Demystifying Artificial Intelligence Sorcery

(Part 3: Deep Learning)[a]

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

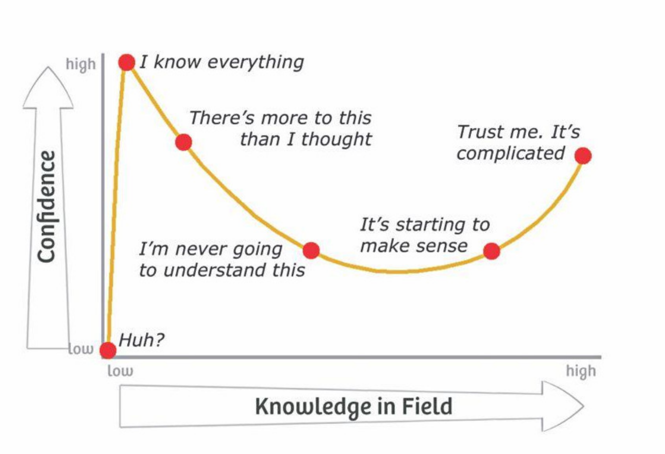
Dr.-Ing. in Electrical Engineering
Senior Lecturer at ISET Bizerte

[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.

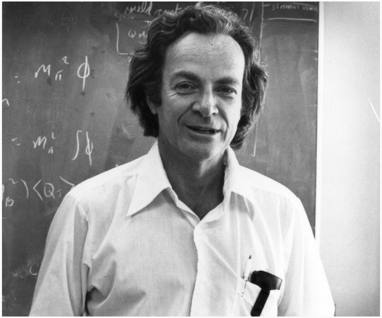
Dunning-Kruger Effect



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. Natural Language Processing
2. CNN & Transfer Learning
3. GAN and VAE
4. Reinforcement Learning
5. Quizzes

REMINDER

Programming Language

julialang.org/

Development Environments

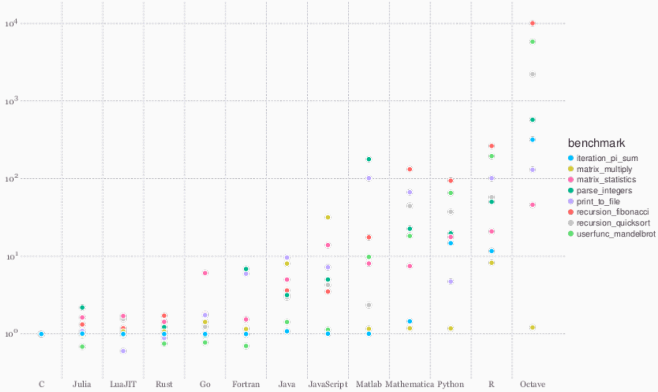
- ▲ $ docker compose up
- ▼ $ docker compose down

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

Julia Micro-Benchmarks (1/2)



https://julialang.org/benchmarks

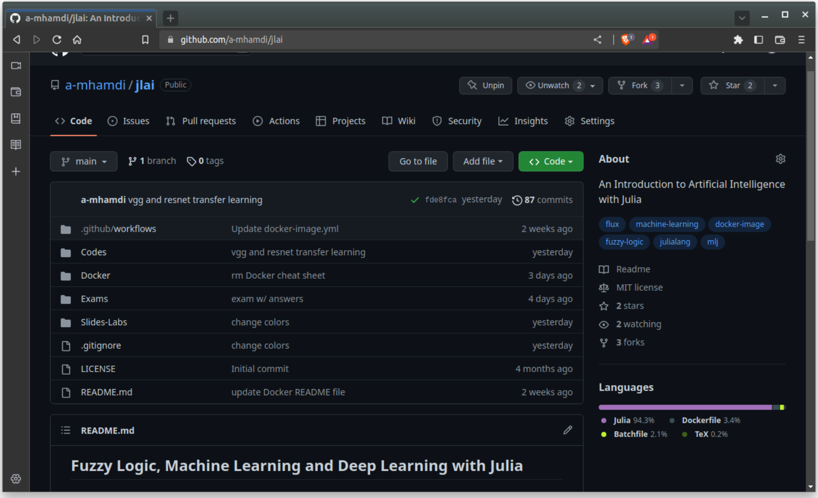
Julia Micro-Benchmarks (2/2)

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

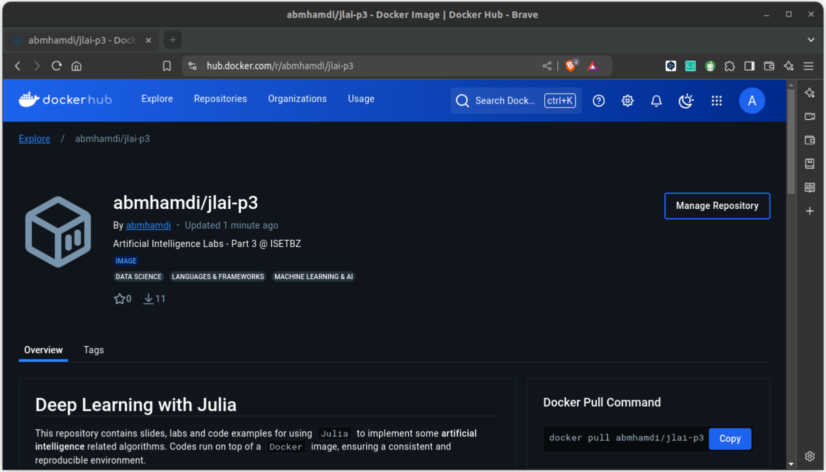
[1]Measure of central tendency expressed as $\left(x_1 \times x_2 \times \cdots \times x_n\right)^{1/n}$

Source Control Management (SCM)



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

Docker Image



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

NLP

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.

NLP

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)

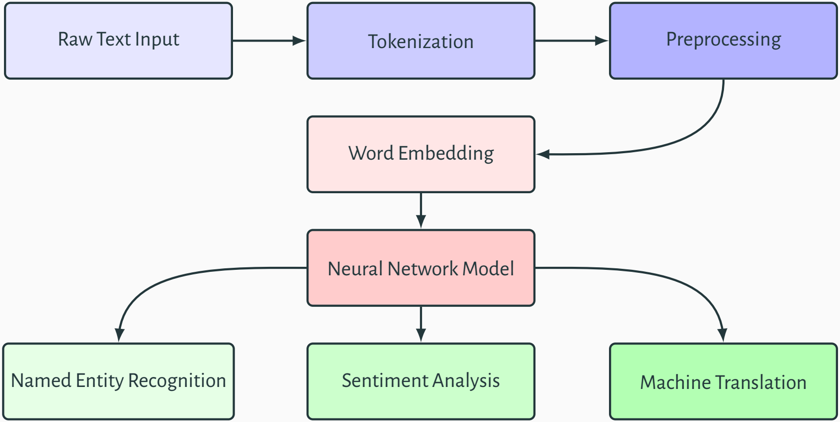
Sentiment analysis Determining the sentiment *(e.g., positive, neutral, negative)* of a piece of text

Machine translation Translating text from one language to another

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

NLP

Pipeline



NLP

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.

NLP

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

$\longrightarrow$ *nlp* $\longrightarrow$ *nlp.jl*

Pluto.jl

$\longrightarrow$ *nlp* $\longrightarrow$ *nlp.ipynb*

jupyter

CNN & Transfer Learning

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.

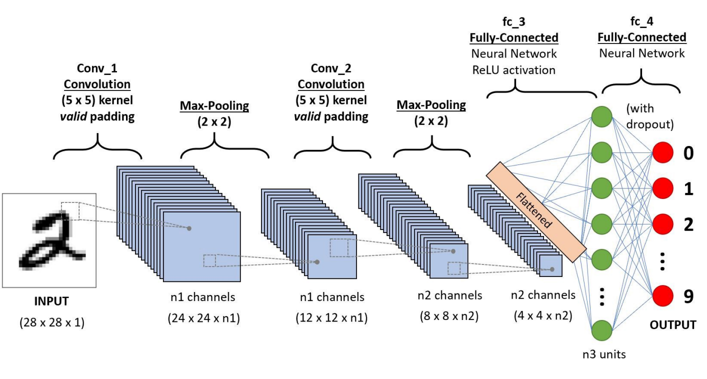
Image classification | Object detection | Image segmentation

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

Robotics | Natural language processing

CNN

Architecture



Source

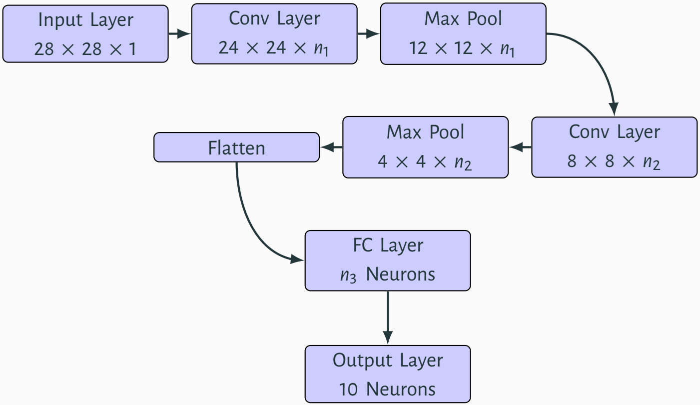
CNN

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

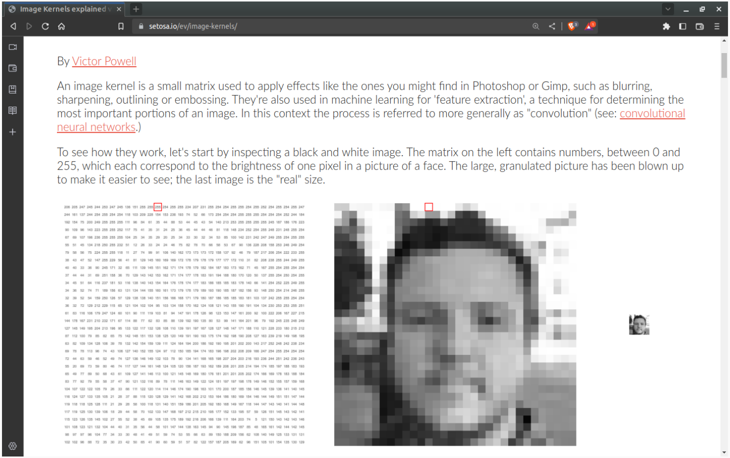
CNN

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

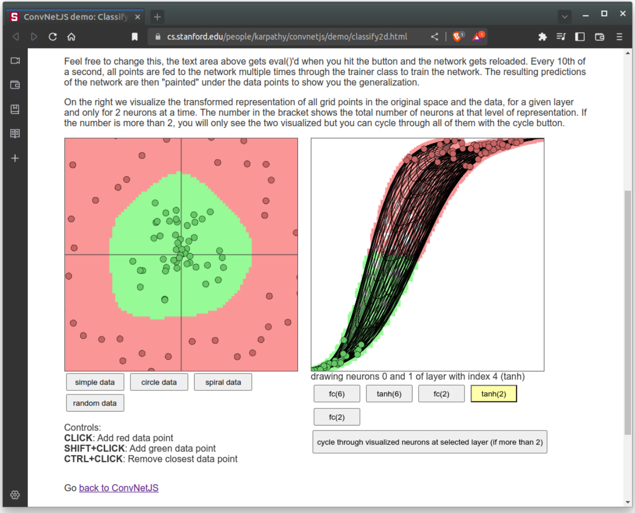
$$\boxed{\text{output\_shape} = \text{input\_shape} - k + 1}$$

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

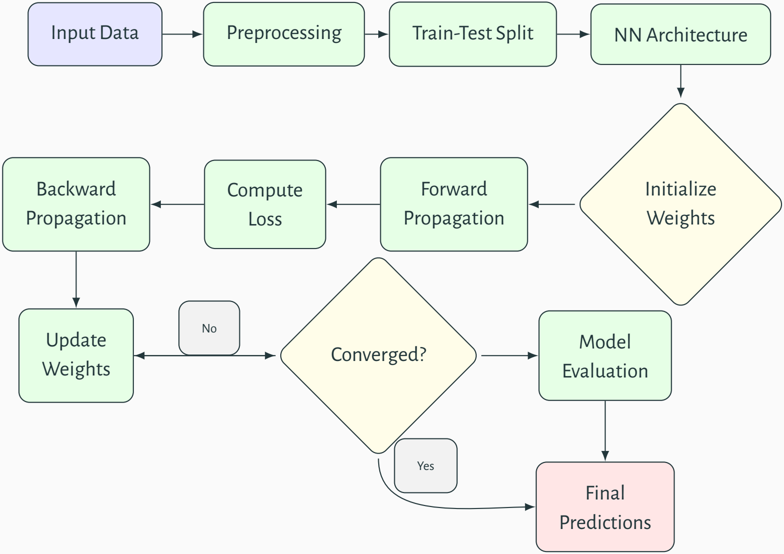
ConvNetJS Demo



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

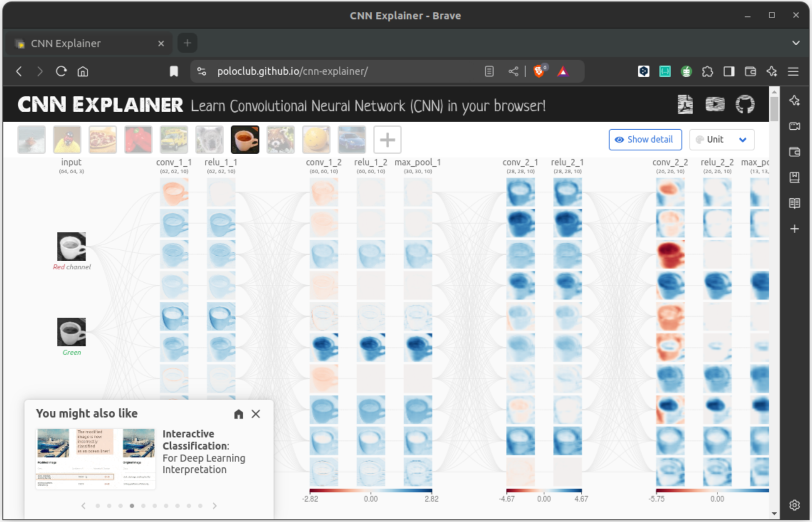
CNN

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*

Pluto.jl

→ *cnn* → *cnn.ipynb*

jupyter

Transfer Learning

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

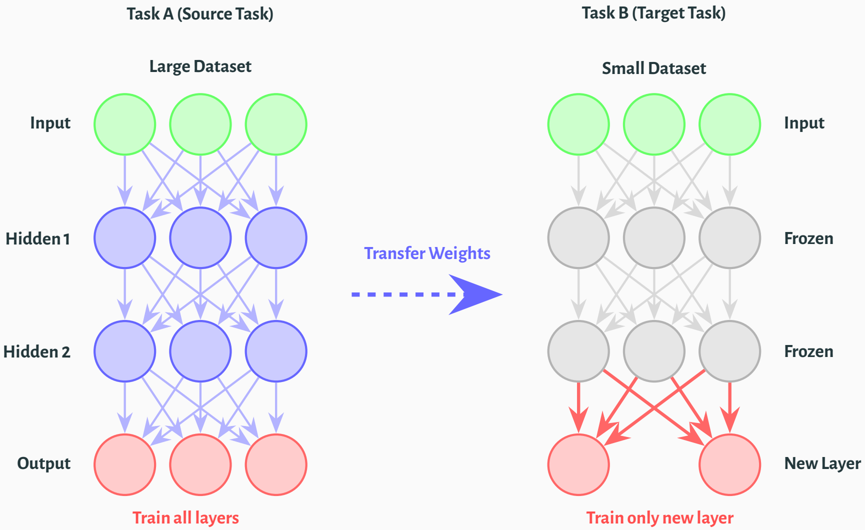
- Example: A model pre-trained on `ImageNet` *(1M+ images, 1000 classes)* can be fine-tuned for specialized tasks like medical image diagnosis or autonomous vehicle perception
- Typical strategy: Freeze early layers *(general feature extractors)*, retrain later layers *(task-specific)*
- Results in better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

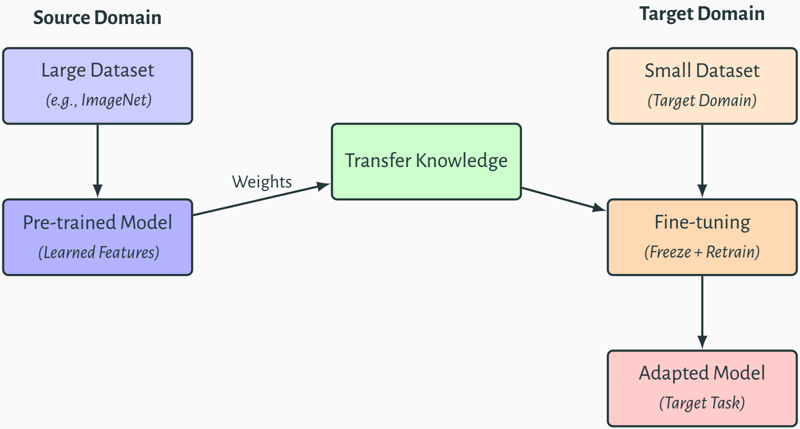
Image classification | Computer vision | Natural language processing | Robotics

Transfer Learning



Transfer Learning

Pipeline



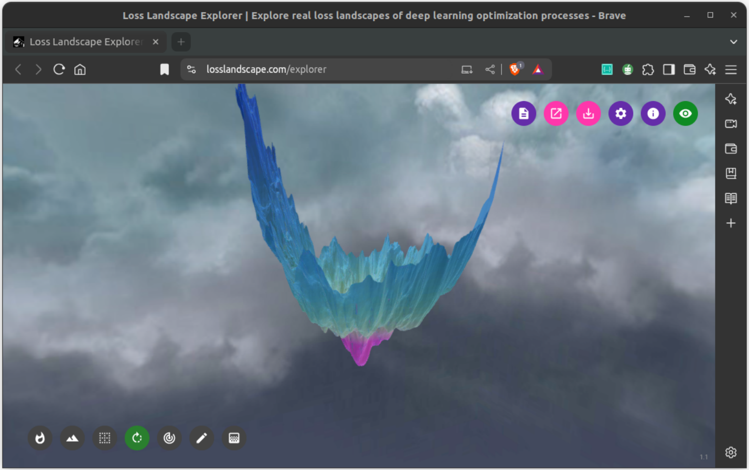
Transfer Learning

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

Transfer Learning

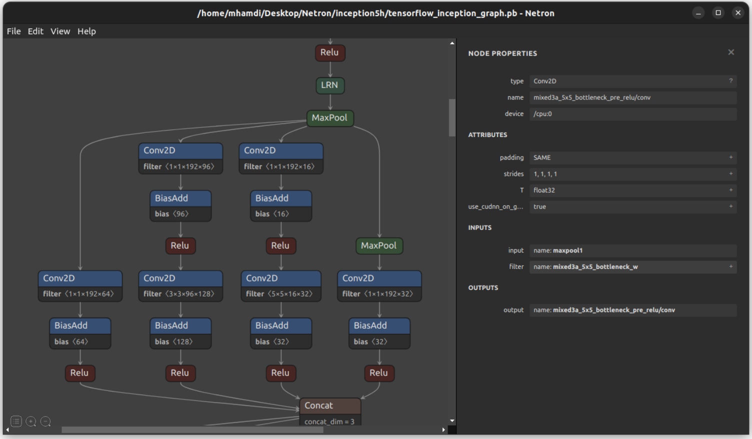
Why Skip Connection



https://losslandscape.com/explorer

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

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

Transfer Learning

julia

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

⟶ *transfer-learning* ⟶ *transfer-learning.jl*

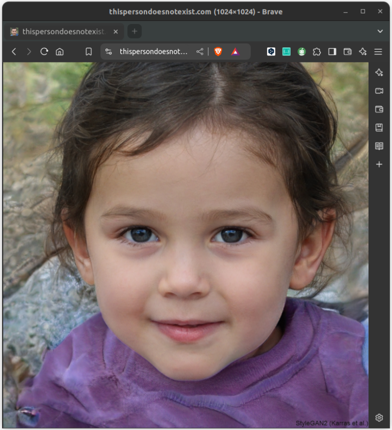
Pluto.jl

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

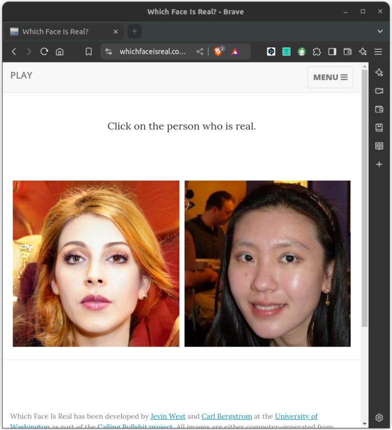
jupyter

GAN and VAE

DEMOS



https://thispersondoesnotexist.com/



https://www.whichfaceisreal.com

GAN

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

$$\boxed{\max_{G} \ln D(G(z))}$$

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

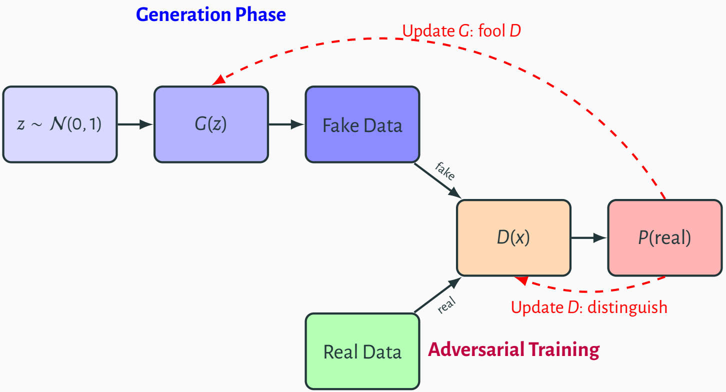
$$\boxed{\max_{D} \left[\ln D(x) + \ln(1 - D(G(z)))\right]}$$

- 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

GAN

Architecture & Use Cases



GAN

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:

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

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

GAN

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:

$$\mathcal{L}_\text{D} = -\left[\ln(\text{D}(\text{x}_\text{real})) + \ln(1 - \text{D}(\text{x}_\text{fake}))\right] \tag{1}$$

3. Generator Loss

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

$$\mathcal{L}_\text{G} = -\ln(\text{D}(\text{x}_\text{fake})) \tag{2}$$

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)?

GAN

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

$$\mathcal{L}_{\mathrm{D}} = -\left[\ln(\mathrm{D}(3)) + \ln(1 - \mathrm{D}(1))\right] \approx 0.96$$

3. Generator Loss

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

4. Gradient Direction

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

GAN

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

→ *gan* → *gan.jl*   **Pluto**.jl

→ *gan* → *gan.ipynb*   jupyter

VAE

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 $\mathcal{N}(0, I)$

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

- Balances faithful reconstruction with smooth, structured latent space

Generative modeling | Anomaly detection | Data compression

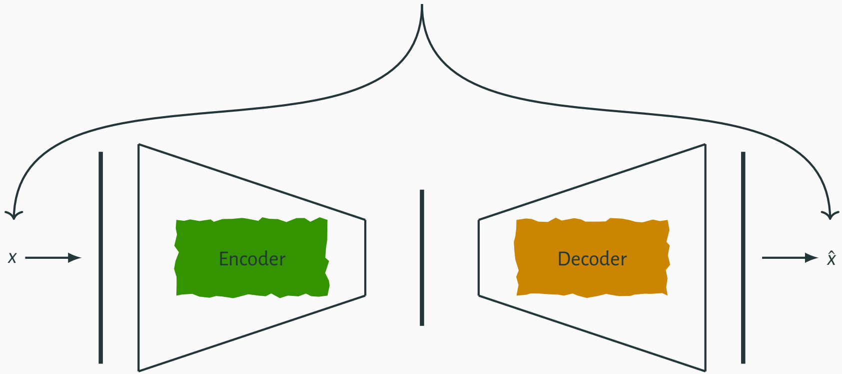
Representation learning | Semi-supervised learning

VAE

Loss of Vanilla Autoencoder

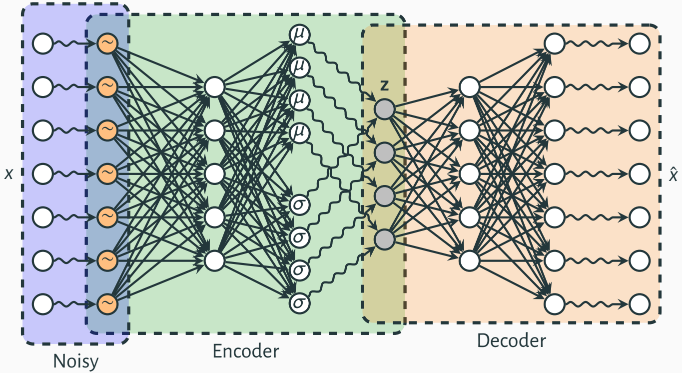
Minimize Squared Error Loss

$$\mathcal{J} = \|x - \hat{x}\|_2^2$$



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.

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

(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)

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

VAE

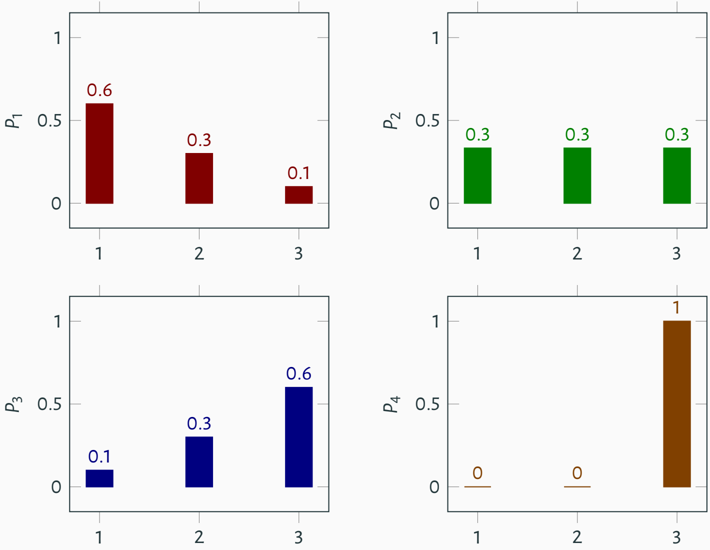
Task:
Consider the following four discrete probability distributions over the support {1, 2, 3}:

- 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



P_1
1
0.5
0
0.6
0.3
0.1
1
2
3
P_2
1
0.5
0
0.3
0.3
0.3
1
2
3
P_3
1
0.5
0
0.1
0.3
0.6
1
2
3
P_4
1
0.5
0
0
0
1
1
2
3

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:

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

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

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.

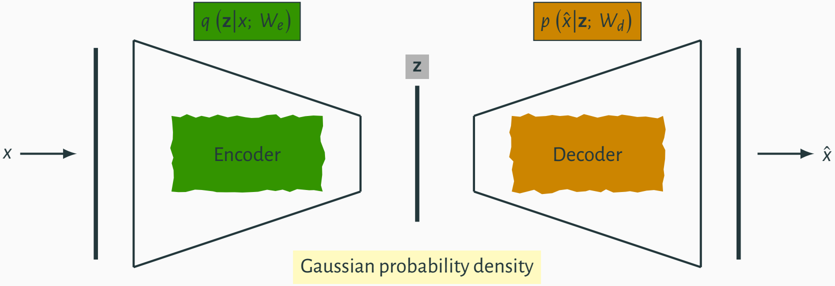
VAE

Loss of Variational Autoencoder

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

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 $\mathbf{z}$ is calculated by:

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

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

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

The latent prior is given by

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

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

▸ Stack Exchange

VAE

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

$\longrightarrow$ *vae* $\longrightarrow$ *vae.jl* Pluto.jl

$\longrightarrow$ *vae* $\longrightarrow$ *vae.ipynb* jupyter

Reinforcement Learning

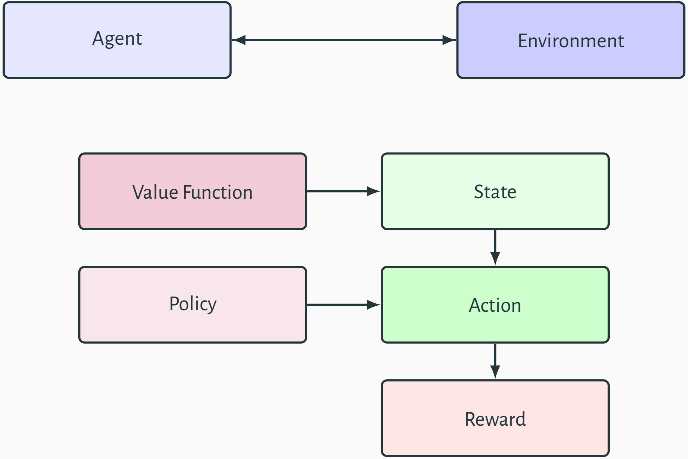
Reinforcement Learning

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

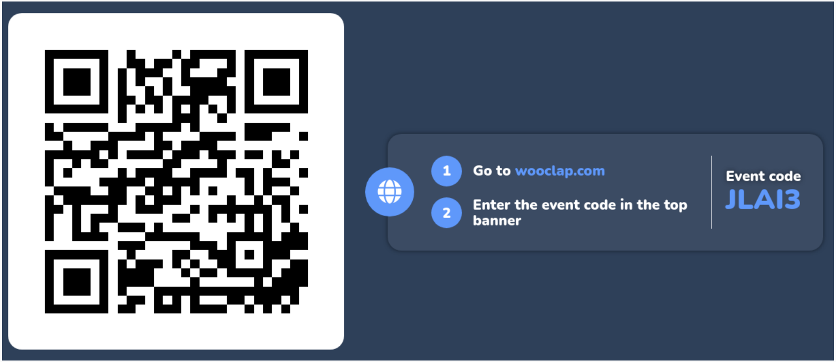
The code is available @ github.com/a-mhamdi/jlai ⟶ *Codes* ⟶ *Julia* ⟶ *Part-3*

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

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

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.