trained from scratch.

Image classification

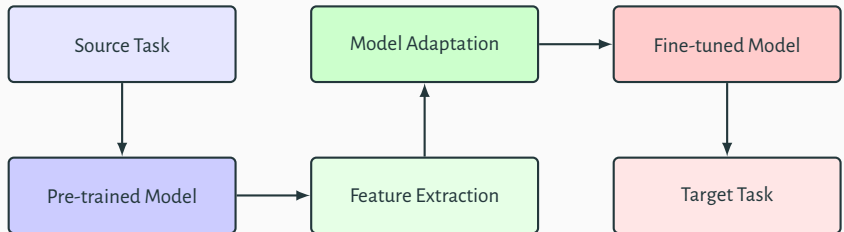
Computer vision

Natural language processing

Robotics

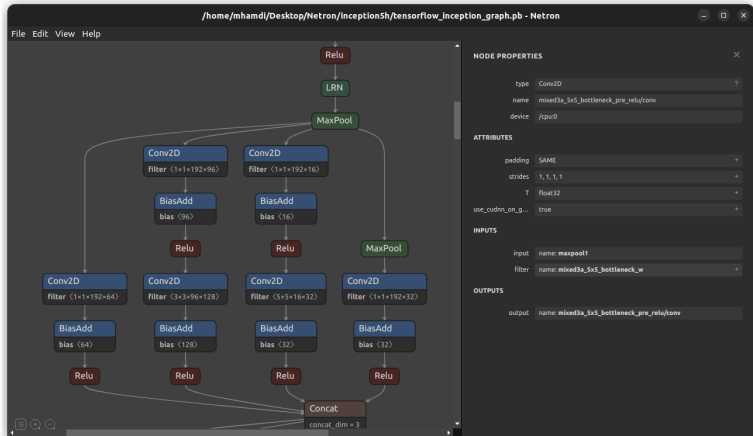
TRANSFER LEARNING

PIPELINE



TRANSFER LEARNING

NETRON



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

TRANSFER LEARNING

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.

TRANSFER LEARNING



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*



Reinforcement Learning

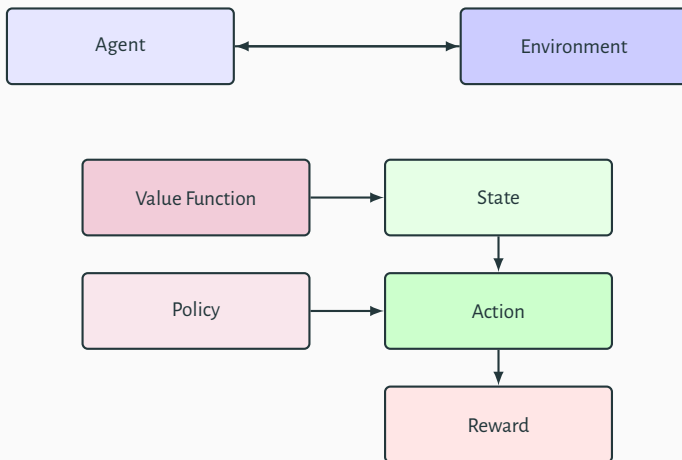
REINFORCEMENT LEARNING

SYNOPSIS

- ▶ **Reinforcement Learning** is a type of machine learning in which an agent learns to interact with its environment in order to maximize a reward. It involves learning to map situations (called states) to actions that will maximize a reward. The agent receives feedback in the form of rewards and penalties for its actions, which it uses to adjust its behavior accordingly.
- ▶ In **reinforcement Learning**, the goal is to learn a policy that maximizes the cumulative reward over time. The agent learns this policy through **trial and error**, by exploring different actions in different states and receiving feedback in the form of rewards or penalties.
- ▶ **Reinforcement Learning** is used in a variety of applications, including control systems, game playing, and natural language processing. It has been successful in a number of tasks, including teaching a computer to play chess and Go at a high level.

REINFORCEMENT LEARNING

PIPELINE



REINFORCEMENT LEARNING



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.