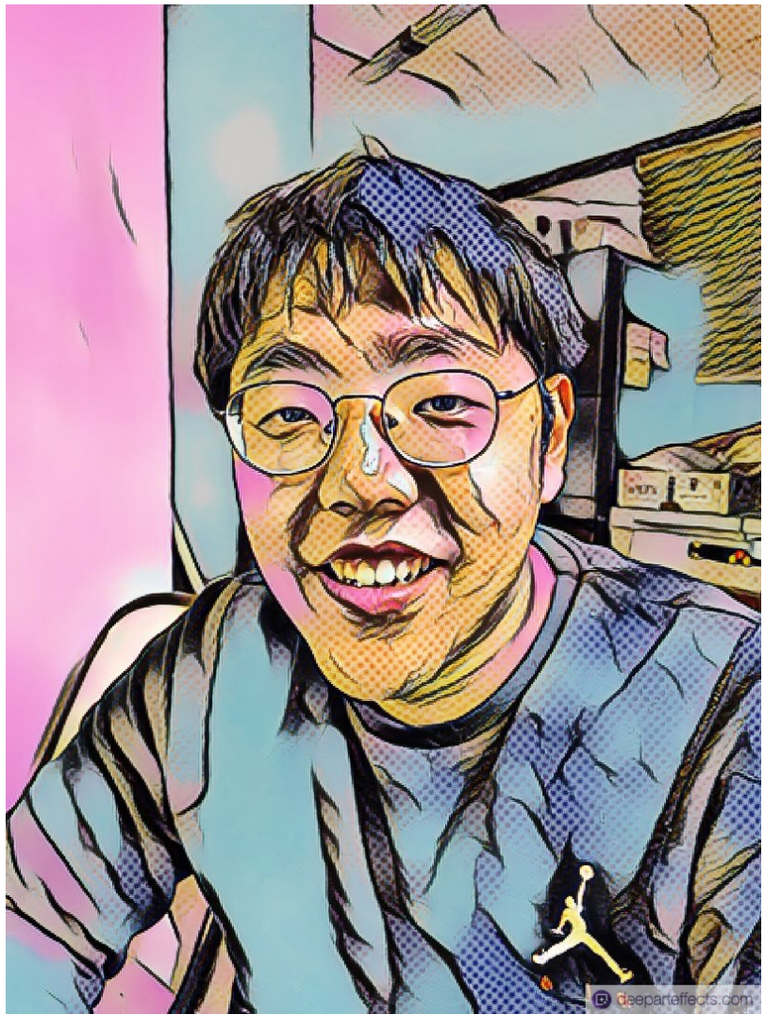
Natural Language Processing with PyTorch

Week 1 딥러닝을 위한 PyTorch 실무환경 구축

Course introduction

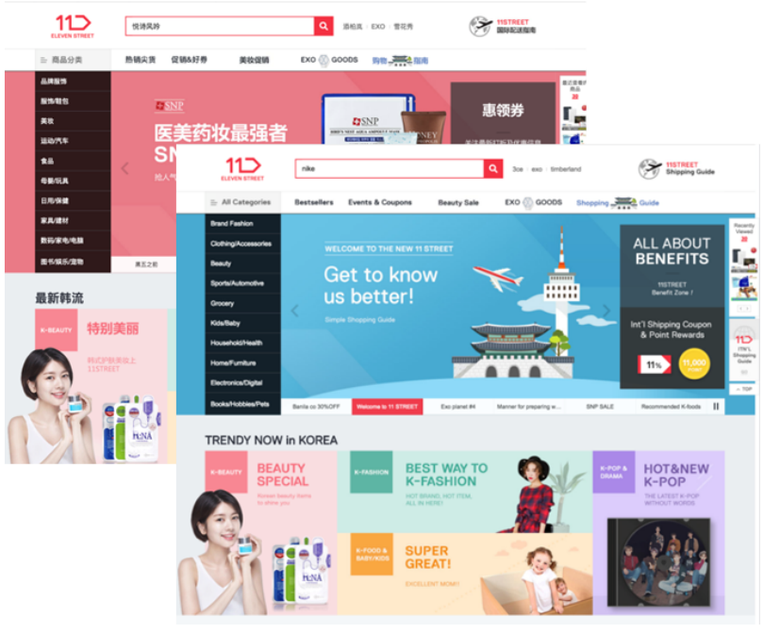
Ki Hyun Kim

- Machine Learning Researcher @ MakinaRocks
- Linkedin: https://www.linkedin.com/in/ki-hyun-kim/
- Github: https://github.com/kh-kim/
- Email: pointzz.ki@gmail.com



Ki Hyun Kim

- Machine Learning Researcher @ SKPlanet
  - Neural Machine Translation
  - 글로벌 11번가
    - 한영/영한, 한중/중한 기계번역
    - 7000만 개 이상의 상품타이틀 번역, 리뷰 실시간 번역
  - SK AI asset 공유
    - SK C&C Aibril: 한중/중한, 한영/영한, 영중/중영 API 제공
    - SK 그룹 한영중 통번역기 API 제공



Ki Hyun Kim

- Machine Learning Engineer @ TMON
  - Recommender System
  - QA-bot
- Researcher @ ETRI
  - Automatic Speech Translation
  - GenieTalk
- BS + MS of CS @ Stony Brook Univ.

오상준

- Deep Learning Engineer @ Deep Bio
  - 병리영상 기반 전립선암 진단모델 연구개발
  - GPU 서버 분산 스케쥴링 시스템 개발
- Co-founder, Research Engineer
  @ QuantumSurf
  - 선물거래 알고리즘을 위한 API 설계 및 UX 개발
  - IPTV 영상품질 예측모델 연구개발
- BS in English Literature @ 한국외국어대학교

오상준

- Github: https://github.com/juneoh
- Email: me@juneoh.net

This lecture is for

To follow the hands-on sessions,

- people who know basic Python programming (data structures, OOP)
  - = 무작정 따라하는 파이썬 프로그래밍 (YouTube)
- people who have heard of the basics of machine learning and/or deep learning
  - = 모두를 위한 딥러닝/머신러닝 or PyTorch Zero to All

This lecture is for

And preferrably, to fully catch the theory,

- people who understand the basic concepts of related mathematics (statistics, probability, calculus, and linear algebra)
  - = Machine Learning by Andrew Ng (Coursera)

But even if not, **just ask!**

What this course does NOT offer

- mere scrap information from all those deep learning YouTubes and blogs on the web
- basic programming and mathematics
  - (왕초보) 프로그래밍 첫걸음 시작하기 (Fast campus)
  - (초보) 파이썬으로 개발 시작하기 (Fast campus)
  - (중고급) 파이썬 완전 정복 CAMP (Fast campus)
  - (초급) 딥러닝을 위한 최적화와 수치해석 CAMP (Fast campus)

What this course does NOT offer

- *everything* deep learning can do
  - (초급) PyTorch로 시작하는 딥러닝 입문 CAMP (Fast campus)
- deeper topics of NLP: seq2seq, neural machine translation
  - (고급) PyTorch를 활용한 자연어처리 심화 CAMP (본 강의와 상호 할인)

What this course DOES offer

- a quick piece-together understanding of core concepts in deep NLP for intuition and insight
- hands-on sessions with working project codebase to freely exploit
- real-life tips and thorough support from field experts
- a comprehensive guide of how you should study further

The goal

A rapid, focused deep NLP bootcamp:

the journey from a beginner in deep learning to a junior NLP researcher, capable of setting up the environment and building business-level text classification models

Syllabus

- 1회차: 딥러닝을 위한 PyTorch 실무환경 구축 - 오상준
  - [이론] Introduction to Deep Learning, Hello PyTorch
  - [실습] Image Classification with PyTorch

- 2회차: CNN and RNN in PyTorch - 오상준
  - [이론] Deep Neural Networks
  - [실습] Cryptocurrency price prediction with PyTorch

Syllabus

- 3회차: Neural Language Processing - 김기현
  - [이론] Natural Language Processing Introduction

- 4회차: Word Sense Disambiguation - 김기현
  - [이론] Word Sense Disambiguation
  - [실습] Word Sense Disambiguation Excercise

Syllabus

- 5회차: Word Embedding - 오상준
  - [이론] Word Vector Embedding
  - [실습] Word Embedding & Visualization

- 6회차: Text Classification with PyTorch - 김기현
  - [이론] Text Classification
  - [실습] Text Classification using RNN and CNN

Course mechanics

- Course materials
  - By e-mail and GitHub
- Questions
  - Any time: during, before, or after lectures
  - In person, by e-mail or Facebook(TBD)
- FastCampus regulations
  - Maximum 2 absences allowed
  - 3 e-mail surveys: 1st, 3rd, 6th week

Complementary materials

Intermediate

The materials below cover the theoretical parts of this course. They are recommended for further commitment.

- Deep Learning Book by Ian Goodfellow et al.
  - 딥러닝에 필요한 기초 선형대수 및 미적분부터 차근차근 설명합니다.
- CS231n: Convolutional Neural Networks for Visual Recognition at Standford (Materials)

Complementary materials

Intermediate

- CS224d: Deep Learning for Natural Language Processing at Standford (Materials)
  - Or, Oxford Deep NLP 2017
- Google 머신러닝 단기집중과정
- 초보를 위한 도커 안내서 - 도커란 무엇인가?

Complementary materials

Advanced

- Pattern Recognition and Machine Learning 우리말 정리

Other

- 라온피플 머신러닝 아카데미
- colah's blog
  - 잘 찾아보시면 번역본도 몇 편 있습니다.

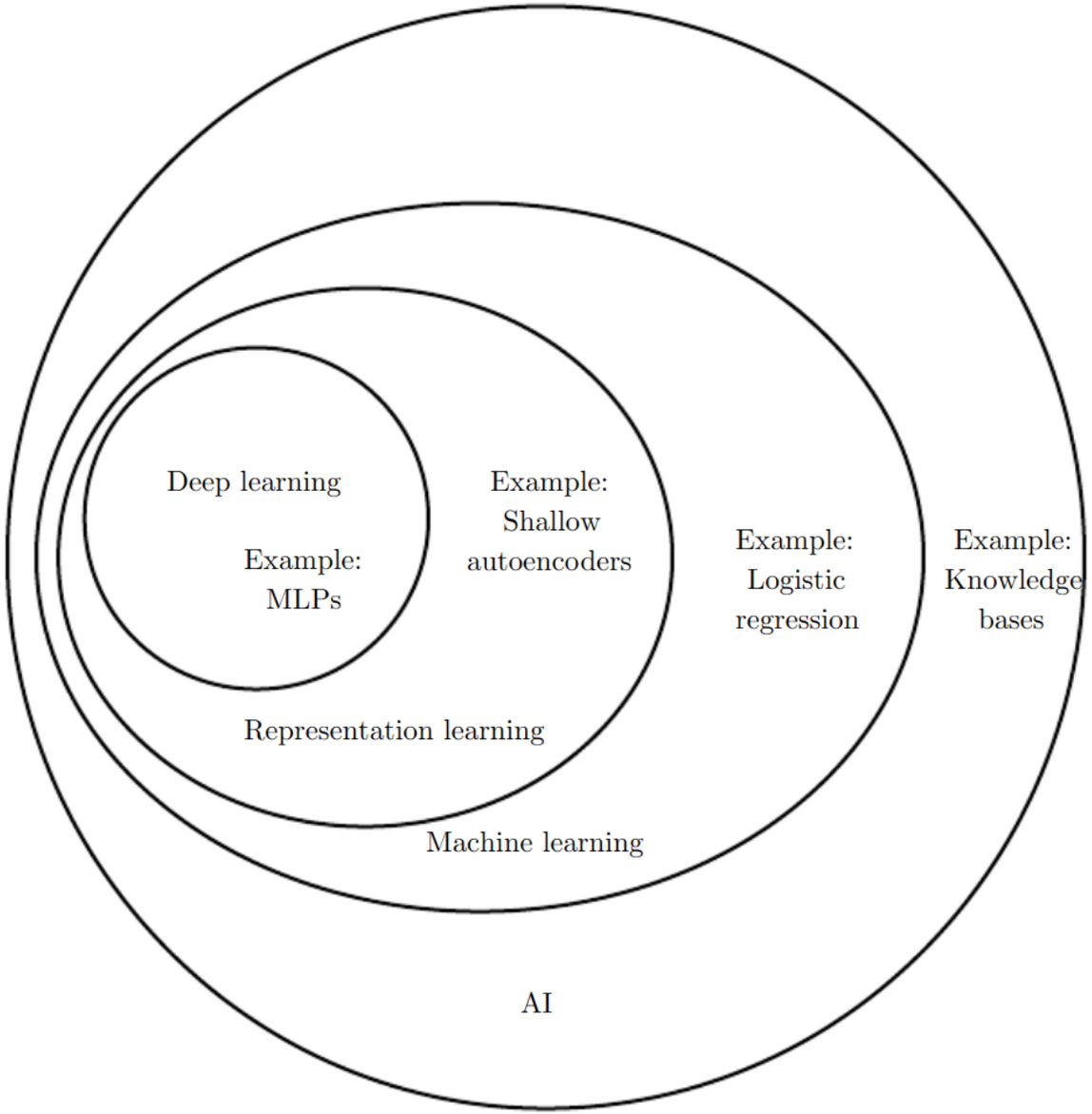
Complementary materials

Tips and trends

- The official PyTorch forums
- PyTorch KR (Facebook)
- TensorFLow KR (Facebook)
- PR12 딥러닝 논문읽기 모임 (YouTube)
- Two Minute Papers (YouTube)
- Machine Learning (Reddit)
- Google AI Blog

1. Introduction to Deep Learning

Fast campus

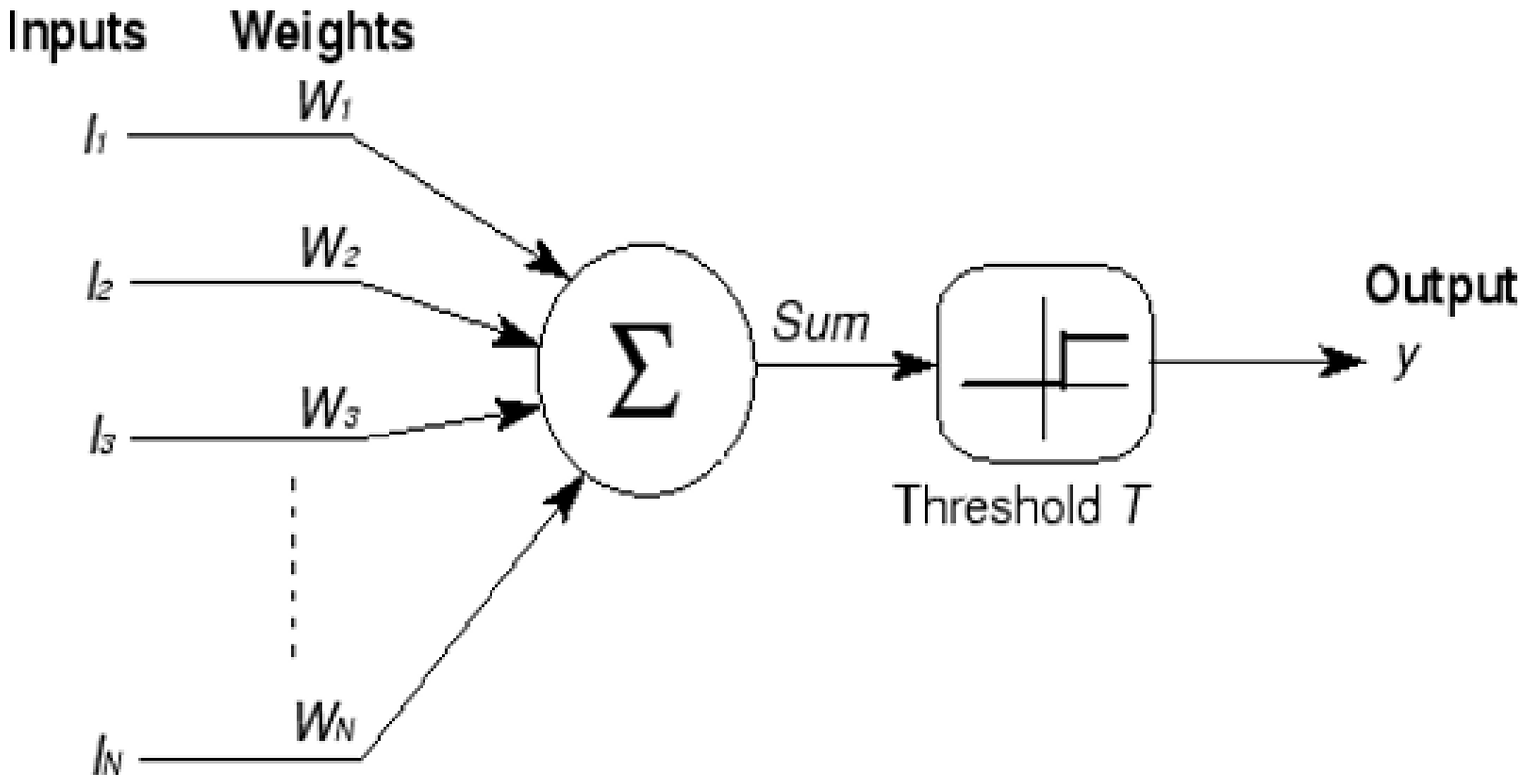

Deep learning
Example:
MLPs
Example:
Shallow
autoencoders
Representation learning
Example:
Logistic
regression
Machine learning
Example:
Knowledge
bases
AI

Genealogy

Timeline

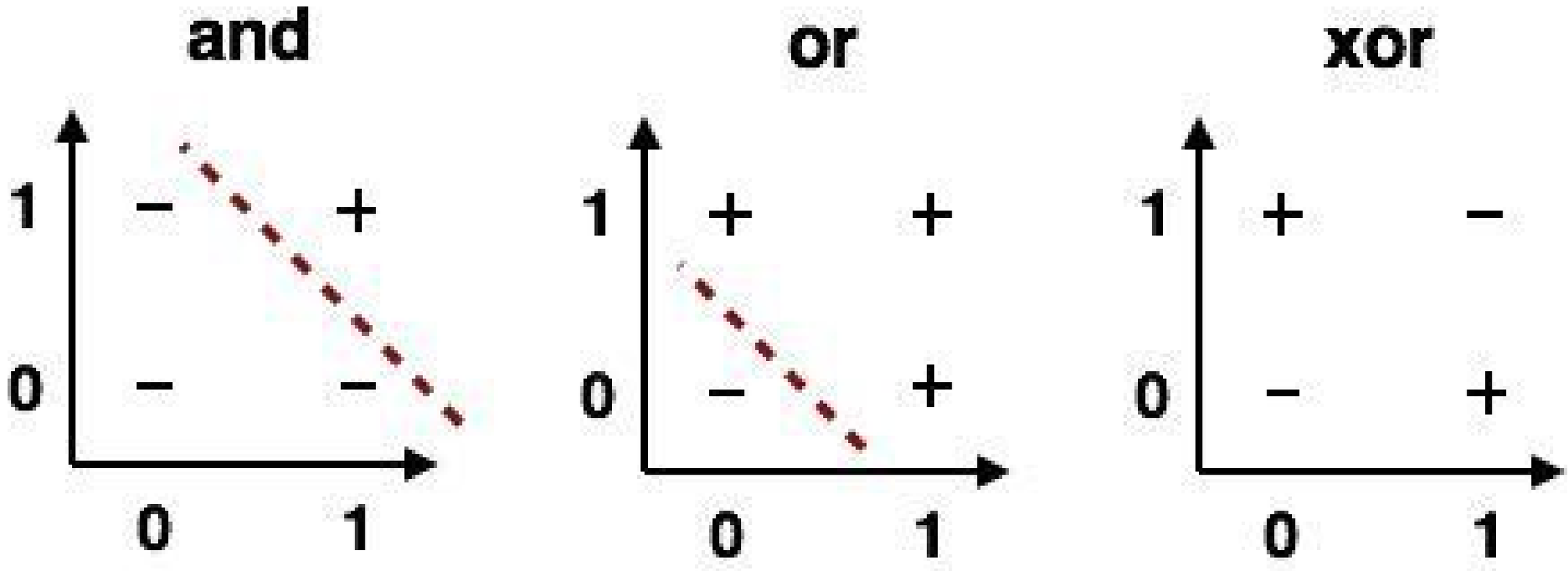
- **Cybernetics** 1940s-1960s
  - McCulloch-Pitts neuron
    - McCulloch and Pitts, 1942. A Logical Calculus of the Ideas Immanent in Nervous Activity.
  - Hebbian learning
    - Hebb, 1949. The Organization of Behaviour.
  - Perceptron
    - Rosenblatt, 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.

Timeline



A McCulloch-Pitts neuron.

Timeline



Timeline

Timeline

- **Connectionism** 1980s-1990s
  - Backpropagation
    - Rumelhart et al, 1986. Learning Representations by Back-propagating Errors.
  - Convolutional Neural Networks
    - Fukushima, 1980. Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position.

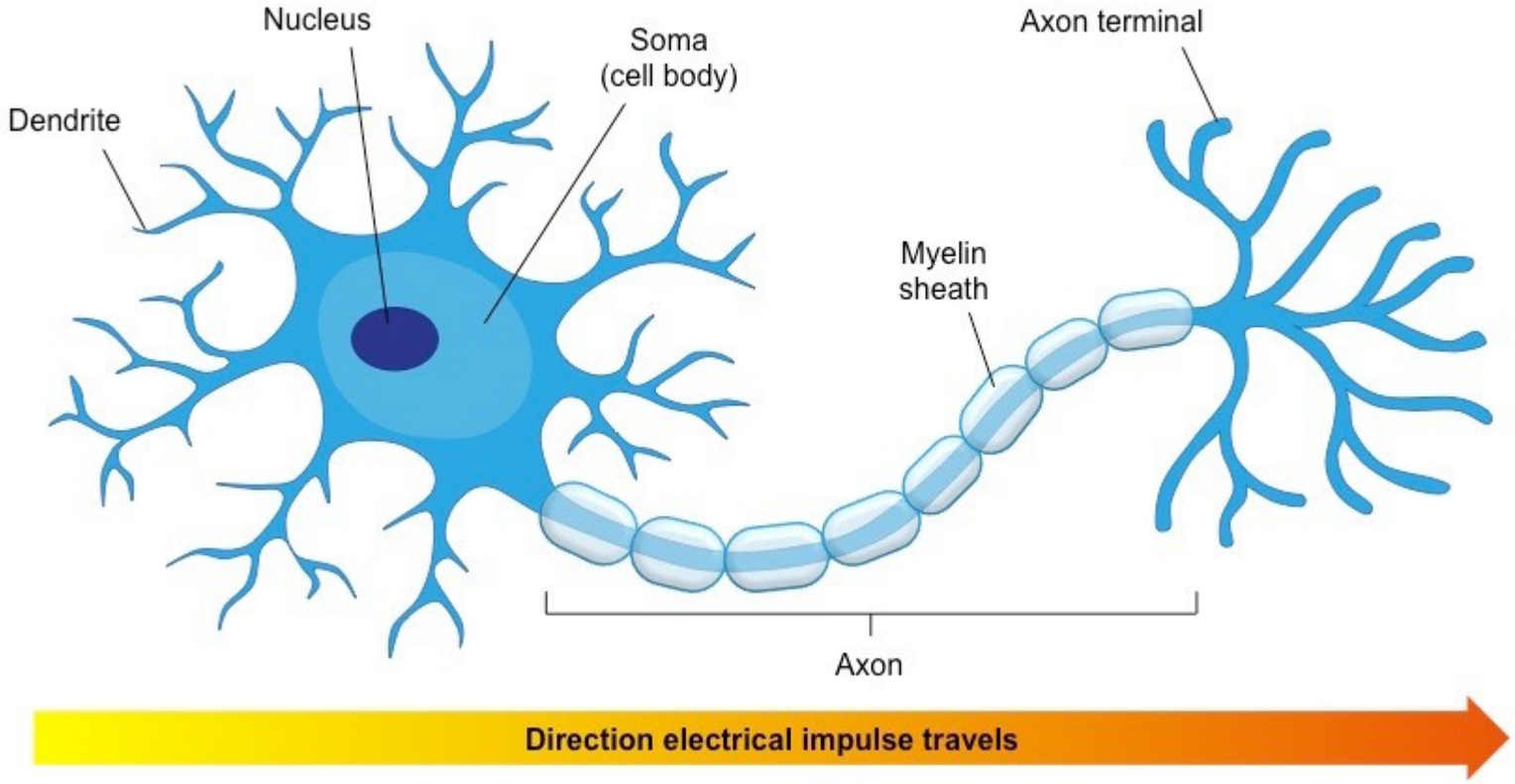

SK Telecom
이젠, 삐삐라고 하지않는다!
초고속 인공지능, 스피드012 - 삐삐라고 하기엔 너무 똑똑하다!
파워 VMS로 더 강해져!
지역번경안내는 자동으로!
세계표준시각을 배경으로!
모든 서비스를 패턴하나로!
검색지 수명 최고 4배까지!
미수신 메시지는 재전송!
초고속 인공지능
SPEED 012

Timeline

- **Deep Learning** 2006-
  - Deep Neural Networks
    - Hinton et al, 2006. A Fast Learning Algorithm for Deep Belief Nets.
  - Rectified Linear Units
    - Golorot et al, 2011. Deep Sparse Rectifier Neural Networks.
  - AlexNet
    - Krizhevsky et al, 2012. ImageNet Classification with Deep Convolutional Neural Networks.

Neural Networks

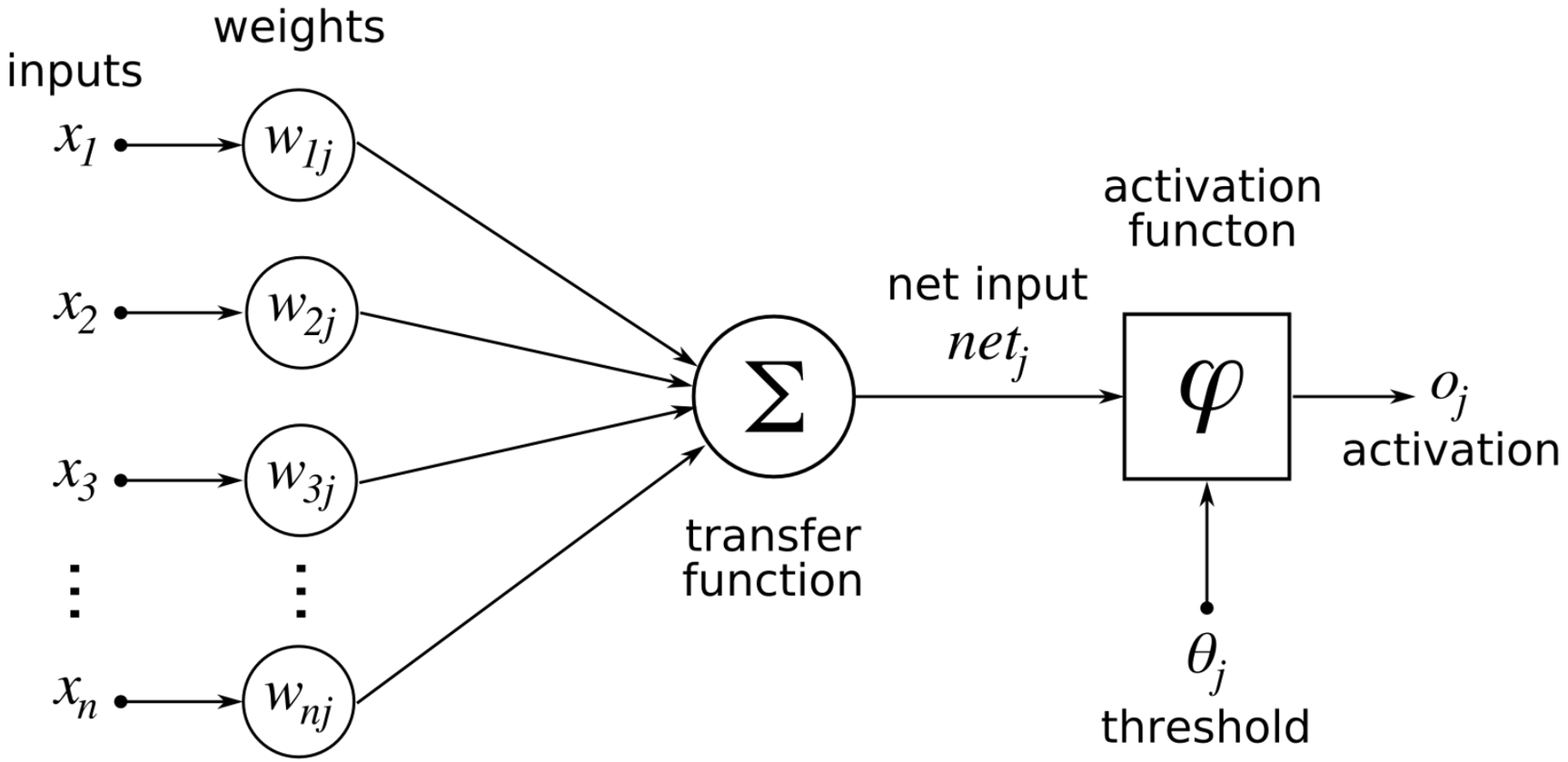
- Feed-forward Network



A biological neuron.

Neural Networks

- Feed-forward Network



A perceptron.

Neural Networks

- Feed-forward Network
  - In Python (with NumPy):

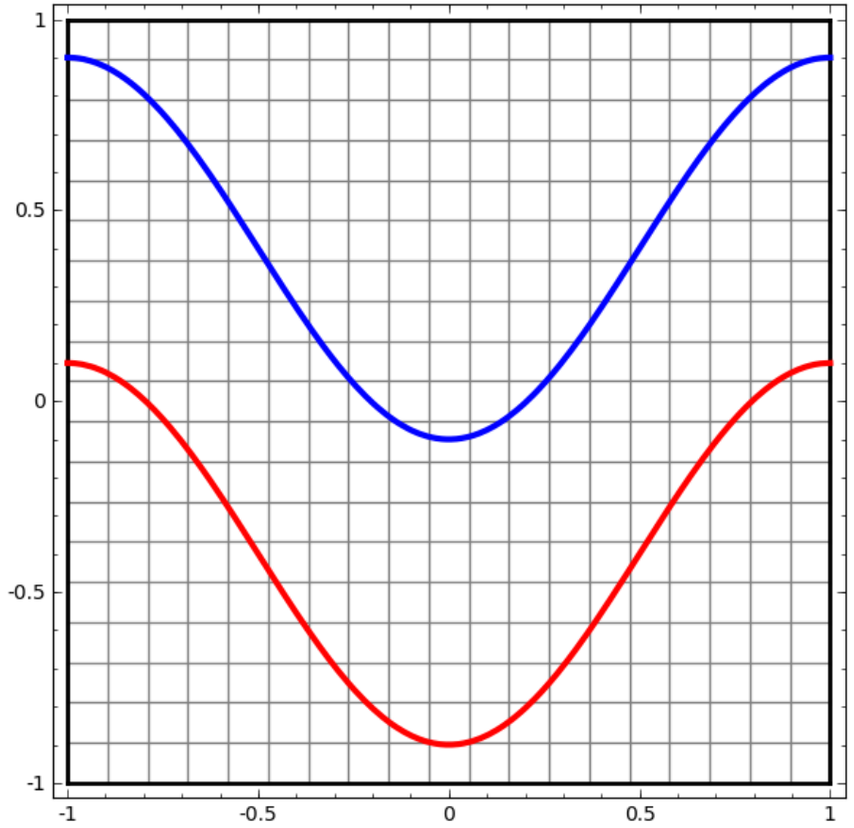
```
def perceptron(inputs, weights, biases):
    return sum(inputs * weights + biases)
```

  - In PyTorch:

```
outputs = module(inputs)
```

Neural Networks

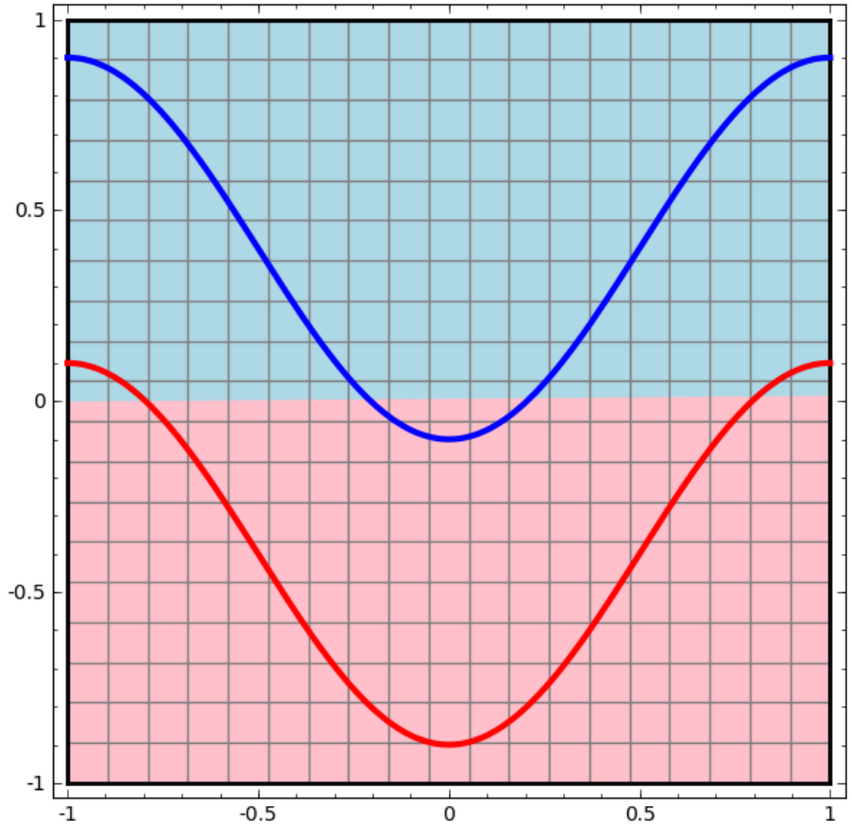
Problem: draw a single straight line to separate colors.



http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Neural Networks

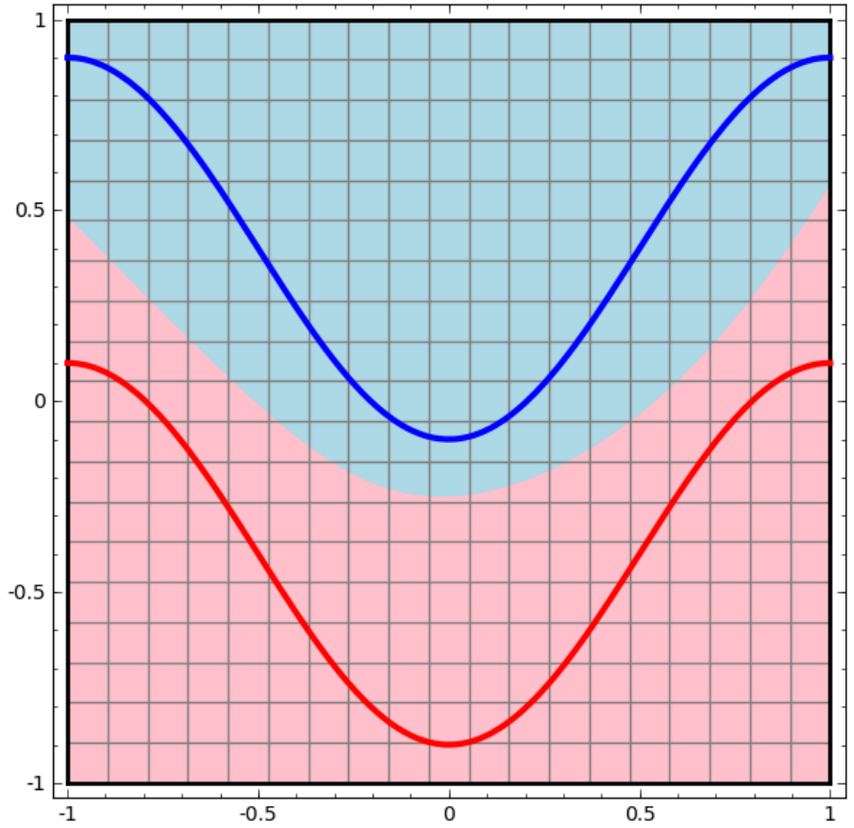
Problem: draw a single straight line to separate colors.



http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Neural Networks

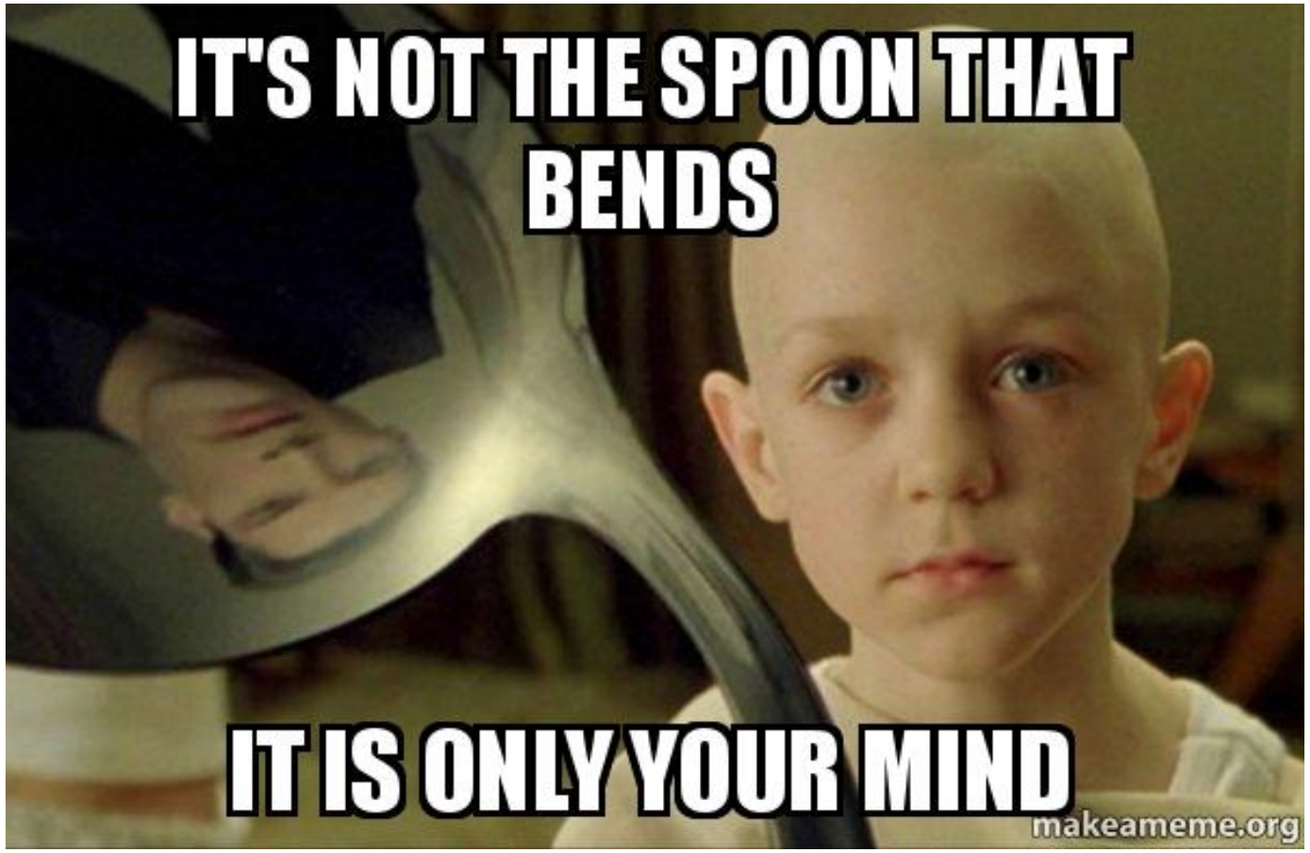
Problem: draw a single straight line to separate colors.



http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

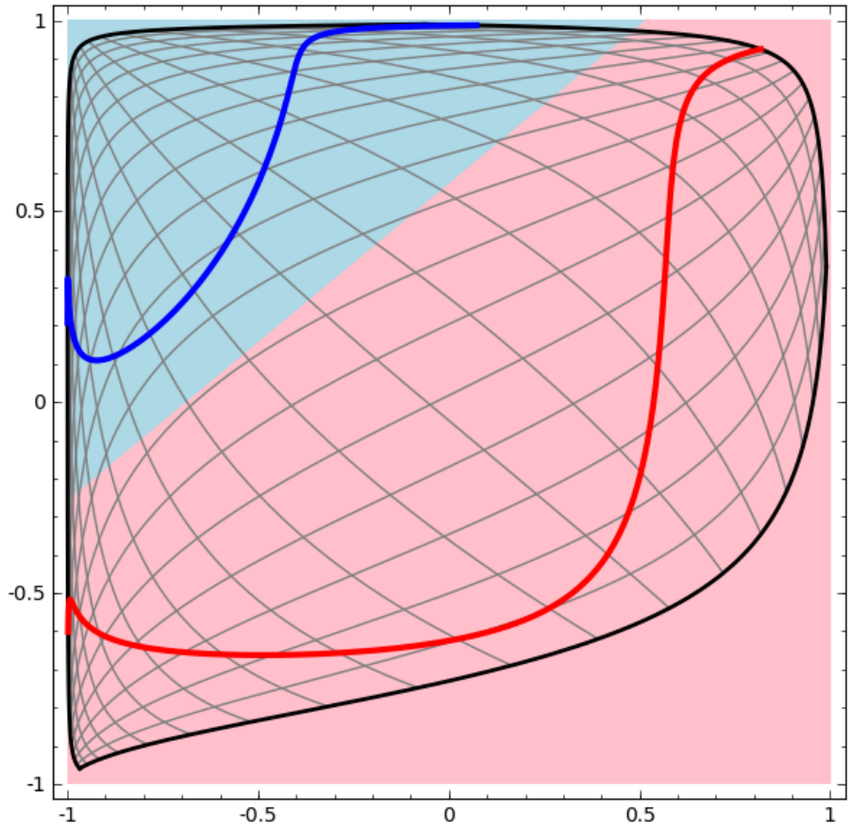
Neural Networks

Problem: draw a single straight line to separate colors.



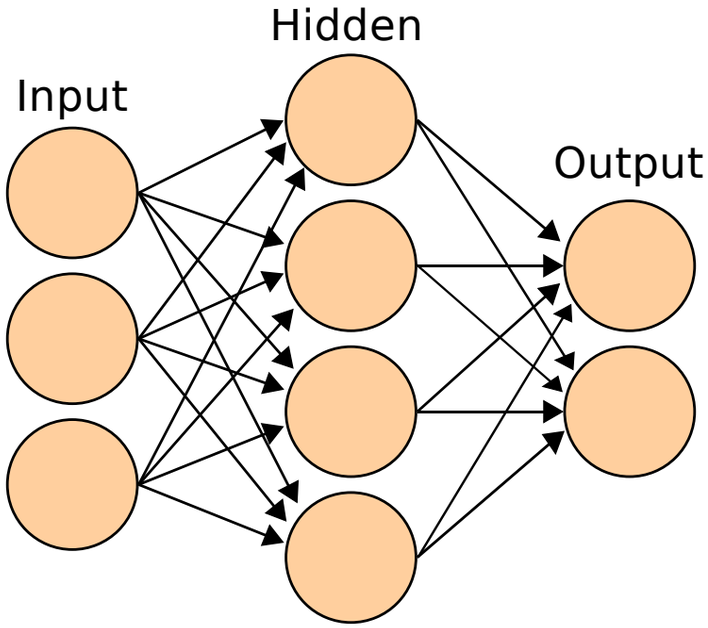
Neural Networks

Problem: draw a single straight line to separate colors.



http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Neural Networks

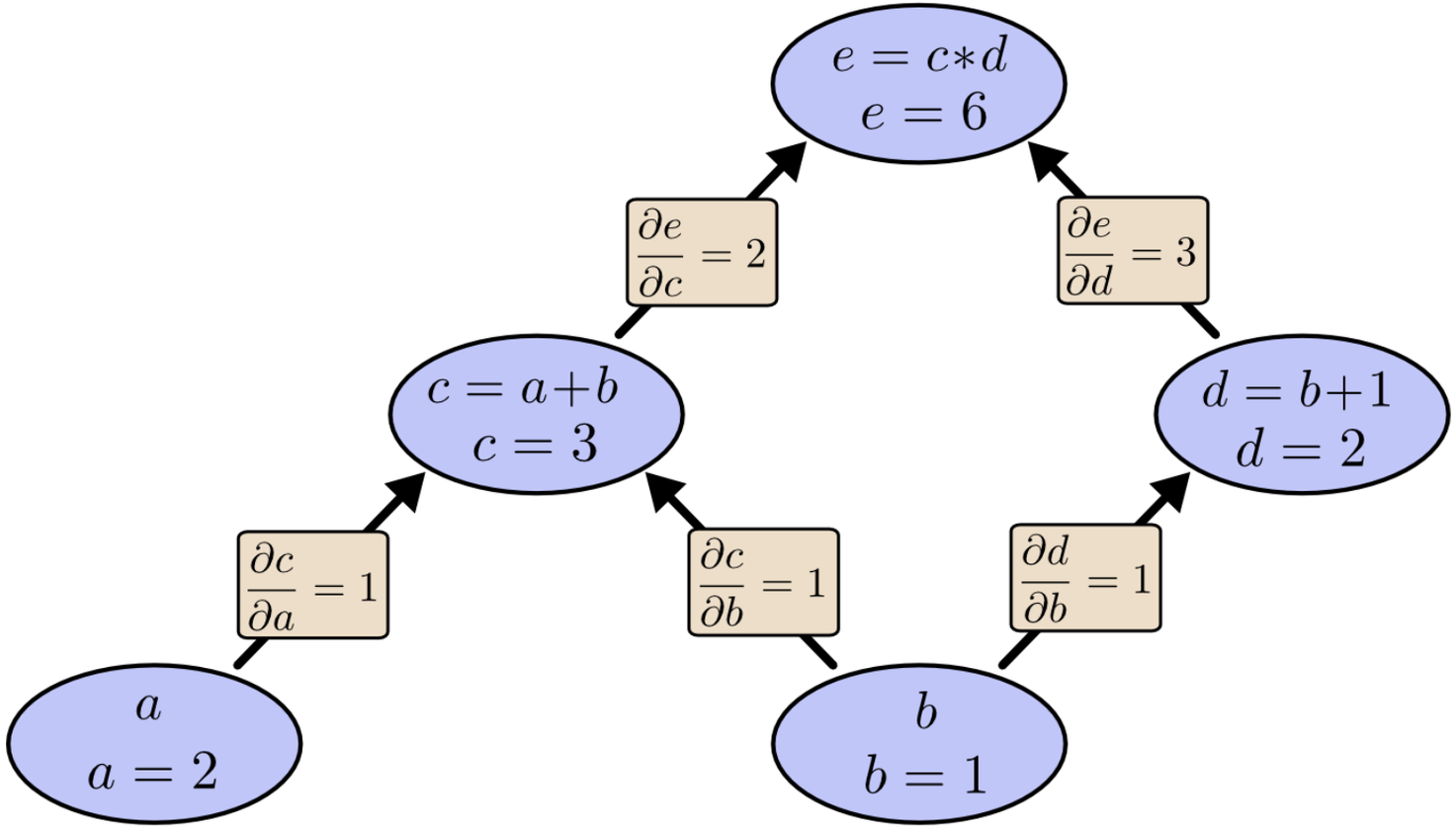


The hidden layer learns a representation,
so that the data is linearly separable.

http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Neural Networks

- Backpropagation



http://colah.github.io/posts/2015-08-Backprop/

Neural Networks

- Backpropagation
  - In Python (with NumPy):

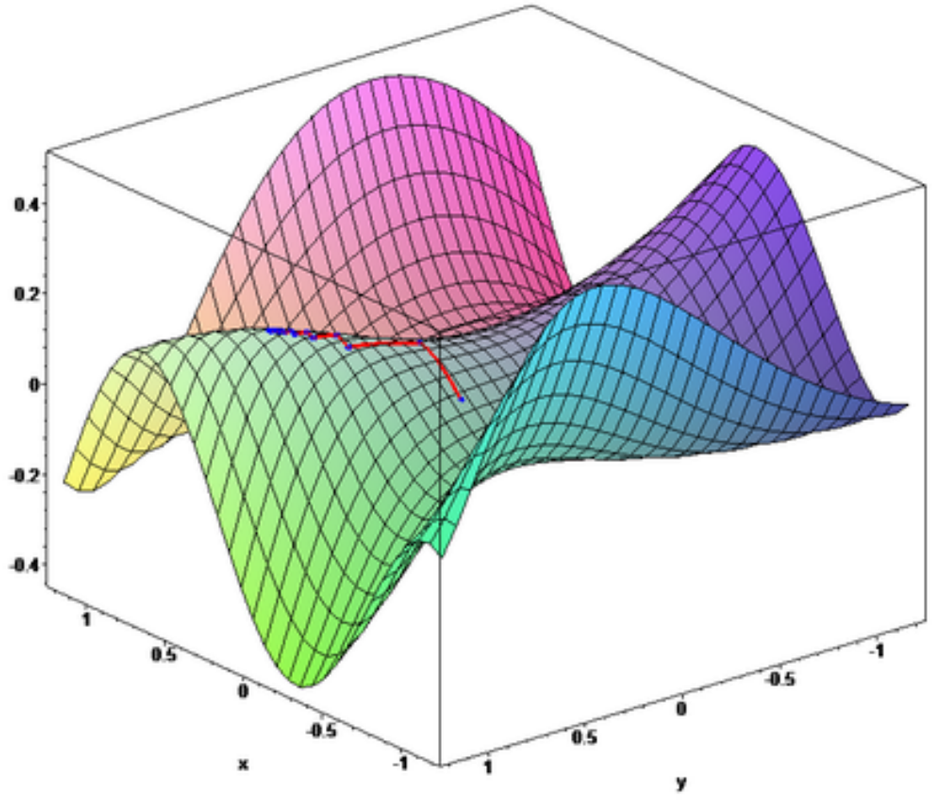
```
def backpropagate(weights, derivative, learning_rate):
    return weights - learning_rate * (derivative - weights)
```

  - In PyTorch:

```
loss = loss_function(outputs, targets)
loss.backward()
```

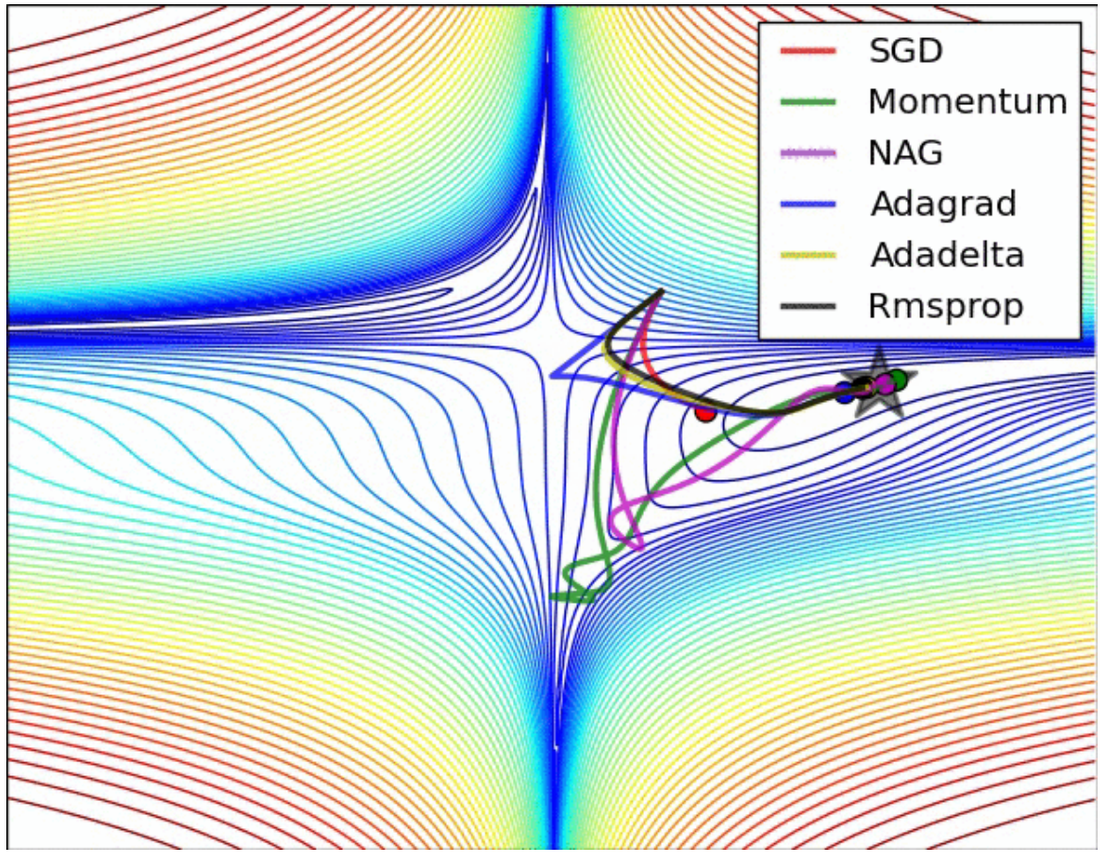
Neural Networks

- Gradient Descent
  - Stochastic Gradient Descent, Momentum, Adagrad, Adam, ...



Neural Networks

- Gradient Descent
  - Stochastic Gradient Descent, Momentum, Adagrad, Adam, ...

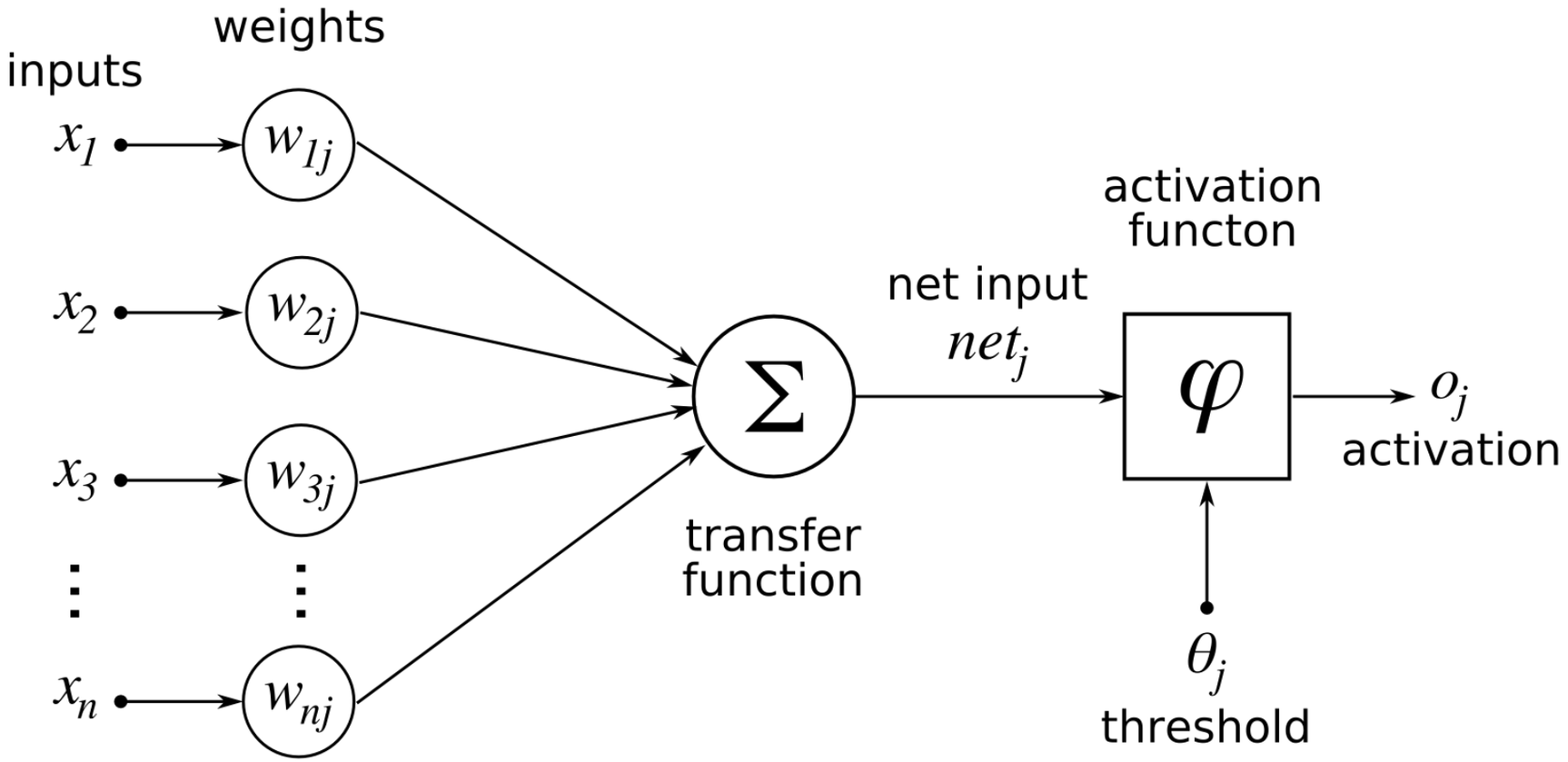


Neural Networks

- Gradient Descent
  - Stochastic Gradient Descent, Momentum, Adagrad, Adam, ...
    - **SGD** is steady and stable. `torch.optim.SGD`
    - **Adam** is fast, but sometimes wacky. `torch.optim.Adam`
  - In PyTorch: `torch.optim`

```
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

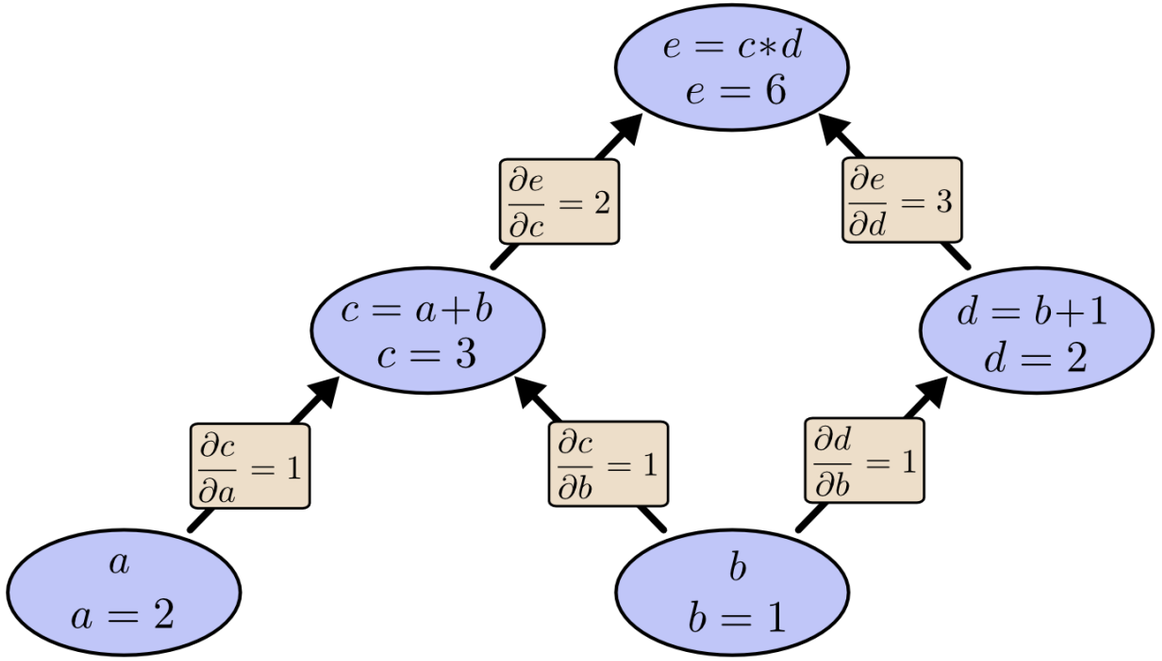
Activation functions and non-linearity



A perceptron.

Activation functions and non-linearity

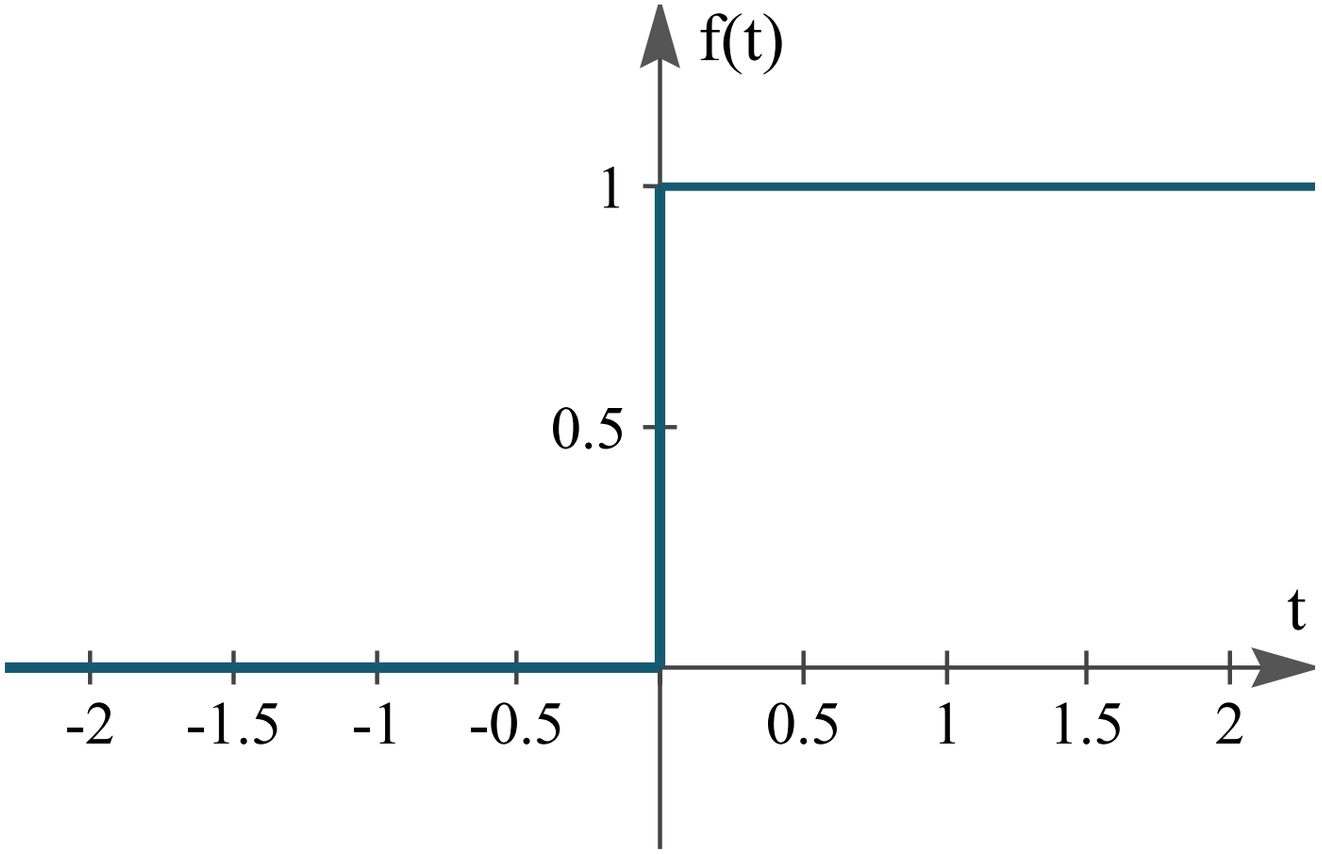
- Where activation functions are used
  - In-between layers: **ReLU** and its variants
  - After the final layer: **Sigmoid** or **Softmax**



http://colah.github.io/posts/2015-08-Backprop/

Activation functions and non-linearity

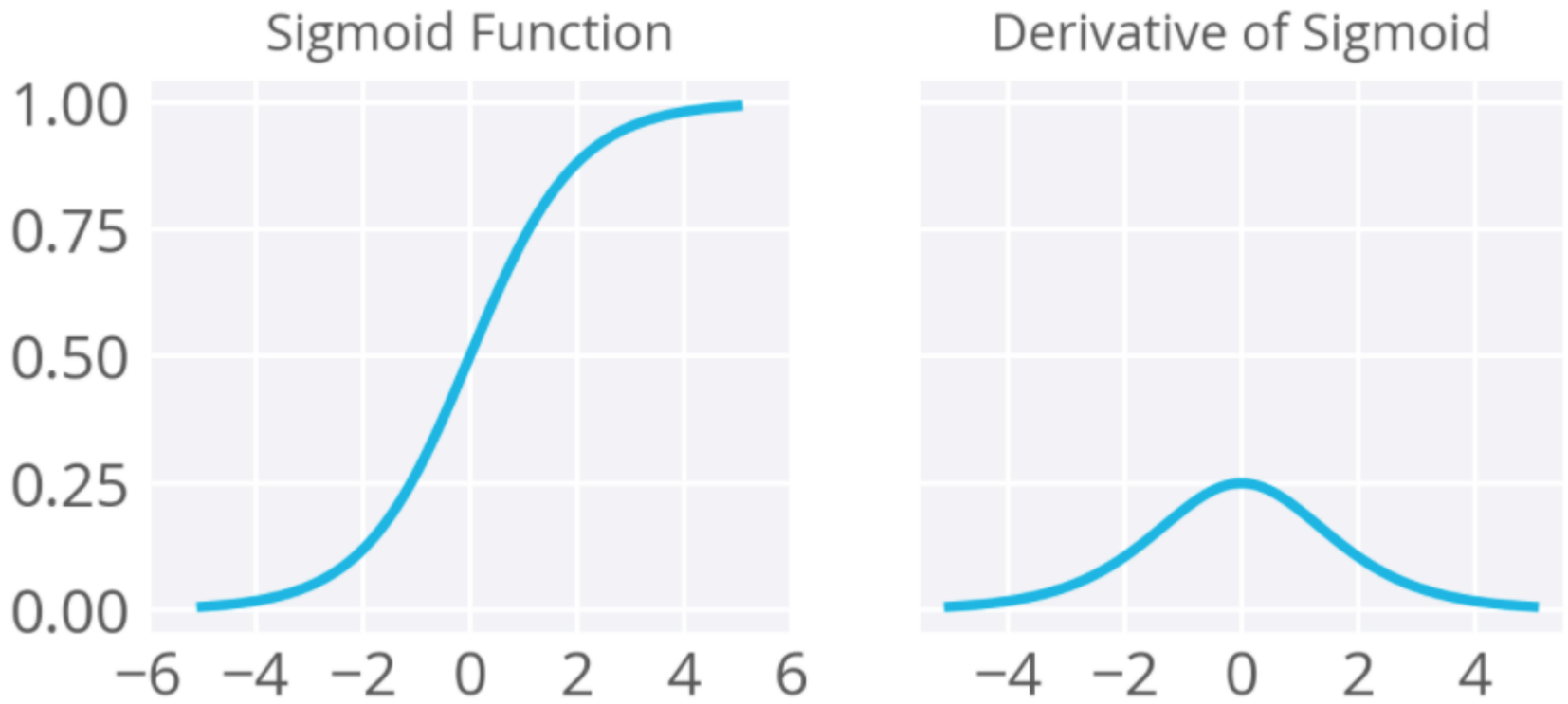
Step function




I'm afraid you couldn't be more wrong.

Activation functions and non-linearity

Sigmoid function



https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions

Activation functions and non-linearity

Sigmoid function

$$\frac{1}{1 + e^{-x}}$$

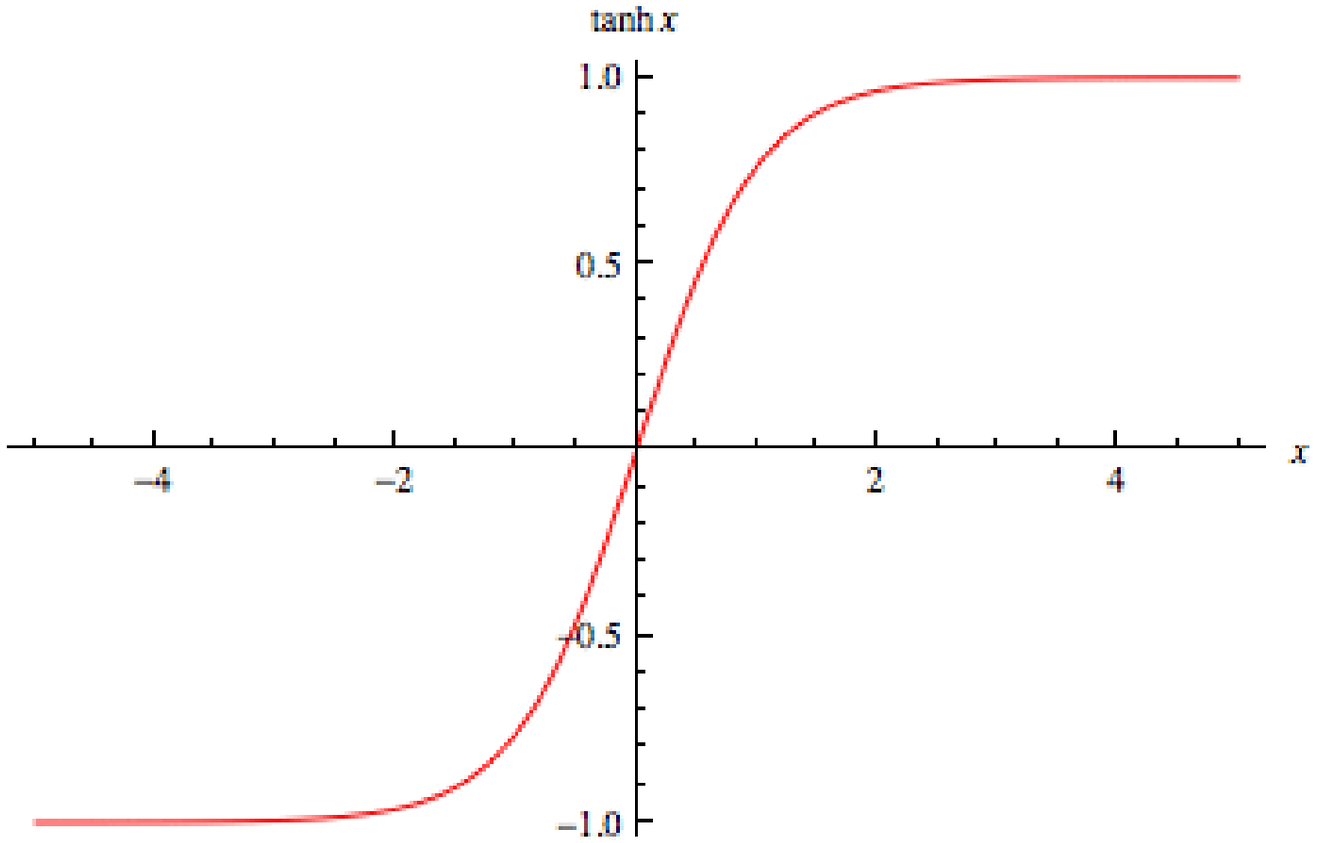
- In Python (with NumPy):

```
def sigmoid(inputs):
    return 1.0 / (1.0 + exp(-inputs))
```

- In PyTorch: `torch.nn.Sigmoid`
  - Provides automatic gradient calculation, guards against divide-by-zero errors, scales to batches, supports GPU, etc.

Activation functions and non-linearity

Hyperbolic Tangent (tanh)



http://mathworld.wolfram.com/HyperbolicTangent.html

Activation functions and non-linearity

Hyperbolic Tangent (tanh)

$$\frac{e^x - e^{-x}}{e^x + e^{-x}}$$

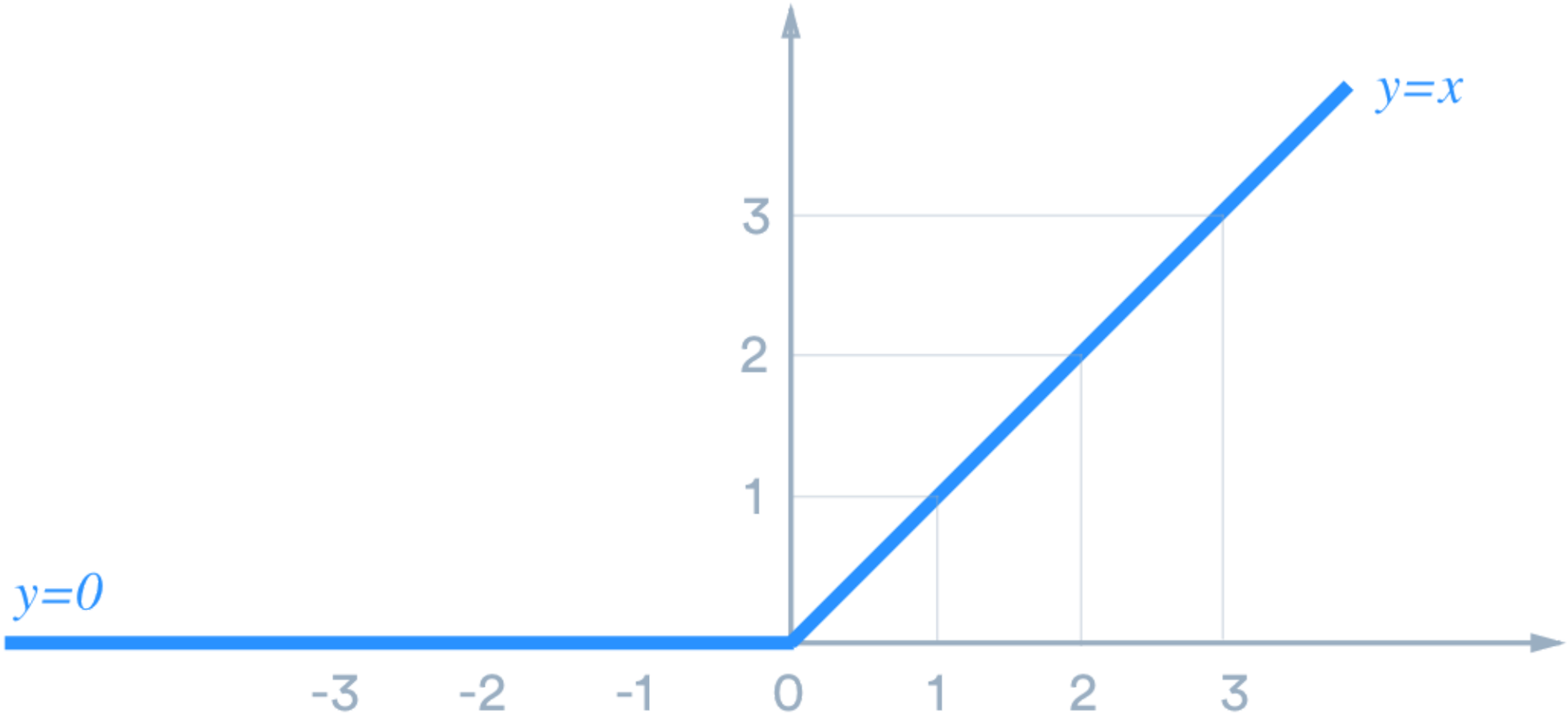
- In Python (with NumPy):

```
def tanh(inputs):
    return (exp(inputs) - exp(-inputs)) / (exp(inputs) + exp(-
inputs))
```

- In PyTorch: `torch.nn.Tanh`

Activation functions and non-linearity

Rectified Linear Units (ReLU)



Activation functions and non-linearity

Rectified Linear Units (ReLU)

$$max(0, x)$$

- In Python (with NumPy):

```
def relu(inputs):
    return max(0, inputs)
```

- In PyTorch: `torch.nn.ReLU`

Activation functions and non-linearity

Softmax function



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture03.pdf

Activation functions and non-linearity

Softmax function

https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions

Activation functions and non-linearity

Softmax function

$$\frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

- In Python (with NumPy):

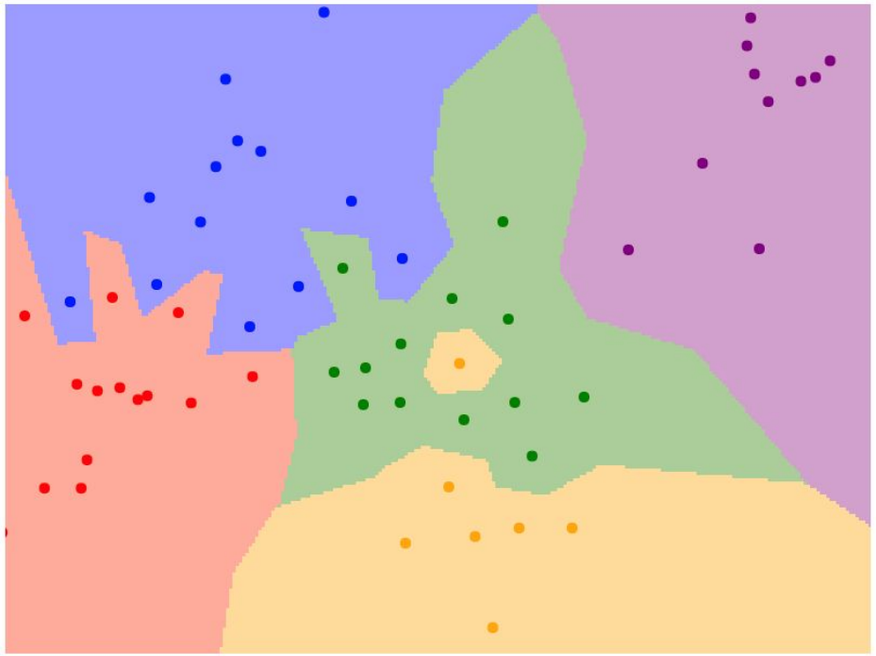
```
def softmax(inputs):
  return exp(inputs) / sum(exp(inputs))
```

- In PyTorch: `torch.nn.Softmax`

Loss functions

L1 loss and L2 loss

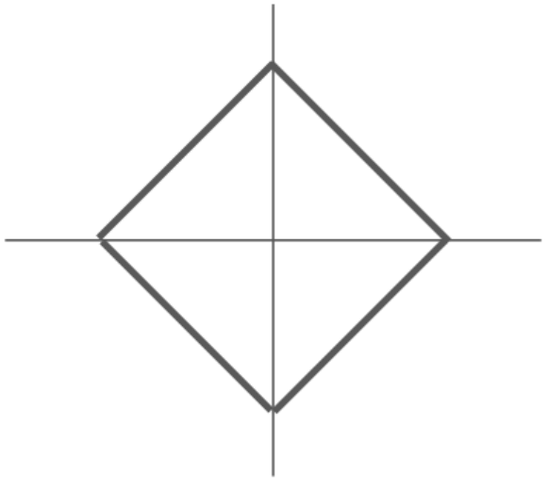
- k-Nearest Neighbors



Loss functions

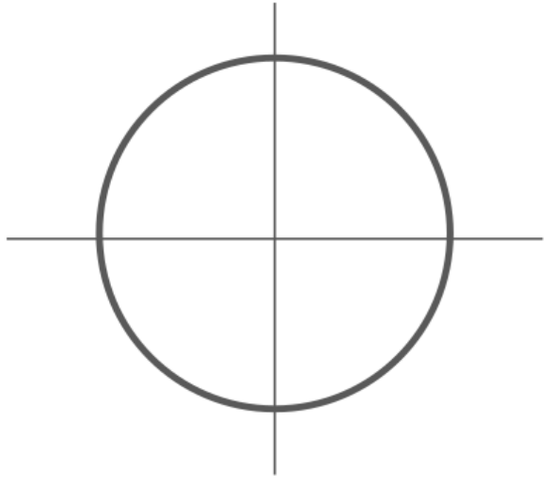
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture02.pdf

Loss functions

L1 loss

$$\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

- In Python (with NumPy):

```
def l1_loss(targets, outputs):
    return sum(abs(targets - outputs))
```

- In PyTorch: `torch.nn.L1Loss`

Loss functions

L2 loss

````
$$\sum^n_{i=1}(y_i - \hat{y}_i)^2$$
* In Python (with NumPy):
  ```python
  def l2_loss(targets, outputs):
      return sum(sqrt((targets - outputs)**2))
  ```
* In PyTorch: [`torch.nn.MSELoss`]
(https://pytorch.org/docs/stable/nn.html#torch.nn.MSELoss)<br>
````

Loss functions

Mean Square Error

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- In Python (with NumPy):

```
def mean_square_error(targets, outputs):
    return mean(sqrt((targets - outputs)**2))
```

- In PyTorch: `torch.nn.MSELoss`

Loss functions

Cross Entropy

Entropy(in information theory)

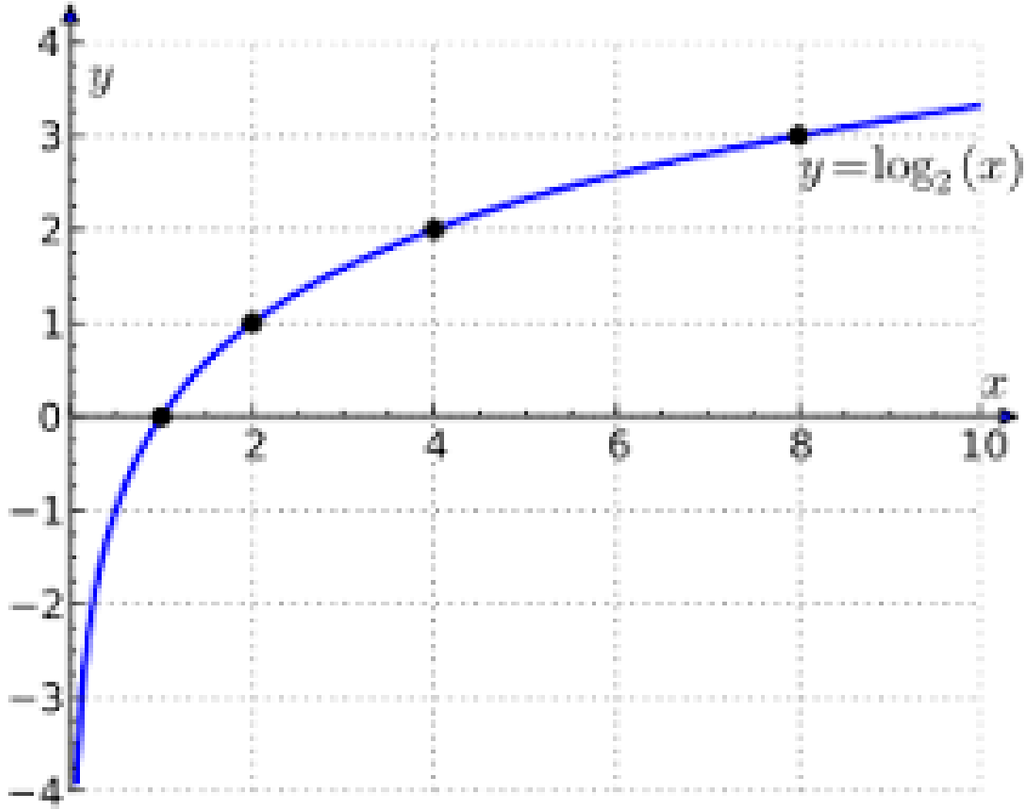
= amount of information in an event

= amount of surprise

Loss functions

Cross Entropy

- Entropy



Loss functions

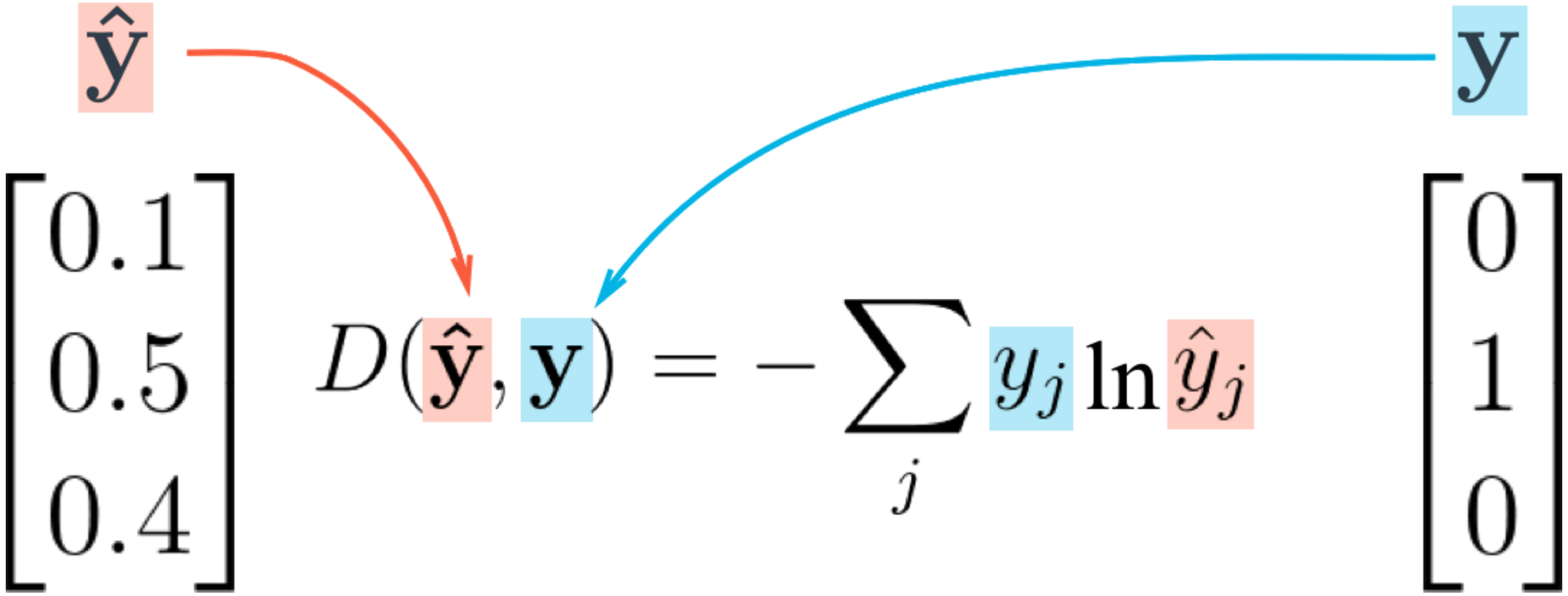
Cross Entropy

- Entropy

$$h[x] = -log(p(x))$$

Loss functions

Cross Entropy



https://towardsdatascience.com/deep-learning-concepts-part-1-ea0b14b234c8

Loss functions

Cross Entropy

$$-\sum_{i=1}^{n} y_i ln(\hat{y}_i)$$

- In Python:

```
def cross_entropy_loss(targets, outputs):
    return -sum(targets * log(outputs))
```

- In PyTorch: `torch.nn.CrossEntropyLoss`

Loss functions

Cross Entropy

$$-\frac{1}{n}\sum_{i=1}^{n}[y_i ln(\hat{y}_i) + (1-y_i) ln(1-\hat{y}_i)]$$

- In Python:

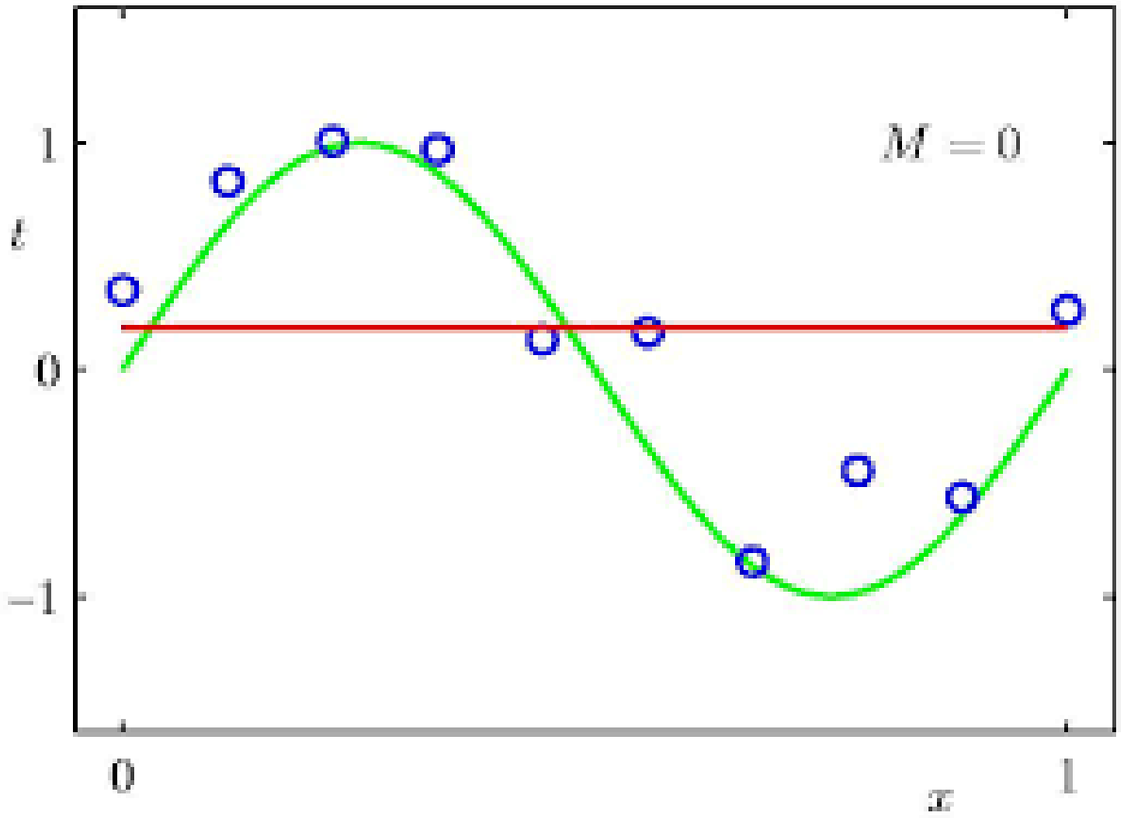
```
def binary_cross_entropy_loss(targets, outputs):
    return -mean(targets * log(outputs) + (1 - targets) * log(1 - outputs))
```

- In PyTorch: `torch.nn.BCELoss`

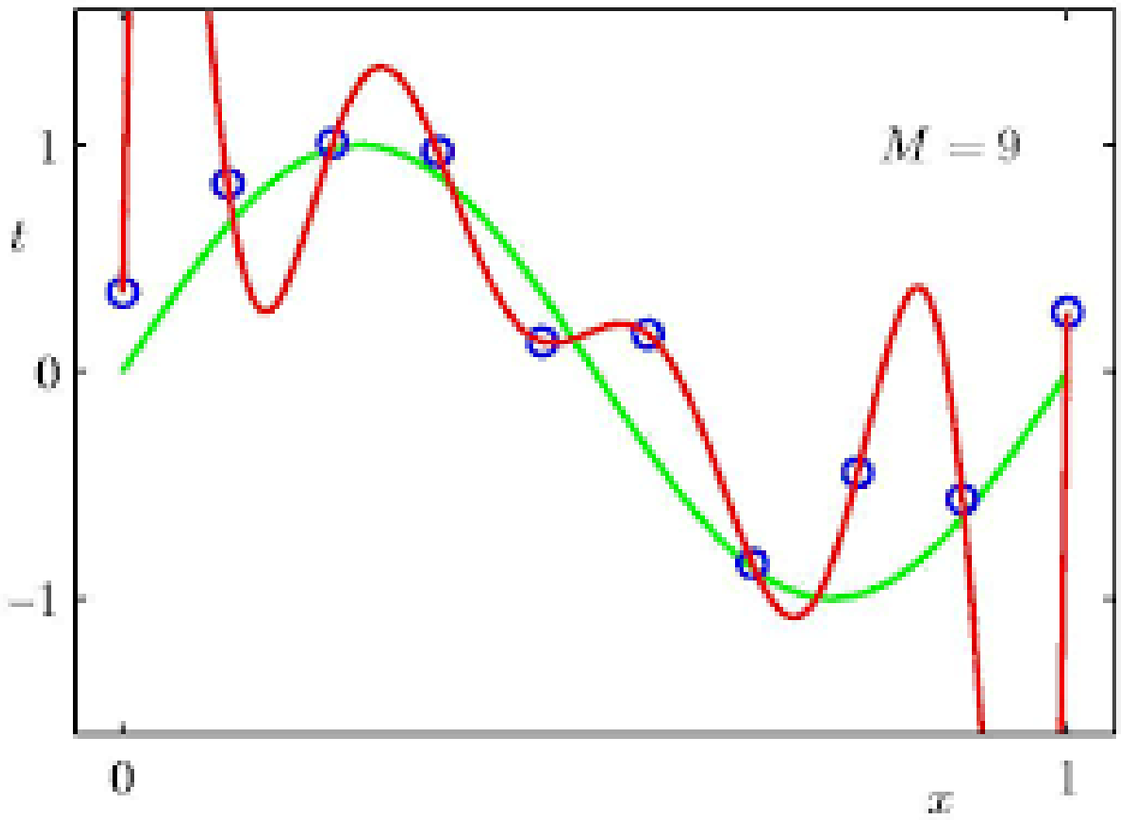
Loss functions

- Baselines
  - Use softmax and cross entropy loss in multi-class classifications
  - Use sigmoid and binary cross entropy loss in binary classifications

Regularization methods



Regularization methods



Regularization methods

Weight decay

$$W \leftarrow W - \lambda(\frac{\partial L}{\partial W} + \gamma \|W\|)$$

- In Python (with NumPy):

```
def backpropagate(weights, derivative, learning_rate,
weight_decay):
  weight_penalty = weight_decay * sum(sqrt(weights ** 2))
  return weights - learning_rate * (derivative @ weights +
weight_penalty)
```

Regularization methods

Weight decay

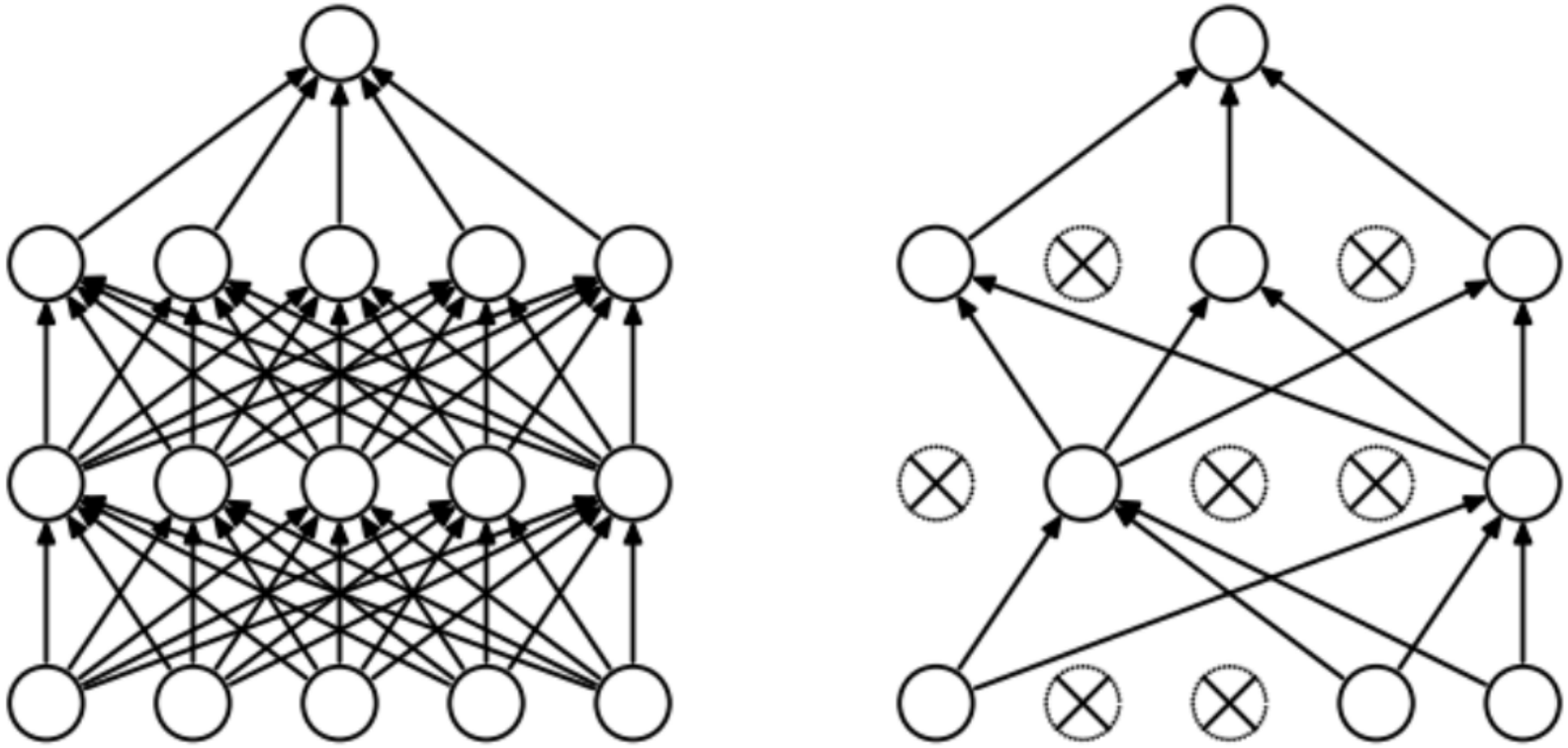
$$W \leftarrow W - \lambda(\frac{\partial L}{\partial W} + \gamma \|W\|)$$

- In PyTorch:

```
optimizer = torch.optim.SGD(learning_rate=0.1, weight_decay=0)
```

Regularization methods

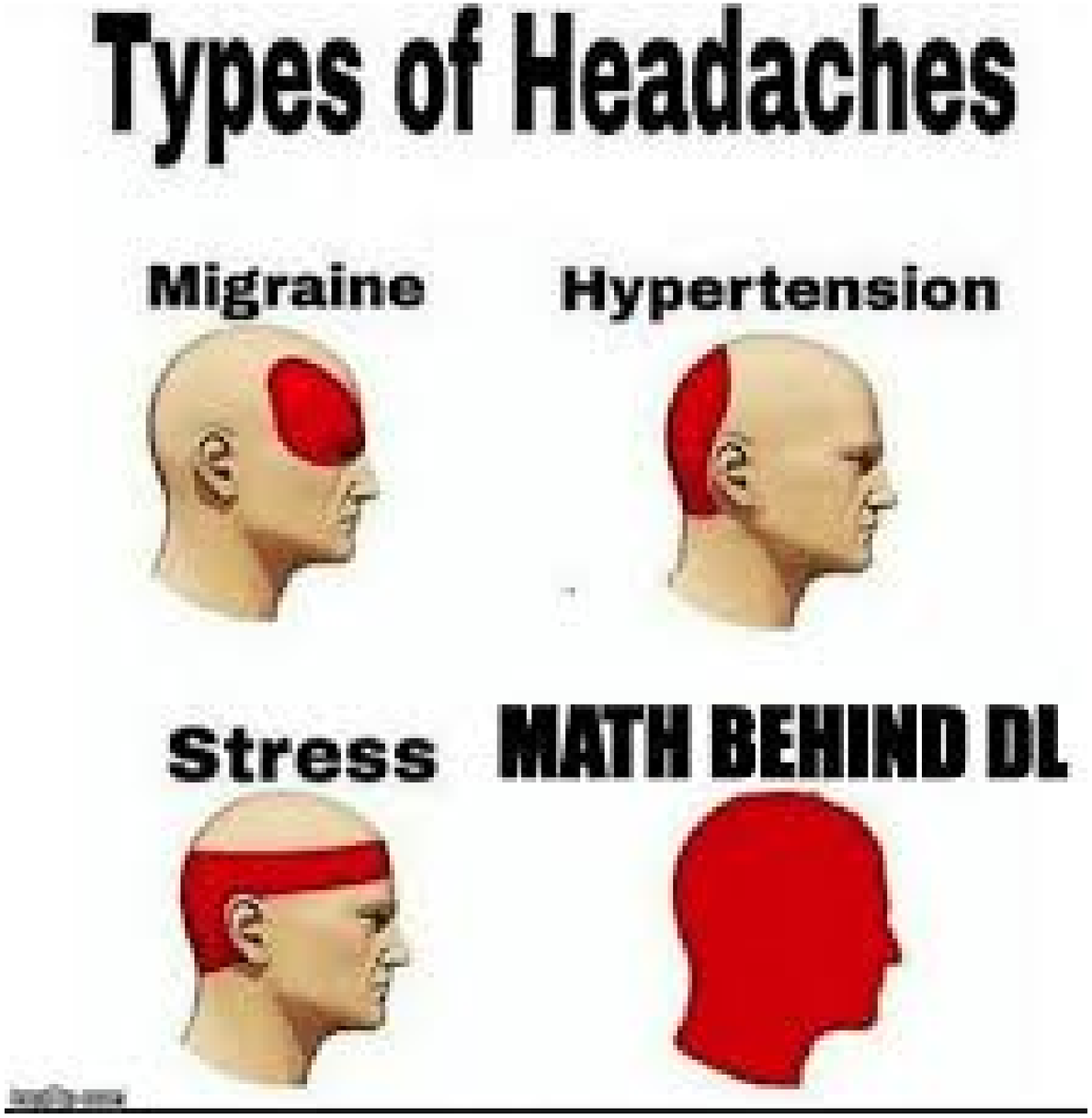
Dropout



Hinton et al. Improving neural networks by preventing co-adaptation of feature detectors. 2012

FAQ

- Typical DL project workflow
  - Obtain and cleanse data
  - Overfit using latest baseline models
  - Tune and optimize
    - Hard example mining, model pruning, custom loss functions, etc.
  - Deploy
    - Open Neural Network Exchange Format


Types of Headaches
Migraine
Hypertension
Stress
MATH BEHIND DL

Hello PyTorch

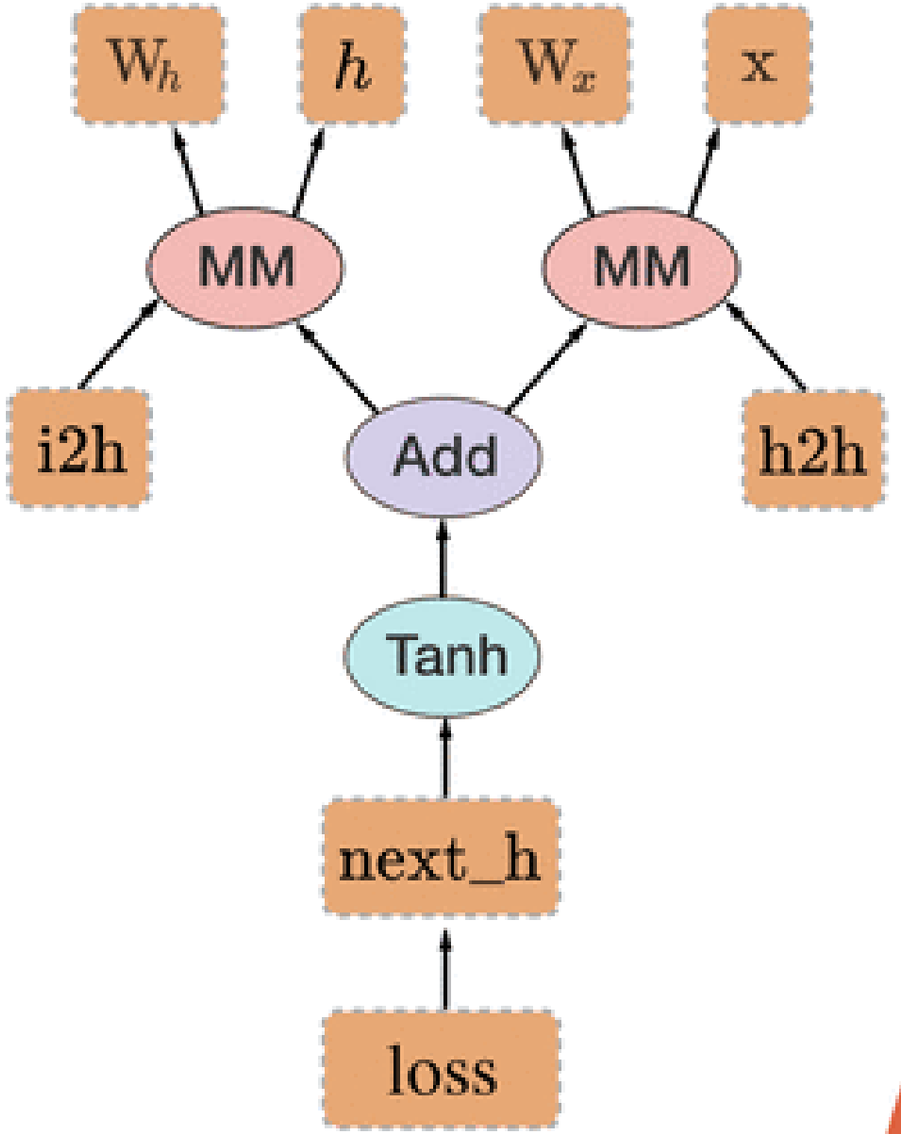
02

Fast campus

- Deep Learning Framework
  - Tensorflow, Keras, Torch, Chainer, MXNet
- Python-native, NumPy-friendly
- Dynamic graphs
- https://pytorch.org/
- https://pytorch.org/docs/stable/index.html

Back-propagation
uses the dynamically created graph

```
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
W_h = torch.randn(20, 20)
W_x = torch.randn(20, 10)

i2h = torch.mm(W_x, x.t())
h2h = torch.mm(W_h, prev_h.t())
next_h = i2h + h2h
next_h = next_h.tanh()

loss = next_h.sum()
loss.backward()  # compute gradients!
```



PyTorch core modules

`torch.Tensor`

- ≈ `numpy.array`
- Creation

```
>>> torch.Tensor([[1, 1, 1], [1, 1, 1]])

 1  1  1
 1  1  1
[torch.FloatTensor of size 2x3]
```

PyTorch core modules

`torch.Tensor`

- Broadcasting

```
>>> torch.Tensor([[1, 1, 1], [1, 1, 1]]) * 2

 2  2  2
 2  2  2
[torch.FloatTensor of size 2x3]
```

PyTorch core modules

`torch.Tensor`

- In-place operations (marked by an underscore `_`)

```
>>> a = torch.Tensor([[1, 1, 1], [1, 1, 1]])
>>> a.add_(2)
>>> a

 3  3  3
 3  3  3
[torch.FloatTensor of size 2x3]
```

PyTorch core modules

`torch.nn.Module`

- The building block of PyTorch deep neural network models
- The Python magic method `__call__()` executes `forward()` method
- If `backward()` method is not defined, the backward is automatically created by `forward()`
- Allows nesting and chaining

PyTorch core modules

`torchvision.models.resnet`

- The PyTorch reference implementation of the ResNet
- A great example of building PyTorch models with Python OOP

Our stack

Python 3.6+

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
```

Our stack

Conda

- Package manager + virtual environments
- Accessed via the terminal (Ubuntu and macOS) or Anaconda Prompt (Windows)
- Distributions: Anaconda and Miniconda
- https://conda.io/

Our stack

Conda

- Creating a new virtual environment

```
$ conda create -n ENV
```

- Activating a virtual environment

```
$ source activate ENV
```

Remove `source` on Windows.

Our stack

Conda

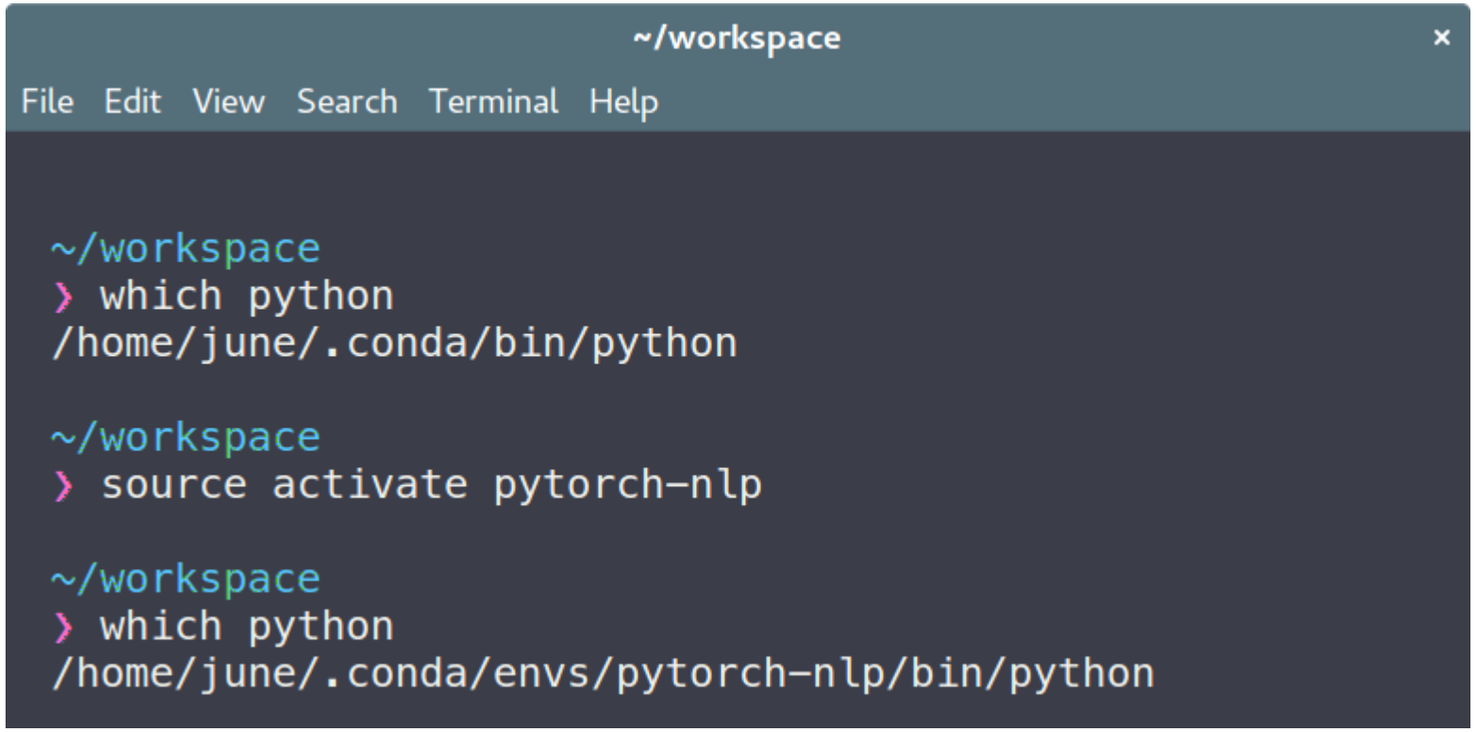
- Deactivating a virual environment

```
$ deactivate
```

Our stack

Conda

- Conda works by altering the `PATH` environment variable



Our stack

Jupyter Notebook

- Document and visualize live code
- http://jupyter.org/

jupyter Untitled Last Checkpoint: Last Saturday at 10:22 AM (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run | Markdown

In [1]:
```
import this
```

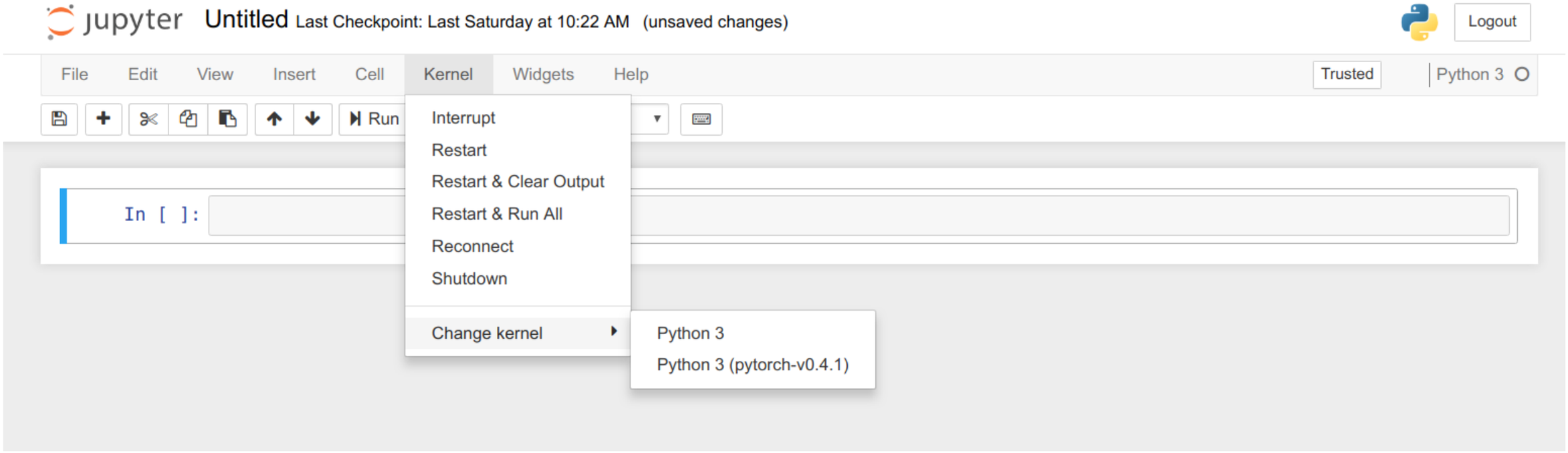
```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

```
Markdown text
```

Our stack

Jupyter Notebook

- Kernels



Our stack

Jupyter Notebook

- Kernels
  - ≈ Conda environments
  - Adding a new kernel

```
$ source activate pytorch-nlp # Remove "source" in Windows
$ pip install ipykernel
$ python -m ipykernel -n pytorch-nlp --display-name "Python 3.6
(pytorch-nlp)"
```

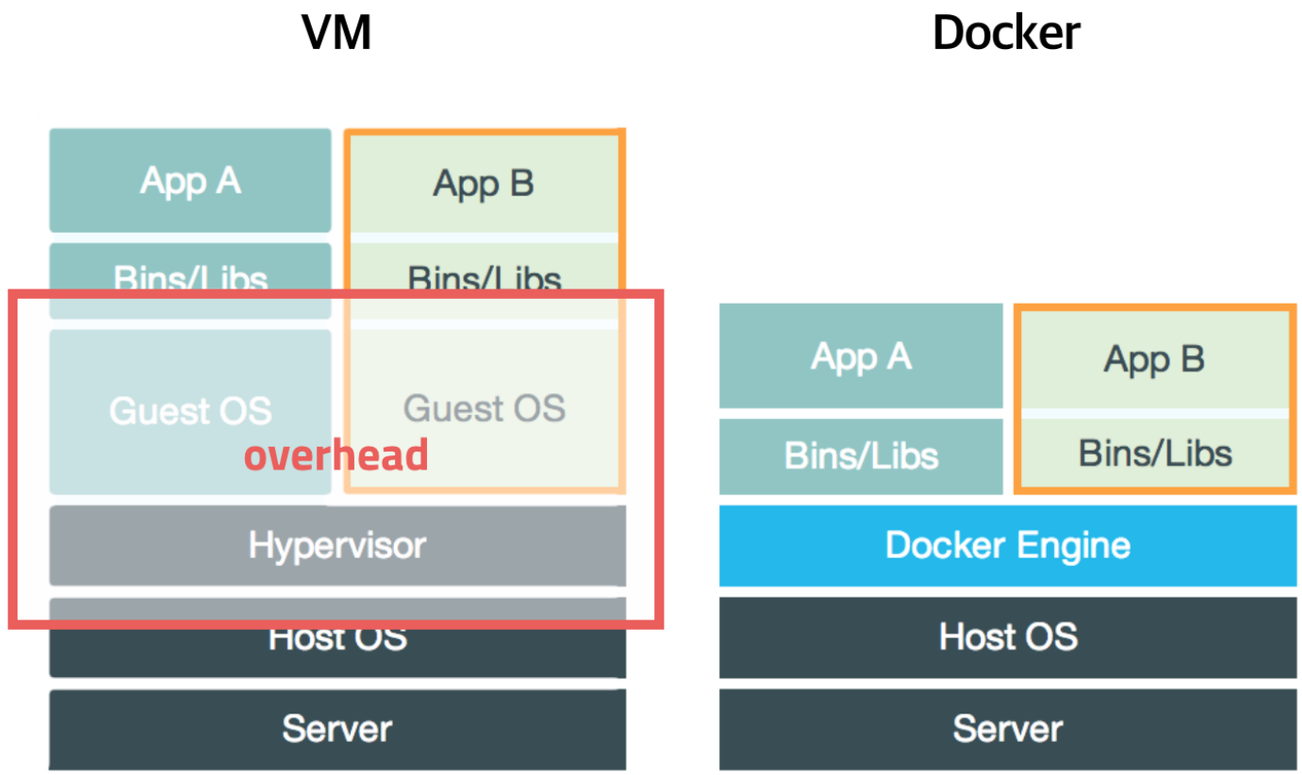
Our stack

Docker CE

- Container-based virtualization
- https://www.docker.com/
  - Install on Ubuntu
  - Install on macOS
  - Install on Windows 10

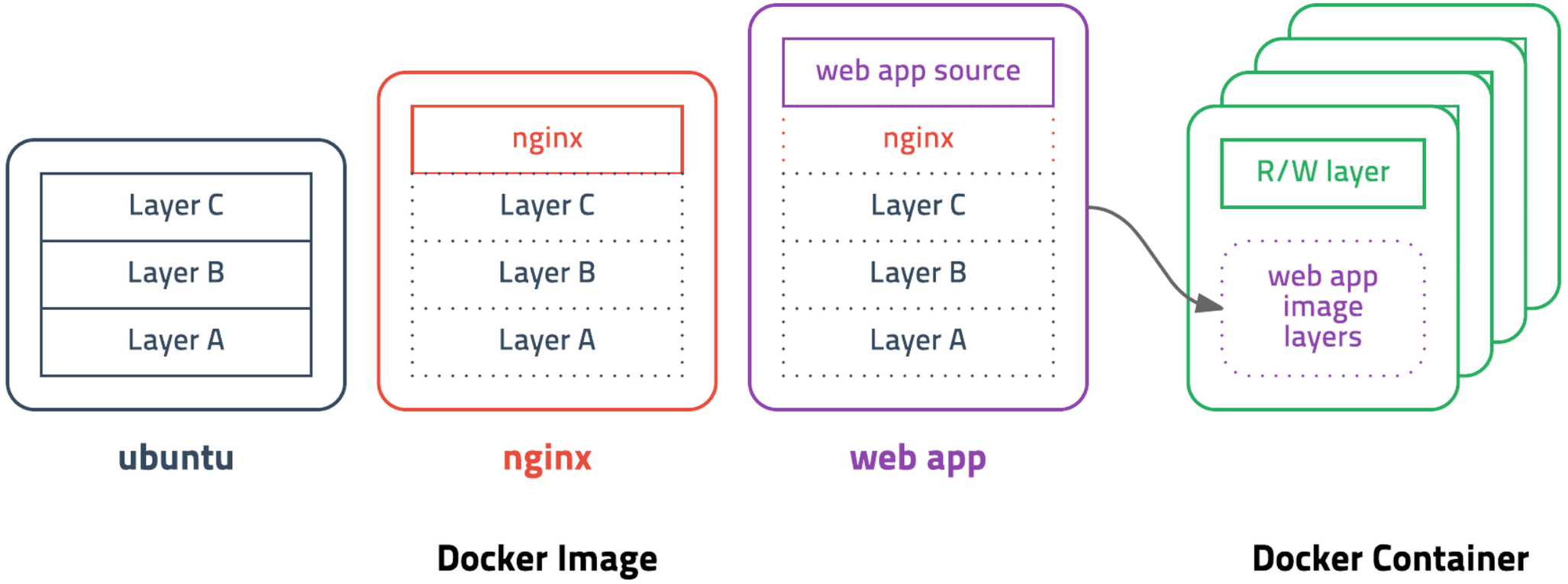
Our stack

Docker CE



Our stack

Docker CE



Our stack

Docker CE

- Commands (requires `sudo` privileges)
  - Create & start a new container

```
$ docker run [OPTIONS] IMAGE
```

  - Stop a running container

```
$ docker stop CONTAINER
```

Our stack

Docker CE

- Commands (requires `sudo` privileges)
  - Start a stopped container

```
$ docker start CONTAINER
```

  - Delete a container

```
$ docker rm CONTAINER
```

Our stack

Docker CE

- Commands (requires `sudo` privileges)
  - Copy files into the container

```
$ docker cp FILE CONTAINER:PATH
```

Installation guides

- PyTorch, PyCharm, Windows 10
- AWS에 PyTorch 작업환경 꾸리기
- Windows Subsystem for Linux에 PyTorch 설치하기

Image Classification with PyTorch

Fast campus

Preparation

1. Install **Anaconda**.
   - https://conda.io/ > Next > Installation > Regular installation > Choose your OS
2. In terminal(Ubuntu and macOS) or Anaconda Prompt(Windows), upgrade Conda and create a new virtual environment named `pytorch-nlp`.

```
$ conda install conda
$ conda create -y --name pytorch-nlp python=3.6 numpy pyyaml scipy
ipython mkl tqdm
```

Preparation

3. Install **PyTorch** on the new environment (this may take a while).
    - Ubuntu (terminal)

```
$ conda install -n pytorch-nlp pytorch-cpu torchvision-cpu -c
pytorch
```

    - macOS (terminal)

```
$ conda install -n pytorch-nlp pytorch torchvision -c pytorch
```

    - Windows (Anaconda Prompt)

```
$ conda install -n pytorch-nlp pytorch-cpu -c pytorch
$ activate pytorch-nlp
$ pip install torchvision
```

Preparation

4. Create a new kernel for **Jupyter Notebook**.
   - Ubuntu and macOS (terminal)

```
$ source activate pytorch-nlp
$ pip install ipykernel
$ python -m ipykernel install -n pytorch-nlp --display-name
"Python 3 (pytorch-nlp)"
```

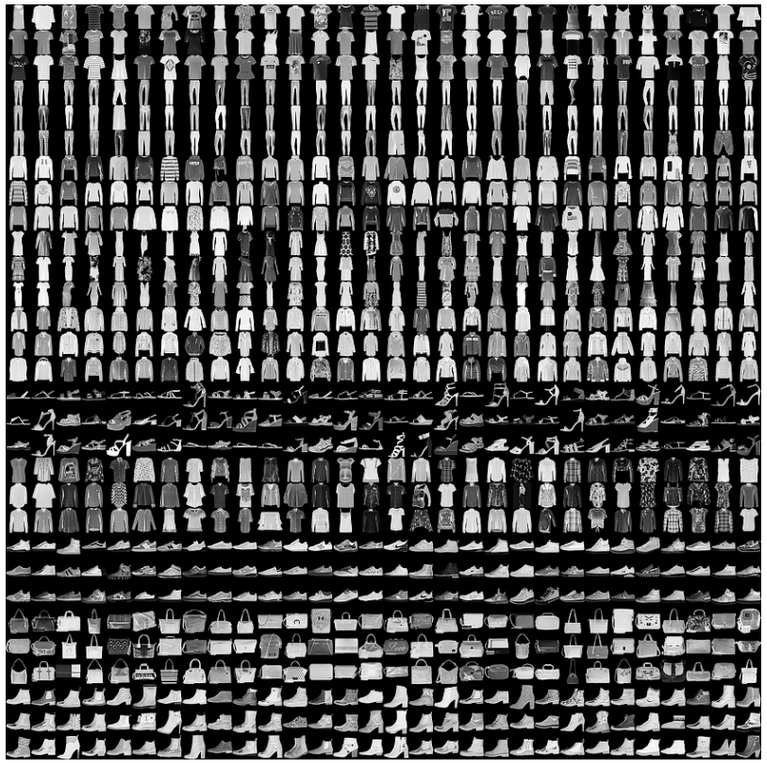
   - Windows (Anaconda Prompt)

```
$ activate pytorch-nlp
$ pip install ipykernel
$ python -m ipykernel install -n pytorch-nlp --display-name
"Python 3 (pytorch-nlp)"
```

The data

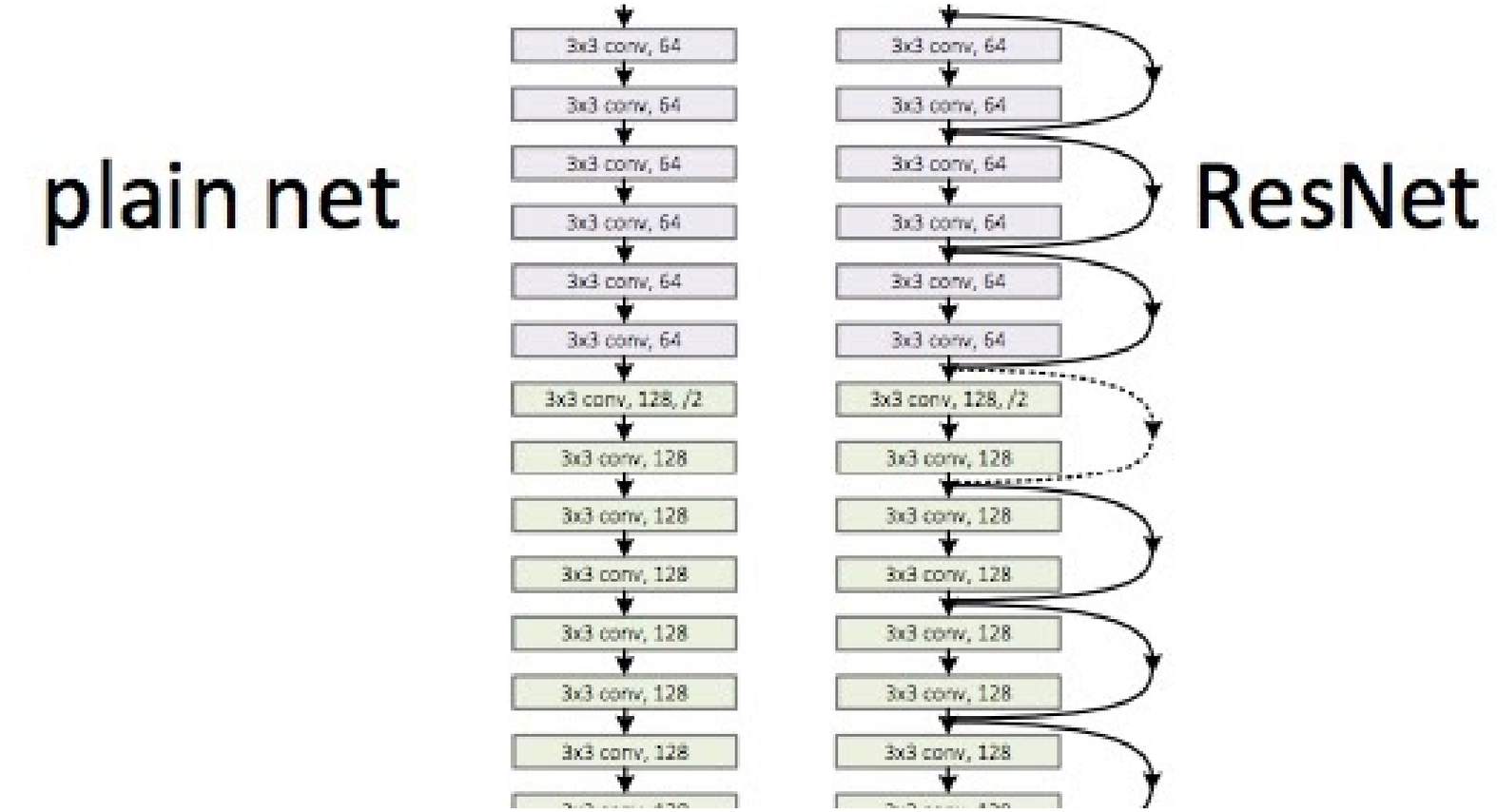
FashionMNIST

- Zalando's clothing product images
- 28-pixel-square grayscale images
- 60k examples for training, 10k samples for testing
- 10 classes



The model

ResNet



The source

`git clone` or download
https://github.com/juneoh/sample_pytorch_project

- `Dockerfile` if you want to use Docker.
- `README.md` the repository description.
- `main.py` the main code.
- `requirements.txt` the package requirements to run this example, for `pip`.

The process

1. Prepare the data: training, validation, test.
2. Create the model and the loss function.
3. Create the optimizer and attach it to the model.
4. For each epoch, train, evaluate and save model.
5. Finally, evaluate the model on the test dataset.

The process

🤔 Why not use cross-validation?

Yoshua Bengio, My lab has been one of the three that started the deep learning approach, back in 2006, along with Hinton's...

Answered Jan 20, 2016 · Upvoted by Naran Bayanbat, MSCS with focus in machine learning and Boxun Zhang, Data Scientist at Spotify; PhD in Computer Science · Author has **172** answers and **3.6m** answer views

We mostly have large datasets when it is not worth the trouble to do something like k-fold cross-validation. We just use a train/valid/test split. Cross-validation becomes useful when the dataset is tiny (like hundreds of examples), but then you can't typically learn a complex model.

36.6k Views · View Upvoters

https://www.quora.com/Is-cross-validation-heavily-used-in-deep-learning-or-is-it-too-expensive-to-be-used

Into the code!

https://github.com/juneoh/sample_pytorch_project


WHEN YOU FIND THE
RIGHT HYPERPARAMETERS
FROM THE FIRST RUN

Thank you!