

# Natural Language Processing with PyTorch

**Week 1 딥러닝을 위한 PyTorch 실무환경 구축**

# Quick preparation

## 1. Install Anaconda.

- <https://conda.io/> > Next > Installation > Regular installation > Choose your OS

## 2. Open Anaconda console, and create a new virtual environment.

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

## 3. Install PyTorch on the new environment (this may take a while).

- `conda install --name pytorch-nlp pytorch-cpu torchvision -c pytorch`

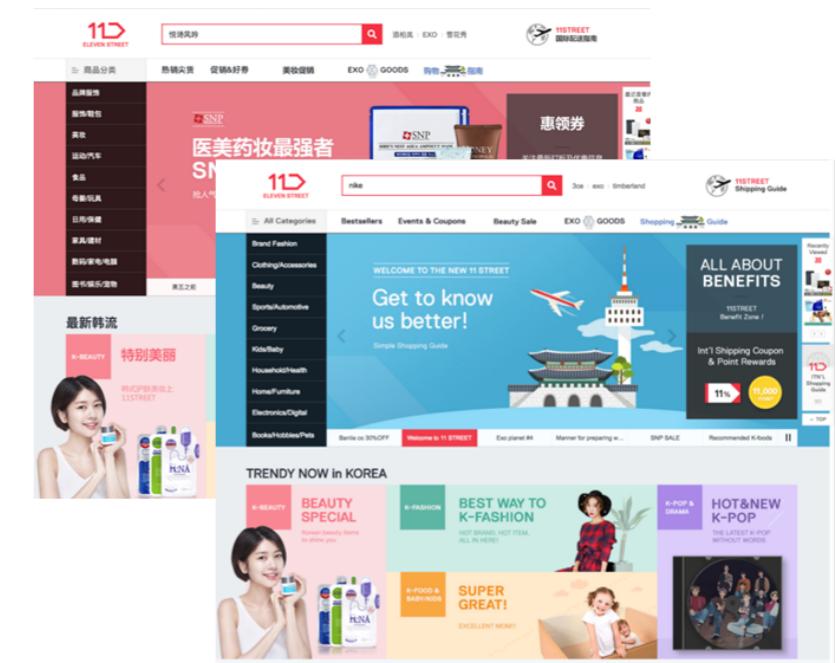
# Ki Hyun Kim

- Machine Learning Researcher @ MakinaRocks
- Linkedin: <https://www.linkedin.com/in/ki-hyun-kim/>
- Github: <https://github.com/kh-kim/>
- Email: [pointzz.ki@gmail.com](mailto:pointzz.ki@gmail.com)



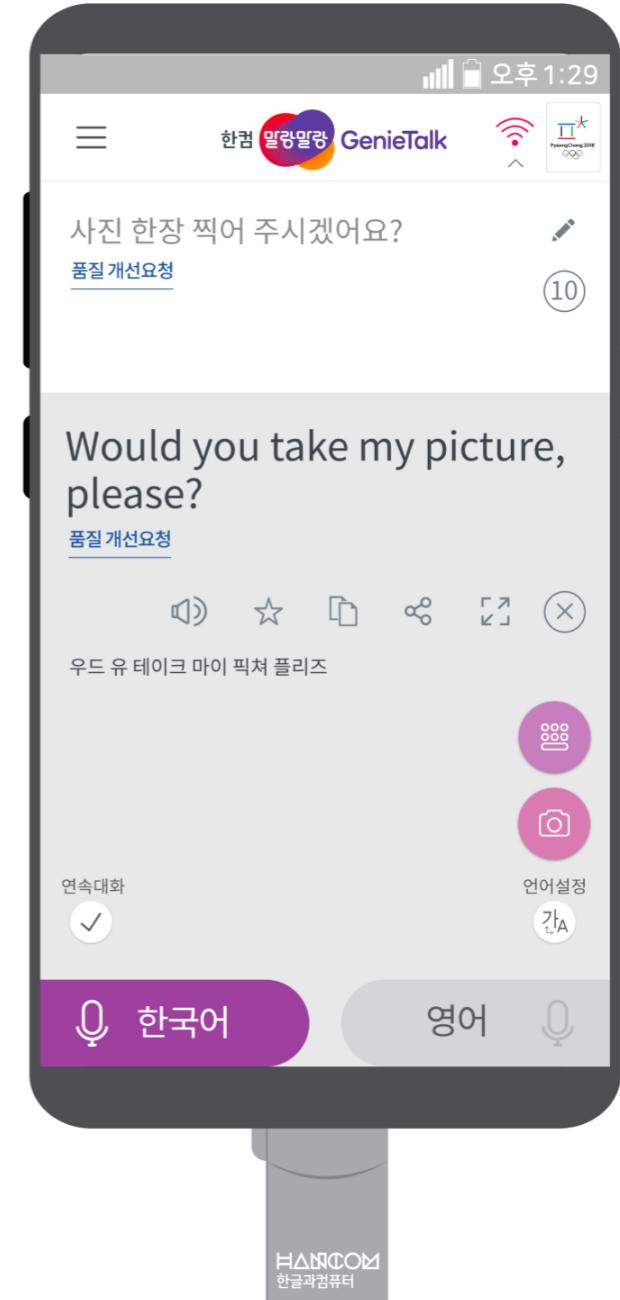
# Ki Hyun Kim

- 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 스케줄링 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](mailto:me@juneoh.net)

# 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

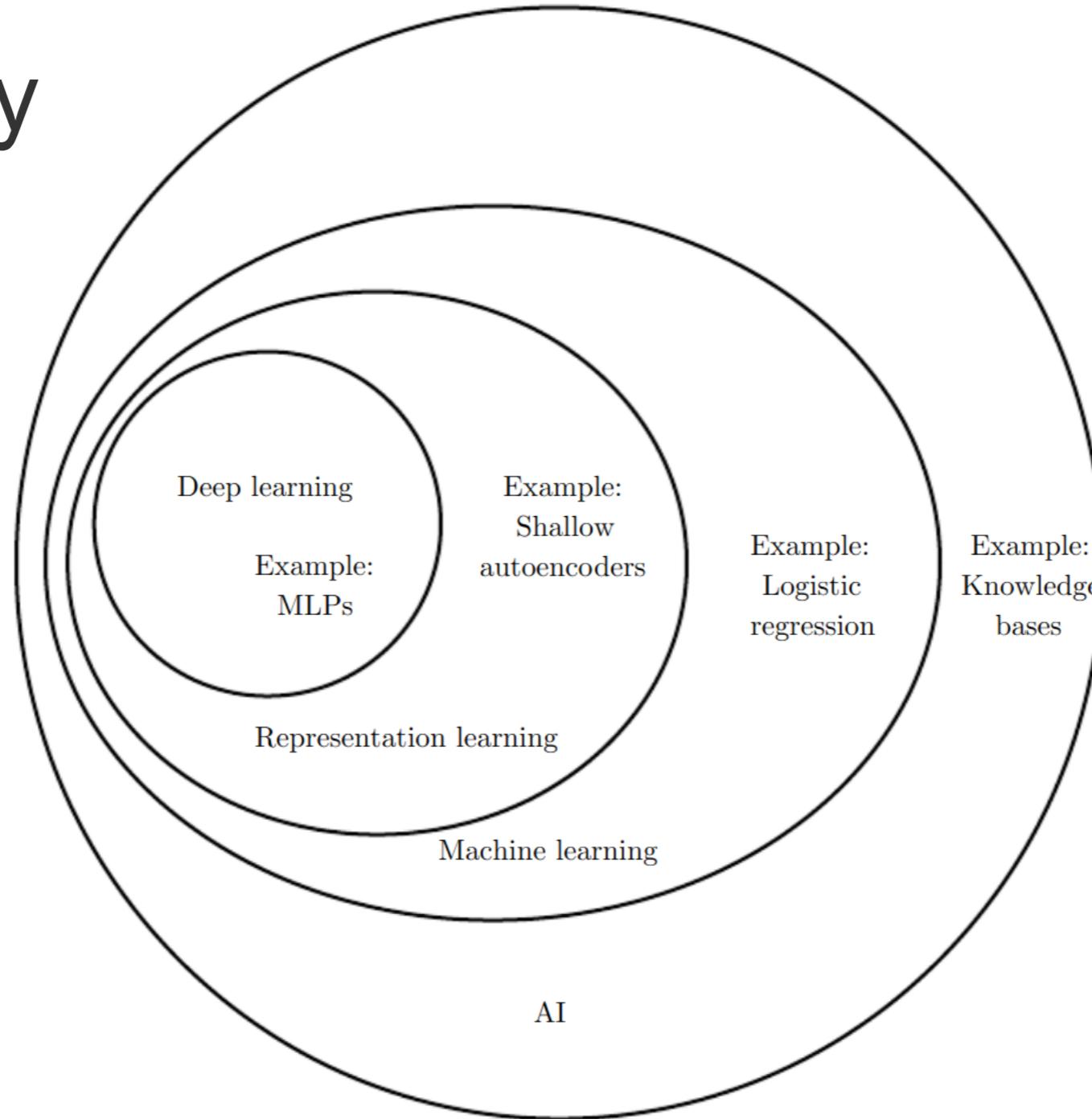
# Course mechanics

- 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

# 1. Introduction to Deep Learning



# Genealogy



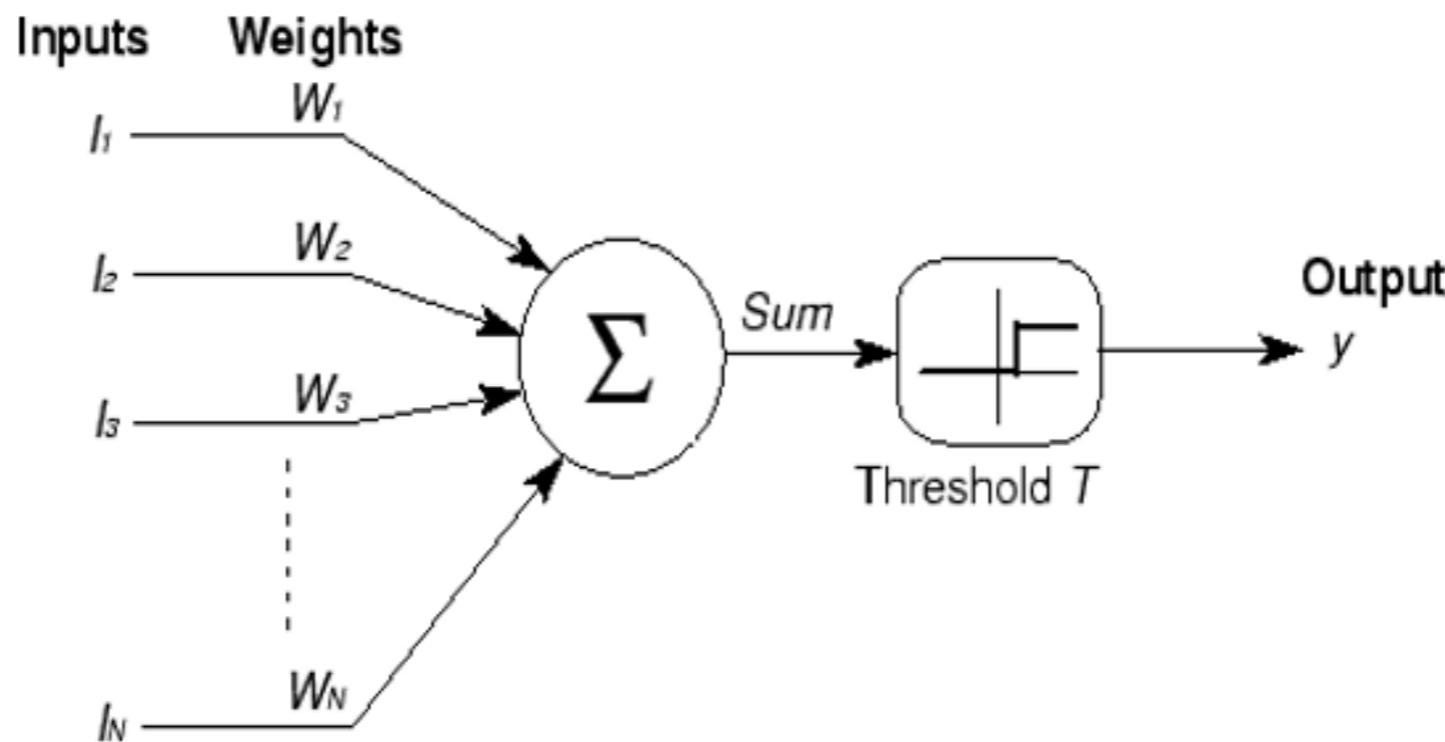
# Genealogy



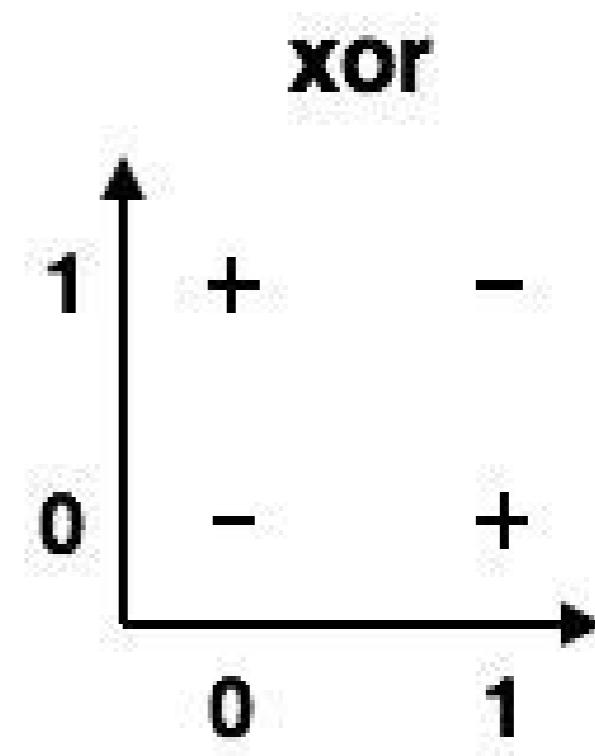
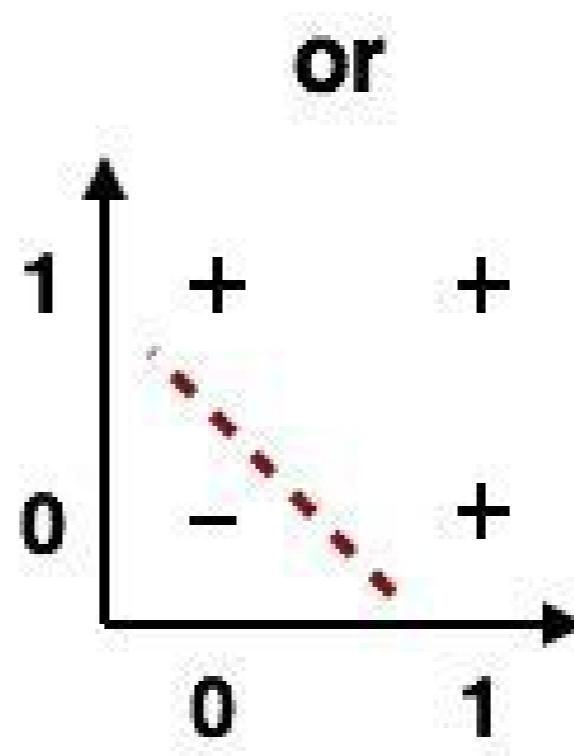
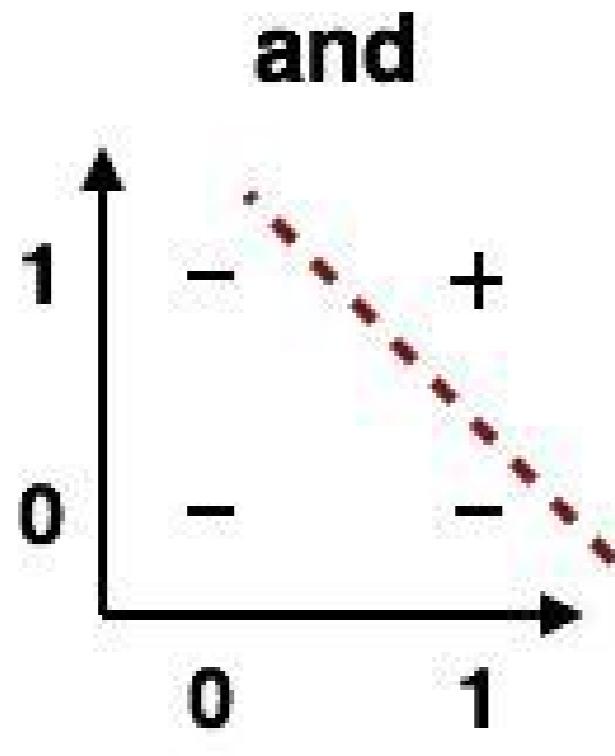
# Timeline

- **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



# 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.

# 이젠, 빠빠라고 하지 않는다!

초고속 인공지능 스피드012 - 빠빠라고 하기엔 너무 똑똑하다!

## 파워 VMS로 빠른 3G!

인터넷 20Mbps, 폰서버 30Mbps  
1회당 1시간 30Mbps  
인터넷 기준이며 대역 관리됩니다.

## 지역변경인식은 차운으로!

전국 어디에서나 내가 찾는 위치와  
현재의 지역을 차운으로 인식  
변경마다 즉시 통신해요.

## 세계표준시작을 확장으로!

전화기를 고매우도, 편의기 다시 거두.  
위성을 통해 세계로 펼쳐온 시장을 확장하세요.

## 모든 서비스를 빠른 하나로!

전국 어디에서나 빠른 통·인터넷·인터넷  
전화, 웹·인터넷·인터넷 서비스를 모두 경험할 수 있다.

## 간편하게 셋팅하고 사용까지!

총务이 필요한 간편하게 사용하는 차운인터넷으로  
간편하게 셋팅하고 사용하세요.

## 미수신 폴리시는 차운송!

총务이 있는 원서대로 번호를 빠져  
마무리 폴리시를 스스로 세우고 100% 제한을 끝낸다.

## 인터넷 백화점인 차운샵!

인터넷 쇼핑몰 스스로 판매하여 쉽지  
인터넷 고매우도 편한 그대로 사용할 수 있다.

초고속 인공지능

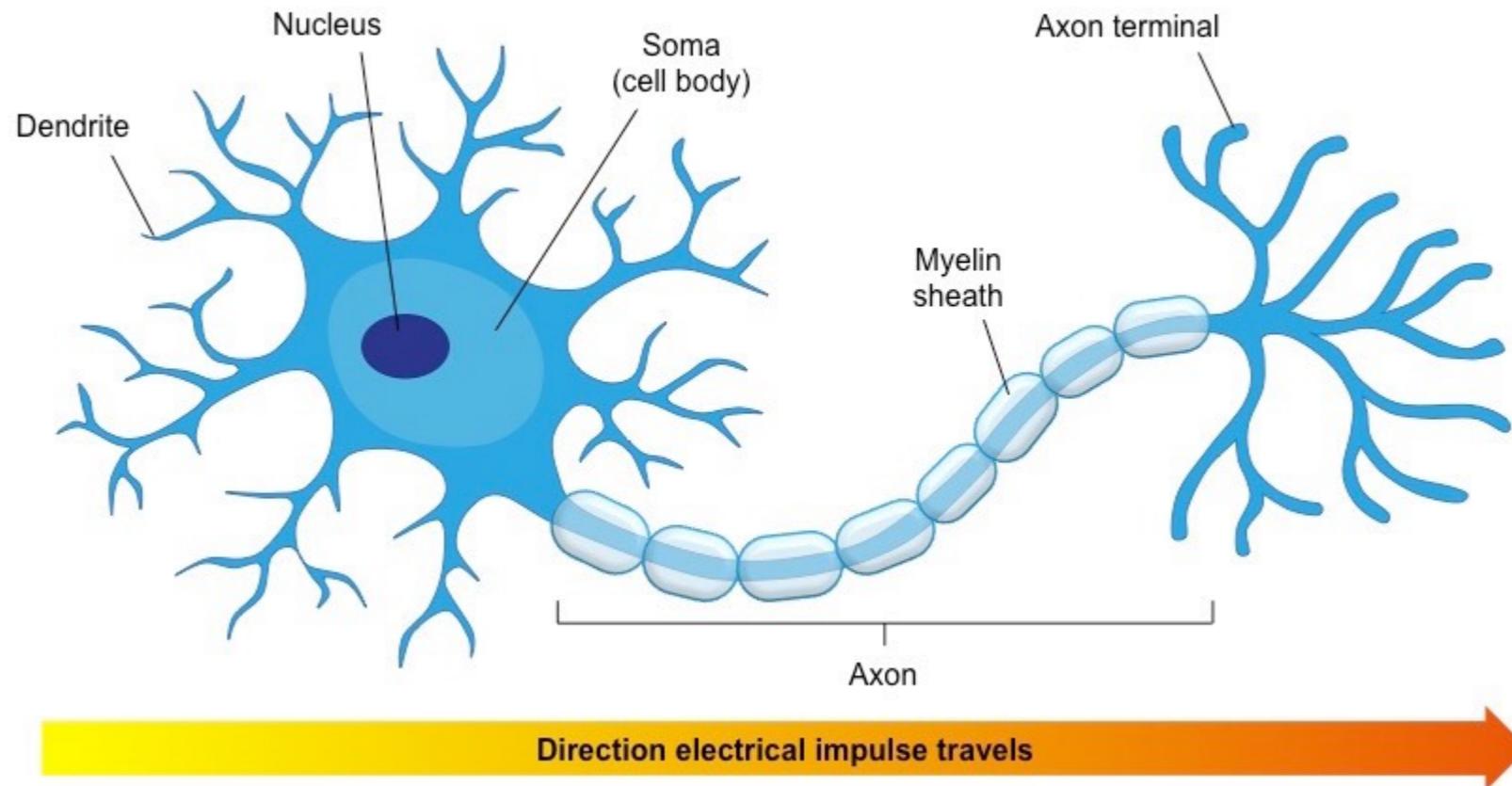
**SPEED 012**

# 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.

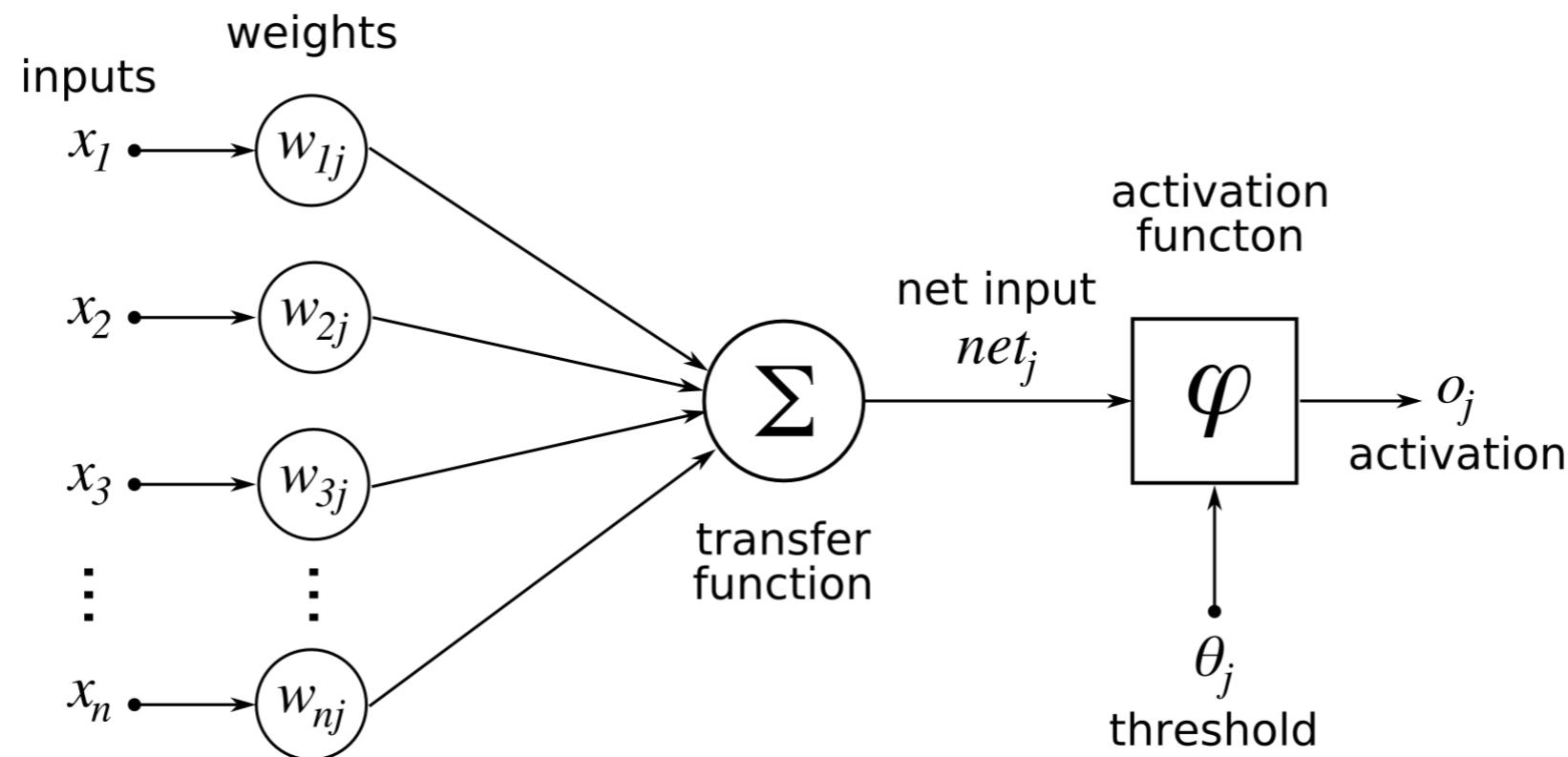
# Neural Networks

- Feed-forward Network



# Neural Networks

- Feed-forward Network



# Neural Networks

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

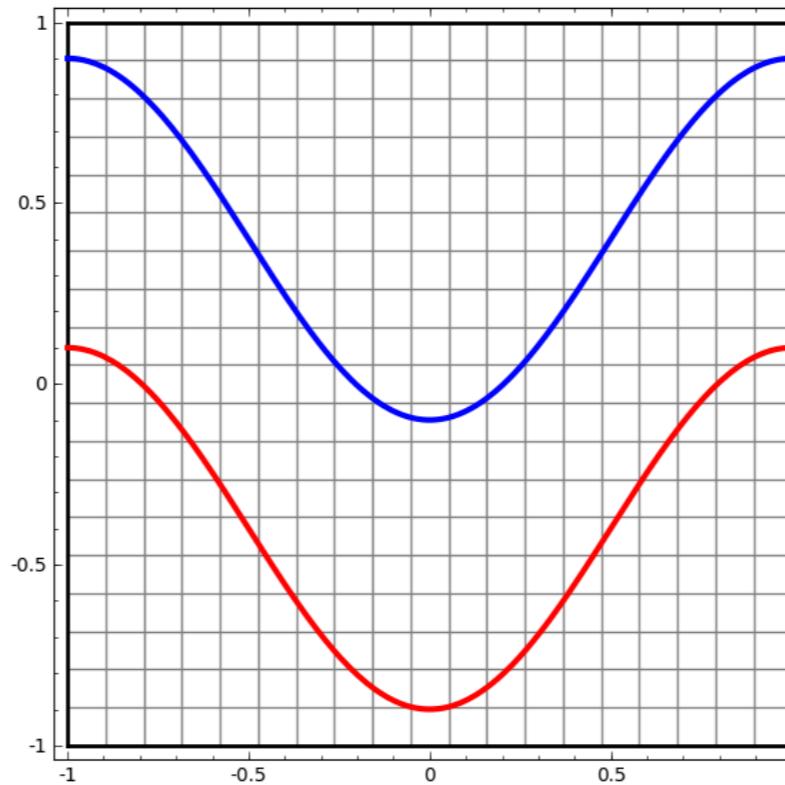
```
def perceptron(inputs, weights, biases):  
    return sum(inputs * weights + biases)
```

- In PyTorch:

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

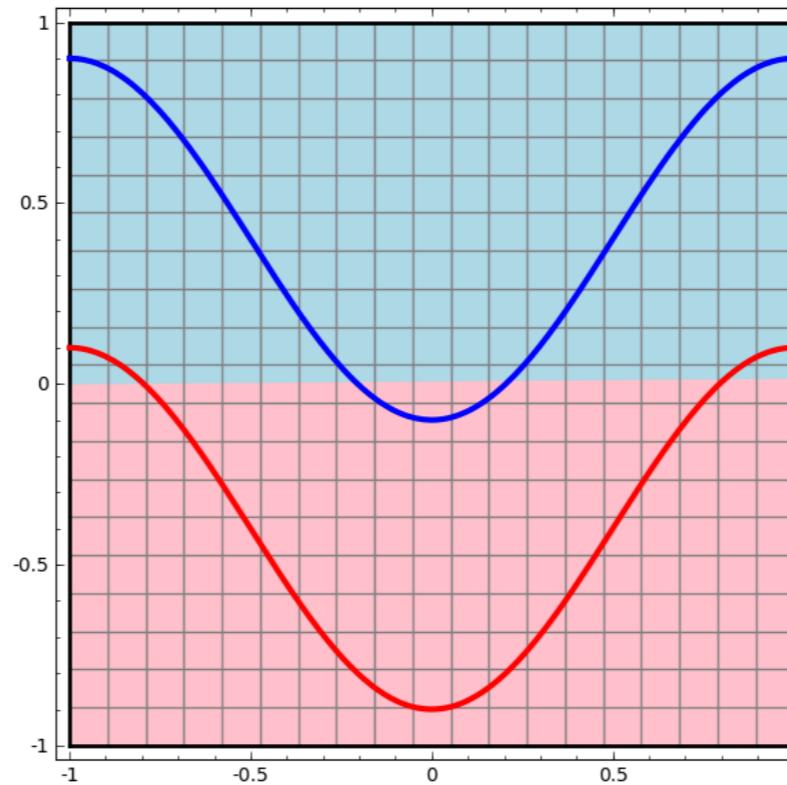
# Neural Networks

Problem: draw a single straight line to separate colors.



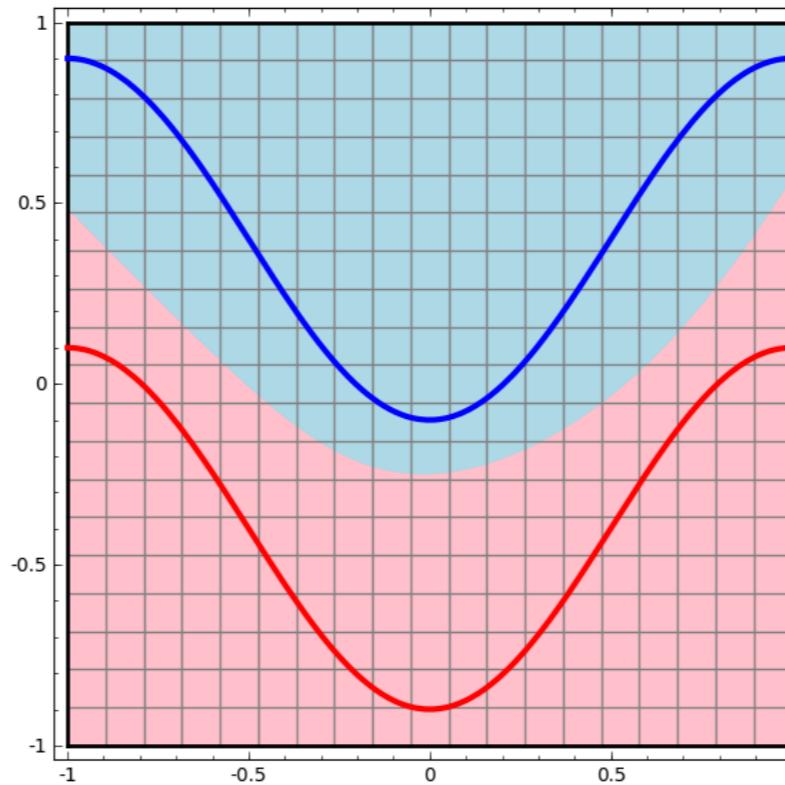
# Neural Networks

Problem: draw a single straight line to separate colors.



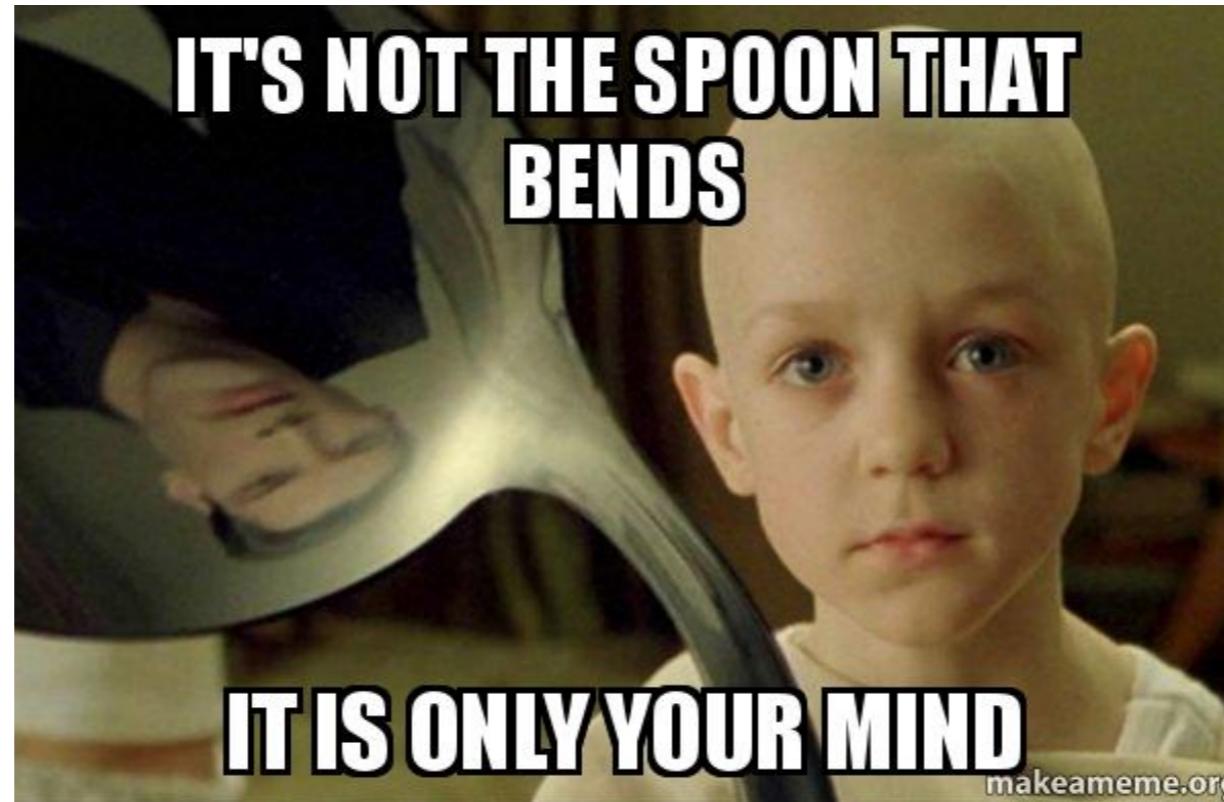
# Neural Networks

Problem: draw a single straight line to separate colors.



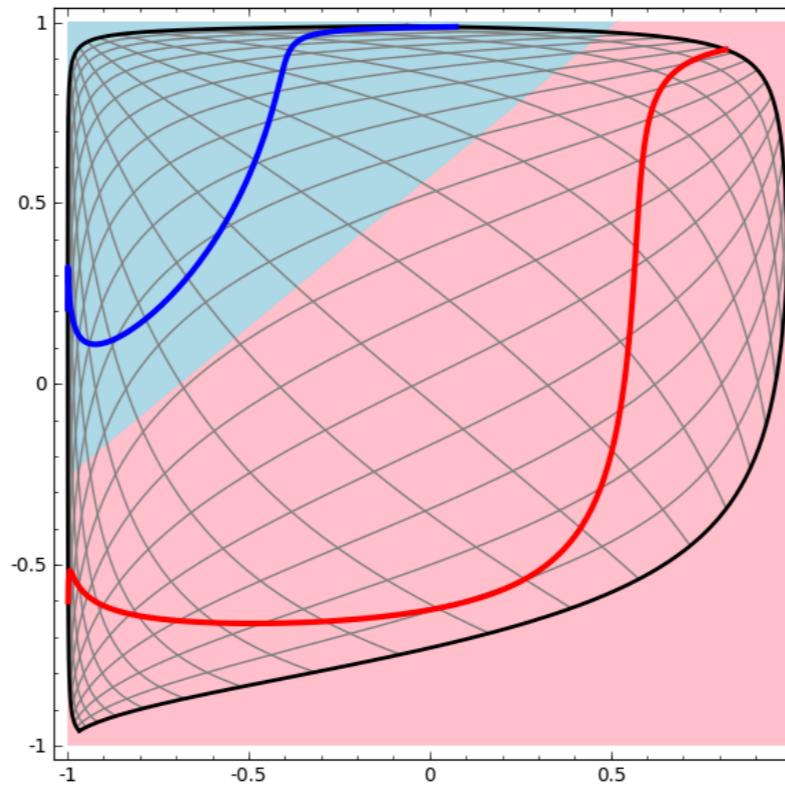
# Neural Networks

Problem: draw a single straight line to separate colors.

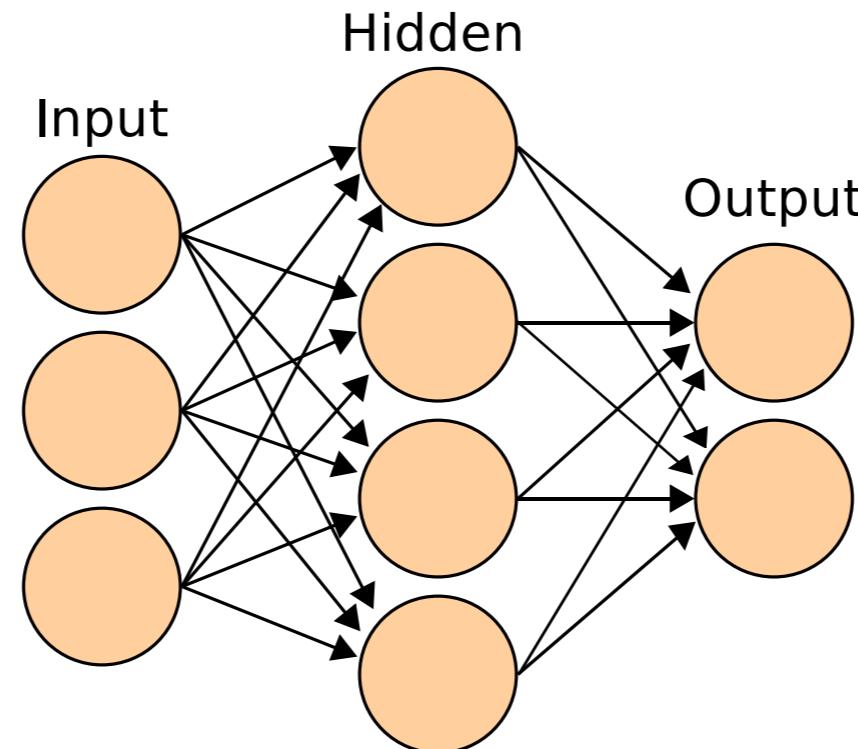


# Neural Networks

Problem: draw a single straight line to separate colors.



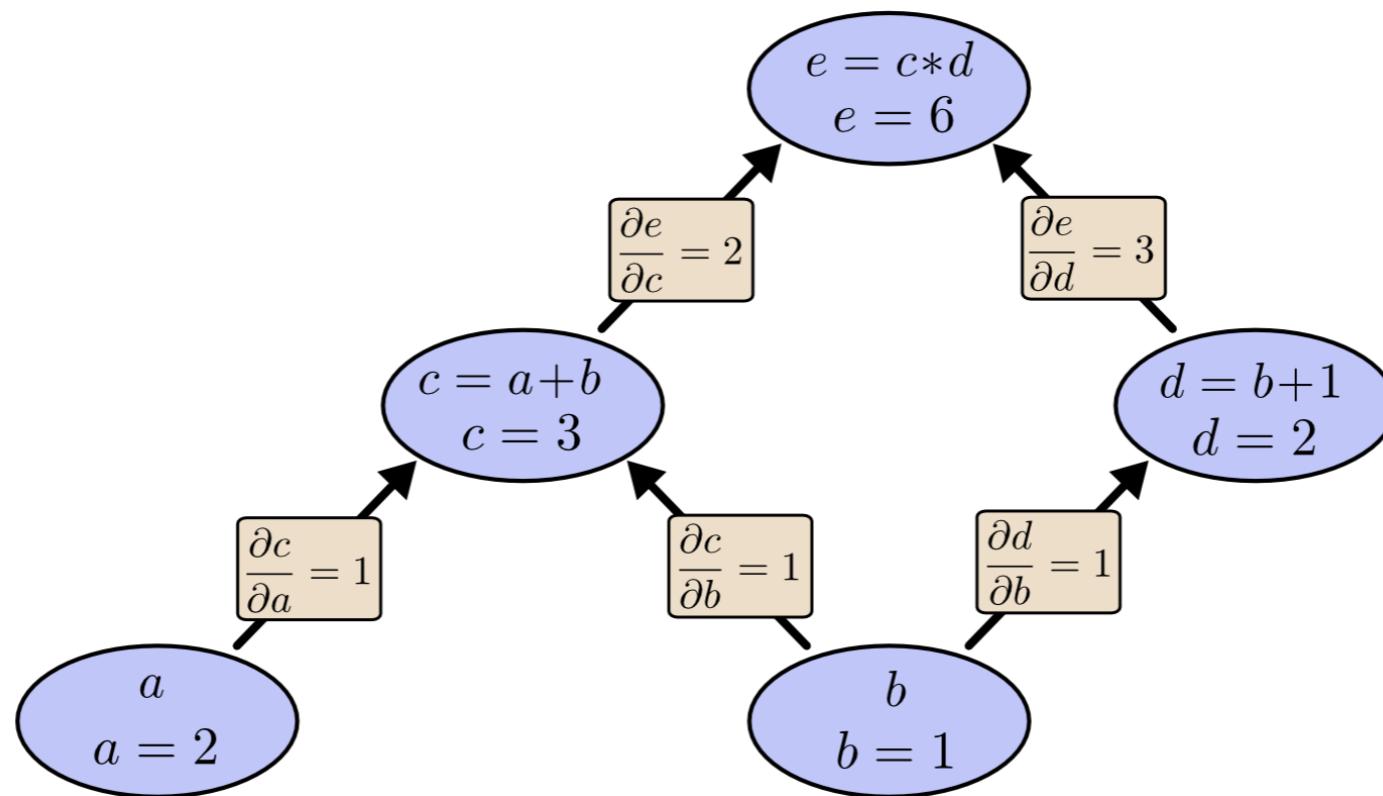
# Neural Networks



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

# Neural Networks

- Backpropagation



# Neural Networks

- Backpropagation
  - In Python (with NumPy):

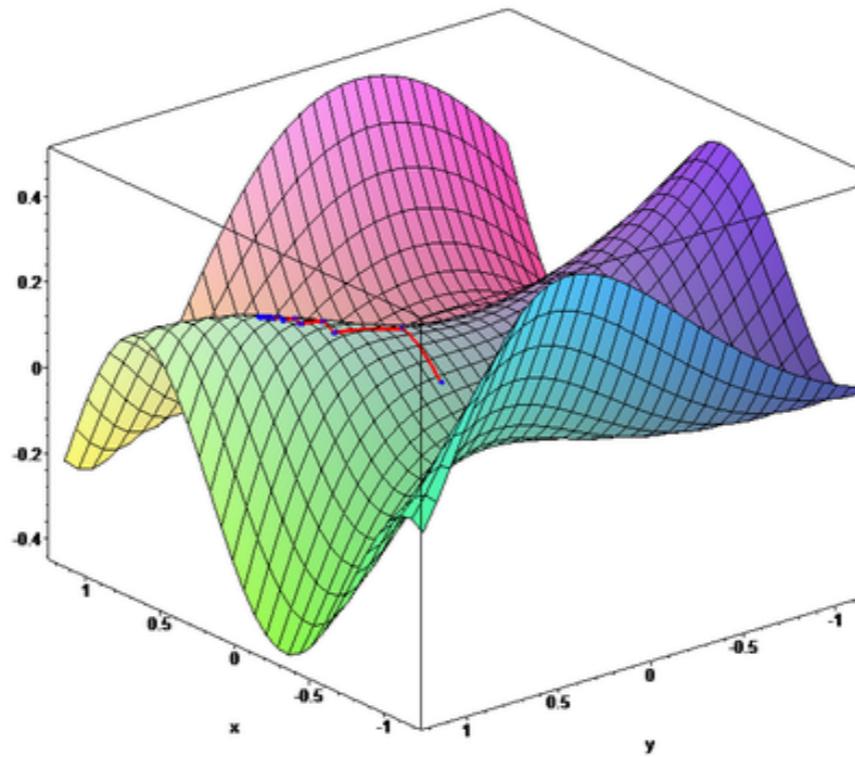
```
def backpropagate(weights, derivative, learning_rate):  
    return weights - learning_rate * (derivative - weights)
```

- In PyTorch:

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

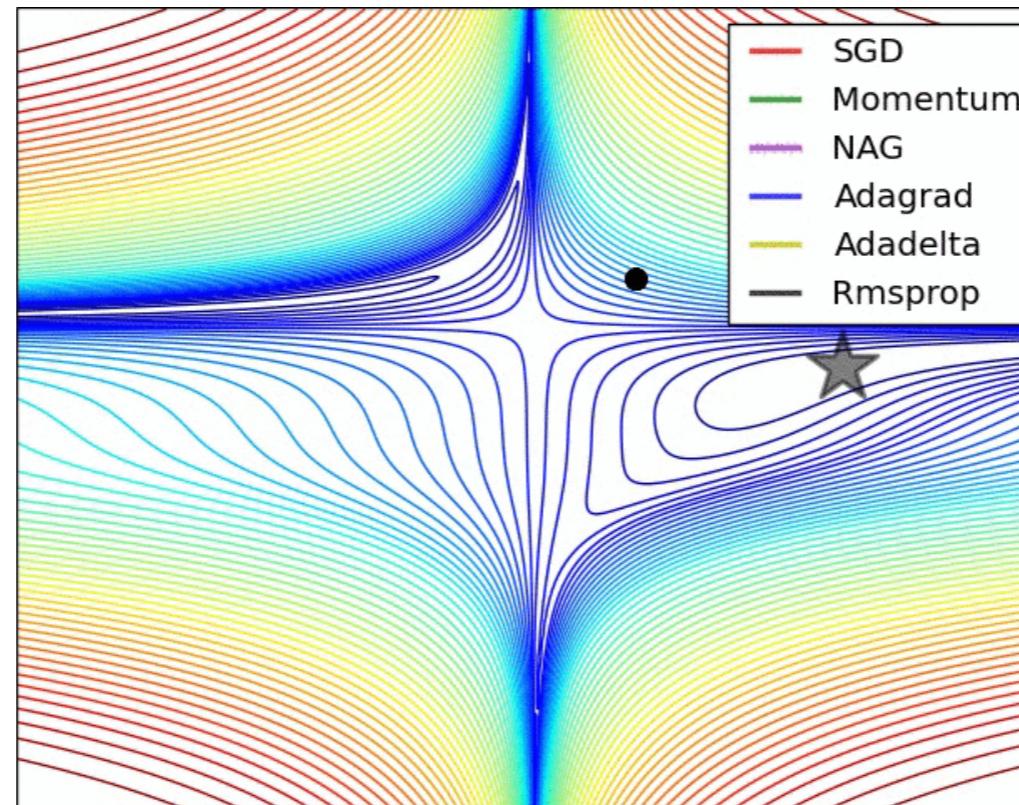
# Neural Networks

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



# Neural Networks

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

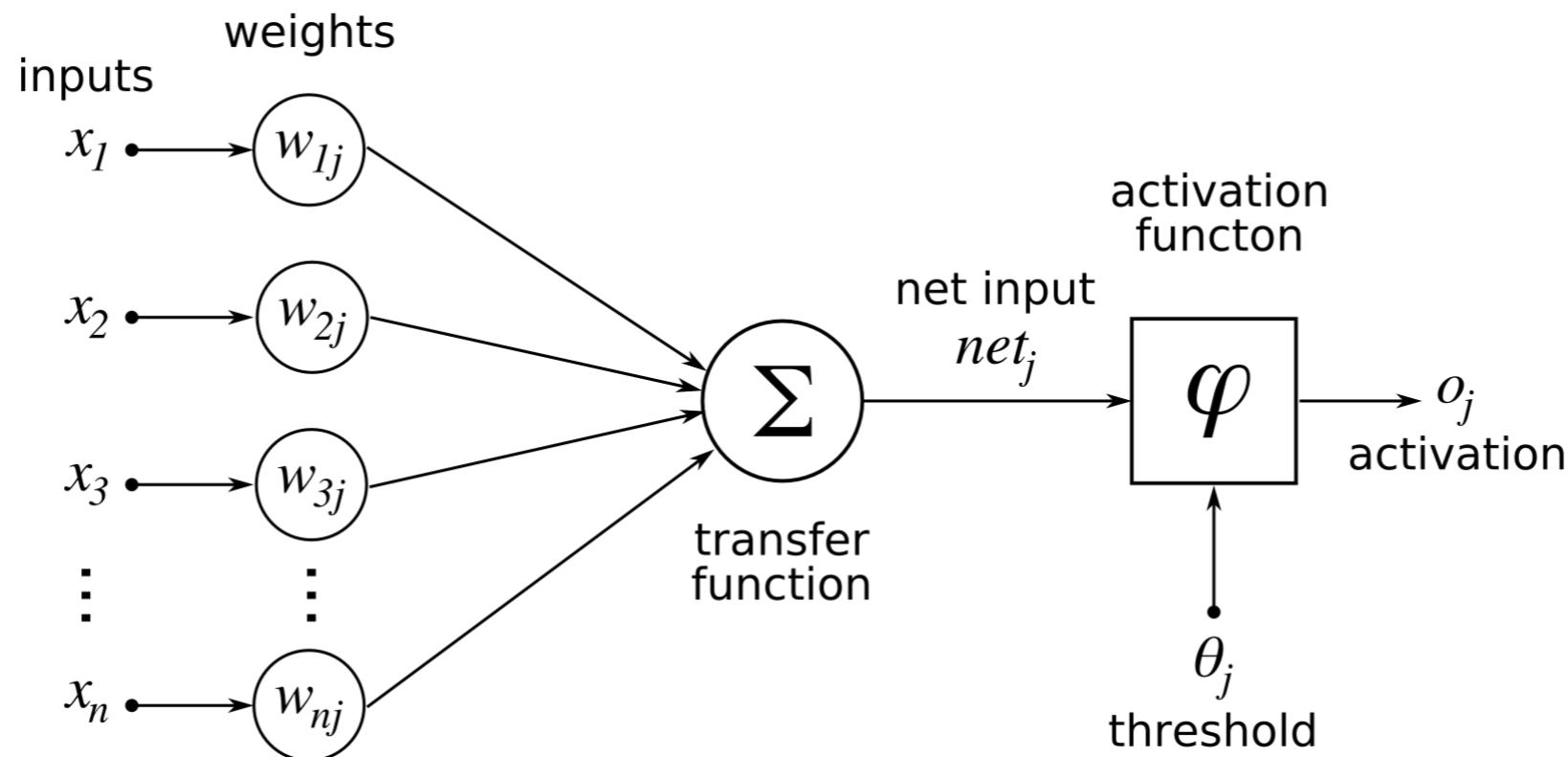


# Neural Networks

- 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`

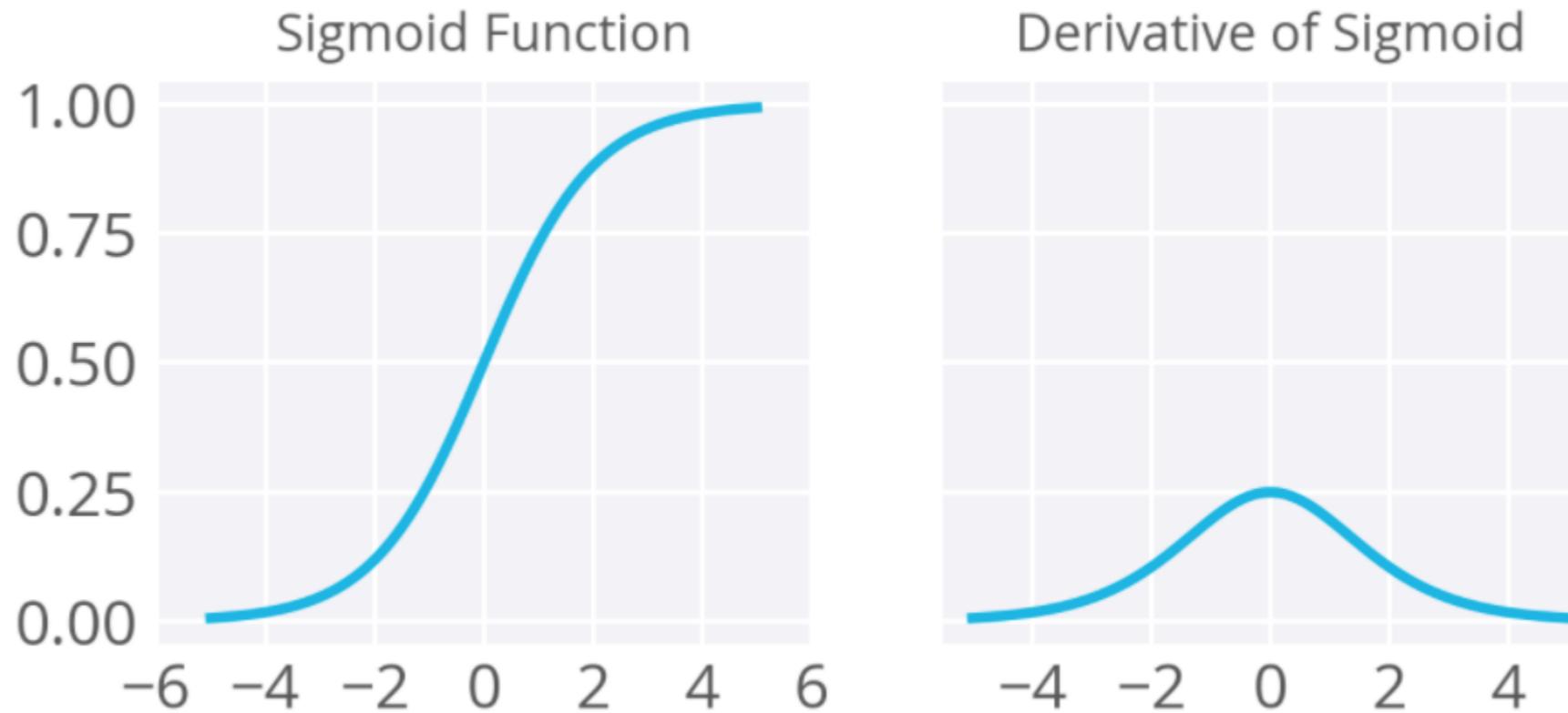
```
optimizer.zero_grad()  
loss.backward()  
optimizer.step()
```

# Activation functions and non-linearity



# Activation functions and non-linearity

- Sigmoid



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

# Activation functions and non-linearity

- Sigmoid

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

- In Python (with NumPy):

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

# Activation functions and non-linearity

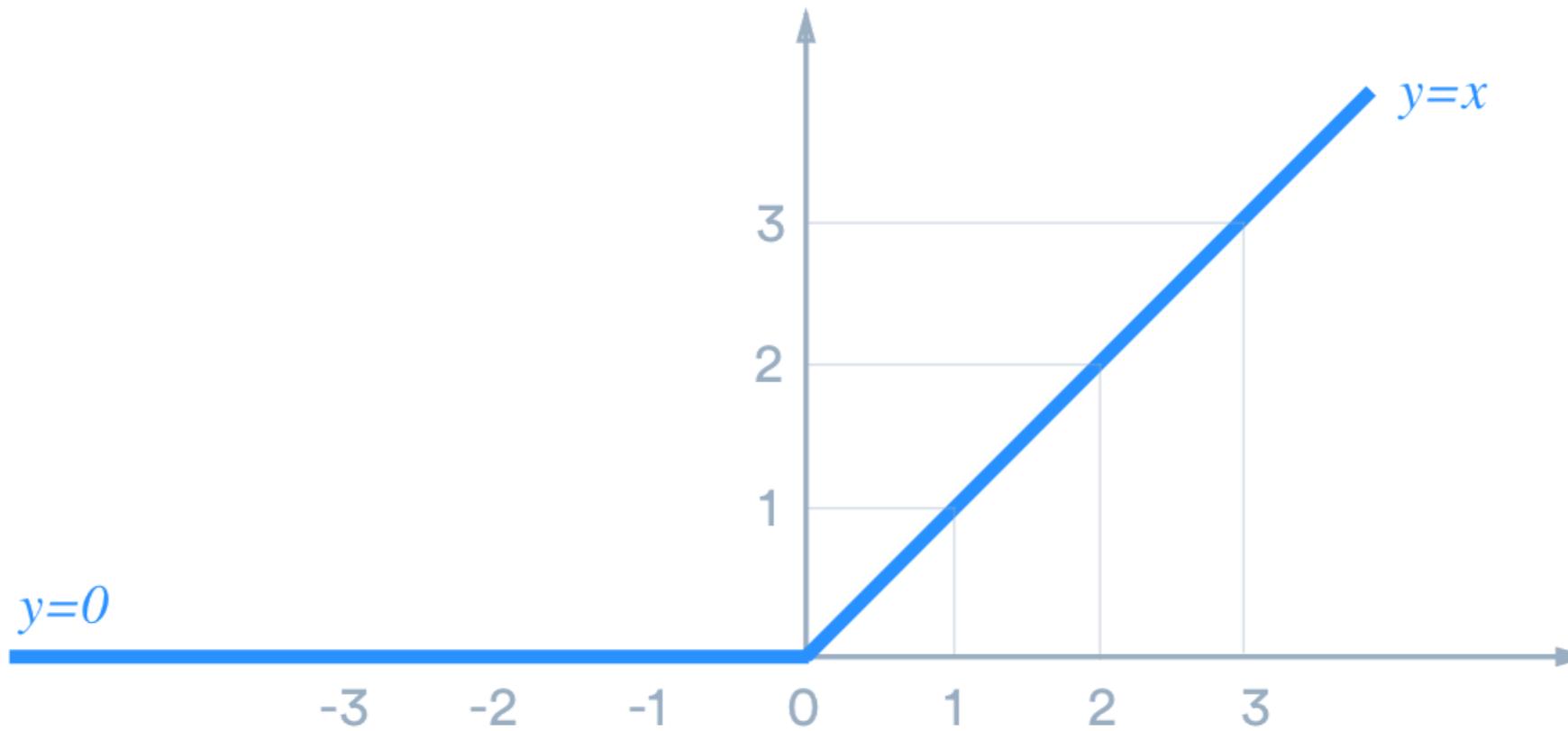
- Sigmoid

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

- 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

- Rectified Linear Units (ReLU)



# Activation functions and non-linearity

- Rectified Linear Units (ReLU)

$$\max(0, x)$$

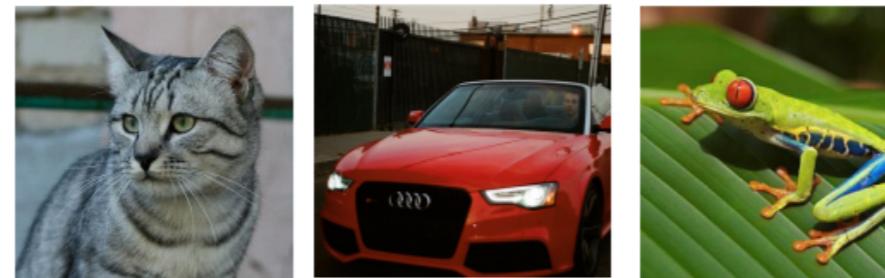
- In Python (with NumPy):

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

- In PyTorch: `torch.nn.ReLU`

# Activation functions and non-linearity

- Softmax



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

# Activation functions and non-linearity

- Softmax



# Activation functions and non-linearity

- Softmax

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

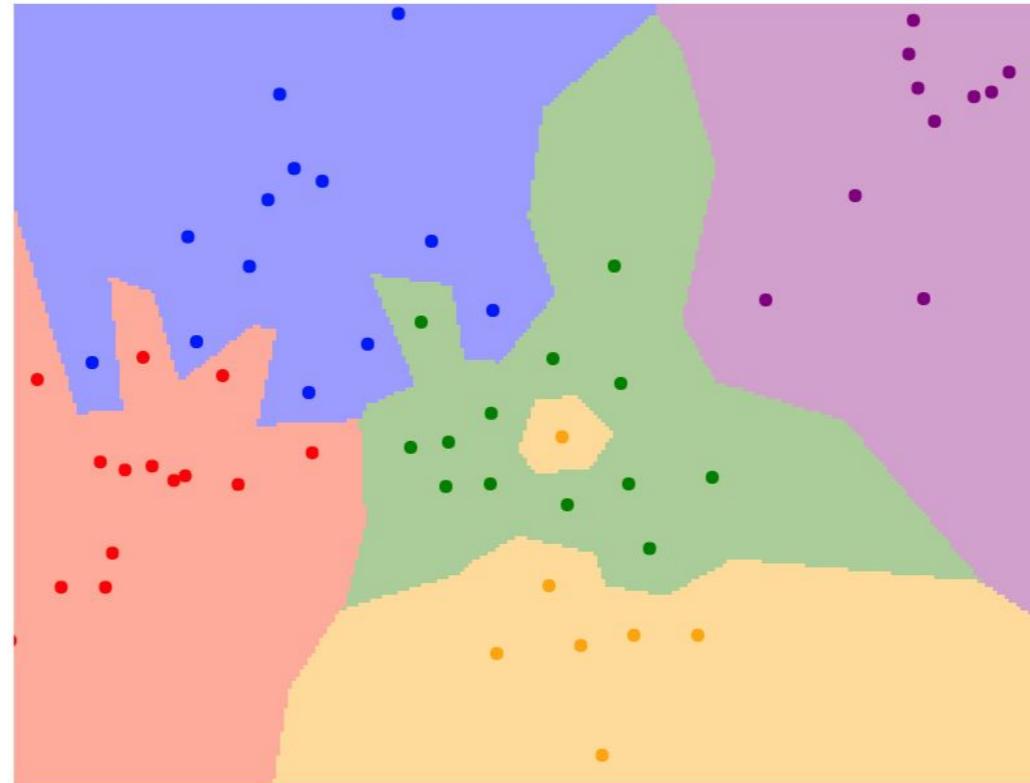
- In Python (with NumPy):

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

- In PyTorch: `torch.nn.Softmax`

# Loss functions

- L1 loss and L2 loss
  - k-Nearest Neighbors

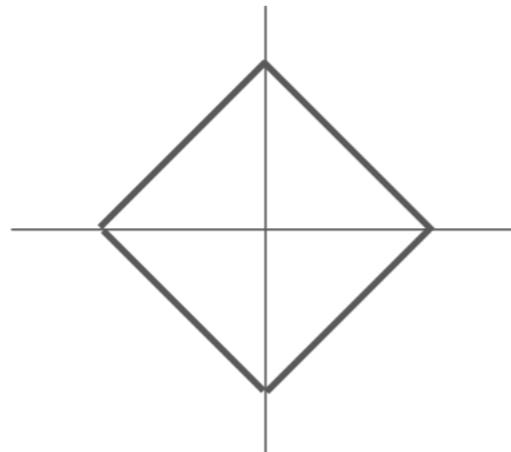


# Loss functions

- L1 loss and L2 loss

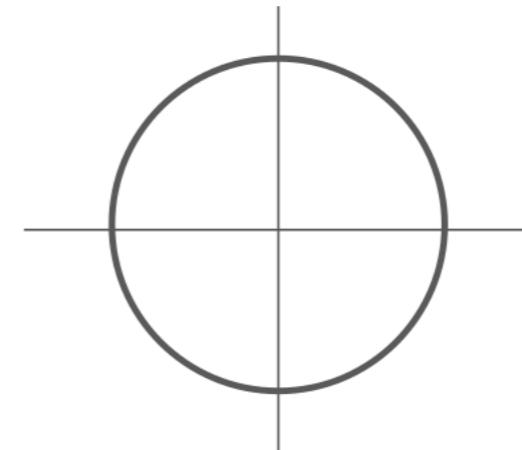
L1 (Manhattan) distance

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



L2 (Euclidean) distance

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



# 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

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

- In Python (with NumPy):

```
def l2_loss(targets, outputs):
    return sum(sqrt((targets - outputs)**2))
```

- In PyTorch: `torch.nn.MSELoss`

# 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

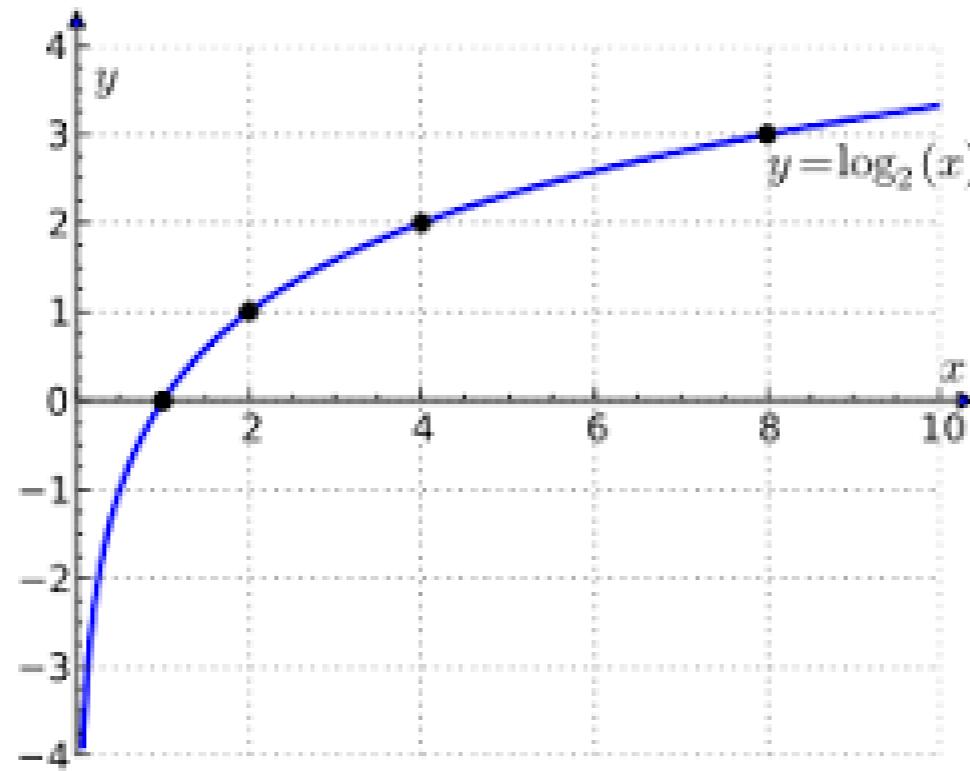
**Entropy**(in information theory)

= 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

The diagram illustrates the Cross Entropy loss function. It shows two vectors:  $\hat{y}$  (predicted) and  $y$  (target). The vector  $\hat{y}$  is shown in a red box and has values [0.1, 0.5, 0.4]. The vector  $y$  is shown in a blue box and has values [0, 1, 0]. A red curved arrow points from the label  $D(\hat{y}, y)$  to the vector  $\hat{y}$ . A blue curved arrow points from the label  $D(\hat{y}, y)$  to the vector  $y$ . Below the vectors, the formula for the Cross Entropy loss is given:

$$D(\hat{y}, y) = - \sum_j y_j \ln \hat{y}_j$$

# 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:

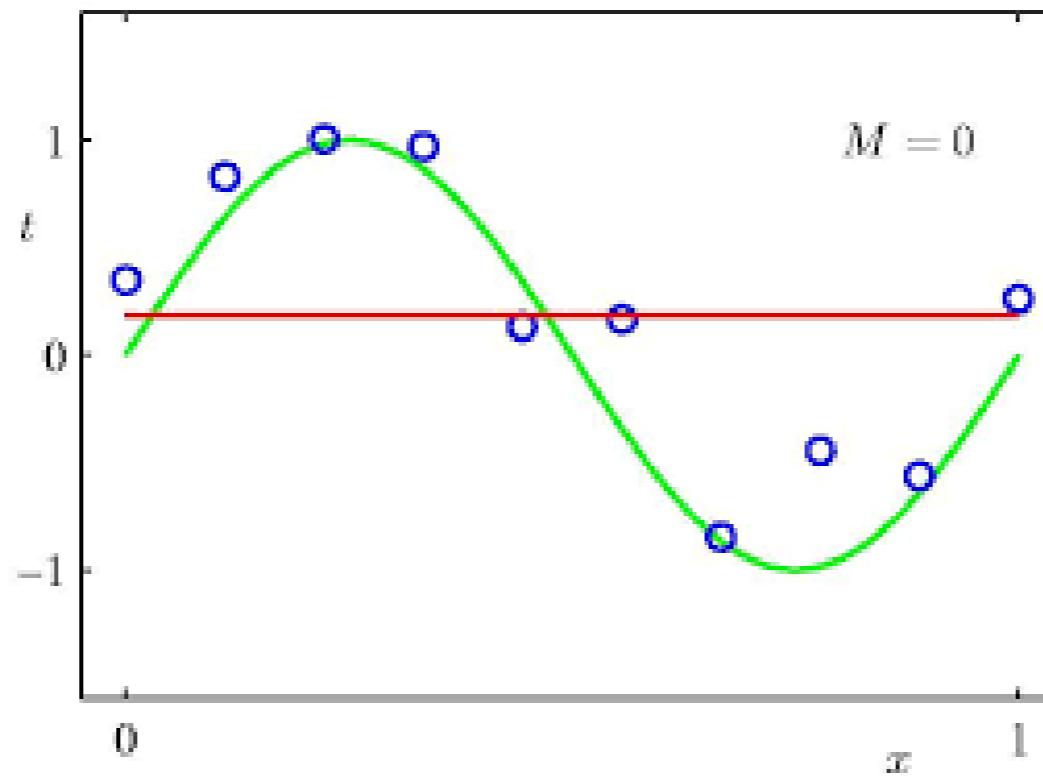
```
def binary_cross_entropy_loss(targets, outputs):
    return -mean(targets * log(outputs) + (1 - targets) * log(1 - outputs))
```

- In PyTorch: `torch.nn.BCELoss`

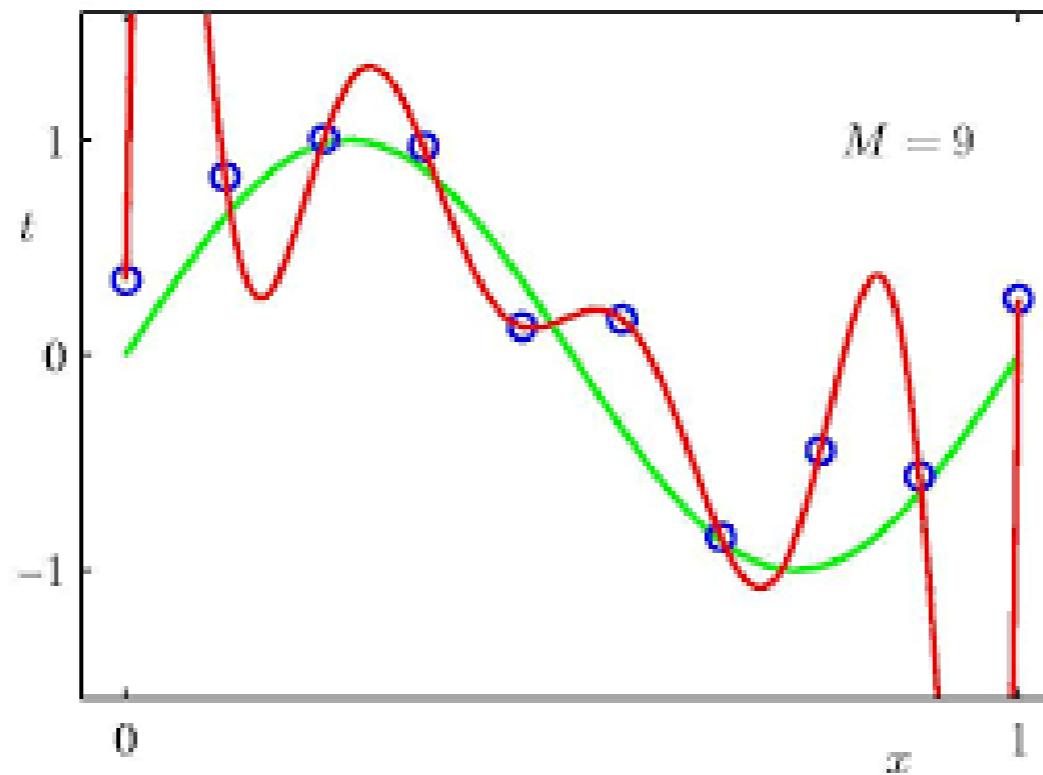
# Loss functions

- In most cases,
  - 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

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

- 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

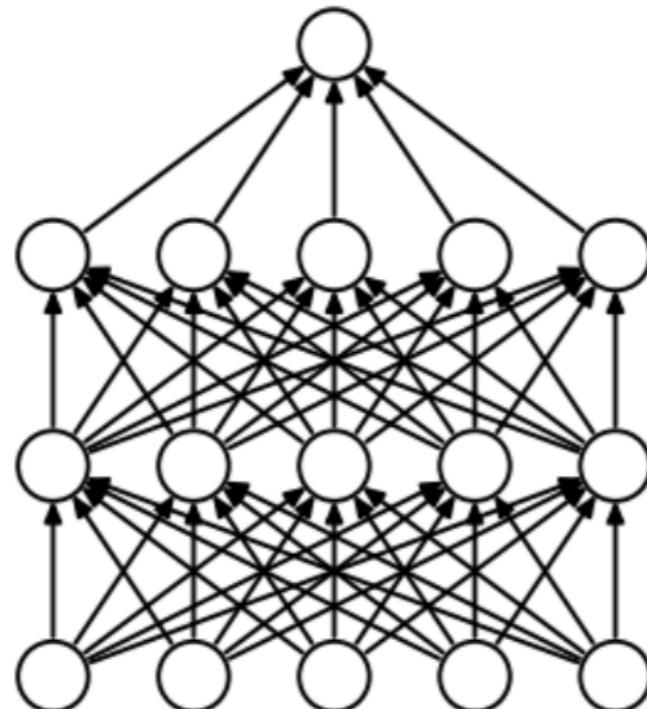
$$W \leftarrow W - \lambda \left( \frac{\partial L}{\partial W} + \gamma \|W\| \right)$$

- In PyTorch:

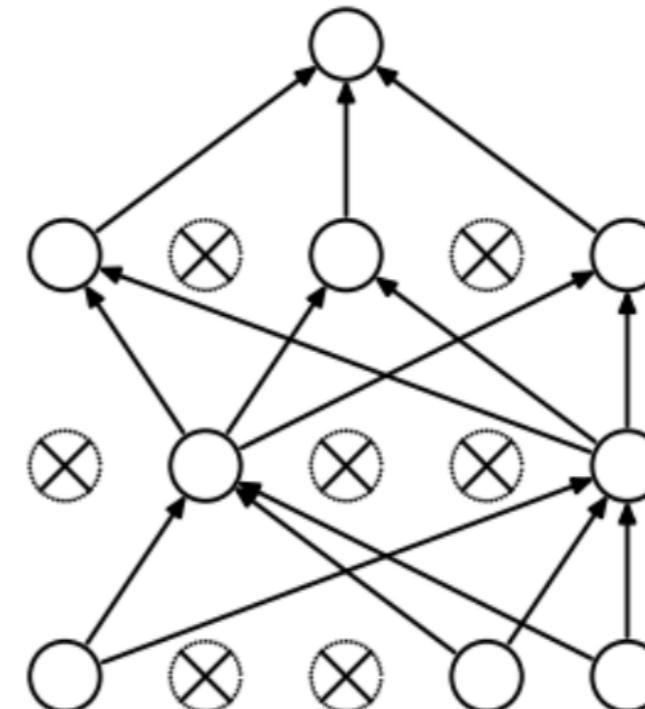
```
optimizer = torch.optim.SGD(learning_rate=0.1, weight_decay=0)
```

# Regularization methods

- Dropout



(a) Standard Neural Net



(b) After applying dropout.

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

# Types of Headaches

Migraine



Hypertension



Stress



MATH BEHIND DL



# Hello PyTorch





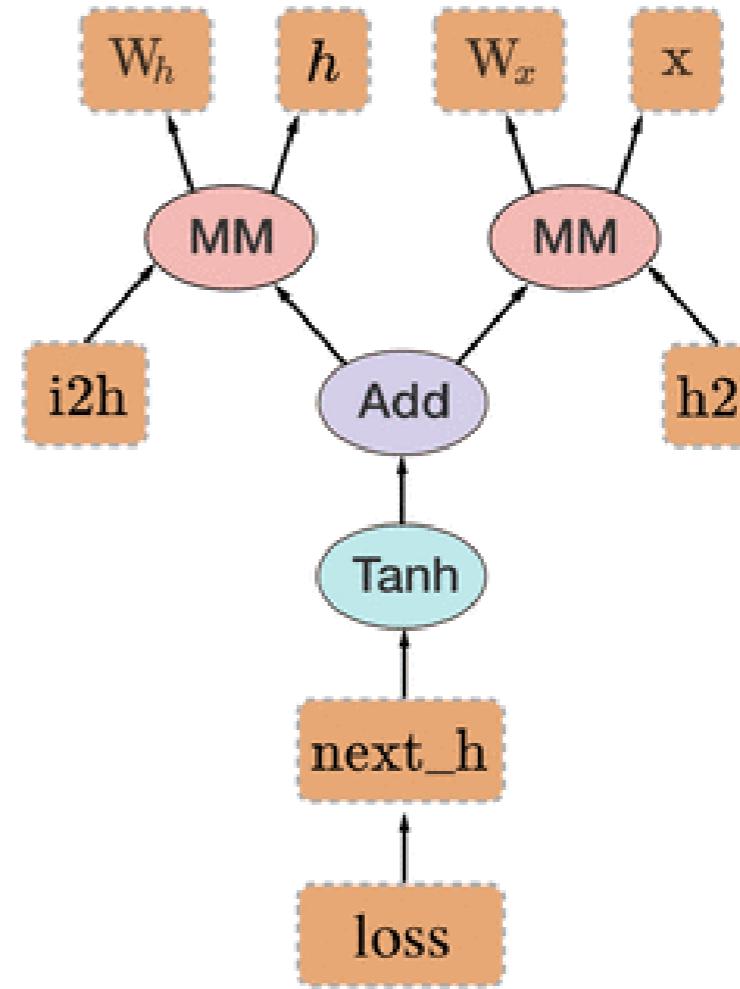
- Deep Learning Framework
  - Tensorflow, Keras, Torch, Chainer, MXNet
- Python-native, NumPy-friendly
- Dynamic graphs
- <https://pytorch.org/>
- <https://pytorch.org/docs/stable/index.html>

## 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!
```



# 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
  - <https://conda.io/>



# Our stack

- **Jupyter Notebook**
  - Document and visualize live code
  - <http://jupyter.org/>



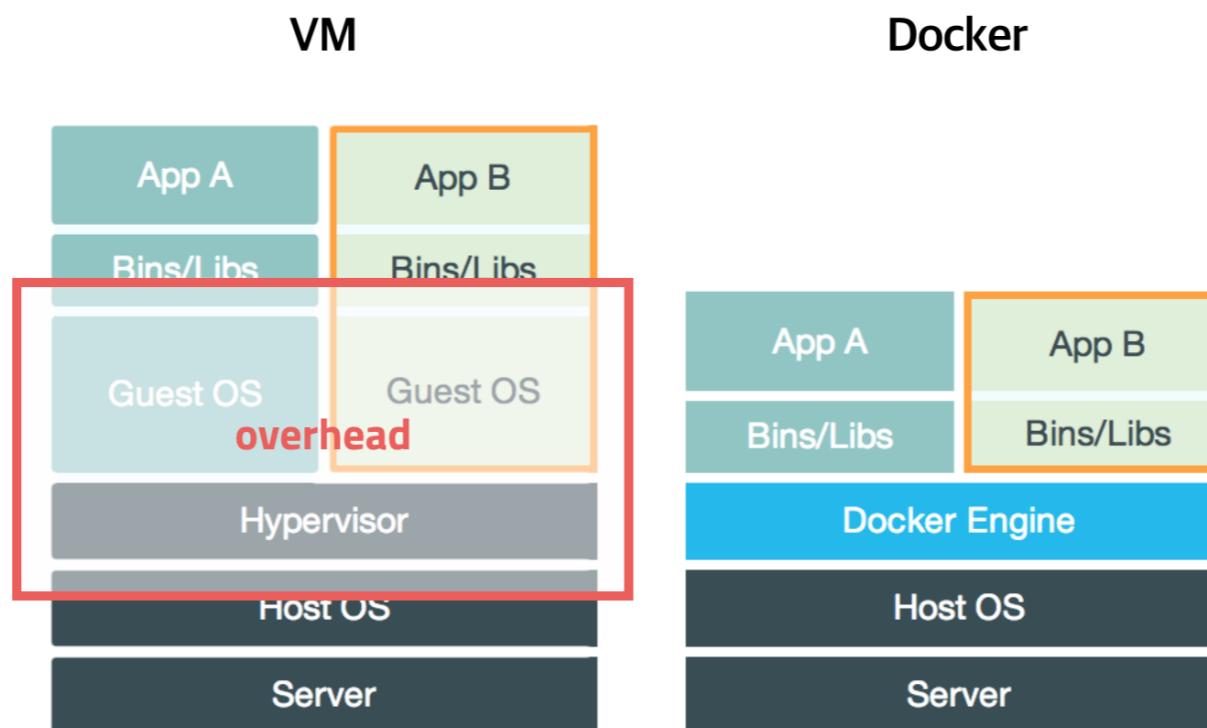
# Our stack

- **Docker CE**
  - Container-based virtualization
  - <https://www.docker.com/>
    - Install on Ubuntu
    - Install on macOS
    - Install on Windows 10



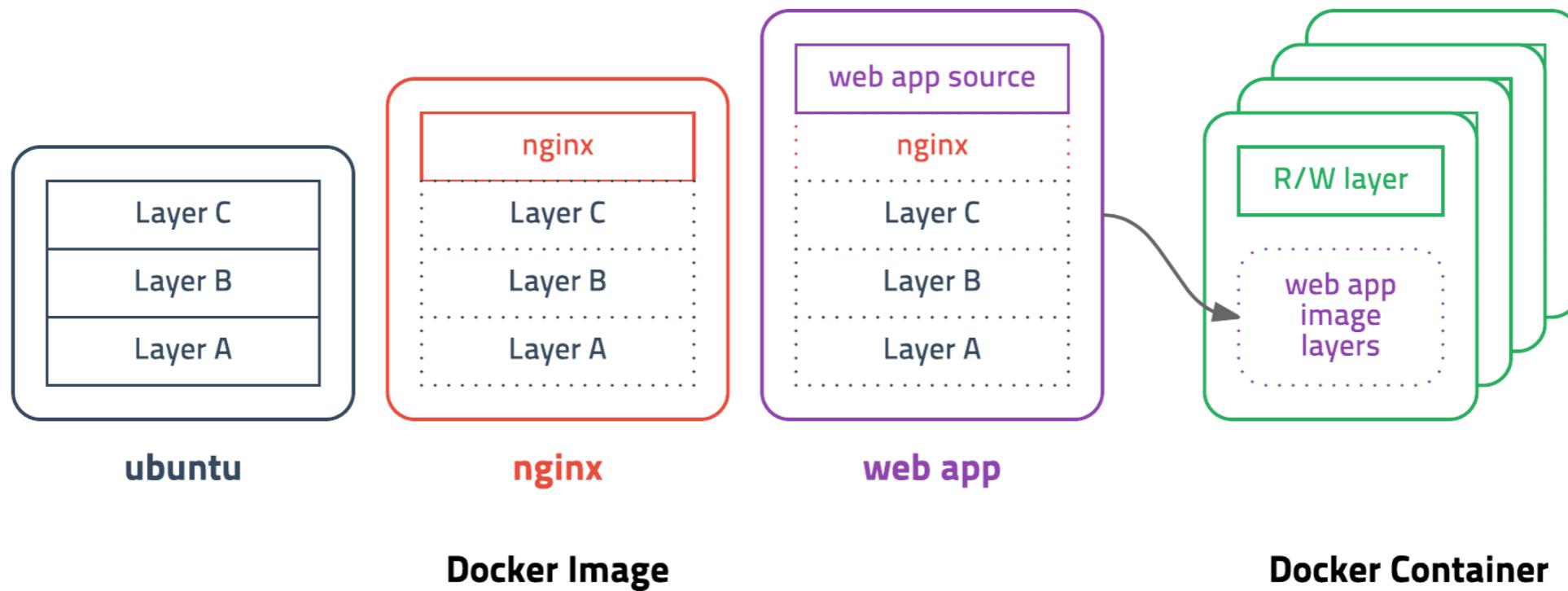
# Our stack

- Docker CE



# Our stack

- Docker CE



# Quick preparation

1. Install Anaconda.
  - <https://conda.io/> > Next > Installation > Regular installation > Choose your OS
2. Open Anaconda console, and create a new virtual environment.
  - `conda create -y --name pytorch-nlp python=3.6 numpy pyyaml scipy ipython mkl tqdm`
3. Install PyTorch on the new environment (this may take a while).
  - `conda install --name pytorch-nlp pytorch-cpu torchvision -c pytorch`

# Installation guides

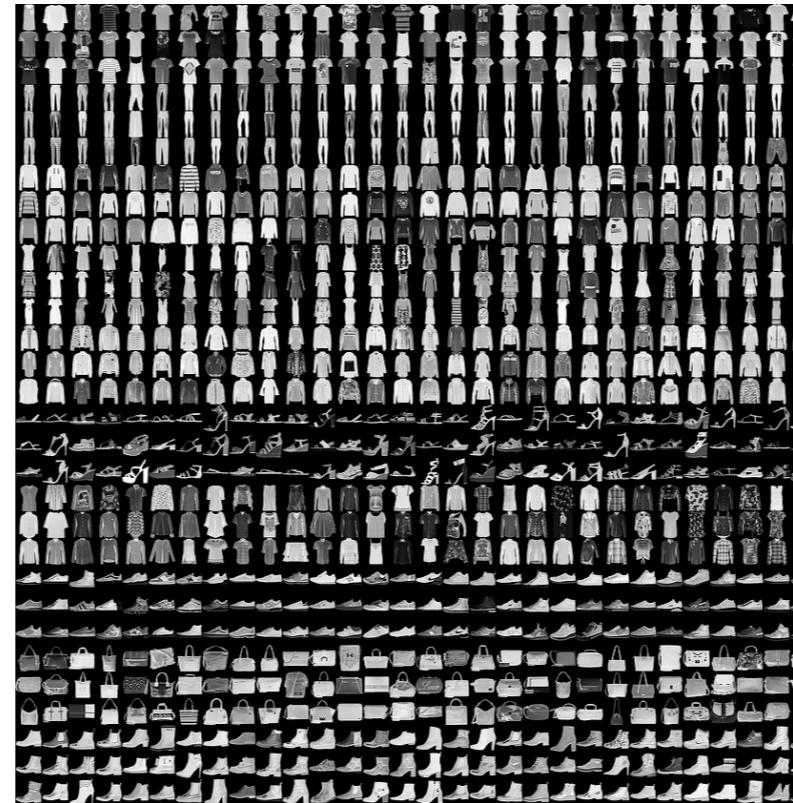
- PyTorch, PyCharm, Windows 10
- AWS에 PyTorch 작업환경 꾸리기
- Windows Subsystem for Linux에 PyTorch 설치하기

# Image Classification with PyTorch

PyTorch

# The data

- FashionMNIST



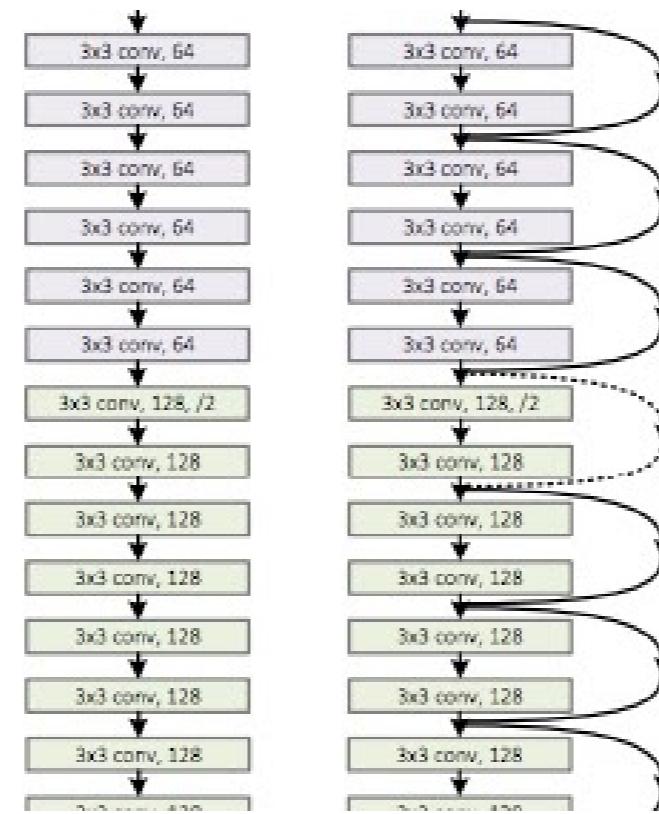
# The data

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

# The model

- ResNet

plain net



ResNet

# The source

git clone or download

[https://github.com/juneoh/sample\\_pytorch\\_project](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.

# PyTorch modules

- `torch.Tensor`
- `torch.nn.Module`
- `torchvision.models.resnet`

# Into the code!

[https://github.com/juneoh/sample\\_pytorch\\_project](https://github.com/juneoh/sample_pytorch_project)



**WHEN YOU FIND THE  
RIGHT HYPERPARAMETERS**

**FROM THE FIRST RUN**

# Additional tasks

- Try changing the optimizer to Adam and see how it works.
- Try using the [step learning rate scheduler](#).
- Try training on the GPU.

# Complementary materials

- 라온피플 머신러닝 아카데미
- Oxford CS Deep NLP 2017
- Pattern Recognition & Machine Learning 요약정리
- Calculaus on Computational Graphs: Backpropagation
- 초보를 위한 도커 안내서 - 도커란 무엇인가?
- nvidia-docker

# Thank you!