Demystifying Artificial Intelligence Sorcery

(Part 3: Deep Learning)[a]

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

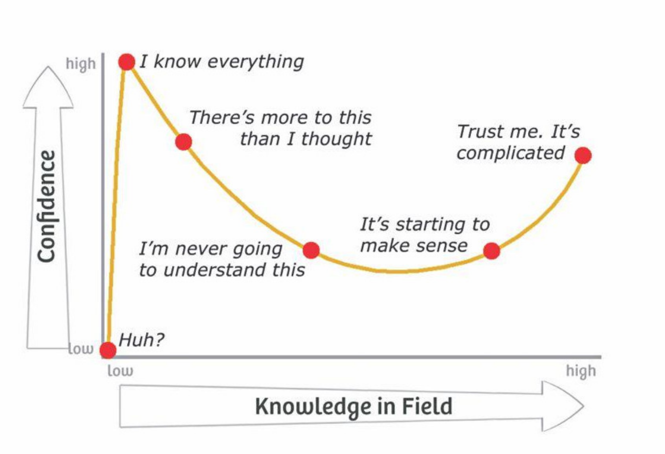
Dr.-Ing. in Electrical Engineering
Senior Lecturer at ISET Bizerte

[a]Available @ 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

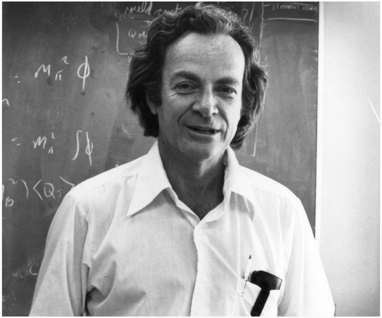


Kruger, J. and Dunning, D. (1999) *Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments.* **J Pers Soc Psychol.** 77(6) pp. 1121–1134.

doi 10.1037/0022-3514.77.6.1121

"Knowledge isn't free. You have to pay attention."

Richard P. Feynman



Roadmap

1. CNN, VAE, GAN & NLP
2. Transfer Learning
3. Reinforcement Learning
4. Quizzes

REMINDER

Programming Language

julialang.org/

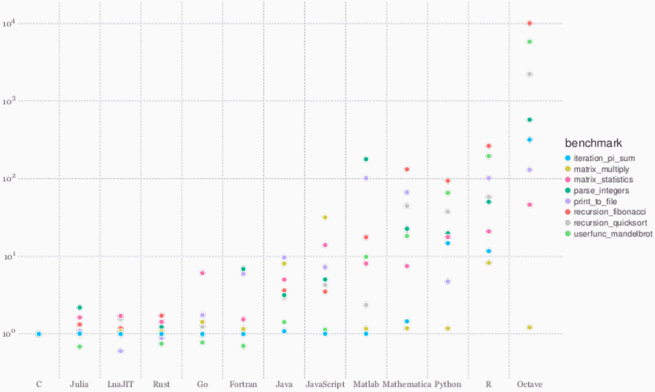
Development Environments

- ▲ $ docker compose up
- ▼ $ docker compose down

Julia in a Nutshell

- **Fast:** native code for multiple platforms via `LLVM`;
- **Dynamic:** good support for interactive use *(like a 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

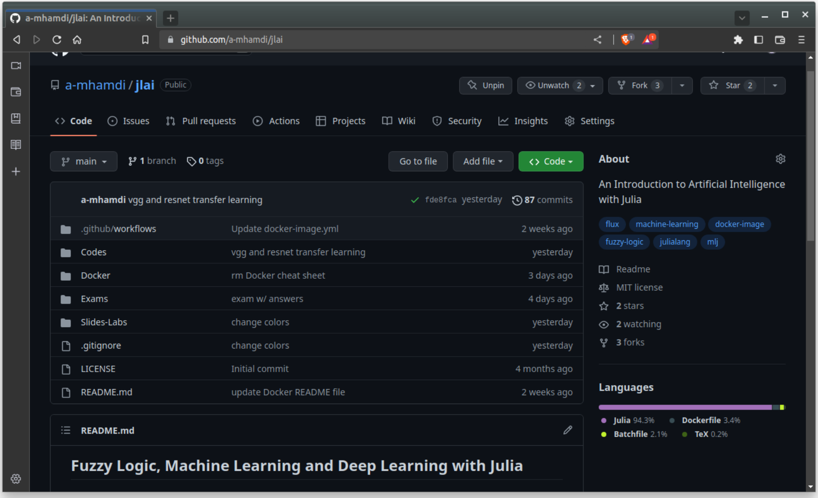
Julia Micro-Benchmarks (2/2)

Geometric Means[1] 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

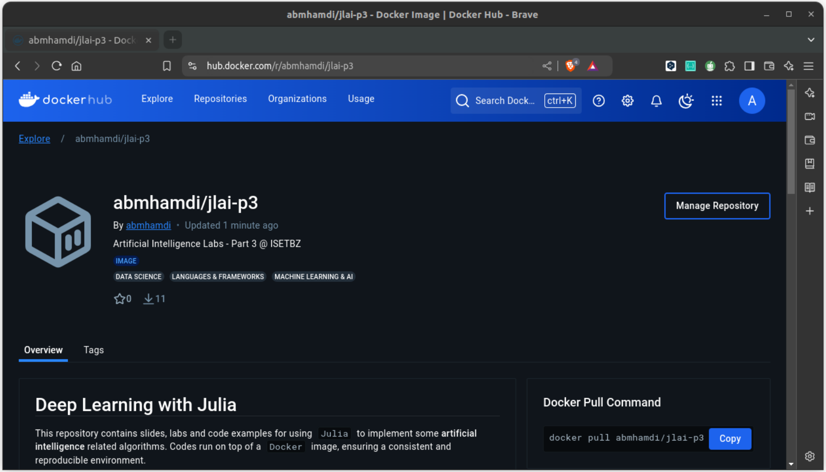
[1] Measure of central tendency expressed as $(x_1 \times x_2 \times \cdots \times x_n)^{1/n}$

SOURCE CONTROL MANAGEMENT (SCM)



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

Docker Image



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:

1. The convolutional layers apply filters to the input data, which are used to detect patterns and features in the data.
2. 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.
3. The fully connected layers combine the features learned by the convolutional and pooling layers to make a prediction.

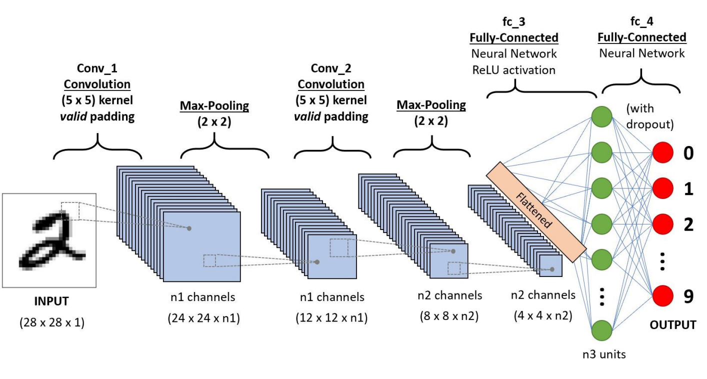
Image classification | Object detection | Image segmentation

Medical image analysis | Medical image analysis | Self-driving cars

Robotics | Natural language processing

CNN

Architecture



Source

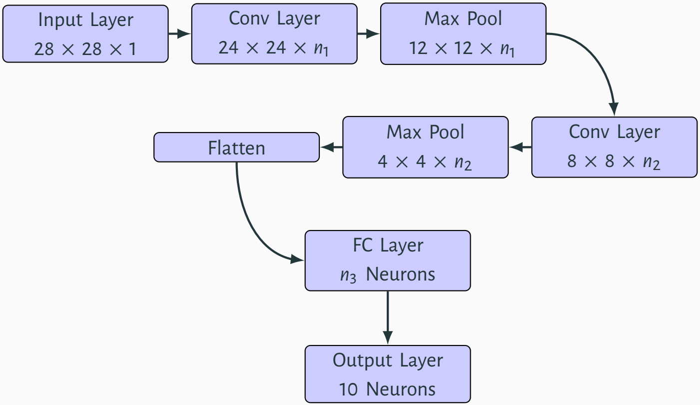
CNN

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

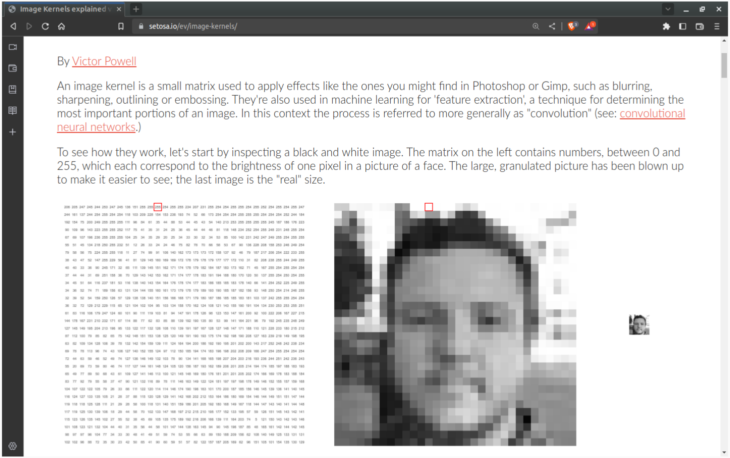
CNN

Pipeline



CNN

Image Kernels



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.

$$\boxed{\text{output\_shape} = \left\lfloor \frac{\text{input\_shape} + 2 \times \overbrace{\text{padding}}^{p} - \overbrace{\text{filter\_size}}^{k}}{\underbrace{\text{stride}}_{s}} \right\rfloor + 1}$$

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

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

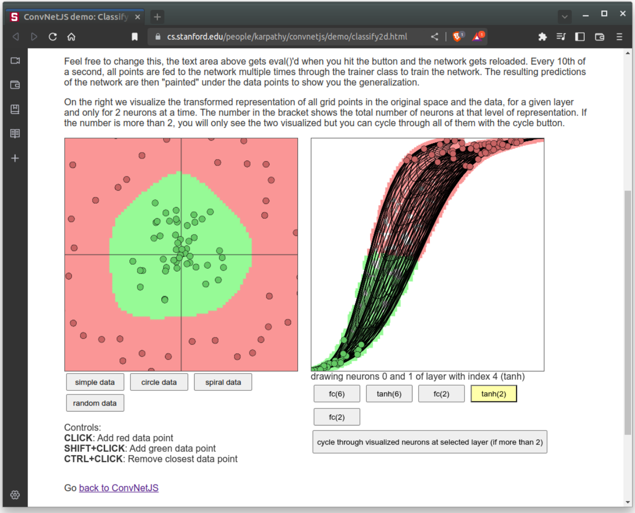
$$\boxed{\text{output\_shape} = \text{input\_shape} - k + 1}$$

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

$$\boxed{\text{output\_shape} = \text{input\_shape}}$$

CNN

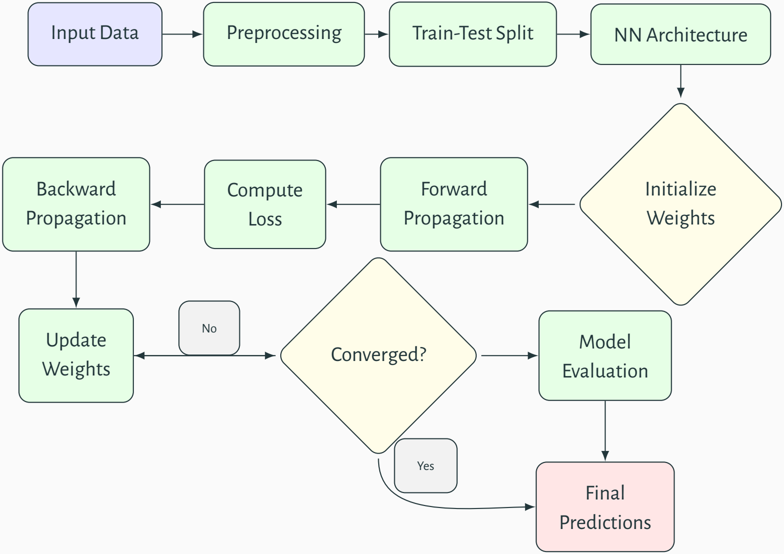
ConvNetJS Demo



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

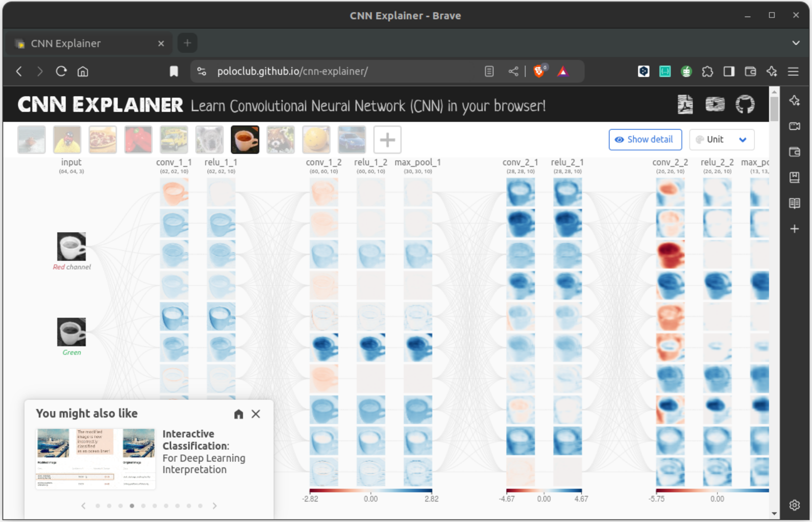
CNN

Neural Network Learning Algorithm



CNN

CNN Explainer



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

CNN

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

⟶ *cnn* ⟶ *cnn.jl*

⟶ *cnn* ⟶ *cnn.ipynb*

VAE

Motivating Factors & Use Cases

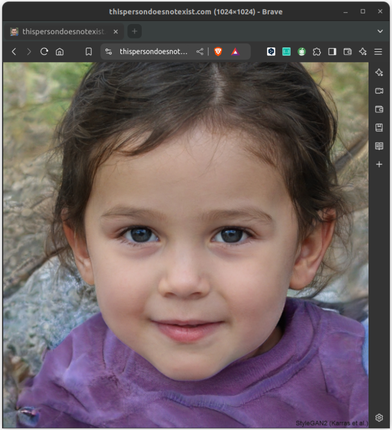
- A **Variational Autoencoder (VAE)** is a type of deep learning model that is used to learn latent representations of data. It is a generative model, which means that it can generate new samples of data that are similar to the training data.
- **VAEs** are trained to encode the data into a low-dimensional 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.

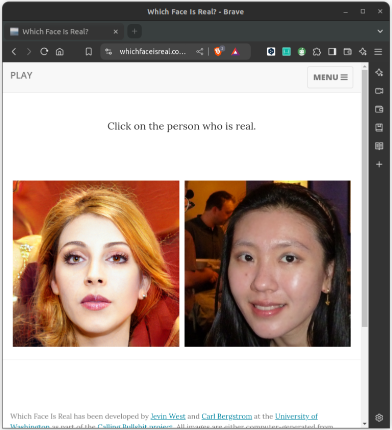
Generative modeling | Anomaly detection | Data compression

Representation learning | Semi-supervised learning

VAE

DEMOS





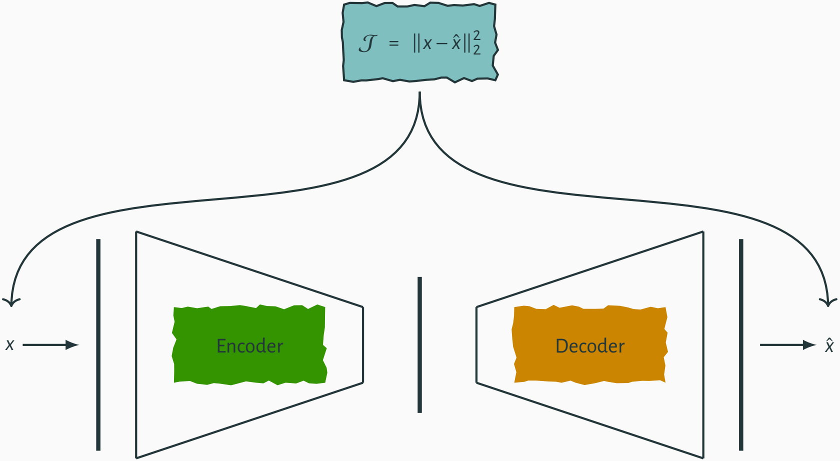
https://thispersondoesnotexist.com/

https://www.whichfaceisreal.com

VAE

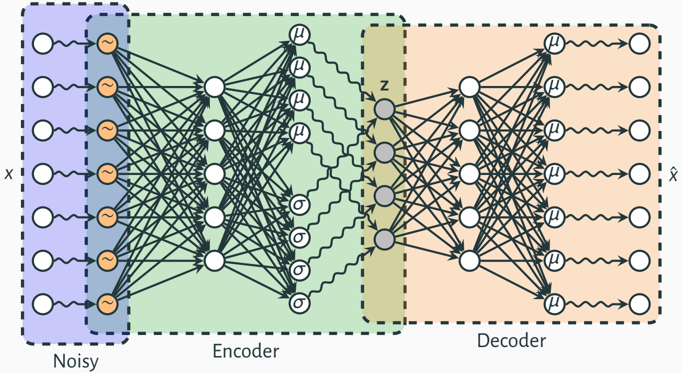
Loss of Vanilla Autoencoder

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

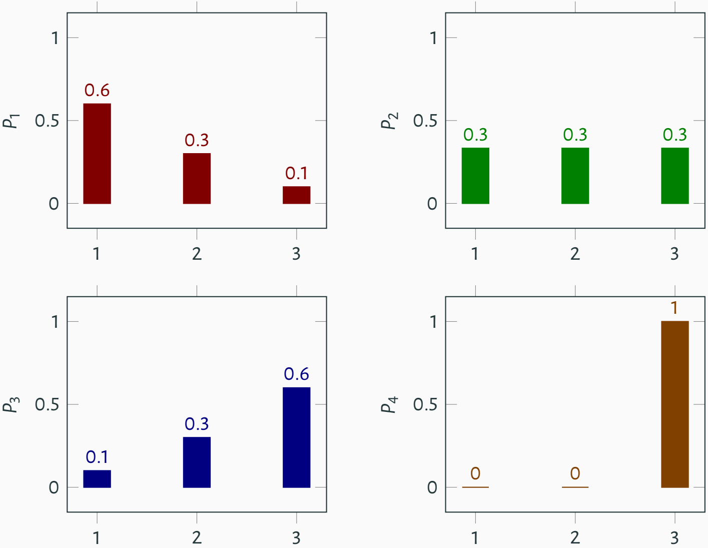
Task:
Consider the following four discrete probability distributions over the support {1, 2, 3}:

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

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

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:
  - $\ln(1.8) \approx 0.588$
  - $\ln(0.9) = -0.105$
  - $\ln(0.3) \approx -1.204$
- ✓ 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, $D_{\mathcal{KL}}\left(P_1 || P_2\right) \approx 0.201$ nats.

VAE

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

$$\mathcal{JS}\left(\mathrm{P}_1, \mathrm{P}_2\right) = \frac{1}{2}\mathrm{D}_{\mathcal{KL}}\left(\mathrm{P}_1 \| \mathrm{M}\right) + \frac{1}{2}\mathrm{D}_{\mathcal{KL}}\left(\mathrm{P}_2 \| \mathrm{M}\right)$$

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

VAE

Let's compute the average distribution M:

- $M(1) = \dfrac{0.6 + 1/3}{2} \approx \dfrac{0.6 + 0.333}{2} = 0.467$
- $M(2) = \dfrac{0.3 + 1/3}{2} \approx \dfrac{0.3 + 0.333}{2} = 0.317$
- $M(3) = \dfrac{0.1 + 1/3}{2} \approx \dfrac{0.1 + 0.333}{2} = 0.217$

$$\boxed{D_{\mathcal{KL}}\left(P_1 \| M\right) \approx 0.056} \qquad \boxed{D_{\mathcal{KL}}\left(P_2 \| M\right) \approx 0.047}$$

Finally:

$$\mathcal{JS}\left(\mathrm{P}_1, \mathrm{P}_2\right) = \frac{1}{2}(0.058) + \frac{1}{2}(0.047) \approx 0.053$$

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

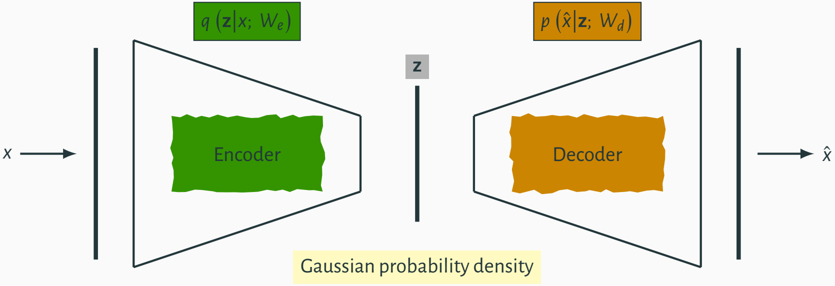
VAE

Loss of Variational Autoencoder

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

Expected negative log likelihood term wrt to encoder distribution

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



VAE

$D_{\mathcal{KL}}$ Loss Derivation

In a **VAE**, the latent vector **z** is calculated by:

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

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

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

The latent prior is given by

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

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

▸ Stack Exchange

VAE

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

⟶ *vae* ⟶ *vae.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.

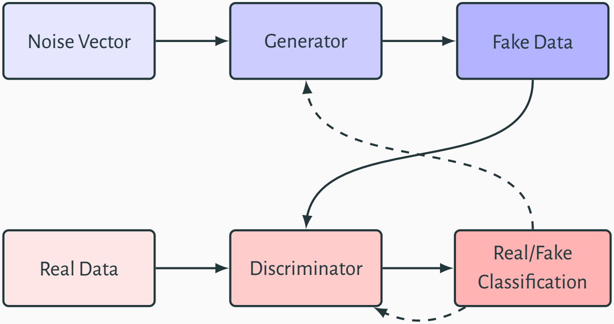
Image generation | Image style transfer | Text-to-image synthesis

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

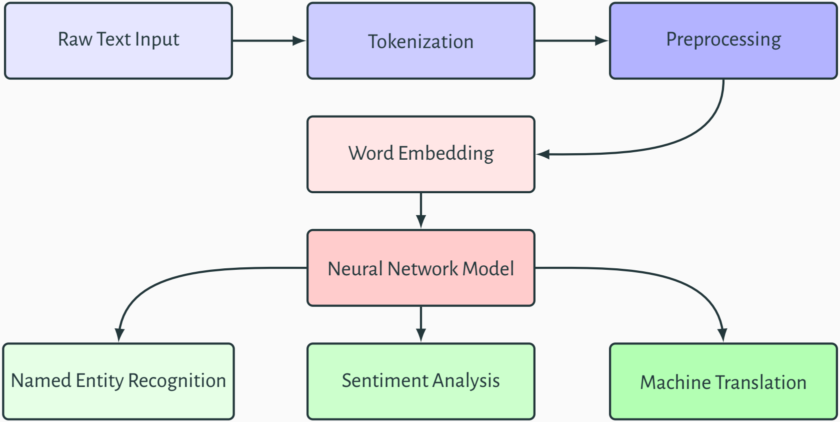
- **Natural Language Processing (NLP)** is a field of artificial intelligence and computer science that focuses on the interaction between computers and humans using natural language.
- **NLP** involves the development of algorithms and models that can understand, interpret, and generate human language.
- **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*

⟶ *nlp* ⟶ *nlp.ipynb*

Transfer Learning

Transfer Learning

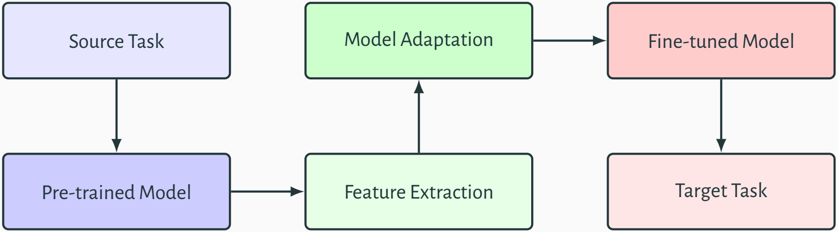
Driving Forces & Use Cases

- **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

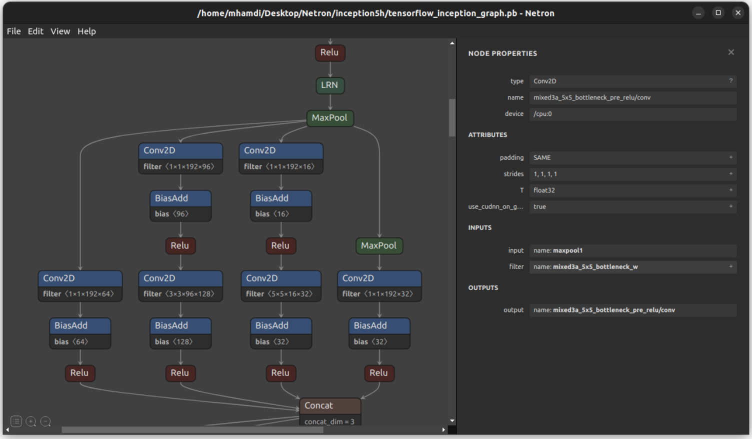
Transfer Learning

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*

⟶ *transfer-learning* ⟶ *transfer-learning.ipynb*

Reinforcement Learning

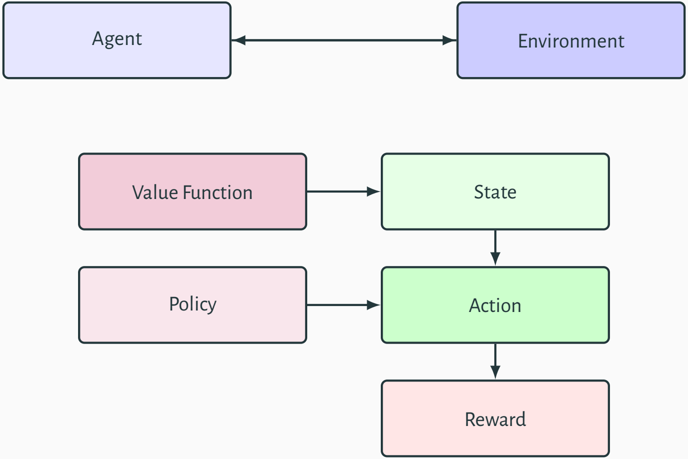
Reinforcement Learning

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

julia

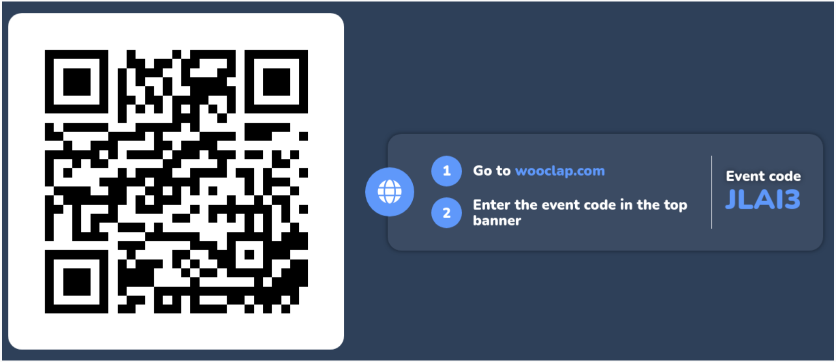
The code is available @ github.com/a-mhamdi/jlai ⟶ *Codes* ⟶ *Julia* ⟶ *Part-3*

⟶ *reinforcement-learning* ⟶ *reinforcement-learning.jl* **Pluto**.jl

⟶ *reinforcement-learning* ⟶ *reinforcement-learning.ipynb* jupyter

Quizzes

Knowledge Check



https://app.wooclap.com/JLAI3

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.