Demystifying Artificial Intelligence Sorcery

(Part 3: Deep Learning)[a]

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

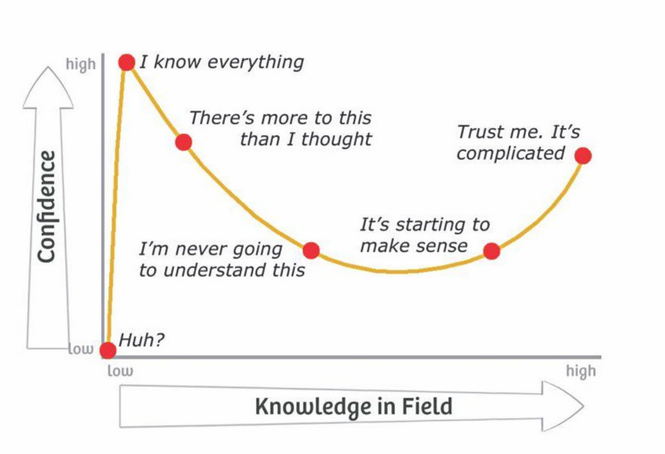
Dr.-Ing. in Electrical Engineering
Senior Lecturer at ISET Bizerte

[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

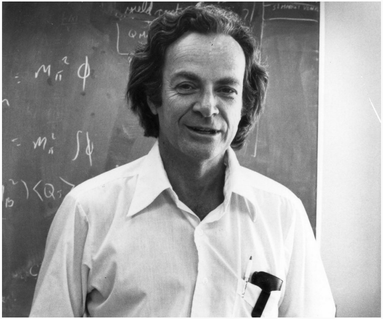


Kruger, J. and Dunning, D. (1999) *Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments.* **J Pers Soc Psychol.** 77(6) pp. 1121–1134.

doi 10.1037/0022-3514.77.6.1121

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

Richard P. Feynman



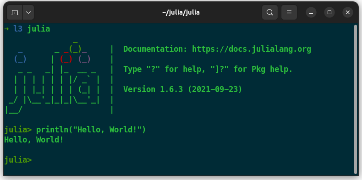
Roadmap

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

REMINDER

Programming Language

julialang.org/

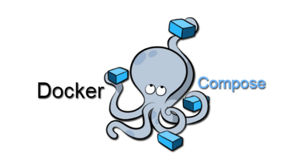


```
→ l3 julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.6.3 (2021-09-23)
 _/ |\__'_|_|_|\__'_|  |
|__/                   |

julia> println("Hello, World!")
Hello, World!

julia>
```

Development Environments

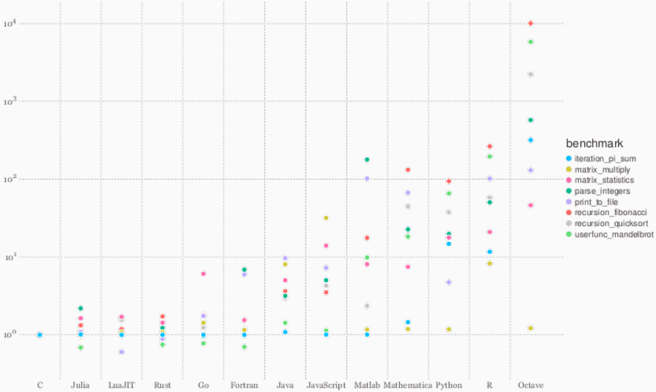


- ▲ $ docker compose up
- ▼ $ docker compose down

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

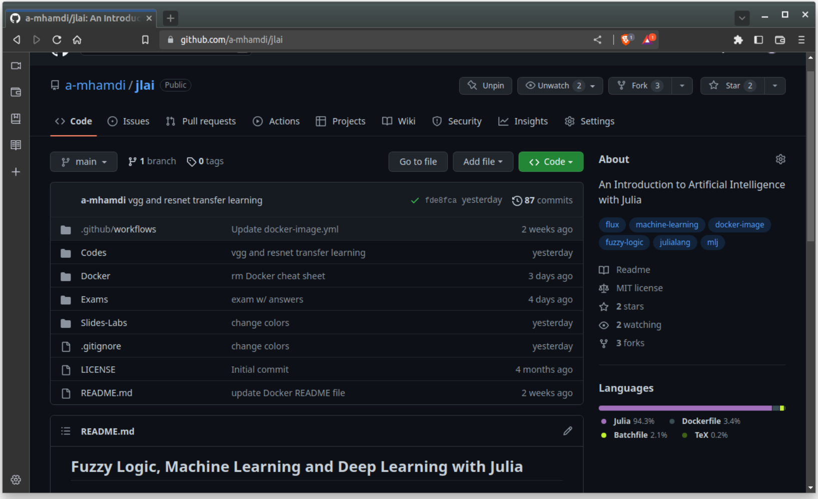
Julia Micro-Benchmarks (2/2)

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)



a-mhamdi / jlai Public

a-mhamdi vgg and resnet transfer learning ✓ fde8fca yesterday 87 commits

.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

README.md

Fuzzy Logic, Machine Learning and Deep Learning with Julia

About

An Introduction to Artificial Intelligence with Julia

flux machine-learning docker-image fuzzy-logic julialang mlj

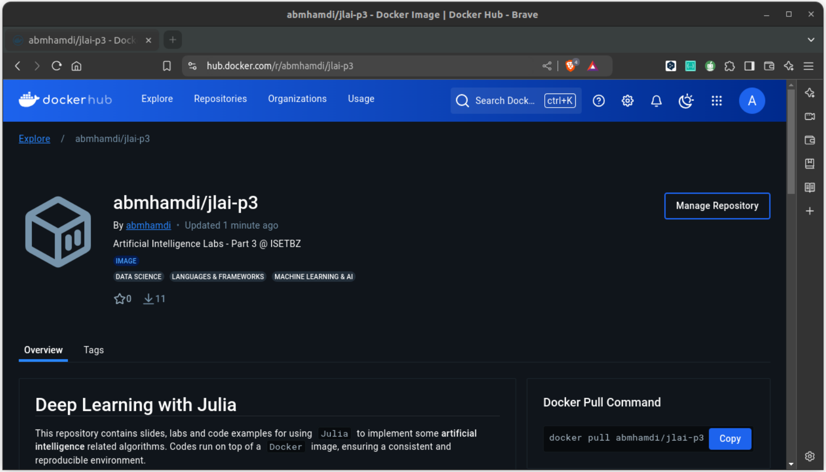
Readme
MIT license
2 stars
2 watching
3 forks

Languages

- Julia 94.3%
- Dockerfile 3.4%
- Batchfile 2.1%
- TeX 0.2%

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

Docker Image



abmhamdi/jlai-p3

By abmhamdi · Updated 1 minute ago

Artificial Intelligence Labs - Part 3 @ ISETBZ

IMAGE

DATA SCIENCE LANGUAGES & FRAMEWORKS MACHINE LEARNING & AI

☆0 ↓11

Deep Learning with Julia

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.

Docker Pull Command

```
docker pull abmhamdi/jlai-p3
```

Copy

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)

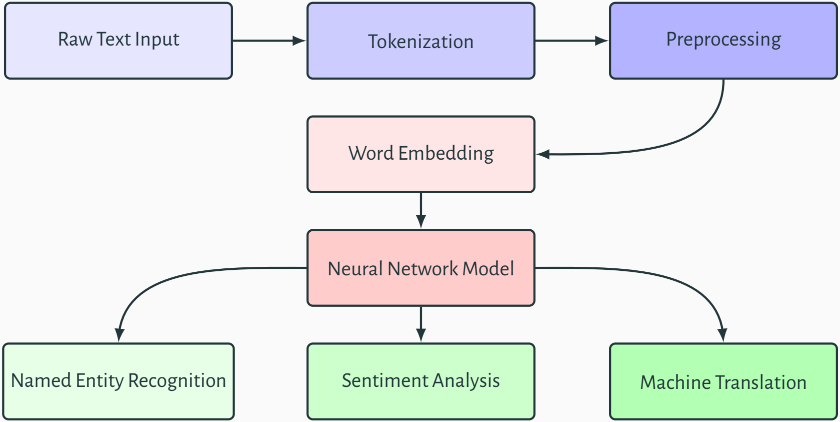
Sentiment analysis Determining the sentiment *(e.g., positive, neutral, negative)* of a piece of text

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 ⟶ *Codes* ⟶ *Julia* ⟶ *Part-3*

⟶ *nlp* ⟶ *nlp.jl* **Pluto.jl**

⟶ *nlp* ⟶ *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:

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.

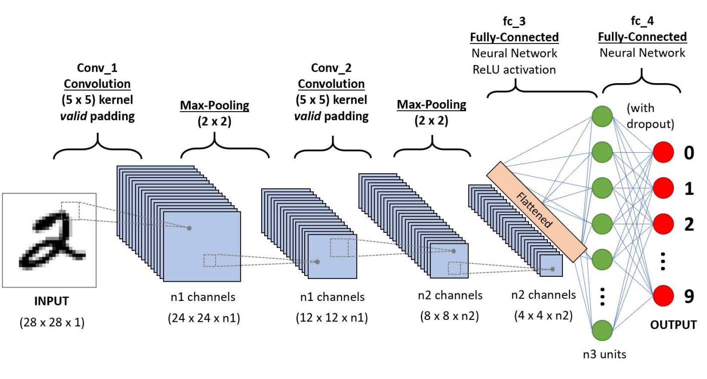
Image classification | Object detection | Image segmentation

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

Robotics | Natural language processing

CNN

Architecture



▶ Source

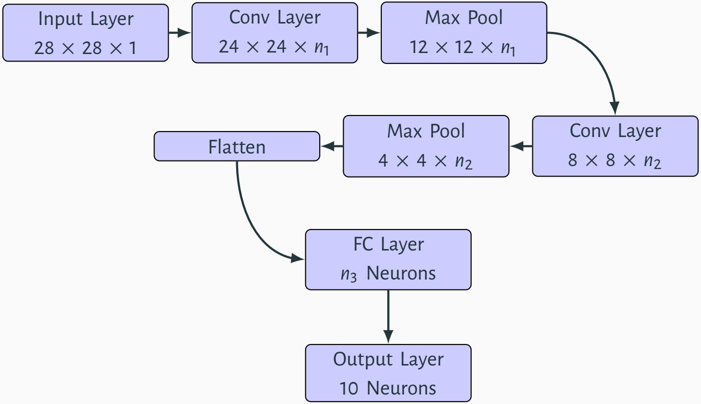
CNN

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

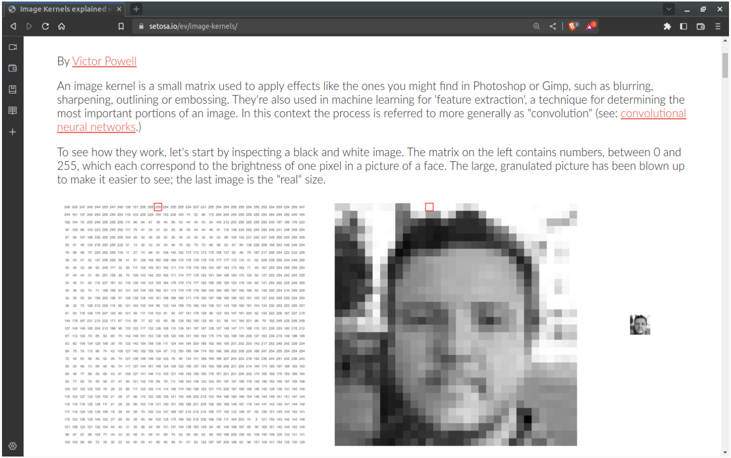
CNN

Pipeline



CNN

Image Kernels



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.

206	205	247	245	244	253	247	245	136	151	255	255	255	254	255	255	234	207	231	255	254	254	255	255	254	255	252	254	255	254	255	247
244	161	137	244	254	255	254	254	118	103	209	228	154	153	236	193	74	52	66	173	254	254	254	255	255	255	254	254	254	252	244	184
192	154	75	200	249	255	255	255	111	96	84	61	35	44	88	53	44	45	43	54	140	213	253	255	255	255	255	245	187	186	176	223
90	109	96	143	223	255	255	252	117	75	41	35	31	24	25	36	45	44	44	46	81	118	148	234	252	254	255	248	231	248	255	254
67	69	107	196	236	255	255	255	104	25	34	35	29	20	25	34	33	30	32	34	53	85	100	142	231	242	247	249	255	254	255	255
55	51	45	134	218	250	255	232	51	12	26	33	24	24	46	75	82	78	70	66	58	53	67	90	136	228	206	158	253	246	249	254
79	58	56	75	224	255	255	118	11	27	74	99	91	106	140	162	173	173	173	172	158	137	92	46	79	187	217	206	254	222	233	255
38	43	47	52	147	255	229	56	41	81	129	145	160	169	169	172	178	179	178	179	177	177	172	110	31	82	208	238	255	244	249	255
40	40	33	36	90	245	171	32	65	111	139	145	151	162	171	174	178	179	182	184	187	183	173	162	71	45	167	255	254	255	254	254
37	44	44	31	69	251	158	36	70	129	143	142	153	162	171	174	177	178	183	191	194	188	180	170	120	50	137	255	254	250	254	255
34	45	51	64	116	237	181	53	116	138	140	143	154	164	176	178	174	177	183	186	185	185	183	178	140	66	141	254	252	225	249	255
34	36	52	74	71	189	156	63	131	134	144	155	160	161	173	179	178	179	189	193	190	185	187	182	156	93	148	250	254	214	246	255
32	39	52	54	159	250	126	57	129	138	138	140	151	156	166	168	171	179	180	187	186	185	185	183	181	103	137	242	255	255	254	254
36	32	72	129	212	229	115	65	121	104	102	104	95	103	134	158	170	162	124	108	121	143	155	190	191	104	134	230	253	253	255	251
61	83	116	106	179	247	124	60	101	90	111	119	103	81	94	147	191	178	126	98	123	153	147	161	200	92	100	222	206	167	227	215
144	178	167	231	210	232	171	67	114	88	77	62	83	85	88	139	192	190	135	80	53	99	141	164	201	96	79	192	245	235	248	249
127	145	149	195	204	213	196	95	133	122	117	132	126	108	110	139	191	197	167	128	127	148	147	171	188	110	121	228	233	180	215	212
87	112	100	79	85	82	65	75	142	148	151	153	138	125	120	149	191	190	193	175	174	192	198	190	208	127	183	239	219	149	198	195
63	82	109	134	128	106	39	78	132	142	154	159	139	111	124	164	194	200	186	192	190	195	201	202	200	143	217	252	248	242	238	234
69	78	78	113	96	74	43	106	127	140	152	155	124	97	112	150	185	194	174	183	196	198	202	208	209	166	247	254	255	254	254	254
72	44	63	59	46	52	49	74	127	136	146	149	132	103	78	90	134	141	168	165	196	207	204	203	216	193	236	244	251	242	236	243
55	20	69	73	59	80	46	74	117	127	144	161	148	124	105	120	156	187	193	162	189	206	201	205	214	194	174	185	197	188	183	193
65	49	77	89	50	68	43	61	109	127	141	146	113	100	121	145	148	169	180	176	181	201	201	205	202	174	166	169	178	183	188	184
83	77	92	79	55	58	37	47	90	121	132	116	89	78	111	146	163	149	122	124	181	197	197	198	178	149	146	152	155	157	159	168
104	107	122	122	105	79	26	33	66	111	122	120	114	114	146	174	190	196	163	101	170	200	187	185	156	146	145	139	136	141	140	145
116	124	127	133	135	105	21	28	37	88	115	120	128	129	141	142	168	202	212	153	164	186	180	169	154	146	144	149	151	151	147	144
119	118	118	125	128	111	21	29	28	58	100	118	131	140	151	159	186	201	205	192	180	168	149	167	118	144	147	143	140	141	144	148
117	119	125	130	139	106	18	29	44	58	70	102	133	147	168	197	212	215	210	195	177	152	133	195	57	59	126	151	145	143	142	141
115	123	126	135	145	102	27	55	52	38	45	69	105	135	175	189	192	216	206	166	139	111	164	203	74	5	121	150	143	142	143	146
101	108	123	121	132	104	44	40	31	35	56	44	58	101	147	144	138	163	145	94	90	145	196	187	85	48	165	161	142	144	142	145
98	97	97	96	104	77	34	33	30	48	41	49	51	59	74	53	55	66	63	89	150	188	209	156	62	108	140	149	125	133	131	131
102	102	96	88	72	35	30	23	42	50	65	41	90	60	59	51	57	82	122	157	187	205	169	62	96	151	105	101	154	135	130	129

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

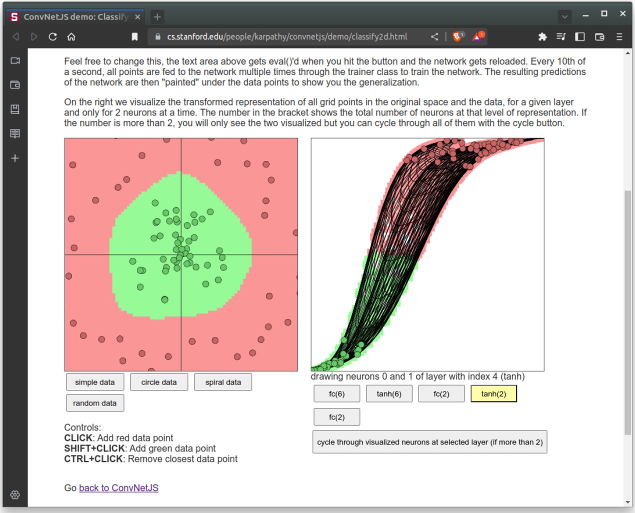
$$\boxed{\text{output\_shape} = \text{input\_shape} - k + 1}$$

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



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.

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)

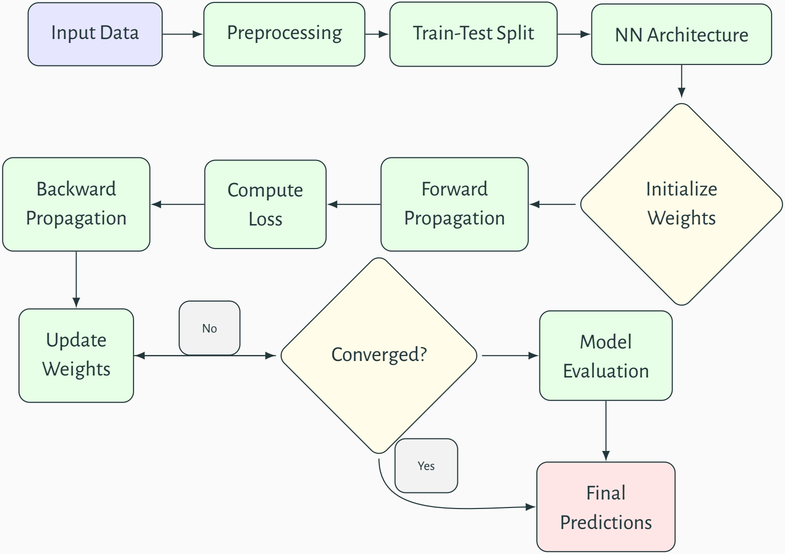
Controls:
CLICK: Add red data point
SHIFT+CLICK: Add green data point
CTRL+CLICK: Remove closest data point

Go back to ConvNetJS

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

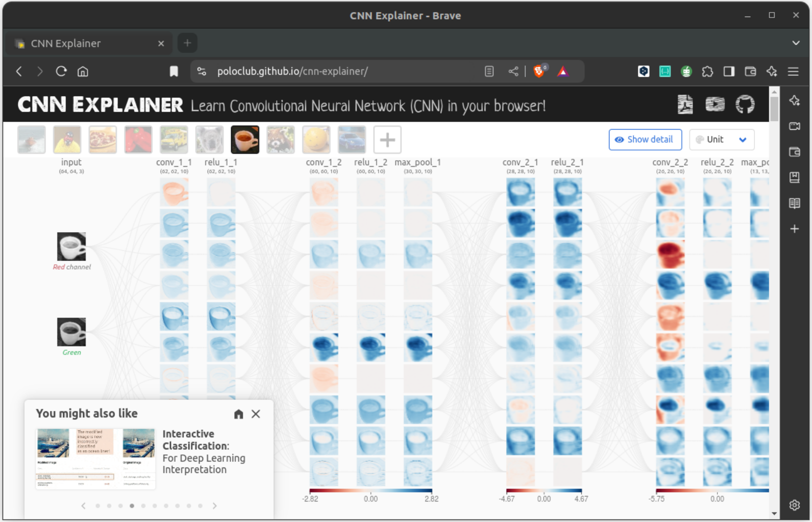
CNN

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

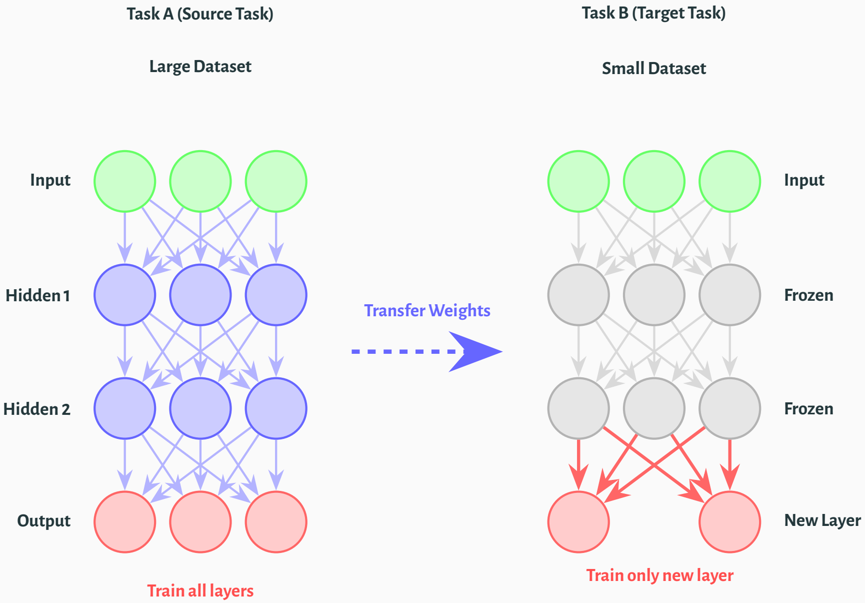
- Example: A model pre-trained on `ImageNet` *(1M+ images, 1000 classes)* can be fine-tuned for specialized tasks like medical image diagnosis or autonomous vehicle perception
- Typical strategy: Freeze early layers *(general feature extractors)*, retrain later layers *(task-specific)*
- Results in better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

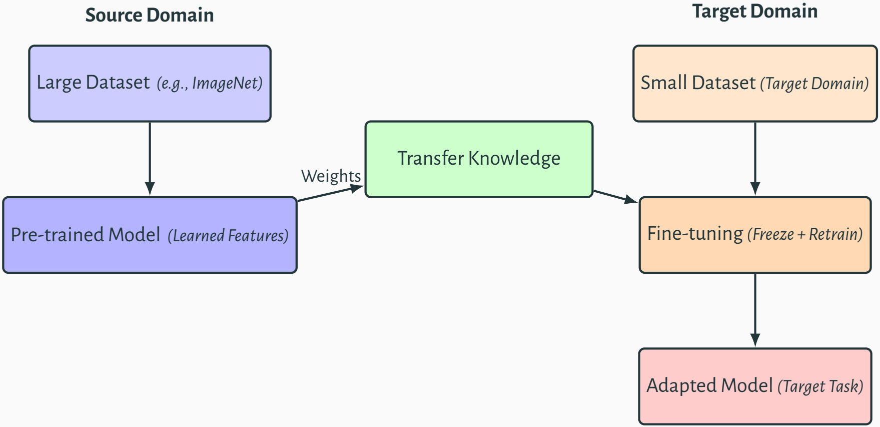
Image classification | Computer vision | Natural language processing | Robotics

Transfer Learning



Transfer Learning

Pipeline


Source Domain
Target Domain
Large Dataset *(e.g., ImageNet)*
Pre-trained Model *(Learned Features)*
Weights
Transfer Knowledge
Small Dataset *(Target Domain)*
Fine-tuning *(Freeze + Retrain)*
Adapted Model *(Target Task)*

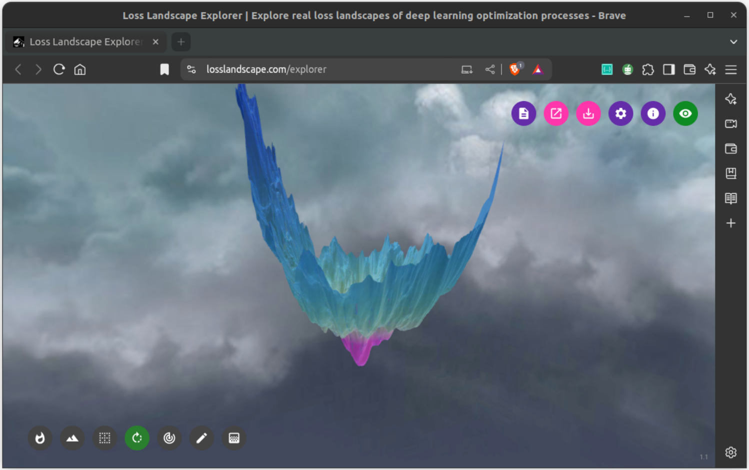
Transfer Learning

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

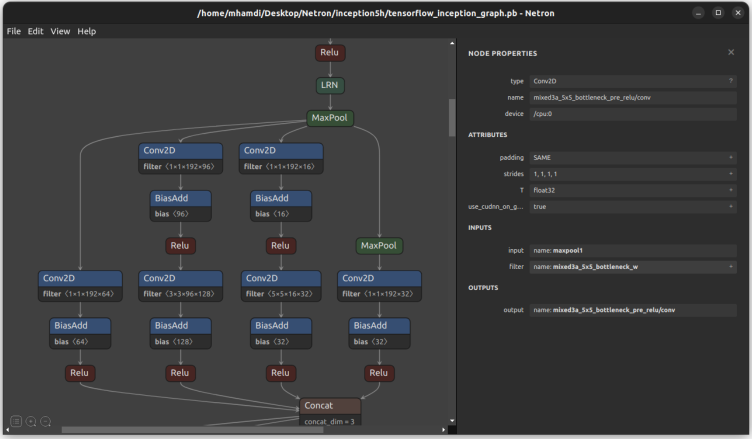
Why Skip Connection


Loss Landscape Explorer | Explore real loss landscapes of deep learning optimization processes - Brave
Loss Landscape Explorer
losslandscape.com/explorer
1.1

https://losslandscape.com/explorer

Transfer Learning

Netron


/home/mhamdi/Desktop/Netron/inception5h/tensorflow_inception_graph.pb - Netron

NODE PROPERTIES

type: Conv2D

name: mixed3a_5x5_bottleneck_pre_relu/conv

device: /cpu:0

ATTRIBUTES

padding: SAME

strides: 1, 1, 1, 1

T: float32

use_cudnn_on_g...: true

INPUTS

input: name: **maxpool1**

filter: name: **mixed3a_5x5_bottleneck_w**

OUTPUTS

output: name: **mixed3a_5x5_bottleneck_pre_relu/conv**

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

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

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

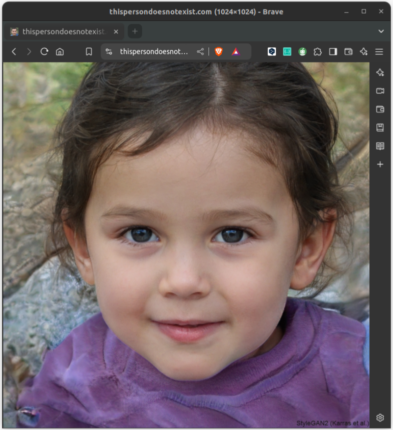
Pluto.jl

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

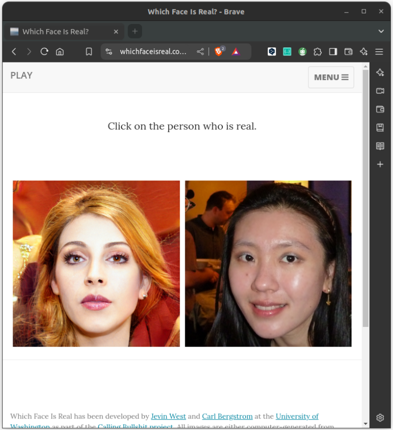
jupyter

GAN and VAE

DEMOS


thispersondoesnotexist.com (1024×1024) - Brave
thispersondoesnotexist
thispersondoesnot...
StyleGAN2 (Karras et al.)

https://thispersondoesnotexist.com/



Click on the person who is real.

Which Face Is Real has been developed by Jevin West and Carl Bergstrom at the University of Washington as part of the Calling Bullshit project. All images are either computer-generated from

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

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

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

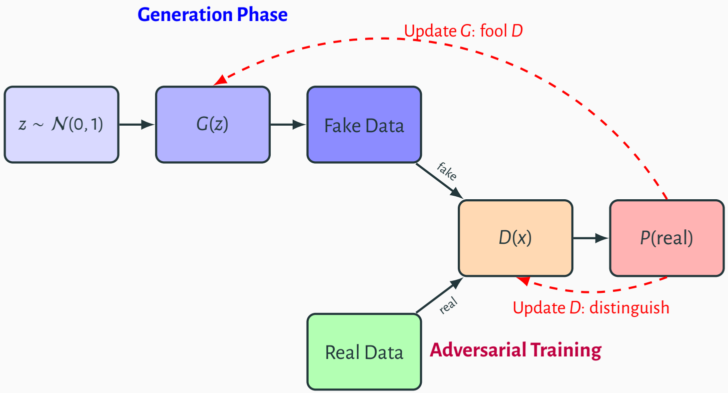
$$\boxed{\max_{D} \left[\log D(x) + \log(1 - D(G(z)))\right]}$$

- 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:

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

3. Generator Loss

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

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

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

$$\mathcal{L}_\mathrm{D} = -\left[\log(\mathrm{D}(3)) + \log(1 - \mathrm{D}(1))\right] \approx -2(-0.48) = 0.96$$

3. Generator Loss

$$\begin{aligned}\mathcal{L}_\mathrm{G} &= -\log(\mathrm{D}(\mathrm{G}(1))) = -\log(\mathrm{D}(3)) \\ &= -\log(0.62) \approx 0.48\end{aligned}$$

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

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

- Balances faithful reconstruction with smooth, structured latent space

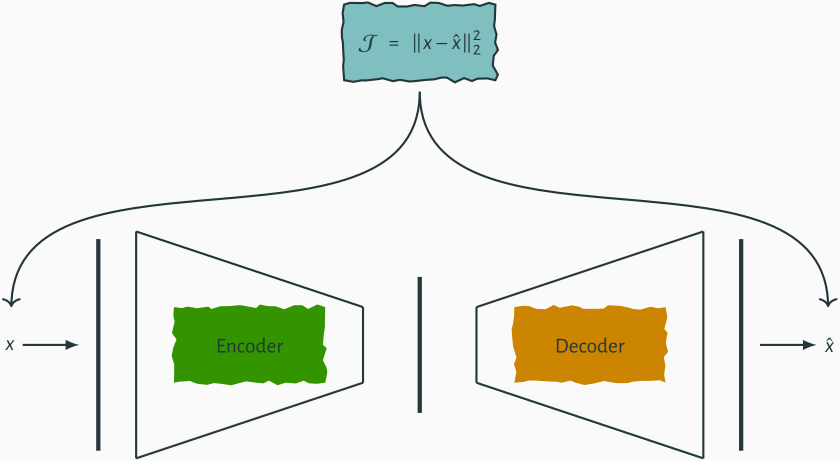
Generative modeling | Anomaly detection | Data compression

Representation learning | Semi-supervised learning

VAE

Loss of Vanilla Autoencoder

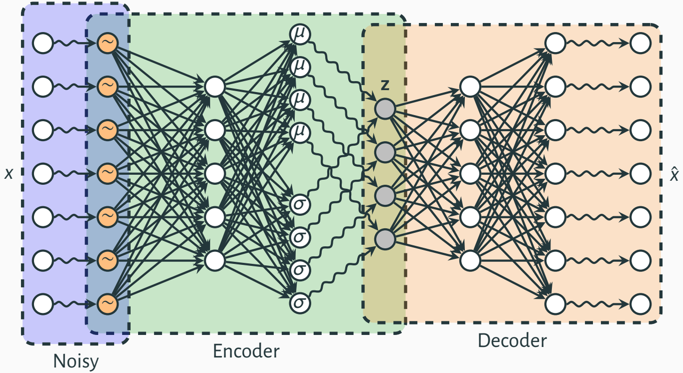
Minimize Squared Error Loss



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

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.

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

(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

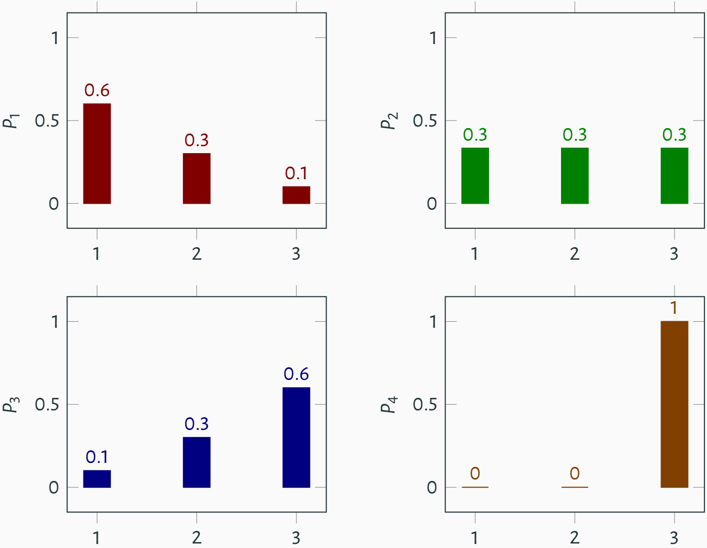
Task:
Consider the following four discrete probability distributions over the support {1, 2, 3}:

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:

$$\mathrm{D}_{\mathcal{KL}}\left(\mathrm{P}_1 \| \mathrm{P}_2\right) = \sum_{x=1}^{3} \mathrm{P}_1(x) \ln\left(\frac{\mathrm{P}_1(x)}{\mathrm{P}_2(x)}\right)$$

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:

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

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

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.

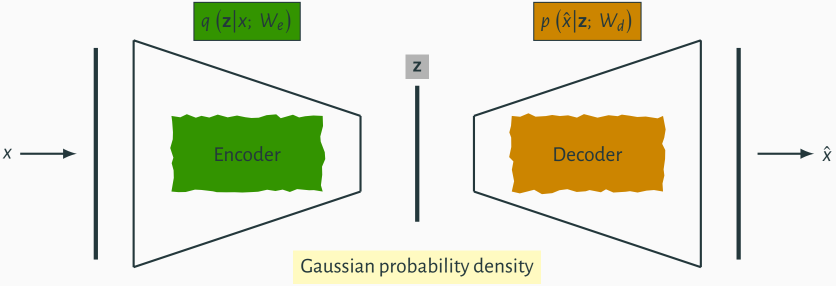
VAE

Loss of Variational Autoencoder

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

Expected negative log likelihood term wrt to encoder distribution

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



$q\left(\mathbf{z}|x;\, W_e\right)$

$p\left(\hat{x}|\mathbf{z};\, W_d\right)$

$\mathbf{z}$

x

Encoder

Decoder

$\hat{x}$

Gaussian probability density

VAE

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

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

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

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

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

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

▸ Stack Exchange

VAE

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

→ *vae* → *vae.jl* Pluto.jl

→ *vae* → *vae.ipynb* jupyter

Reinforcement Learning

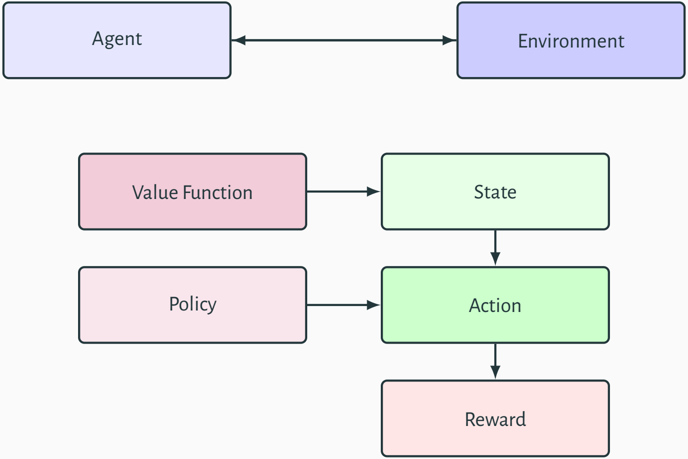
Reinforcement Learning

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

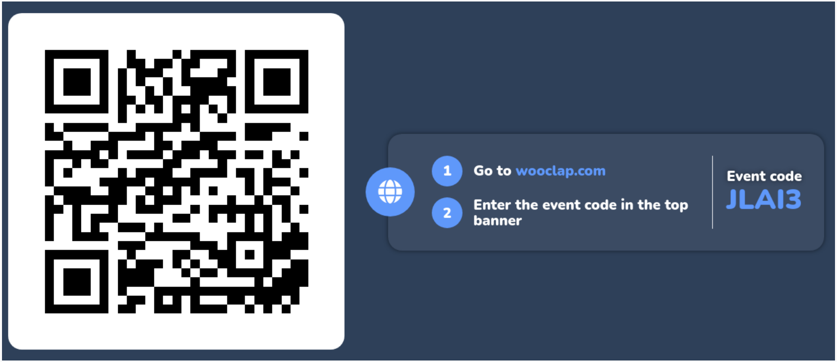
The code is available @ github.com/a-mhamdi/jlai ⟶ *Codes* ⟶ *Julia* ⟶ *Part-3*

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

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

Quizzes

Knowledge Check



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.