Africa TechUp Tour

CLASSE VIRTUELLE N°147

Thématique :

Initiation au Deep Learning

SÈGBÉDJI R.T.JUNIOR GOUBALAN, PH.D

Présentation

Sègbédji Junior **GOUBALAN**
Co-fondateur & **CSO** AdjibolaTech

ADJIBOLA
Le bien-être en Afrique grâce aux systèmes intelligents

- Ph.D en IA de Paris Saclay/Polytechnique Montréal (2016)
- Qualif. Maître de conférences CNU 27 & 61 (2017-21)
- Data Scientist chez General Electric Healthcare (2017)
- Data Scientist chez Safran Morpho (2017-2018)
- Data Scientist & Head IA chez Philips Healthcare (2018-)

Auteur de 3 brevets et de 4 secrets commerciaux

Implication dans l'écosystème IA au Bénin/Afrique

iSHEERO
INITIATIVE DES ACTEURS BÉNINOIS DE L'IA

Plan

- Introduction
- Historique et définition du *Deep Learning*
- Concepts mathématiques fondamentaux
- Quelques astuces pour l'apprentissage
- TP & Quelques ressources

Deep Learning for generative models

TEXT PROMPT

an armchair in the shape of an avocado [...]

AI-GENERATED IMAGES

View more or edit prompt

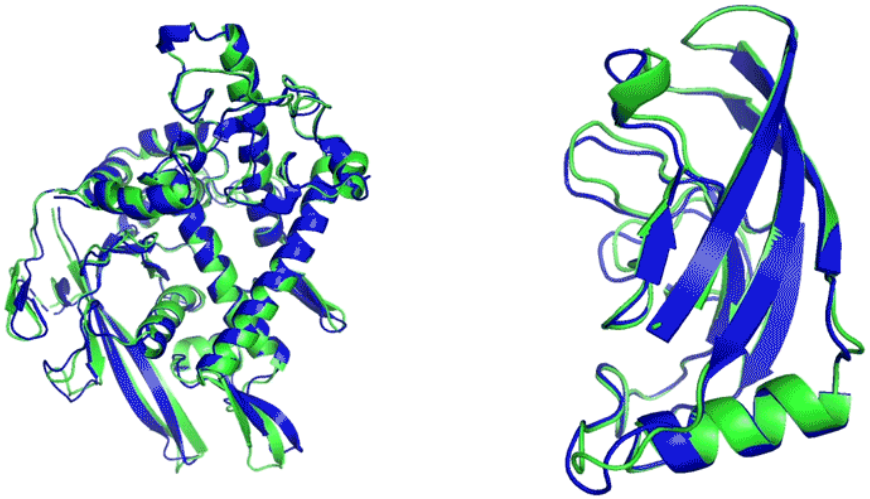
TEXT PROMPT

a store front that has the word 'openai' written on it [...]

AI-GENERATED IMAGES

View more or edit prompt

Deep Learning for genomics



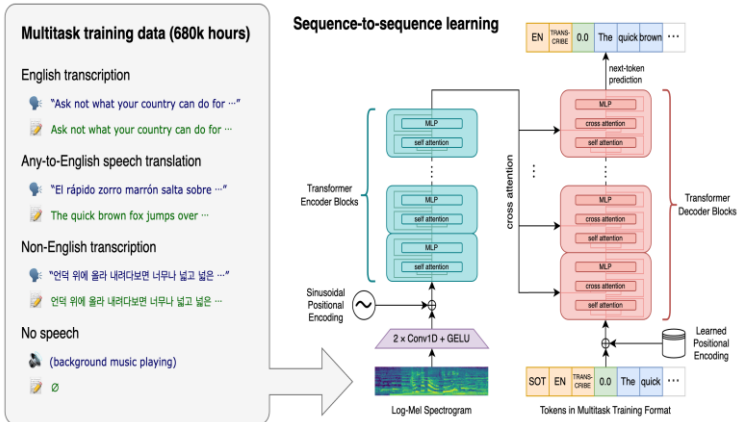
T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)

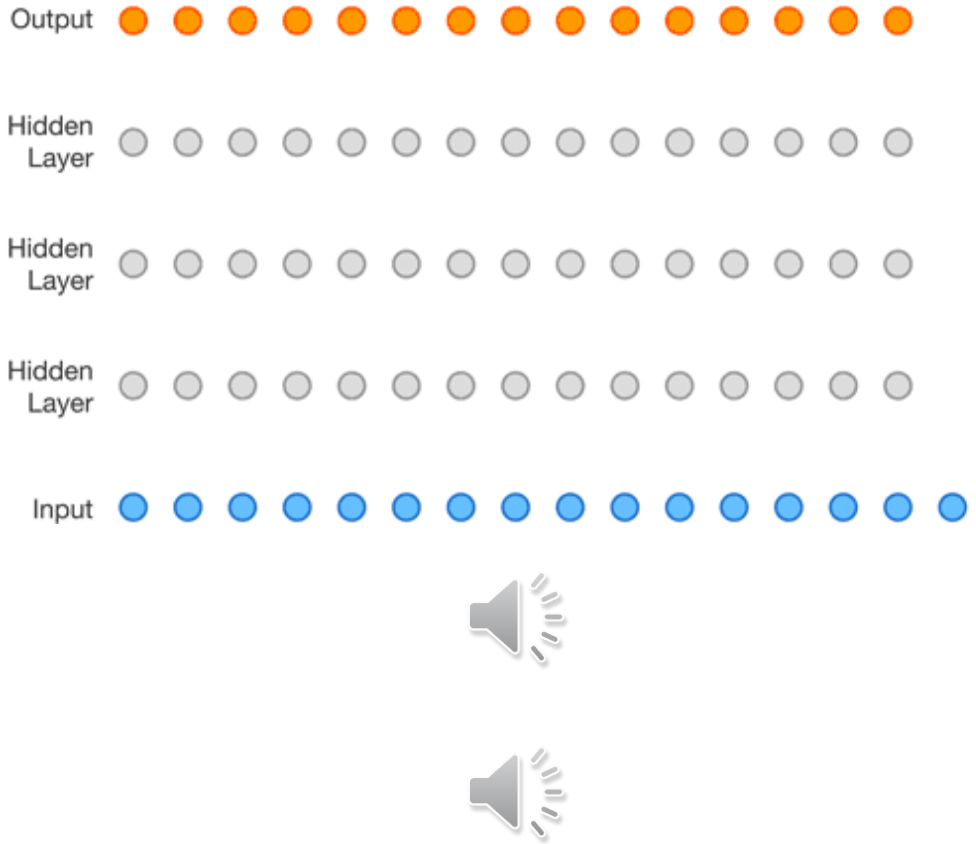
T1049 / 6y4f
93.3 GDT
(adhesin tip)

Experimental result
Computational prediction

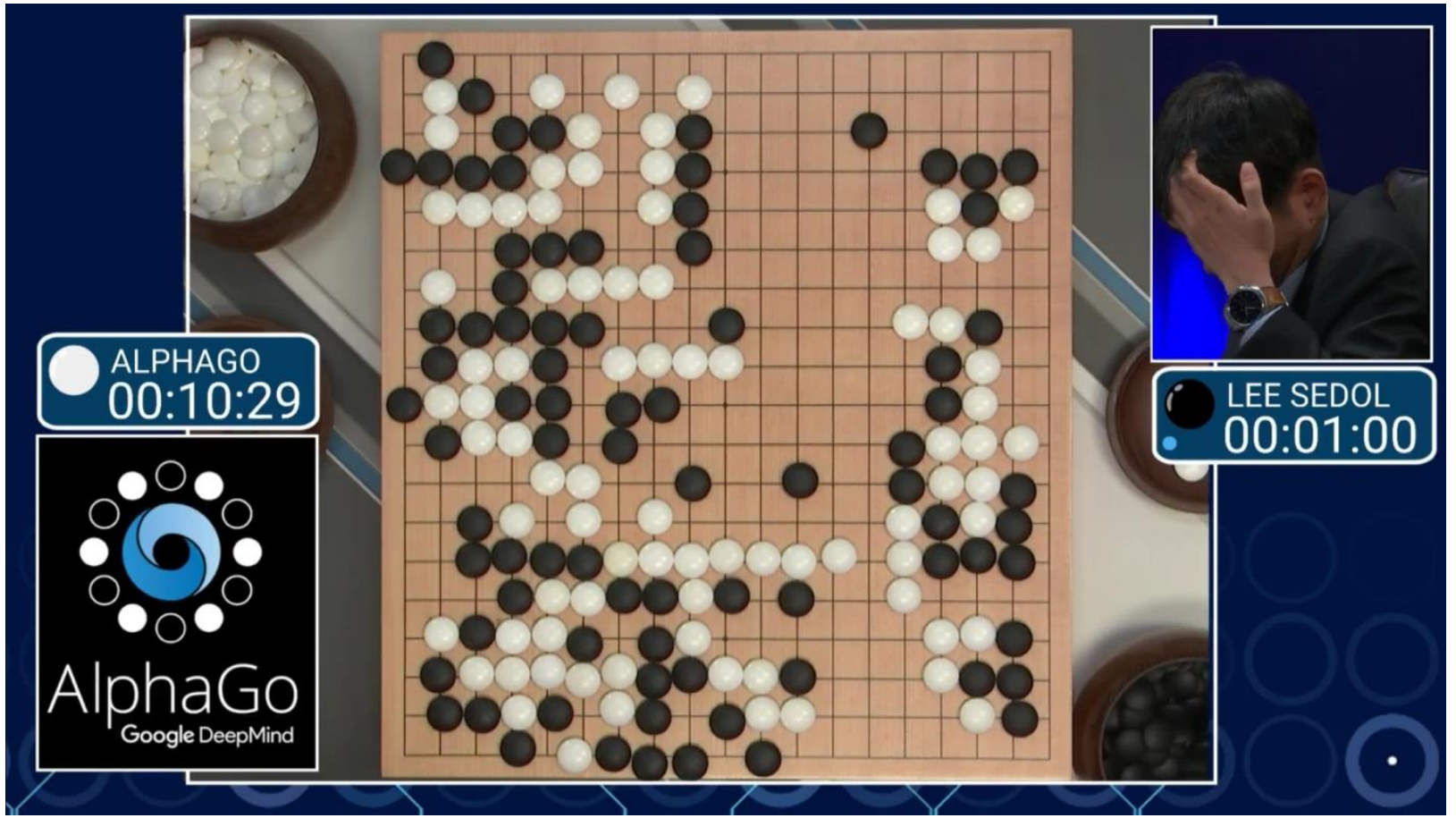
Deep Learning for Speech/Text

OpenAI Whisper

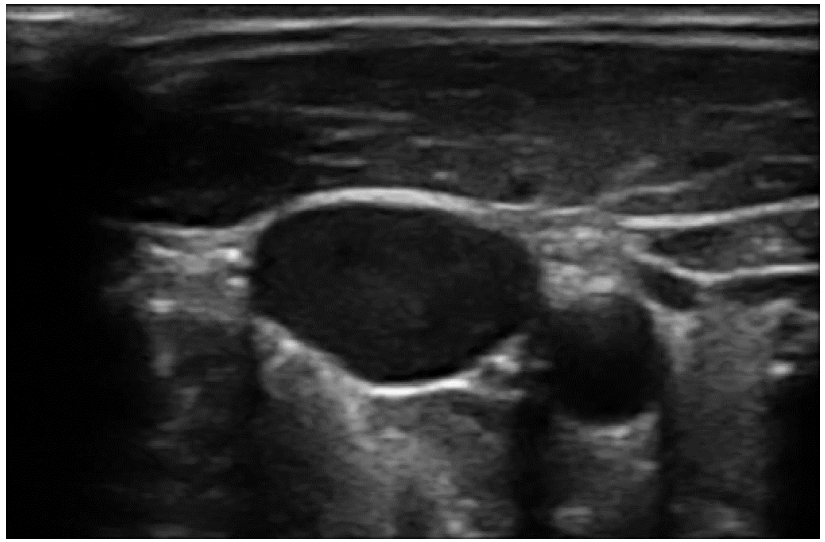


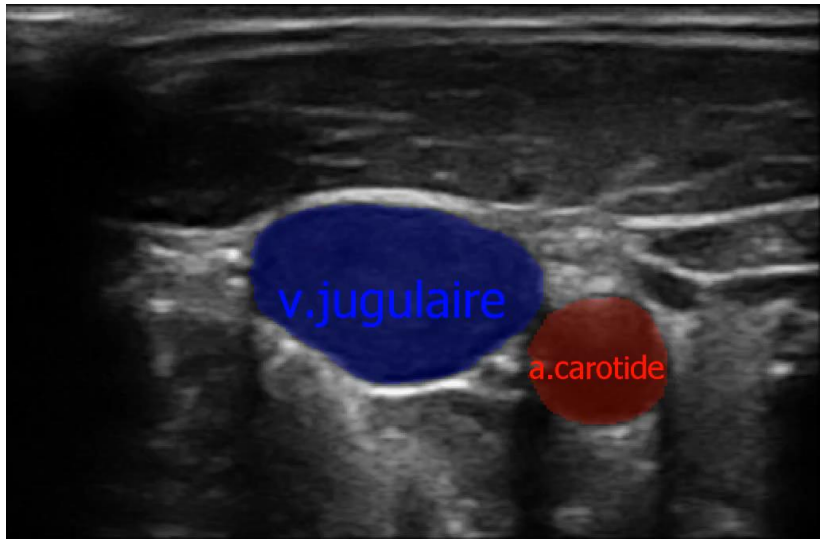


Deep Learning for games

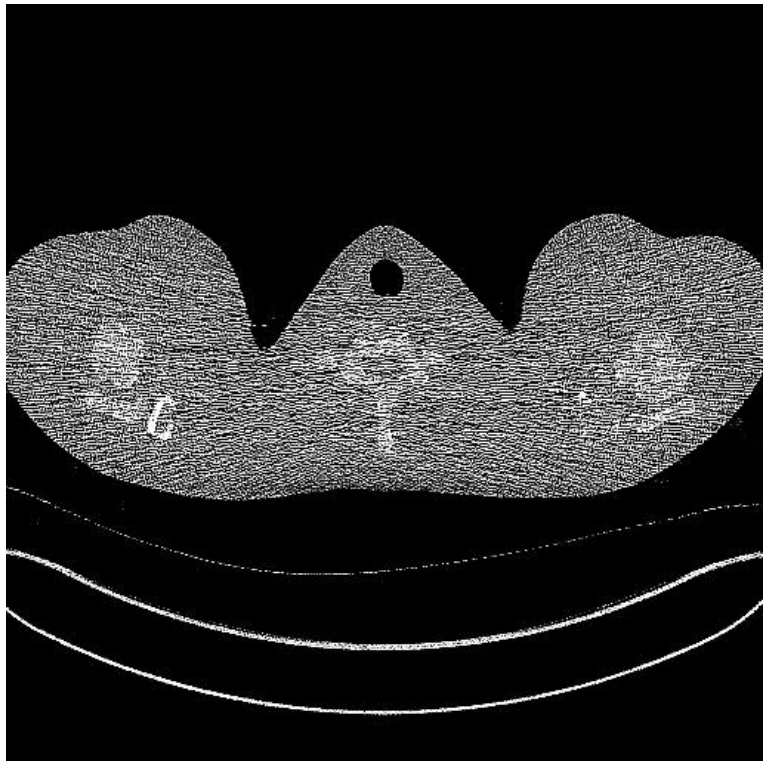


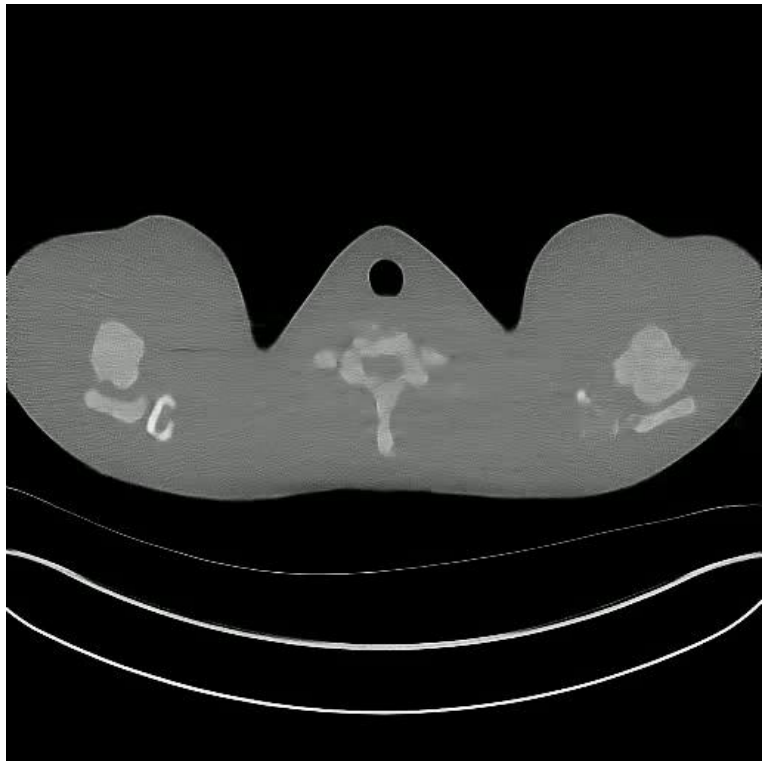
Deep Learning for Vision (1/5)

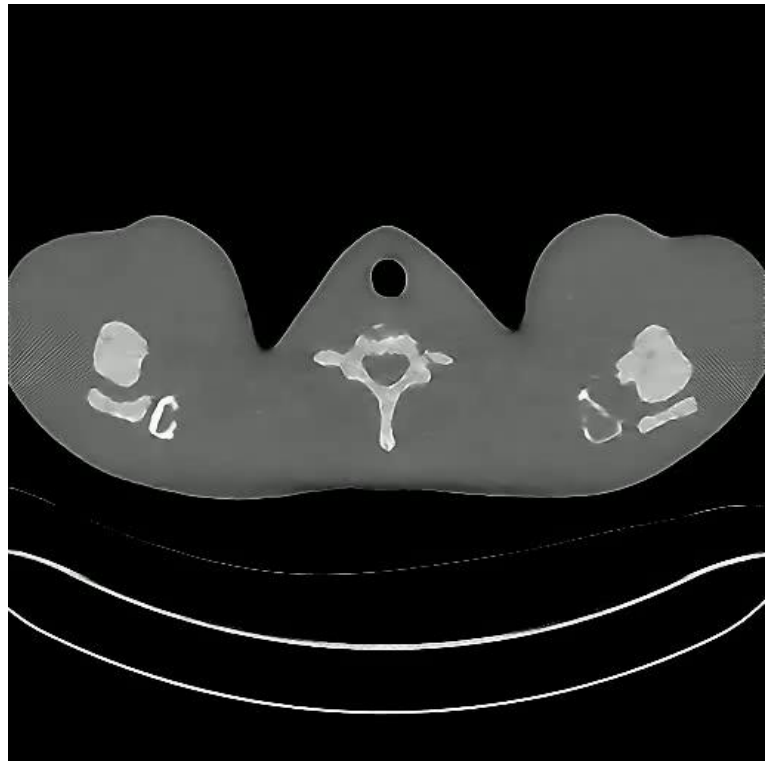




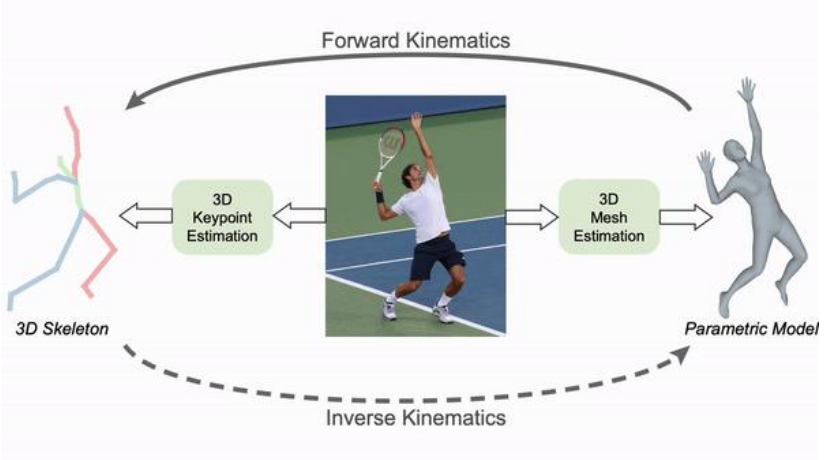
Deep Learning for Vision (2/5)

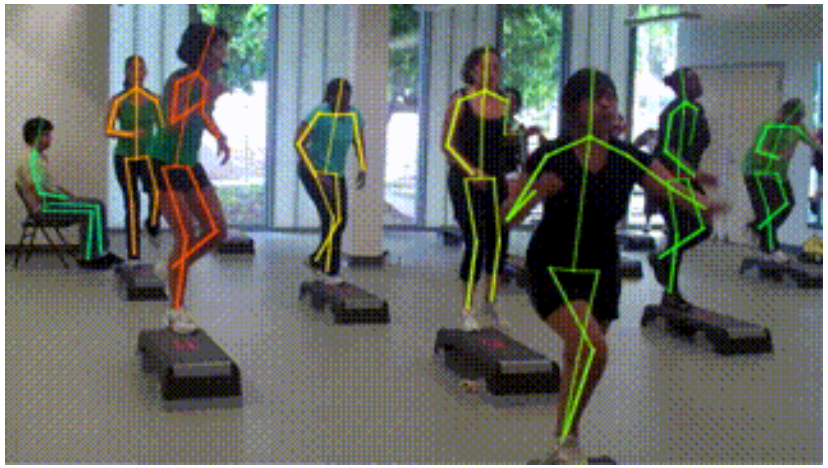


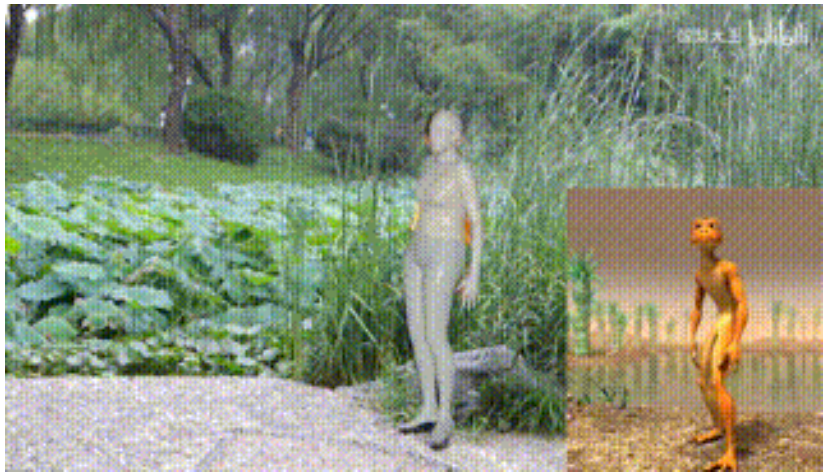


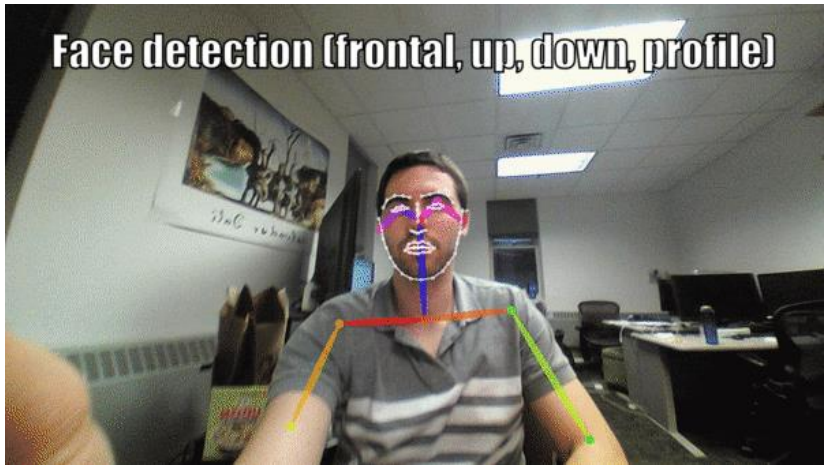


Deep Learning for Vision (3/5)









Deep Learning for Vision (4/5)

Marguerite

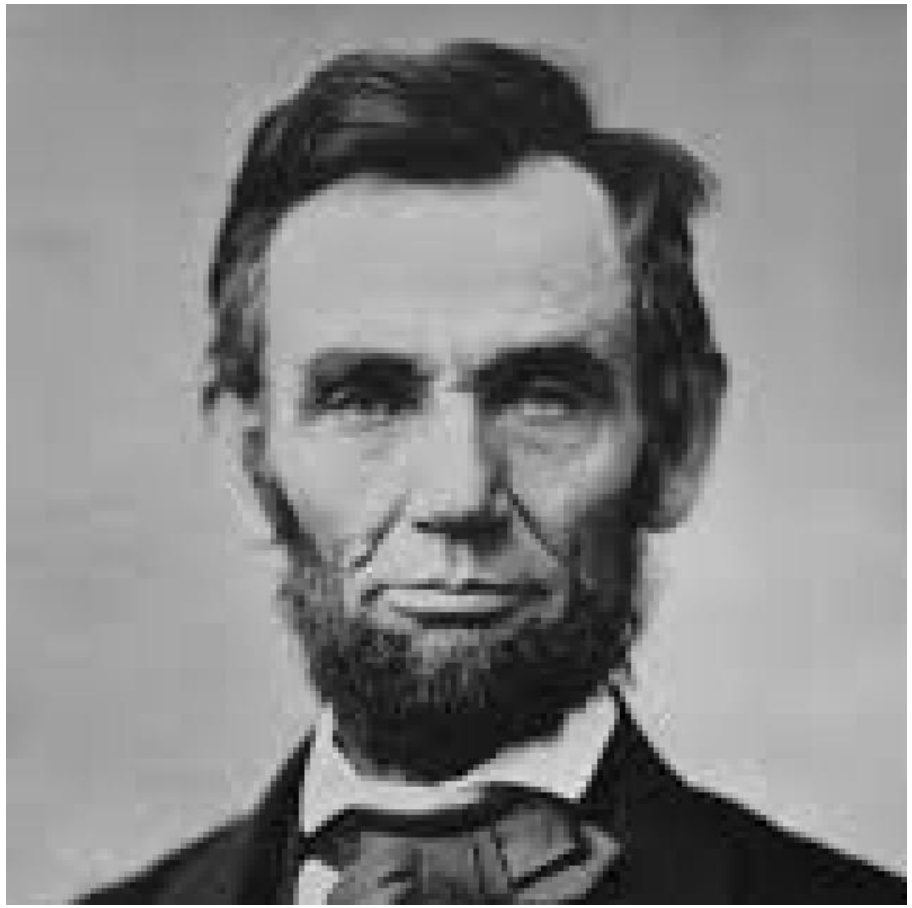
Pissenlit

Rose

Tournesol

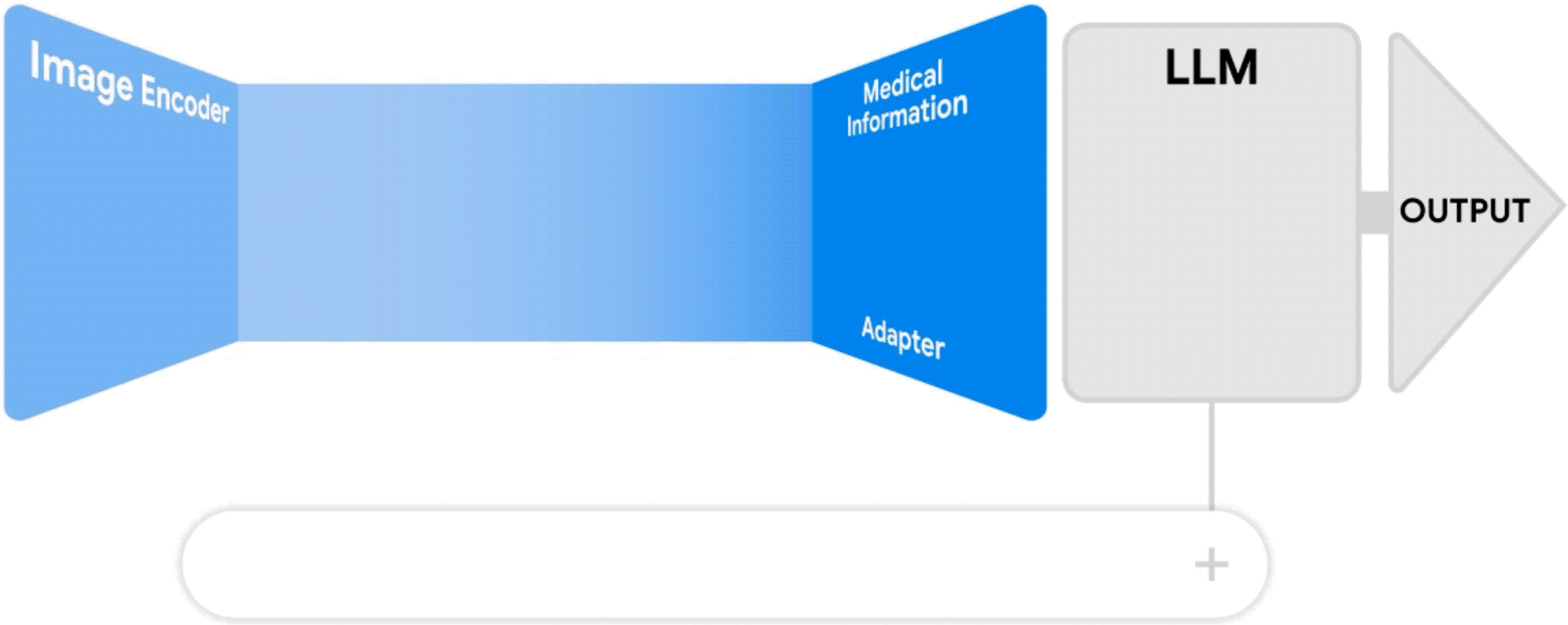
Tulipe

Deep Learning for Vision (5/5)

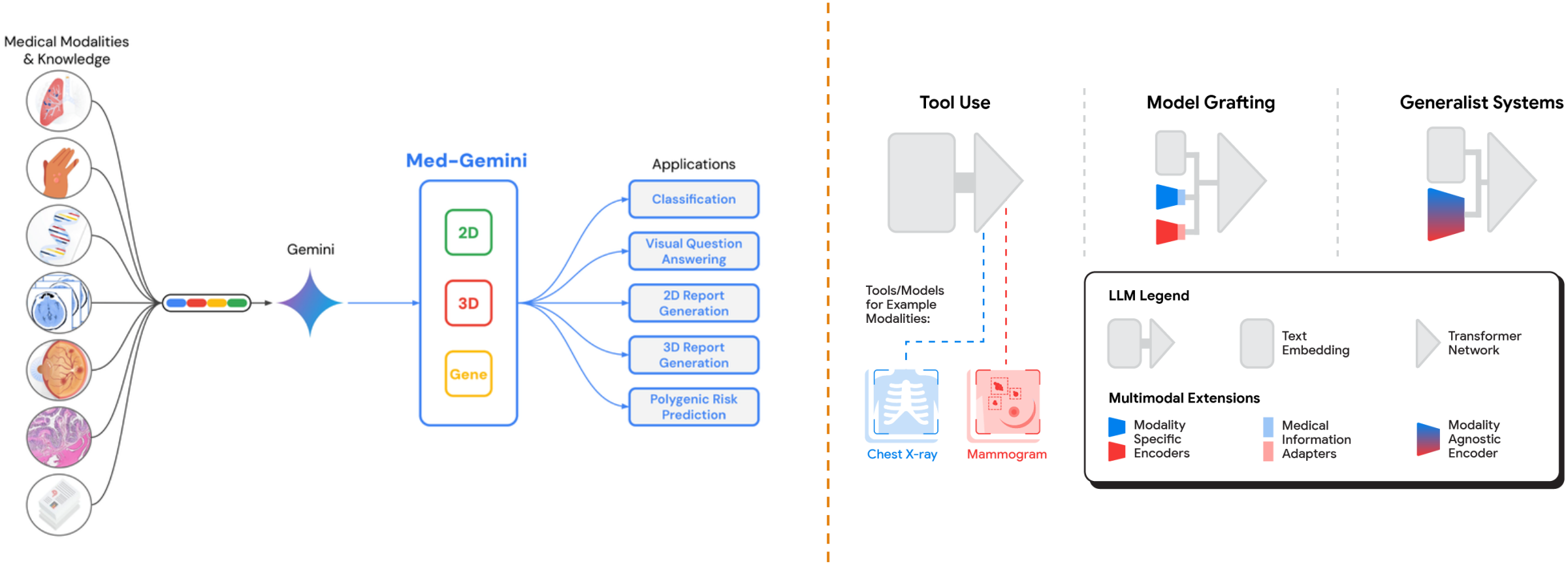


Deep Learning for Vision & NLP (1/3)

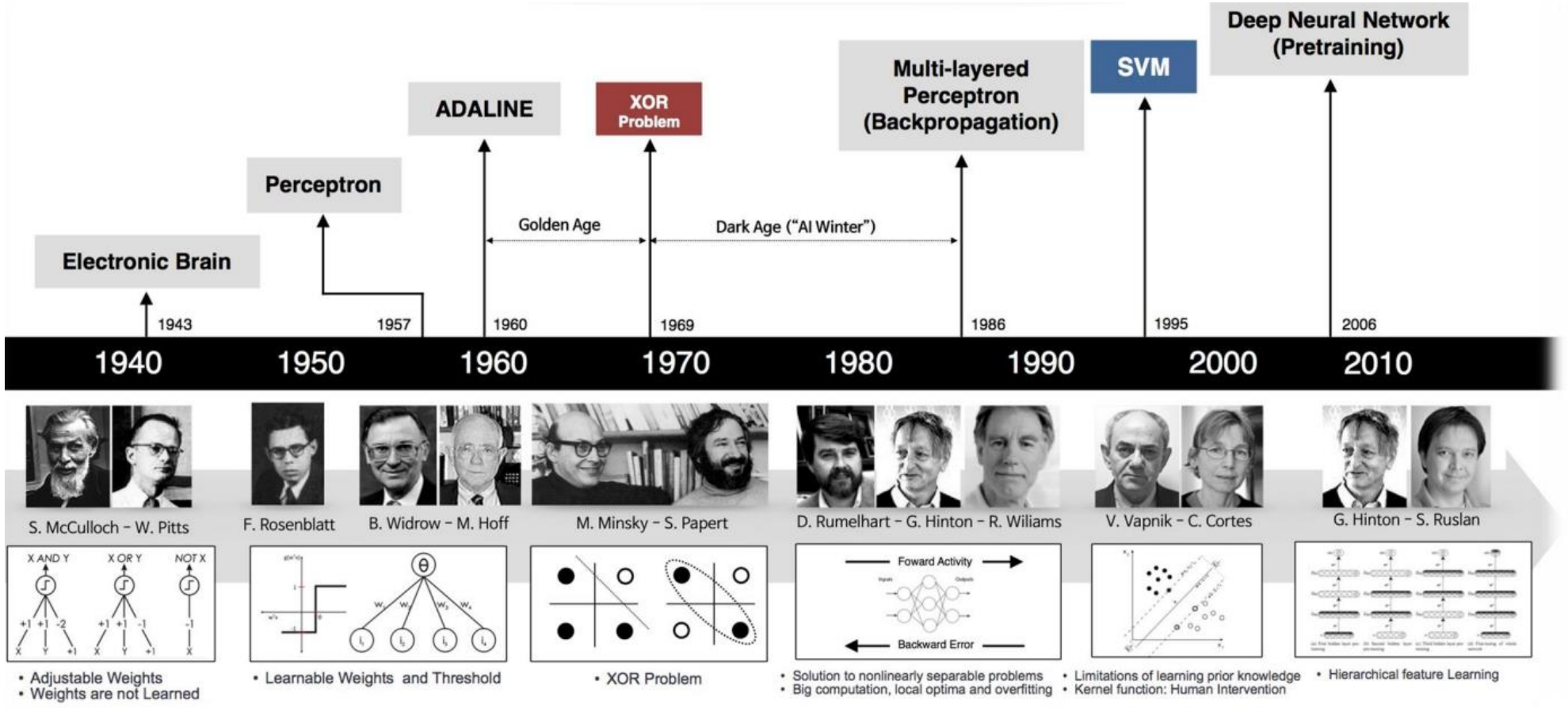
Deep Learning for Vision & NLP (2/3)



Deep Learning for Vision & NLP (3/3)

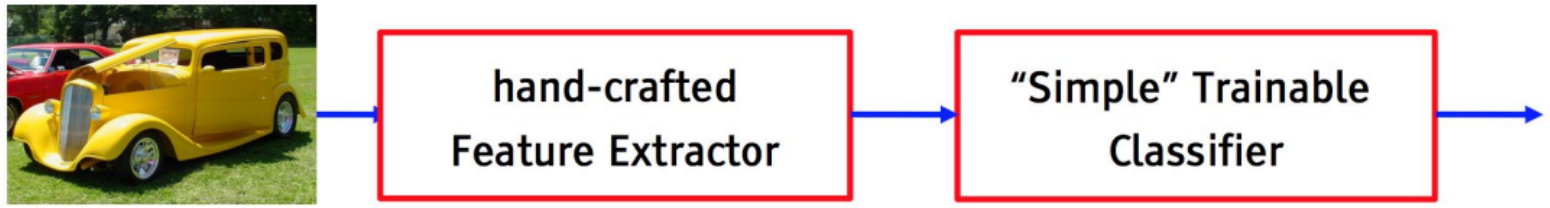


A brief history



How we did before ? (1/8)

- The traditional model of pattern recognition (since the late 50's)
  - Fixed/engineered features (or fixed kernel) + trainable classifier



From Y. LeCun's Slides

How we did before ? (2/8)

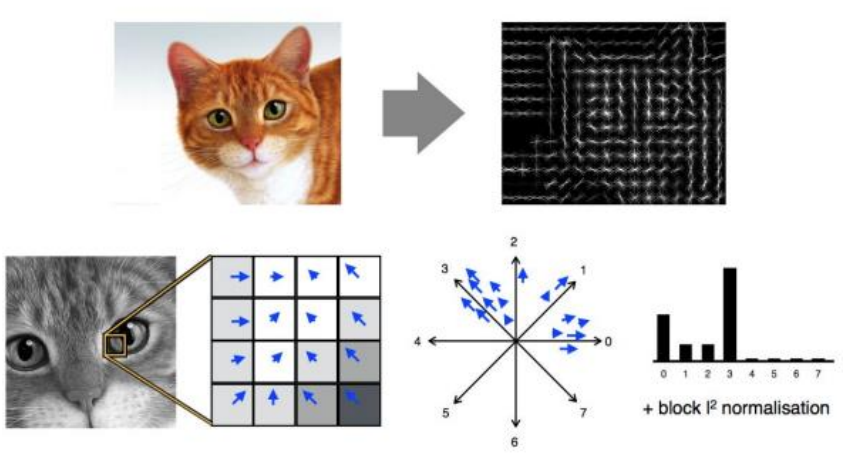
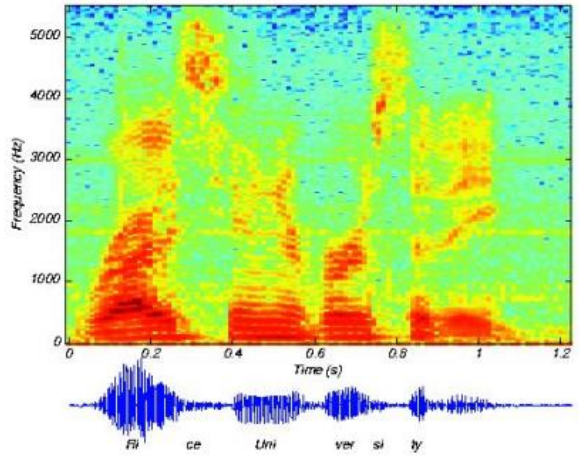


Image: HoG



Audio: Spectrogram

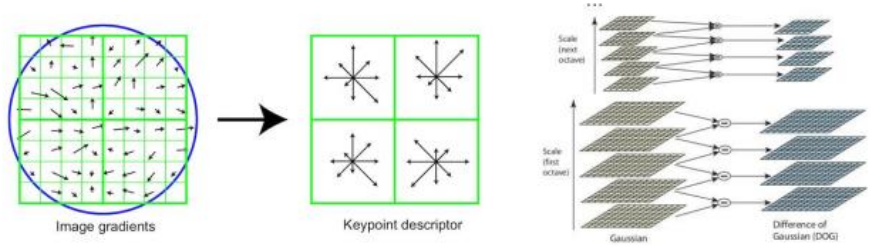
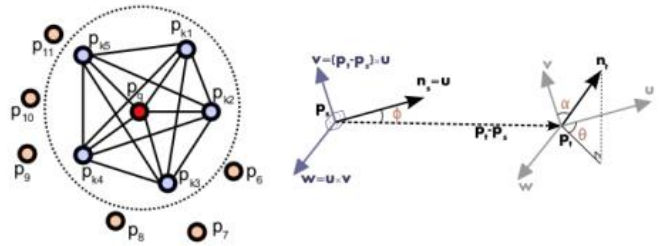
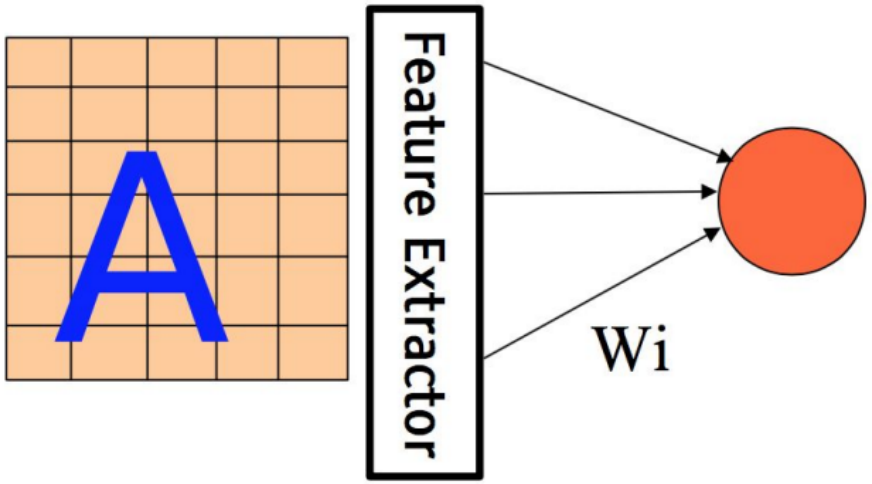


Image: SIFT



Point Cloud: PFH

How we did before ? (3/8)



Linear Regression
SVM
Decision Trees
Random Forest
...

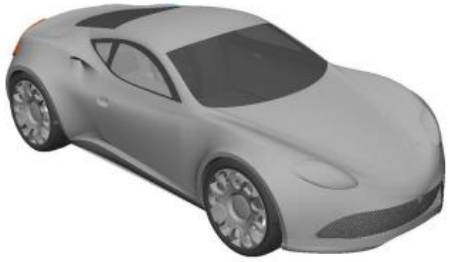
$$y = sign\left(\sum_{i=1}^{N} W_i F_i(X) + b\right)$$

From Y. LeCun's Slides

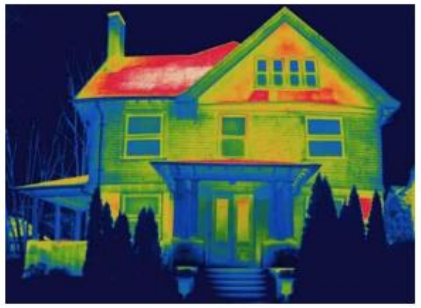
How we did before ? (4/8)

Image

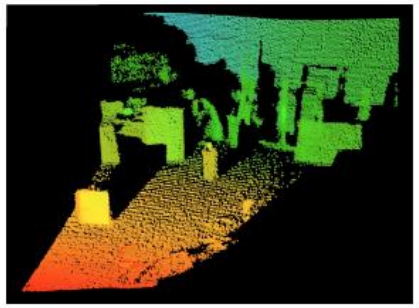
Video



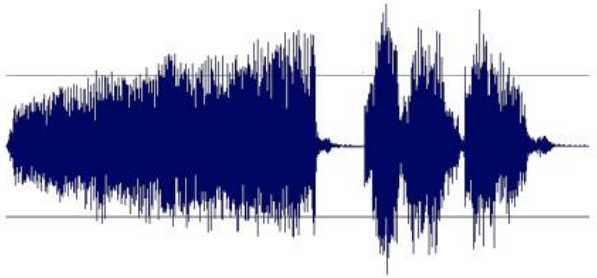
3D CAD Model



Thermal Infrared



Depth Scan

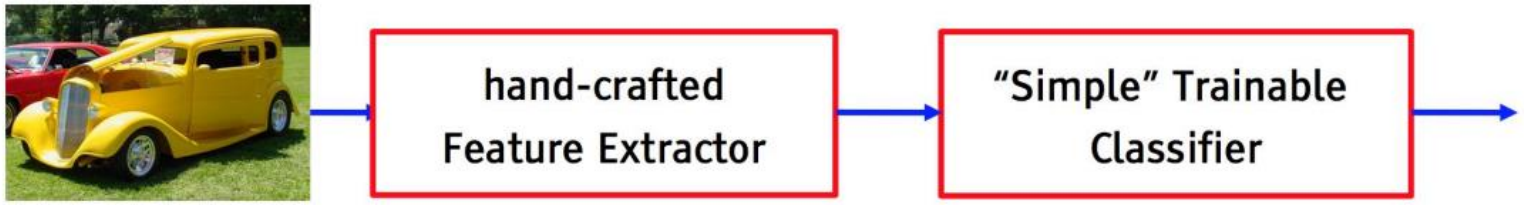


Audio

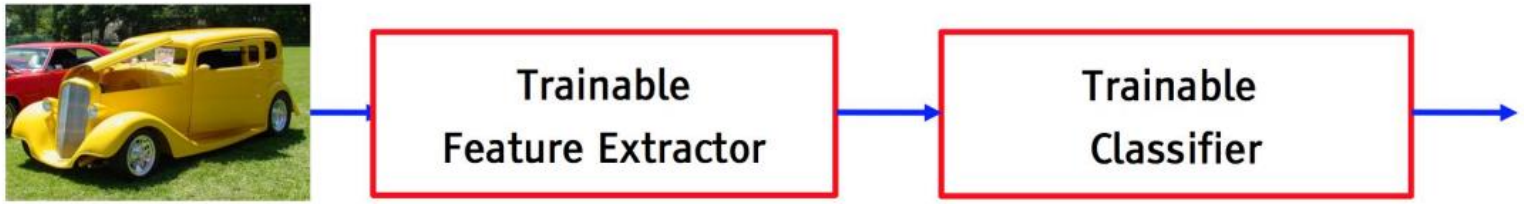
Can we automatically learn “good” feature representations?

How we did before ? (5/8)

- The traditional model of pattern recognition (since the late 50's)
  - Fixed/engineered features (or fixed kernel) + trainable classifier



- End-to-end learning / Feature learning / Deep learning
  - Trainable features (or kernel) + trainable classifier



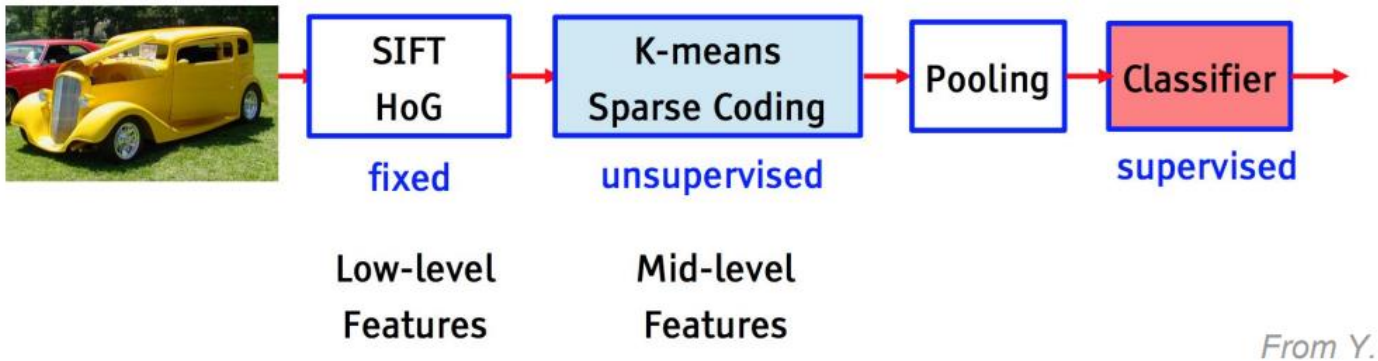
From Y. LeCun's Slides

How we did before ? (6/8)

Modern architecture for pattern recognition

- Speech recognition: early 90's – 2011

- Object Recognition: 2006 - 2012



From Y. LeCun's Slides

How we did before ? (7/8)

Traditional Pattern Recognition: Fixed/Handcrafted Feature Extractor

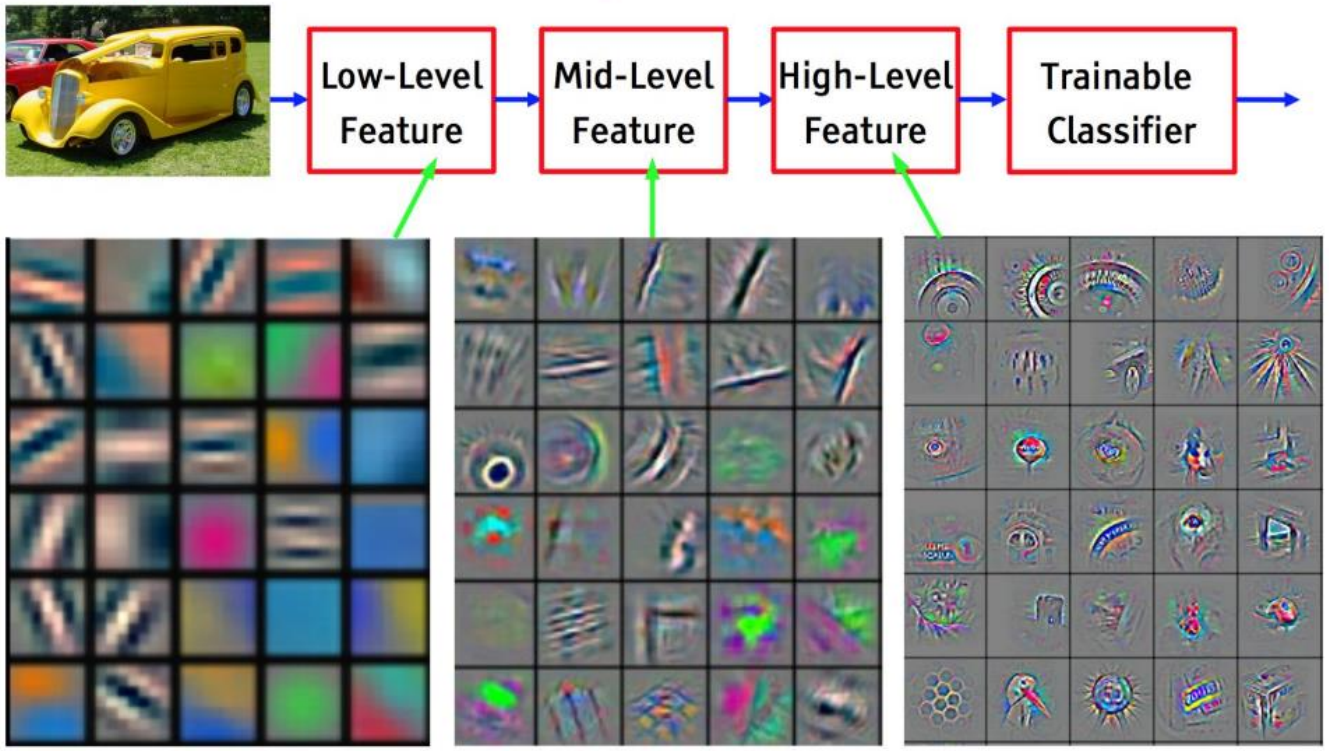
Mainstream Modern Pattern Recognition: Unsupervised mid-level features

Deep Learning: Representations are hierarchical and trained

From Y. LeCun's Slides

How we did before ? (8/8)

It's deep if it has more than one stage of non-linear feature transformation

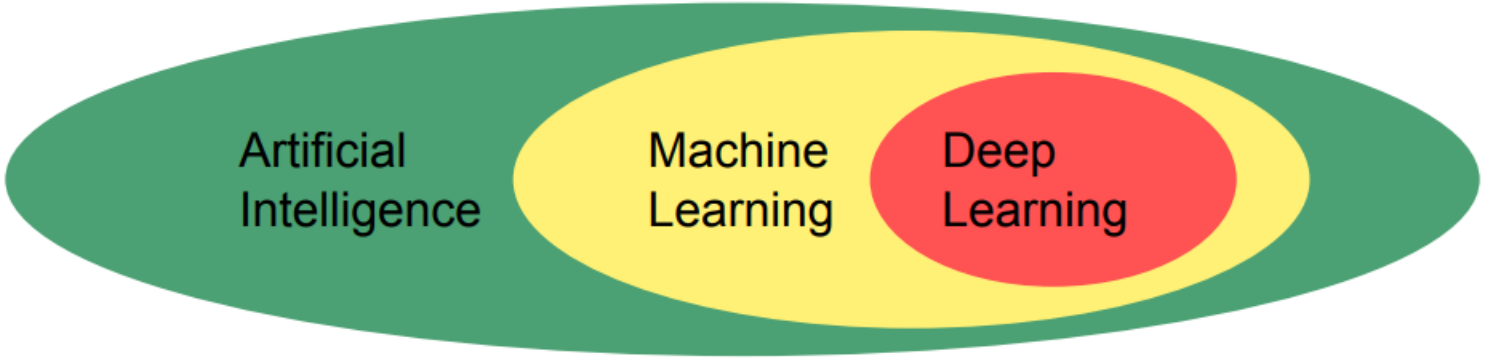


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

What is Deep Learning ? (1/2)

Deep learning allows computational models that are composed of **multiple processing layers to learn representations of data** with **multiple levels of abstraction**.

Deep Learning by Y. LeCun et al. Nature 2015



What is Deep Learning ? (2/2)

Good old Neural Networks, with more layers/modules

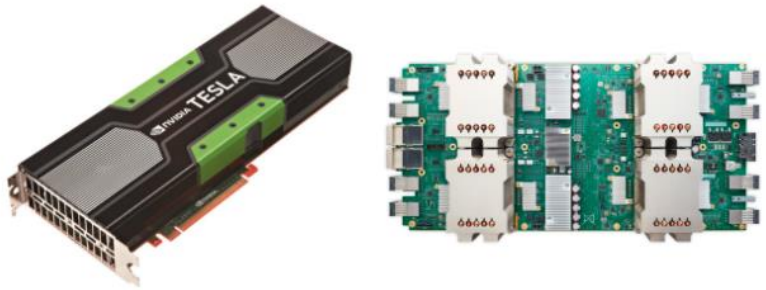
Non-linear, hierarchical, abstract representations of data

Flexible models with any input/output type and size

Differentiable Functional Programming

What is Deep Learning now ? (1/3)

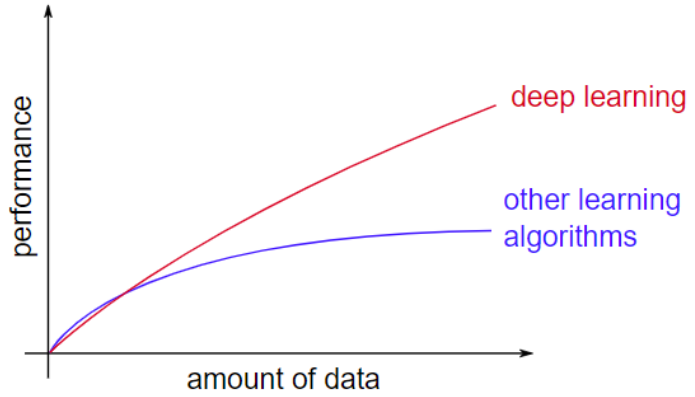
- Better algorithms & understanding
- Computing power (GPUs, TPUs, ...)
- Data with labels
- Open source tools and models



GPU and TPU

What is Deep Learning now ? (2/3)

- Better algorithms & understanding
- Computing power (GPUs, TPUs, ...)
- Data with labels
- Open source tools and models



What is Deep Learning now ? (3/3)

- Better algorithms & understanding
- Computing power (GPUs, TPUs, ...)
- Data with labels
- Open source tools and models

Neural network for classification

Vector function with tunable parameters θ

$$\mathbf{f}(\cdot;\theta) : \mathbb{R}^N \to (0,1)^K$$

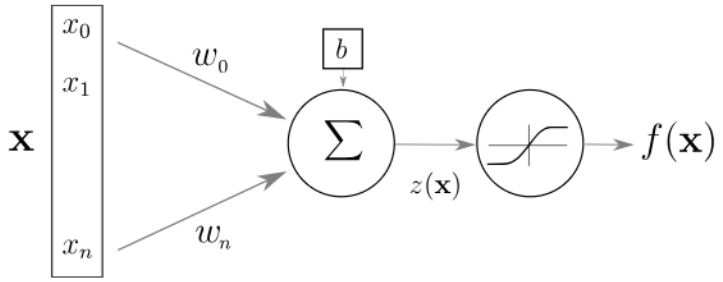
Sample s in dataset S:

- input: $\mathbf{x}^s \in \mathbb{R}^N$
- expected output: $y^s \in [0, K-1]$

Output is a conditional probability distribution:

$$\mathbf{f}(\mathbf{x}^s;\theta)_c = P(Y = c | X = \mathbf{x}^s)$$

Artificial Neuron

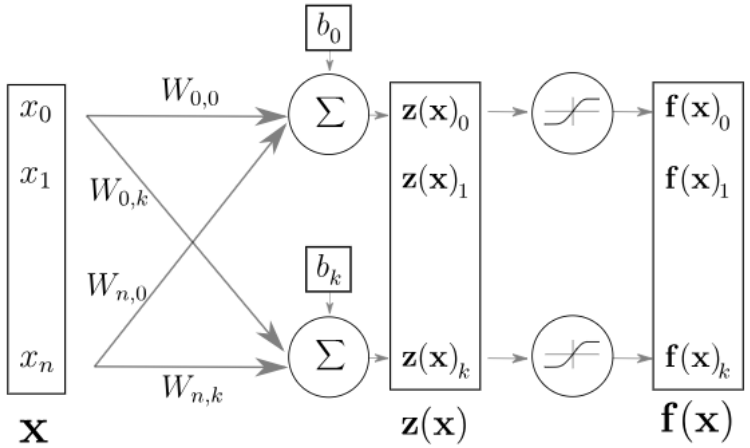


$$z(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

$$f(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + b)$$

- $\mathbf{x}, f(\mathbf{x})$ input and output
- $z(\mathbf{x})$ pre-activation
- $\mathbf{w}, b$ weights and bias
- g activation function

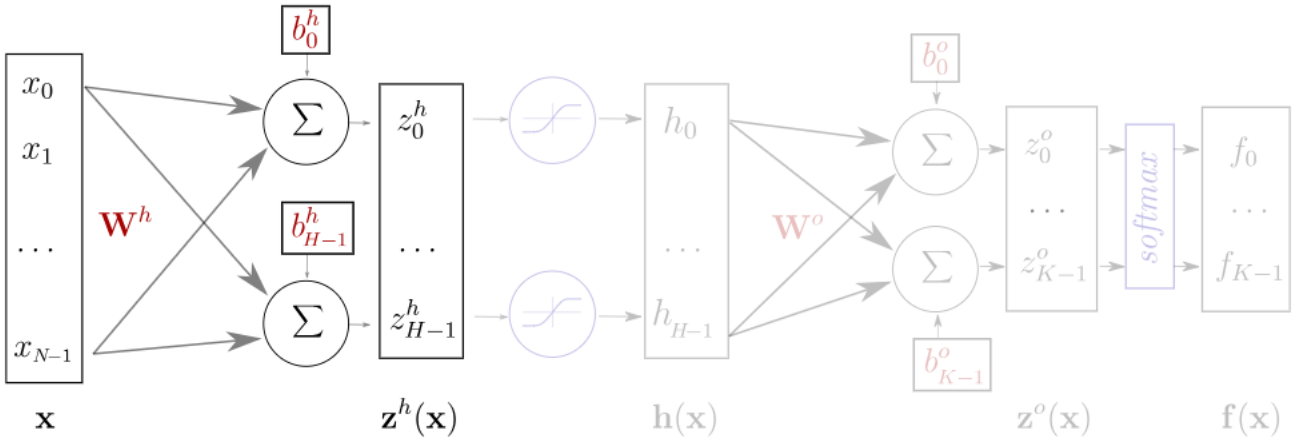
Layer of Neurons



$$\mathbf{f}(\mathbf{x}) = g(\mathbf{z}(\mathbf{x})) = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

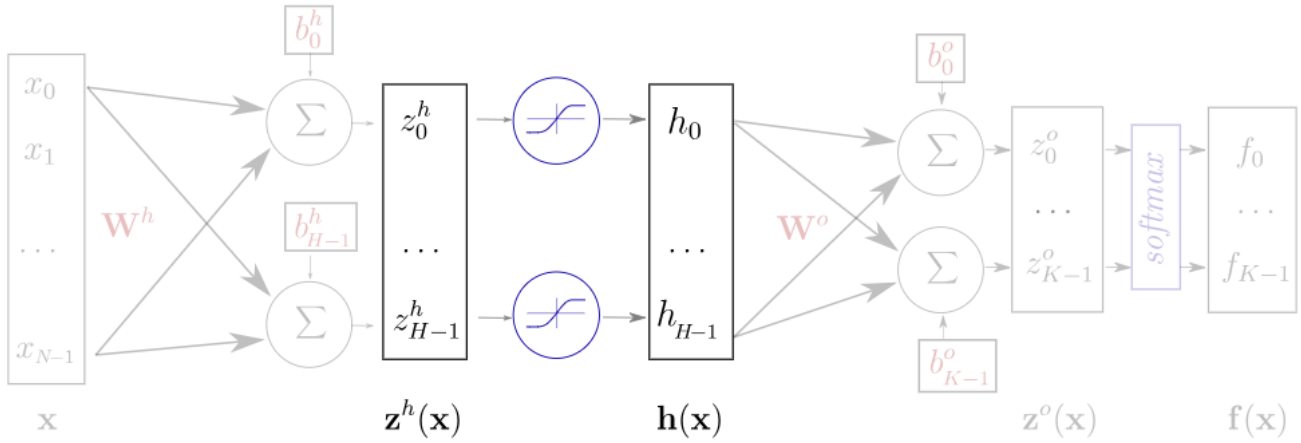
- $\mathbf{W}, \mathbf{b}$ now matrix and vector

One Hidden Layer Network (1/6)



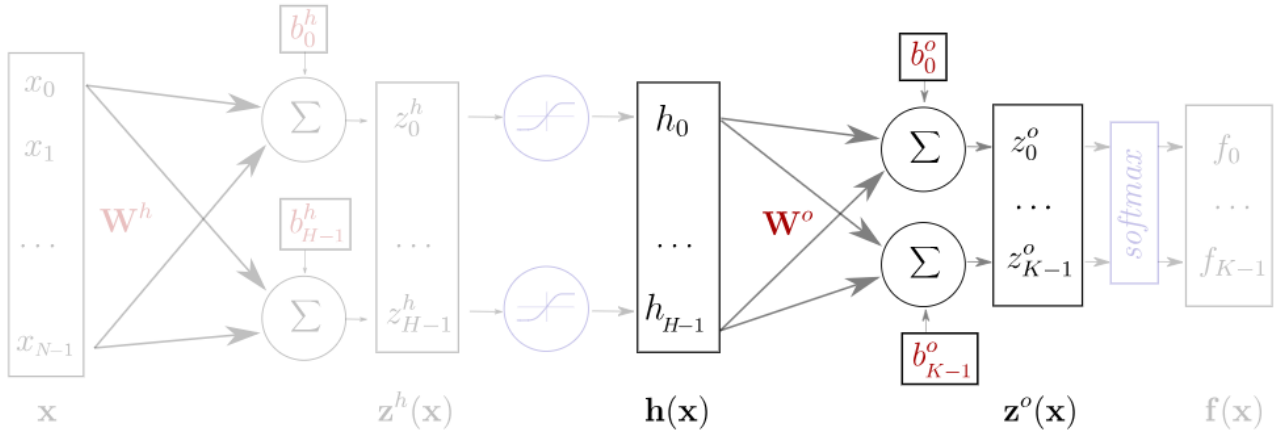
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

One Hidden Layer Network (2/6)



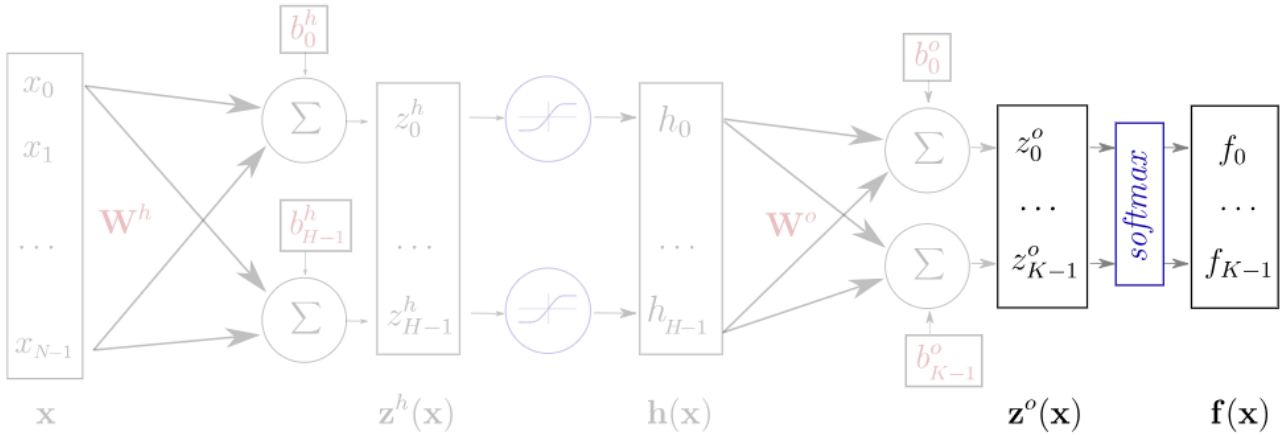
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

One Hidden Layer Network (3/6)



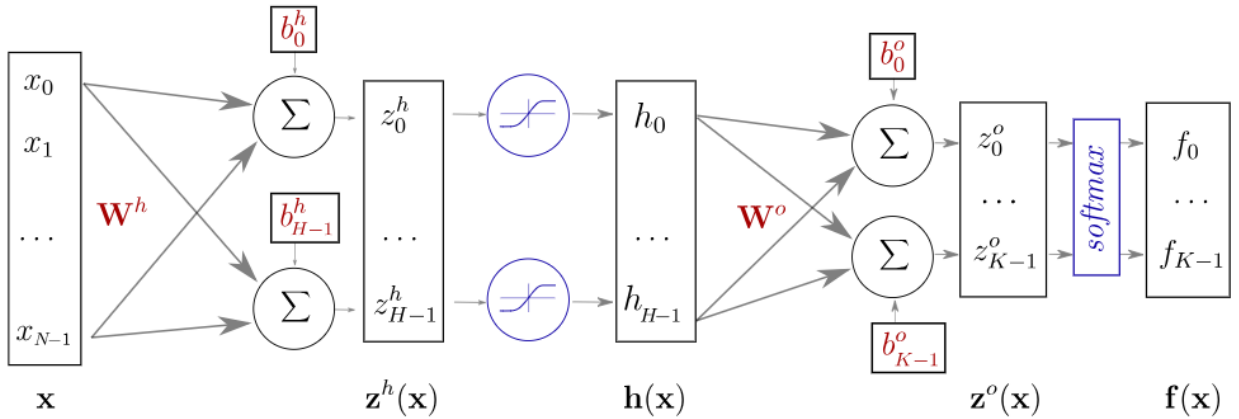
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

One Hidden Layer Network (4/6)

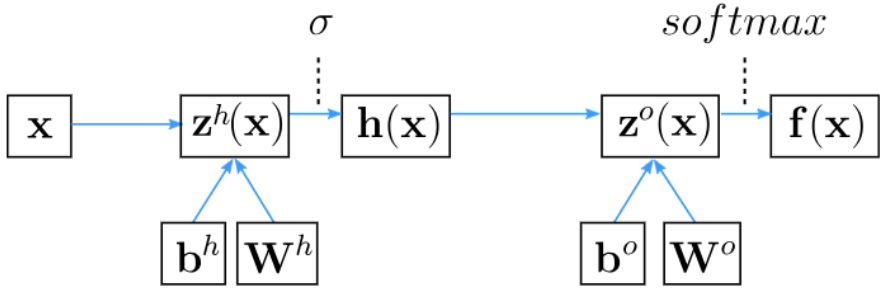


- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

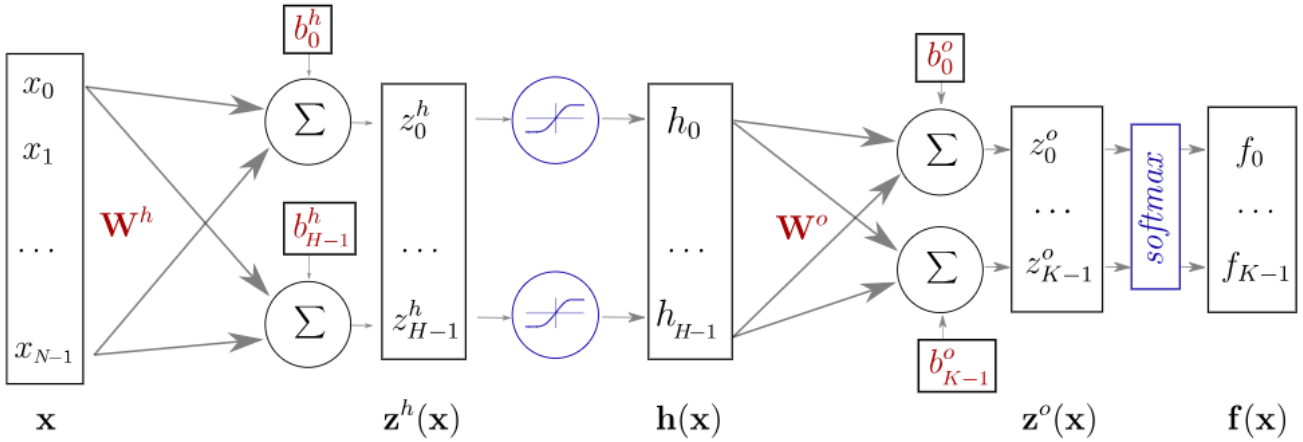
One Hidden Layer Network (5/6)



Alternate representation



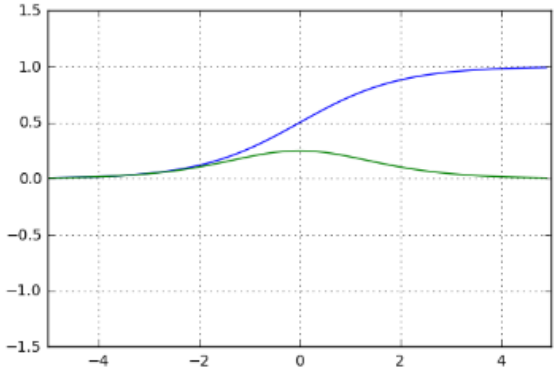
One Hidden Layer Network (6/6)



Keras implementation

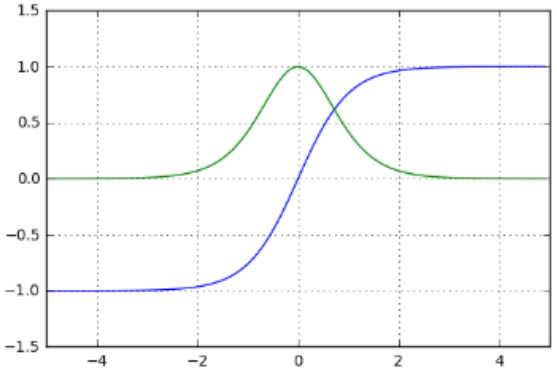
```
model = Sequential()
model.add(Dense(H, input_dim=N))  # weight matrix dim [N * H]
model.add(Activation("tanh"))
model.add(Dense(K))               # weight matrix dim [H x K]
model.add(Activation("softmax"))
```

Element-Wise Activation Functions



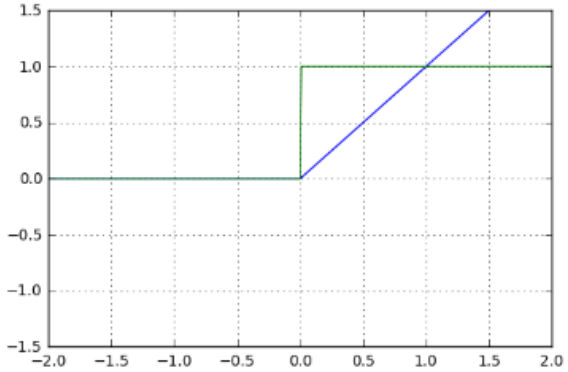
$$\text{sigm}(x) = \frac{1}{1 + e^{-x}}$$

$$\text{sigm}'(x) = \text{sigm}(x)(1 - \text{sigm}(x))$$



$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

$$\tanh'(x) = 1 - \tanh(x)^2$$



$$\text{relu}(x) = \max(0, x)$$

$$\text{relu}'(x) = 1_{x>0}$$

- blue: activation function
- green: derivative

Softmax Function

$$softmax(\mathbf{x}) = \frac{1}{\sum_{i=1}^{n} e^{x_i}} \cdot \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n} \end{bmatrix}$$

$$\frac{\partial softmax(\mathbf{x})_i}{\partial x_j} = \begin{cases} softmax(\mathbf{x})_i \cdot (1 - softmax(\mathbf{x})_i) & i = j \\ -softmax(\mathbf{x})_i \cdot softmax(\mathbf{x})_j & i \neq j \end{cases}$$

- vector of values in (0, 1) that add up to 1
- $p(Y = c|X = \mathbf{x}) = \text{softmax}(\mathbf{z}(\mathbf{x}))_c$
- the pre-activation vector $\mathbf{z}(\mathbf{x})$ is often called "the logits"

Training the Network (1/3)

Find parameters $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize the **negative log likelihood** (or cross entropy)

The loss function for a given sample $s \in S$:

$$l(\mathbf{f}(\mathbf{x}^s; \theta), y^s) = nll(\mathbf{x}^s, y^s; \theta) = -\log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s}$$

example $y^s = 3$

$$l(\mathbf{f}(\mathbf{x}^s; \theta), y^s) = l\left(\begin{bmatrix} f_0 \\ \dots \\ f_3 \\ \dots \\ f_{K-1} \end{bmatrix}, \begin{bmatrix} 0 \\ \dots \\ 1 \\ \dots \\ 0 \end{bmatrix}\right) = -\log\ f_3$$

Training the Network (2/3)

Find parameters $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize the **negative log likelihood** (or cross entropy)

The loss function for a given sample $s \in S$:

$$l(\mathbf{f}(\mathbf{x}^s; \theta), y^s) = nll(\mathbf{x}^s, y^s; \theta) = -\log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s}$$

The cost function is the negative likelihood of the model computed on the full training set (for i.i.d. samples):

$$L_S(\theta) = -\frac{1}{|S|} \sum_{s \in S} \log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s}$$

Training the Network (3/3)

Find parameters $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize the **negative log likelihood** (or cross entropy)

The loss function for a given sample $s \in S$:

$$l(\mathbf{f}(\mathbf{x}^s; \theta), y^s) = nll(\mathbf{x}^s, y^s; \theta) = -\log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s}$$

The cost function is the negative likelihood of the model computed on the full training set (for i.i.d. samples):

$$L_S(\theta) = -\frac{1}{|S|} \sum_{s \in S} \log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s} + \lambda \Omega(\theta)$$

$\lambda \Omega(\theta) = \lambda(||W^h||^2 + ||W^o||^2)$ is an optional regularization term.

Stochastic Gradient Descent

Initialize θ randomly

For E epochs perform:

- Randomly select a small batch of samples $(B \subset S)$
  - Compute gradients: $\Delta = \nabla_\theta L_B(\theta)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$
  - $\eta > 0$ is called the learning rate
- Repeat until the epoch is completed (all of S is covered)

Stop when reaching criterion:

- nll stops decreasing when computed on validation set

Computing Gradients

Output Weights: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial W^o_{i,j}}$

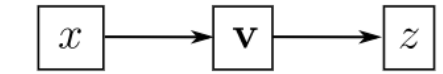
Output bias: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial b^o_i}$

Hidden Weights: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial W^h_{i,j}}$

Hidden bias: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial b^h_i}$

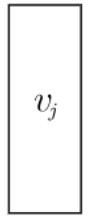
- The network is a composition of differentiable modules
- We can apply the "chain rule"

Chain Rule (1/4)



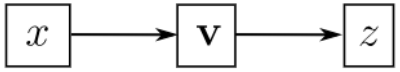
$$z = u(\mathbf{v}(x))$$

$$\frac{\partial z}{\partial x} = ?$$



$\mathbf{v}$

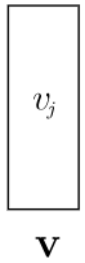
Chain Rule (2/4)



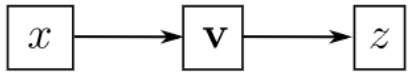
$$z = u(\mathbf{v}(x))$$

chain-rule

$$\frac{\partial z}{\partial x} = \sum_j \frac{\partial z}{\partial v_j} \frac{\partial v_j}{\partial x}$$



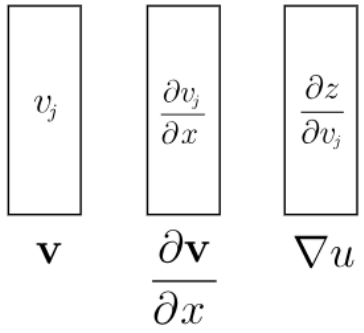
Chain Rule (3/4)



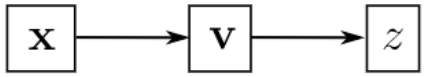
$$z = u(\mathbf{v}(x))$$

chain-rule

$$\frac{\partial z}{\partial x} = \sum_j \frac{\partial z}{\partial v_j} \frac{\partial v_j}{\partial x} = \nabla u . \frac{\partial \mathbf{v}}{\partial x}$$



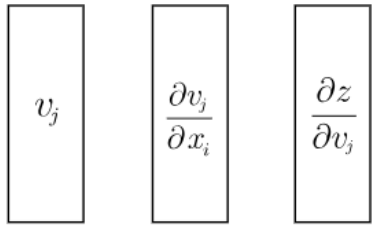
Chain Rule (4/4)



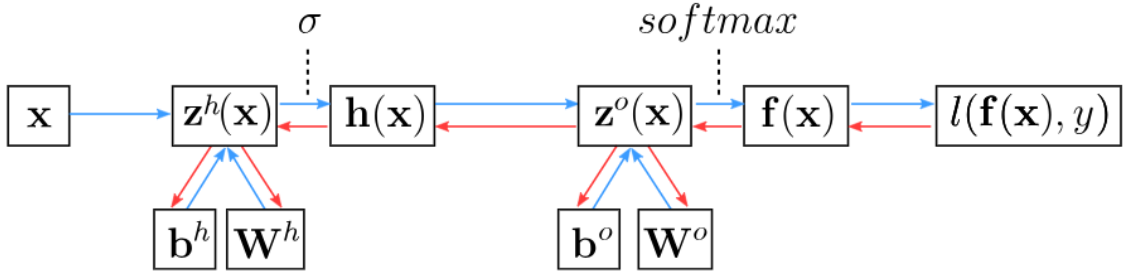
$$z = u(\mathbf{v}(\mathbf{x}))$$

chain-rule

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial v_j} \frac{\partial v_j}{\partial x_i} = \nabla u . \frac{\partial \mathbf{v}}{\partial x_i}$$



Backpropagation (1/8)



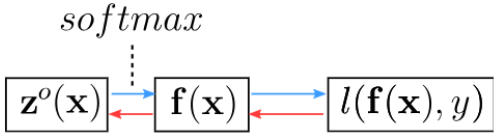
Compute partial derivatives of the loss

- $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial \mathbf{f}(\mathbf{x})_i} = \frac{\partial -\log \mathbf{f}(\mathbf{x})_y}{\partial \mathbf{f}(\mathbf{x})_i} = \frac{-1_{y=i}}{\mathbf{f}(\mathbf{x})_y} = \frac{\partial \boldsymbol{l}}{\partial \mathbf{f}(\mathbf{x})_i}$
- $\frac{\partial \boldsymbol{l}}{\partial \mathbf{z}^o(\mathbf{x})_i} = ?$

Backpropagation (2/8)

$$\frac{\partial l}{\partial \mathbf{z}^o(\mathbf{x})_i} = \sum_j \frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_j} \frac{\partial \mathbf{f}(\mathbf{x})_j}{\partial \mathbf{z}^o(\mathbf{x})_i}$$

Chain rule!



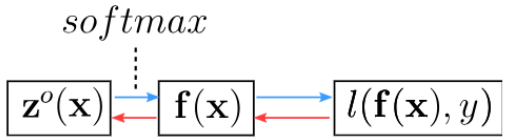
Backpropagation (3/8)

$$\frac{\partial l}{\partial \mathbf{z}^o(\mathbf{x})_i} = \sum_j \frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_j} \frac{\partial \mathbf{f}(\mathbf{x})_j}{\partial \mathbf{z}^o(\mathbf{x})_i}$$

$$= \sum_j \frac{-1_{y=j}}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_j}{\partial \mathbf{z}^o(\mathbf{x})_i}$$

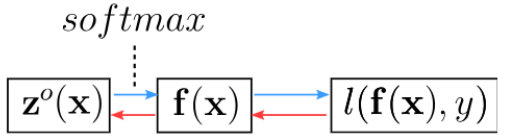
$$\frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_i} = \frac{-1_{y=i}}{\mathbf{f}(\mathbf{x})_y}$$

$$\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o(\mathbf{x}))$$



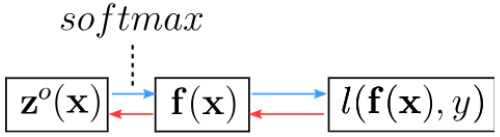
Backpropagation (4/8)

$$
\begin{aligned}
\frac{\partial l}{\partial \mathbf{z}^o(\mathbf{x})_i} &= \sum_j \frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_j} \frac{\partial \mathbf{f}(\mathbf{x})_j}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= \sum_j \frac{-1_{y=j}}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_j}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= -\frac{1}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_y}{\partial \mathbf{z}^o(\mathbf{x})_i}
\end{aligned}
$$



Backpropagation (5/8)

$$
\begin{aligned}
\frac{\partial l}{\partial \mathbf{z}^o(\mathbf{x})_i} &= \sum_j \frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_j} \frac{\partial \mathbf{f}(\mathbf{x})_j}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= \sum_j \frac{-1_{y=j}}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_j}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= -\frac{1}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_y}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= \begin{cases} -\frac{1}{\mathbf{f}(\mathbf{x})_y} softmax(\mathbf{z}^o(\mathbf{x}))_y (1 - softmax(\mathbf{z}^o(\mathbf{x}))_y) & \textbf{if } i = y \\ \frac{1}{\mathbf{f}(\mathbf{x})_y} softmax(\mathbf{z}^o(\mathbf{x}))_y softmax(\mathbf{z}^o(\mathbf{x}))_i & \textbf{if } i \neq y \end{cases} \\
&= \begin{cases} -1 + \mathbf{f}(\mathbf{x})_y & \textbf{if } i = y \\ \mathbf{f}(\mathbf{x})_i & \textbf{if } i \neq y \end{cases}
\end{aligned}
$$

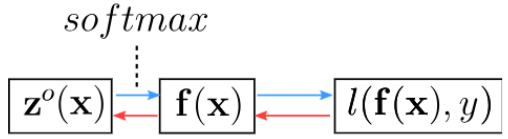


Backpropagation (6/8)

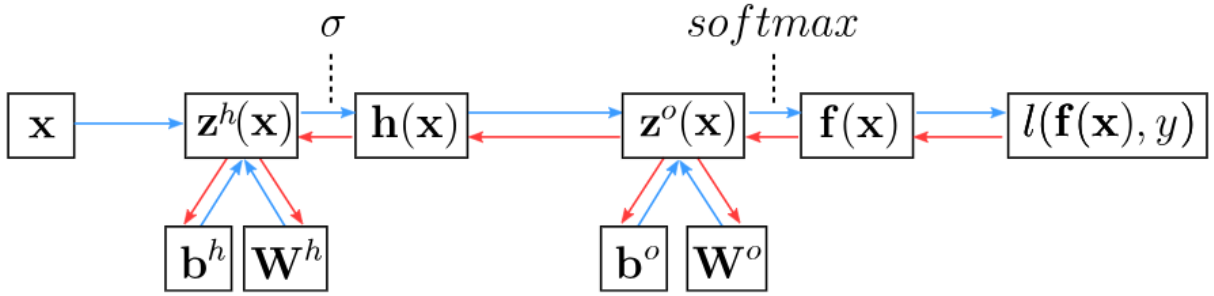
$$
\begin{aligned}
\frac{\partial l}{\partial \mathbf{z}^o(\mathbf{x})_i} &= \sum_j \frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_j} \frac{\partial \mathbf{f}(\mathbf{x})_j}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= \sum_j \frac{-1_{y=j}}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_j}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= -\frac{1}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_y}{\partial \mathbf{z}^o(\mathbf{x})_i} \\
&= \begin{cases} -\frac{1}{\mathbf{f}(\mathbf{x})_y} softmax(\mathbf{z}^o(\mathbf{x}))_y (1 - softmax(\mathbf{z}^o(\mathbf{x}))_y) & \text{if } i = y \\ \frac{1}{\mathbf{f}(\mathbf{x})_y} softmax(\mathbf{z}^o(\mathbf{x}))_y softmax(\mathbf{z}^o(\mathbf{x}))_i & \text{if } i \neq y \end{cases} \\
&= \begin{cases} -1 + \mathbf{f}(\mathbf{x})_y & \text{if } i = y \\ \mathbf{f}(\mathbf{x})_i & \text{if } i \neq y \end{cases}
\end{aligned}
$$

$$\nabla_{\mathbf{z}^o(\mathbf{x})} l(\mathbf{f}(\mathbf{x}), y) = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$$

$\mathbf{e}(y)$: one-hot encoding of y



Backpropagation (7/8)

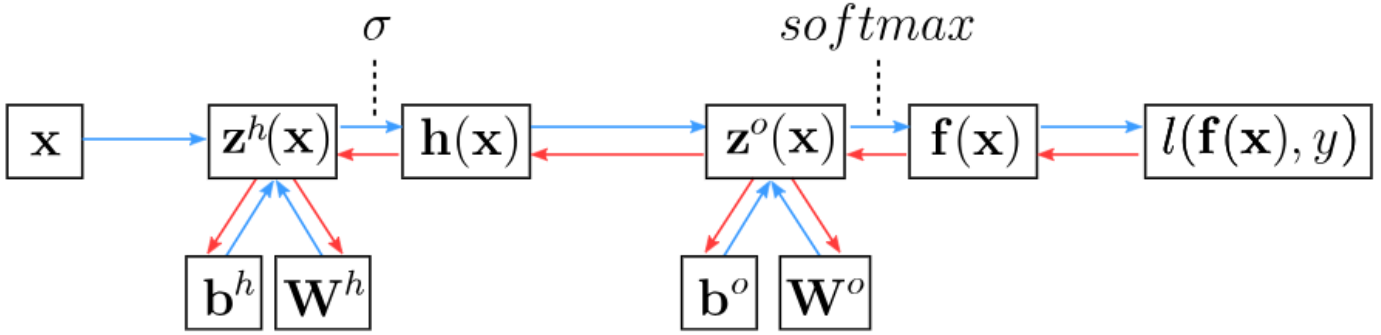


Gradients

- $\nabla_{\mathbf{z}^o(\mathbf{x})} l = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$
- $\nabla_{\mathbf{b}^o} l = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$

because $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o \mathbf{h}(\mathbf{x}) + \mathbf{b}^o$ and then $\frac{\partial \mathbf{z}^o(\mathbf{x})_i}{\partial \mathbf{b}^o_j} = 1_{i=j}$

Backpropagation (8/8)



Partial derivatives related to $\mathbf{W}^o$

- $\frac{\partial \boldsymbol{l}}{\partial W^o_{i,j}} = \sum_k \frac{\partial \boldsymbol{l}}{\partial \mathbf{z}^o(\mathbf{x})_k} \frac{\partial \mathbf{z}^o(\mathbf{x})_k}{\partial W^o_{i,j}}$
- $\nabla_{\mathbf{W}^o} \boldsymbol{l} = (\mathbf{f}(\mathbf{x}) - \mathbf{e}(y)).\mathbf{h}(\mathbf{x})^\top$

Backprop Gradients

Compute activation gradients

- $\nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l} = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$

Compute layer params gradients

- $\nabla_{\mathbf{W}^o} \boldsymbol{l} = \nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l} \cdot \mathbf{h}(\mathbf{x})^\top$
- $\nabla_{\mathbf{b}^o} \boldsymbol{l} = \nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l}$

Compute prev layer activation gradients

- $\nabla_{\mathbf{h}(\mathbf{x})} \boldsymbol{l} = \mathbf{W}^{o\top} \nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l}$
- $\nabla_{\mathbf{z}^h(\mathbf{x})} \boldsymbol{l} = \nabla_{\mathbf{h}(\mathbf{x})} \boldsymbol{l} \odot \sigma'(\mathbf{z}^{\mathbf{h}}(\mathbf{x}))$

Discrete Output (Classification)

- Binary classification: $y \in [0, 1]$
  - $Y|X = \mathbf{x} \sim Bernoulli(b = f(\mathbf{x}; \theta))$
  - output function: $logistic(x) = \frac{1}{1+e^{-x}}$
  - loss function: binary cross-entropy
- Multiclass classification: $y \in [0, K-1]$
  - $Y|X = \mathbf{x} \sim Multinoulli(\mathbf{p} = \mathbf{f}(\mathbf{x}; \theta))$
  - output function: $softmax$
  - loss function: categorical cross-entropy

Continuous Output (Regression)

- Continuous output: $\mathbf{y} \in \mathbb{R}^n$
  - $Y|X = \mathbf{x} \sim \mathcal{N}(\mu = \mathbf{f}(\mathbf{x}; \theta), \sigma^2 \mathbf{I})$
  - output function: Identity
  - loss function: square loss
- Heteroschedastic if $\mathbf{f}(\mathbf{x}; \theta)$ predicts both μ and σ^2
- Mixture Density Network (multimodal output)
  - $Y|X = \mathbf{x} \sim GMM_{\mathbf{x}}$
  - $\mathbf{f}(\mathbf{x}; \theta)$ predicts all the parameters: the means, covariance matrices and mixture weights

Initialization and Normalization

- Input data should be normalized to have approx. same range:
  - standardization or quantile normalization
- Initializing W^h and W^o:
  - Zero is a saddle point: no gradient, no learning
  - Constant init: hidden units collapse by symmetry
  - Solution: random init, ex: $w \sim \mathcal{N}(0, 0.01)$
  - Better inits: Xavier Glorot and Kaming He & orthogonal
- Biases can (should) be initialized to zero

SGD Learning Rate

- Very sensitive:
  - Too high $\rightarrow$ early plateau or even divergence
  - Too low $\rightarrow$ slow convergence
  - Try a large value first: $\eta = 0.1$ or even $\eta = 1$
  - Divide by 10 and retry in case of divergence
- Large constant LR prevents final convergence
  - multiply η_t by $\beta < 1$ after each update
  - or monitor validation loss and divide η_t by 2 or 10 when no progress
  - See ReduceLROnPlateau in Keras

Momentum

Accumulate gradients across successive updates:

$$m_t = \gamma m_{t-1} + \eta \nabla_\theta L_{B_t}(\theta_{t-1})$$
$$\theta_t = \theta_{t-1} - m_t$$

γ is typically set to 0.9

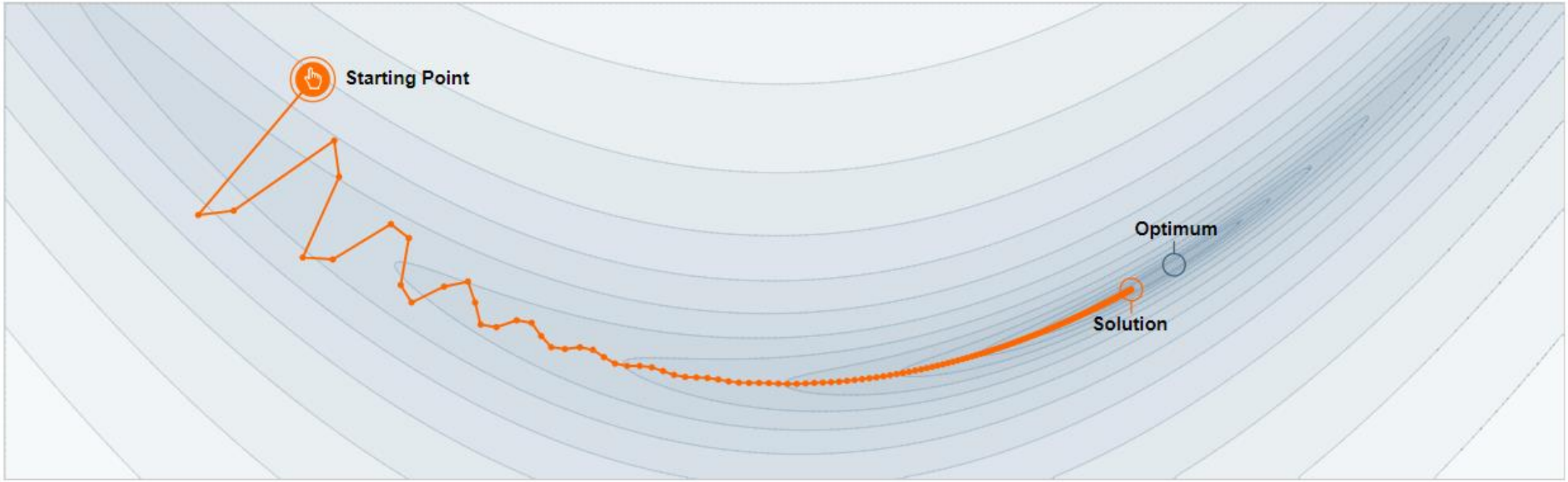
Larger updates in directions where the gradient sign is constant to accelerate in low curvature areas

Nesterov accelerated gradient

$$m_t = \gamma m_{t-1} + \eta \nabla_\theta L_{B_t}(\theta_{t-1} - \gamma m_{t-1})$$
$$\theta_t = \theta_{t-1} - m_t$$

Better at handling changes in gradient direction.

Why Momentum Really Works ?



Step-size α = 0.0029

0 0.003 0.006

Momentum β = 0.74

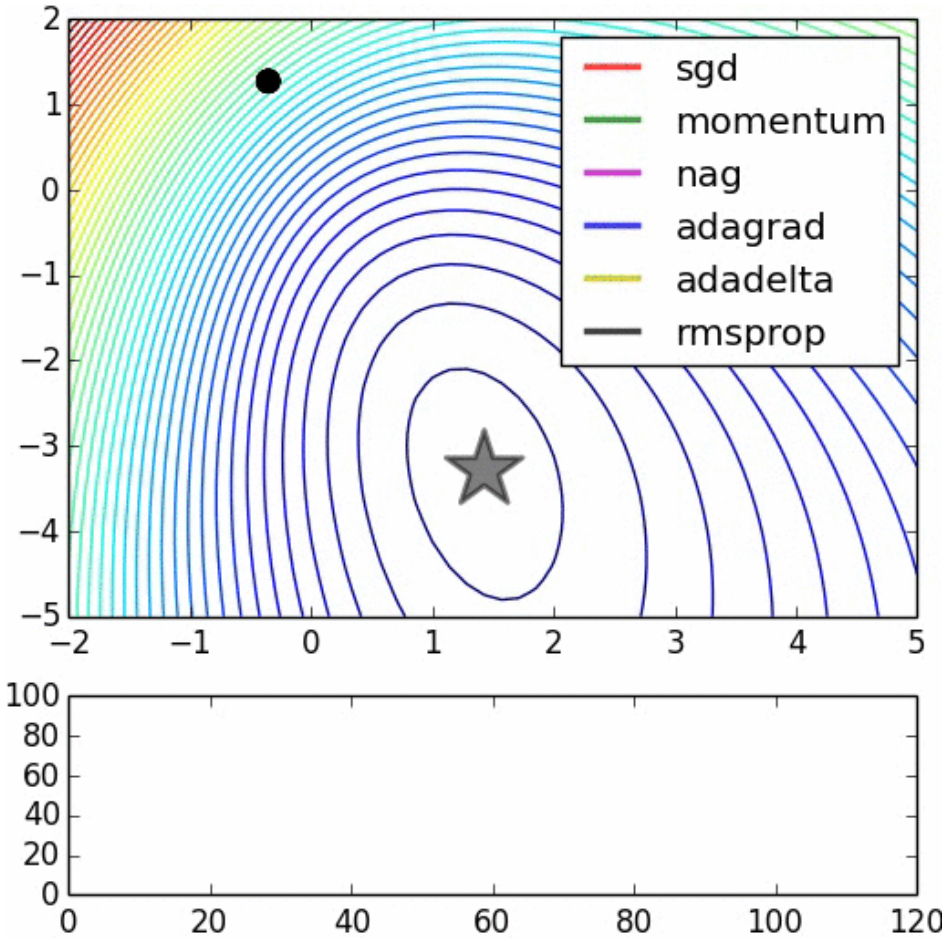
0.00 0.500 0.990

We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

Alternative Optimizers

- SGD (with Nesterov momentum)
  - Simple to implement
  - Very sensitive to initial value of η
  - Need learning rate scheduling
- Adam: adaptive learning rate scale for each param
  - Global η set to 3e-4 often works well enough
  - Good default choice of optimizer (often)
- But well-tuned SGD with LR scheduling can generalize better than Adam (with naive l2 reg)...
- Promising stochastic second order methods: K-FAC and Shampoo can be used to accelerate training of very large models.

Optimizers around a saddle point



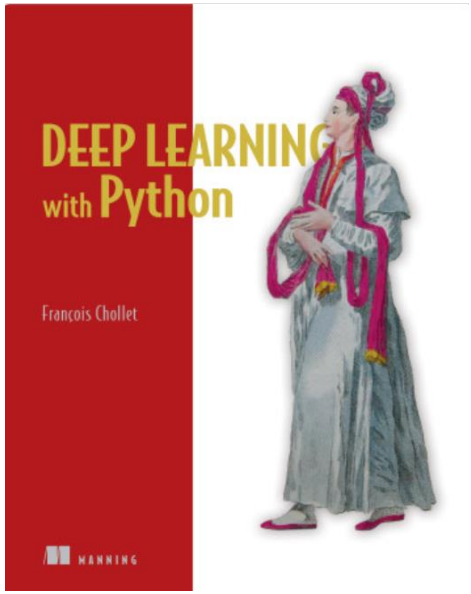
Labs & Homework

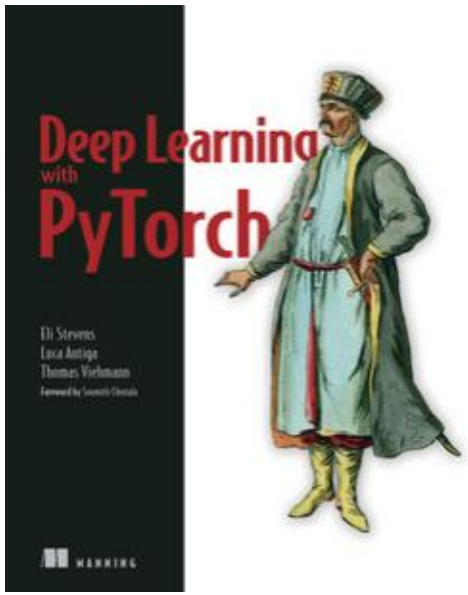
- TP de la séance
  - https://github.com/srtj89/isheero_atut_2024_deep_learning_intro.git
  - télécharger isheero_atut_2024_deep_learning_intro.zip
  - *backpropagation (Numpy)*

- Devoir de maison à rendre le 08/08/2024
  - *backpropagation (Tensorflow)*
  - correction à fournir à ceux qui auront fait l'exercice
  - se documenter sur les CNN

Some Ressources

- Transformer I
- Transformer II
- Math and main concepts





- Hands-on Machine Learning with Scikit-Learn, Keras and Tensor..

Merci de votre attention

retrouvez la version mobile de Shuri à but de formation via ce QR Code

https://shuri.adjibola.tech/