Lecture 12

Introduction to deep learning

Machine Learning

Andrey Filchenkov

15.05.2018

Lecture plan

- Deep vs shallow networks
- Compositionality
- Representation learning
- Modularity
- Computational graph
- DNN best practices
- Modern architectures

- The presentation is prepared with materials of
  - K.V. Vorontsov “Machine Leaning”,
  - E. Gaaves "UVA Deep Learning Course“,
  - A. Erdem "Foundations of machine learning“,
  - F.F. Li and A. Karpathy's course "Convolutional Neural Networks for Visual Recognition"

Lecture plan

- Deep vs shallow networks
- Compositionality
- Representation learning
- Modularity
- Computational graph
- DNN best practices
- Modern architectures

Lecture plan

- Deep vs shallow networks
- Compositionality
- Representation learning
- Modularity
- Computational graph
- DNN best practices
- Modern architectures

First AI winter

1943 Neuron

1957 Perceptron

1969 “Perceptrons” by Minski and Papert

Despite rise in interest, simple counterexamples and the lack of results in contrast to the high expectations caused the **1st AI winter**. Symbolic reasoning took “the throne”.

Second AI winter

1974 Backpropagation algorithm

1980 CNNs

1982 RNNs

1991 “Vanishing gradient problem”

Despite earlier success, it was hard to learn networks with more than two layers, and many tricks were required.

It caused 2nd **AI winter**, SVM took “the throne”.

What now?

Since 2012, machine learning is getting focused on deep networks, AI and ML are usually referred to as deep learning.

State-of-the art in:

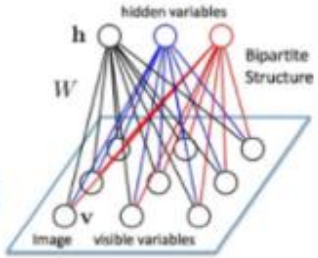
- image processing
- speech recognition
- image generation
- natural language processing and understanding
- learning strategies for games
- …

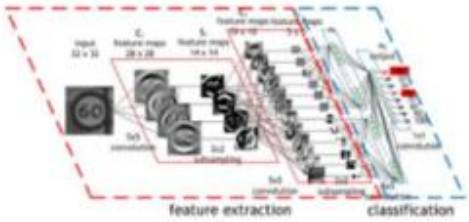
Why now?

Huge datasets

Powerful hardware

New algorithms





Why to go deep?

- Some functions cannot be efficiently represented (in terms of number of tunable elements) by architectures that are too shallow
- Functions that can be compactly represented by a depth k architecture might require an exponential number of computational elements to be represented by a depth $k-1$ architecture
- Deep representations might allow non-local generalization and comprehensibility
- Deep learning gets state of the art results in many fields (vision, audio, NLP, etc.)!

Three ideas

- (Hierarchical) Compositionality
  - Cascade of non-linear transformations
  - Multiple layers of representations
- End-to-End Learning
  - Learning (goal-driven) representations
  - Learning to feature extract
- Distributed Representations
  - Groups of neurons work together
  - Everything is a module

Lecture plan

- Deep vs shallow networks
- Compositionality
- Representation learning
- Modularity
- Computational graph
- DNN best practices
- Modern architectures

Linear non-separability

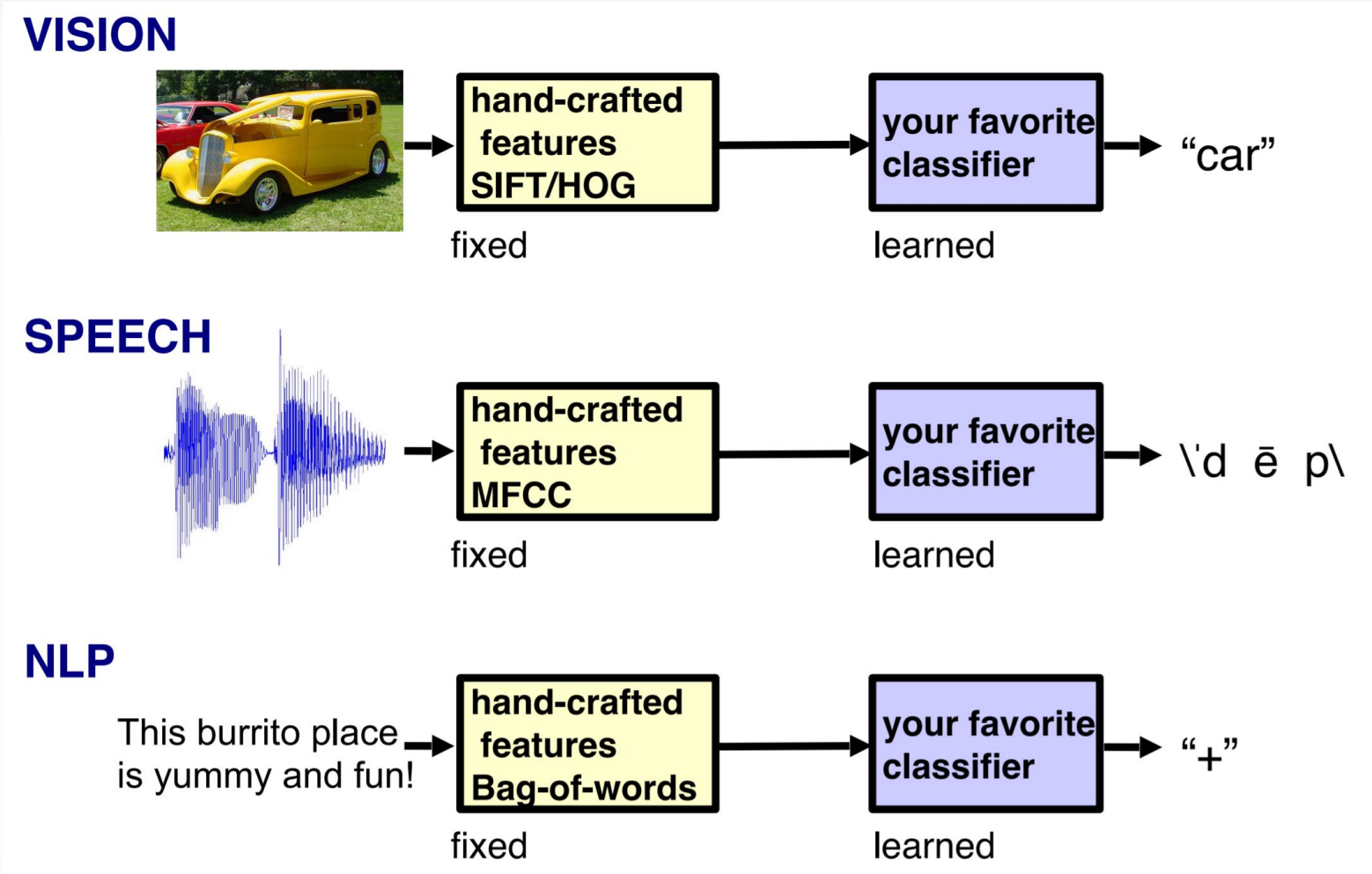
- $X = (x_1, x_2, \ldots, x_n) \in R^d$
- Given the n points there are in total 2
- n dichotomies
- Only about d are linearly separable
- With $n > d$ the probability X is linearly separable converges to 0 very fast
- The chances that a dichotomy is linearly separable is very small

Which features to use?

- Non-linear

- Invariant (but not too invariant)
- Repeatable (but not bursty)
- Discriminative (but not too class-specific)
- Robust (but sensitive enough)

Traditional machine learning



Hierarchical compositionality

VISION

pixels → edge → texton → motif → part → object

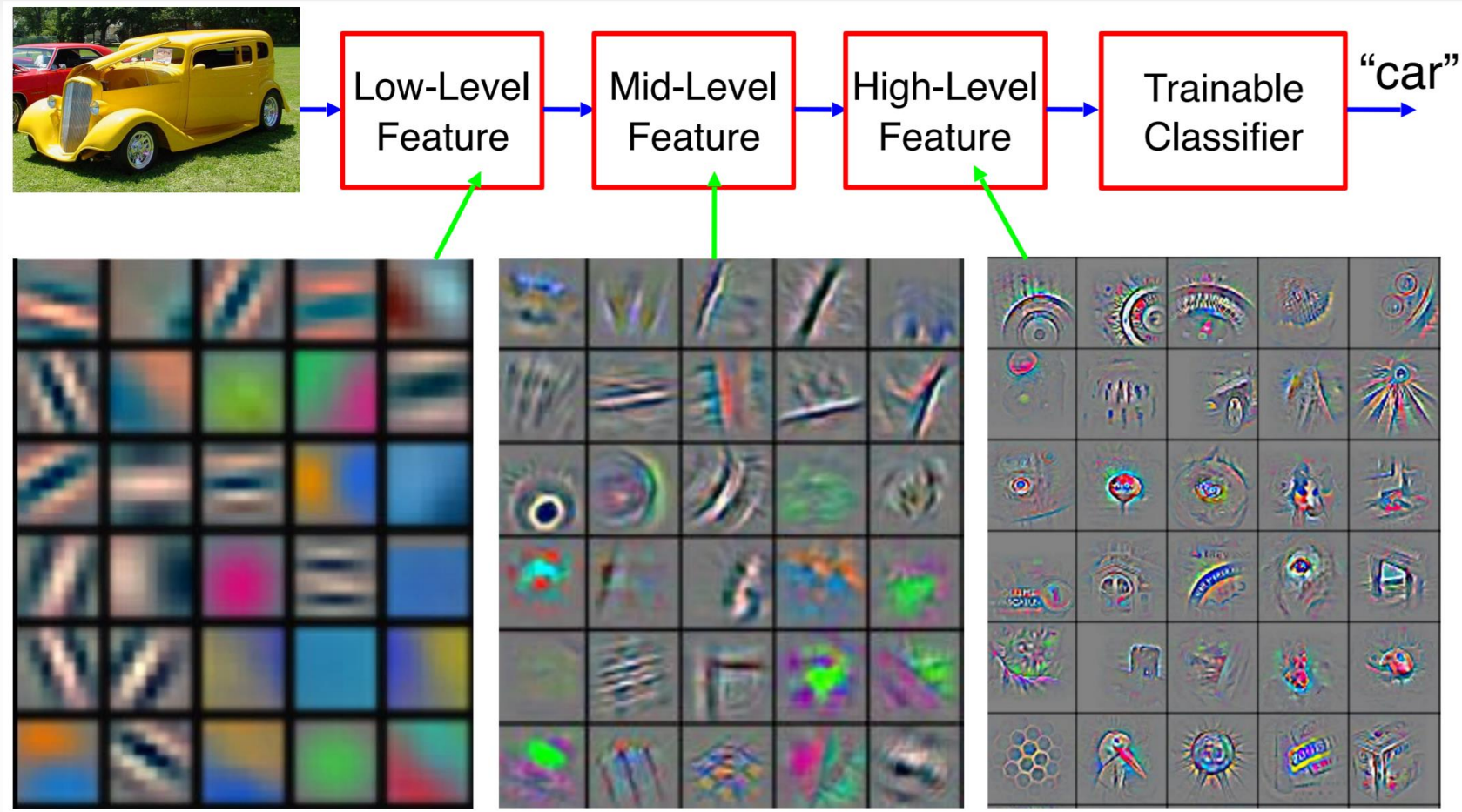
SPEECH

sample → spectral band → formant → motif → phone → word

NLP

character → word → NP/VP/.. → clause → sentence → story

Hierarchical compositionality for image



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

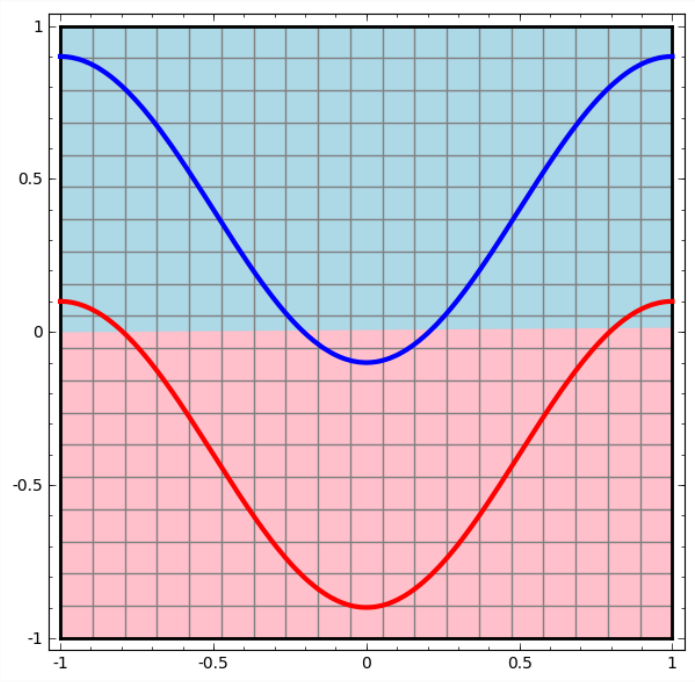
How it changes our vision?

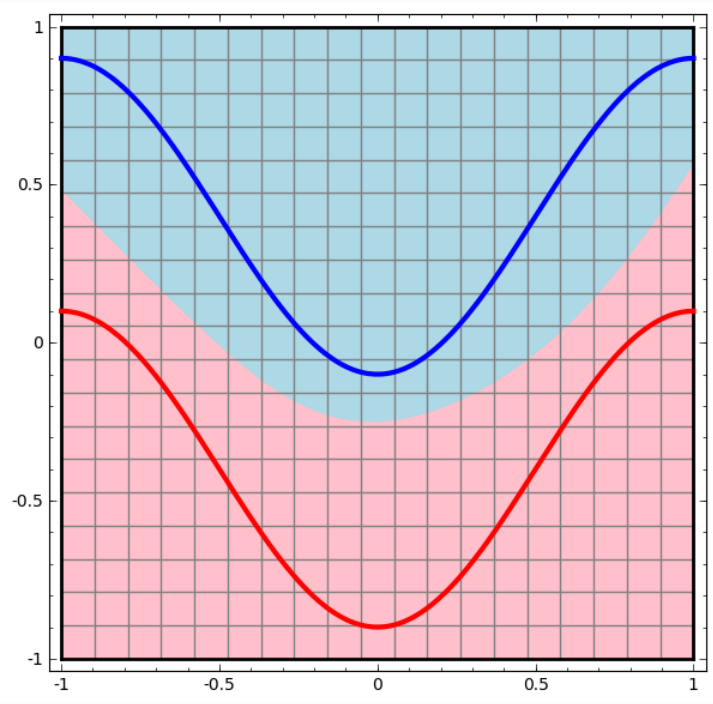
- A network is a pipeline of **modules**
- Modules produce features of higher and higher abstractions
- Input may be as raw as possible, and we **learn everything in between**
- Time spent for designing features now spent for designing architectures

Lecture plan

- Deep vs shallow networks
- Compositionality
- Representation learning
- Modularity
- Computational graph
- DNN best practices
- Modern architectures

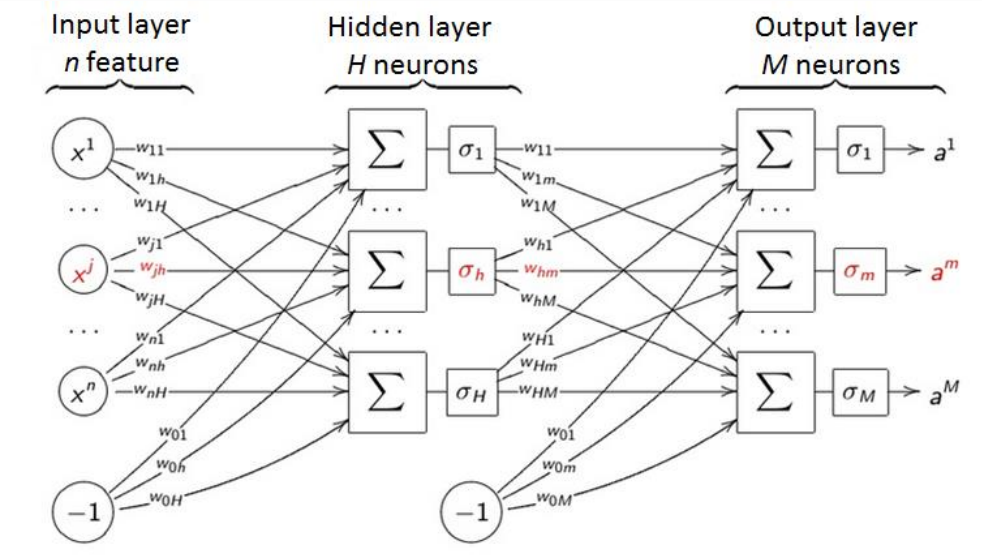
Non-linear case

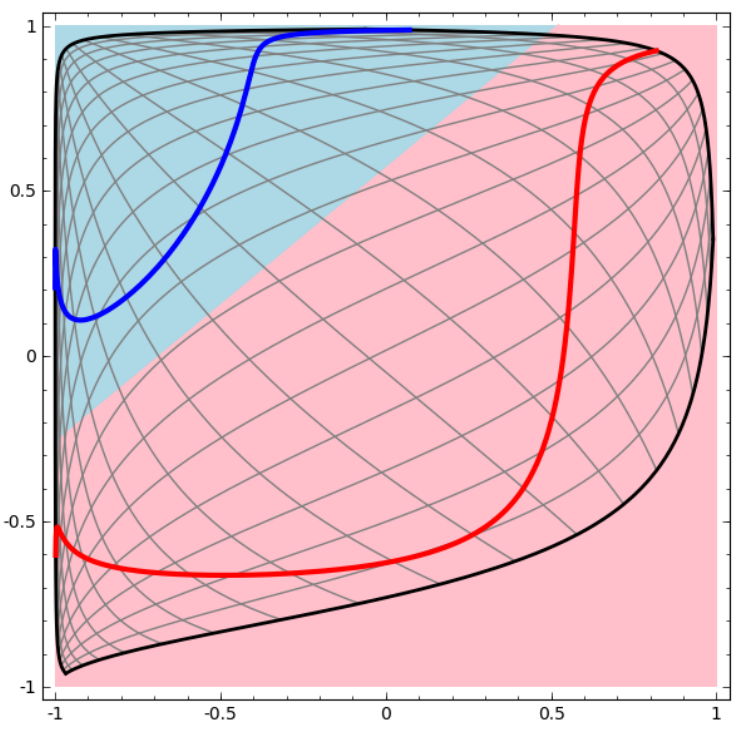




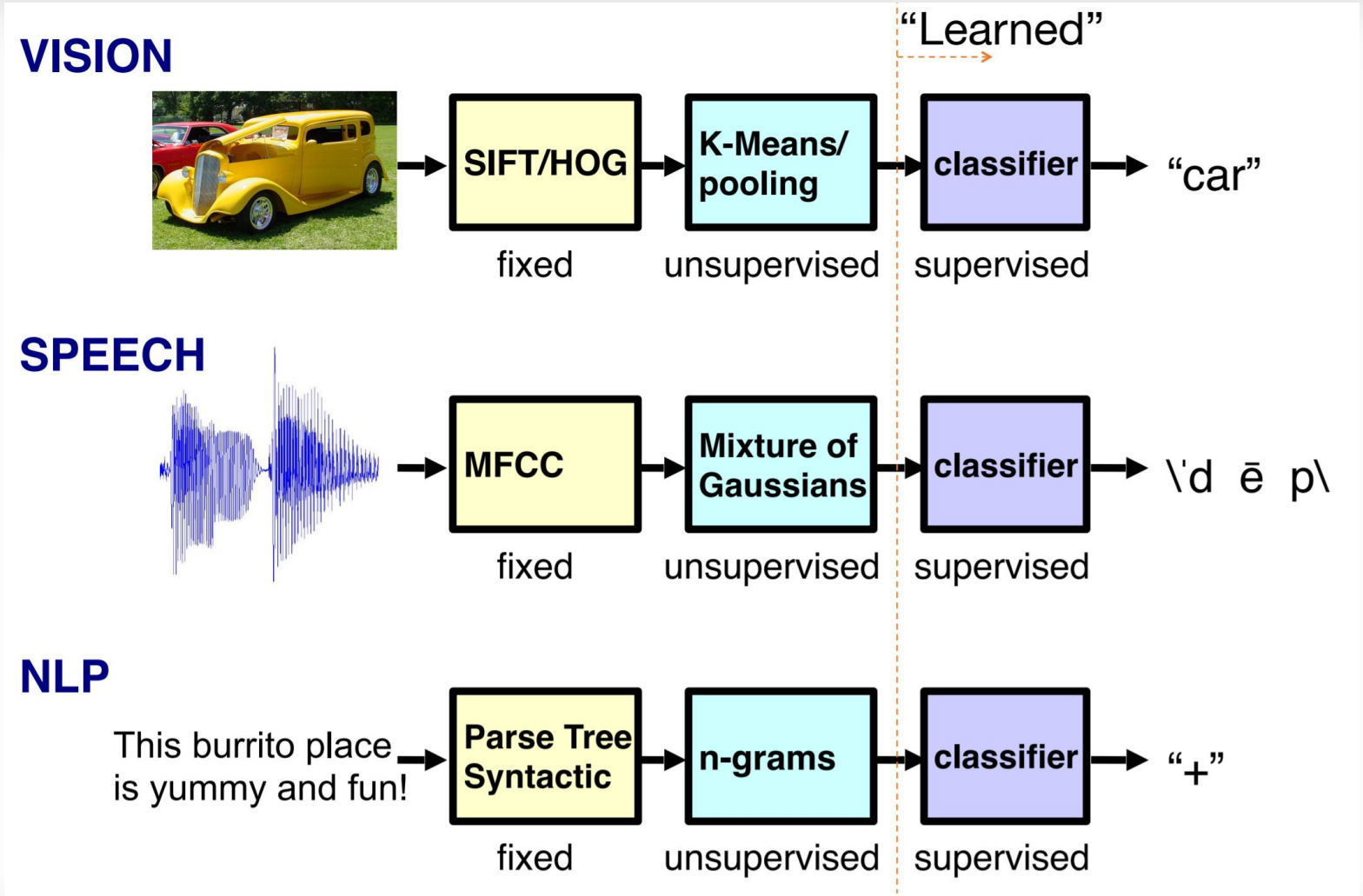
Neural network and non-lineraity

We just build a composition of neurons (as functions)

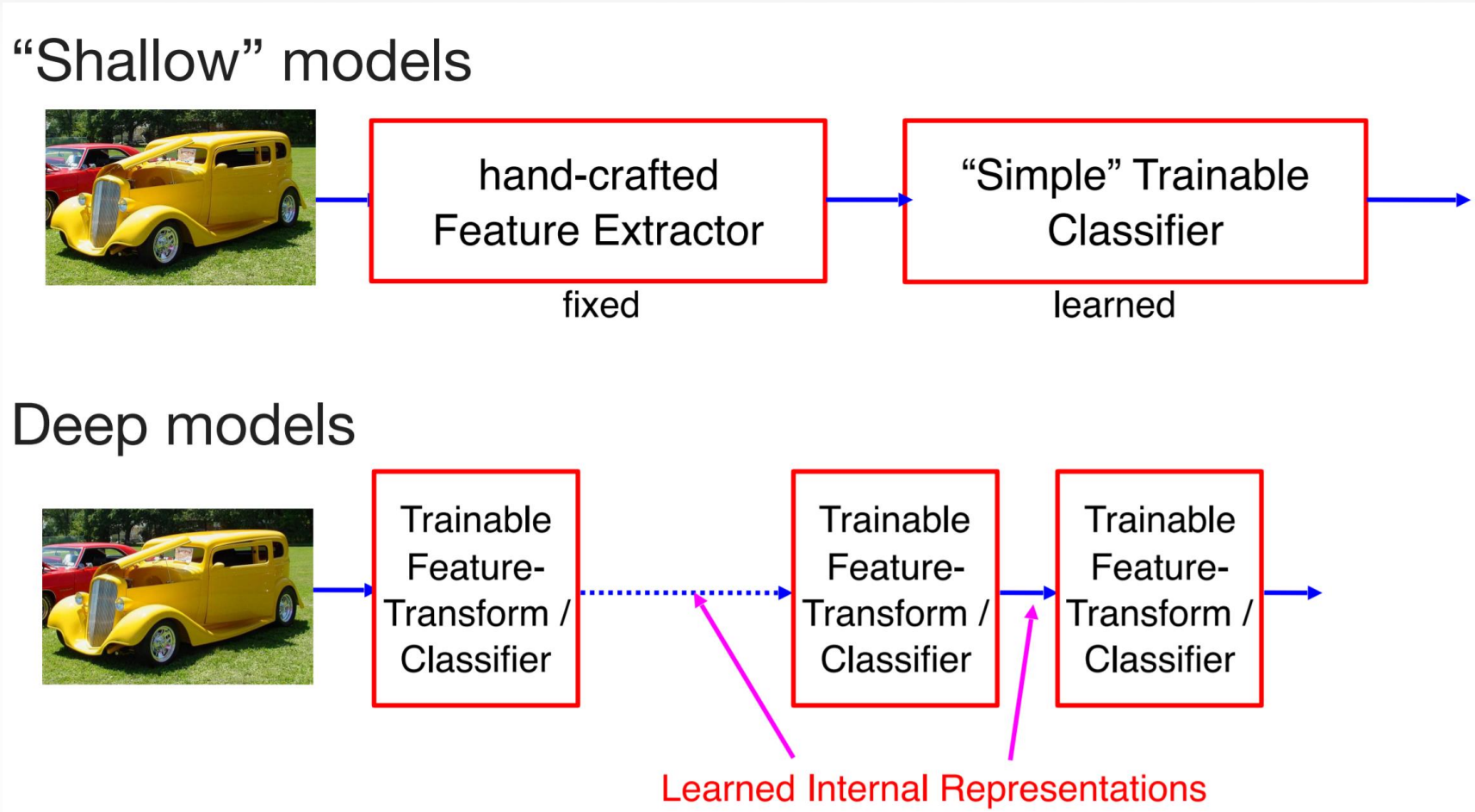




Shallow model learning



Shallow vs deep



Lecture plan

- Deep vs shallow networks
- Compositionality
- Representation learning
- Modularity
- Computational graph
- DNN best practices
- Modern architectures

Module

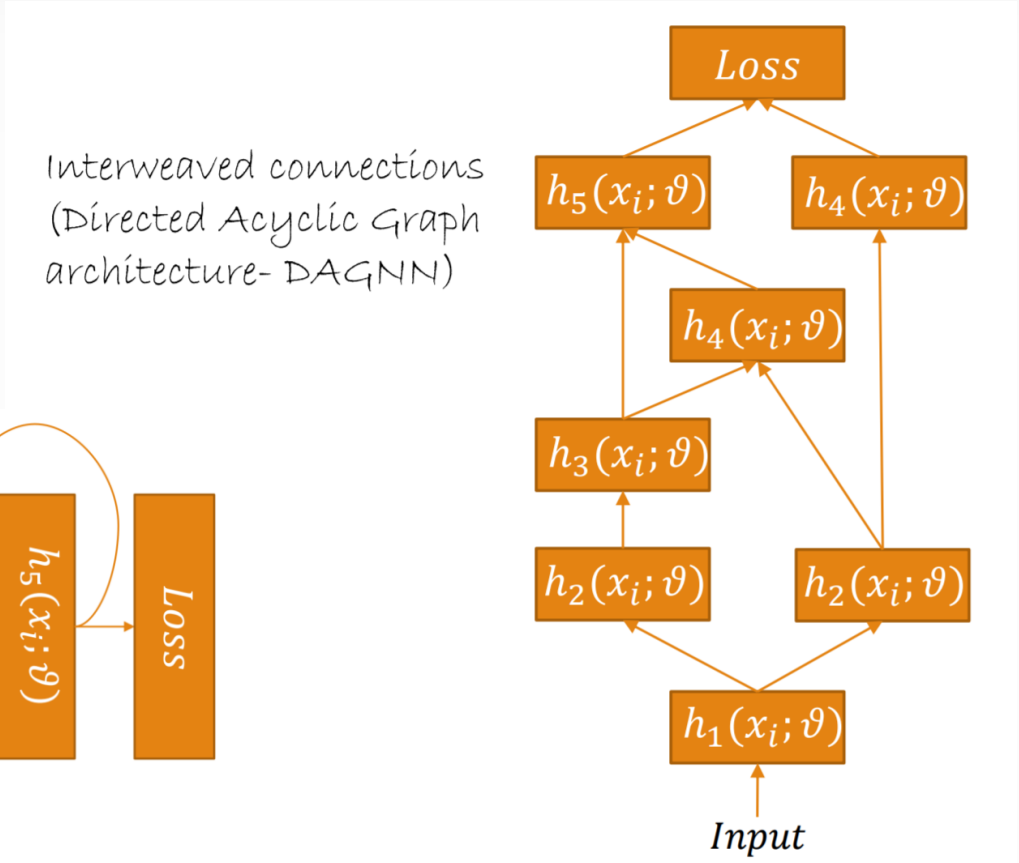
- A module is a building block for our network
- Each module is an object/function $a = h(x; \theta)$ that
  - Contains trainable parameters (θ)
  - Receives as an argument an input x
  - And returns an output a based on the activation function h …
- The activation function should be (at least) **first order differentiable (almost) everywhere**

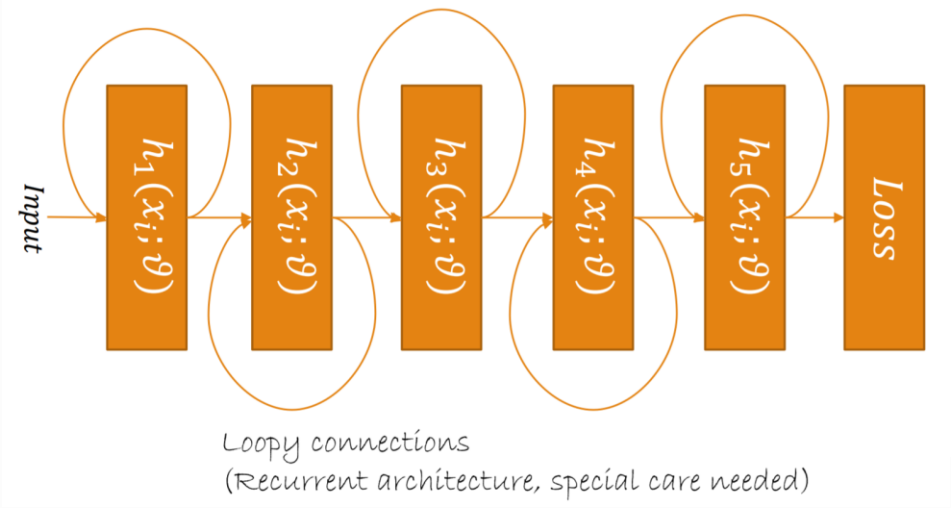
Composition of modules

- A neural network is a composition of modules (building blocks)
- Any architecture works
- If the architecture is a feedforward cascade, no special care
- If acyclic, there is right order of computing the forward computations
- If there are loops, these form recurrent connections

Examples

A network may be complicated enough

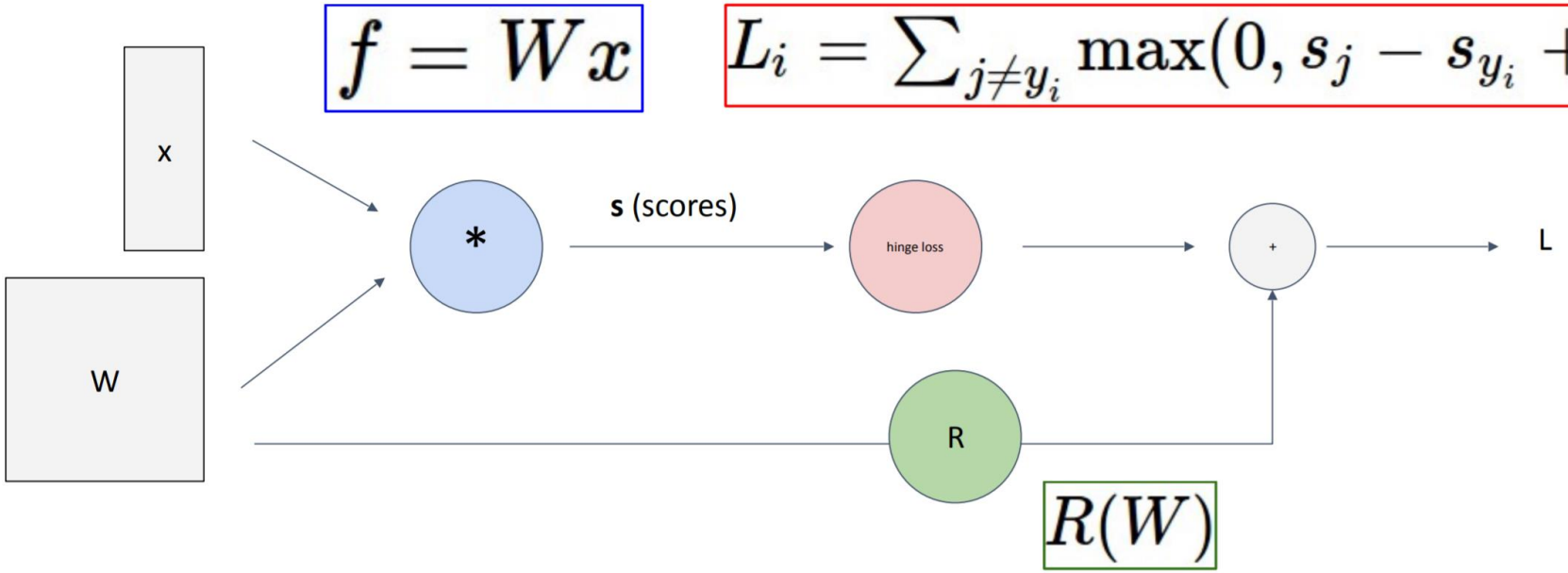




Lecture plan

- Deep vs shallow networks
- Compositionality
- Representation learning
- Modularity
- Computational graph
- DNN best practices
- Modern architectures

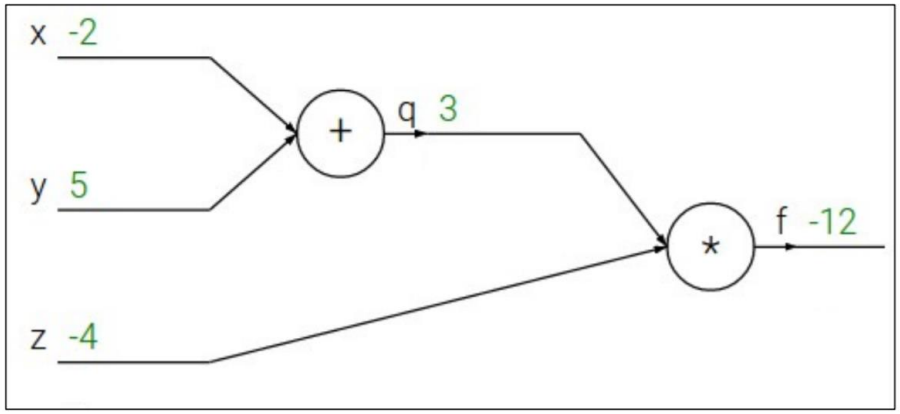
Computational graph



Example of conputations (1/13)

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



Example of computations (2/13)

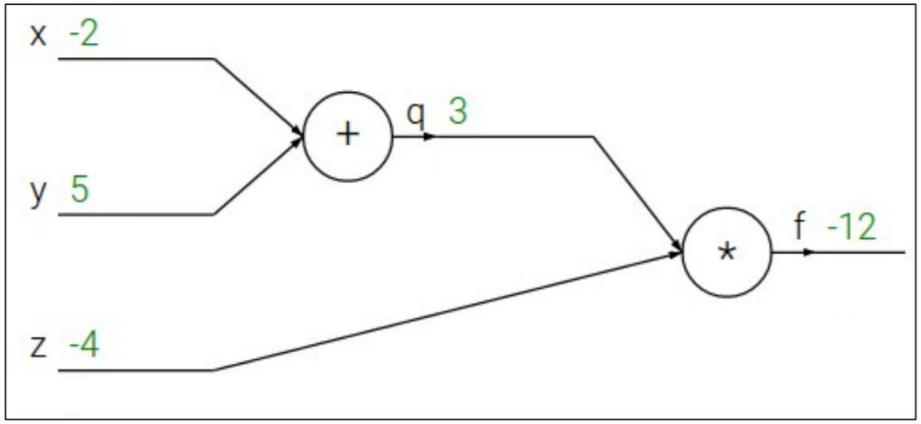
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Example of computations (3/13)

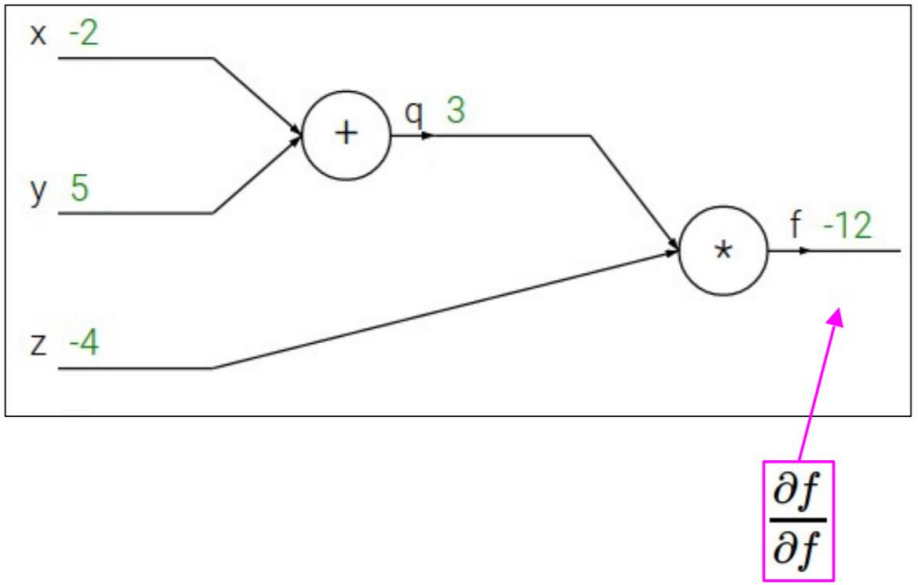
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Example of computations (4/13)

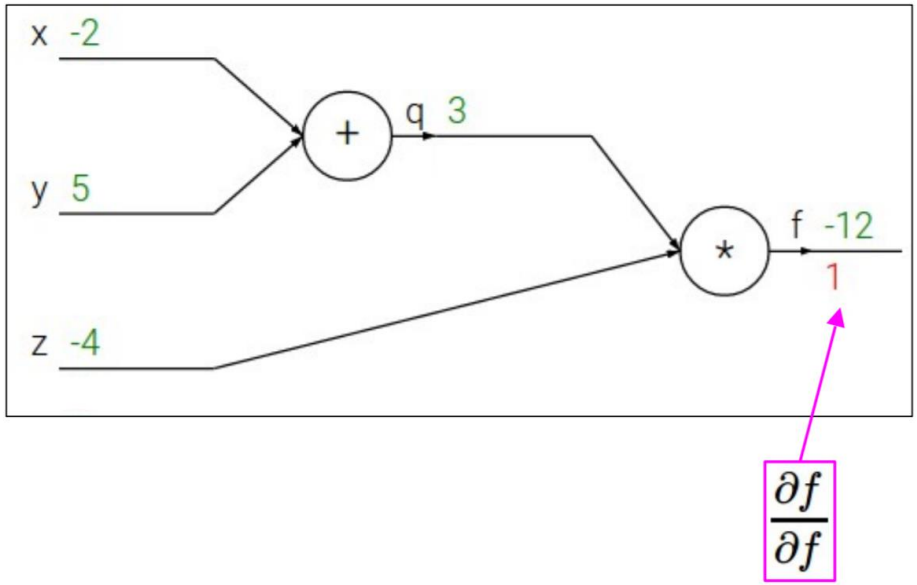
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Example of computations (5/13)

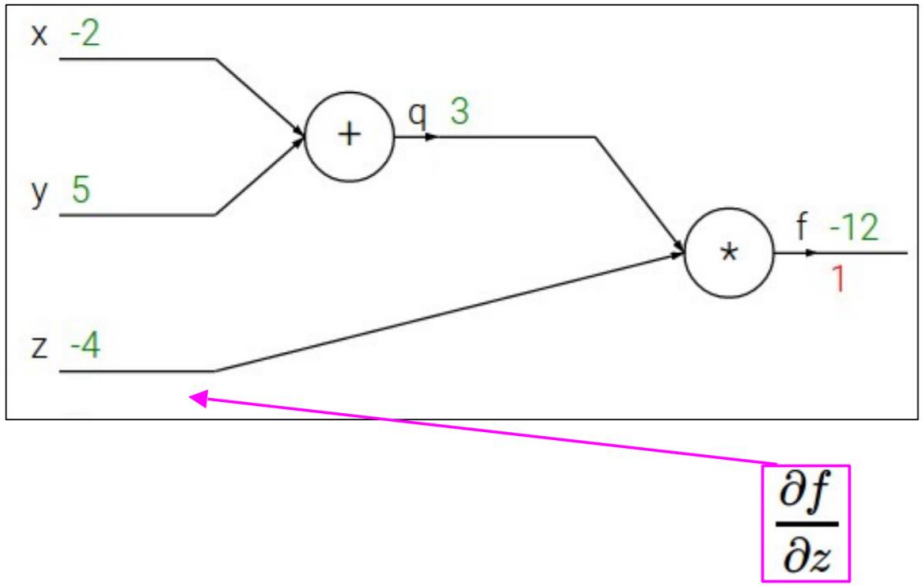
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Example of computations (6/13)

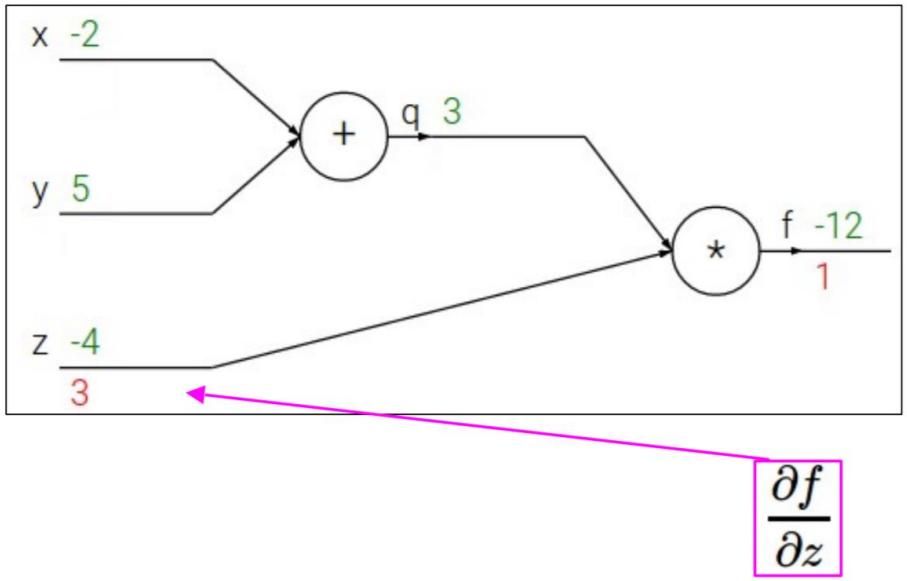
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Example of computations (7/13)

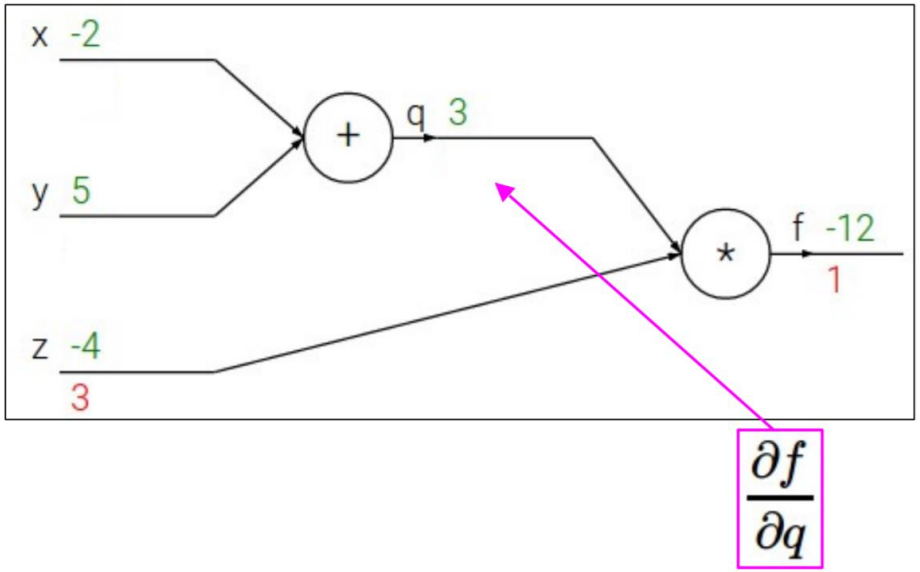
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Example of computations (8/13)

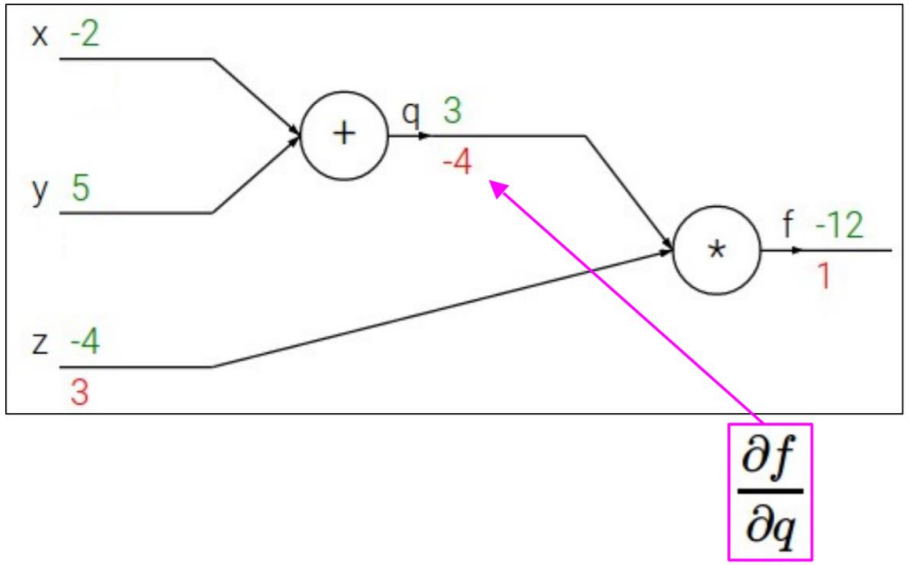
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Example of computations (9/13)

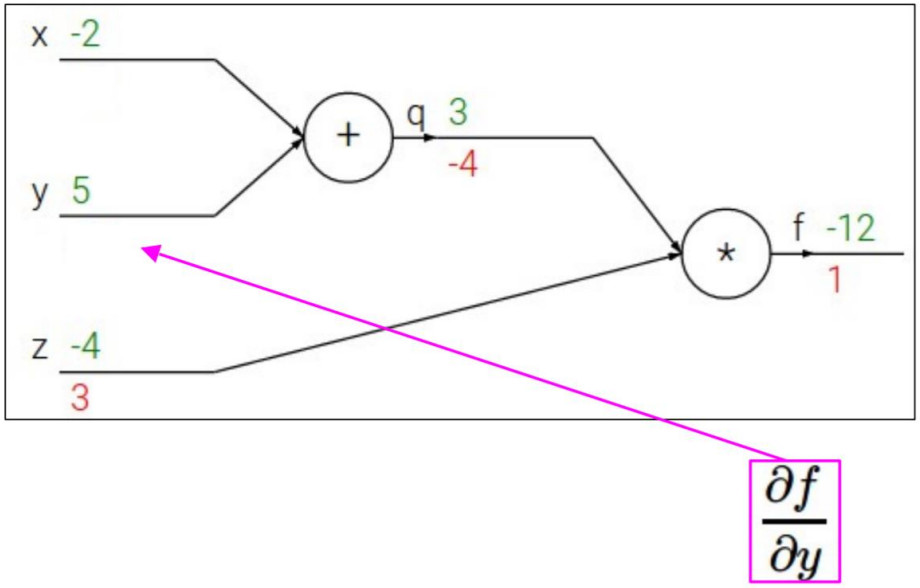
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Example of computations (10/13)

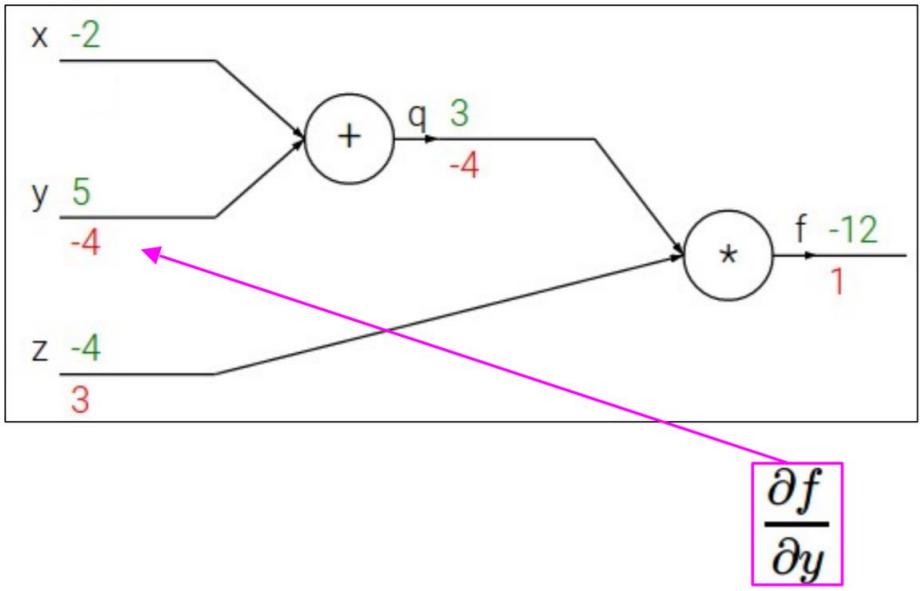
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Example of computations (11/13)

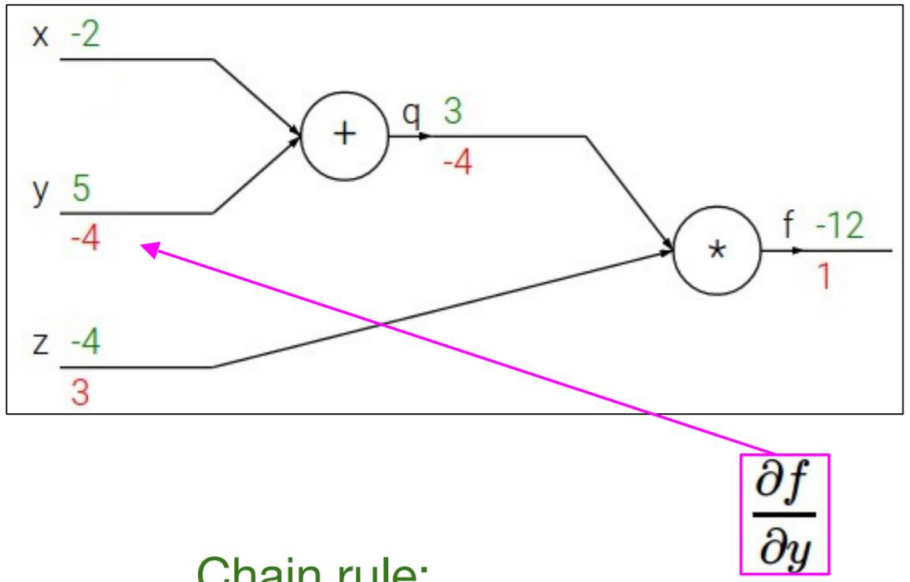
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

Example of computations (12/13)

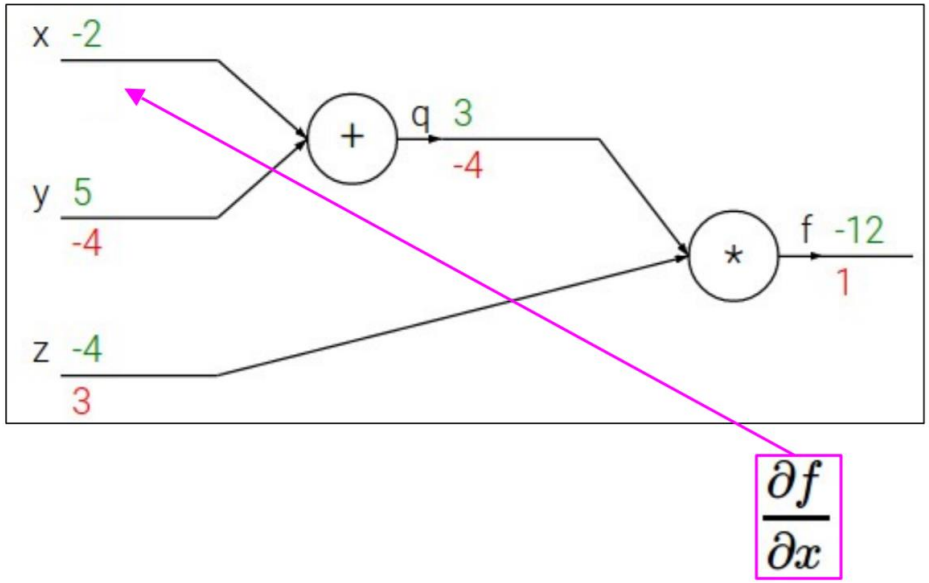
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Example of computations (13/13)

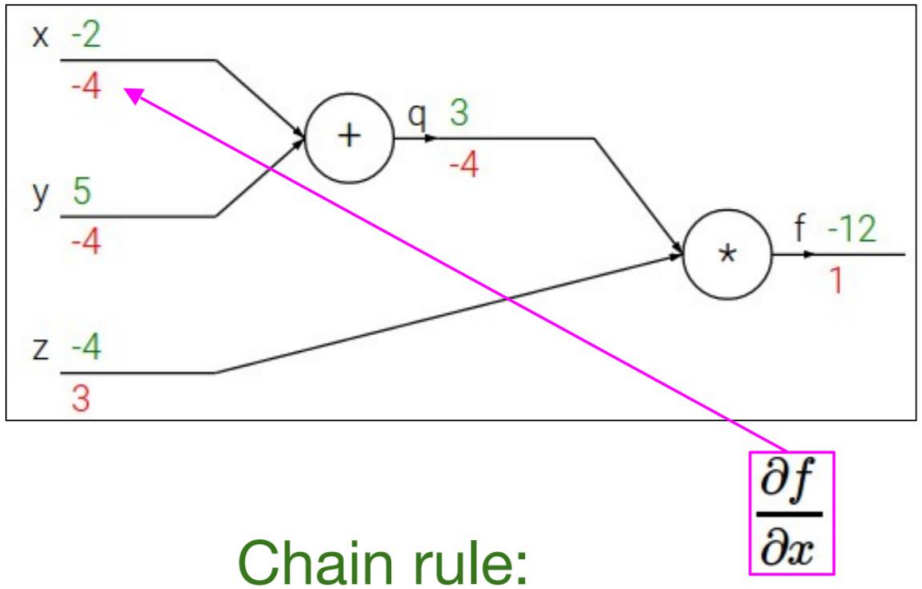
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

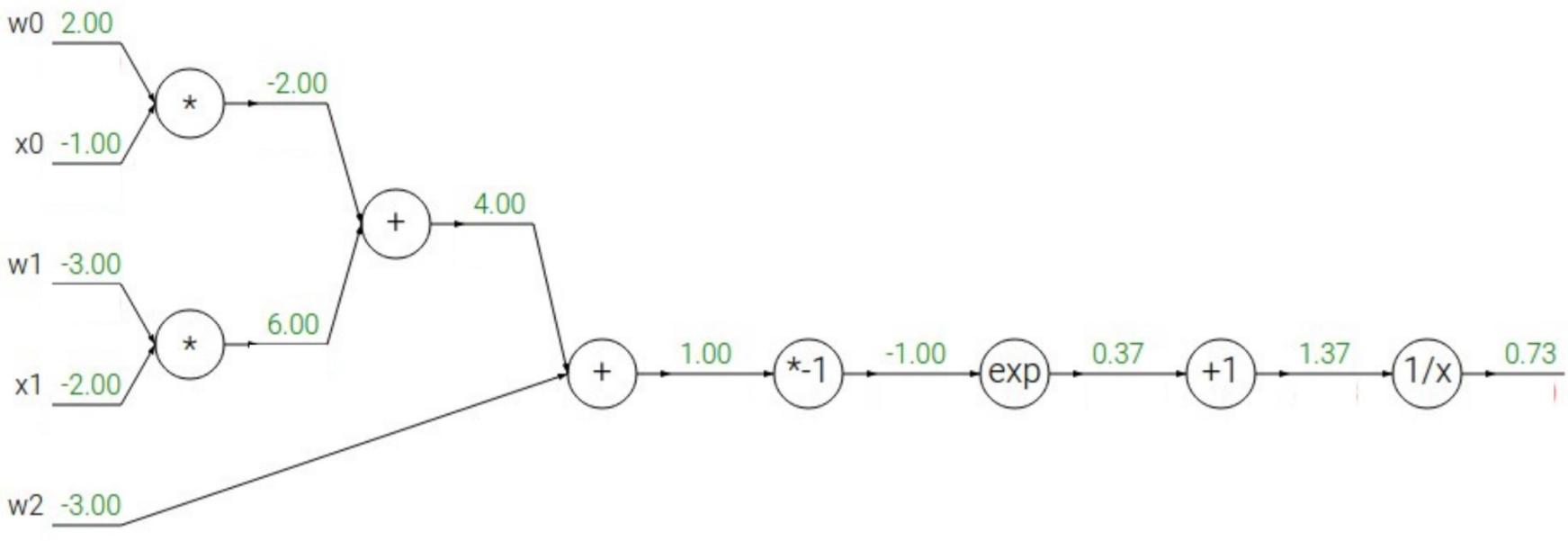


Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Another example (1/2)

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



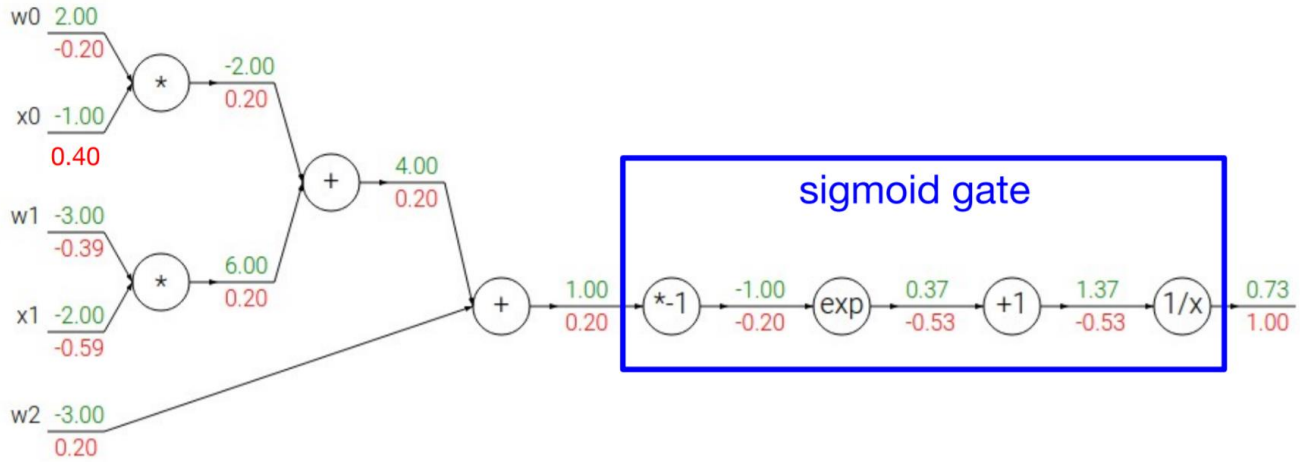
Another example (2/2)

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

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

sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



Lecture plan

- Deep vs shallow networks
- Compositionality
- Representation learning
- Modularity
- Computational graph
- DNN best practices
- Modern architectures

The problems

- A deep network has many parameters, which cause overfitting
- A deep network is trying to reveal deep features, which requires a lot of data (and causes overfitting)
- It has many modules in a raw, which cases vanishing gradient problem (the far a module is from loss, the less it learns)

Best practices

- A deep network has many parameters, use **dropout**
- A deep network is trying to reveal deep features, use **augmented data**
- It has many modules in a raw, which cases the vanishing gradient problem, use **special activation functions**

Dropout

Dropout: set the output of each hidden neuron to zero w.p. 0.5

- The neurons which are "dropped out" in this way do not contribute to the forward pass and do not participate in backpropagation
- So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights
- This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons
- It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons
- Without dropout, a network exhibits substantial overfitting
- Dropout roughly doubles the number of iterations required to converge

Data augmentation

The easiest and most common method to **reduce overfitting** on image data is to artificially **enlarge the dataset** using label-preserving transformations.

Types of data augmentation:

- Image translation
- Horizontal/vertical reflections + cropping
- Changing RGB intensities

Sigmoid disadvantages

- Always multiply with < 1, so the gradients can be small in deep networks
- The gradients at the tails flat to 0, causing no serious SGD updates
  - Overconfident, but not necessarily “correct”
  - Neurons get stuck

Tanh

- Activation function $a = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Gradient with respect to the input

$$\frac{\partial a}{\partial x} = 1 - \tanh^2(x)$$

- Similar to sigmoid, but with different output range $[-1, +1]$
- Stronger gradients, because data is centered around 0 (not 0.5)
- Less bias to hidden layer neurons as now outputs can be both positive and negative (more likely to have zero mean in the end)

ReLU

- Activation function a $= h(x) = \max(0, x)$
- Gradient with respect to the input

$$\frac{\partial a}{\partial x} = \begin{cases} 1, \text{if } x > 0, \\ 0, \text{otherwise.} \end{cases}$$

- Very popular in computer vision and speech recognition

ReLU analysis

- Much faster computations, gradients
- No vanishing or exploding problems, only comparison, addition, multiplication
- People claim biological plausibility
- Sparse activations
- No saturation

- Non-symmetric
- Non-differentiable at 0
- A large gradient during training can cause a neuron to "die". Higher learning rates mitigate the problem

Softplus

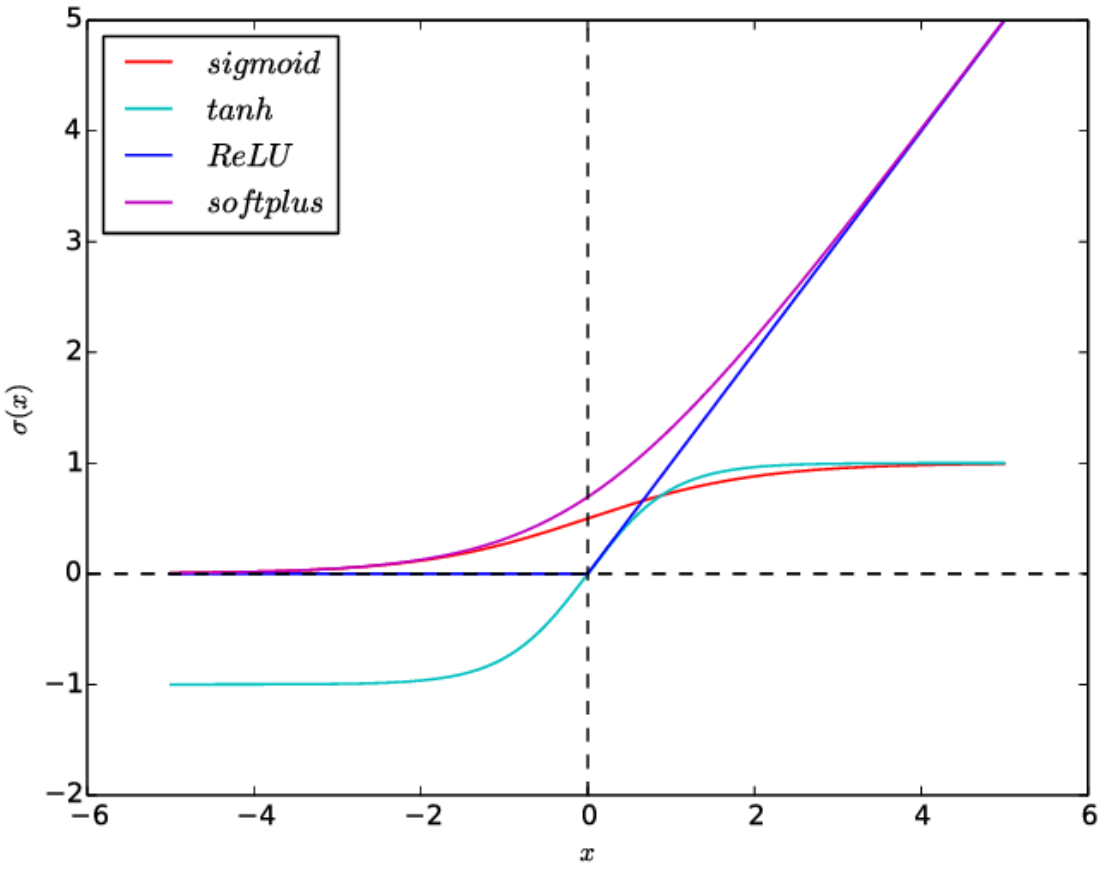
Soft approximation (softplus):

$$a = h(x) = \ln(1 + e^x)$$

- Differentiable at 0

- Slower
- Empirically, do not outperforms ReLU

Activation functions in one plot



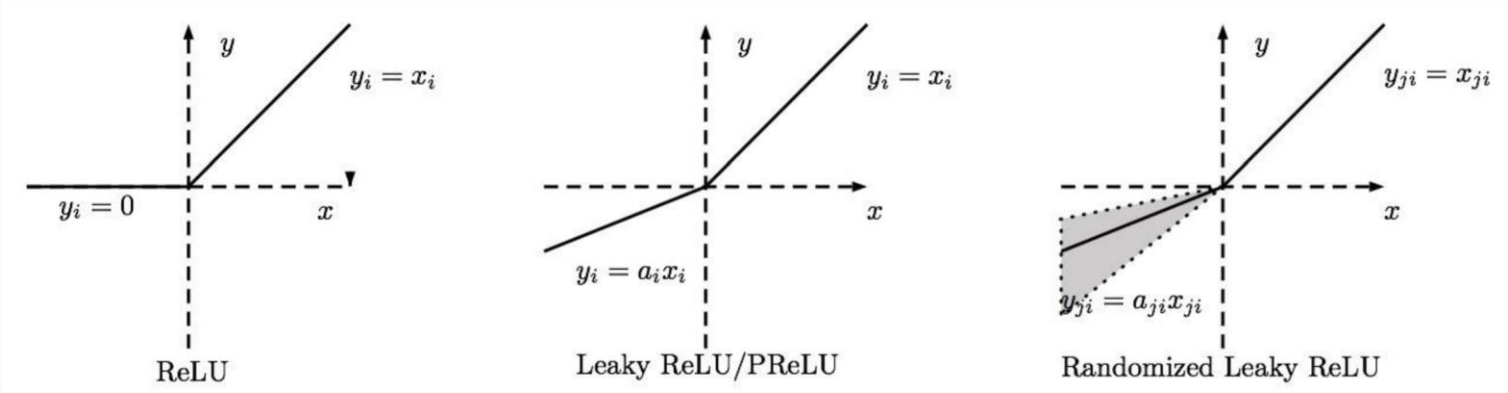
Other ReLUs

Noisy ReLU: $h(x) = \max(0, x + \varepsilon), \varepsilon \sim N(0, \sigma(x))$

Leaky ReLU: $h(x) \begin{cases} x, \text{if } x > 0, \\ 0.01x, \text{otherwise.} \end{cases}$

Parametric ReLu: $h(x) = \begin{cases} x, \text{if } x > 0 \\ \beta x, \text{otherwise} \end{cases}$

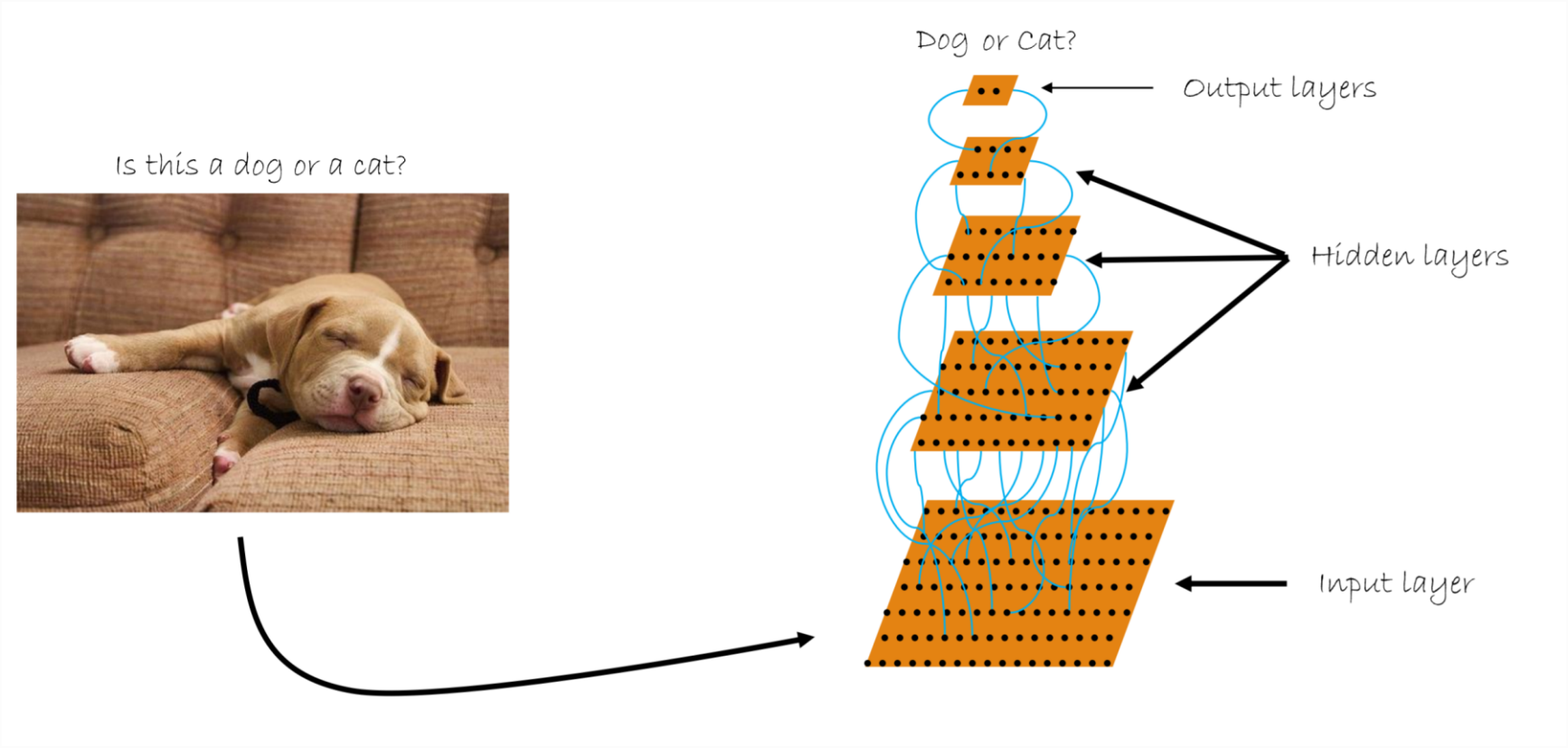
(parameter β is trainable)



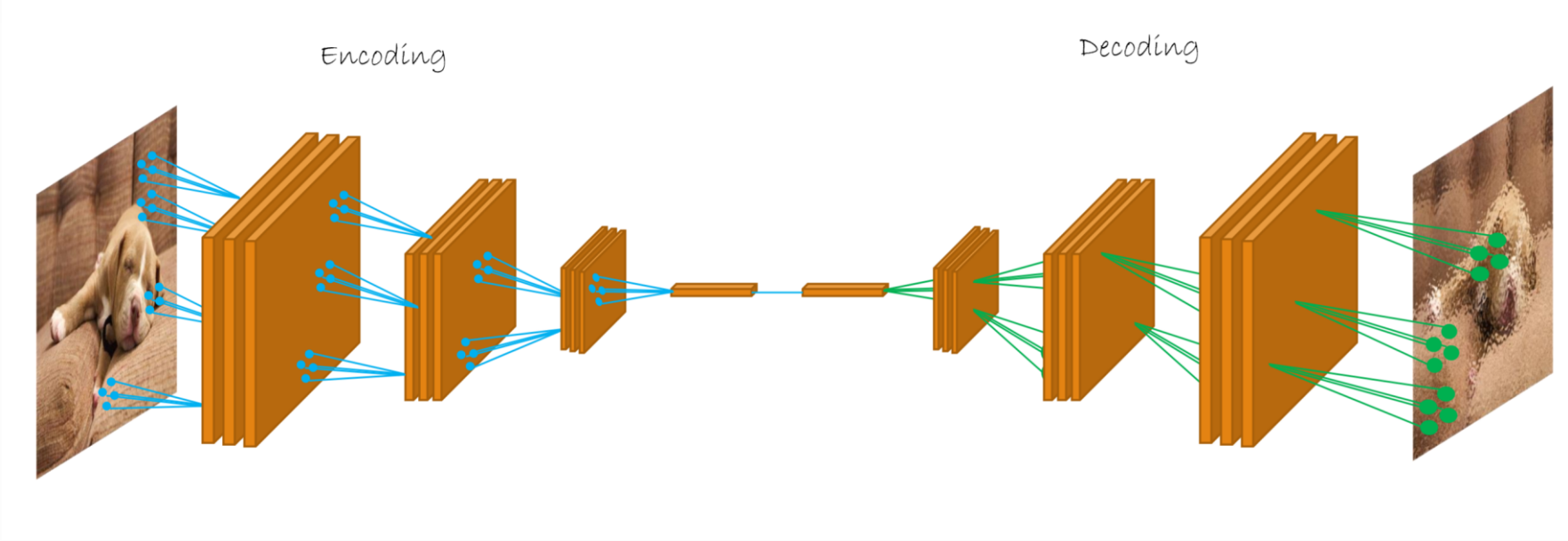
Lecture plan

- Deep vs shallow networks
- Compositionality
- Representation learning
- Modularity
- Computational graph
- DNN best practices
- Modern architectures

Convolutional neural networks



Autoencoders



Recurrent neural networks