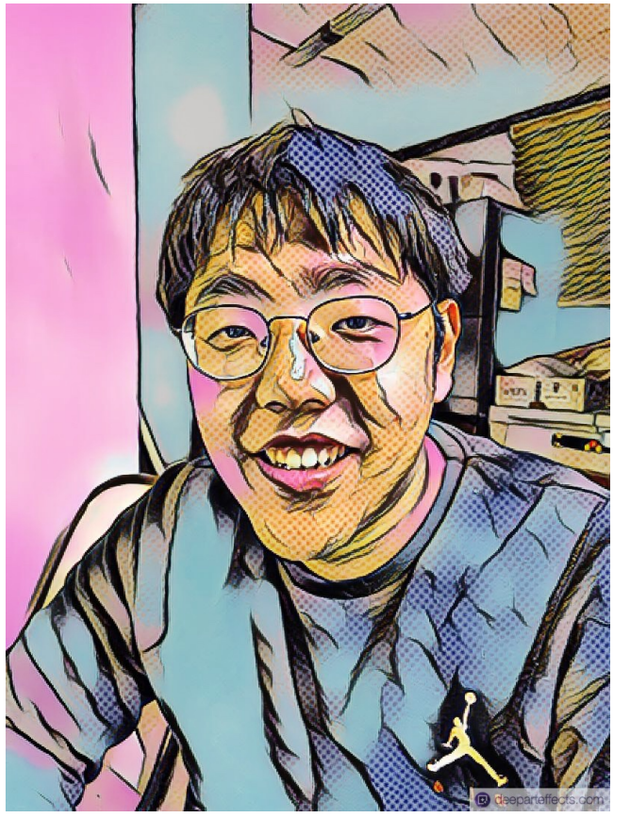
# Neural Language Processing with PyTorch

**Week 1** 딥러닝을 위한 PyTorch 실무환경 구축

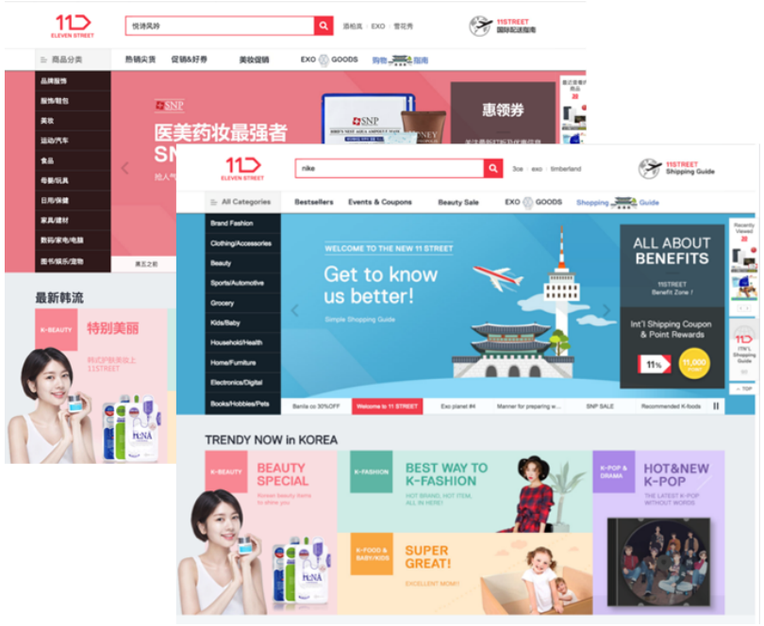
# Ki Hyun Kim


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

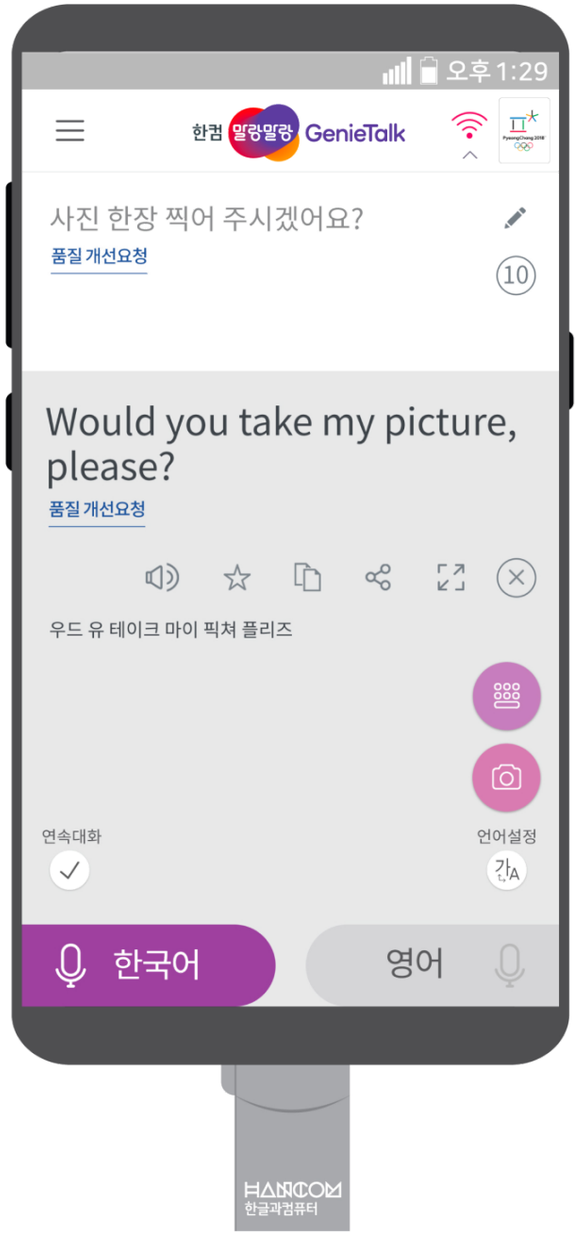
# Ki Hyun Kim

- 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

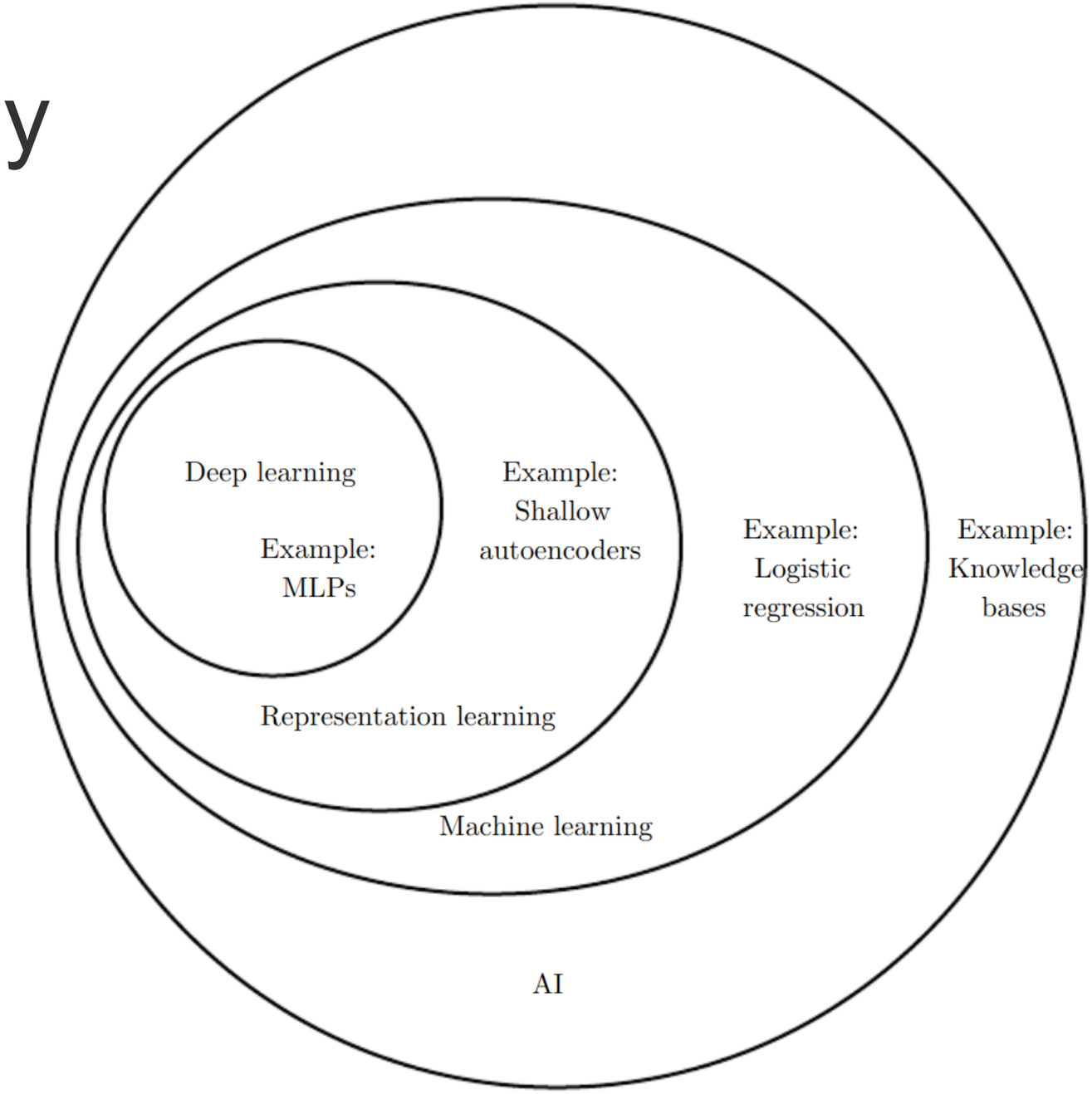
- Deep Learning Engineer @ Deep Bio
  - 병리영상 기반 전립선암 진단모델 연구개발
  - GPU 서버 분산 스케쥴링 시스템 개발
- Co-founder, Research Engineer @ QuantumSurf
  - 선물거래 알고리즘을 위한 API 설계 및 UX 개발
  - IPTV 영상품질 예측모델 연구개발
- BS in English Literature, minor in Philosophy @ 한국외국어대학교

# 오상준


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

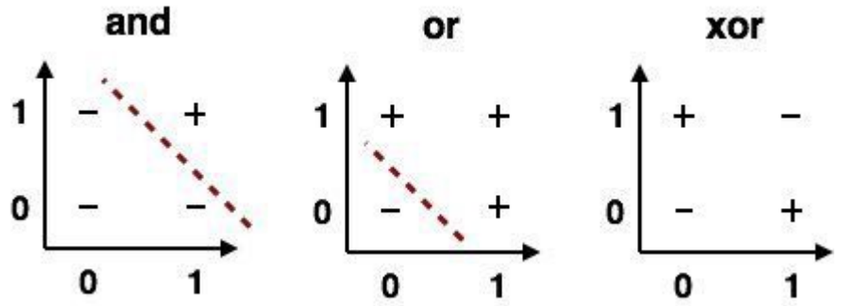
# 1. Introduction to Deep Learning

# Genealogy


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

# Timeline

- **Cybernetics** 1940s-1960s
  - McCulloh-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.


and
or
xor
1
0
0
1
−
+


BRACE YOURSELVES
AI WINTER IS COMING
memegenerator.net

# Timeline

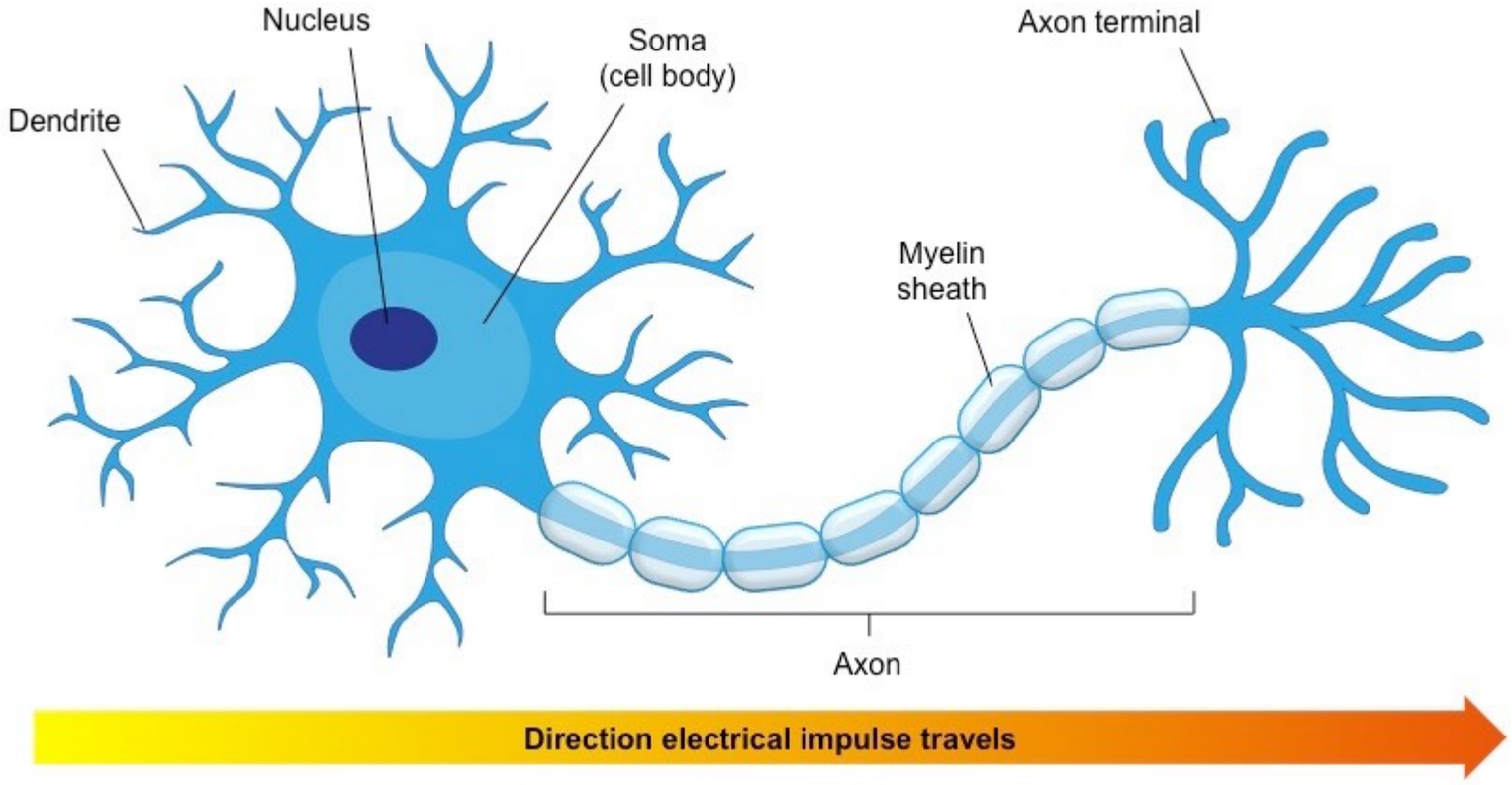
- **Connectionism** 1980s-1990s
  - Backpropagation
    - Rumelhart et al, 1986. Learning Representations by Back-propagating Errors.
  - Convolutional Neural Networks
    - Fukushima, 1980. Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift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

# 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.

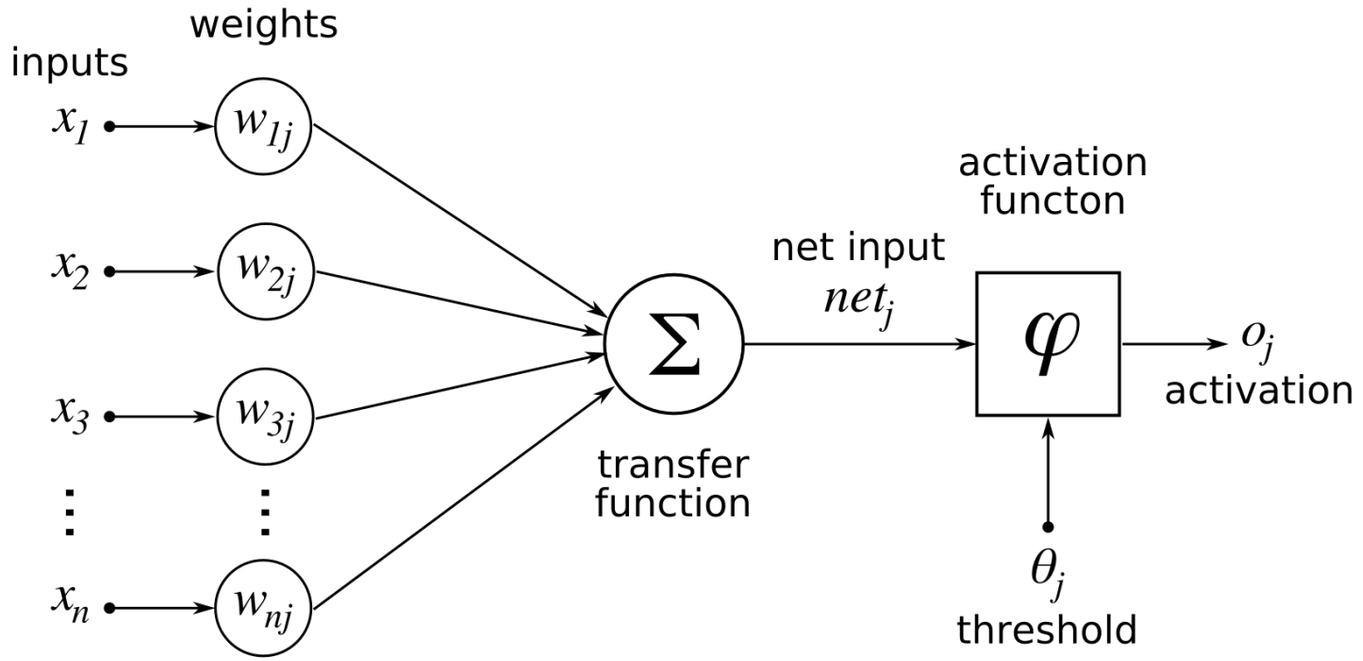
# Neural Networks

- Feed-forward Network `y = network(x)`



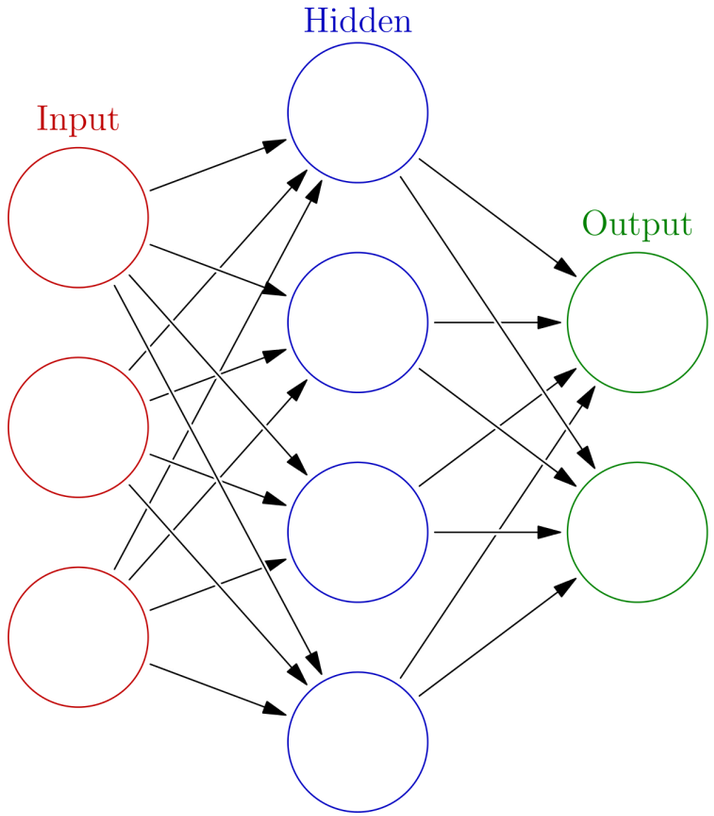
# Neural Networks

- Feed-forward Network `y = network(x)`



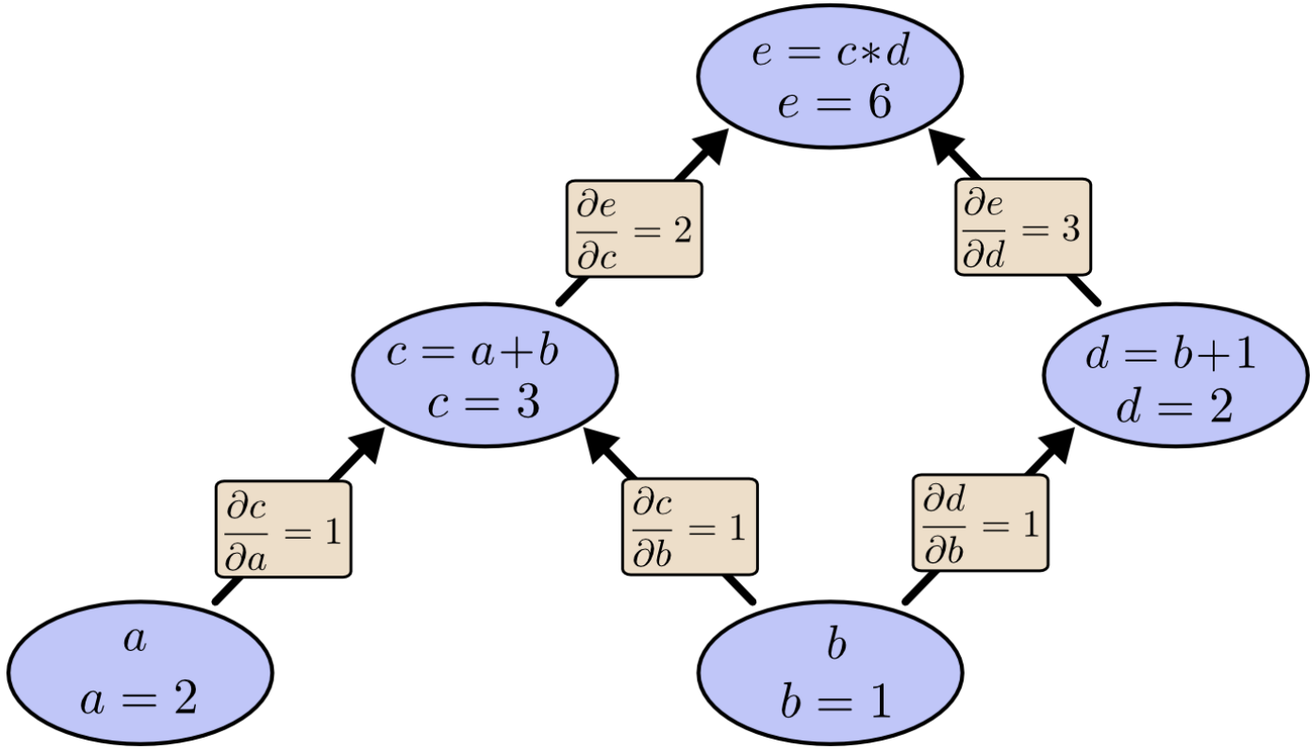
# Neural Networks

- Feed-forward Network `y = network(x)`



# Neural Networks

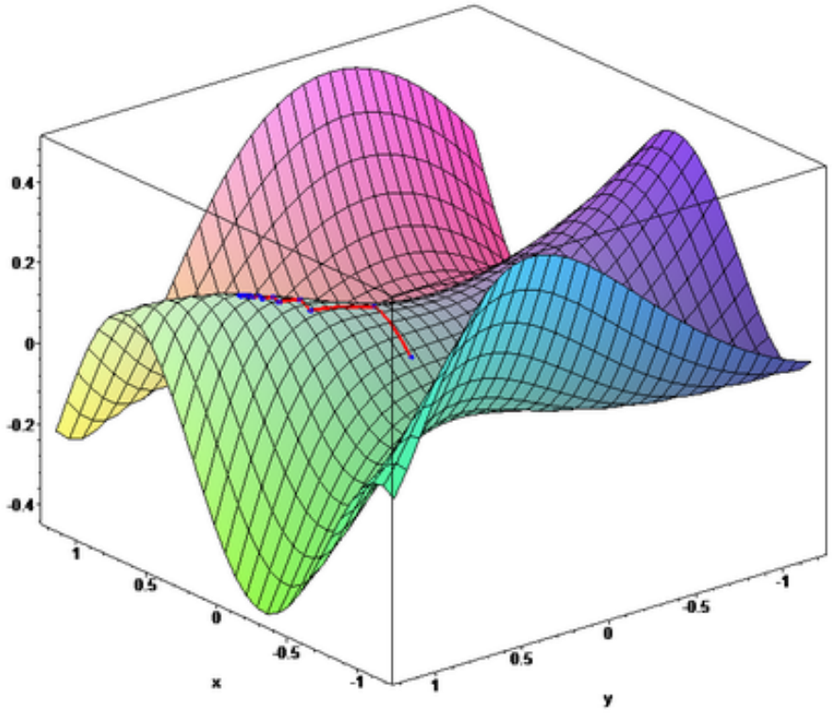
- Backpropagation `(loss_function(y_pred, y_true)).backward()`

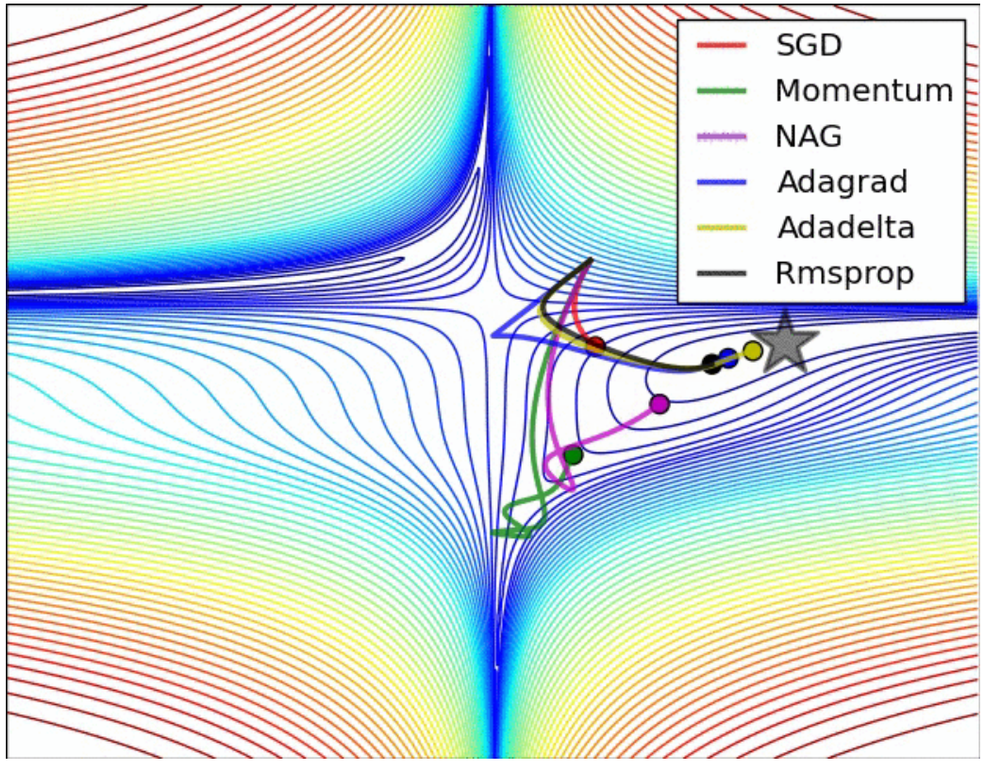



http://colah.github.io/posts/2015-08-Backprop/

# Neural Networks

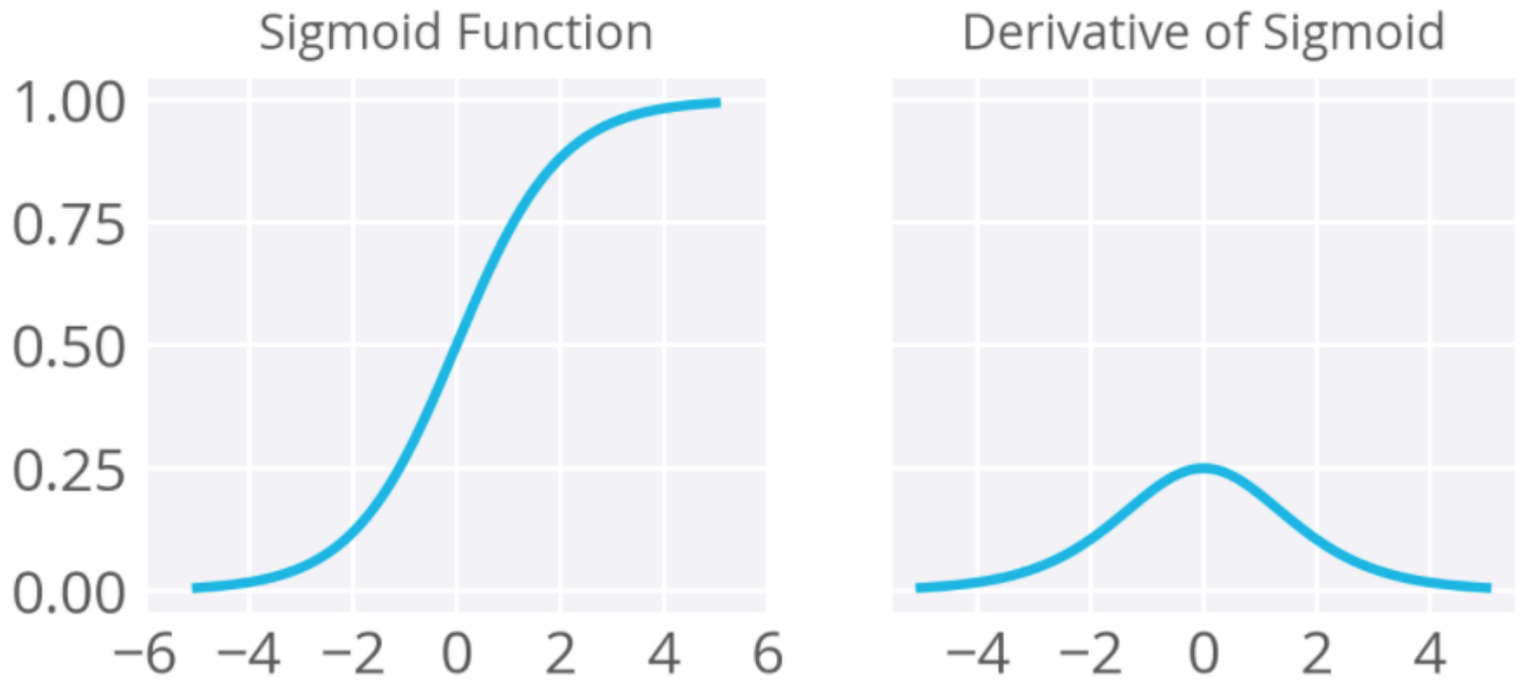
- Gradient Descent `torch/optim`
  - Stochastic Gradient Descent, Momentum, Adagrad, Adam, ...




SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

# Activation functions and non-linearity

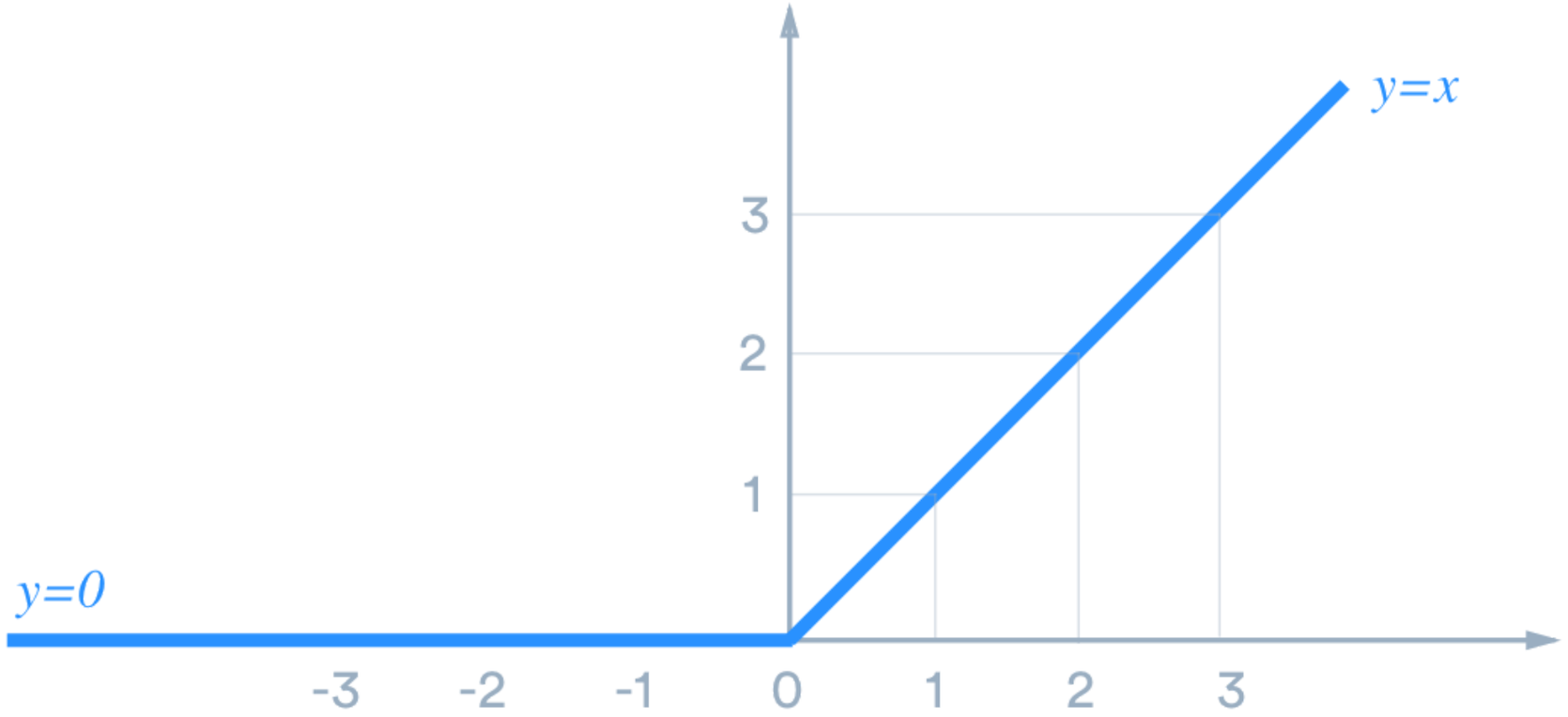
- $sigmoid(x) = \frac{1}{1+e^x}$ `torch.nn.Sigmoid`




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

# Activation functions and non-linearity

- $relu(x) = max(0, x)$ `torch.nn.ReLU`



# Activation functions and non-linearity

- $softmax(z) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ `torch.nn.Softmax`

# Loss functions

- L1 loss

$$L_1 = \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

- L2 loss

$$L_2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

# Loss functions

- Mean Square Error `torch.nn.MSELoss`

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

- Cross Entropy `torch.nn.CrossEntropyLoss`

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

# Loss functions

- Binary Cross Entropy `torch.nn.BCELoss`

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

# Regularization methods

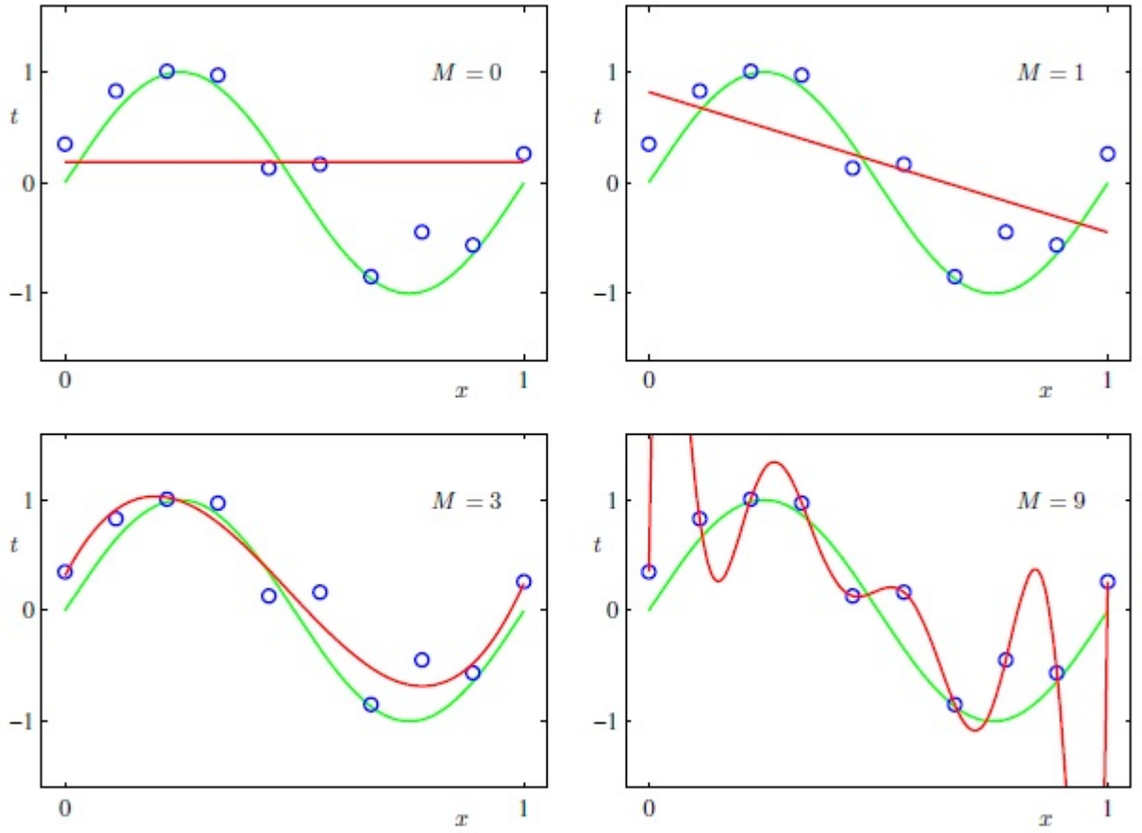



**Figure 1.4** Plots of polynomials having various orders $M$, shown as red curves, fitted to the data set shown in Figure 1.2.

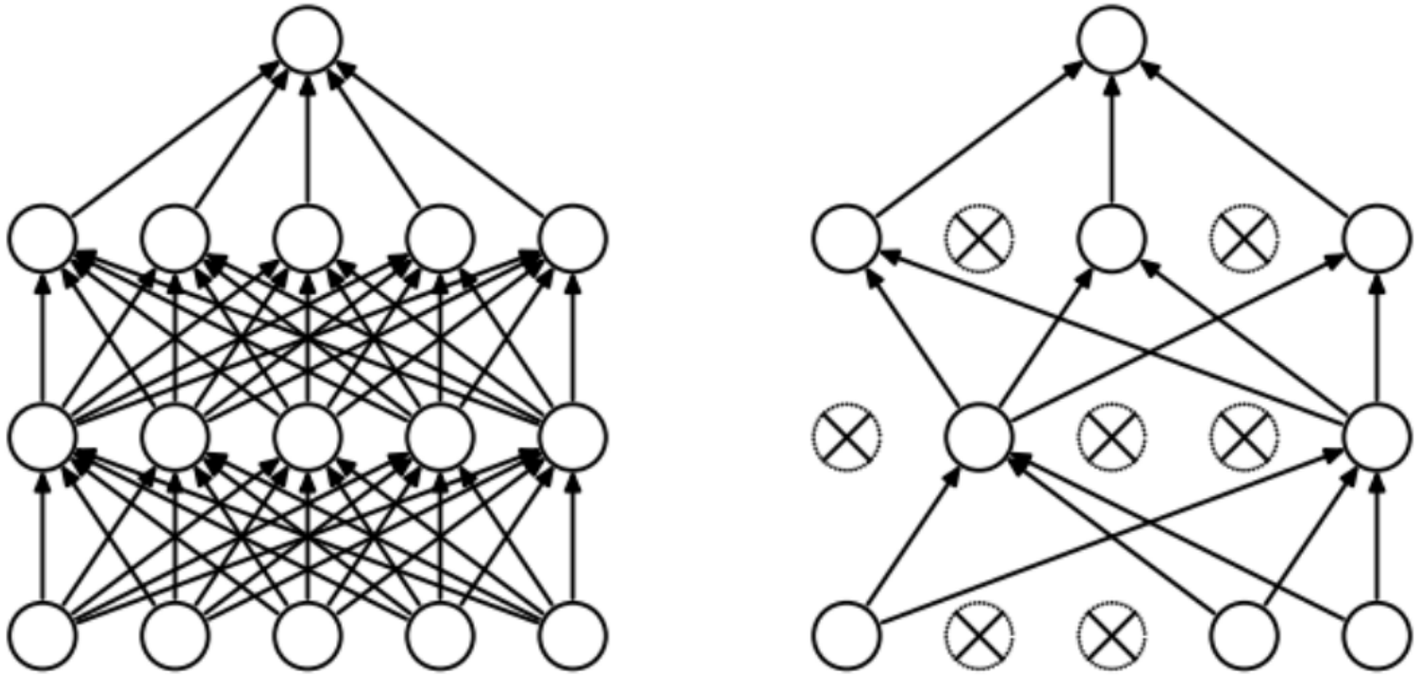
# Regularization methods

- Weight decay

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

# Regularization methods

- Dropout



(a) Standard Neural Net

(b) After applying dropout.

# 2. Hello PyTorch

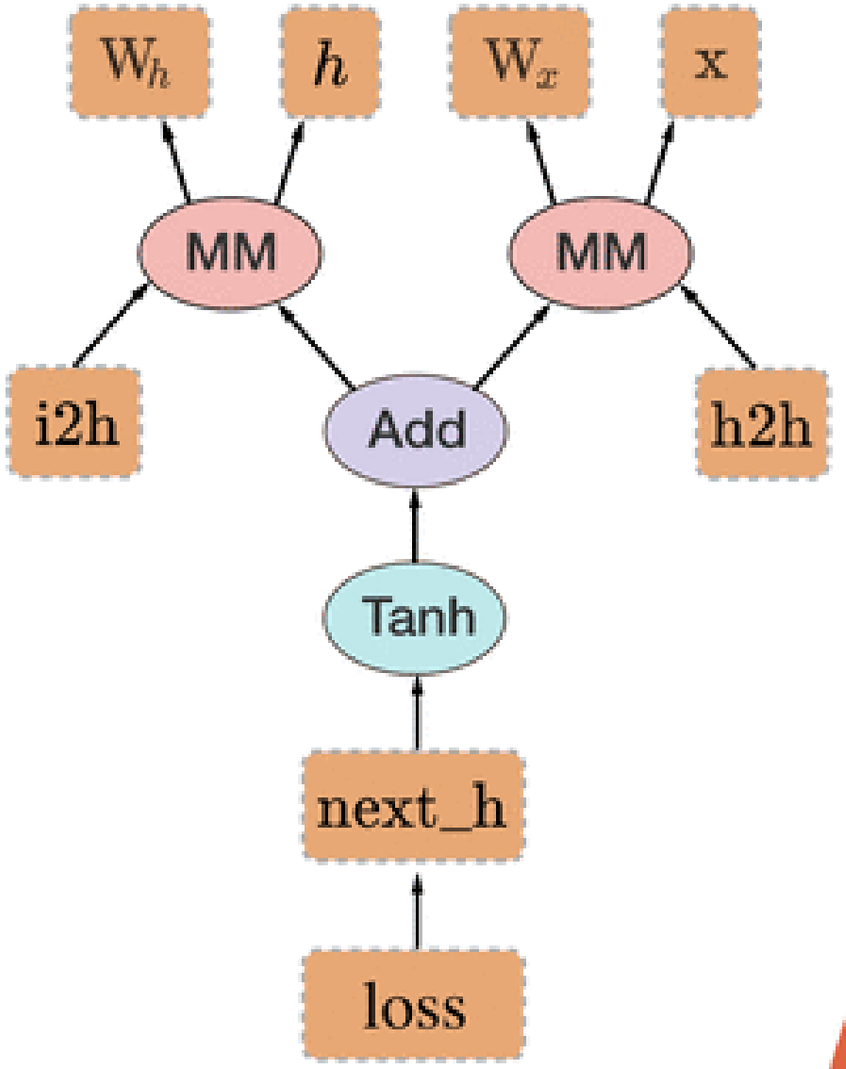

- Deep Learning Framework
  - Tensorflow, Keras, Torch, Chainer, MXNet
- Python-native, NumPy-friendly
- Dynamic graphs
- https://pytorch.org/
- https://pytorch.org/docs/stable/index.html

# 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

- **Ubuntu 16.04**
  - Basic shell functions:

    `ls -lah`, `cd`, `cp`, `mv`, `cat`, `grep -irv`, `kill`, `ps -ef`, `history`, `tar`, `unzip`, `apt`, `curl`

# 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

- **git**
  - Version control system
  - Basic commands:
    `git clone`, `git checkout`, `git add`, `git commit`, `git push`, `git pull`, `git diff`, `git log`
  - https://github.com/, https://gitlab.com/

# Our stack

- Docker CE
  - Container virtualization
  - Basic commands:

    `docker run`, `docker exec`, `docker start`, `docker stop`