Natural Language Processing with PyTorch

Week 1 딥러닝을 위한 PyTorch 실무환경 구축

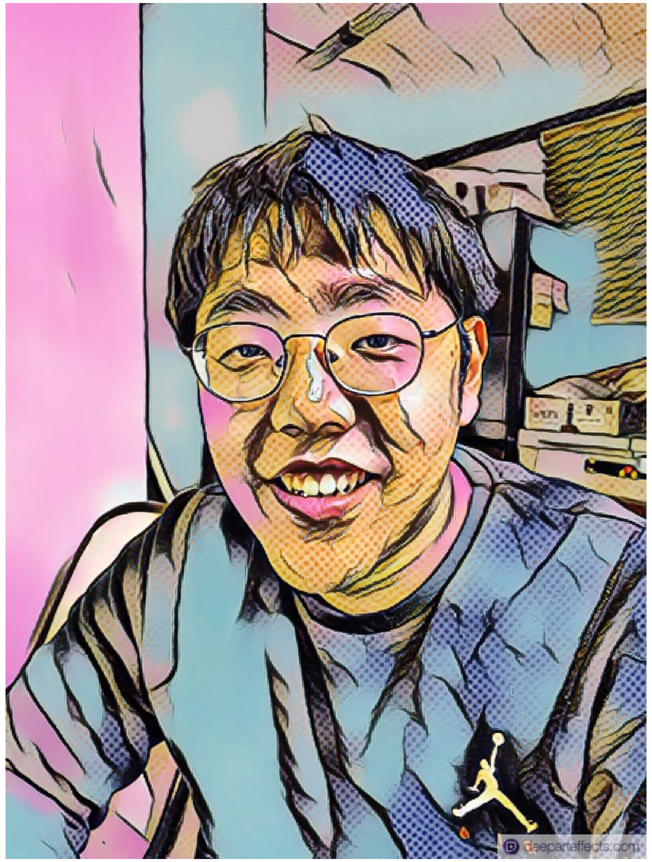
Fast campus

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 thenew environment (this may take a while).
   - `conda install --name pytorch-nlp pytorch-cpu torchvision -c pytorch`

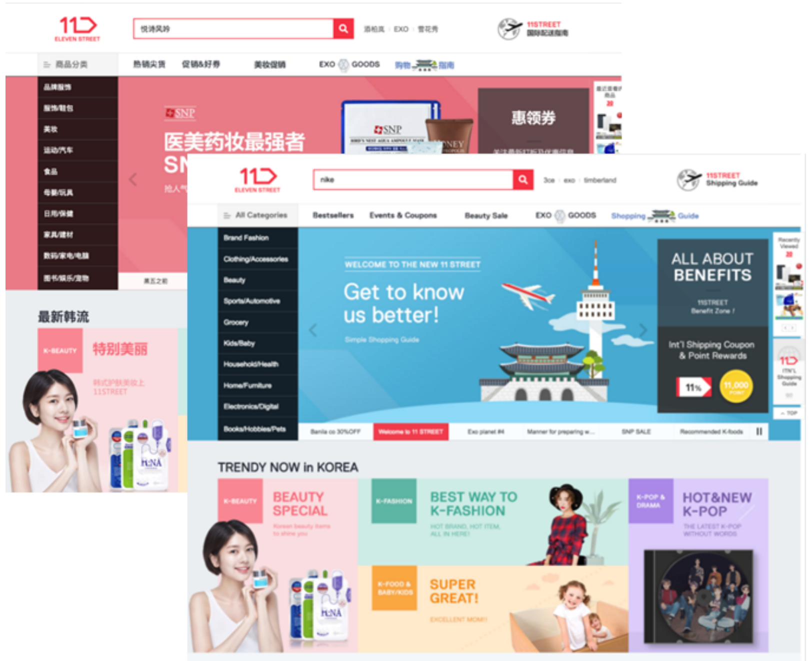
Ki Hyun Kim

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



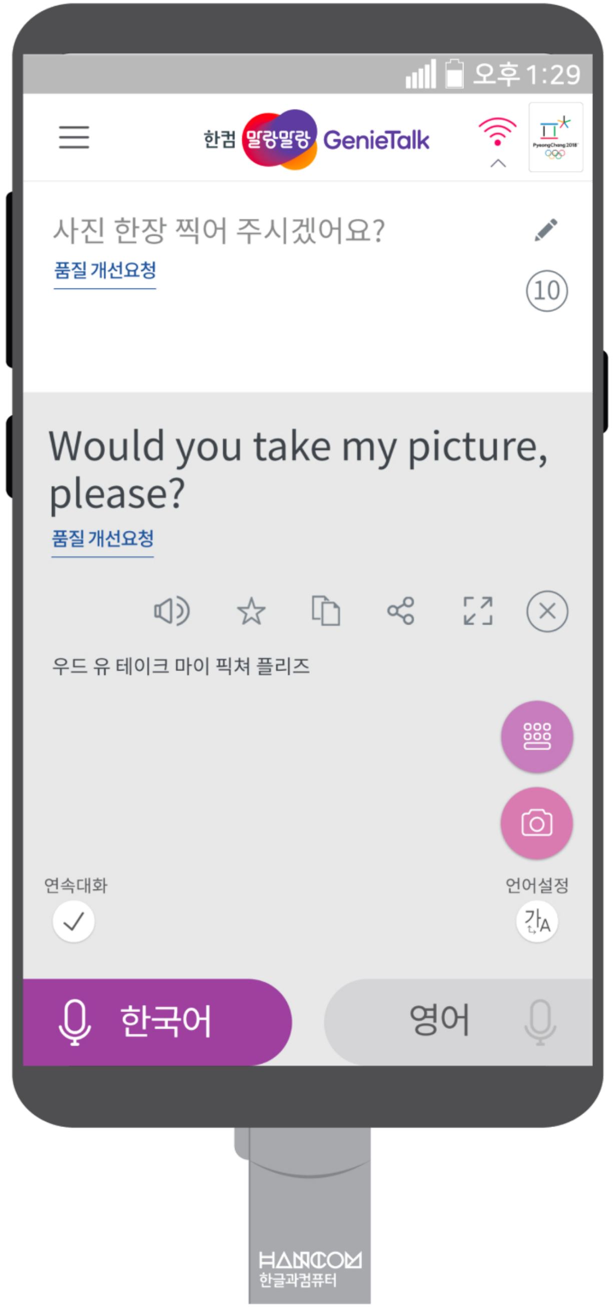
Ki Hyun Kim

- 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

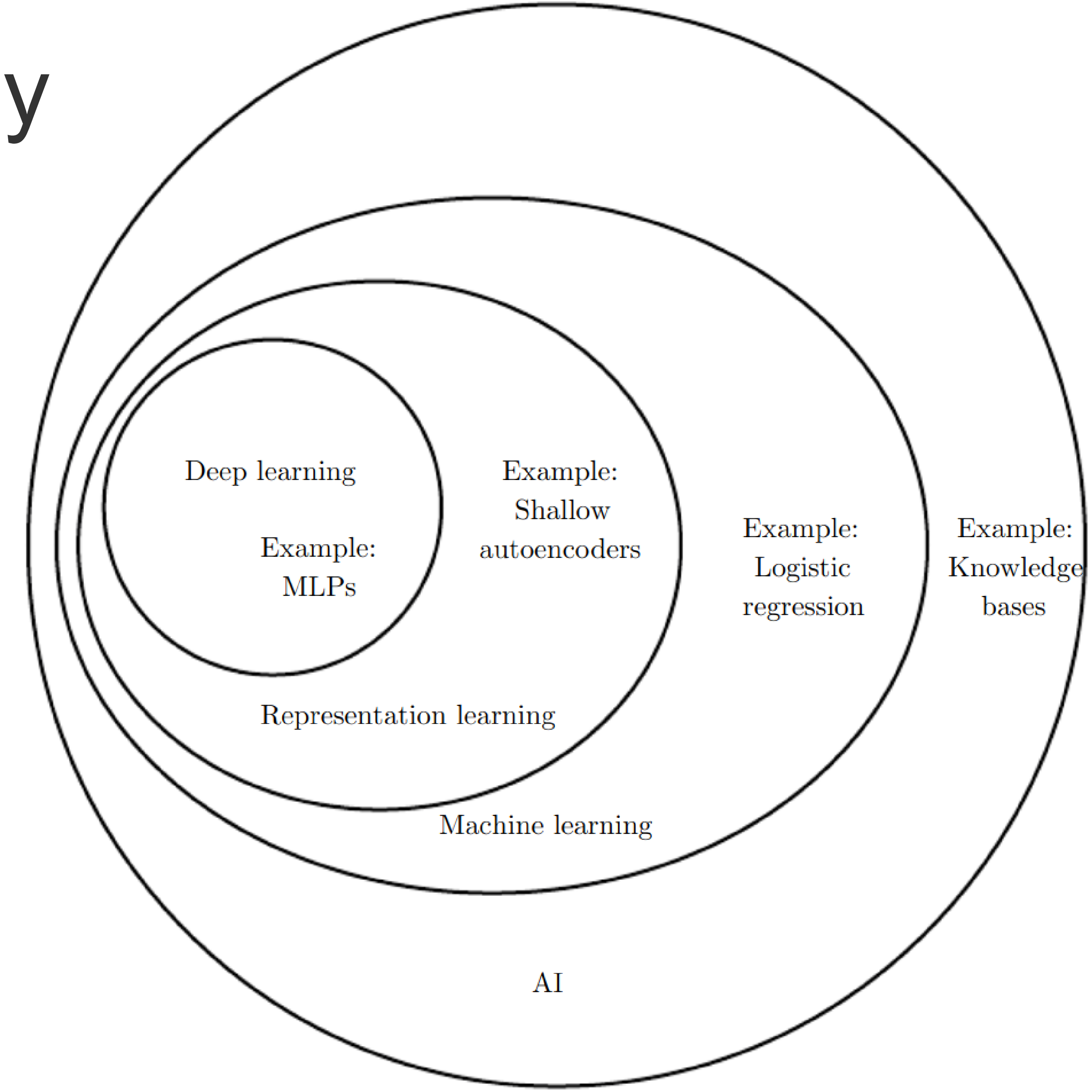
Course mechanics

- 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

01

Genealogy

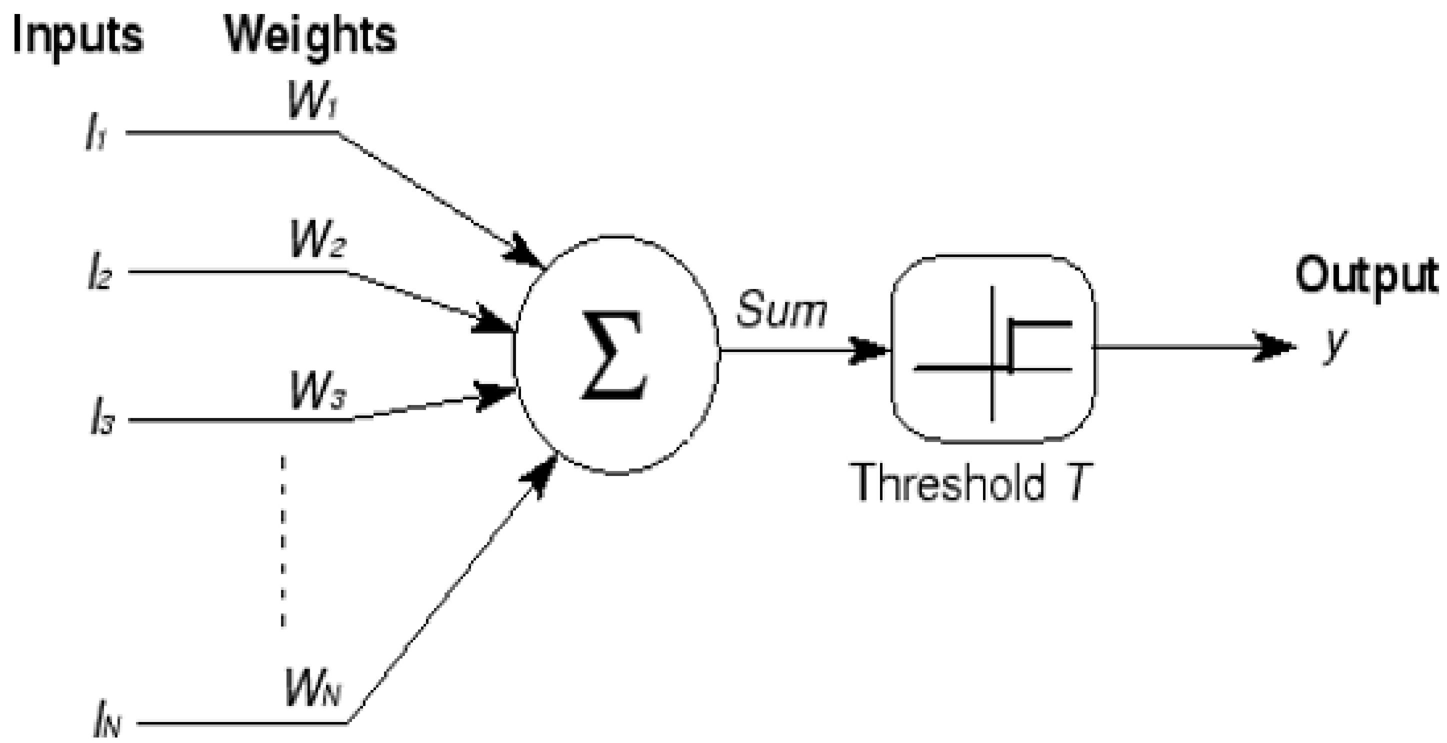

Deep learning
Example:
MLPs
Example:
Shallow
autoencoders
Representation learning
Example:
Logistic
regression
Machine learning
Example:
Knowledge
bases
AI

Genealogy

Timeline

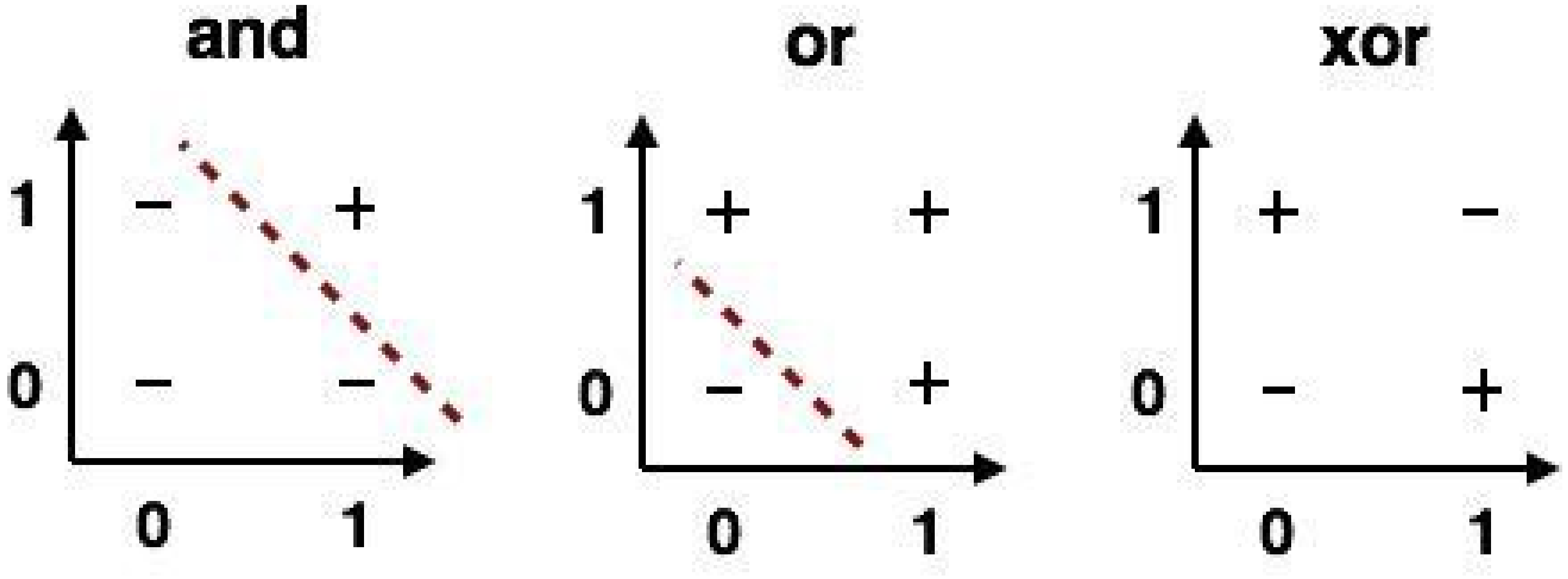
- **Cybernetics** 1940s-1960s
  - McCulloch-Pitts neuron
    - McCulloch and Pitts, 1942. A Logical Calculus of the Ideas Immanent in Nervous Activity.
  - Hebbian learning
    - Hebb, 1949. The Organization of Behaviour.
  - Perceptron
    - Rosenblatt, 1958. The Perceptron: A Probabilistic Model for 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!
초고속 인공지능
SPEED 012

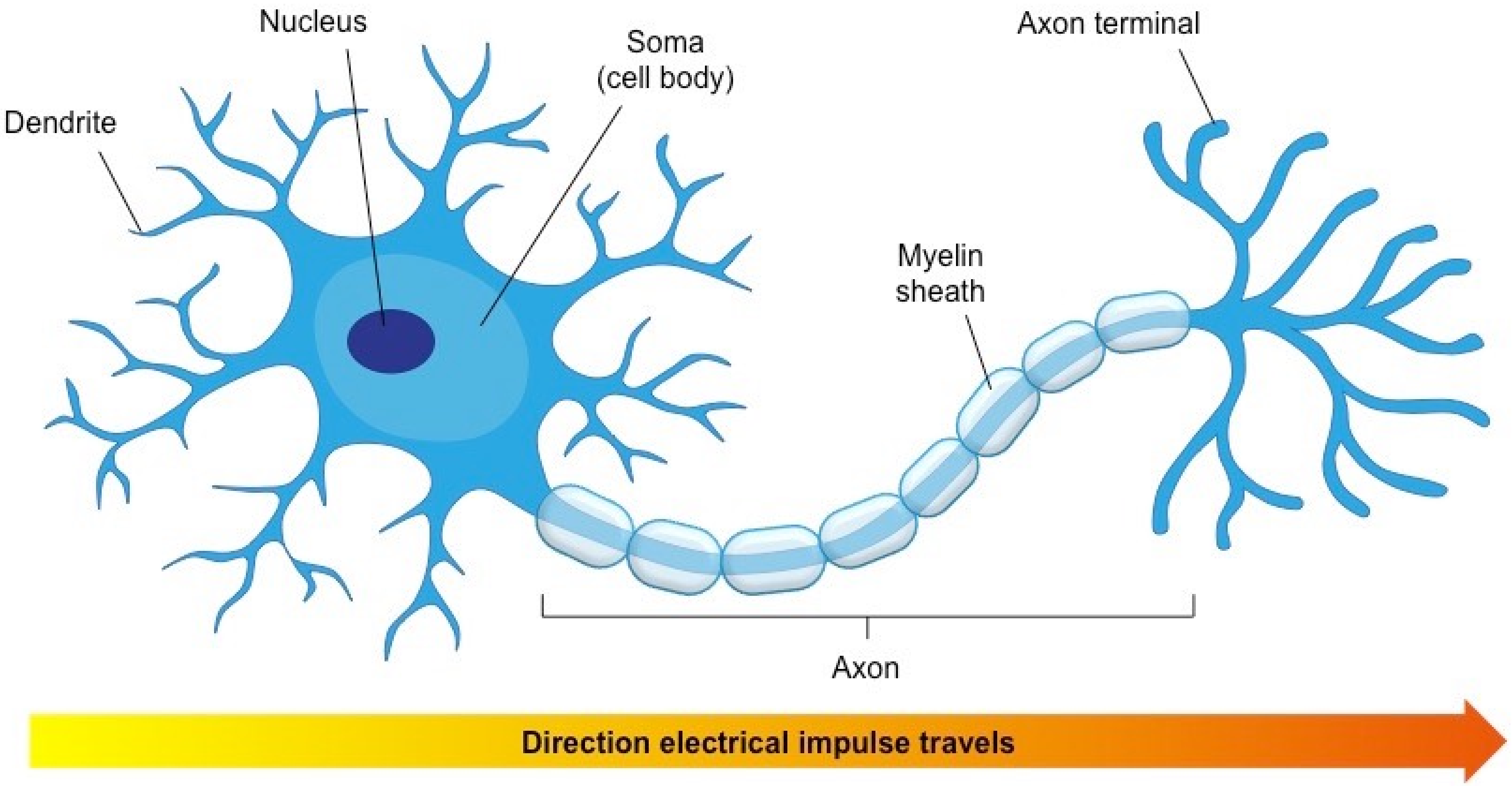
Fast campus

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 Classifification with Deep Convolutional Neural Networks.

Neural Networks

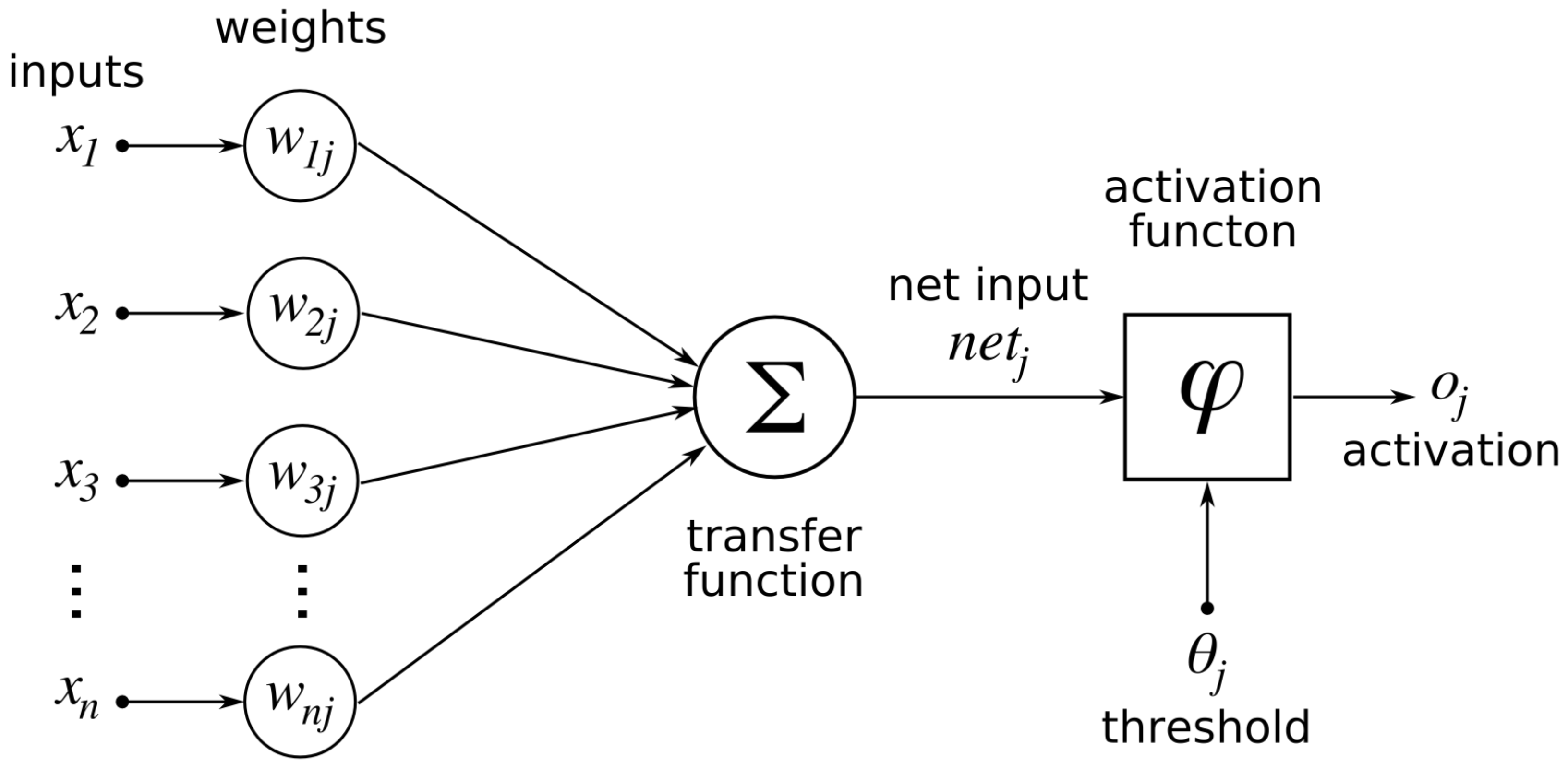
- Feed-forward Network



A biological neuron.

Neural Networks

- Feed-forward Network



A perceptron.

Neural Networks

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

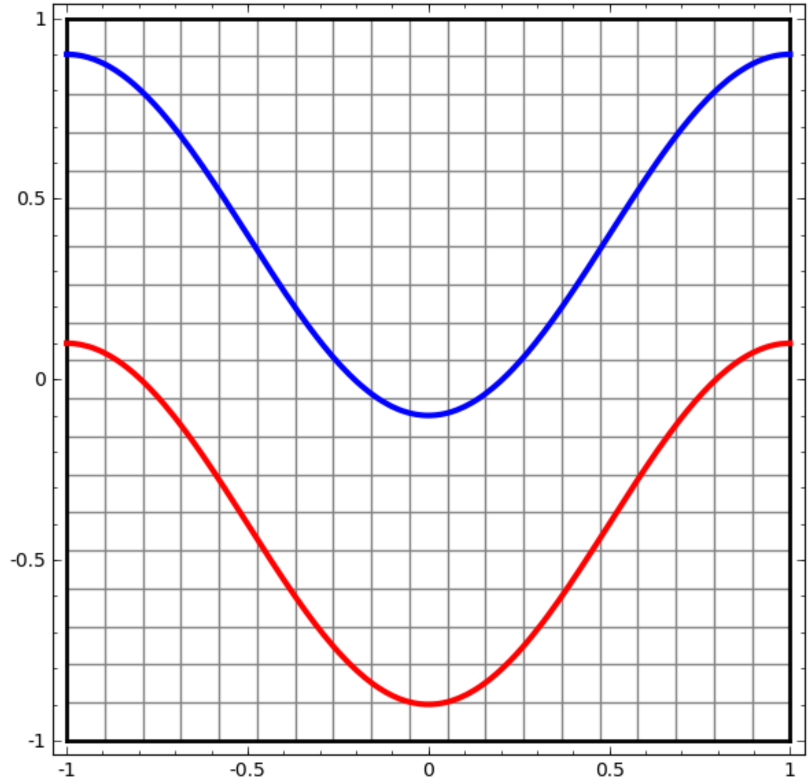
```
def perceptron(inputs, weights, biases):
    return sum(inputs * weights + biases)
```

  - In PyTorch:

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

Neural Networks

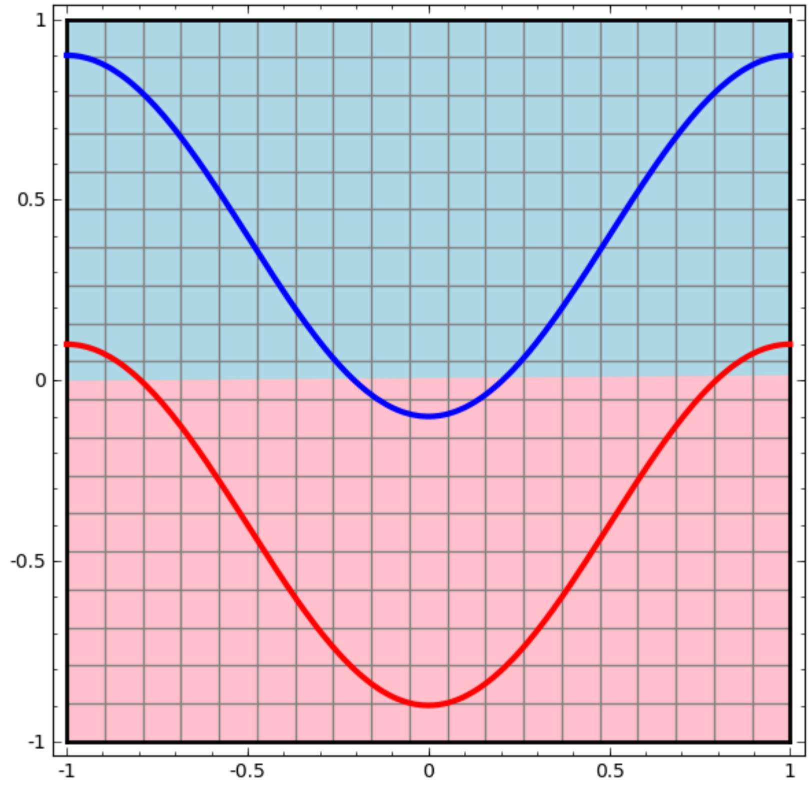
Problem: draw a single straight line to separate colors.



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

Neural Networks

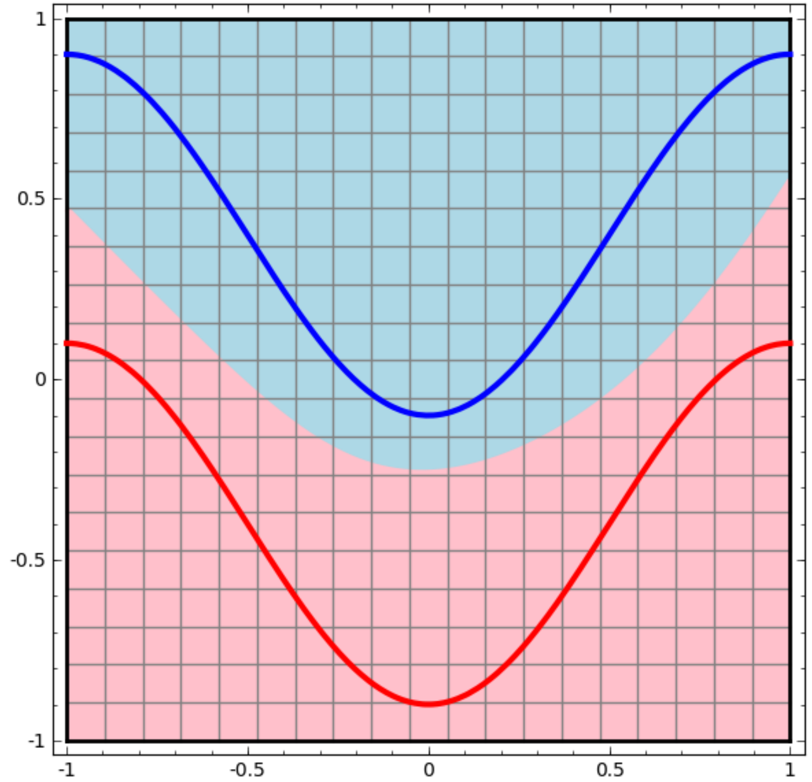
Problem: draw a single straight line to separate colors.



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

Neural Networks

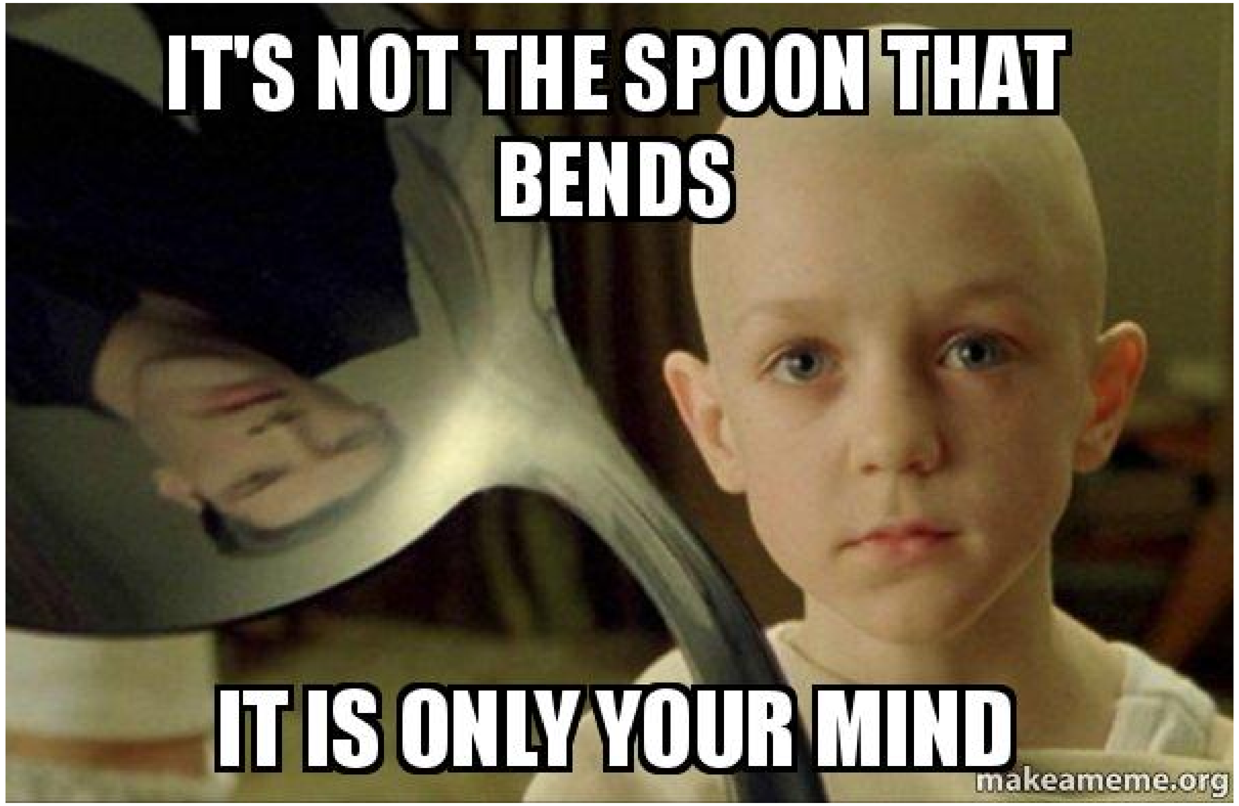
Problem: draw a single straight line to separate colors.



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

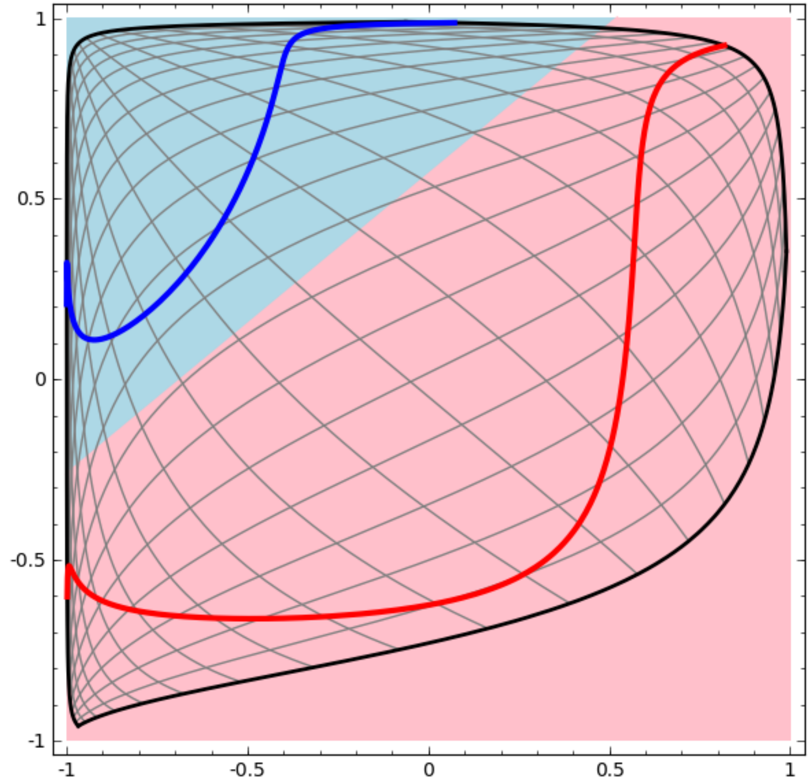
Neural Networks

Problem: draw a single straight line to separate colors.

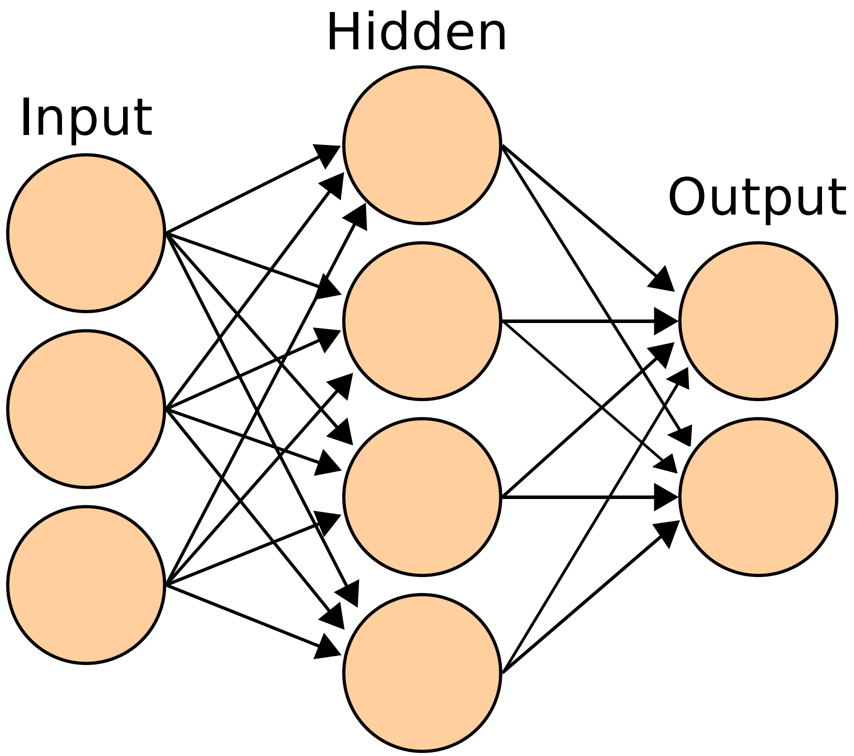


Neural Networks

Problem: draw a single straight line to separate colors.



Neural Networks

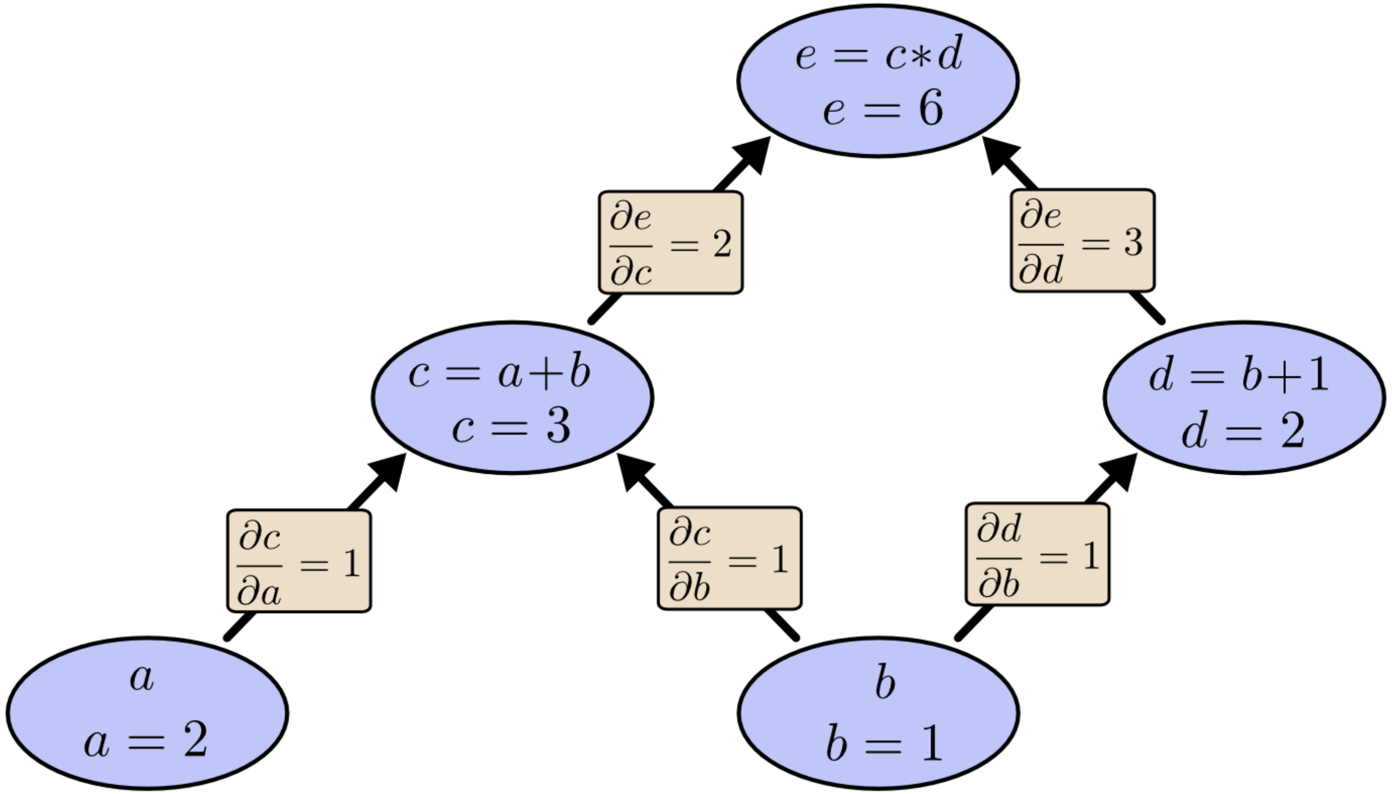


The hidden layer learns a representation,
so that the data is linearly separable.

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

Neural Networks

- Backpropagation



Neural Networks

- Backpropagation
  - In Python (with NumPy):

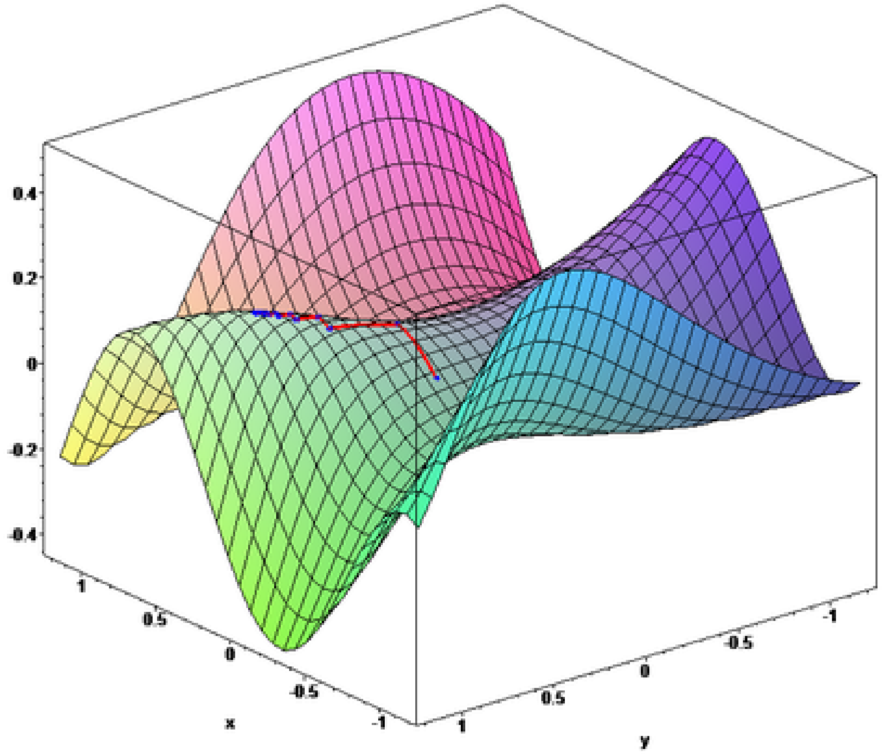
```
def backpropagate(weights, derivative, learning_rate):
    return weights - learning_rate * (derivative - weights)
```

  - In PyTorch:

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

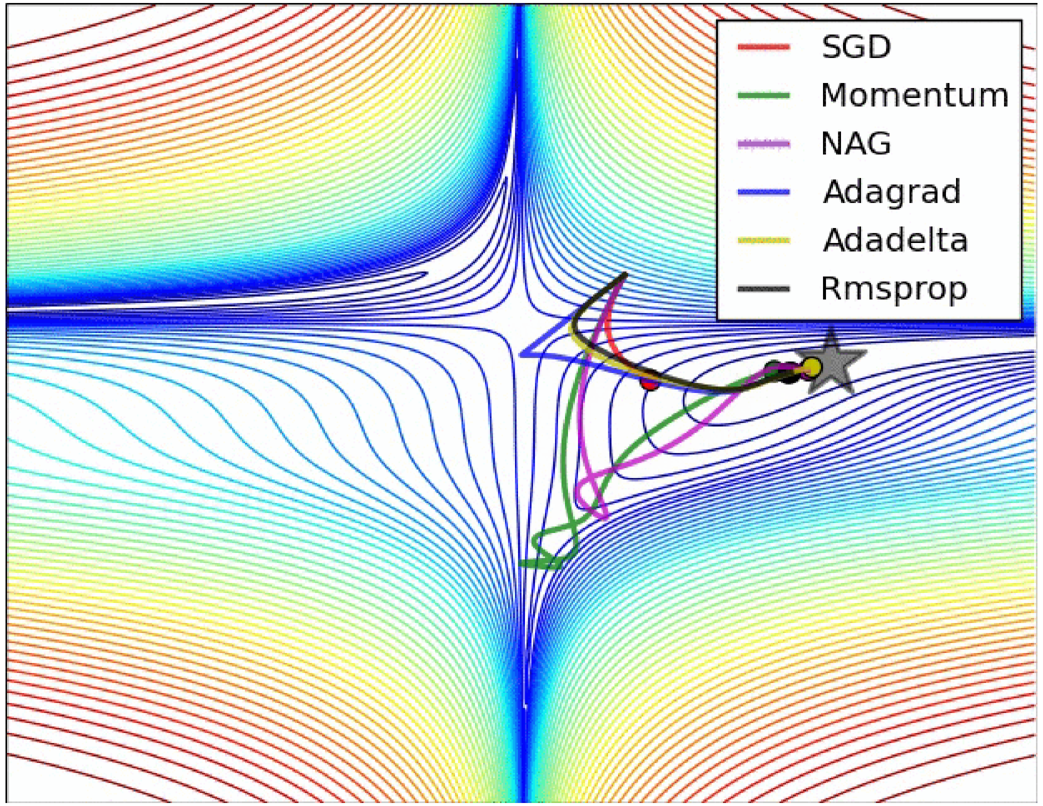
Neural Networks

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



Neural Networks

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

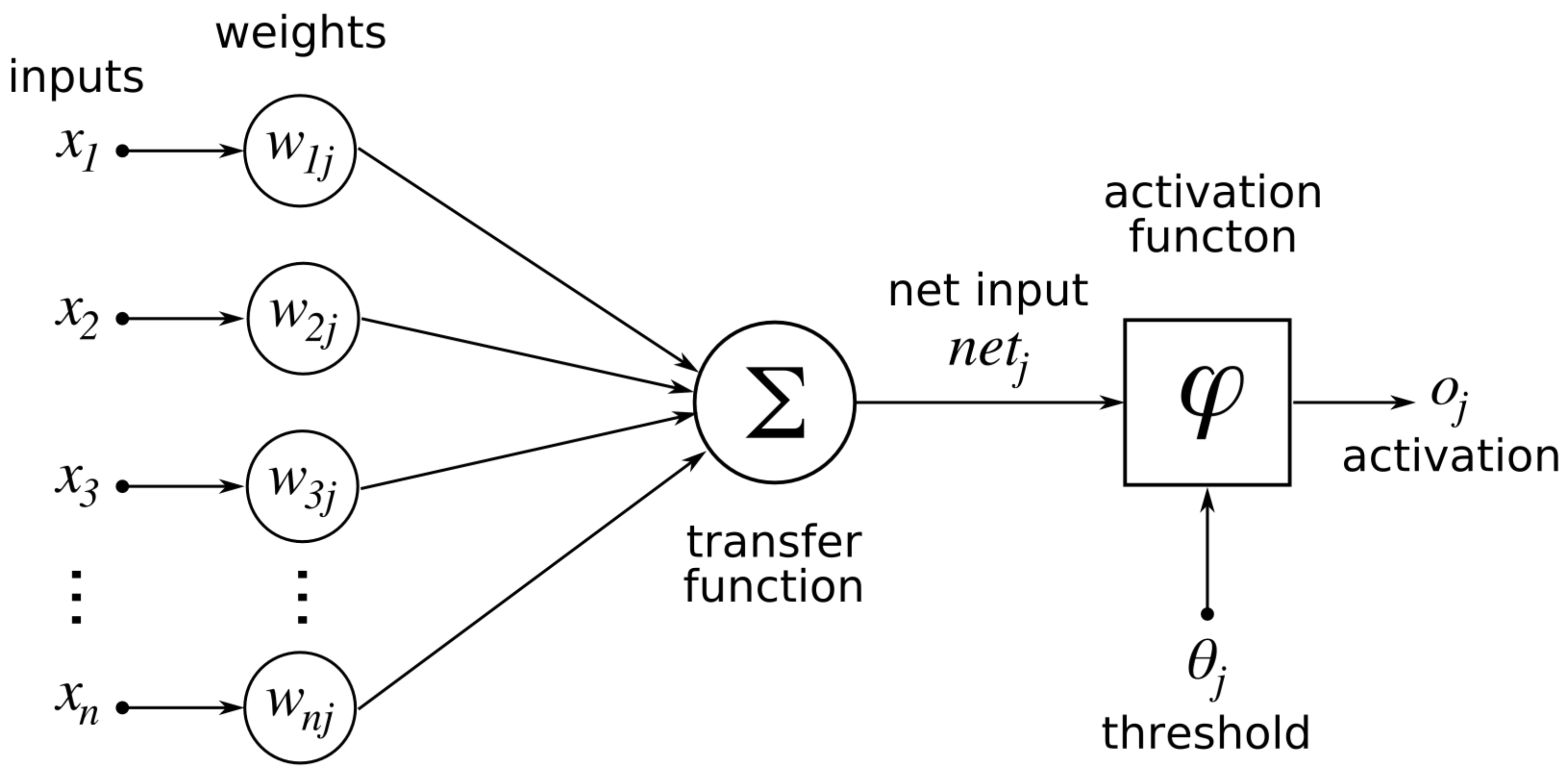


Neural Networks

- 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()
```

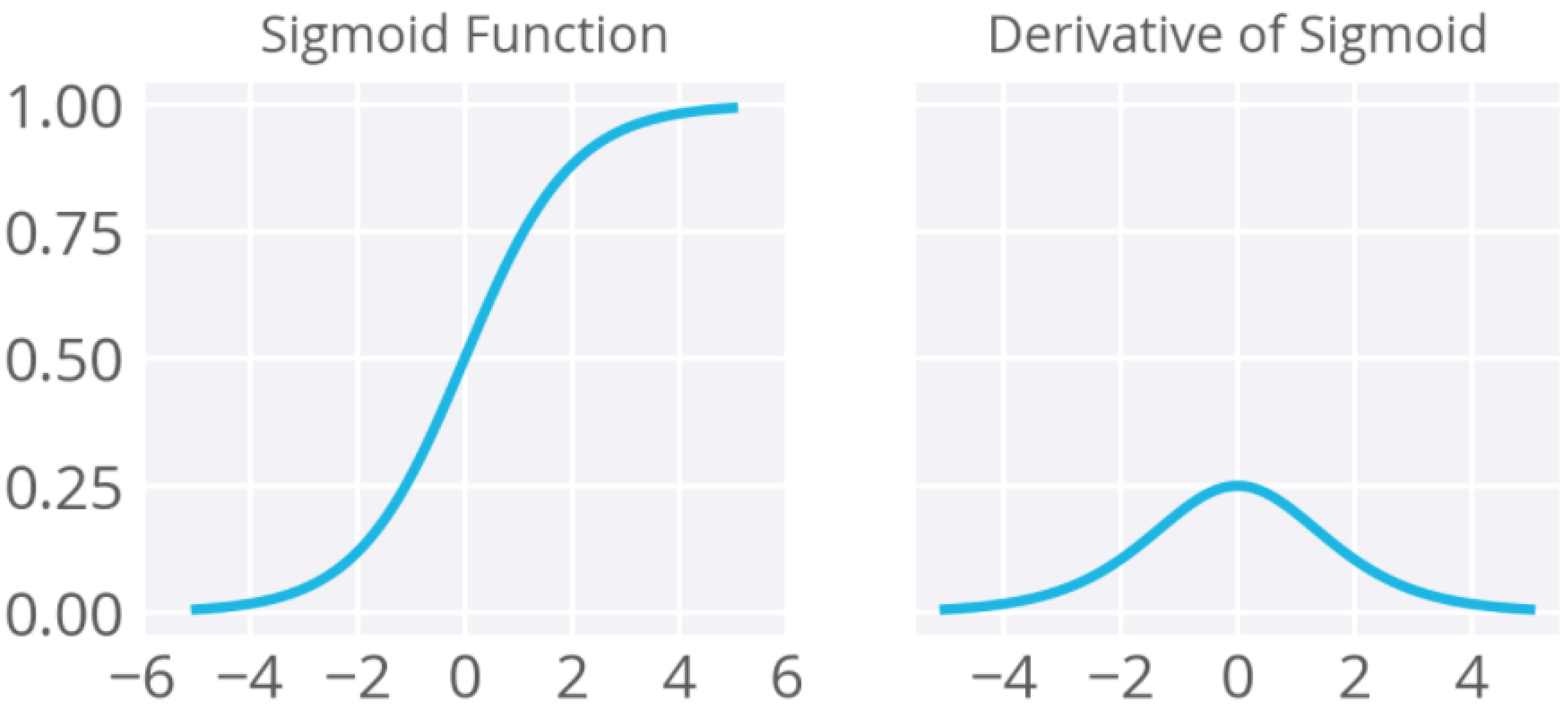
Activation functions and non-linearity



A perceptron.

Activation functions and non-linearity

- 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

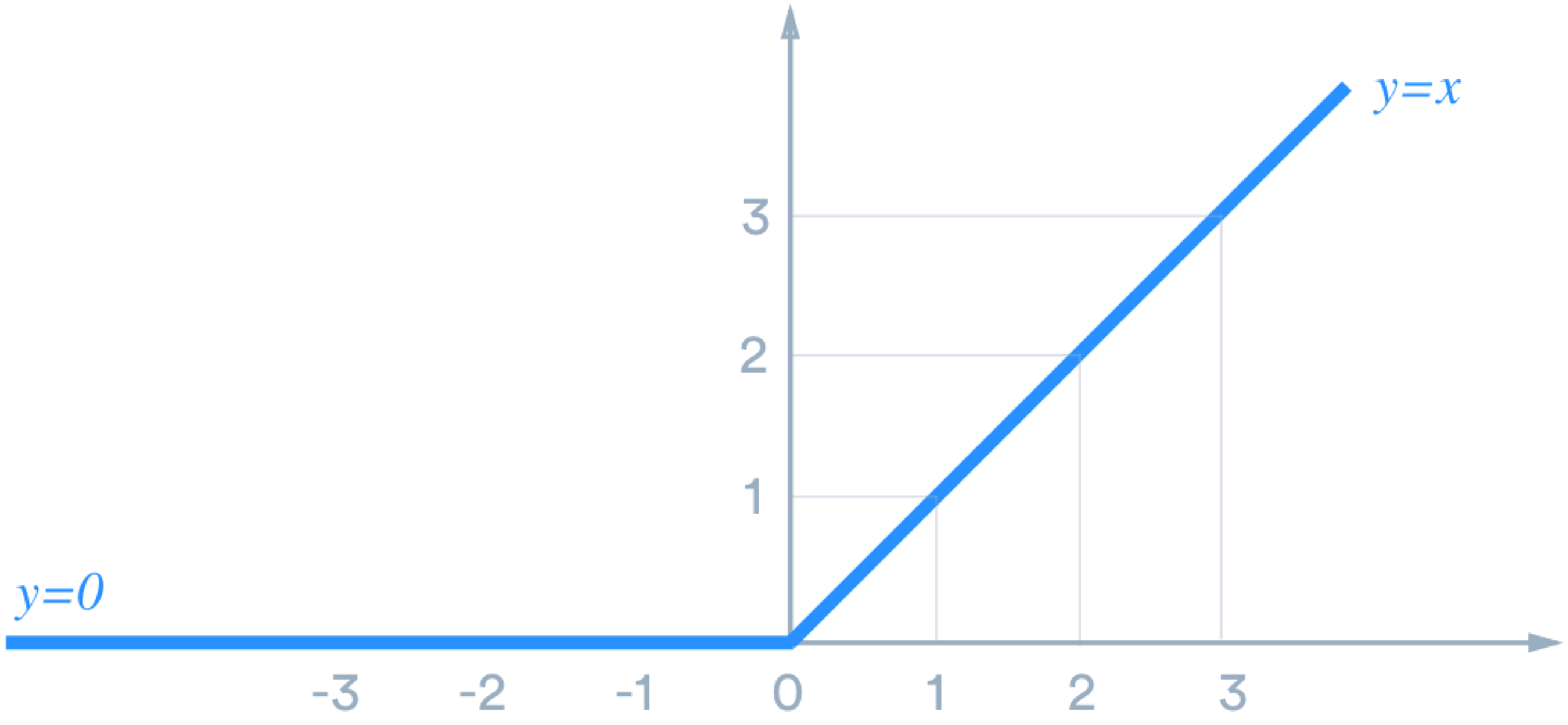
- Sigmoid

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

  - 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)

$$max(0, x)$$

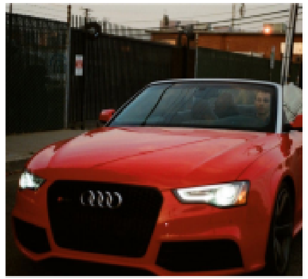
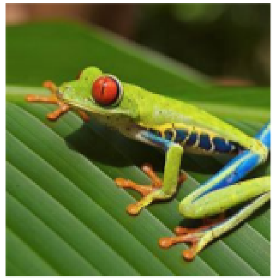
  - In Python (with NumPy):

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

  - In PyTorch: `torch.nn.ReLU`

Activation functions and non-linearity

- Softmax

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

Activation functions and non-linearity

- Softmax

$$\begin{bmatrix} 1.2 \\ 0.9 \\ 0.4 \end{bmatrix} \longrightarrow \boxed{\text{Softmax}} \longrightarrow \begin{bmatrix} 0.46 \\ 0.34 \\ 0.20 \end{bmatrix}$$

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

Activation functions and non-linearity

- Softmax

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

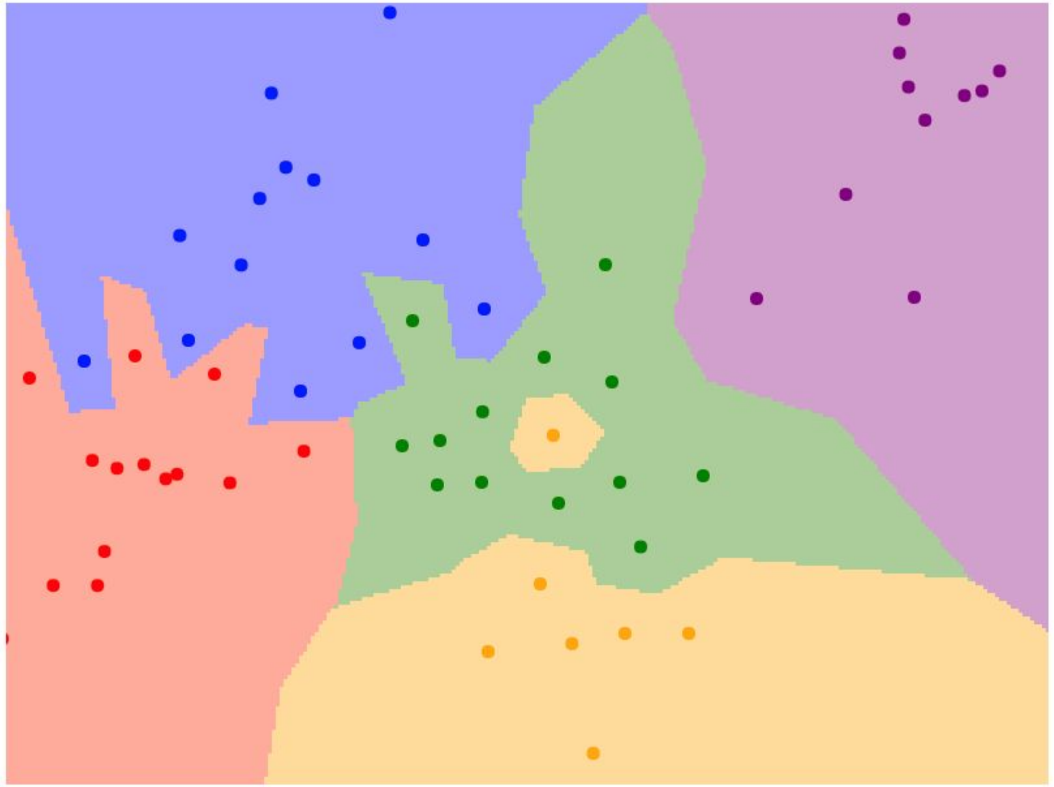
  - In Python (with NumPy):

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

  - In PyTorch: `torch.nn.Softmax`

Loss functions

- L1 loss and L2 loss
  - k-Nearest Neighbors



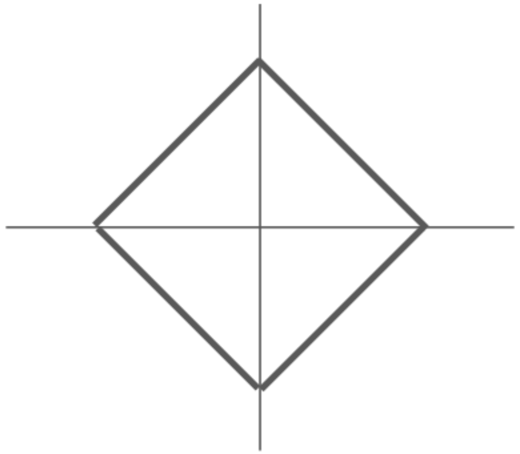
http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture02.pdf

Loss functions

- L1 loss and L2 loss

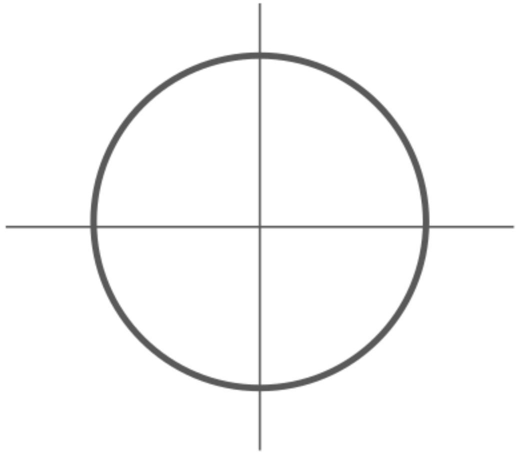
L1 (Manhattan) distance

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



L2 (Euclidean) distance

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



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

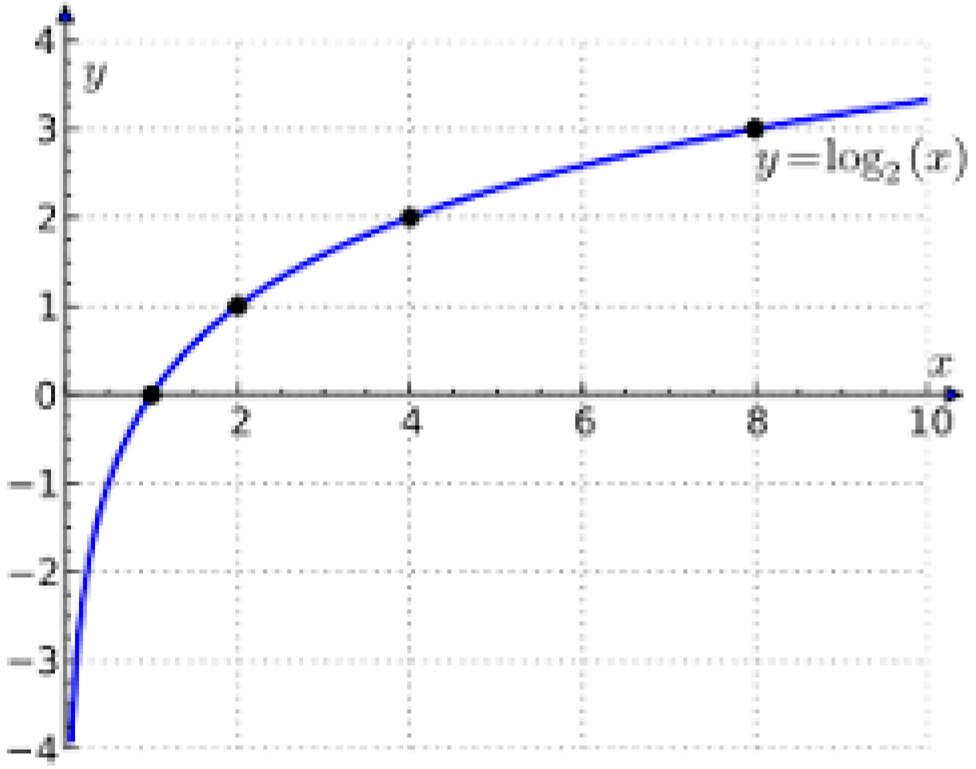
Entropy(in information theory)

= 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

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.1 \\ 0.5 \\ 0.4 \end{bmatrix} \qquad D(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_j y_j \ln \hat{y}_j \qquad \mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

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:

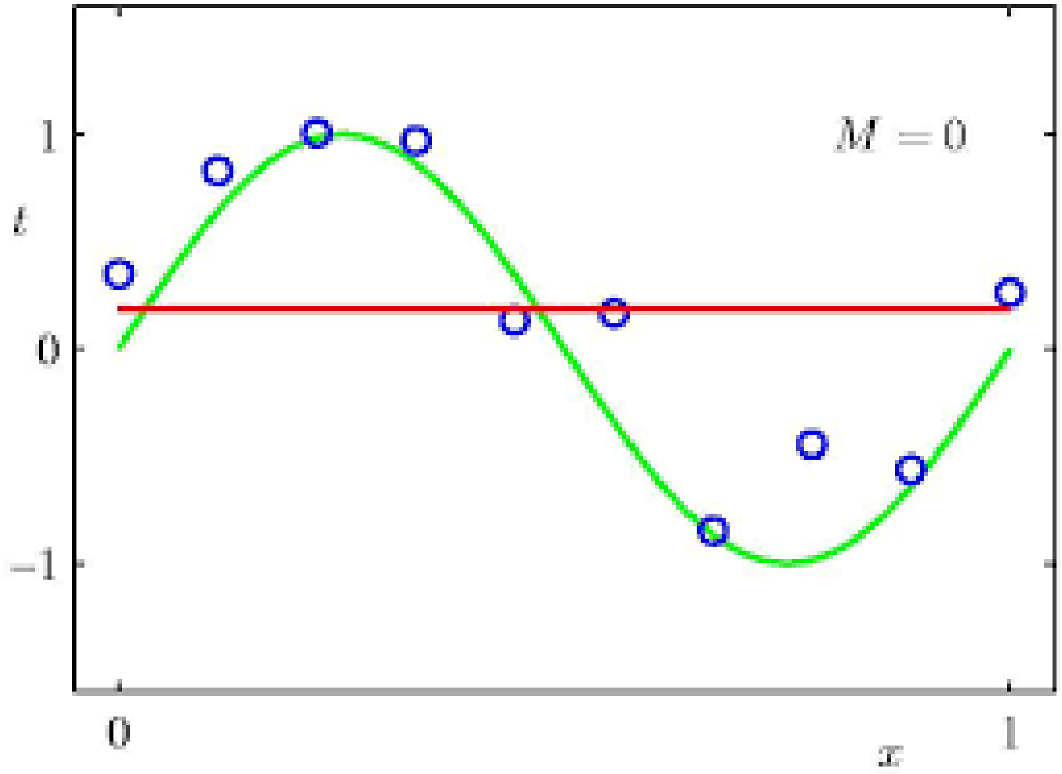
```
def binary_cross_entropy_loss(targets, outputs):
    return -mean(targets * log(outputs) + (1 - targets) * log(1 - outputs))
```

  - In PyTorch: `torch.nn.BCELoss`

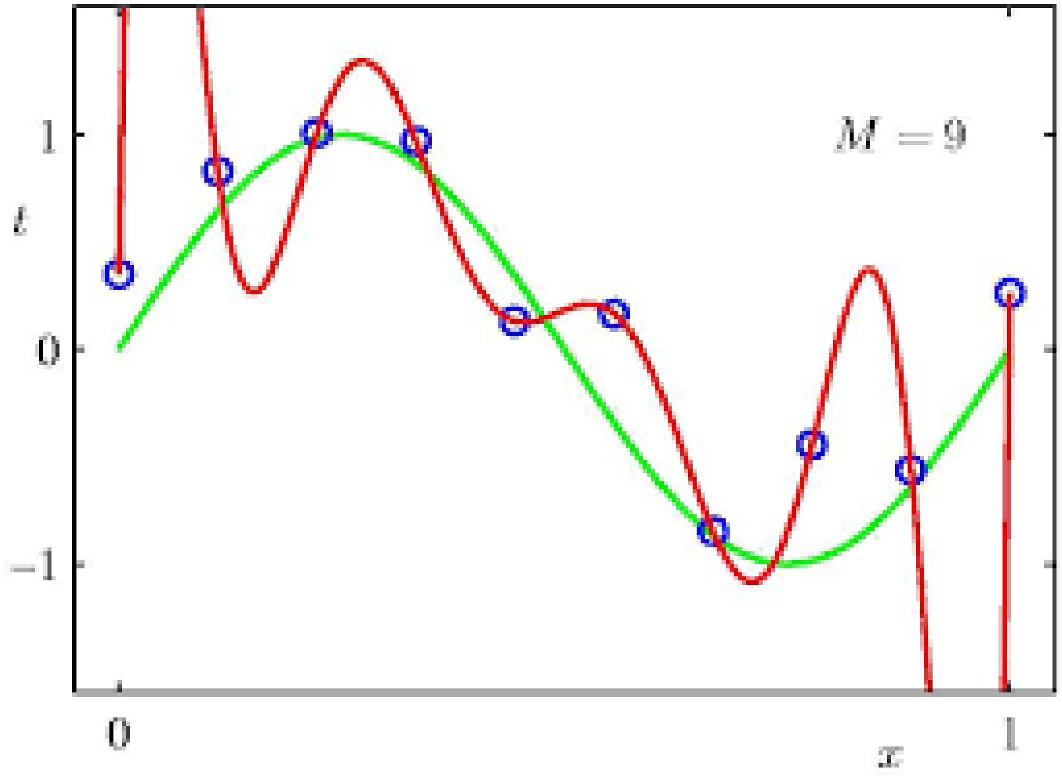
Loss functions

- 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

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

  - 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

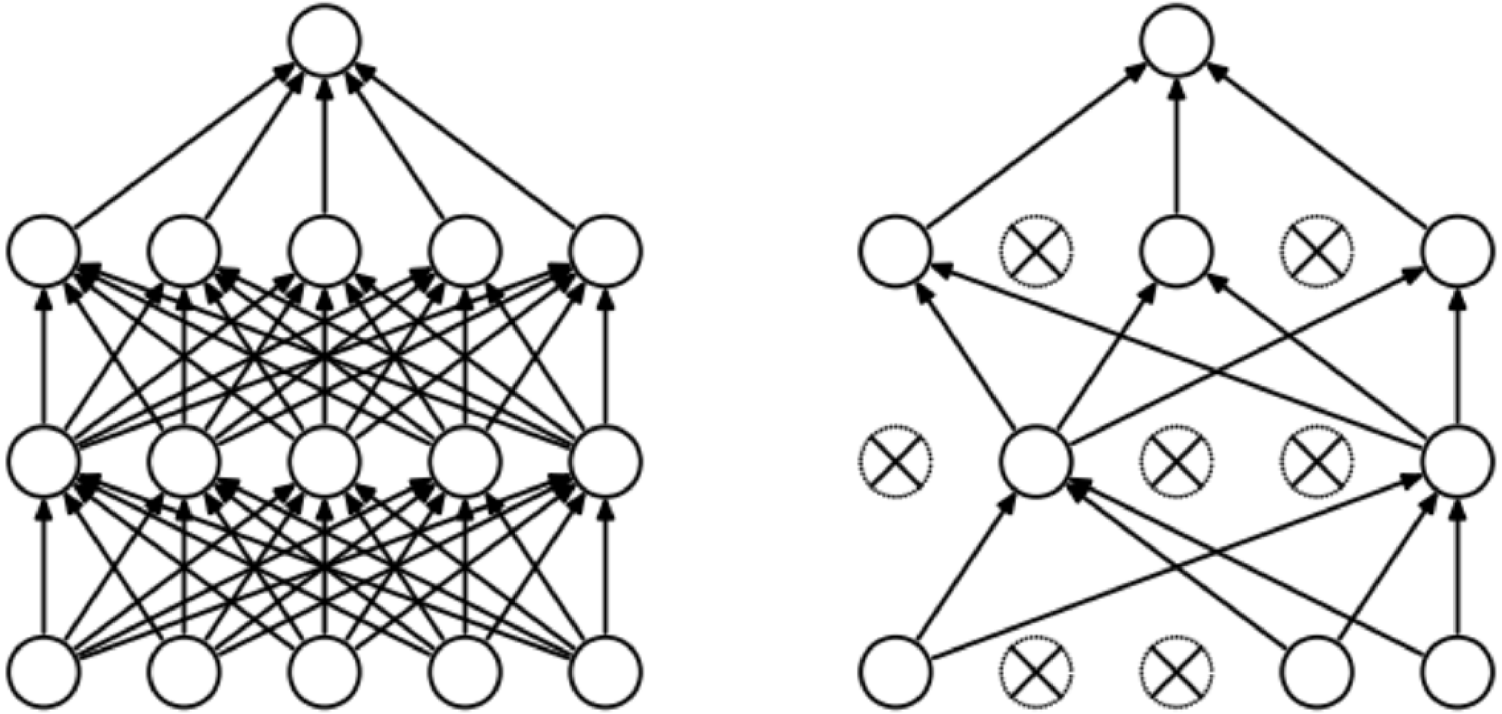
$$W \leftarrow W - \lambda(\frac{\partial L}{\partial W} + \gamma \|W\|)$$

  - In PyTorch:

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

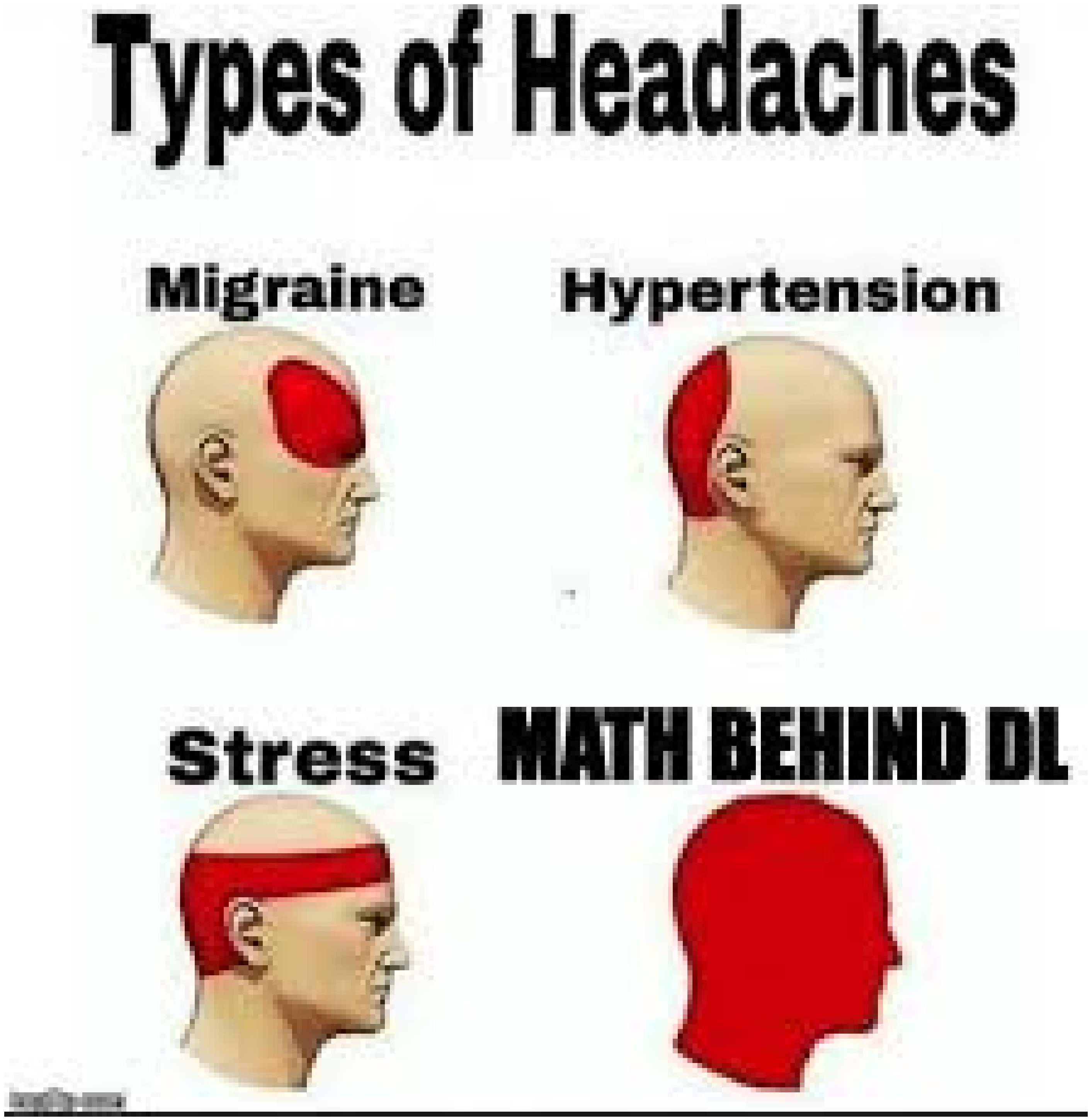
Regularization methods

- Dropout



(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

02

PyTorch

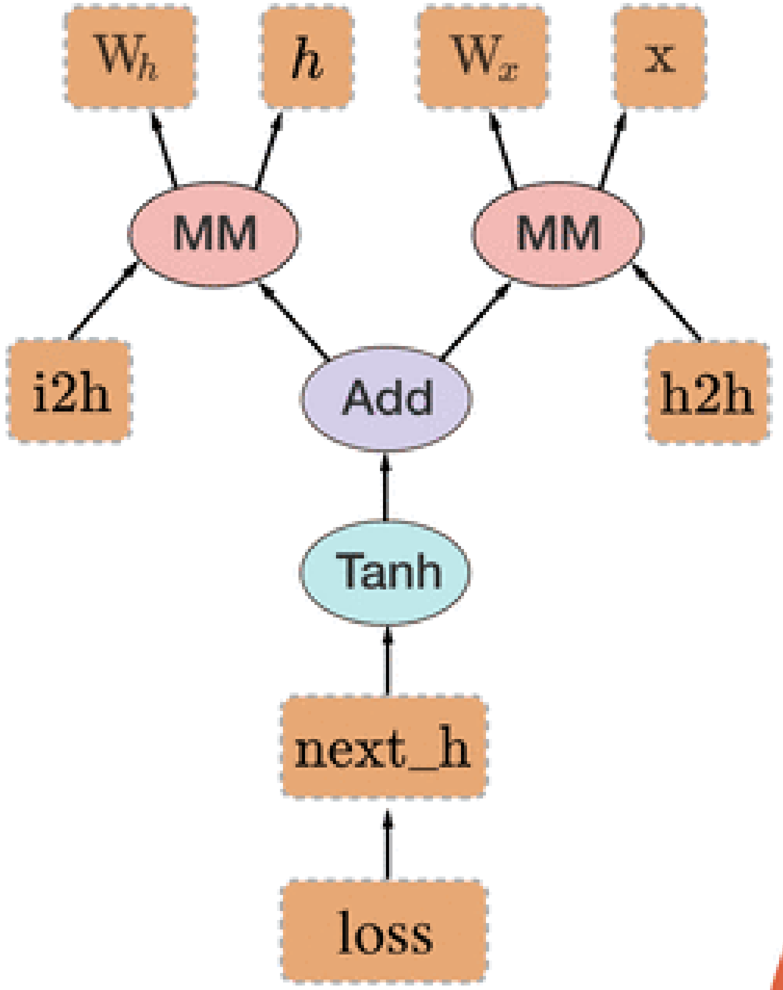
- Deep Learning Framework
  - Tensorflow, Keras, Torch, Chainer, MXNet
- Python-native, NumPy-friendly
- Dynamic graphs
- https://pytorch.org/
- https://pytorch.org/docs/stable/index.html

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/

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 thenew 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

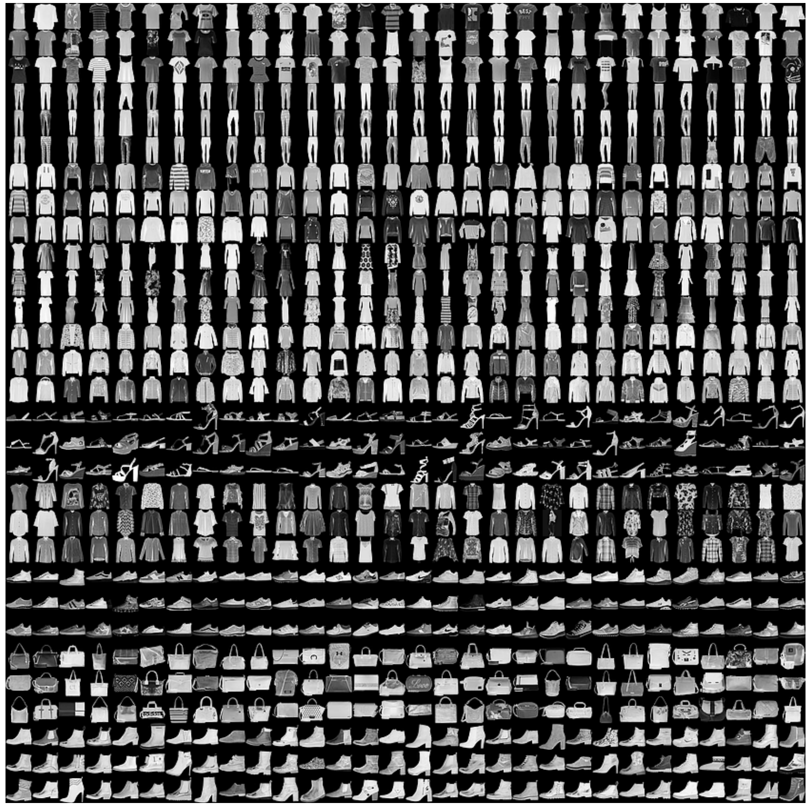
Image Classification with PyTorch

03

Fast campus

The data

- FashionMNIST

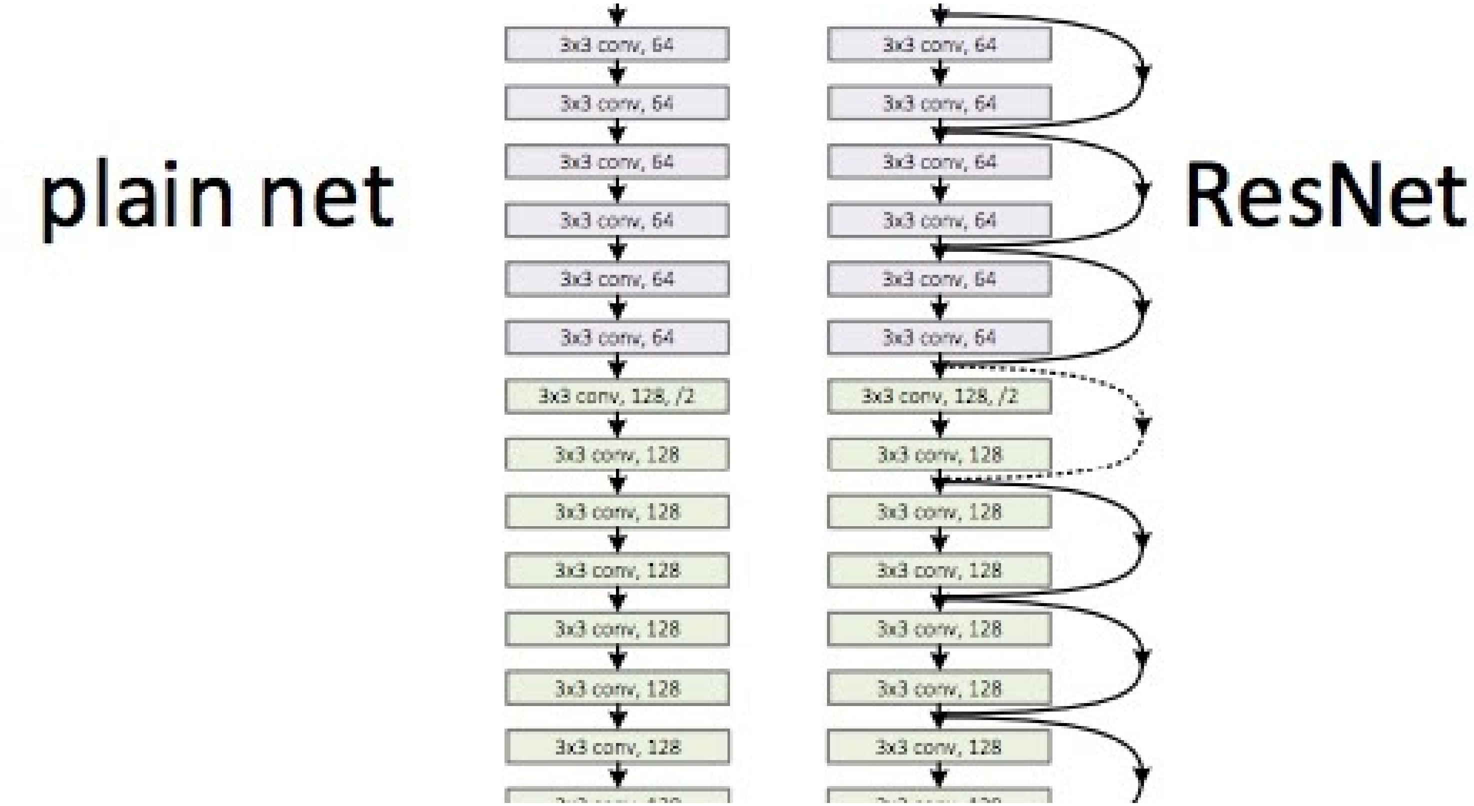


The data

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

The model

- ResNet



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.

PyTorch modules

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

Into the code!

https://github.com/juneoh/sample_pytorch_project

Fast campus

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.

Thank you!