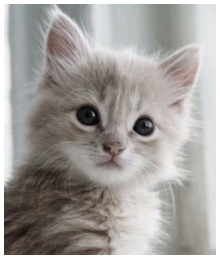
42578 Advanced Business Analytics

Deep Learning

Traditional approach

- Traditional machine learning models use hand-crafted features and a relatively simple trainable model

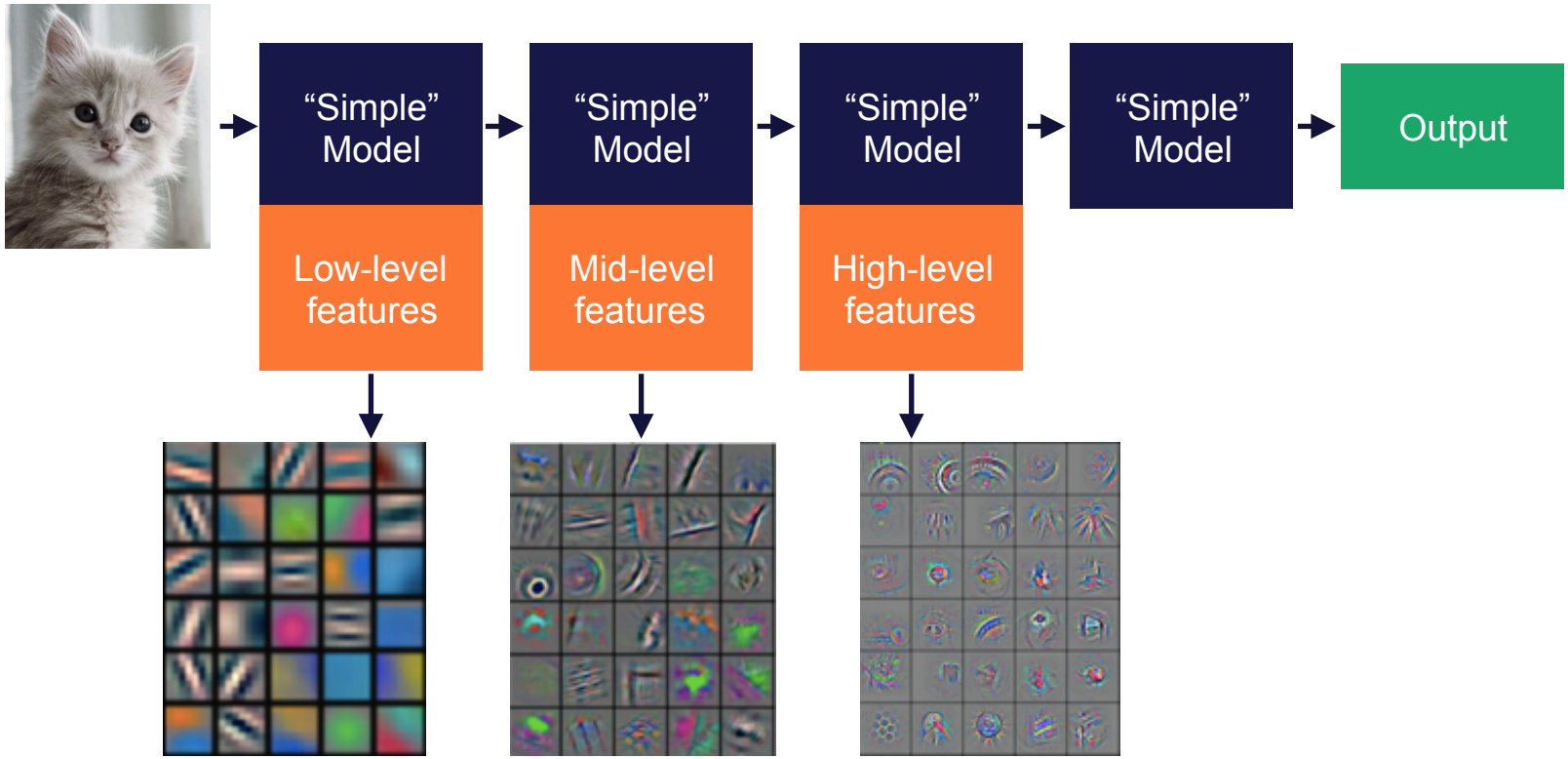


→ Hand-crafted feature extractor → “Simple” Model → Output

- **This approach has the following limitations:**
  - It is very tedious and costly to develop hand-crafted features
  - The hand-crafted features are usually highly depended on one application, and cannot be transferred easily to other applications

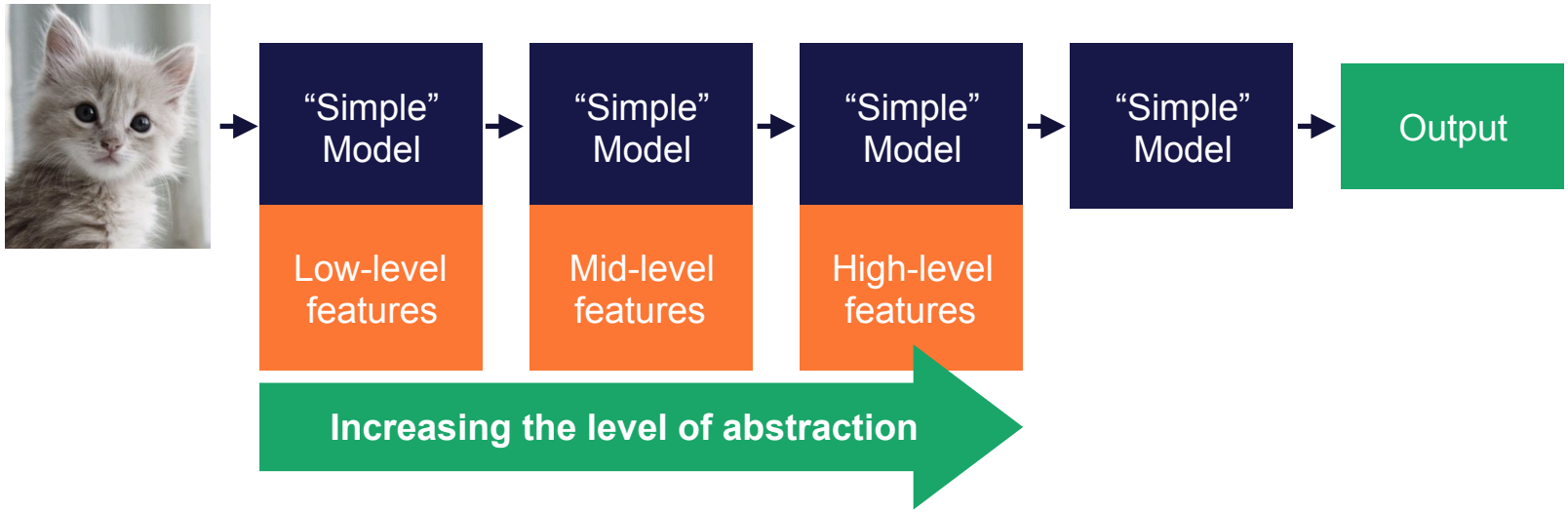
Deep learning approach

- Deep learning seeks to learn rich hierarchical representations (i.e. features) automatically through multiple stages of feature learning process



Learning hierarchical representations

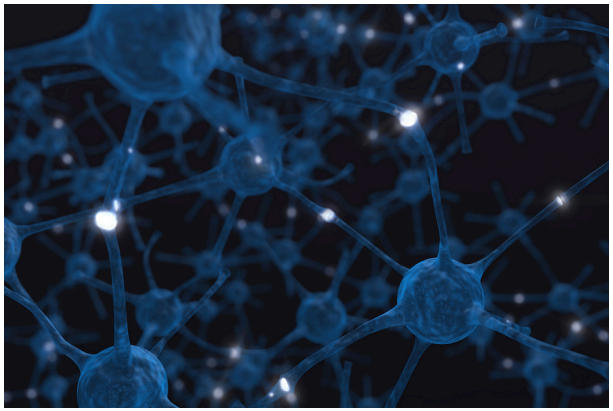
- Hierarchy of representations with increasing level of abstraction. Each stage is a kind of a trainable nonlinear feature transformation

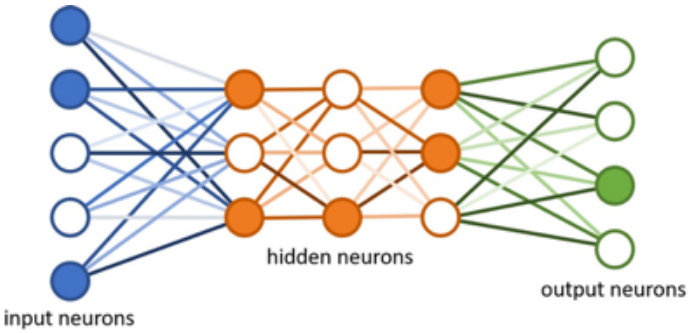


- **Image recognition**
  Pixel → edge → shape → motif → part → object

- **Text**
  Character → word → word group → clause → sentence → story

Neural-Network-Based Models

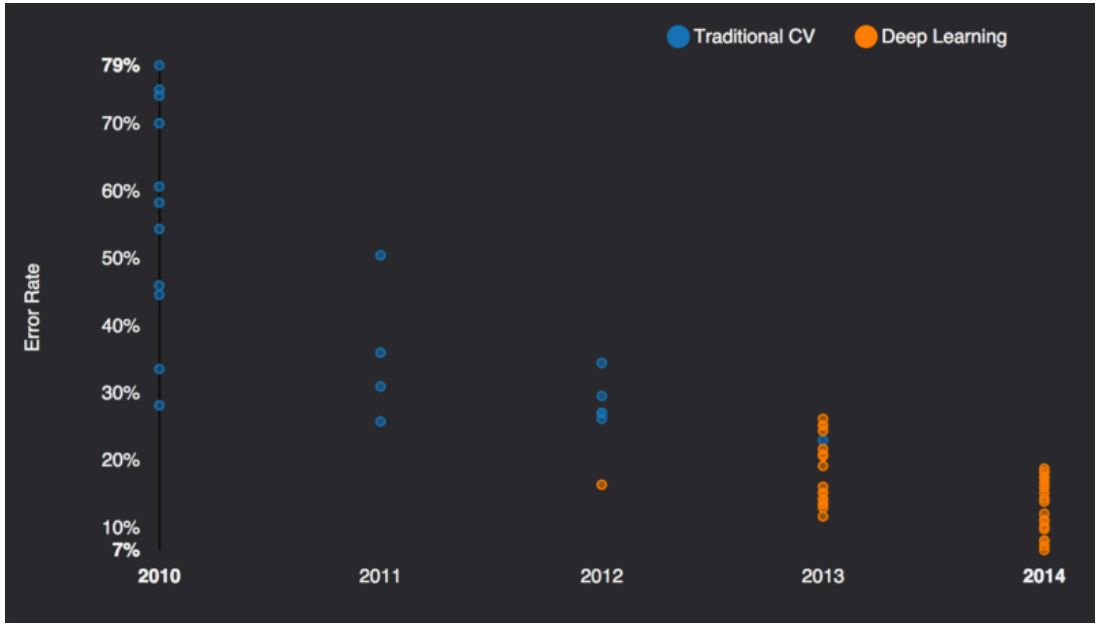




https://medium.com/xanaduai/making-a-neural-network-quantum-34069e284bcf

- High model capacity (complexity)
- Layered structure - hierarchical abstraction of the features!

Deep learning progress: Computer vision



Deep learning progress: NLP

- 2006 - Traditional NLP statistical machine translation
- 2016 - Neural machine translation

Deep learning progress: Style Transfer

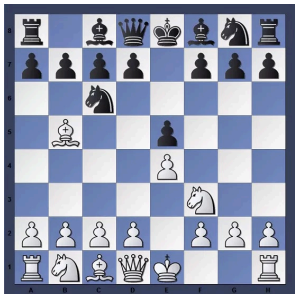
(Gatys, Ecker, and Bethge; 2015)

Deep learning progress: Generative Models

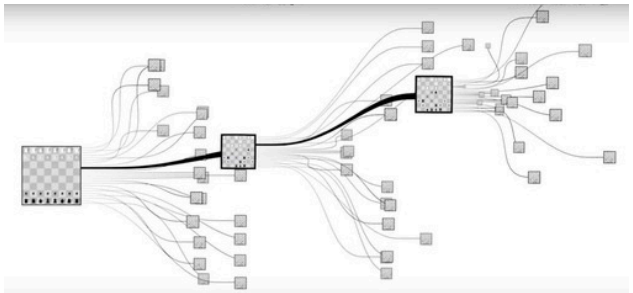
- Variational Autoencoder (VAE)
- Generative Adversarial Network (GAN)
- …

Deep learning progress: Reinforcement learning

Chess

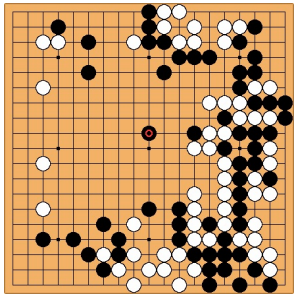


Complexity: 10^{47}

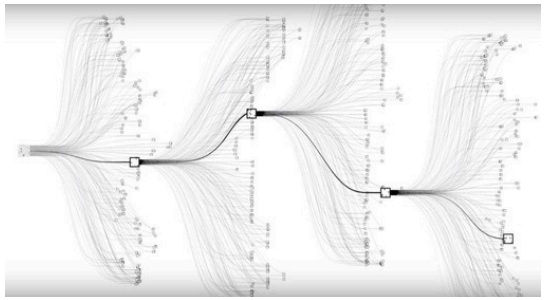


Deep Blue, 1996

Go



Complexity: 10^{170}



AlphaGo, 2016

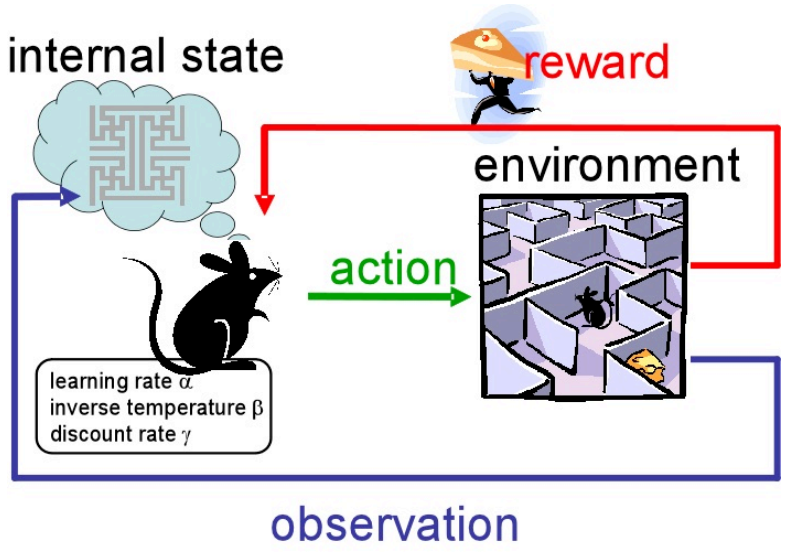
https://www.pinterest.com/pin/827747606489038112/
https://www.pinterest.com/pin/346284658845590457/

Number of atoms in the observable universe: 10^{78}-10^{82}

Deep learning progress: Reinforcement learning

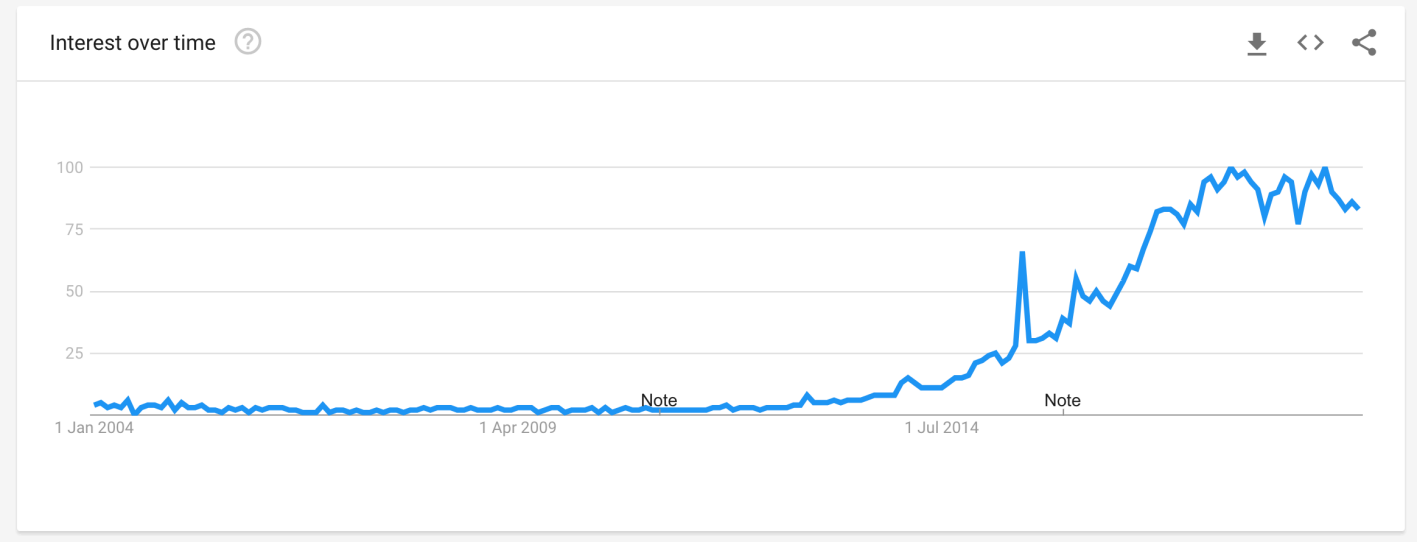
- Computer Games: https://blog.openai.com/reinforcement-learning-with-prediction-based-rewards/

- Motion learning: https://www.youtube.com/watch?v=gn4nRCC9TwQ



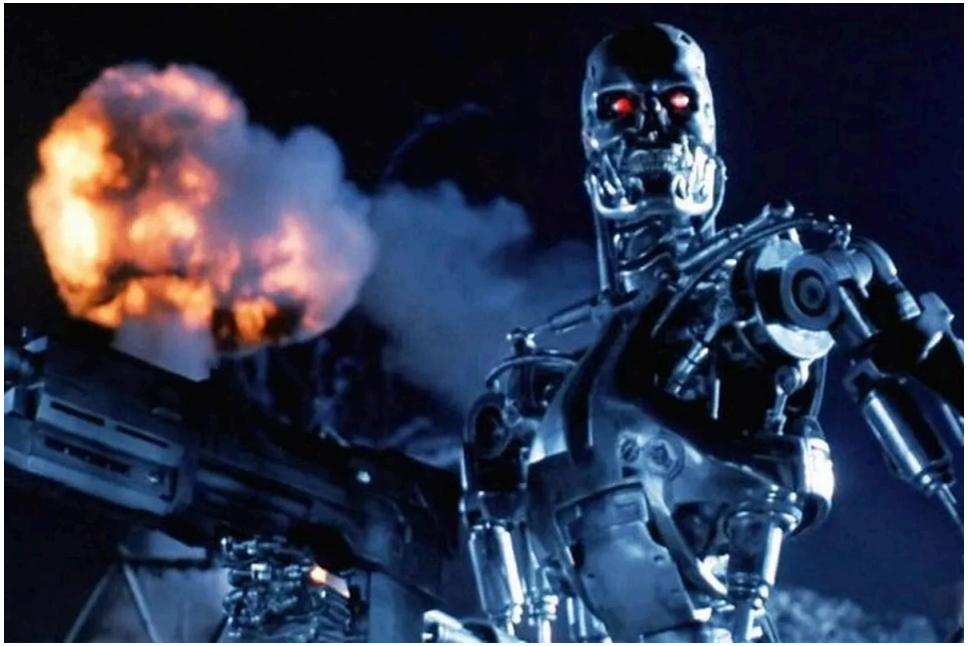
https://becominghuman.ai/the-very-basics-of-reinforcement-learning-154f28a79071

Deep learning hype



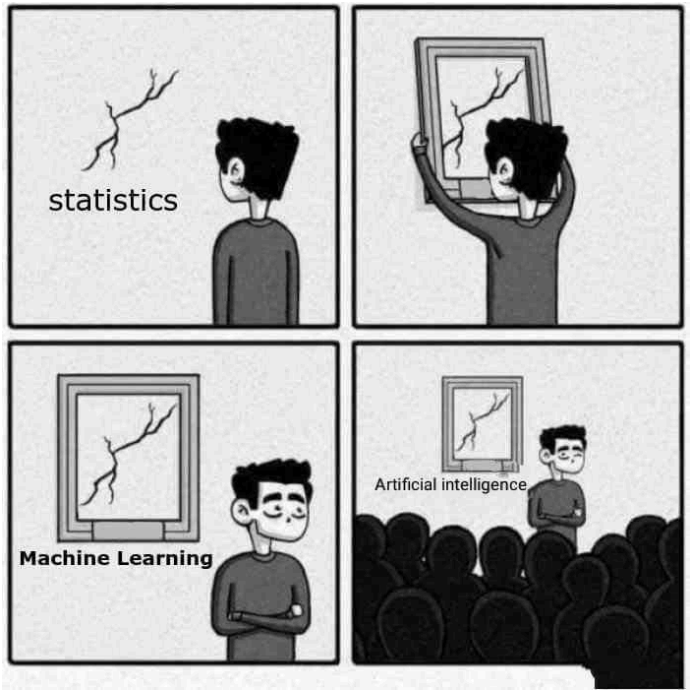
Google trends for “Deep learning”

Public opinion



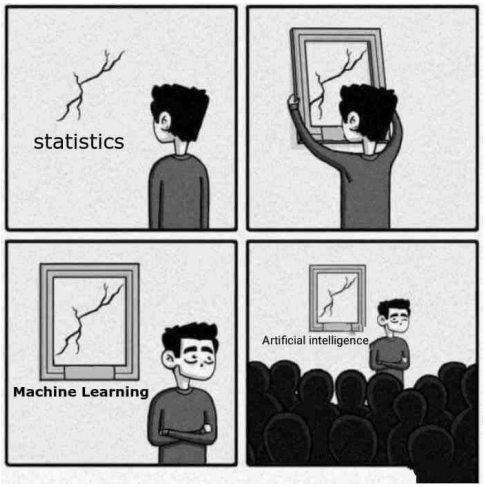
"Terminator"

Criticism

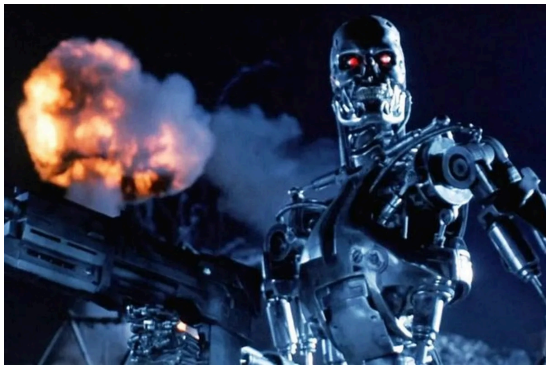


https://www.instagram.com/sandserifcomics/

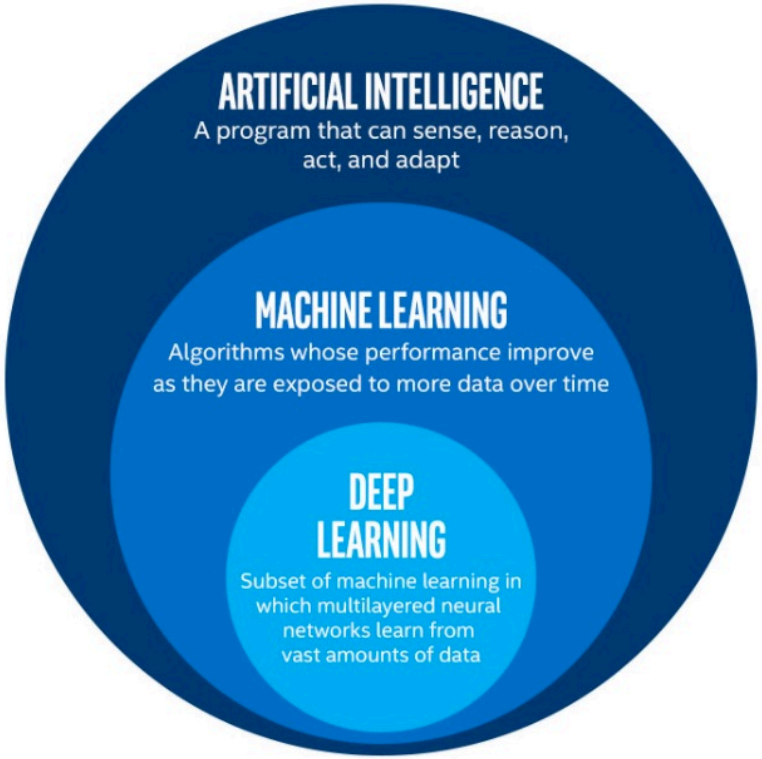
Reality?



VS



Reality



https://towardsdatascience.com/cousins-of-artificial-intelligence-dda4edc27b55

When you’re fundraising, it’s AI

When you’re hiring, it’s ML

When you’re implementing, it’s linear regression

When you’re debugging, it’s printf()

— Baron Schwartz (@xaprb) November 15, 2017

Deep learning applications

- Supervised (regression/classification)
- Unsupervised
- Reinforcement learning

- **Requirements:**
  - a complex task
  - a lot of data
  - computational power

Artificial Neural Networks Recap

Linear neuron (= Linear regression)

$$\hat{y} = b + \sum_{i=1}^{N} w_i x_i$$

model

+

const

$$L(b, \mathbf{w}) = \frac{1}{2|\mathscr{D}|} \sum_{\mathbf{x}, y \in \mathscr{D}} \left(y - \hat{y}(\mathbf{x}; b, \mathbf{w})\right)^2$$

regression loss (MSE)

+ **Optimisation** = **Trained model**

$\mathscr{D}$ — training data (samples), (**x**, y) pairs

$|\mathscr{D}|$ — number of training samples

y is numerical, $y \in \mathbb{R}$

Linear neuron (= Linear regression)

$$\hat{y} = b + \sum_{i=1}^{N} w_i x_i \quad + \quad L(b, \mathbf{w}) = \frac{1}{2|\mathscr{D}|} \sum_{\mathbf{x}, y \in \mathscr{D}} \left(y - \hat{y}(\mathbf{x}; b, \mathbf{w})\right)^2 \quad + \quad \text{Optimisation} \quad = \quad \text{Trained model}$$

const

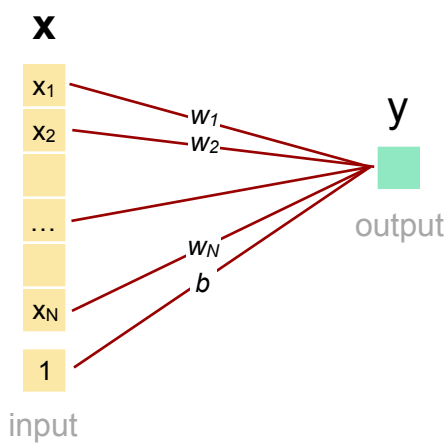
model

regression loss (MSE)

$\mathscr{D}$ — training data (samples), (**x**, y) pairs

$|\mathscr{D}|$ — number of training samples

y is numerical, $y \in \mathbb{R}$



Note: *Bias is often not shown but implied*

“Traditional” neuron (= Logistic regression)

$$z = b + \sum_{i=1}^{N} w_i x_i$$

$$\hat{y} = \frac{1}{1+e^{-z}} = \frac{e^{z}}{1+e^{z}} = \text{sigmoid}(z)$$

model

y is binary, $y \in \{0,1\}$

+

$$L(b, \mathbf{w}) \propto -\sum_{\mathbf{x}, y \in \mathscr{D}} y \log \hat{y}(\mathbf{x}; b, \mathbf{w}) + (1-y) \log\left(1 - \hat{y}(\mathbf{x}; b, \mathbf{w})\right)$$

classification loss (binary cross-entropy)

+ =

“Traditional” neuron (= Logistic regression)

$$z = b + \sum_{i=1}^{N} w_i x_i$$

$$\hat{y} = \frac{1}{1+e^{-z}} = \frac{e^{z}}{1+e^{z}} = \text{sigmoid}(z)$$

activation function

model

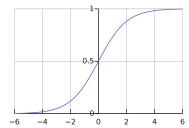
y is binary, $y \in \{0,1\}$

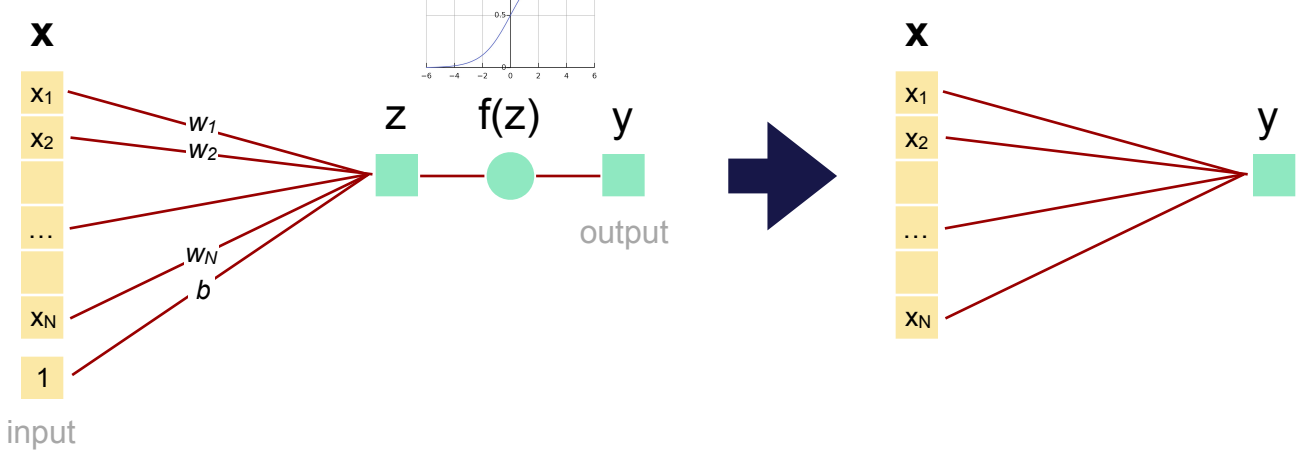
+

$$L(b, \mathbf{w}) \propto -\sum_{\mathbf{x}, y \in \mathcal{D}} y \log \hat{y}(\mathbf{x}; b, \mathbf{w}) + (1-y) \log\left(1 - \hat{y}(\mathbf{x}; b, \mathbf{w})\right)$$

classification loss (binary cross-entropy)

+ =





Note: *Activation is usually not shown but implied*

Softmax (= “Multinomial” logistic regression)

$$z = b + \sum_{i=1}^{N} w_i x_i \qquad \hat{y} = \frac{e^z}{1+e^z}$$

$$L(b,\mathbf{w}) \propto -\sum_{\mathbf{x},y\in\mathcal{D}} y \log \hat{y}(\mathbf{x};b,\mathbf{w}) + (1-y)\log\left(1-\hat{y}(\mathbf{x};b,\mathbf{w})\right)$$

binary classification

K=2 classes

=

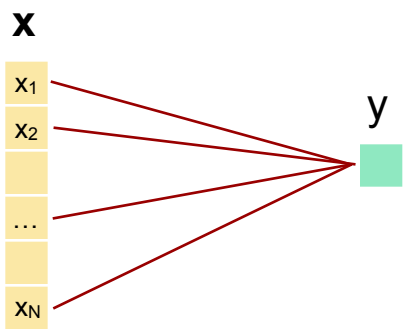
softmax output

$$z_k = b_k + \sum_{i=1}^{N} w_{ki} x_i \qquad \hat{y}_k = \frac{e^{z_k}}{\sum_{k=1}^{K} e^{z_k}}$$

$$L(b,\mathbf{w}) \propto -\sum_{\mathbf{x},y\in\mathcal{D}} \sum_{k=1}^{K} y_k \log \hat{y}_k(\mathbf{x};b,\mathbf{w})$$

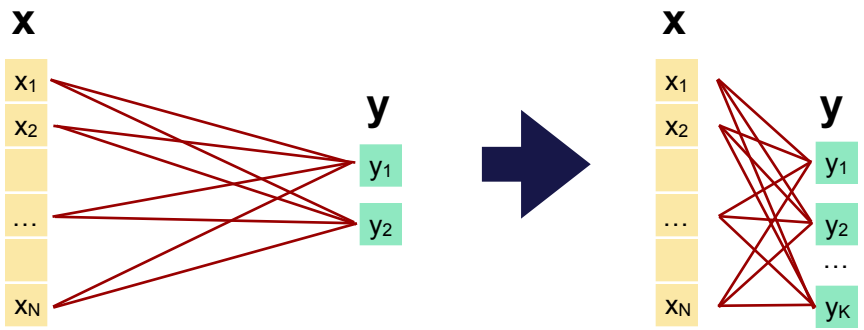
cross-entropy loss

multiclass classification



K=2 classes

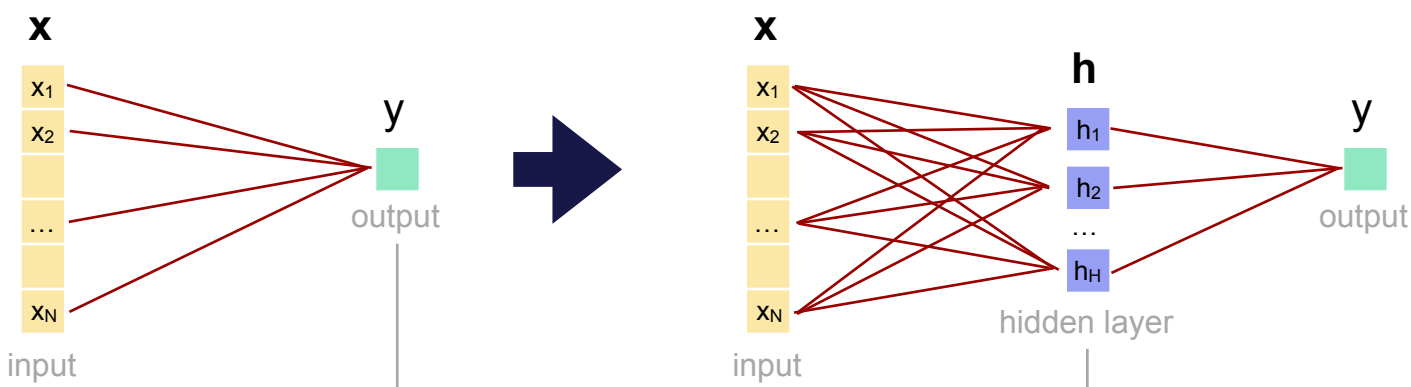
=



y is binary, $y \in \{0,1\}$

y is categorical (K=2), $y_i \in \{0,1\}$

Multilayer perceptron



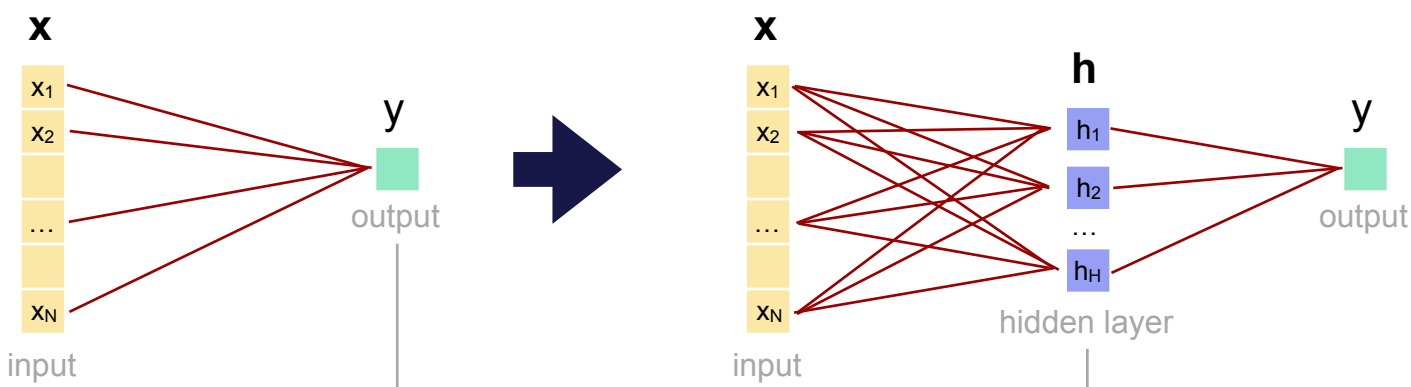
Problem-specific

- Regression: Linear neuron, MSE loss
- Binary classification: Sigmoid, binary CE loss
- Multiclass classification: Softmax, CE loss

Linear sum of inputs + <u>nonlinearity</u>

For example, if y is categorical and sigmoid neurons are used in h, then NN = stacked logistic regressions

Multilayer perceptron



Problem-specific

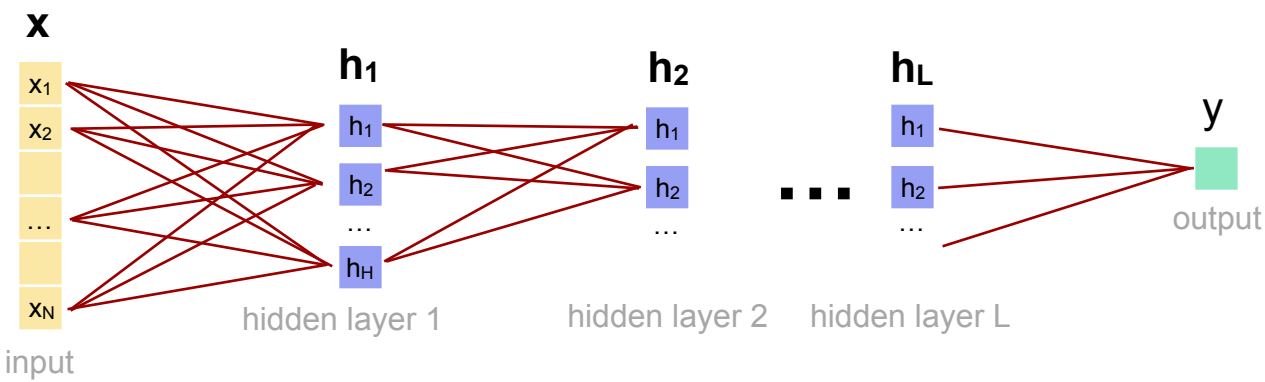
- Regression: Linear neuron, MSE loss
- Binary classification: Sigmoid, binary CE loss
- Multiclass classification: Softmax, CE loss

Linear sum of inputs + <u>nonlinearity</u>

For example, if y is categorical and sigmoid neurons are used in h, then NN = stacked logistic regressions

- **In theory:** MLP is already a universal approximator, i.e., it can approximate any y=f(x) given a large number of hidden neurons
- **In practice:** Works only for simple nonlinear problems (more details in the next slides)

Deep (fully connected) neural nets

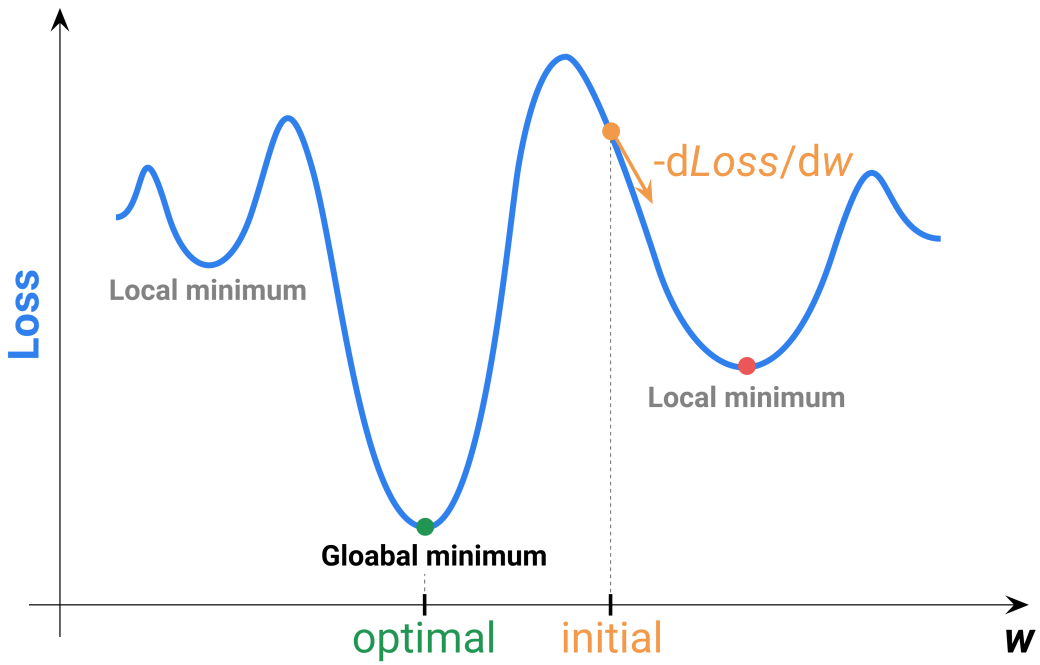


$$\hat{y} = o\left(f\left(f\left(\ldots f(\mathbf{x};\mathbf{w}_1)\ldots;\mathbf{w}_{L-1}\right);\mathbf{w}_L\right);\mathbf{w}_o\right)$$

$$L(\mathbf{W};\mathscr{D}) = \sum_{\mathscr{D}} \mathscr{L}\left(\hat{y}(\mathbf{W};\mathbf{x});y\right)$$

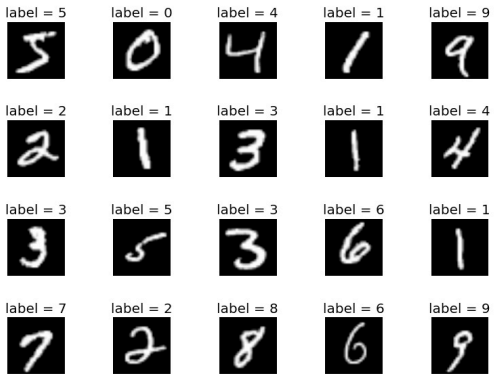
How to learn the parameters?

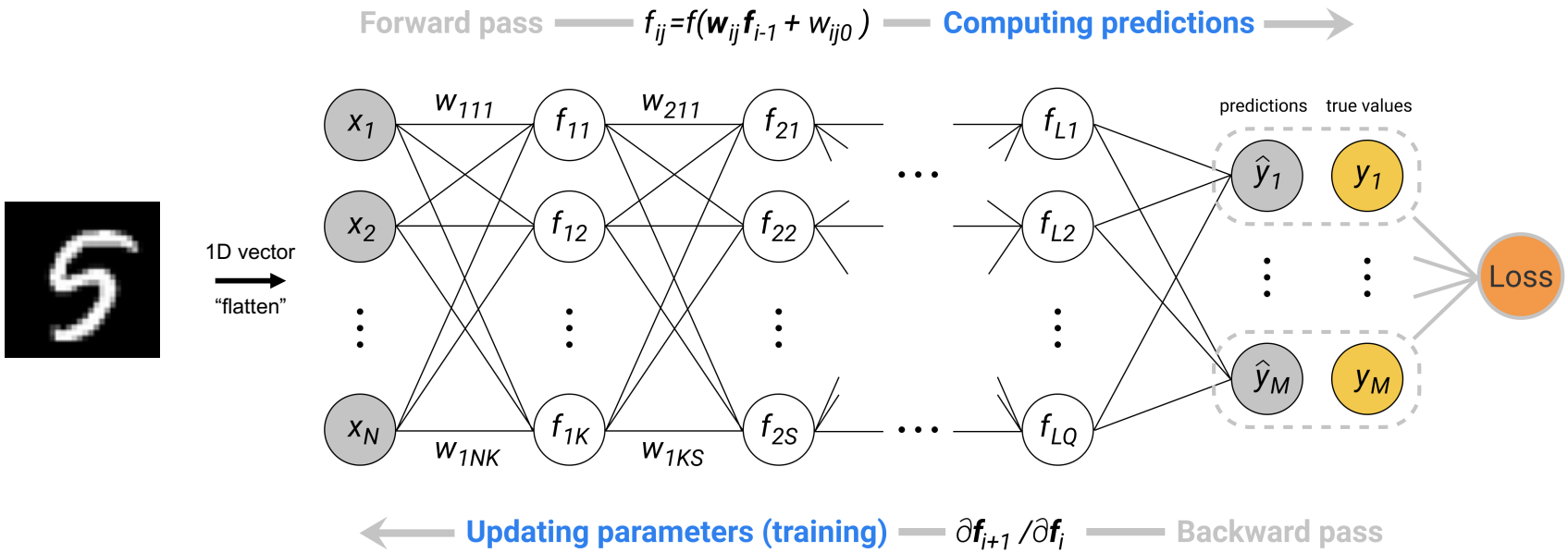
Optimization (a.k.a. “training” / “learning”)



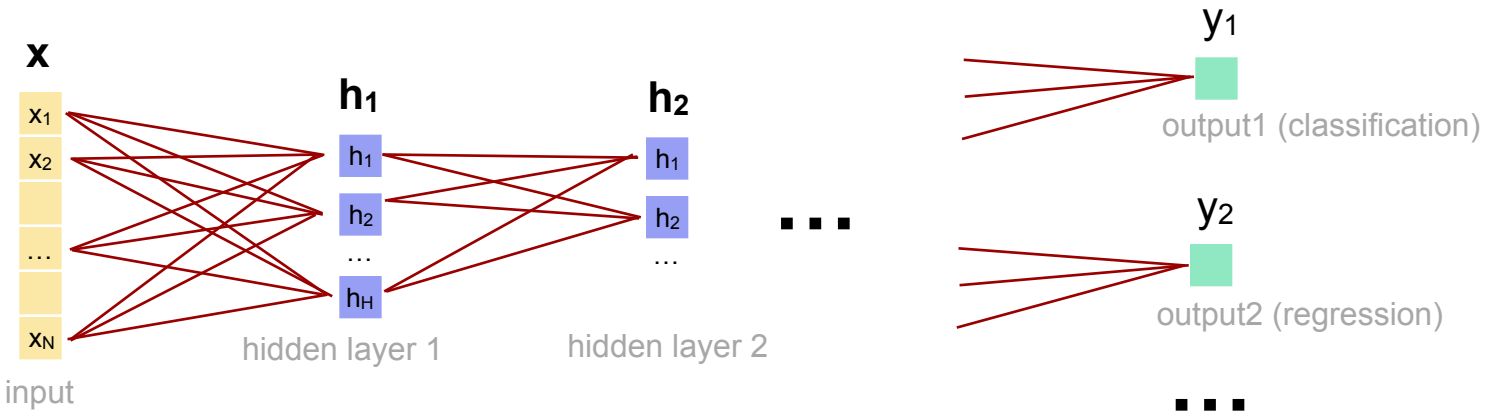
- **Gradient-based methods**
- **Back-propagation** is simply a chain differentiation rule applied to the loss function to calculate its gradients w.r.t. the parameters

MNIST dataset





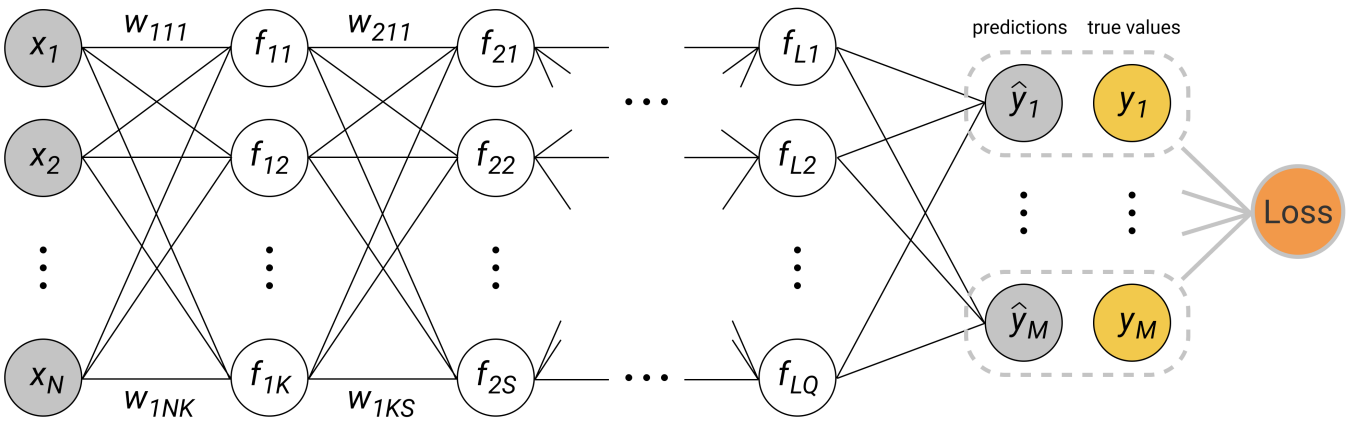
Multiple tasks at the same time? Easy!



$$L(\mathbf{W}; \mathscr{D}) = \alpha L_1 + \beta L_2 + \ldots$$

Vanishing gradient problem

Forward pass — $f_{ij}=f(\mathbf{w}_{ij}\mathbf{f}_{i-1}+w_{ij0})$ — Computing predictions →



← Updating parameters (training) — $\partial\mathbf{f}_{i+1}/\partial\mathbf{f}_i$ — Backward pass

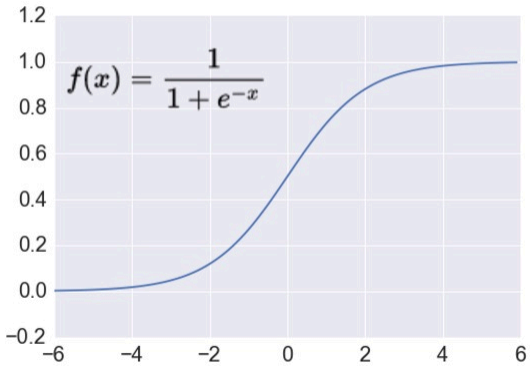
$$\frac{\partial L}{\partial \mathbf{w}_l} \propto \underline{\frac{\partial L}{\partial \mathbf{y}_L}\frac{\partial \mathbf{y}_L}{\partial \mathbf{y}_{L-1}}\cdots\frac{\partial \mathbf{y}_l}{\partial \mathbf{w}_l}}$$

Product grows exponentially with the number of layers

↓

Gradients go to 0 with the depth of the network (especially, when sigmoid activation is used)

Activation functions: Sigmoid



http://adilmoujahid.com/images/activation.png

Takes a real-valued number and “squashes” it into the range between 0 and 1.

$$R \rightarrow [0, 1]$$

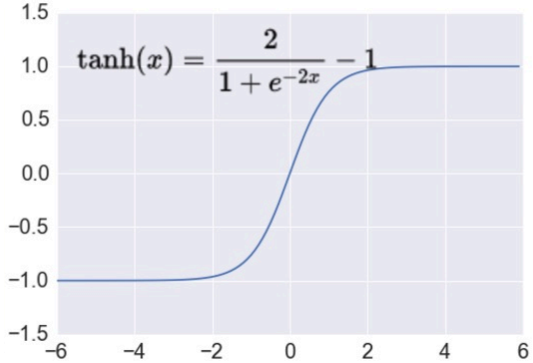
+ Nice interpretation as the firing rate of a neuron

- 0 = not firing at all
- 1 = fully firing

- Sigmoid neurons saturate and kill gradients, thus NN will barely learn when the neuron’s activation are 0 or 1 (saturate)

- 🙁 gradient at these regions almost zero
- 🙁 almost no signal will flow to its weights
- 🙁 if initial weights are too large then most neurons would saturate

Activation functions: Tanh



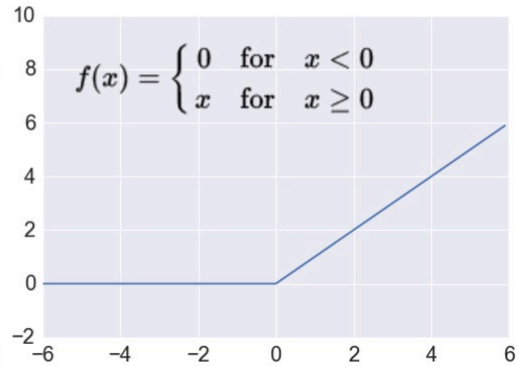
http://adilmoujahid.com/images/activation.png

Takes a real-valued number and “squashes” it into the range between -1 and 1.

$$R \rightarrow [-1, 1]$$

- Like sigmoid, tanh neurons **saturate**
- Unlike sigmoid, output is **zero-centered**
- Tanh is a **scaled sigmoid**: *tanh(x)=2sigm(2x)-1*

Activation functions: ReLU



http://adilmoujahid.com/images/activation.png

Takes a real-valued number and thresholds it at zero

$f(x) = \max(0,x) \qquad R \rightarrow R_+$

- **Most Deep Networks use ReLU nowadays**

🙂 Trains much **faster**
- accelerates the convergence of SGD
- due to linear, non-saturating form

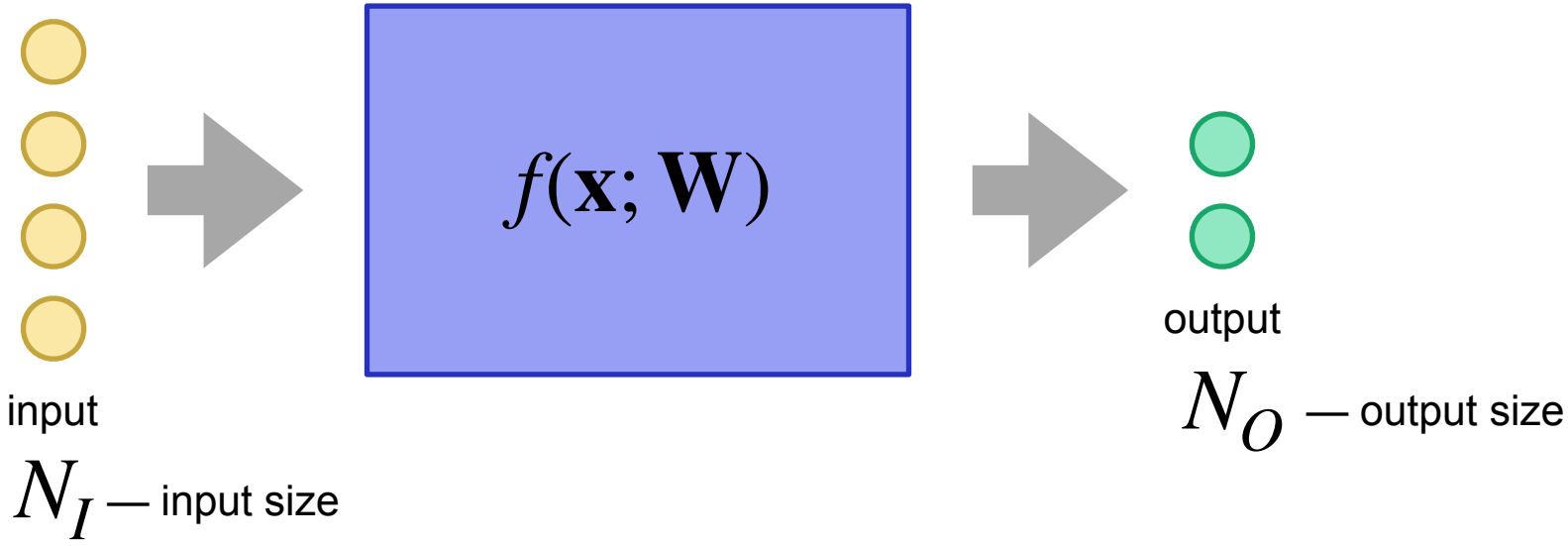
🙂 **Less expensive** operations
- compared to sigmoid/tanh (exponentials etc.)
- implemented by simply thresholding a matrix at zero

🙂 More **expressive**

🙂 **Prevents** the gradient vanishing problem

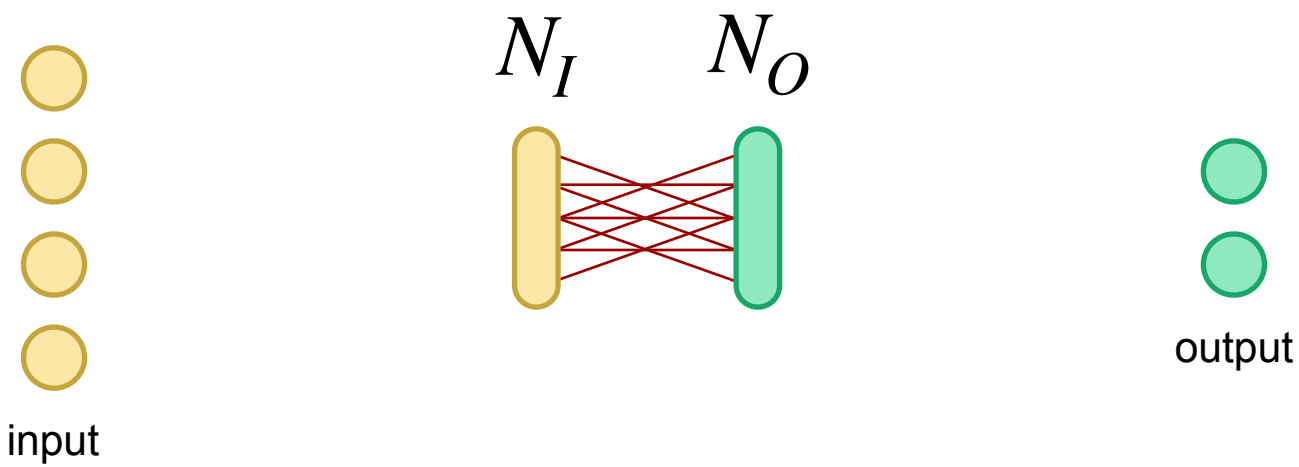
Neural Networks Architectures

Input → Output

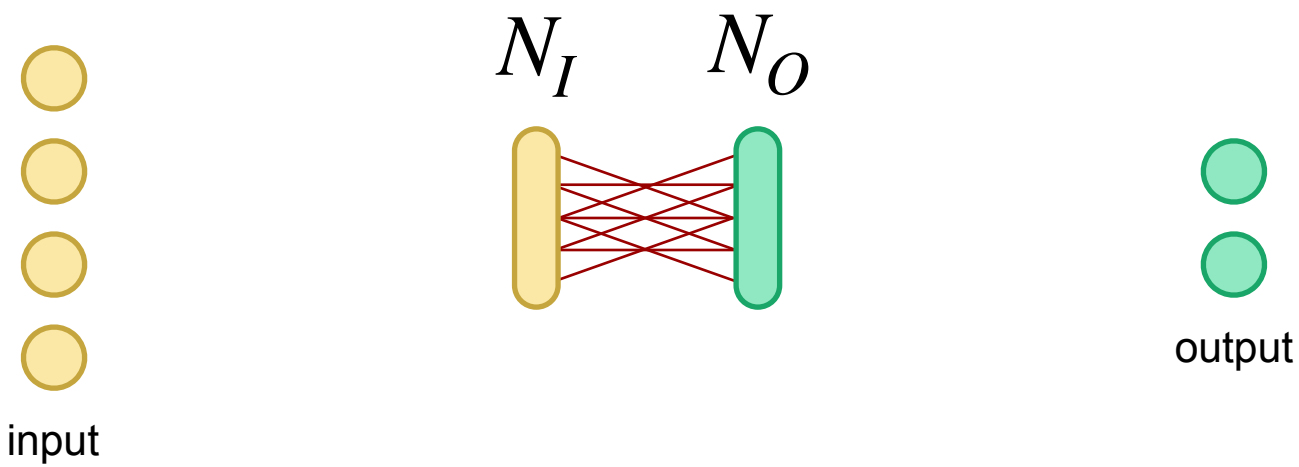


N — number of parameters in f (=dim(**W**))

No hidden neurons



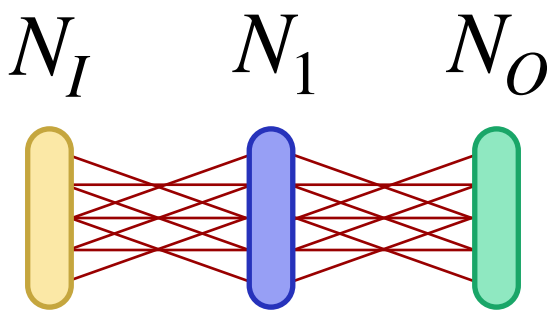
No hidden neurons



$$N = N_I N_O$$

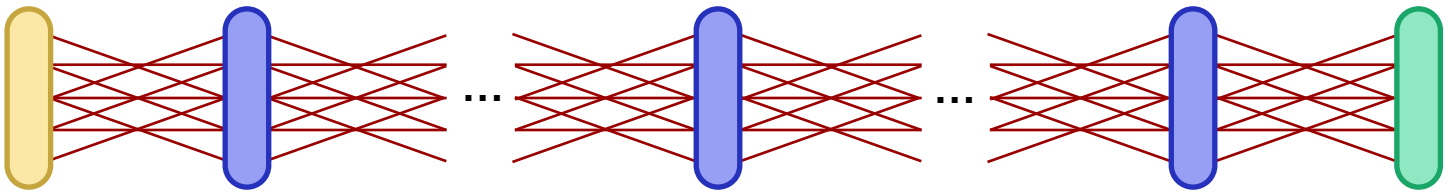
(ignoring bias term for simplicity)

Fully connected NN: 1 hidden layer



$$N = N_I N_1 + N_1 N_O$$

Fully connected NN: *L* hidden layers

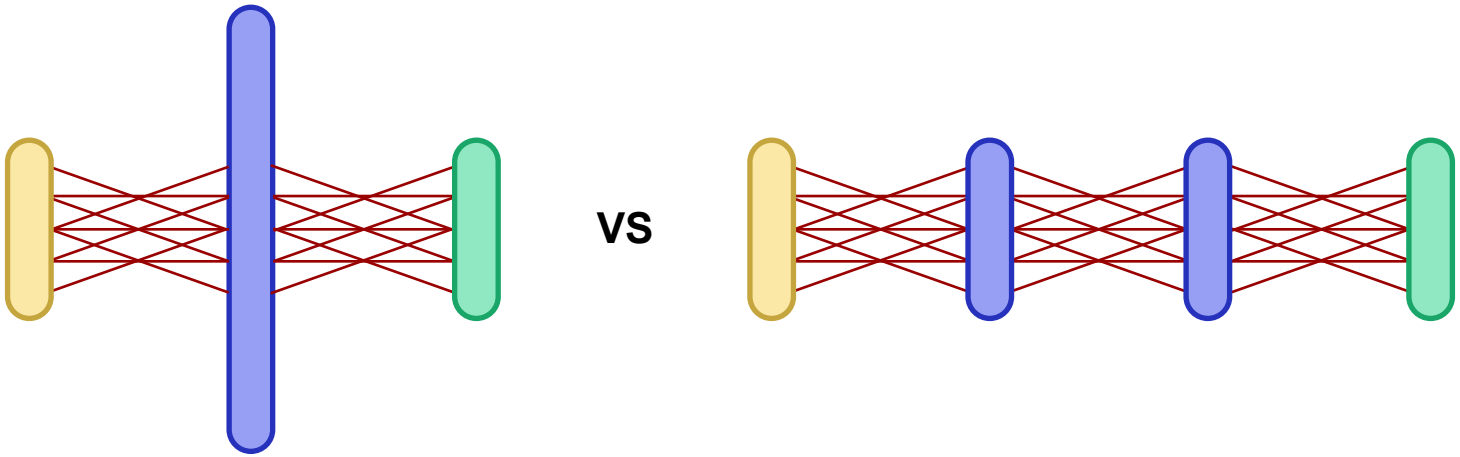


$$N = N_I N_1 + N_1 N_2 + \ldots + N_{L-1} N_L + N_L N_O$$

$$N \sim L \overline{N_h}^2$$

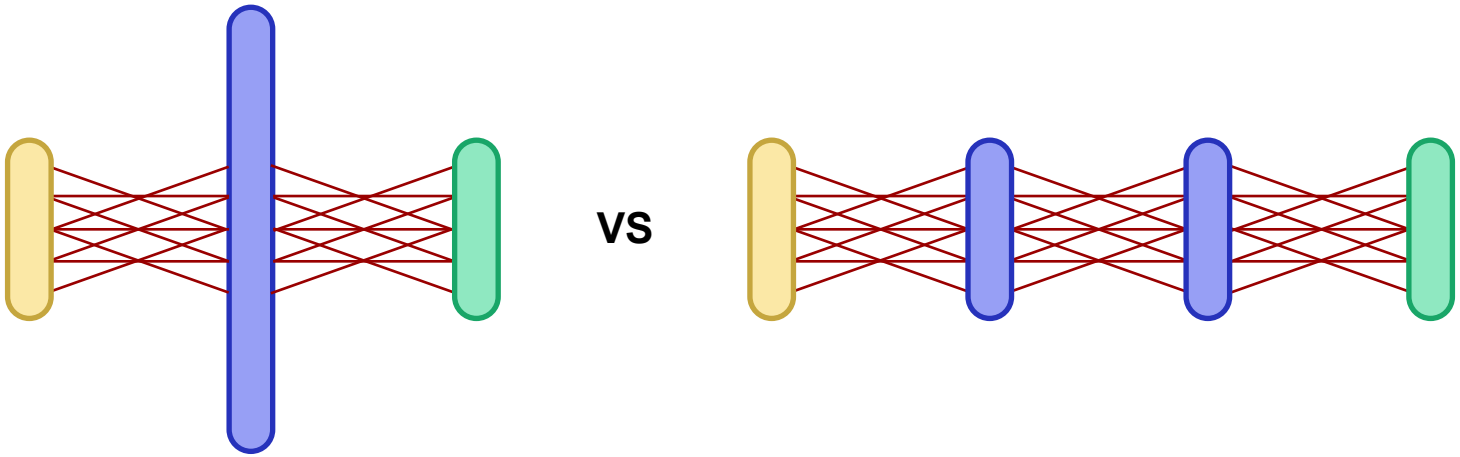
$\overline{N_h}$ — average number of neurons in a hidden layer

Fully connected NN: Depth vs breadth



Given the number of parameters in both is the same, which one is better?

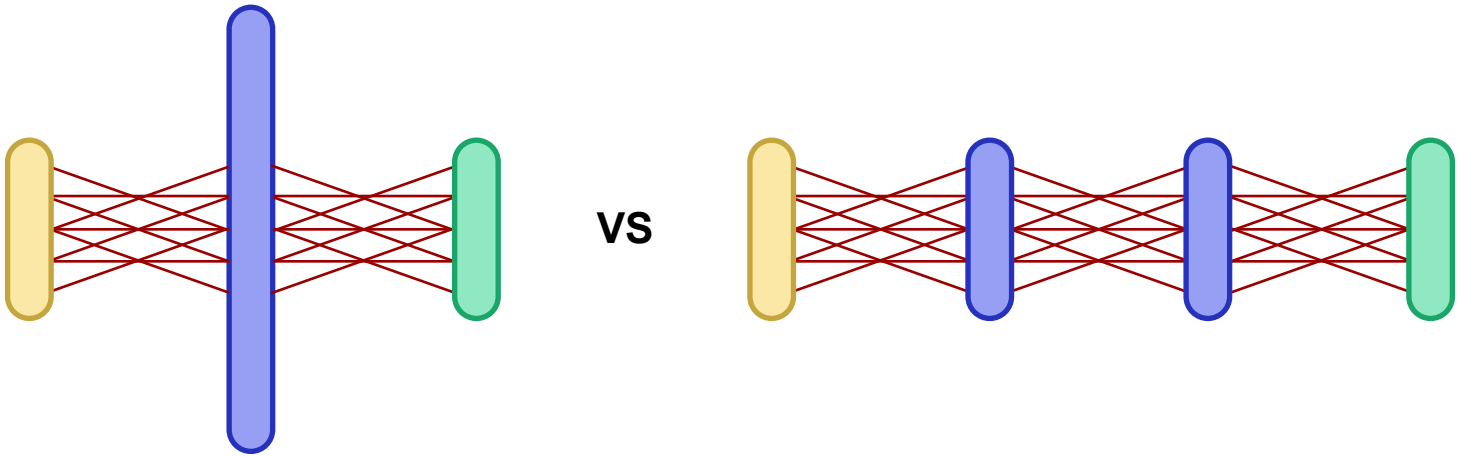
Fully connected NN: Depth vs breadth



Given the number of parameters in both is the same, which one is better?

It depends…

Fully connected NN: Depth vs breadth



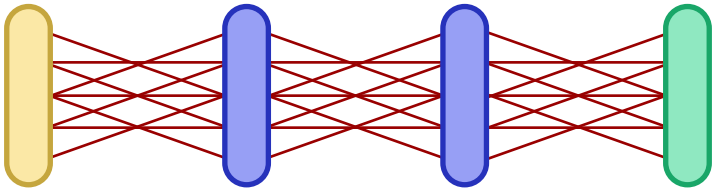
Given the number of parameters in both is the same, which one is better?

It depends…

Assumption: Hierarchical abstraction of features (pixel→edge→shape→object)

Layered architecture!

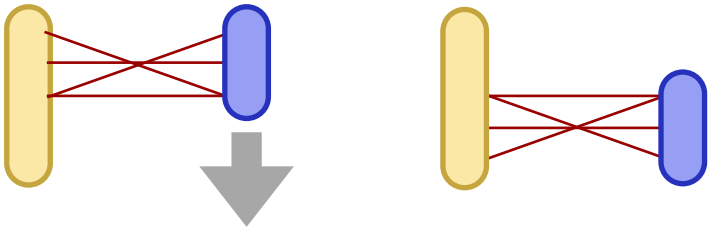
Can we do better?



Yes! But we need more assumptions…

For example: Dependencies in the input data for images (spatial) or text (sequential)

Then, we can use **weight sharing:**
The same hidden layers are re-used for different parts of input → significantly less parameters!

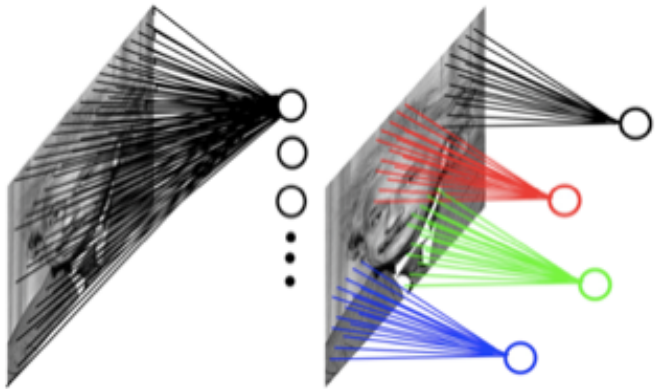


2 most basic deep NN architectures based on the “weight sharing” principle

- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)

Convolutional Neural Network (CNN)

- Input can have very high dimension. Using a fully-connected neural network would need a large amount of parameters.
- Inspired by the neurophysiological experiments conducted by [Hubel & Wiesel 1962], CNNs are a special type of neural network whose hidden units are only connected to local receptive field. The number of parameters needed by CNNs is much smaller.



Example: 200x200 image

a) fully connected: 40,000 hidden units => 1.6 billion parameters
b) CNN: 5x5 kernel, 100 feature maps => 2,500 parameters

Convolutional Neural Network (CNN)

- **Built on the two ideas:**
  - Convolution
  - Pooling

Convolution operation in CNN

- Input **X**: an image (2-D array) x 1 channel (grey)
- Convolution kernel/operator/filter (2-D array): **w**
- Feature map (2-D array of processed data): **s**
- Convolution operation in 2-D domains:

X * **w** = **s**

1	0	2	3	0
5	0	3	2	1
4	1	1	0	0
0	4	3	4	1
1	0	5	0	4

1	0	-1
1	0	-1
1	0	-1

Convolution operation in CNN

- Input **X**: an image (2-D array) x 1 channel (grey)
- Convolution kernel/operator/filter (2-D array): **w**
- Feature map (2-D array of processed data): **s**
- Convolution operation in 2-D domains:

X * **w** = **s**

1	0	2	3	0
5	0	3	2	1
4	1	1	0	0
0	4	3	4	1
1	0	5	0	4

1	0	-1
1	0	-1
1	0	-1

	4			

= 1*1 + 0*0 + 2*-1
+ 5*1 + 0*0 + 3*-1
+ 4*1 + 1*0 + 1*-1

Convolution operation in CNN

- Input **X**: an image (2-D array) x 1 channel (grey)
- Convolution kernel/operator/filter (2-D array): **w**
- Feature map (2-D array of processed data): **s**
- Convolution operation in 2-D domains:

X * **w** = **s**

1	0	2	3	0
5	0	3	2	1
4	1	1	0	0
0	4	3	4	1
1	0	5	0	4

1	0	-1
1	0	-1
1	0	-1

	4	-4		

= 0*1 + 2*0 + 3*-1
+ 0*1 + 3*0 + 2*-1
+ 1*1 + 1*0 + 0*-1

Convolution operation in CNN

- Input **X**: an image (2-D array) x 1 channel (grey)
- Convolution kernel/operator/filter (2-D array): **w**
- Feature map (2-D array of processed data): **s**
- Convolution operation in 2-D domains:

X * **w** = **s**

1	0	2	3	0
5	0	3	2	1
4	1	1	0	0
0	4	3	4	1
1	0	5	0	4

1	0	-1
1	0	-1
1	0	-1

	4	-4	5	
	2	-1	5	
	-4	1	4	

Convolution operation in CNN

- Input **X**: an image (2-D array) x 1 channel (grey)
- Convolution kernel/operator/filter (2-D array): **w**
- Feature map (2-D array of processed data): **s**
- Convolution operation in 2-D domains:

X

0	**0**	**0**	**0**	**0**	**0**	**0**
0	1	0	2	3	0	**0**
0	5	0	3	2	1	**0**
0	4	1	1	0	0	**0**
0	0	4	3	4	1	**0**
0	1	0	5	0	4	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**

*****

w

1	0	-1
1	0	-1
1	0	-1

=

s

0	**1**	**-5**	**4**	**5**
-1	4	-4	5	**5**
-5	2	-1	5	**6**
-5	-4	1	4	**4**
-4	**-7**	**0**	**3**	**4**

Convolution operation in CNN

- Input **X**: an image (2-D array) x 1 channel (grey)
- Convolution kernel/operator/filter (2-D array): **w**
- Feature map (2-D array of processed data): **s**
- Convolution operation in 2-D domains:

X * **w** = **s**

"padding" (=1)

0	0	0	0	0	0	0
0	1	0	2	3	0	0
0	5	0	3	2	1	0
0	4	1	1	0	0	0
0	0	4	3	4	1	0
0	1	0	5	0	4	0
0	0	0	0	0	0	0

5 (+1) x 5 (+1)

1	0	-1
1	0	-1
1	0	-1

3 x 3

0	1	-5	4	5
-1	4	-4	5	5
-5	2	-1	5	6
-5	-4	1	4	4
-4	-7	0	3	4

5 x 5

"stride" (stride_x=stride_y=1)

Paddings and strides are hyperparameters like the number of layers and hidden neurons, etc

Convolution operation in CNN

- Input **X**: an image (2-D array) x 1 channel (grey)
- Convolution kernel/operator/filter (2-D array): **w**
- Feature map (2-D array of processed data): **s**
- Convolution operation in 2-D domains:

X * **w** = **s**

1	0	-1
1	0	-1
1	0	-1

Vertical edge detector (derivative)

Convolution operation in CNN

- Input **X**: an image (2-D array) x 1 channel (grey)
- Convolution kernel/operator/filter (2-D array): **w**
- Feature map (2-D array of processed data): **s**
- Convolution operation in 2-D domains:

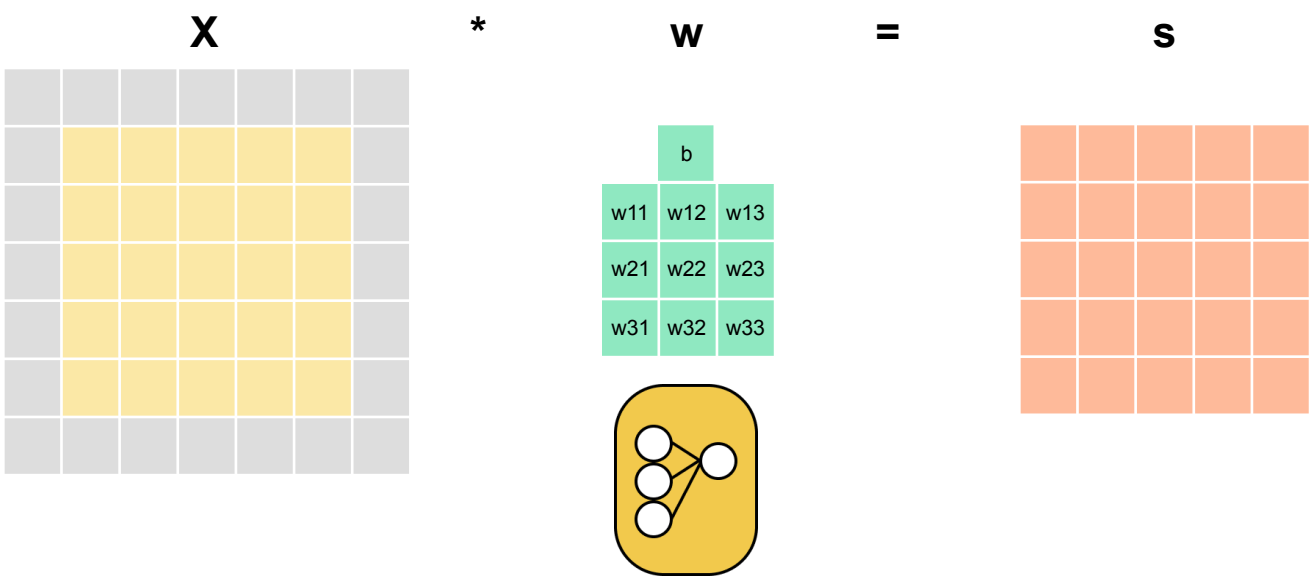
X * **w** = **s**

1	0	-1
1	0	-1
1	0	-1

1	1	1
0	0	0
-1	-1	-1

Convolution operation in CNN

- Input **X**: an image (2-D array) x 1 channel (grey)
- Convolution kernel/operator/filter (2-D array): **w**
- Feature map (2-D array of processed data): **s**
- Convolution operation in 2-D domains:

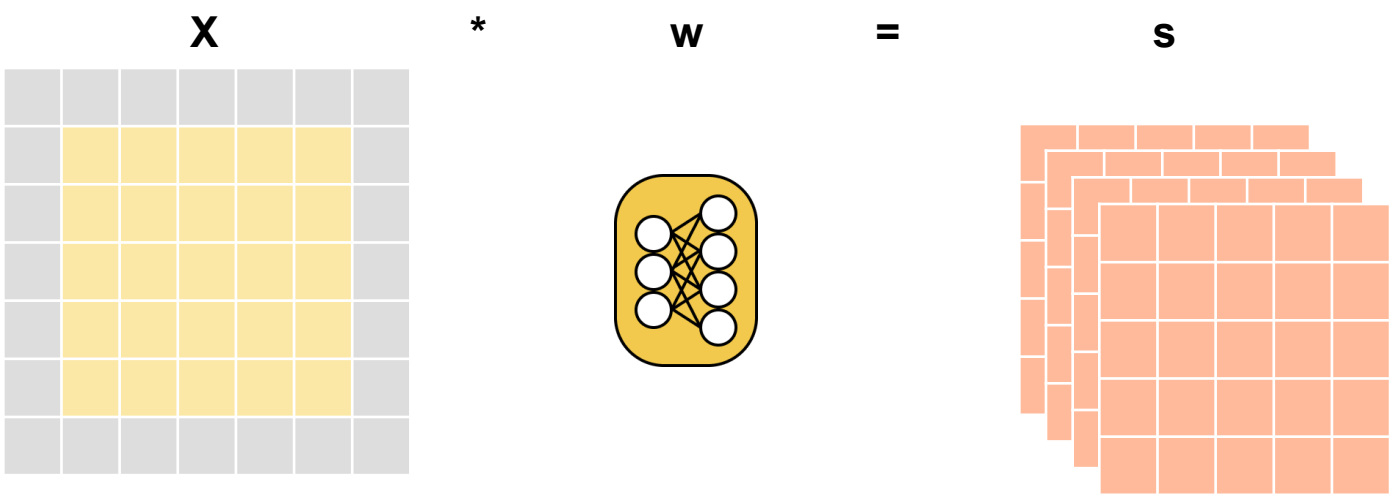


FCNN: weight sharing

no hidden layers, just input and output

Convolution operation in CNN

- Input **X**: an image (2-D array) x 1 channel (grey)
- Convolution kernel/operator/filter (2-D array): **w**
- Feature map (2-D array of processed data): **s**
- Convolution operation in 2-D domains:



Pooling operation in CNN

0	1	-5	4	5
-1	4	-4	5	5
-5	2	-1	5	6
-5	-4	1	4	4
-4	-7	0	3	4

MaxPool
(2 x 2, stride=1)

4		

Pooling operation in CNN

0	1	-5	4	5
-1	4	-4	5	5
-5	2	-1	5	6
-5	-4	1	4	4
-4	-7	0	3	4

MaxPool
(2 x 2, stride=1)

4	-5		

Pooling operation in CNN

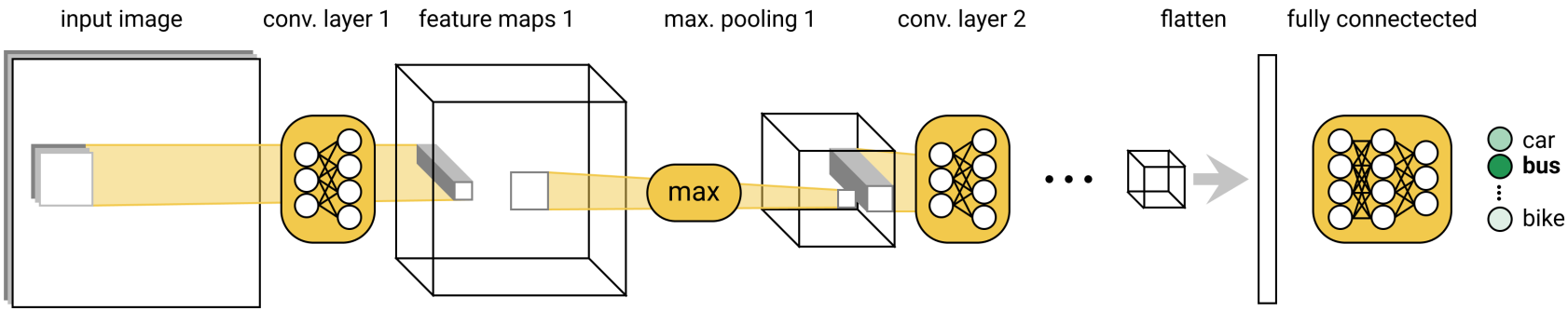
0	1	-5	4	5
-1	4	-4	5	5
-5	2	-1	5	6
-5	-4	1	4	4
-4	-7	0	3	4

MaxPool
(2 x 2, stride=1)

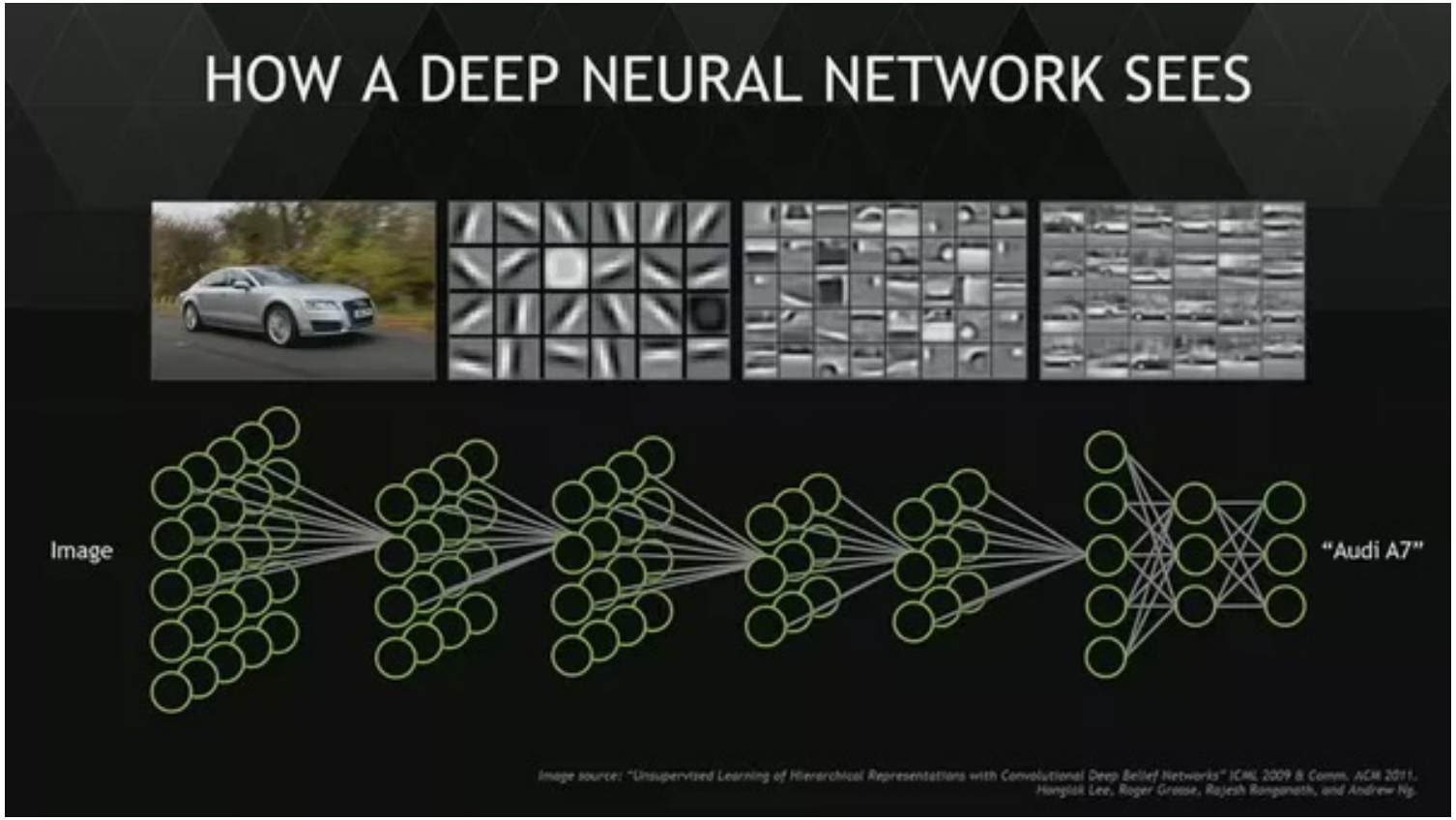
4	-5	5	5
-5	4	5	6
-5	-4	5	6
-7	-7	4	4

- Common pooling operations:
  - **Max pooling:** reports the maximum output within a rectangular neighbourhood
  - **Average pooling:** reports the average output of a rectangular neighbourhood (possibly weighted by the distance from the central pixel)

Typical CNN architecture



Try to count the number of hyperparameters in this stuff… (a lot of them!)

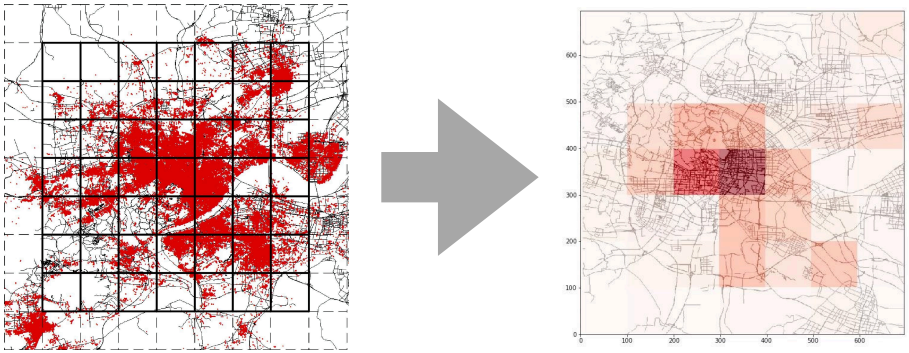


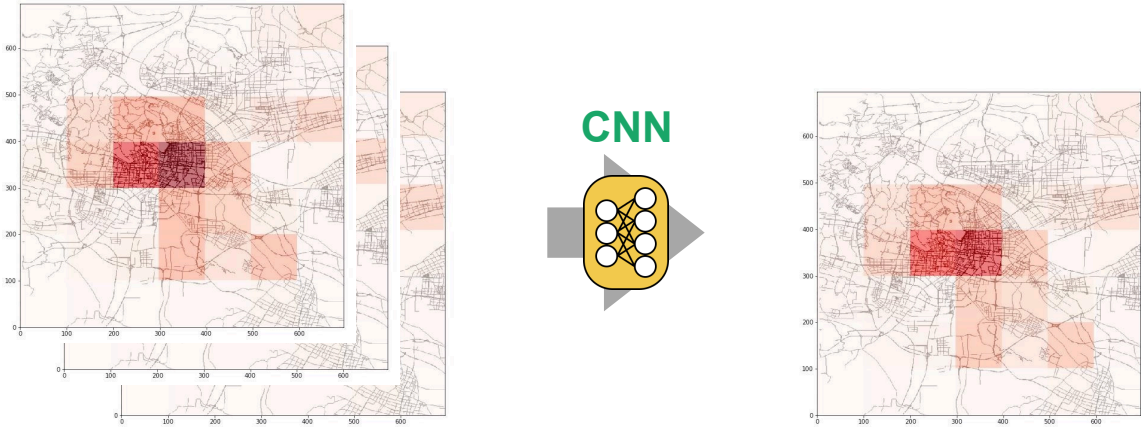
http://web.eecs.umich.edu/~honglak/cacm2011-researchHighlights-convDBN.pdf

CNN applications

- Images
- Time series (1D convolutions)
- Video, spatio-temporal series (3D convolutions)
- Text (1D convolutions)
- …

Example: Spatio-temporal series





t-T, …, t-2, t-1, t

t+1

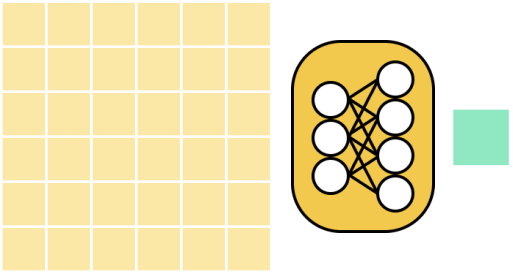
Recurrent Neural Network (RNN)

C	U	R	R	E	N	?
1	0	0	0	0	0	
0	1	0	0	0	0	
0	0	1	1	0	0	
0	0	0	0	1	0	
0	0	0	0	0	1	
0	0	0	0	0	0	

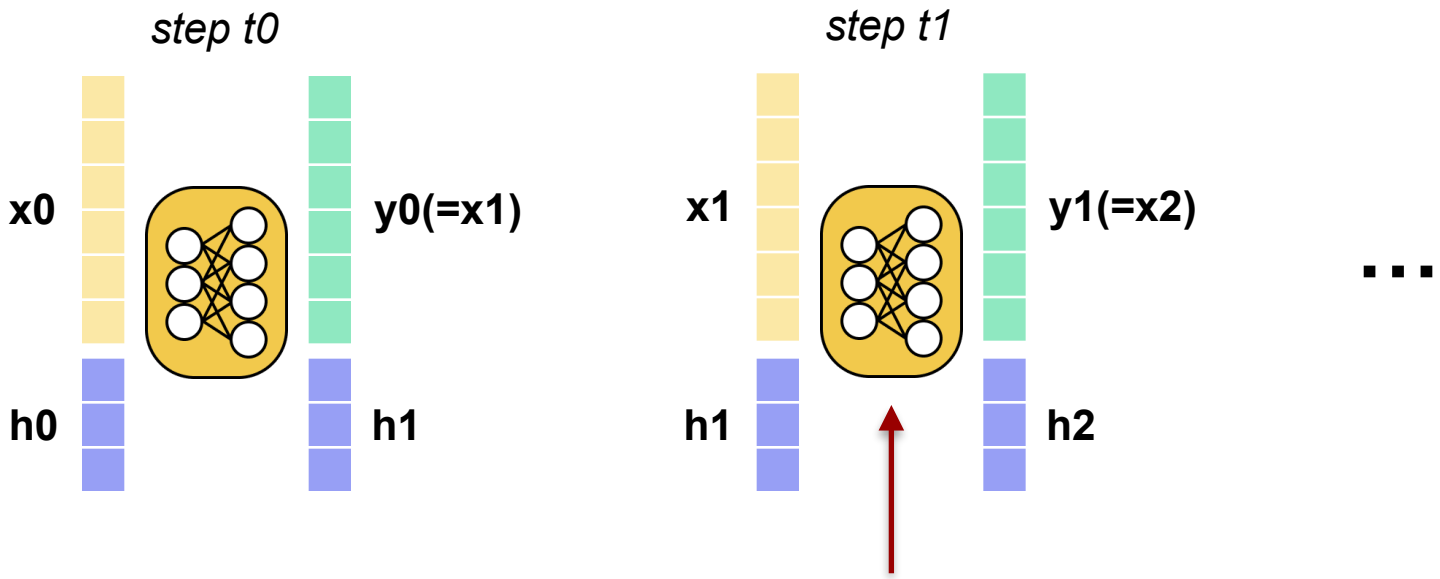
p(x_t | x_t-1, x_t-2, …)

FCNN: moving window

yes… but we can do better!

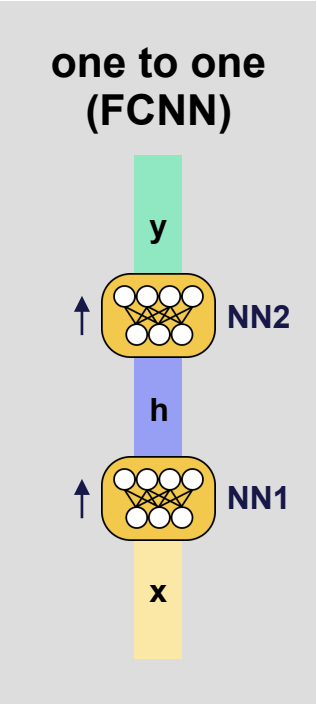


Recurrent Neural Network (RNN)



The same network
(“weight sharing” again!)

RNN variations

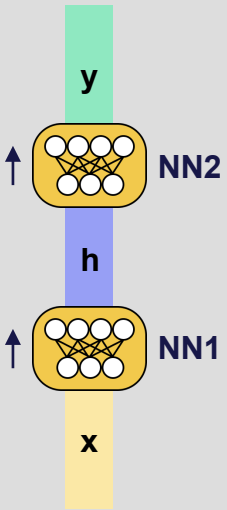


Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output
(e.g. image classification)

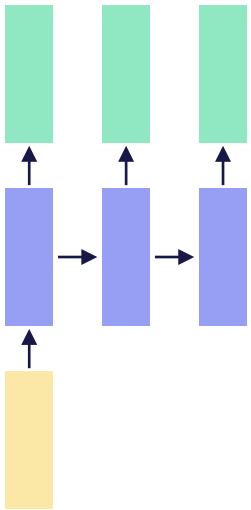
Andrej Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks"
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN variations

one to one (FCNN)



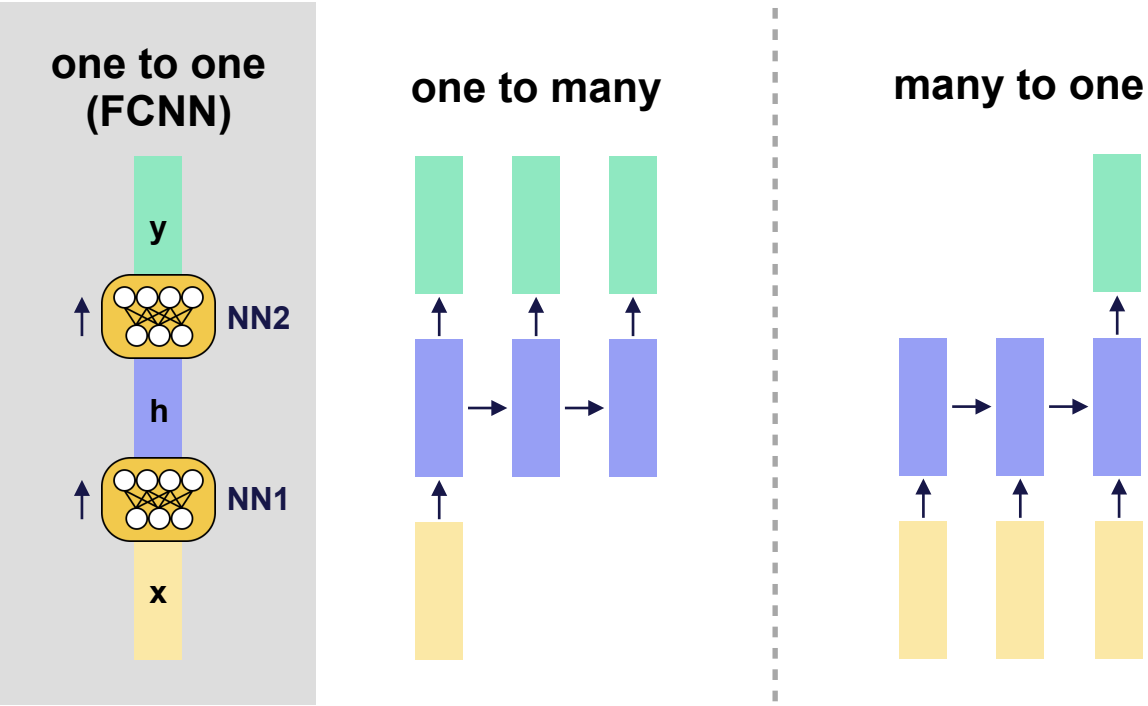
one to many



Sequence output
(e.g. image captioning takes an image and outputs a sentence of words)

Andrej Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks"
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

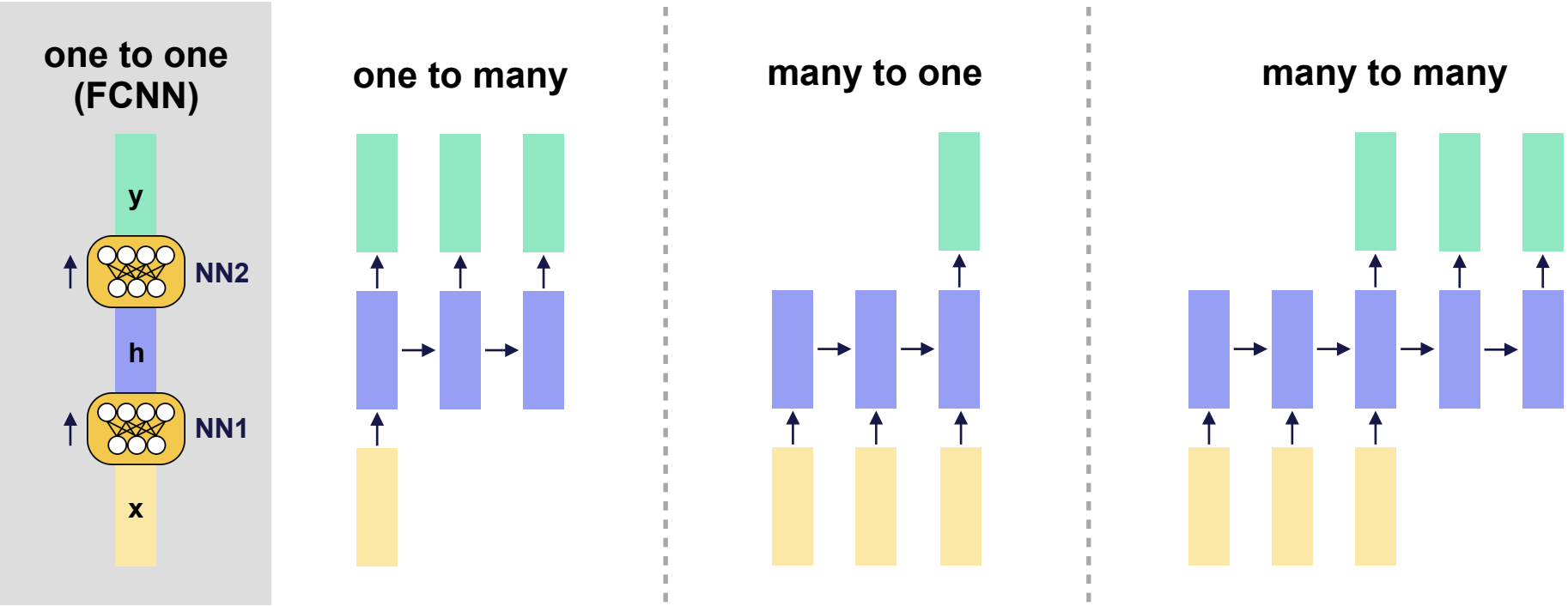
RNN variations



Sequence input
(e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment)

Andrej Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks"
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

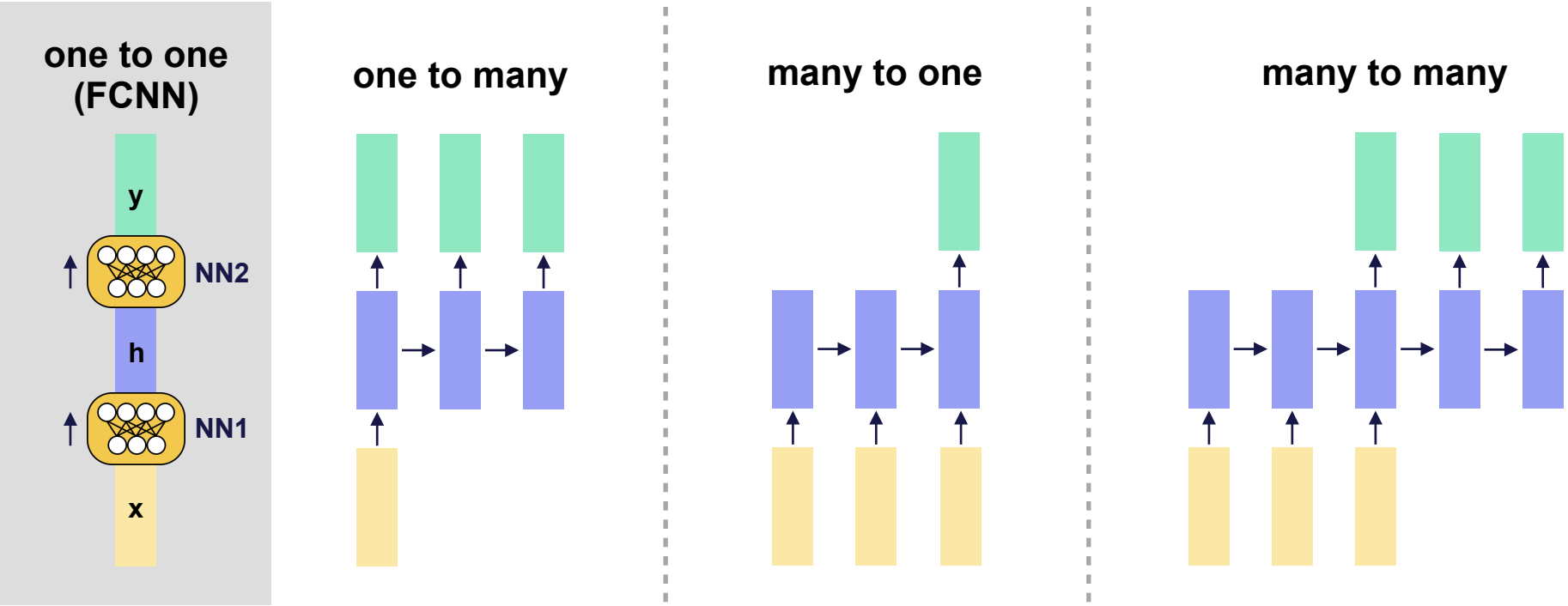
RNN variations



Sequence input and sequence output
(e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French)

Andrej Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks"
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN variations

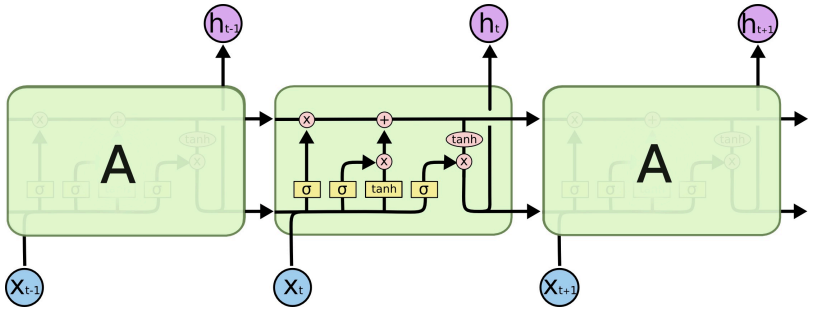


😞 **Vanishing gradient problem** 😞

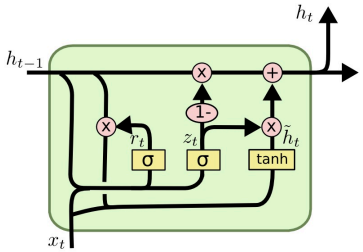
Andrej Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks"
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN modifications

Long Short-Term Memory Unit (LSTM)



Gated Recurrent Unit (GRU)



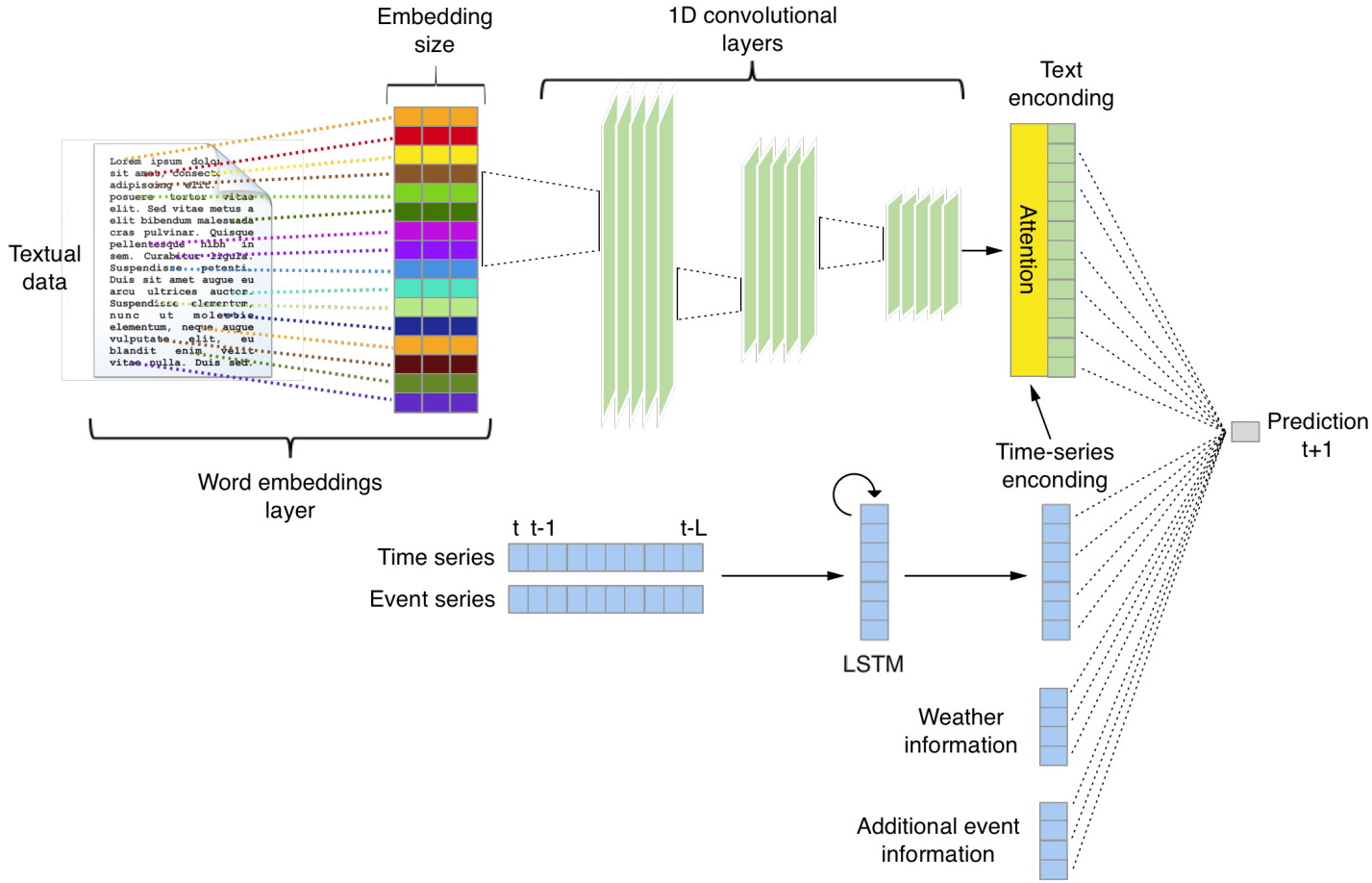
“Skip connections” idea: f(x)=r(x)+x
(also used in Residual Neural Networks)

“Understanding LSTM Networks”
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN applications

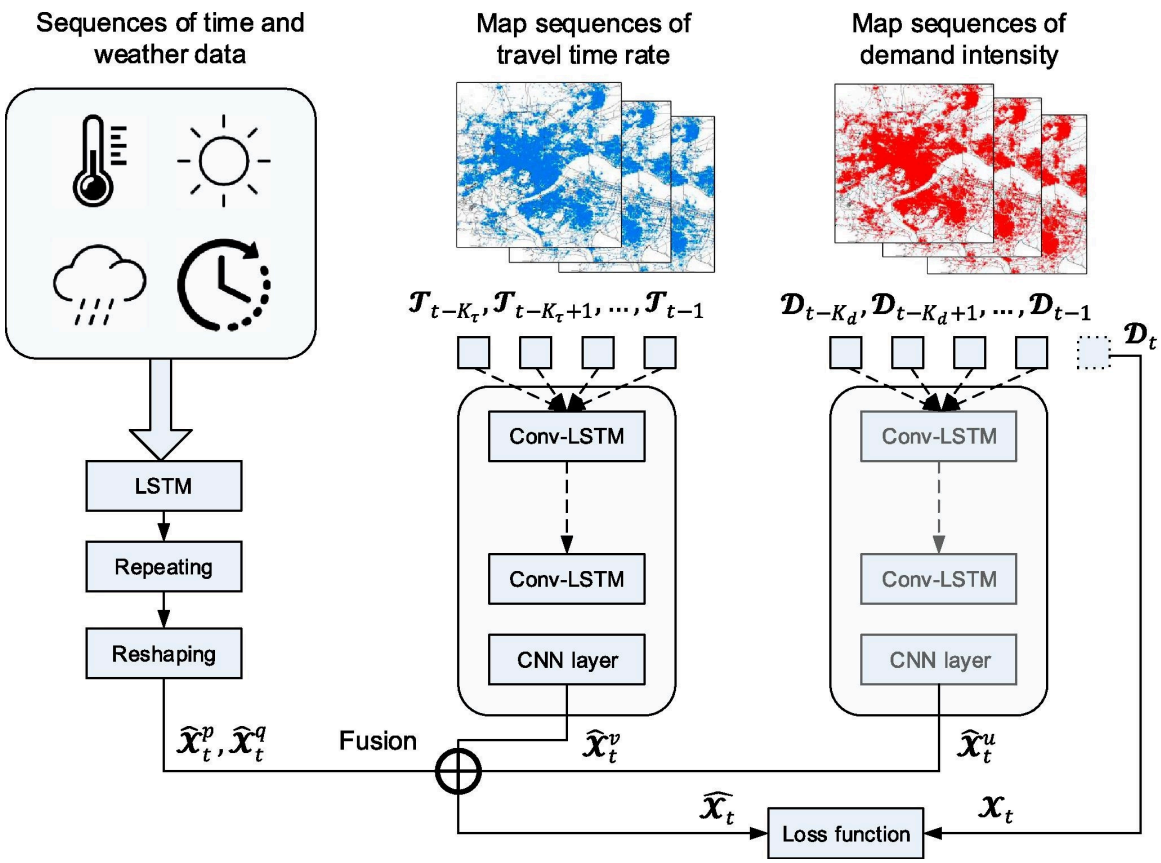
- The same as CNN!
- RNN better works for long and varying-length sequences, like text and time series

Deep NN models are like Lego blocks



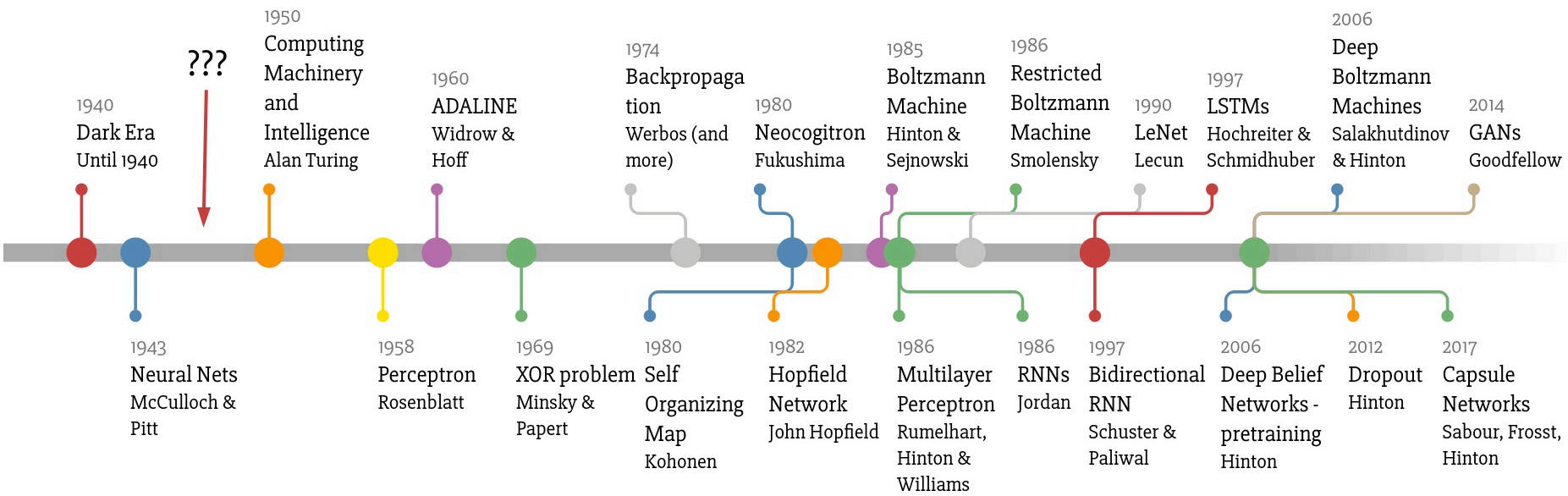
Rodrigues, F. and Markou, I. and Pereira, F. C.
Combining time-series and textual data for taxi demand prediction in event areas: a deep learning approach in Information Fusion, Elsevier, 2018

Deep NN models are like Lego blocks



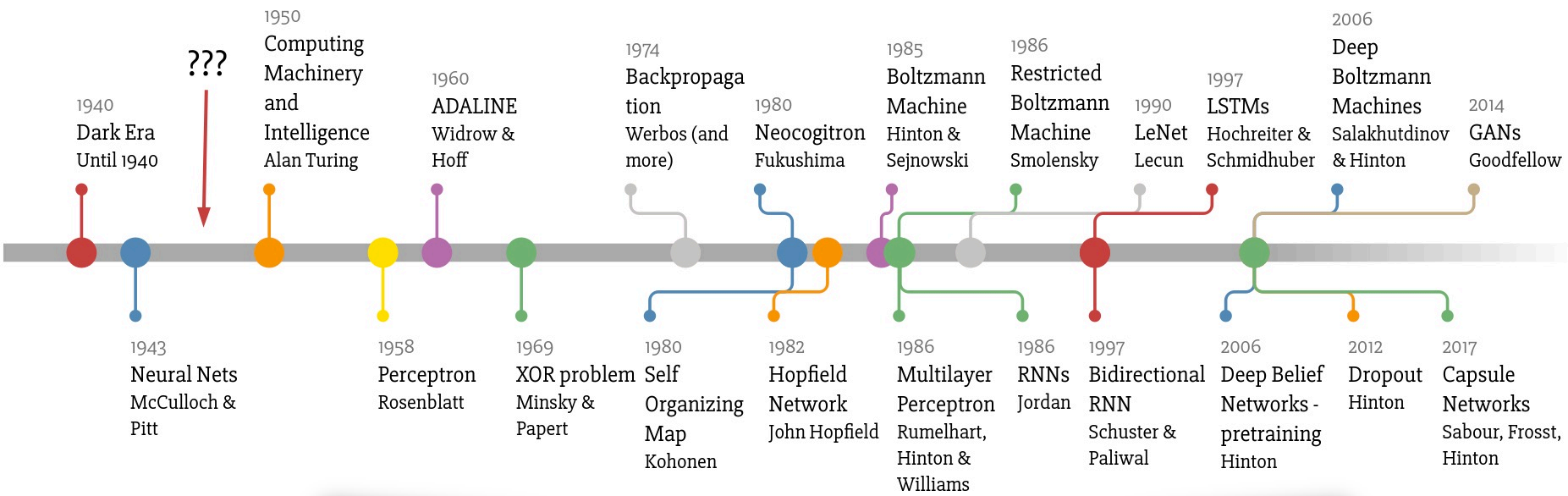
Jintao Ke, Hongyu Zheng, Hai Yang, Xiqun (Michael) Chen
Short-term forecasting of passenger demand under on-demand ride services: A spatio-temporal deep learning approach
Transportation Research Part C: Emerging Technologies (2017)

Deep Learning timeline



Made by Favio Vázquez

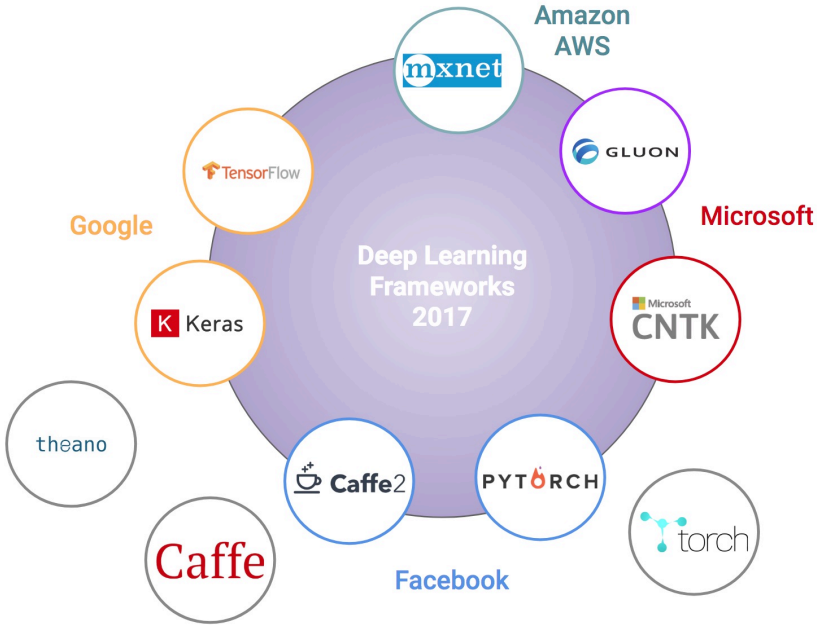
Deep Learning timeline



Made by Favio Vázquez

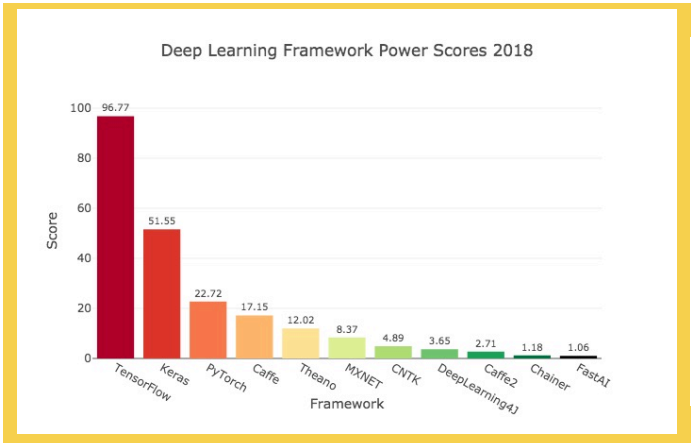
Reason for the current success of DL:
Sufficient amount of data and computational power (GPUs) for CNNs and RNNs (but not for FCNNs yet)

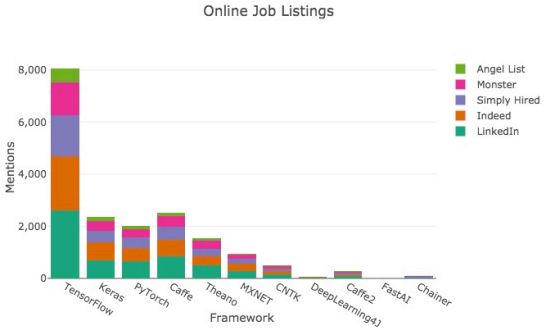
Deep Learning frameworks

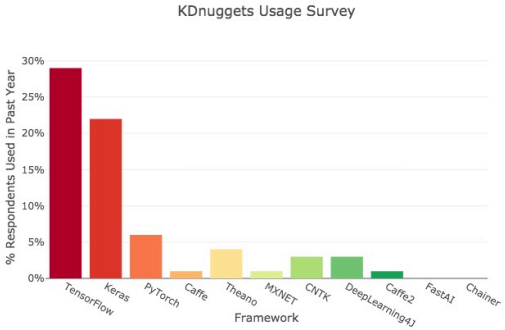


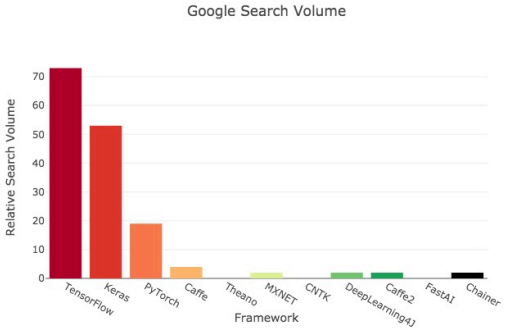
https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

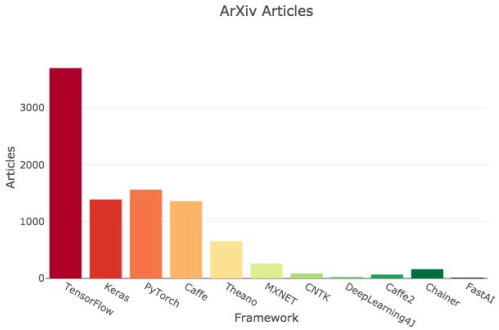
Deep Learning frameworks

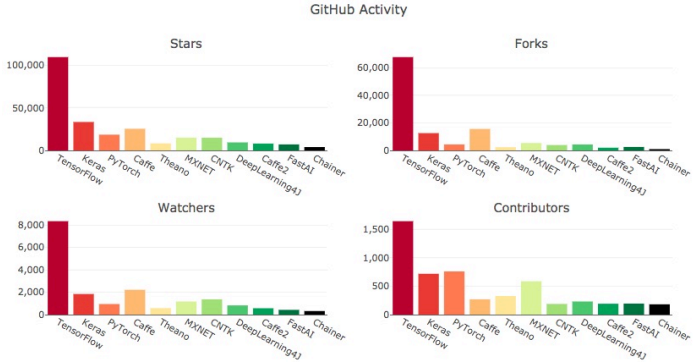








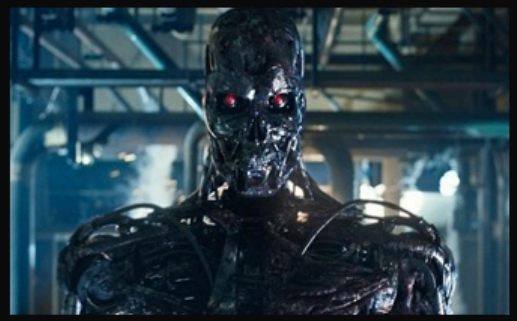




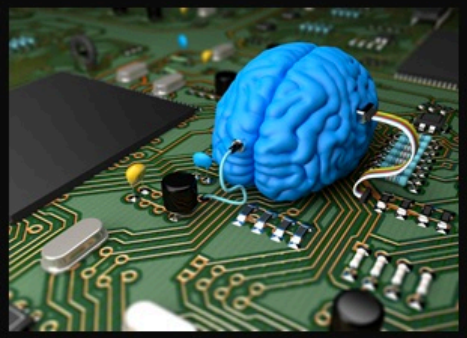
*** In 2019, PyTorch become more popular than TF/Keras in academia**

https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Deep Learning

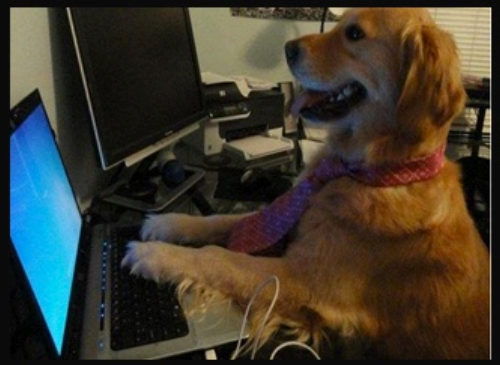


What society thinks I do



What my friends think I do

What other computer scientists think I do



What mathematicians think I do

What I think I do

```
import keras
```

```
Using TensorFlow backend.
```

What I actually do

Exercises

- “Part 1 - Keras Intro and CNN.ipynb”
- “Part 2 - RNN.ipynb”
- [Extra, optional] “Predictive Maintenance.ipynb” (from Intro to BA 2019)

Recommended reading

- *Mobility Patterns, Big Data and Transport Analytics,* Chapter 2. Machine Learning Fundamentals
- *Neural Networks and Deep Learning*, Chapter 1 http://neuralnetworksanddeeplearning.com/chap1.html
- Keras cheat sheet
- [highly optional] DS, ML and DL in production and at scale:
  - *“The Definitive Guide to Machine Learning Platforms”*
  - *“Kubernetes for Machine Learning, Deep Learning & AI”*