Introduction to Deep Learning

Charles Ollion - Olivier Grisel

What is Deep Learning

Good old Neural networks, with more layer/modules

What is Deep Learning

Good old Neural networks, with more layer/modules

Compute hierarchical, abstract representations of data

What is Deep Learning

Good old Neural networks, with more layer/modules

Compute hierarchical, abstract representations of data

Flexible models with any input/output type and size

What is Deep Learning

Feature Extractions

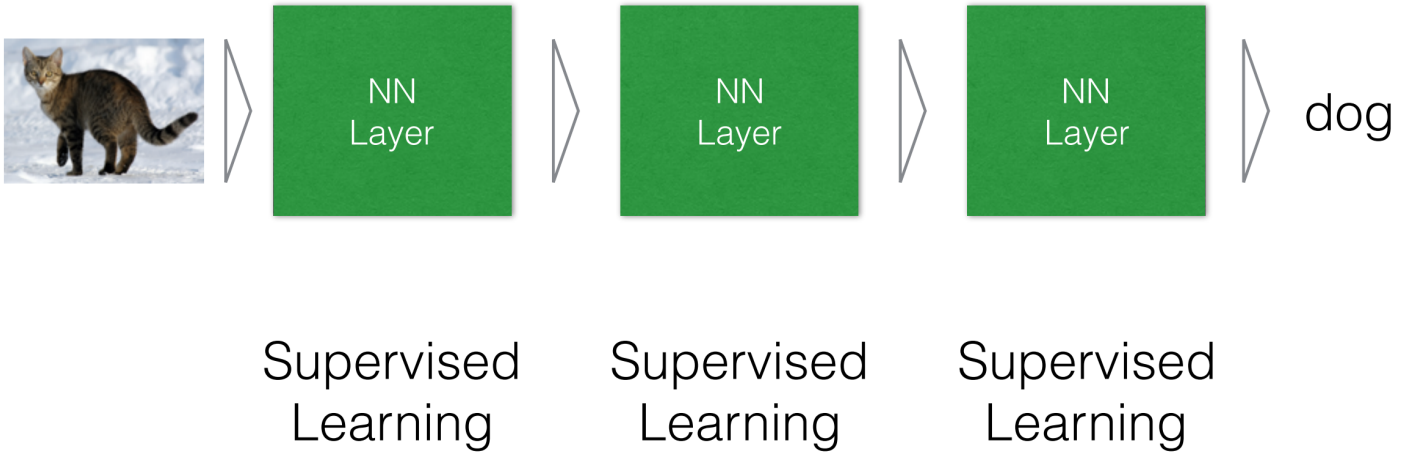
Classification

dog

Data independent

Supervised Learning

What is Deep Learning

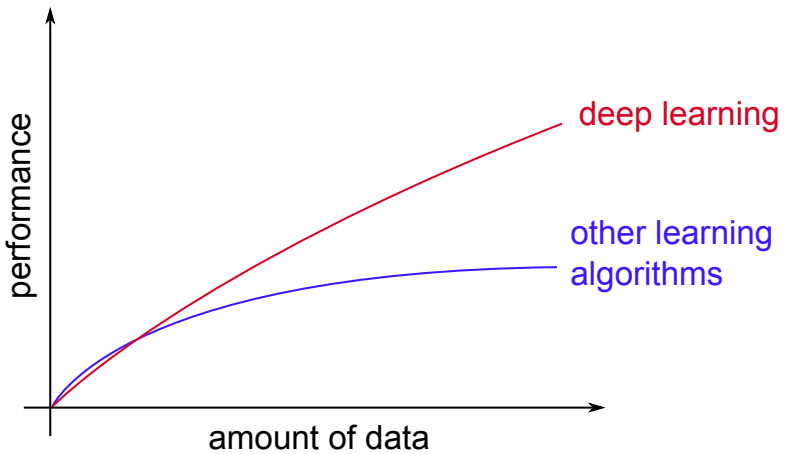


Why Deep Learning Now?

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

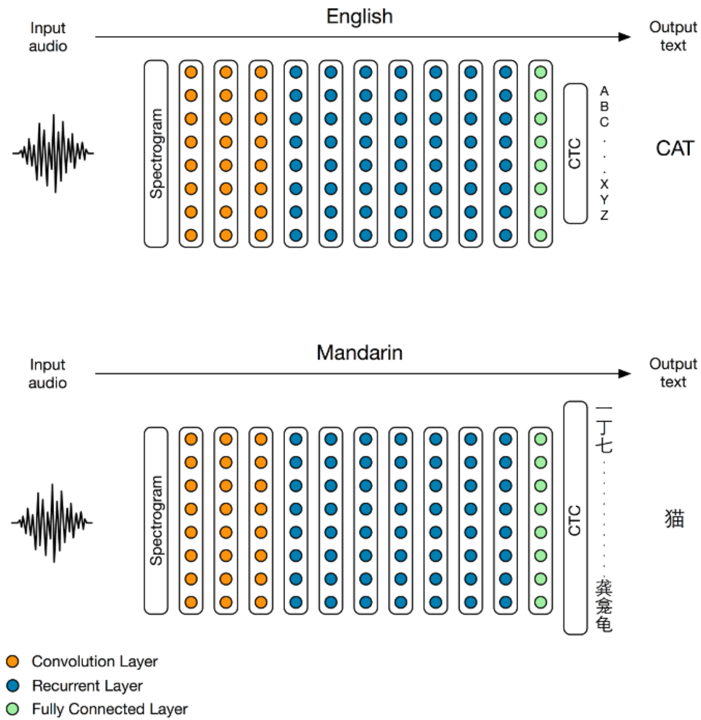
Why Deep Learning Now?

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



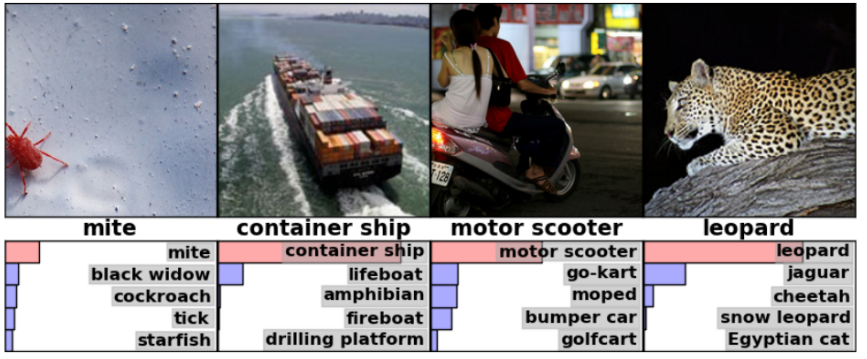
Adapted from Andrew Ng

Where is Deep Learning today



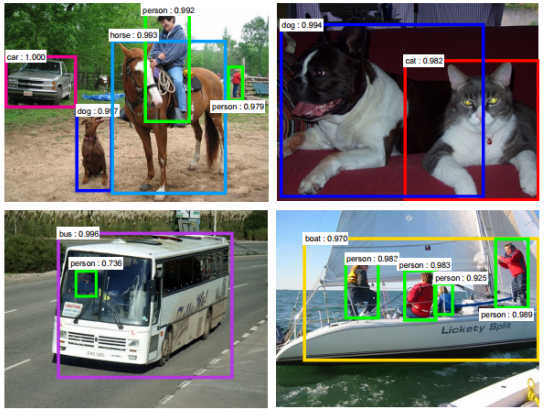
[Baidu 2014]

Where is Deep Learning today



[Krizhevsky 2012]

[Ciresan et al. 2013]

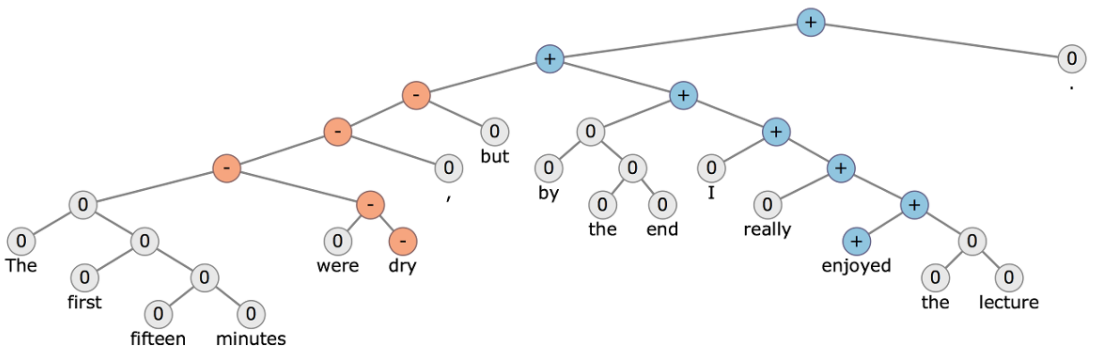


[Faster R-CNN - Ren 2015]

[NVIDIA dev blog]

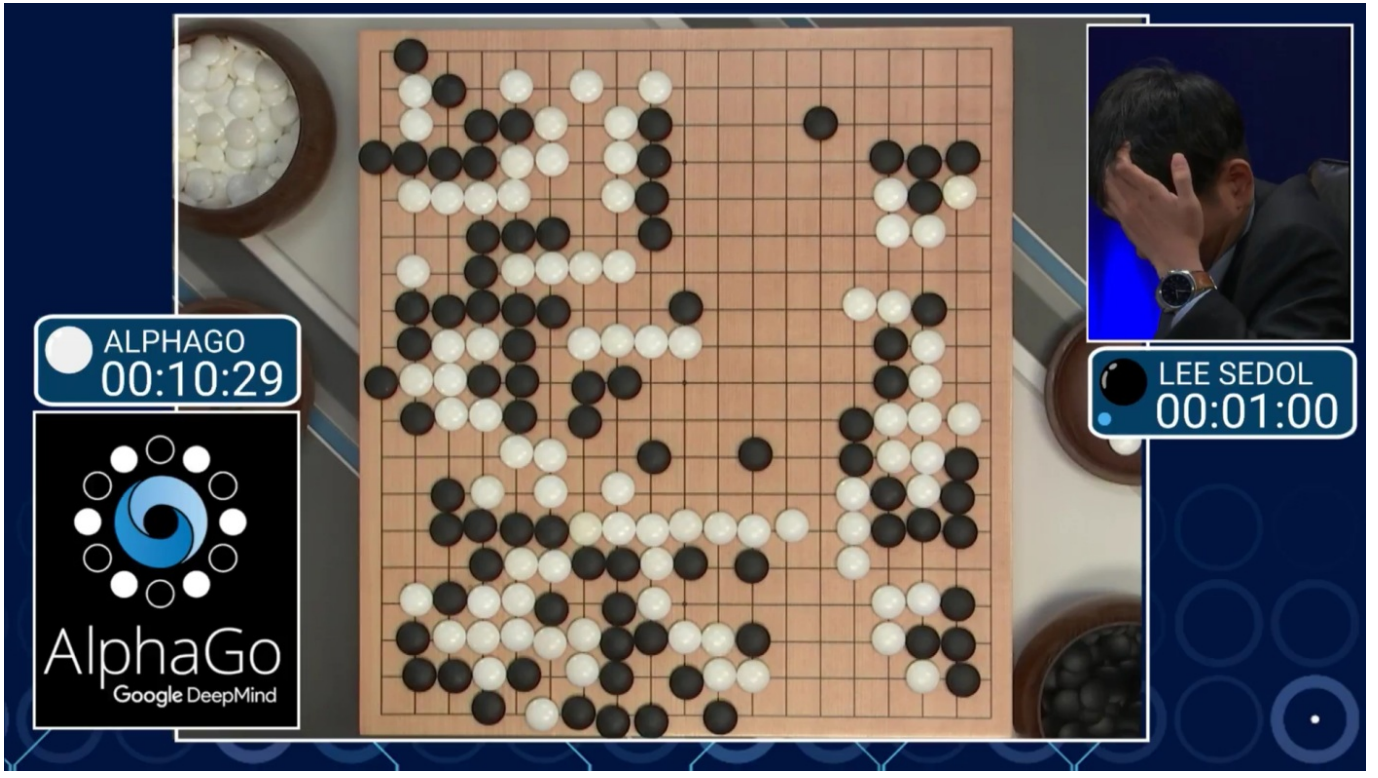
Where is Deep Learning today

[Google Translate System - 2016]



[Socher 2015]

Where is Deep Learning today



Where is Deep Learning today

Stanford | News

Search Stanford news...

Home Find Stories For Journalists Contact

JANUARY 25, 2017

Deep learning algorithm does as well as dermatologists in identifying skin cancer

In hopes of creating better access to medical care, Stanford researchers have trained an algorithm to diagnose skin cancer.

BY TAYLOR KUBOTA

It's scary enough making a doctor's appointment to see if a strange mole could be cancerous. Imagine, then, that you were in that situation while also living far away from the nearest doctor, unable to take time off work and unsure you had the money to cover the cost of the visit. In a scenario like this, an option to receive a diagnosis through your smartphone could be lifesaving.

Universal access to health care was on the minds of computer scientists at Stanford when they set out to create an artificially intelligent diagnosis algorithm for skin cancer. They made a database of nearly 130,000 skin disease images and trained their algorithm to visually diagnose potential cancer. From the very first test, it performed with inspiring accuracy.

Goal of the class

Overview

- When and where to use DL
- "How" it works
- Frontiers of DL

Goal of the class

Overview

- When and where to use DL
- "How" it works
- Frontiers of DL

Arcanes of DL

- Implement using `Numpy`, `TensorFlow` and `Keras`
- Engineering knowledge for building and training DL

Outline of the class

Backpropagation

Outline of the class

Backpropagation

Recommender Systems

Outline of the class

Backpropagation

Recommender Systems

Computer Vision (1 & 2)

Outline of the class

Backpropagation

Recommender Systems

Computer Vision (1 & 2)

Expressive power and optimization of deep networks

Outline of the class

Backpropagation

Recommender Systems

Computer Vision (1 & 2)

Expressive power and optimization of deep networks

Natural Language Processing

How this unit works

Lecture 1h-1h30

Coding sessions 2h-2h30

- 5 min multiple choice evaluation of previous lab
- split into 2 groups
- BYO laptop, work by pairs
- Homework 3h per week

How this unit works

Lecture 1h-1h30

Coding sessions 2h-2h30

- 5 min multiple choice evaluation of previous lab
- split into 2 groups
- BYO laptop, work by pairs
- Homework 3h per week

Final exam 2h

How this unit works

Lecture 1h-1h30

Coding sessions 2h-2h30

- 5 min multiple choice evaluation of previous lab
- split into 2 groups
- BYO laptop, work by pairs
- Homework 3h per week

Final exam 2h

Recommended reading: deeplearningbook.org

Neural Networks & Backpropagation

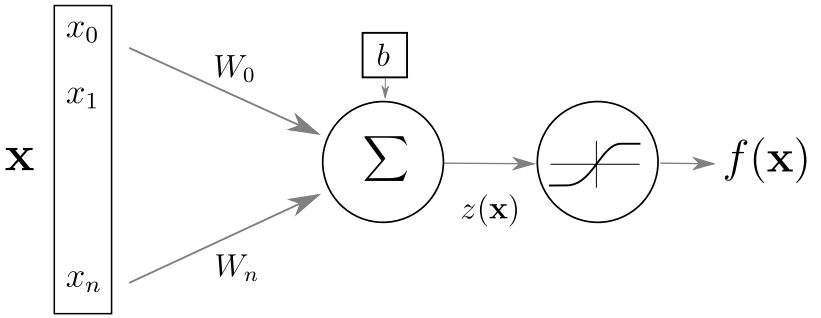
Neural Network for classification

- Vector function with tunable parameters θ

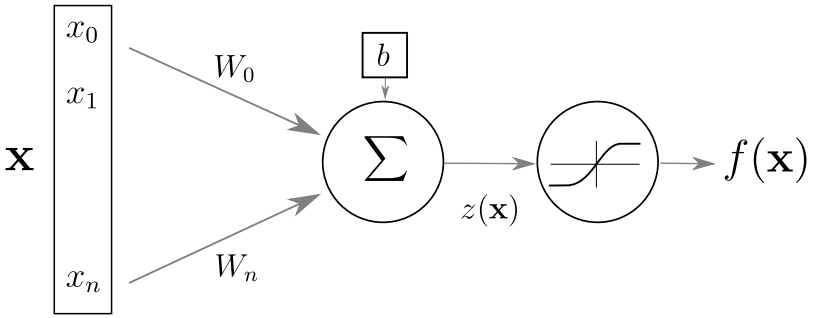
$$\mathbf{f}(\cdot;\theta) : \mathbb{R}^N \to (0,1)^K$$

- s sample in dataset S:
  - input: $\mathbf{x}^s \in \mathbb{R}^N$
  - expected output: $y^s \in [0, K-1]$
- probability: $\mathbf{f}(\mathbf{x}^s;\theta)_c = p(Y = c | X = \mathbf{x}^s)$

Artificial Neuron



Artificial Neuron

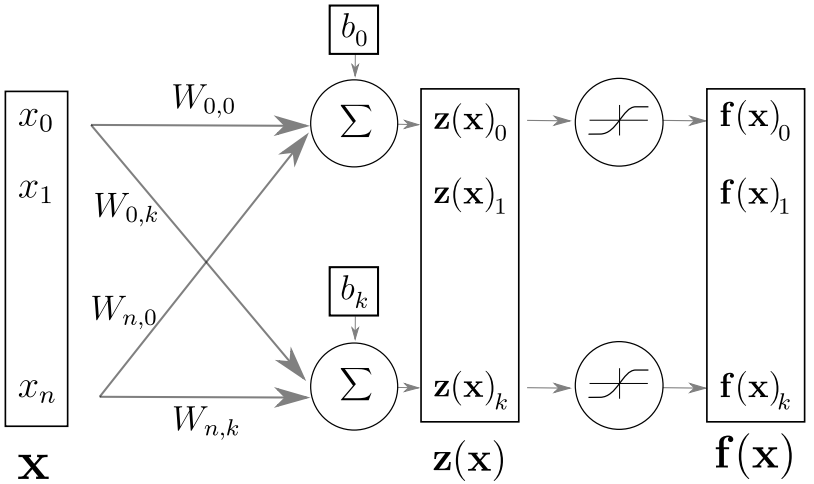


$$z(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

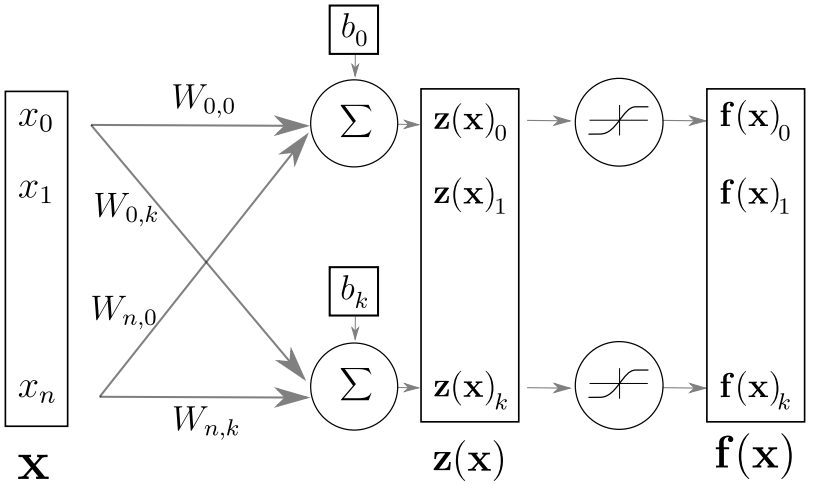
$$f(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + b)$$

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

Layer of Neurons



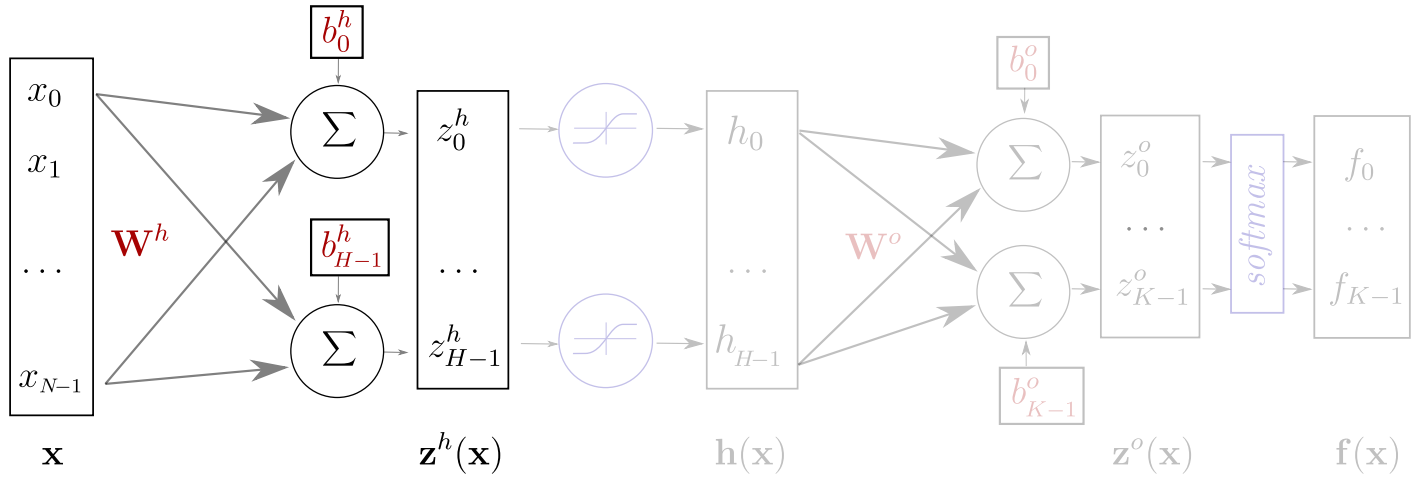
Layer of Neurons



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

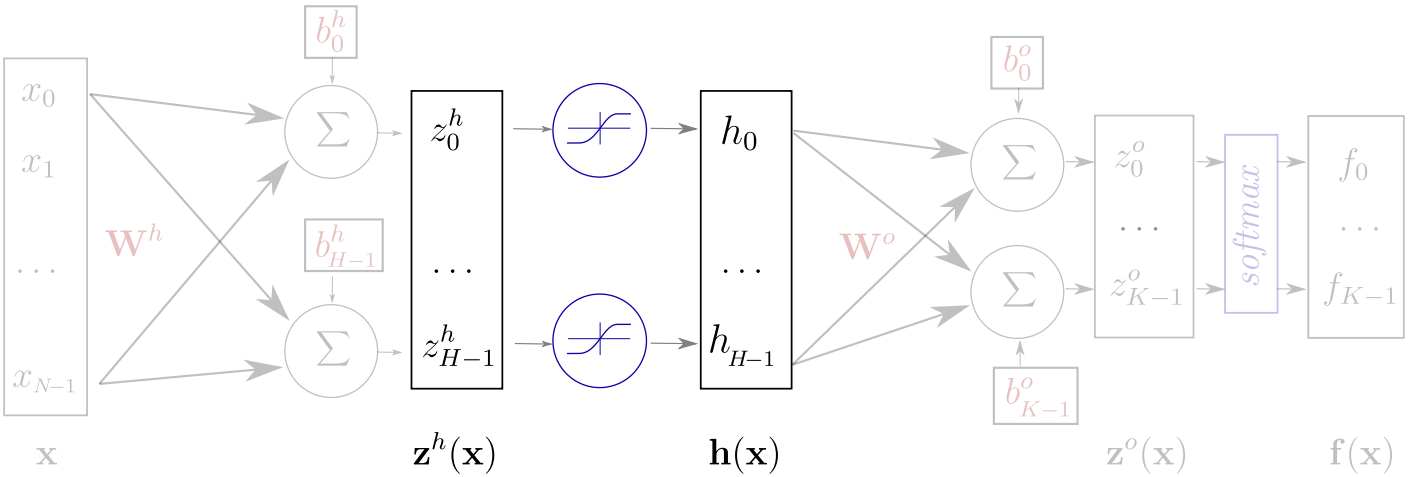
- $\mathbf{W}, \mathbf{b}$ now matrix and vector

One Hidden Layer Network



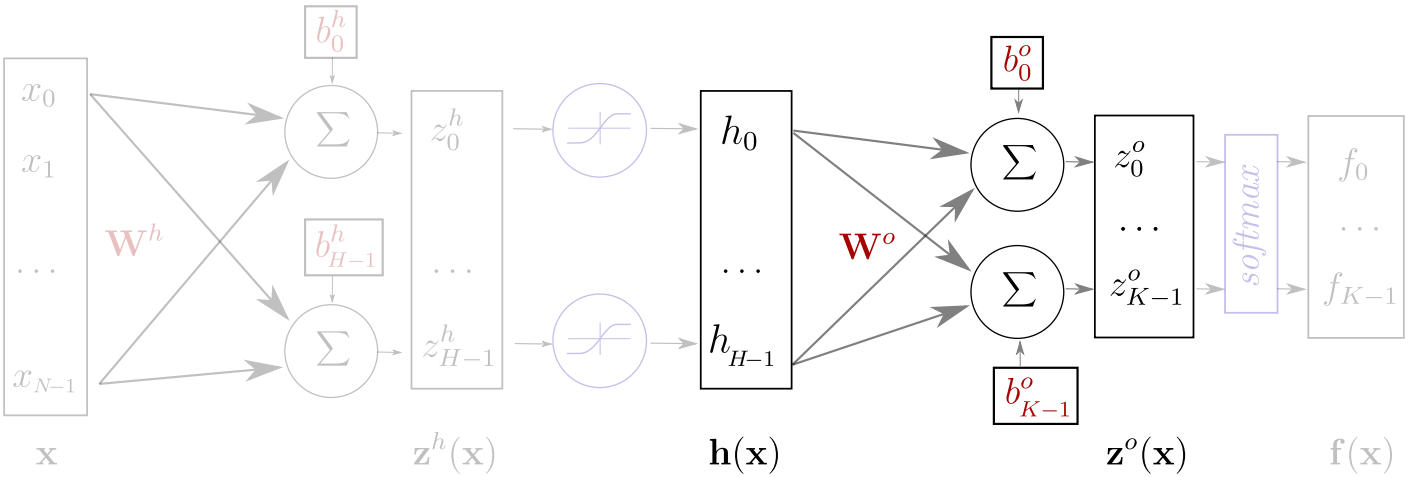
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) =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

One Hidden Layer Network



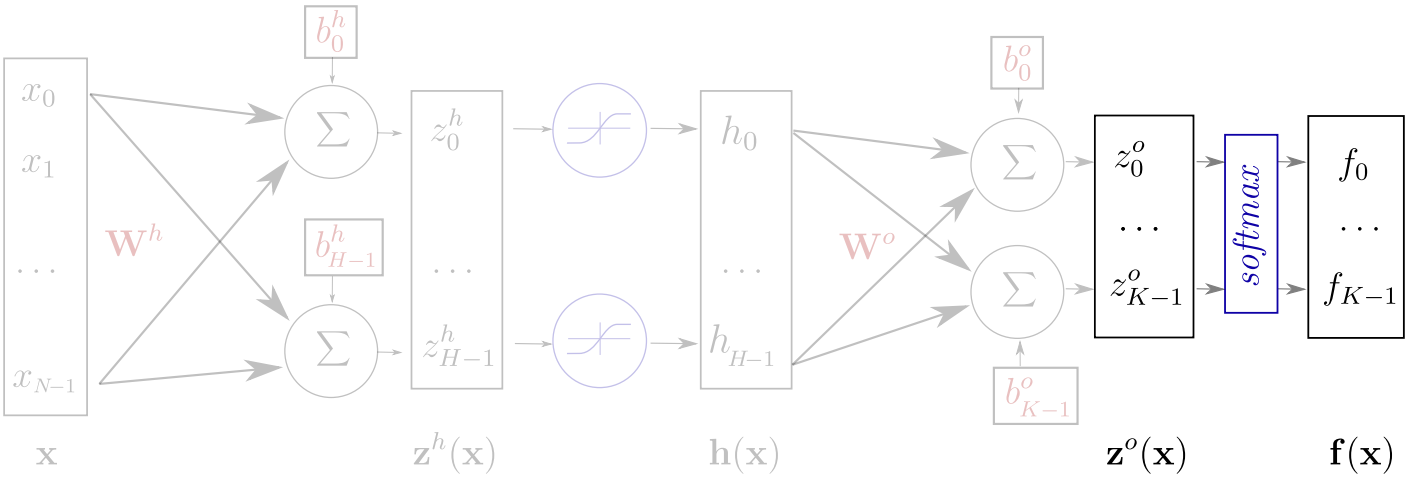
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) =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

One Hidden Layer Network



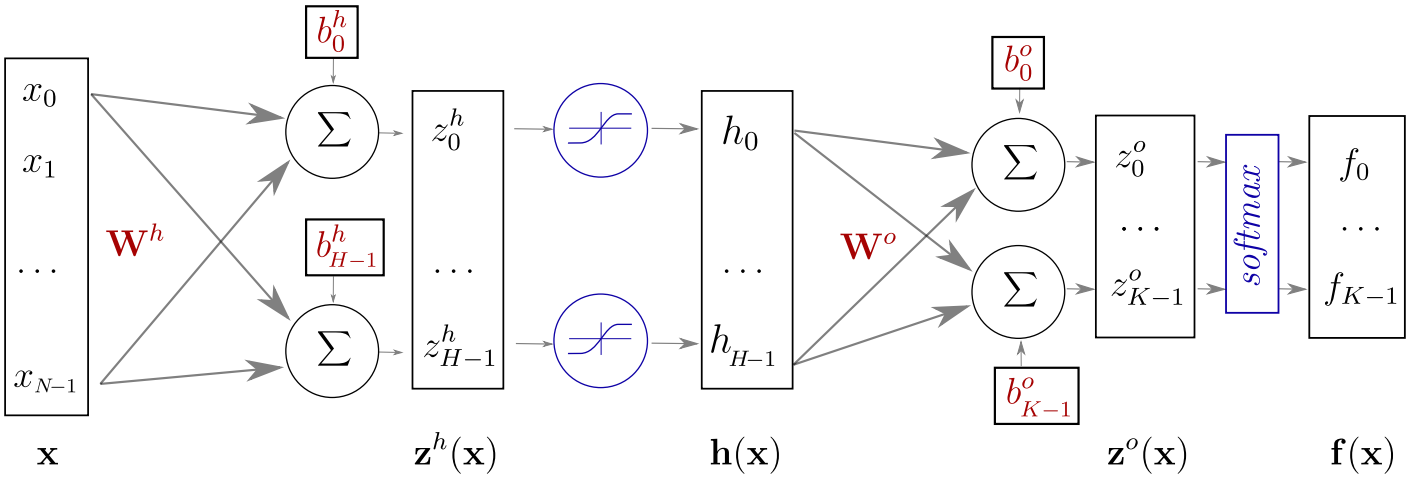
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) =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

One Hidden Layer Network

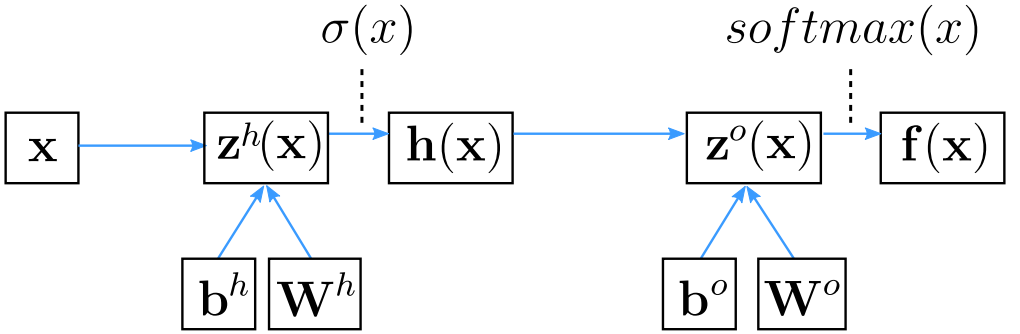


- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) =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

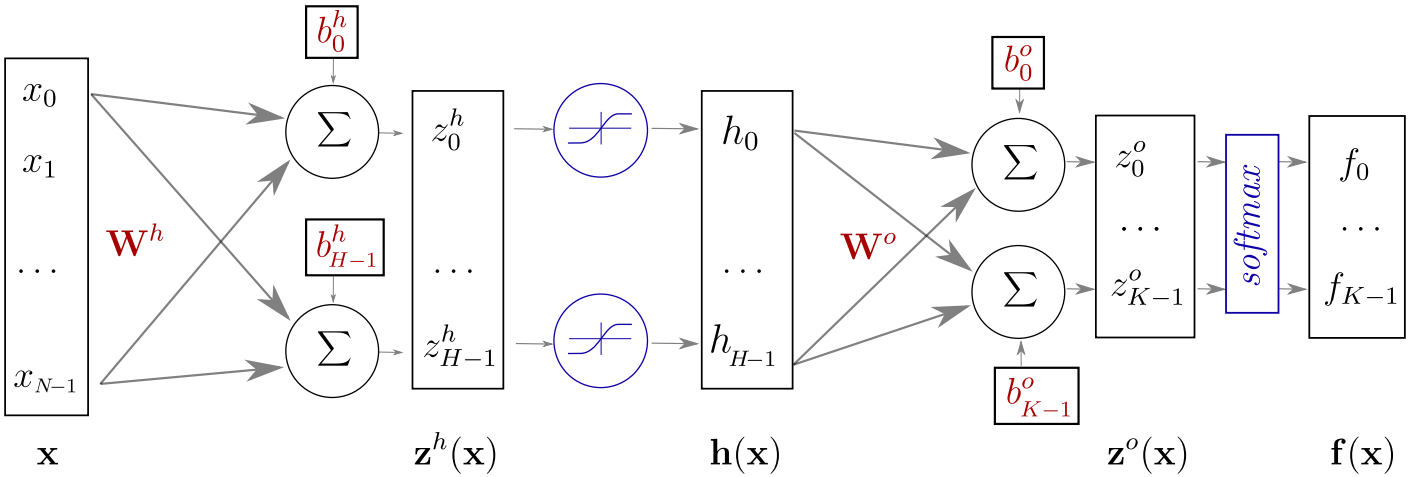
One Hidden Layer Network



Alternate representation



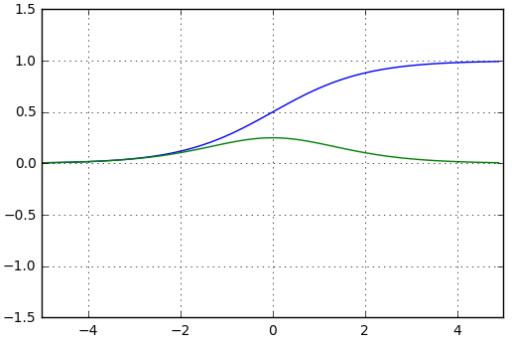
One Hidden Layer Network



Keras implementation

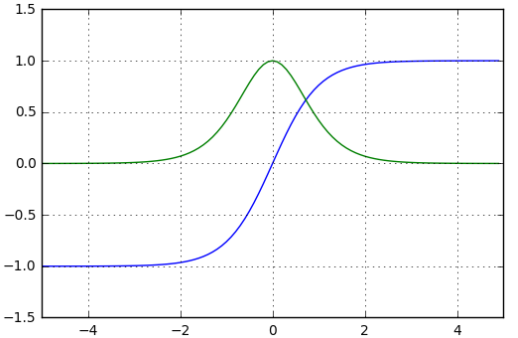
```
model = Sequential()
model.add(Dense(H, input_dim=N))  # weight matrix dim [N * H]
model.add(Activation("tanh"))
model.add(Dense(K))               # weight matrix dim [H x K]
model.add(Activation("softmax"))
```

Element-wise activation functions



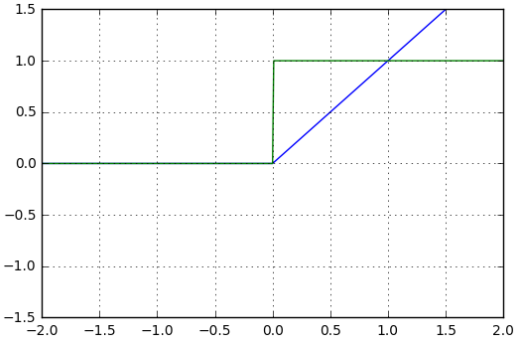
$$\text{sigm}(x) = \frac{1}{1 + e^{-x}}$$

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



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

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



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

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

- blue: activation function
- green: derivative

Softmax function

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

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

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
- $p(Y = c|X = \mathbf{x}) = \mathrm{softmax}(\mathbf{z}((\mathbf{x}))_c$
- the pre-activation vector $\mathbf{z}(\mathbf{x})$ is often called "the logits"

Training the network

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

Training the network

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 $s \in S$:

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

Training the network

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 $s \in S$:

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

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

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

Training the network

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 $s \in S$:

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

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

Stochastic Gradient Descent

Initialize θ randomly

For E epochs perform:

- Randomly select a small batch of samples $(B \subset S)$

Stochastic Gradient Descent

Initialize θ randomly

For E epochs perform:

- Randomly select a small batch of samples $(B \subset S)$
- Compute gradients: $\Delta = \nabla_\theta L_B(\theta)$

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

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

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

Computing Gradients

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

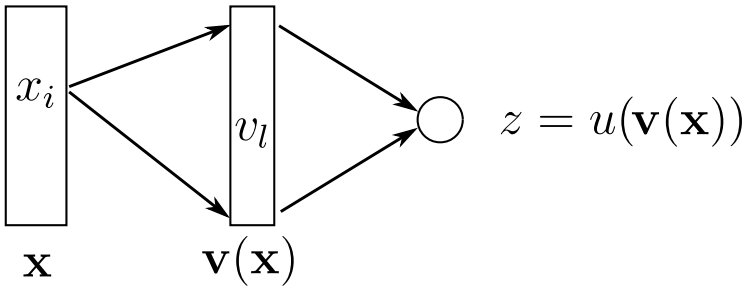
Output bias: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial b^o_i}$

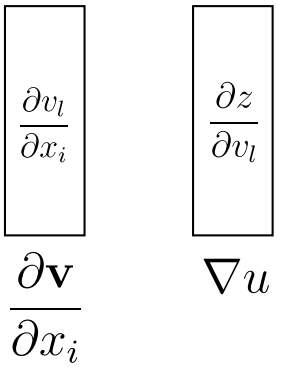
Hidden Weights: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial W^h_{i,j}}$

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"

Computing Gradients

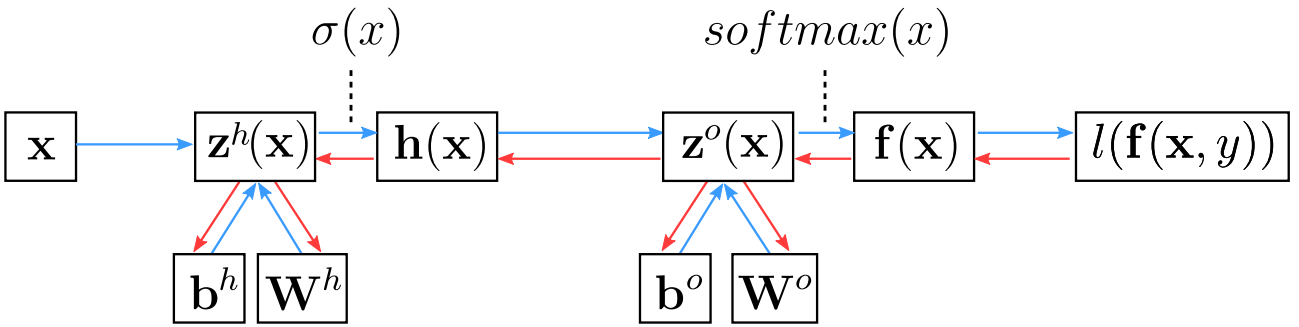




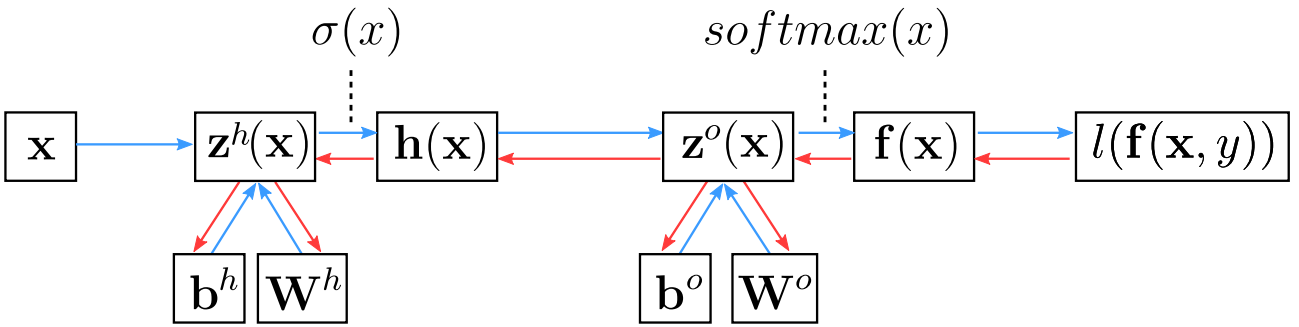
chain-rule

$$\frac{\partial z}{\partial x_i} = \sum_l \frac{\partial z}{\partial v_l}\frac{\partial v_l}{\partial x_i} = \nabla u . \frac{\partial \mathbf{v}}{\partial x_i}$$

Backpropagation



Backpropagation



Compute partial derivatives of the loss

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

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

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

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

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

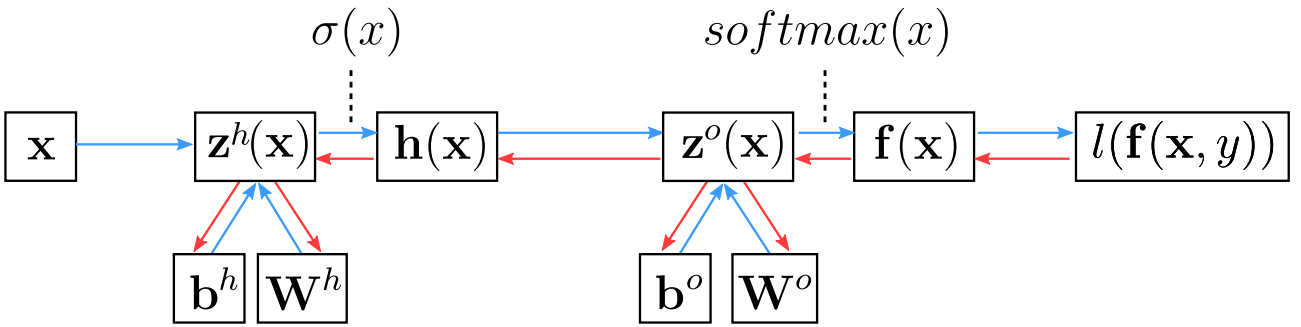
$$\frac{\partial l(\mathbf{f}(\mathbf{x},y))}{\partial \mathbf{f}(\mathbf{x})_i} = \frac{-1_{y=i}}{\mathbf{f}(\mathbf{x})_y} \qquad \mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o(\mathbf{x}))$$

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

$$\mathbf{e}(y) = \begin{bmatrix} 0 \\ \dots \\ 1 \\ \dots \\ 0 \end{bmatrix} y$$

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

Backpropagation

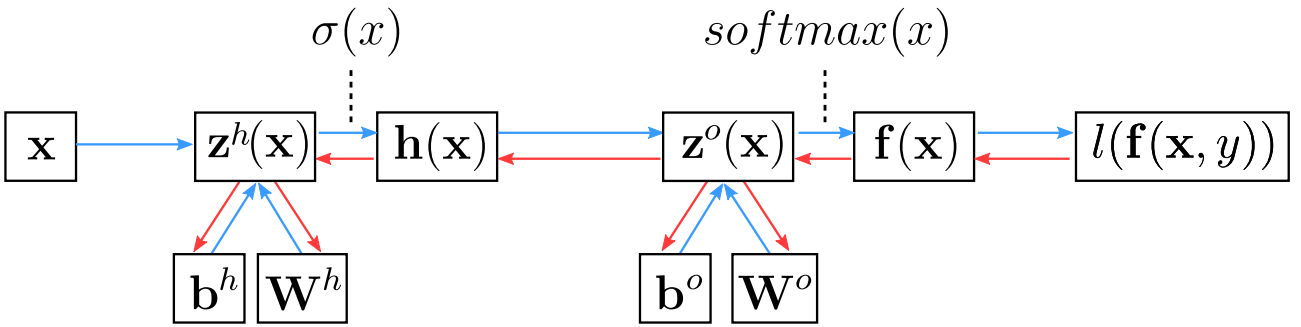


Gradients

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

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

Backpropagation



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

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

Backprop gradients

Compute activation gradients

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

Backprop gradients

Compute activation gradients

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

Compute layer params gradients

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

Backprop gradients

Compute activation gradients

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

Compute layer params gradients

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

Compute prev layer activation gradients

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

Loss, Initialization and Learning Tricks

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

Continous 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

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

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

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: $w \sim \mathcal{N}(0, 0.01)$

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

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 $\longrightarrow$ divergence
  - Too low $\longrightarrow$ slow convergence

SGD learning rate

- Very sensitive:
  - Too high $\rightarrow$ 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

SGD learning rate

- Very sensitive:
  - Too high $\rightarrow$ 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

SGD learning rate

- Very sensitive:
  - Too high $\rightarrow$ 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

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

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

Momentum

Accumulate gradients across successive updates:

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

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

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

Better at handling changes in gradient direction.

Alternative optimizers

- SGD (with Nesterov momentum)
  - Simple to implement
  - Very sensitive to initial value of η
  - Need learning rate scheduling

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 0.001 often works well enough
  - No scheduling required
  - Good default choice of optimizer

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 0.001 often works well enough
  - No scheduling required
  - Good default choice of optimizer
- Active area of research

Libraries & Frameworks

- Automatic differentiation
  - Theano
  - **TensorFlow**
  - MXnet
  - CNTK

Libraries & Frameworks

- Automatic differentiation
  - Theano
  - **TensorFlow**
  - MXnet
  - CNTK
- Higher level
  - **Keras**
  - Lasagne

Libraries & Frameworks

- Automatic differentiation
  - Theano
  - **TensorFlow**
  - MXnet
  - CNTK
- Higher level
  - **Keras**
  - Lasagne
- Dynamic and high level
  - Torch & PyTorch
  - Chainer
  - MinPy ...

Lab 1: Room C48-C49 in 15min!