Deep Feedforward Networks [GBC16]

Sarntal Ferienakademie – Course 10
Computational Medical Imaging

Rayen Manai

TU München

17.09.2023 – 29.09.2023

Friedrich-Alexander-Universität Technische Fakultät

Technische Universität München

Universität Stuttgart

Outline

What do we expect from deep learning?

- Solve tasks that are easy for people to perform but hard for people to describe formally
- problems that we solve intuitively, like recognizing spoken words or faces in images.

What do we expect from deep learning?

- Solve tasks that are easy for people to perform but hard for people to describe formally
- problems that we solve intuitively, like recognizing spoken words or faces in images.
- Goal: allow computers to learn from experience and understand the world in terms of a hierarchy of concepts
- a graph showing how these concepts are built $\rightarrow$ a deep graph $\rightarrow$ deep Learning
- the ability to acquire knowledge, by extracting patterns from raw data

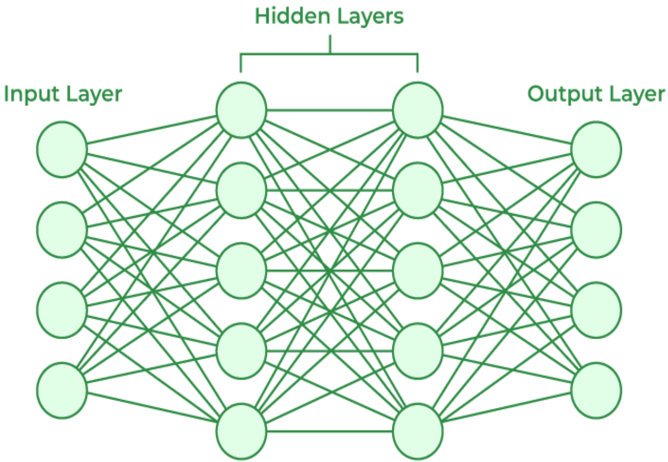
- Deep feedforward network = feedforward neural netwrok = multilayer perceptrons (MLPs)
- essential deep learning model

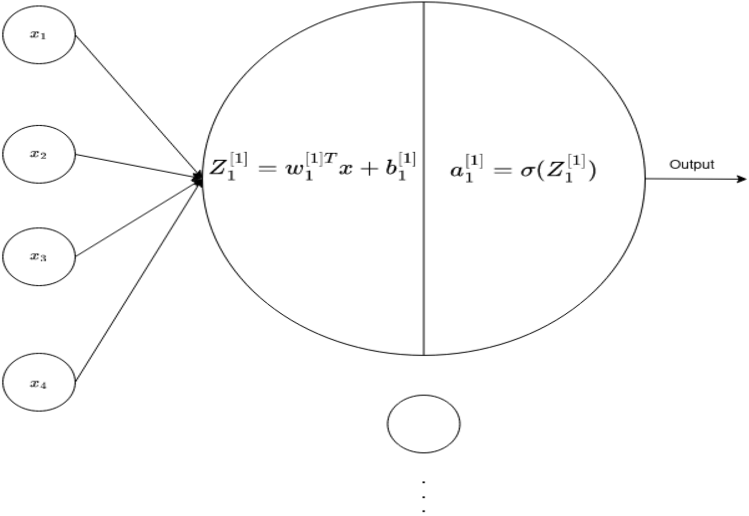
- Deep feedforward network = feedforward neural netwrok = multilayer perceptrons (MLPs)
- essential deep learning model
- Goal: approximate some function f^*
- for example: for a classifier $y = f^*(x) \rightarrow$ approxiamte it with $y = f(x; \theta)$

- Deep feedforward network = feedforward neural netwrok = multilayer perceptrons (MLPs)
- essential deep learning model
- Goal: approximate some function f^*
- for example: for a classifier $y = f^*(x) \rightarrow$ approxiamte it with $y = f(x;\theta)$
- feedfoward? $x \rightarrow$ intermidiate computations $\rightarrow$ output y

- Deep feedforward network = feedforward neural netwrok = multilayer perceptrons (MLPs)
- essential deep learning model
- Goal: approximate some function f^*
- for example: for a classifier $y = f^*(x) \rightarrow$ approxiamte it with $y = f(x; \theta)$
- feedfoward? $x \rightarrow$ intermidiate computations $\rightarrow$ output y
- networks? represented by composing together many different functions
- The model is a directed acyclic graph: how the functions are compoded together, layers, depth, width, hidden layers, units

Overview



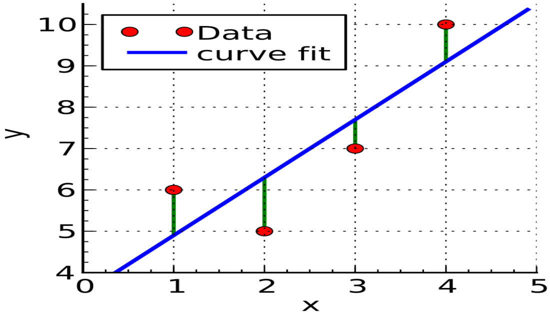

x_1
x_2
x_3
x_4
$Z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$
$a_1^{[1]} = \sigma(Z_1^{[1]})$
Output

Recap: Linear Regression

- a linear approach used to fit a predective model to an observed data set
- error reduction in prediction

Recap: Linear Regression

- a linear approach used to fit a predective model to an observed data set
- error reduction in prediction
- often fitted using the least squares approach



Example: Learning XOR

x1	x2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

- The XOR function provides the target function $y = f^*(x)$that we want to learn.

Example: Learning XOR

x1	x2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

- The XOR function provides the target function $y = f^*(x)$that we want to learn.
- Our model provides a function $y = f(x; \theta)$ and our learning algorithm will adapt the parameters θ to make f as similar as possible to f^*

Example: Learning XOR

x1	x2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

- The XOR function provides the target function $y = f^*(x)$that we want to learn.
- Our model provides a function $y = f(x; \theta)$ and our learning algorithm will adapt the parameters θ to make f as similar as possible to f^*
- Train the model on the four points $\mathbf{X} = \{[0,0]^T, [0,1]^T, [1,0]^T, [1,1]^T\}$

First choice: Linear Model

- Treat the problem as a regression problem and use a mean squared error loss function

$$J(\theta) = \frac{1}{4} \sum_{x \in X} (f * (x) - f(x; \theta))^2$$

First choice: Linear Model

- Treat the problem as a regression problem and use a mean squared error loss function

$$J(\theta) = \frac{1}{4} \sum_{x \in X} (f * (x) - f(x; \theta))^2$$

- we chose a linear model:

$$f(x, w, b) = x^T w + b$$

First choice: Linear Model

- Treat the problem as a regression problem and use a mean squared error loss function

$$J(\theta) = \frac{1}{4} \sum_{x \in X} (f * (x) - f(x; \theta))^2$$

- we chose a linear model:

$$f(x, w, b) = x^T w + b$$

- we minimize (θ) with respect to w and b
- $\rightarrow$ we obtain $w = 0$ and $b = 0.5$
- the linear model simply outputs 0.5 everywhere

First choice: Linear Model

- Treat the problem as a regression problem and use a mean squared error loss function

$$J(\theta) = \frac{1}{4} \sum_{x \in X} (f*(x) - f(x;\theta))^2$$

- we chose a linear model:

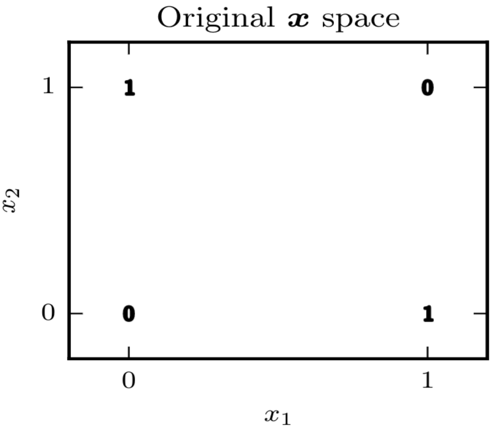
$$f(x, w, b) = x^T w + b$$

- we minimize (θ) with respect to w and b
- $\rightarrow$ we obtain $w = 0$ and $b = 0.5$
- the linear model simply outputs 0.5 everywhere

Why does this happen?

Explanation

- A linear model is not able to represent the XOR function
- XOR is not linearly seperable



Second choice: different feature space

- a simple feedforward network with one hidden layer (containing two hidden units)
- the network now contains two function chained together

$$h = f^{(1)}(x, W, c)$$

and

$$y = f^{(2)}(h, w, b)$$

Second choice: different feature space

- a simple feedforward network with one hidden layer (containing two hidden units)
- the network now contains two function chained together

$$h = f^{(1)}(x, W, c)$$

and

$$y = f^{(2)}(h, w, b)$$

What function should $f^{(1)}$ compute? Can it be linear?

- No, the feedforward netwrok as whole would remain a linear function of its input

- No, the feedforward netwrok as whole would remain a linear function of its input
- $\rightarrow$ we use an affine transformation controlled by learned parameters, followed by a fixed nonlinear function called activation function

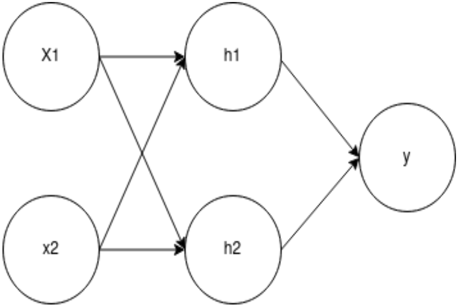
$$h = g(W^T x + b)$$

- No, the feedforward netwrok as whole would remain a linear function of its input
- $\rightarrow$ we use an affine transformation controlled by learned parameters, followed by a fixed nonlinear function called activation function

$$h = g(W^T x + b)$$

- W: provides the weights of a linear transformation
- b: provides the biases

Network Diagram



Solving XOR

- The complete Newtork: $f(x, W, c, w, b) = w^T max\{0, W^T x + c\} + b$

Solving XOR

- The complete Newtork: $f(x, W, c, w, b) = w^T max\{0, W^T x + c\} + b$
- We can specify a solution:

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$$

Solving XOR

- The complete Newtork: $f(x, W, c, w, b) = w^T max\{0, W^T x + c\} + b$
- We can specify a solution:

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$$

- $\mathbf{X}$ the design matrix containing all four inputs with one example per Column: $\mathbf{X} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$

Let's compute it:

- $W^T X + c = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 2 \\ -1 & 0 & 0 & 1 \end{bmatrix}$

Let's compute it:

- $W^T X + c = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 2 \\ -1 & 0 & 0 & 1 \end{bmatrix}$
- Activations of the first layer: the rectified Linear Transformations:

$$max\{0, W^T X + c\} = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let's compute it:

- $W^TX + c = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 2 \\ -1 & 0 & 0 & 1 \end{bmatrix}$
- Activations of the first layer: the rectified Linear Transformations:

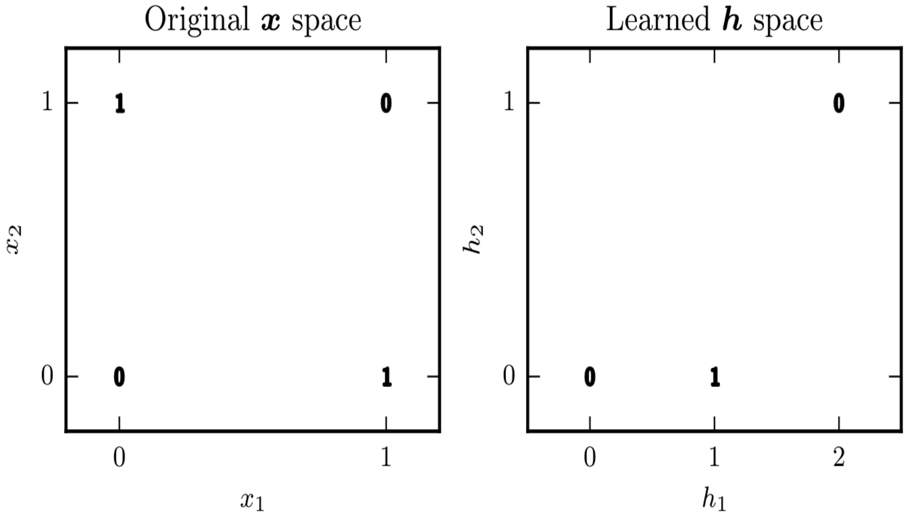
$$max\{0, W^TX + c\} = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Activations of the second layer (output):

$$w^T max\{0, W^Tx + c\} + b = \begin{bmatrix} 1 & -2 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}$$

$\rightarrow$ the correct answer for every example

What have we changed?



Gradient Based Learning:

- Recap on Gradient Descent

Gradient Based Learning:

- Recap on Gradient Descent
- For feedforward neural networks, it is important to initialize all weights to small random values.
- we must choose a cost function, and we must choose how to represent the output of the model

Cost Functions

- An important aspect of the design of a deep neural network is the choice of the cost function

Cost Functions

- An important aspect of the design of a deep neural network is the choice of the cost function
- In general, the model defines a distribution $p(y|x;\theta)$ and we use the principle of maximum likelihood
- $\rightarrow$ The cross-entropy between the training data and the model's predictions as the cost function.

Cost Functions

- An important aspect of the design of a deep neural network is the choice of the cost function
- In general, the model defines a distribution $p(y|x;\theta)$ and we use the principle of maximum likelihood
- $\rightarrow$ The cross-entropy between the training data and the model's predictions as the cost function.
- The total cost function combines a cost function with a regularization term

Learning Conditional Distributions with Maximum Likelihood

- the cost function is simply the negative log-likelihood, equivalently described as the cross-entropy between the training data and the model distribution.

$$J(\theta) = -\mathbb{E}_{x,y \sim \hat{p}_{data}} \log p_{model}(y|x)$$

Learning Conditional Distributions with Maximum Likelihood

- the cost function is simply the negative log-likelihood, equivalently described as the cross-entropy between the training data and the model distribution.

$$J(\theta) = -\mathbb{E}_{x,y\sim\hat{p}_{data}} \log p_{model}(y|x)$$

- used to evaluate the performance of a model's predicted probability distribution against the true distribution of the data

Example Binary Classification:

- In the context of binary classification, where y is the true probablity and $\hat{y}$ the predicted probability:

$$J = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

Example Binary Classification:

- In the context of binary classification, where y is the true probablity and $\hat{y}$ the predicted probability:

$$J = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

- $y = 1 \rightarrow J = -(y \log \hat{y}) \rightarrow \hat{y}$ large, closer to 1

Example Binary Classification:

- In the context of binary classification, where y is the true probablity and $\hat{y}$ the predicted probability:

$$J = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

- $y = 1 \rightarrow J = -(y \log \hat{y}) \rightarrow \hat{y}$ large, closer to 1
- $y = 0 \rightarrow J = -\log(1 - \hat{y}) \rightarrow 1 - \hat{y}$ large, $\hat{y}$ closer to 0

Output Units

- The choice of cost function is tightly coupled with the choice of output unit
- The choice of how to represent the output then determines the form of the cross-entropy function
- Linear Units for Gaussian Output:

$$\hat{y} = W^T h + b$$

- Sigmoid Units:

$$\hat{y} = \sigma(w^T h + b)$$

Hidden Units

- Now: how to choose the type of hidden unit to use in the hidden layers of the model?

Hidden Units

- Now: how to choose the type of hidden unit to use in the hidden layers of the model?
- extremely active area of research and does not yet have many definitive guiding theoretical principles

Hidden Units

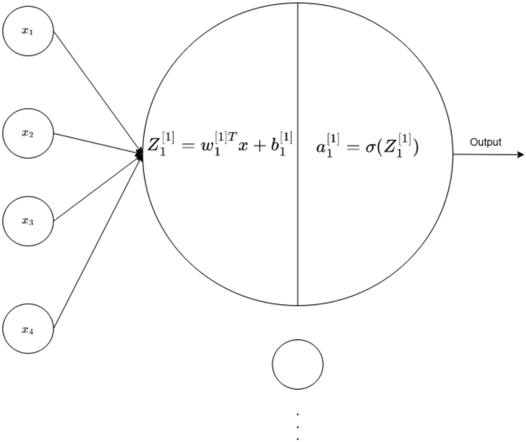
- Now: how to choose the type of hidden unit to use in the hidden layers of the model?
- extremely active area of research and does not yet have many definitive guiding theoretical principles
- Rectified linear units are an excellent default choice of hidden unit

Hidden Units

- Now: how to choose the type of hidden unit to use in the hidden layers of the model?
- extremely active area of research and does not yet have many definitive guiding theoretical principles
- Rectified linear units are an excellent default choice of hidden unit
- Some of the hidden units included are not actually differentiable at all input points
- "in practice one can safely disregard the nondifferentiability of the hidden unit activation functions described below"

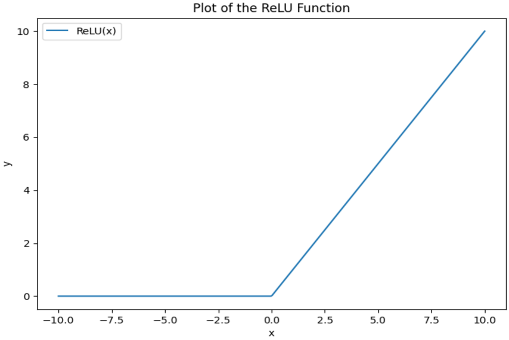
Role of hidden units:

- accept a vector of inputs x
- compute an affine transformation: $z = W^T x + b$

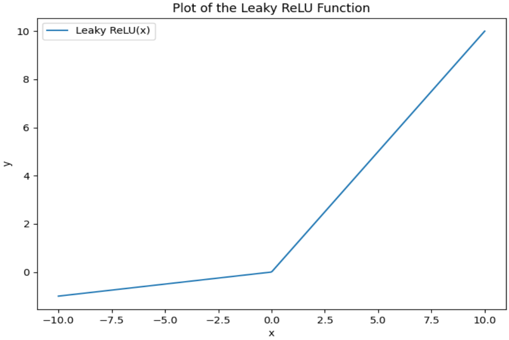


ReLU

- use the activation function $g(z) = max\{0, z\}$
- easy to optimize
- typically used on top of an affine transformation $h = g(W^T x + b)$



Leaky ReLU


Plot of the Leaky ReLU Function
Leaky ReLU(x)
y
x
10
8
6
4
2
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

Logistic Sigmoid and Hyperbolic Tangent

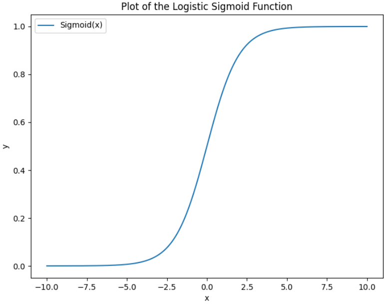
- Prior to the introduction of rectified linear units, most neural networks used the logistic sigmoid activation function

$$g(z) = \sigma(z)$$

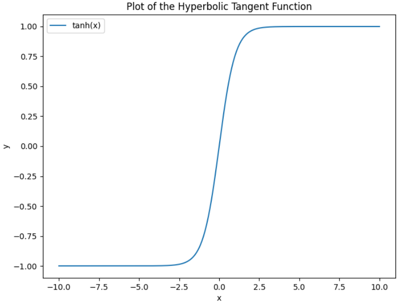
  or the hyperbolic tangent activation function

$$g(z) = \tanh(z)$$

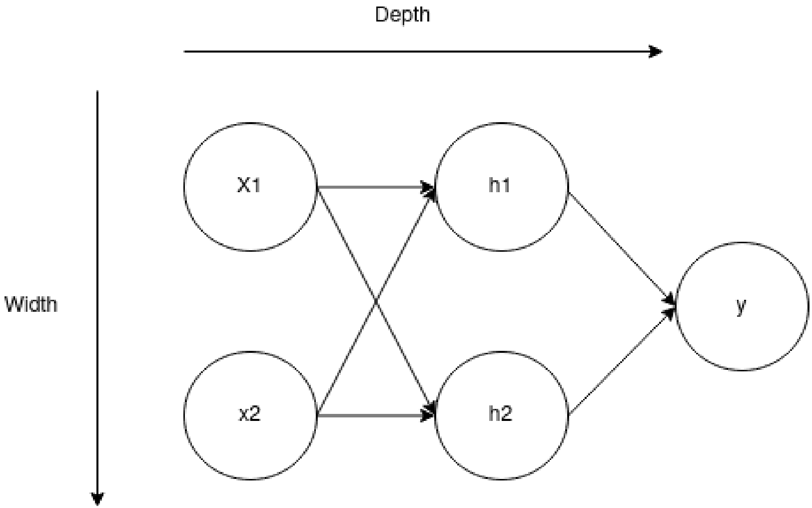
- $\tanh(z) = 2\sigma(2z) - 1$



(a) Logistic Sigmoid



(b) Hyperbolic Tangent


Depth
Width
X1
x2
h1
h2
y

What do we mean with architecture?

- Architecture: the overall structure of the network: how many units it should have and how these units should be connected to each other

What do we mean with architecture?

- Architecture: the overall structure of the network: how many units it should have and how these units should be connected to each other
- Most neural network architectures arrange layers in a chain structure, with each layer being a function of the layer that preceded it

What do we mean with architecture?

- Architecture: the overall structure of the network: how many units it should have and how these units should be connected to each other
- Most neural network architectures arrange layers in a chain structure, with each layer being a function of the layer that preceded it
- The first layer is given by :

$$h^{(1)} = g^{(1)}(W^{(1)T}x + b^{(1)})$$

- The second layer:

$$h^{(2)} = g^{(2)}(W^{(2)T}x + b^{(2)})$$

What do we mean with architecture?

- Architecture: the overall structure of the network: how many units it should have and how these units should be connected to each other
- Most neural network architectures arrange layers in a chain structure, with each layer being a function of the layer that preceded it
- The first layer is given by :

$$h^{(1)} = g^{(1)}(W^{(1)T}x + b^{(1)})$$

- The second layer:

$$h^{(2)} = g^{(2)}(W^{(2)T}x + b^{(2)})$$

- the main architectural considerations are choosing the depth of the network and the width of each layer

The Universal Approximator Theorem

- One hidden layer is enough to represent an approximation of any function to an arbitrary degree of accuracy

The Universal Approximator Theorem

- One hidden layer is enough to represent an approximation of any function to an arbitrary degree of accuracy

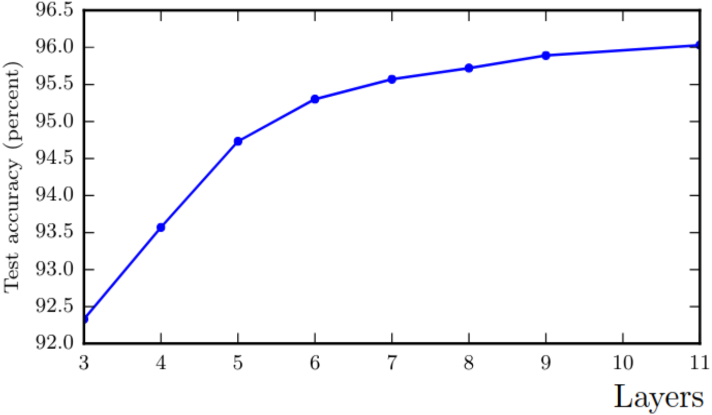
So why deeper Networks?

The Universal Approximator Theorem

- One hidden layer is enough to represent an approximation of any function to an arbitrary degree of accuracy

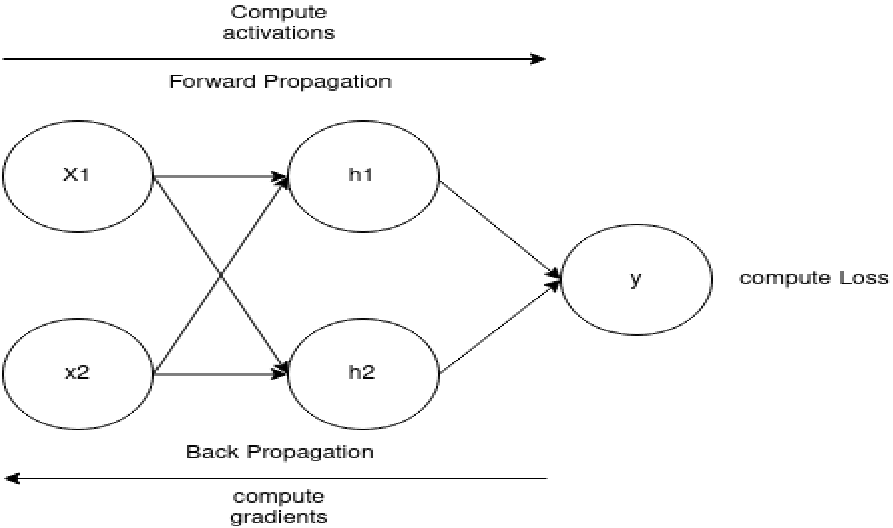
So why deeper Networks?

- Shallow Network may need (exponentially) more width
- Shallow Network may overfit more



96.5
96.0
95.5
95.0
94.5
94.0
93.5
93.0
92.5
92.0
Test accuracy (percent)
3
4
5
6
7
8
9
10
11
Layers

overview


Compute
activations
Forward Propagation
X1
h1
y
compute Loss
x2
h2
Back Propagation
compute
gradients

Computational Graphs:

- To describe the back-propagation Algorithm more precisely $\rightarrow$ a more precise computational Graph language

Computational Graphs:

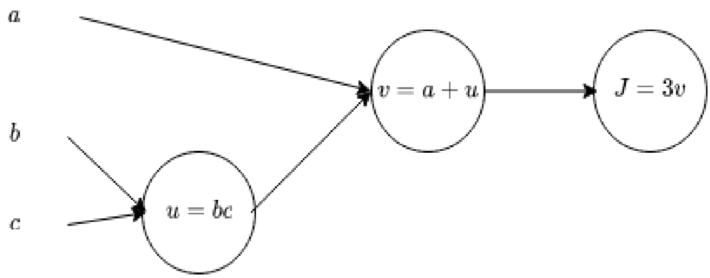
- To describe the back-propagation Algorithm more precisely $\rightarrow$ a more precise computational Graph language
- each node indicates a variable(scalar, vector, matrix, tensor ...)
- Operation: a simple function of one or more variables
- a directed edge from x to y: y is computed by applying an operation to a variable x

Computational Graphs:

- To describe the back-propagation Algorithm more precisely $\rightarrow$ a more precise computational Graph language
- each node indicates a variable(scalar, vector, matrix, tensor ...)
- Operation: a simple function of one or more variables
- a directed edge from x to y: y is computed by applying an operation to a variable x
- $\rightarrow$ just a way of expressing and evaluating a mathematical expression

Example:

$$J = 3(a + bc)$$



Chain Rule of Calculus

- used to compute the derivatives of functions formed by composing other functions

Chain Rule of Calculus

- used to compute the derivatives of functions formed by composing other functions
- Back-propagation is an algorithm that computes the chain rule, with a specific order of operations that is highly efficient

Chain Rule of Calculus

- used to compute the derivatives of functions formed by composing other functions
- Back-propagation is an algorithm that computes the chain rule, with a specific order of operations that is highly efficient
- Suppose that $y = g(x)$ and $z = f(g(x)) = f(y)$ then the chain rule states that:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Chain Rule of Calculus

- used to compute the derivatives of functions formed by composing other functions
- Back-propagation is an algorithm that computes the chain rule, with a specific order of operations that is highly efficient
- Suppose that $y = g(x)$ and $z = f(g(x)) = f(y)$ then the chain rule states that:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

- we can generalize this: $x \in \mathbb{R}^m, y \in \mathbb{R}^n$ g maps from $\mathbb{R}^m$ to $\mathbb{R}^n$ and f from $\mathbb{R}^n$ to $\mathbb{R}$ then:

$$\nabla_x z = (\frac{\partial y}{\partial x})^T \nabla_y z$$

- $\frac{\partial y}{\partial x}$ is the $n \times m$ Jacobian matrix of g

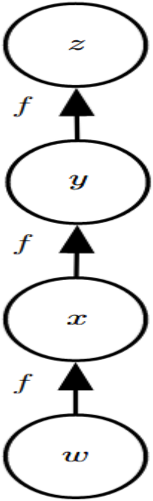
Recursively Applying the Chain Rule to obtain Backprop

- Back-propagation is the chain rule of calculus recursively applied to compute gradients of expressions
- It is a particular implementation of the chain rule

Recursively Applying the Chain Rule to obtain Backprop

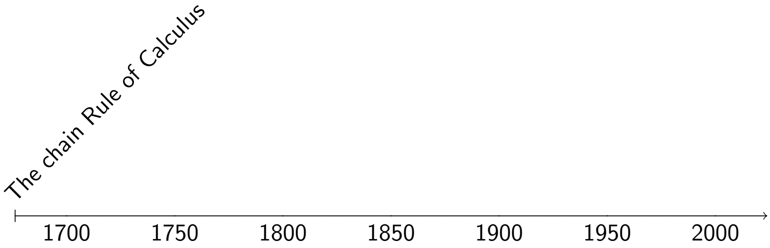
- Back-propagation is the chain rule of calculus recursively applied to compute gradients of expressions
- It is a particular implementation of the chain rule
- uses dynamic programming (table filling) $\rightarrow$ to avoid recomputing repeated subexpressions
- Speed vs memory tradeoff

Repeated subexpressions



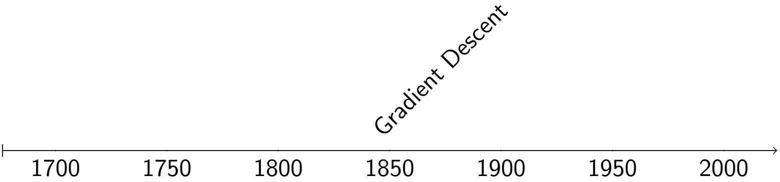
$$
\begin{aligned}
&\frac{\partial z}{\partial w} \\
&= \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} \\
&= f'(y)f'(x)f'(w) \\
&= f'(f(f(w)))f'(f(w))f'(w)
\end{aligned}
$$

A Little History



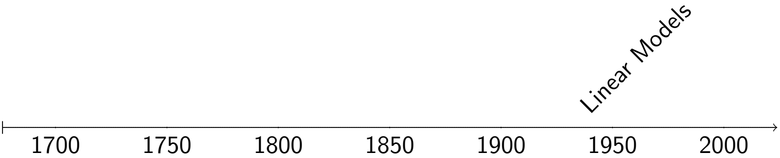
- The chain rule that underlies the back-propagation algorithm was invented in the seventeenth century (Leibniz, 1676; L'Hôpital, 1696)

A little History



- Gradient descent was introduced as a technique for iteratively approximating the solution to optimization problems in the nineteenth century (Cauchy, 1847).

A little History

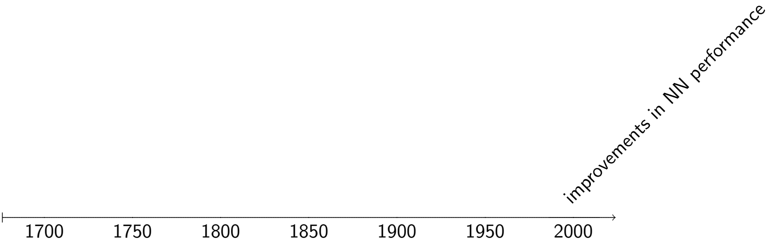


- Beginning in the 1940s, these function approximation techniques were used to motivate machine learning models such as the perceptron. However, the earliest models were based on linear models

A little History

- the first successful experiments with back-propagation presented on the book Parallel Distributed Processing on 1986

A little History



- Most of the improvement in neural network performance from 1986 to 2015 can be attributed to the following factors

Why is Deep Learning taking off?

- larger dataset

Why is Deep Learning taking off?

- larger dataset
- neural networks have become much larger, because of more powerful computers and better software infrastructure

Why is Deep Learning taking off?

- larger dataset
- neural networks have become much larger, because of more powerful computers and better software infrastructure
- some algorithmic changes have also improved the performance of neural networks noticeably

Why is Deep Learning taking off?

- larger dataset
- neural networks have become much larger, because of more powerful computers and better software infrastructure
- some algorithmic changes have also improved the performance of neural networks noticeably
- the replacement of mean squared error with the cross-entropy family of loss functions
- the replacement of sigmoid hidden units with piecewise linear hidden units, such as rectified linear units

Code demo

Putting it together: Planar Data Classification

Thank you

Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT Press, 2016, http://www.deeplearningbook.org.