Introduction to Deep Learning

SIG Research Meeting

Alexandru C. Serban[1,2]

[1]Digital Security
Radboud University, Nijmegen

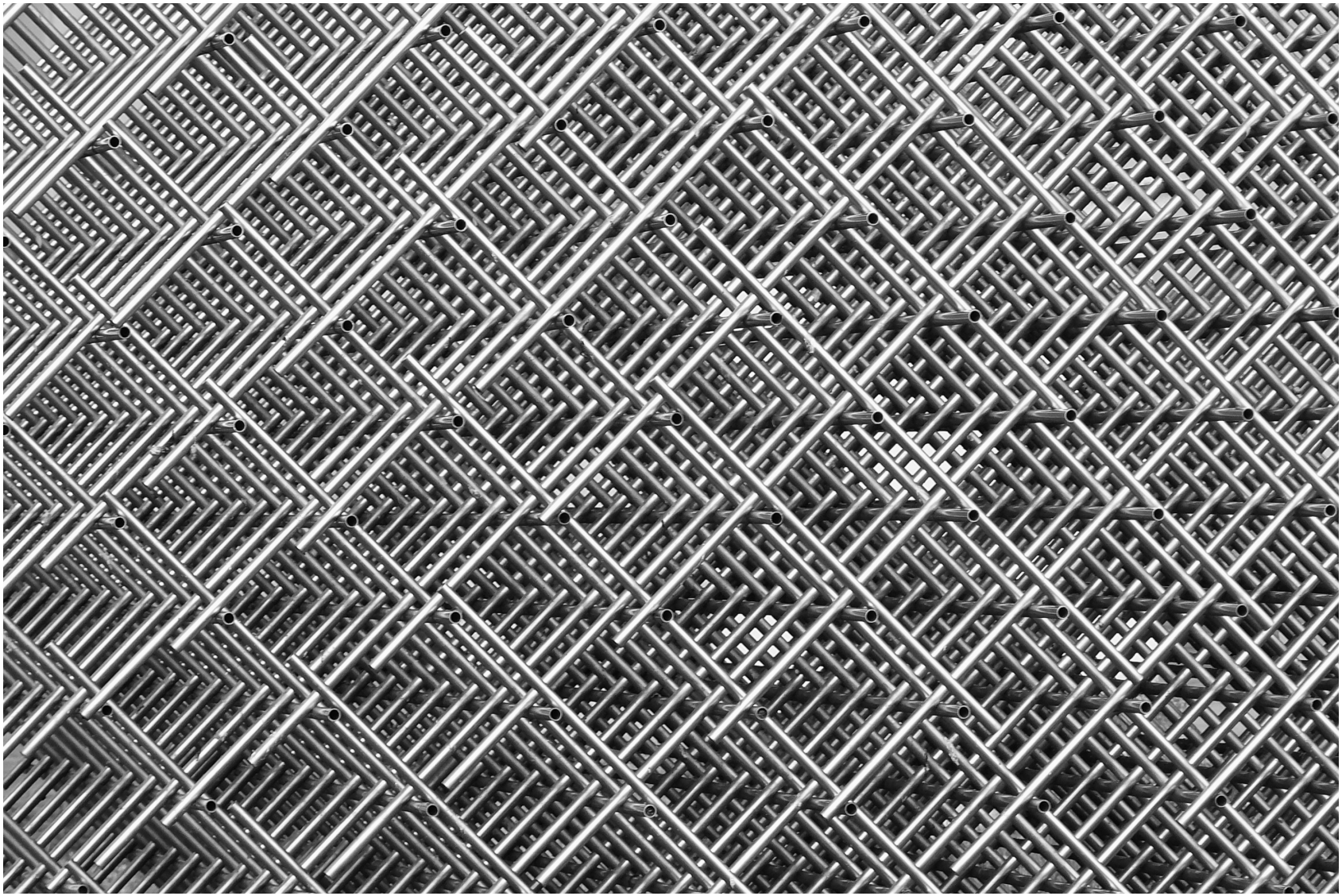
[2]Research Team
Software Improvement Group, Amsterdam

13.04.2018

Contents

Just another tool

For identifying patterns



What is a pattern?

- a *form* or *model* proposed for *imitation*[1]
- artistic, musical, literary *design* or *form*[1]
- a reliable sample of traits, acts, tendencies, or other *observable characteristics* of a . . . [1]

[1] **Merriam-Webster**.

What is a pattern?

- a *form* or *model* proposed for *imitation*[1]
- artistic, musical, literary *design* or *form*[1]
- a reliable sample of traits, acts, tendencies, or other *observable characteristics* of a . . . [1]

Definition

$f : \mathbb{R} \rightarrow \mathbb{R}$

[1] **Merriam-Webster**.

What is a pattern?

- a *form* or *model* proposed for *imitation*[1]
- artistic, musical, literary *design* or *form*[1]
- a reliable sample of traits, acts, tendencies, or other *observable characteristics* of a . . . [1]

Definition

$$f : \mathbb{R} \rightarrow \mathbb{R}$$

$$y = f(\boldsymbol{x}, \boldsymbol{\theta})$$

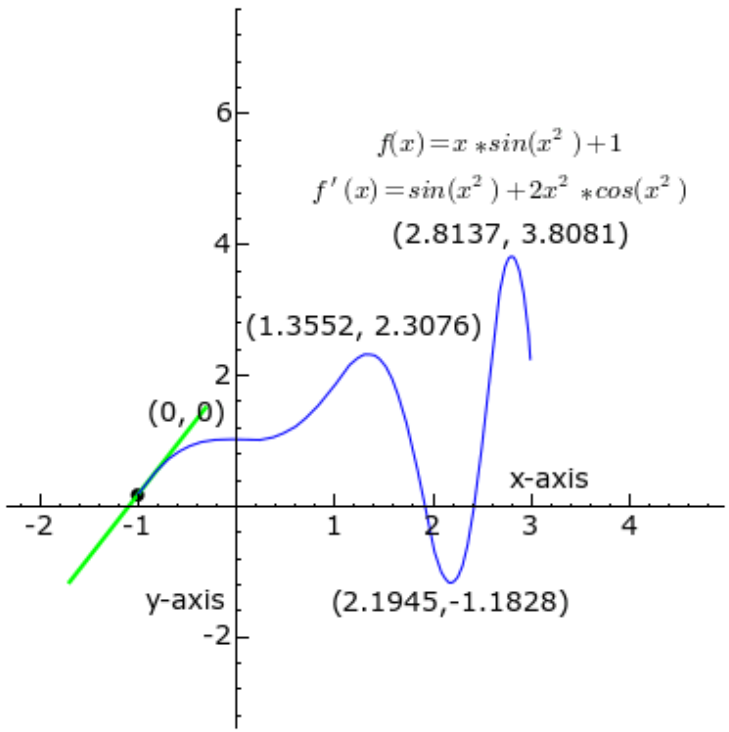
[1] **Merriam-Webster**.

Rate of change

Definition

The derivative of a function $y = f(x)$ of variable x is a measure of the *rate* at which the value y *changes* with respect to the change of x:

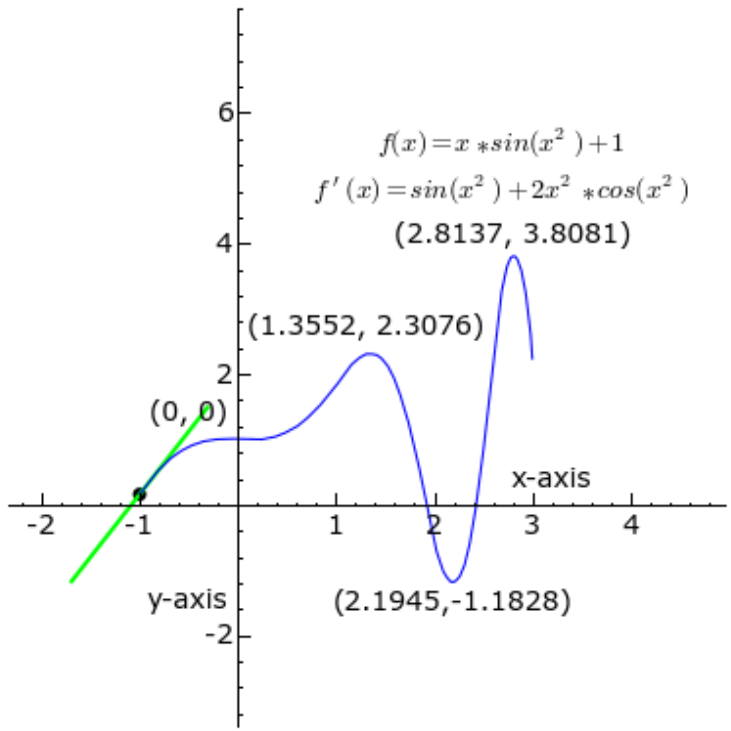
$$m = \frac{\text{change in } y}{\text{change in } x} = \frac{\Delta y}{\Delta x}$$



Searching for Minimum

Definition

A function $f(x)$ is *increasing* on any interval in which $f'(x) > 0$ and it is *decreasing* on any interval in which $f'(x) < 0$.



Gradient

$$f : \mathbb{R} \rightarrow \mathbb{R}$$

Gradient

$$f : \mathbb{R} \to \mathbb{R}$$

$$f : \mathbb{R}^n \to \mathbb{R}$$

Gradient

$$f : \mathbb{R} \rightarrow \mathbb{R}$$

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

$$y = f(x_1, x_2, \ldots x_n, \theta)$$

Gradient

$$f : \mathbb{R} \rightarrow \mathbb{R}$$

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

$$y = f(x_1, x_2, \ldots x_n, \theta)$$

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

Gradient

$$f : \mathbb{R} \rightarrow \mathbb{R}$$

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

$$y = f(x_1, x_2, \ldots x_n, \theta)$$

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

Definition

The gradient is a multi-variable generalisation of the derivative.

Things get hairy: The Jacobian Matrix

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

Things get hairy: The Jacobian Matrix

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$(y_1, y_2, \ldots y_m) = f(x_1, x_2 \ldots x_n, \theta)$$

Things get hairy: The Jacobian Matrix

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$(y_1, y_2, \ldots y_m) = f(x_1, x_2 \ldots x_n, \theta)$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Things get hairy: The Jacobian Matrix

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$(y_1, y_2, \ldots y_m) = f(x_1, x_2 \ldots x_n, \theta)$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Definition

The Jacobian Matrix is the matrix of all *first-order* partial derivatives of a *vector-valued* function.

Machines that learn - a metaphor

Machines that learn - a metaphor

Definition

Computer programs learn from *experience* E with respect to some class of *tasks* T and *performance measure* P, if its performance at tasks in T as measured by P *improves* with experience E [2].

Machines that learn - a metaphor

Definition

Computer programs learn from *experience* E with respect to some class of *tasks* T and *performance measure* P, if its performance at tasks in T as measured by P *improves* with experience E [2].

1. Task - understand language, move boxes, etc.

Machines that learn - a metaphor

Definition

Computer programs learn from *experience* E with respect to some class of *tasks* T and *performance measure* P, if its performance at tasks in T as measured by P *improves* with experience E [2].

1. Task - understand language, move boxes, etc.
2. Experience - interpret *data*

Machines that learn - a metaphor

Definition

Computer programs learn from *experience* E with respect to some class of *tasks* T and *performance measure* P, if its performance at tasks in T as measured by P *improves* with experience E [2].

1. Task - understand language, move boxes, etc.
2. Experience - interpret *data*
3. Performance measure i.e. error/cost/objective functions - number of words known, number of boxes moved, etc.

Learning is used analogously to *modelling* i.e. finding the best function $f(x, \theta)$ that can approximate a task.

A brief classification

A brief classification

By experience:

- Unsupervised learning
- Supervised learning
- Reinforcement learning

A brief classification

By experience:

- Unsupervised learning
- Supervised learning
- Reinforcement learning

By task:

- Classification
- Regression
- Focused tasks i.e. machine translation, anomaly detection, software analysis

A simple example

Problem setting:

- Task: Classify irises (flowers) in two categories.
- Experience: Labeled measurements: $(x, y) = ([height, color, \dots], Virginica)$
- Performance measure?
- *Model* $[f(x, \theta)]$?

Linear Models

Performance: Mean Squared Error (MSE)

Model: $\hat{y} = \boldsymbol{W}^{\boldsymbol{T}} \boldsymbol{x}$

Where:

$\boldsymbol{W} \in \mathbb{R}^n$ is a vector of parameters

$MSE = \frac{1}{m} \sum_i (\hat{y}^{(test)} - y^{(test)})_i^2$

Linear Models

Performance: Mean Squared Error (MSE)

Model: $\hat{y} = \boldsymbol{W}^{\boldsymbol{T}} \boldsymbol{x}$

Where:

$\boldsymbol{W} \in \mathbb{R}^n$ is a vector of parameters

$MSE = \frac{1}{m} \sum_i (\hat{y}^{(test)} - y^{(test)})_i^2$

Goal: Minimise the error

Linear Models

Performance: Mean Squared Error (MSE)

Model: $\hat{y} = \boldsymbol{W}^{\boldsymbol{T}} \boldsymbol{x}$

Where:

$\boldsymbol{W} \in \mathbb{R}^n$ is a vector of parameters

$MSE = \frac{1}{m} \sum_i (\hat{y}^{(test)} - y^{(test)})_i^2$

Goal: Minimise the error

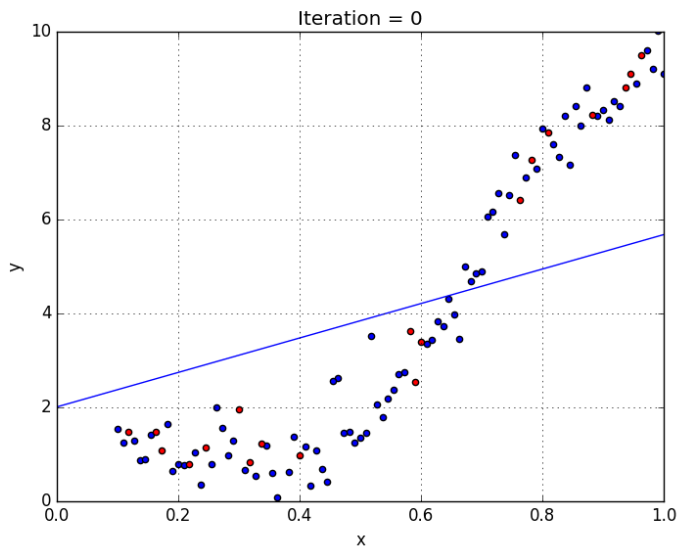
Solution: Compute derivatives!

Linear Models

Solution:

$$\nabla_w MSE_{train} = 0$$

$$\nabla_w \frac{1}{m} \sum_i (\hat{y}^{(test)} - y^{(test)})_i^2$$



The power of derivatives

The power of derivatives

Challenges Motivating Deep Learning

The development of deep learning was motivated in part by the failure of traditional algorithms to generalise well on central problems of AI such as recognising speech or objects [1].

Challenges Motivating Deep Learning

The development of deep learning was motivated in part by the failure of traditional algorithms to generalise well on central problems of AI such as recognising speech or objects [1].

Reasons?

Challenges Motivating Deep Learning

The development of deep learning was motivated in part by the failure of traditional algorithms to generalise well on central problems of AI such as recognising speech or objects [1].

Reasons?

- High Dimensionality [the number of possible configuration of a set of variables increases exponentially with the number of variables]

Challenges Motivating Deep Learning

The development of deep learning was motivated in part by the failure of traditional algorithms to generalise well on central problems of AI such as recognising speech or objects [1].

Reasons?

- High Dimensionality [the number of possible configuration of a set of variables increases exponentially with the number of variables]
- Hard to learn functions with interesting variations across all of $\mathbb{R}$ [Manifold Learning]

In search for better approximations

Requirements:

In search for better approximations

Requirements:

- Represent highly nonlinear functions in large hyper-spaces

In search for better approximations

Requirements:

- Represent highly nonlinear functions in large hyper-spaces
- Easy to compute derivatives

In search for better approximations

Requirements:

- Represent highly nonlinear functions in large hyper-spaces
- Easy to compute derivatives
- Parallel scaling

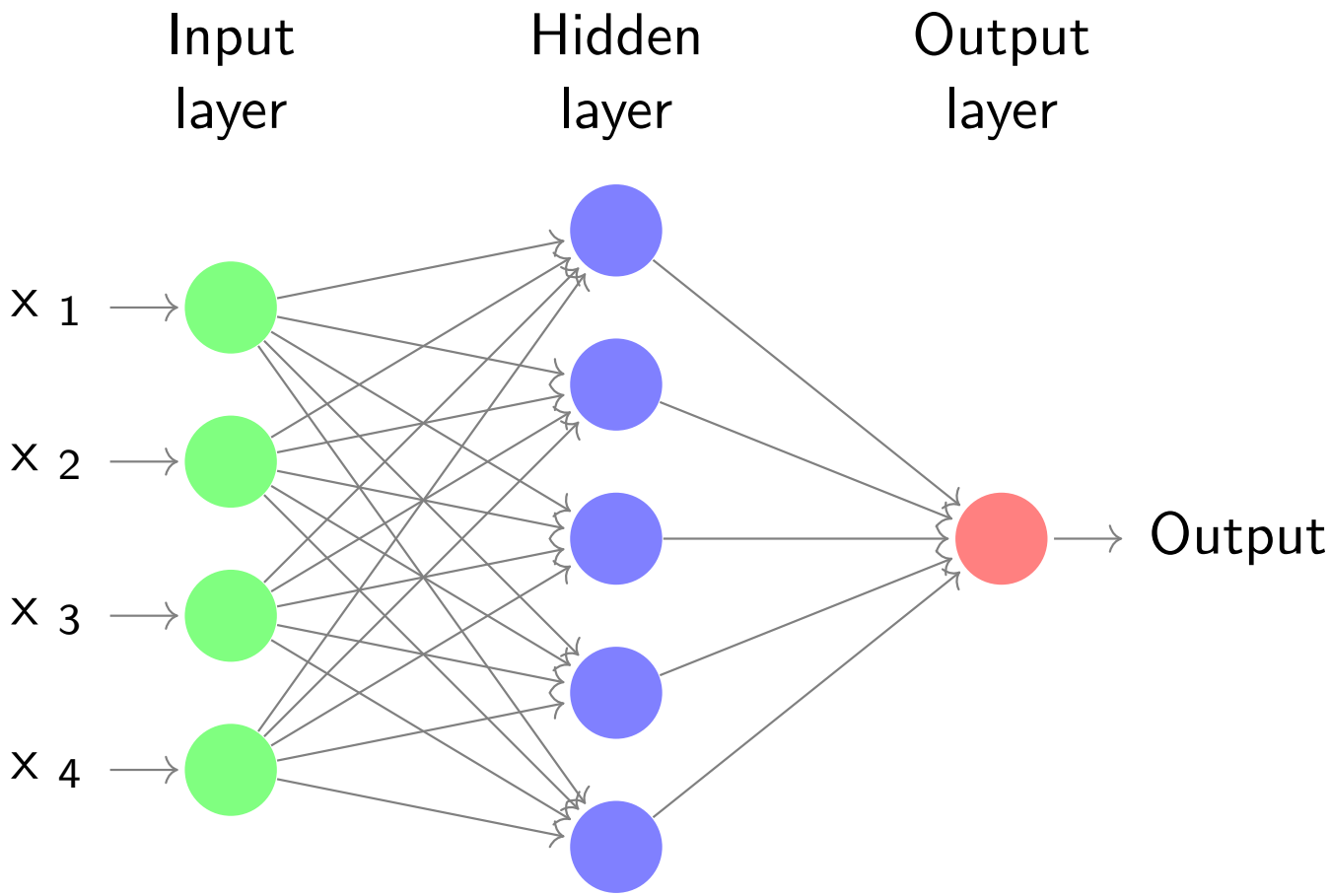
In search for better approximations

Requirements:

- Represent highly nonlinear functions in large hyper-spaces
- Easy to compute derivatives
- Parallel scaling

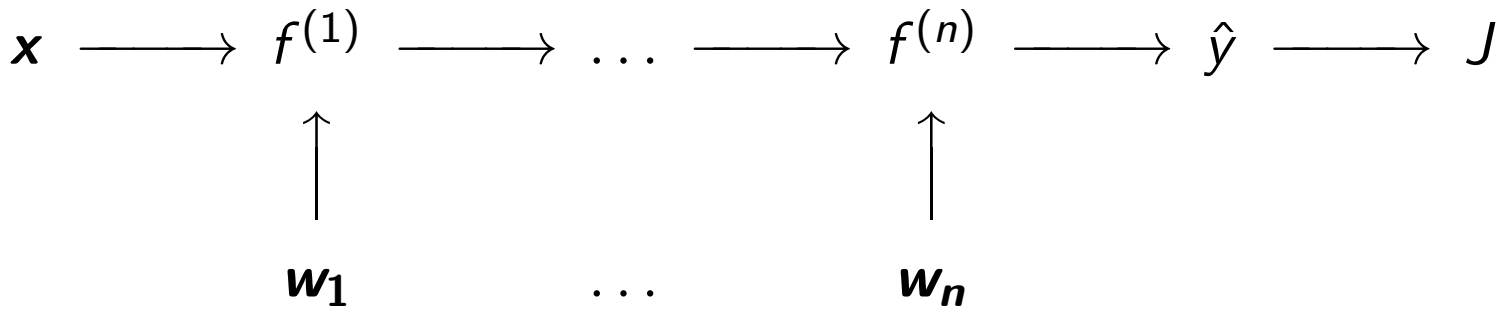
Solution? Functional *composition*!

Neural Networks



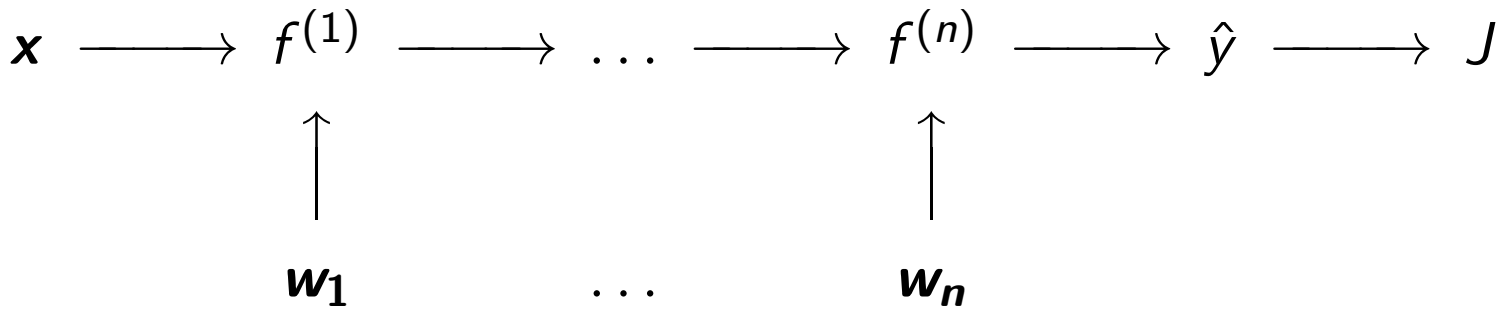
Deep Representations

- A *deep representation* is a composition of many functions

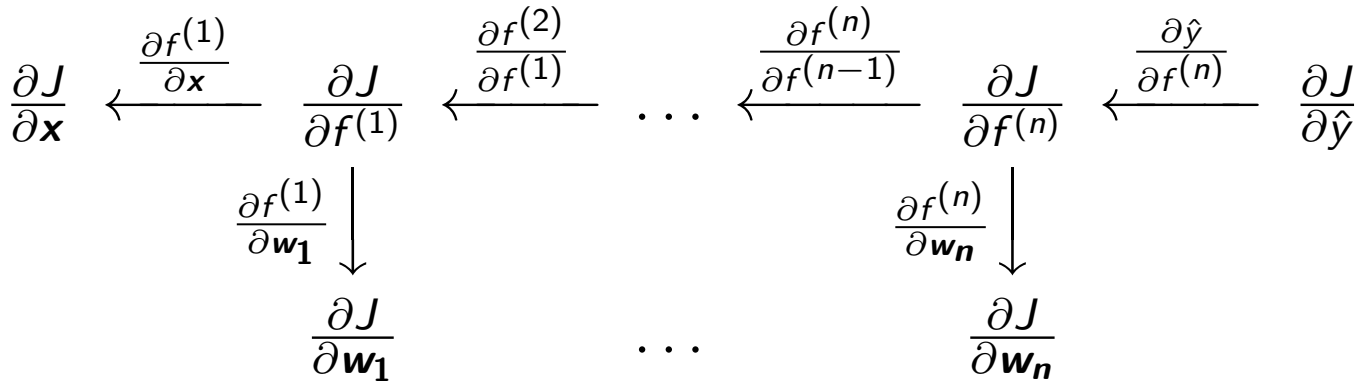


Deep Representations

- A *deep representation* is a composition of many functions



- Its gradient can be *back-propagated* by the chain rule



Deep Neural Networks

A *deep neural network* is typically composed of:

- Linear transformations

$$h_{k+1} = W h_k$$

Deep Neural Networks

A *deep neural network* is typically composed of:

- Linear transformations

$$h_{k+1} = W h_k$$

- Non-linear activation functions

$$h_{k+2} = f(h_{k+1})$$

Deep Neural Networks

A *deep neural network* is typically composed of:

- Linear transformations

$$h_{k+1} = W h_k$$

- Non-linear activation functions

$$h_{k+2} = f(h_{k+1})$$

- A loss function, e.g.
  - Mean-squared error $l = ||\hat{y} - y||^2$
  - Log likelihood $l = \log \mathbb{P}[\hat{y}]$

Universal Approximation Theorem

Definition

A feed-forward network with a *single hidden layer* containing a finite number of neurones can approximate any continuous function on compact subsets of $\mathbb{R}^n$.

Wrapping it up

Ingredients:

- Experiences - data

Wrapping it up

Ingredients:

- Experiences - data
- Tasks - clear goals

Wrapping it up

Ingredients:

- Experiences - data
- Tasks - clear goals
- Performance measures - functions easy to minimise - using basic tools such as *derivatives*

Wrapping it up

Ingredients:

- Experiences - data
- Tasks - clear goals
- Performance measures - functions easy to minimise - using basic tools such as *derivatives*

Key take-away(s):

Wrapping it up

Ingredients:

- Experiences - data
- Tasks - clear goals
- Performance measures - functions easy to minimise - using basic tools such as *derivatives*

Key take-away(s):

- Computers love high dimensions

Wrapping it up

Ingredients:

- Experiences - data
- Tasks - clear goals
- Performance measures - functions easy to minimise - using basic tools such as *derivatives*

Key take-away(s):

- Computers love high dimensions
- It is easier to discover highly non-linear functions through *composition* than to *engineer* them

Wrapping it up

Ingredients:

- Experiences - data
- Tasks - clear goals
- Performance measures - functions easy to minimise - using basic tools such as *derivatives*

Key take-away(s):

- Computers love high dimensions
- It is easier to discover highly non-linear functions through *composition* than to *engineer* them
- Learning is just a tool

References I


Ian Goodfellow et al. *Deep learning*. Vol. 1. MIT press Cambridge, 2016.


Tom M Mitchell et al. "Machine learning. 1997". In: *Burr Ridge, IL: McGraw Hill* 45.37 (1997), pp. 870–877.