Deep Learning

Chapter 1: Feedforward Neural Networks

David Rügamer

Department of Statistics – LMU Munich

Winter Semester 2020

LECTURE OUTLINE

Single Hidden Layer Networks for Regression and Binary Classification

Universal Approximation Property

Single Hidden Layer Networks for Multi-Class Classification

Multi-Layer Feedforward Neural Networks

Deep Learning

References

MOTIVATION

- We have a nice graphical way of representing simple functions/models like logistic regression. Why is that useful?

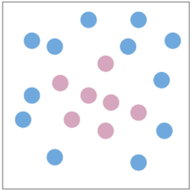
- It is useful because this visual metaphor allows us to use such individual neurons as building blocks of considerably more complicated functions.

- Therefore, networks of neurons can represent extremely complex hypothesis spaces.

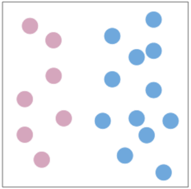
- Most importantly, it allows us to define the “right” kinds of hypothesis spaces to learn functions that are more common in our universe in a data-efficient way (see Lin, Tegmark et al. 2016).

MOTIVATION

- But why do we need more complicated functions? Is logistic regression not enough?
- Because a single neuron is restricted to learning only linear decision boundaries, its performance on the task below will be quite poor.

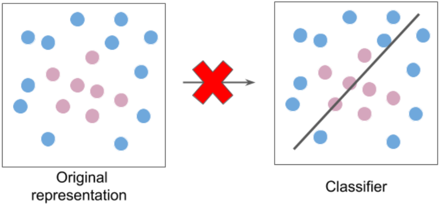


- However, if the original features are transformed (for example from Cartesian to polar coord.), the neuron can easily separate the classes.



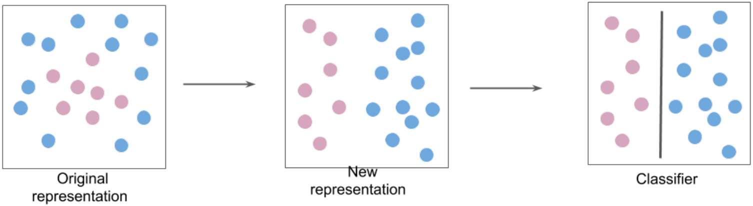
MOTIVATION

Instead of classifying the data in the original representation, ...



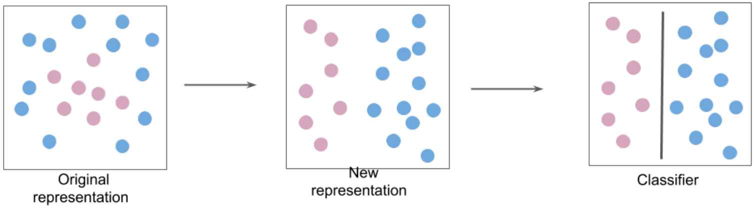
MOTIVATION

we classify it in the new feature space.

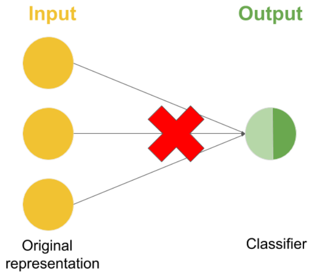


MOTIVATION

we classify it in the new feature space.

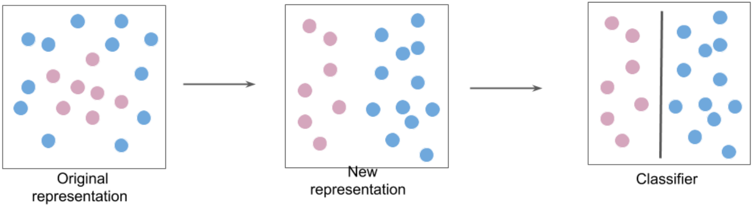


Analogously,

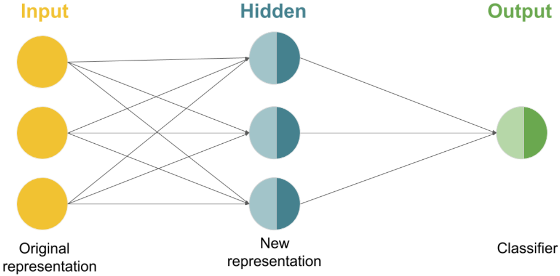


MOTIVATION

we classify it in the new feature space.



Analogously,



REPRESENTATION LEARNING

- Therefore, it is *very* critical to feed a classifier the “right” features in order for it to perform well.

- Before deep learning (DL) took off, features for tasks like machine vision and speech recognition were “hand-designed” by domain experts. This step of the machine learning pipeline is called **feature engineering**.

- The single biggest reason DL is so important is that it automates feature engineering. This is called **representation learning**.

Single Hidden Layer Networks for Regression and Binary Classification

SINGLE HIDDEN LAYER NETWORKS

- The input $\mathbf{x}$ is a column vector with dimensions $p \times 1$.
- $\mathbf{W}$ is a weight matrix with dimensions $p \times m$:

$$\mathbf{W} = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{p,1} & w_{p,2} & \cdots & w_{p,m} \end{pmatrix}$$

- For example, to obtain z_1, we pick the first column of W:

$$\mathbf{W}_1 = \begin{pmatrix} w_{1,1} \\ w_{2,1} \\ \vdots \\ w_{p,1} \end{pmatrix}$$

and compute $z_1 = \sigma(W_1^\top \mathbf{x} + b_1)$, where b_1 is the bias of the first hidden neuron and $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function.

SINGLE HIDDEN LAYER NETWORKS: NOTATION

General notation:

- The network has m hidden neurons $z_1, \dots, z_m$ with

$$z_j = \sigma(\mathbf{W}_j^\top \mathbf{x} + b_j)$$

  - $z_{in,j} = \mathbf{W}_j^\top \mathbf{x} + b_j$
  - $z_{out,j} = \sigma(z_{in,j}) = \sigma(\mathbf{W}_j^\top \mathbf{x} + b_j)$

for $j \in \{1, \dots, m\}$.

SINGLE HIDDEN LAYER NETWORKS: NOTATION

- Vectorized notation:
  - $\mathbf{z}_{in} = (z_{in,1}, \dots, z_{in,m})^\top = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$
    (Note: $\mathbf{W}^\top \mathbf{x}$ = $(\mathbf{x}^\top \mathbf{W})^\top$)
  - $\mathbf{z} = \mathbf{z}_{out} = \sigma(\mathbf{z}_{in}) = \sigma(\mathbf{W}^\top \mathbf{x} + \mathbf{b})$, where the (hidden layer) activation function σ is applied element-wise to $\mathbf{z}_{in}$.
- Bias term:
  - We sometimes omit the bias term by adding a constant feature to the input $\tilde{\mathbf{x}} = (1, x_1, ..., x_p)$ and by adding the bias term to the weight matrix

    $$\tilde{\mathbf{W}} = (\mathbf{b}, \mathbf{W}_1, ..., \mathbf{W}_p).$$

  - **Note**: For simplification purposes, we will not explicitly represent the bias term graphically in the following. However, the above “trick” makes it straightforward to represent it graphically.

SINGLE HIDDEN LAYER NETWORKS: NOTATION

General notation:

- For regression or binary classification: one output unit f where
  - $f_{in} = \mathbf{u}^\top \mathbf{z} + c$, i.e. a linear combination of derived features plus the bias term c of the output neuron, and
  - $f(\mathbf{x}) = f_{out} = \tau(f_{in}) = \tau(\mathbf{u}^\top \mathbf{z} + c)$, where τ is the output activation function.
- For regression τ is the identity function.
- For binary classification, τ is a sigmoid function.
- **Note**: The purpose of the hidden-layer activation function σ is to introduce non-linearities so that the network is able to learn complex functions whereas the purpose of τ is merely to get the final score on the same scale as the target.

SINGLE HIDDEN LAYER NETWORKS: NOTATION

General notation: Multiple inputs

- It is possible to feed multiple inputs to a neural network simultaneously.

- The inputs $\mathbf{x}^{(i)}$, for $i \in \{1, \dots, n\}$, are arranged as rows in the **design matrix X**.
  - $\mathbf{X}$ is a $(n \times p)$-matrix.

- The weighted sum in the hidden layer is now computed as $\mathbf{XW} + \boldsymbol{B}$, where,
  - $\mathbf{W}$, as usual, is a $(p \times m)$ matrix, and,
  - $\boldsymbol{B}$ is a $(n \times m)$ matrix containing the bias vector $\mathbf{b}$ (duplicated) as the rows of the matrix.

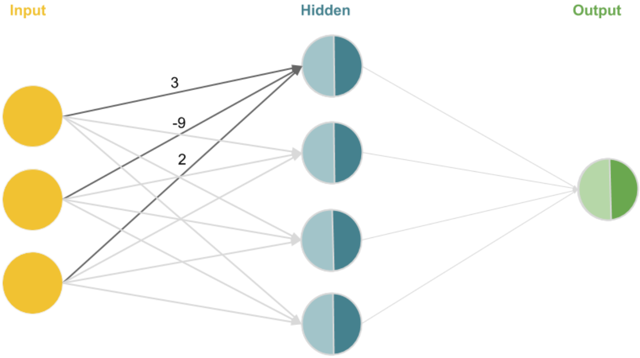
- The *matrix* of hidden activations $\boldsymbol{Z} = \sigma(\mathbf{XW} + \boldsymbol{B})$
  - $\boldsymbol{Z}$ is a $(n \times m)$ matrix.

SINGLE HIDDEN LAYER NETWORKS: NOTATION

- The final output of the network, which contains a prediction for each input, is $\tau(\boldsymbol{Z}\mathbf{u} + \boldsymbol{C})$, where
  - $\mathbf{u}$ is the vector of weights of the output neuron, and,
  - $\boldsymbol{C}$ is a $(n \times 1)$ matrix whose elements are the (scalar) bias c of the output neuron.

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

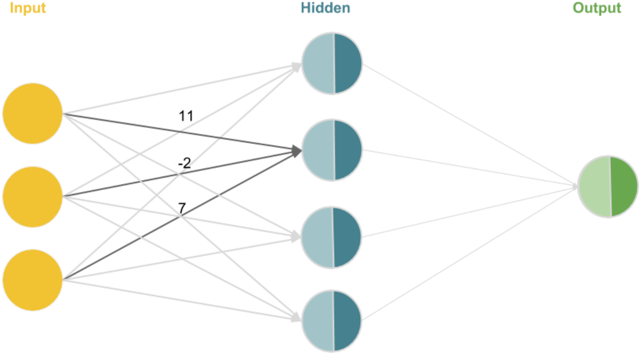
- Weights (and biases) of the network.



$$\underbrace{\begin{pmatrix} 3 & -9 & 2 \\ & & \\ & & \\ & & \end{pmatrix}}_{W^T} \underbrace{\begin{pmatrix} 5 \\ \\ \\ \\ \end{pmatrix}}_{\mathbf{b}} \qquad \underbrace{\begin{pmatrix} & & & \end{pmatrix}}_{\mathbf{u}^T} \underbrace{\begin{pmatrix} \; \end{pmatrix}}_{c}$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

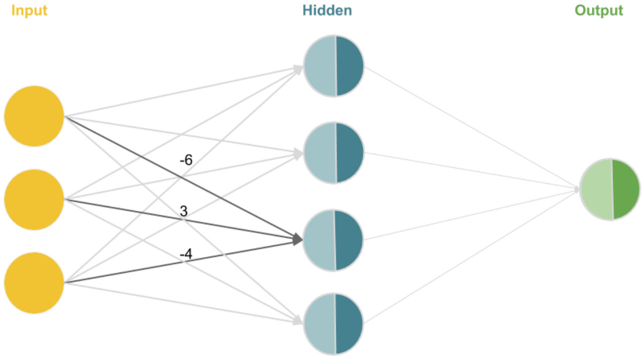
- Weights (and biases) of the network.



$$\underset{W^T}{\begin{pmatrix} 3 & -9 & 2 \\ 11 & -2 & 7 \\ & & \\ & & \end{pmatrix}} \underset{\mathbf{b}}{\begin{pmatrix} 5 \\ 2 \\ \\ \\ \end{pmatrix}} \qquad \underset{\mathbf{u}^T}{\begin{pmatrix} & & & \end{pmatrix}} \underset{c}{\begin{pmatrix} \; \end{pmatrix}}$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

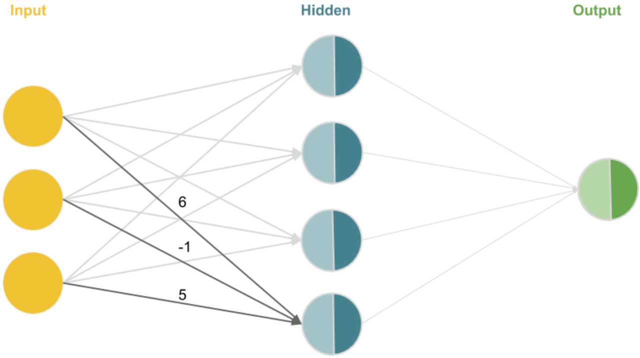
- Weights (and biases) of the network.



$$\underbrace{\begin{pmatrix} 3 & -9 & 2 \\ 11 & -2 & 7 \\ \mathbf{-6} & \mathbf{3} & \mathbf{-4} \end{pmatrix}}_{W^T} \underbrace{\begin{bmatrix} 5 \\ 2 \\ \mathbf{-1} \end{bmatrix}}_{\mathbf{b}} \qquad \underbrace{\begin{bmatrix} \quad \end{bmatrix}}_{\mathbf{u}^T} \underbrace{\begin{bmatrix} \; \end{bmatrix}}_{c}$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

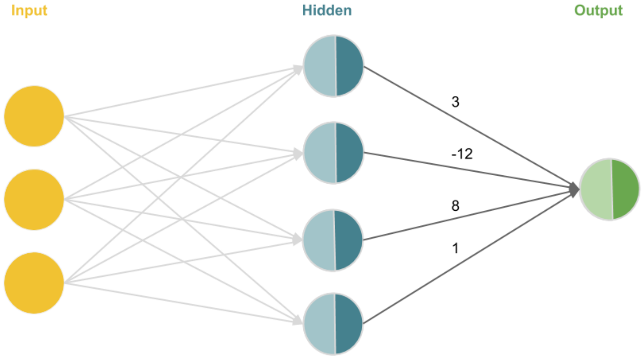
- Weights (and biases) of the network.



$$\underset{W^T}{\begin{pmatrix} 3 & -9 & 2 \\ 11 & -2 & 7 \\ -6 & 3 & -4 \\ \mathbf{6} & \mathbf{-1} & \mathbf{5} \end{pmatrix}} \underset{\mathbf{b}}{\begin{bmatrix} 5 \\ 2 \\ -1 \\ \mathbf{1} \end{bmatrix}} \qquad \underset{\mathbf{u}^T}{\begin{bmatrix} \quad \end{bmatrix}} \underset{c}{\begin{bmatrix} \; \end{bmatrix}}$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

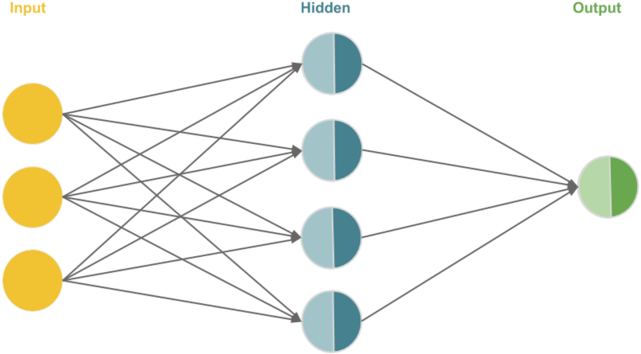
- Weights (and biases) of the network.



$$\underbrace{\begin{pmatrix} 3 & -9 & 2 \\ 11 & -2 & 7 \\ -6 & 3 & -4 \\ 6 & -1 & 5 \end{pmatrix}}_{W^T} \underbrace{\begin{pmatrix} 5 \\ 2 \\ -1 \\ 1 \end{pmatrix}}_{\mathbf{b}} \qquad \underbrace{\begin{pmatrix} 3 & -12 & 8 & 1 \end{pmatrix}}_{\mathbf{u}^T} \underbrace{\begin{pmatrix} 6 \end{pmatrix}}_{c}$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

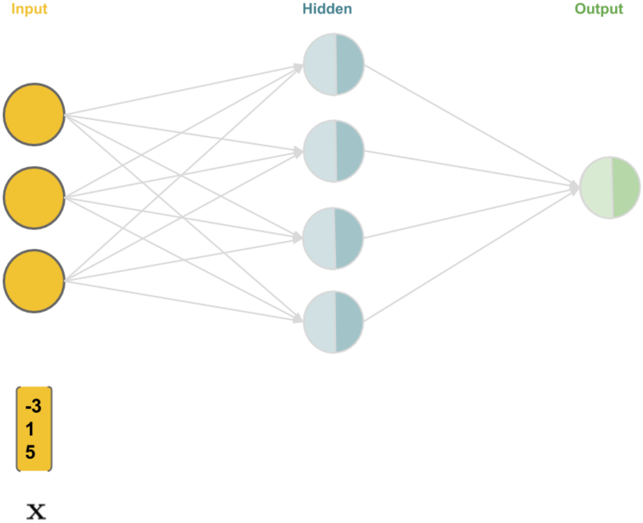
- Weights (and biases) of the network.



$$\underset{W^T}{\begin{pmatrix} 3 & -9 & 2 \\ 11 & -2 & 7 \\ -6 & 3 & -4 \\ 6 & -1 & 5 \end{pmatrix}} \underset{\mathbf{b}}{\begin{pmatrix} 5 \\ 2 \\ -1 \\ 1 \end{pmatrix}} \qquad \underset{\mathbf{u}^T}{\begin{bmatrix} 3 & -12 & 8 & 1 \end{bmatrix}} \underset{c}{\begin{bmatrix} 6 \end{bmatrix}}$$

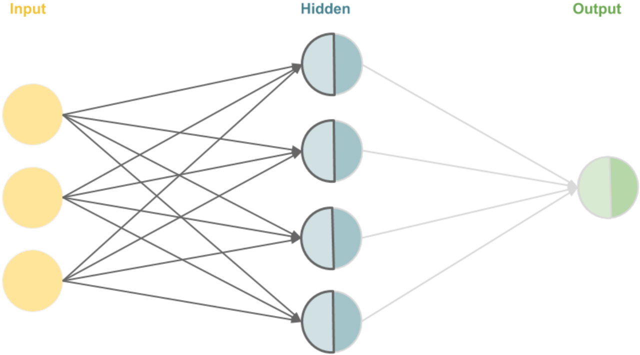
SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Forward pass through the shallow neural network.



SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Forward pass through the shallow neural network.

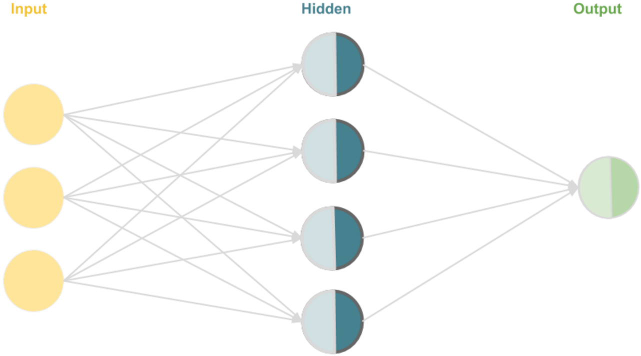


$$\begin{pmatrix} -3 \\ 1 \\ 5 \end{pmatrix} \qquad \begin{pmatrix} (-3)*3 + 1*(-9) + 5*2 + 5 \\ (-3)*11 + 1*(-2) + 5*7 + 2 \\ (-3)*(-6) + 1*3 + 5*(-4) + (-1) \\ (-3)*6 + 1*(-1) + 5*5 + 1 \end{pmatrix}$$

$$\mathbf{x} \qquad \mathbf{z}_{in} = W^{\top}\mathrm{x} + \mathrm{b}$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

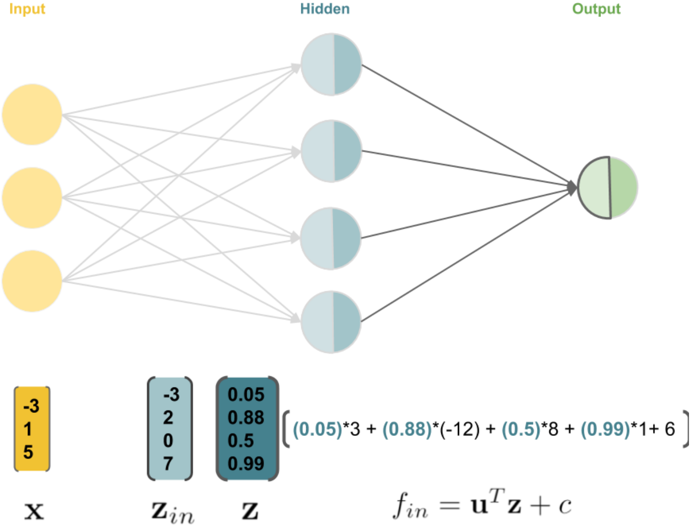
Forward pass through the shallow neural network.



$$\begin{array}{ccc} \begin{bmatrix}-3\\1\\5\end{bmatrix} & \begin{bmatrix}-3\\2\\0\\7\end{bmatrix} & \begin{pmatrix}1/\ (1+\exp(-(-3)) \\ 1/\ (1+\exp(-2) \\ 1/\ (1+\exp(-0) \\ 1/\ (1+\exp(-7)\end{pmatrix} \\ \mathbf{x} & \mathbf{z}_{in} & \mathbf{z} = \mathbf{z}_{out} = \sigma(\mathbf{z}_{in}) \end{array}$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Forward pass through the shallow neural network.

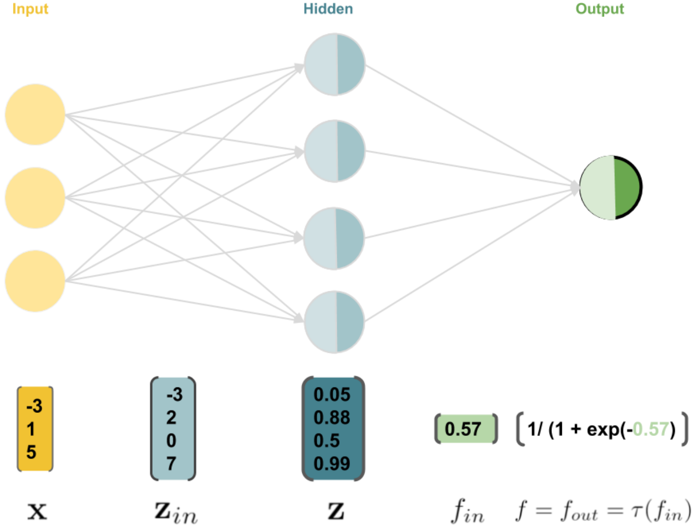


[(0.05)*3 + (0.88)*(-12) + (0.5)*8 + (0.99)*1+ 6]

$$f_{in} = \mathbf{u}^T \mathbf{z} + c$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Forward pass through the shallow neural network.



$\mathbf{x}$: -3, 1, 5

$\mathbf{z}_{in}$: -3, 2, 0, 7

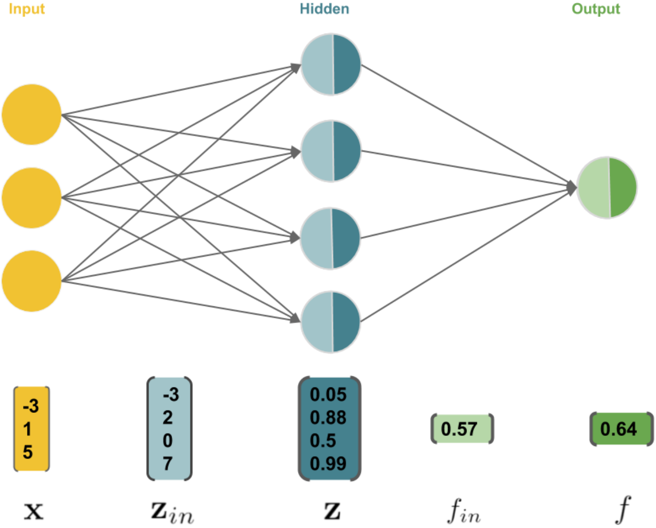
$\mathbf{z}$: 0.05, 0.88, 0.5, 0.99

f_{in}: 0.57

$f = f_{out} = \tau(f_{in})$: 1/ (1 + exp(-0.57))

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Forward pass through the shallow neural network.



HIDDEN LAYER: ACTIVATION FUNCTION

- It is important to note that if the hidden layer does not have a non-linear activation, the network can only learn linear decision boundaries.
- For simplification purposes, we drop the bias terms in notation and let $\sigma = \text{id}$. Then:

$$\begin{aligned} f(\mathbf{x}) &= \tau(\mathbf{u}^\top \mathbf{z}) = \tau(\mathbf{u}^\top \sigma(\mathbf{W}^\top \mathbf{x})) \\ &= \tau(\mathbf{u}^\top \sigma(\mathbf{W}^\top \mathbf{x})) \\ &= \tau(\mathbf{u}^\top \mathbf{W}^\top \mathbf{x}) = \tau(\mathbf{v}^\top \mathbf{x}) \end{aligned}$$

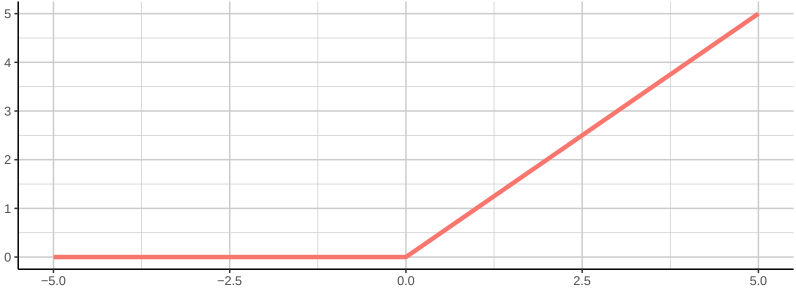
  where $\mathbf{v} = \mathbf{W}\mathbf{u}$. It can be seen that $f(\mathbf{x})$ can only yield a linear decision boundary.

HIDDEN LAYER: ACTIVATION FUNCTION

ReLU activation:

- Currently the most popular choice is the ReLU (rectified linear unit):

$$\sigma(v) = \max(0, v)$$



HIDDEN LAYER: ACTIVATION FUNCTION

- Some important properties of the ReLU function include:
  - limits:

$$\lim_{v \to -\infty} \sigma(v) = 0 \text{ and } \lim_{v \to \infty} \sigma(v) = \infty$$

  - derivative:

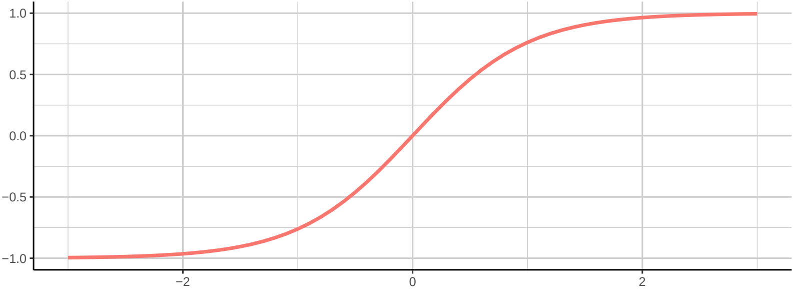
$$\frac{\delta \sigma(v)}{\delta v} = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{else} \end{cases}$$

HIDDEN LAYER: ACTIVATION FUNCTION

Hyperbolic tangent activation:

- Another choice might be the hyperbolic tangent function:

$$\sigma(v) = \tanh(v) = \frac{\sinh(v)}{\cosh(v)} = 1 - \frac{2}{\exp(2v) + 1}$$



HIDDEN LAYER: ACTIVATION FUNCTION

- Some important properties of the hyperbolic tangent function include:
  - limits:

    $$\lim_{v \to -\infty} \sigma(v) = -1 \text{ and } \lim_{v \to \infty} \sigma(v) = 1$$

  - derivative:

    $$\frac{\delta \sigma(v)}{\delta v} = 1 - \tanh^2(v)$$

  - symmetry:

    $$\sigma(v) \text{ is rotationally symmetric about } (0, 0)$$

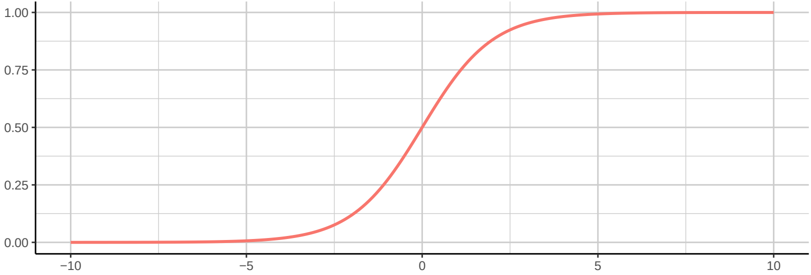
    (that is, a rotation of 180° does not change the graph of the function.)

HIDDEN LAYER: ACTIVATION FUNCTION

Sigmoid activation function:

- Of course, as seen in the previous example, the sigmoid function can be used even in the hidden layer:

$$\sigma(v) = \frac{1}{1 + \exp(-v)}$$



HIDDEN LAYER: ACTIVATION FUNCTION

- Some important properties of the logistic sigmoid function include:

  - limits:

$$\lim_{v \to -\infty} \sigma(v) = 0 \text{ and } \lim_{v \to \infty} \sigma(v) = 1$$

  - the derivative:

$$\frac{\delta \sigma(v)}{\delta v} = \frac{\exp(v)}{(1 + \exp(v))^2} = \sigma(v)(1 - \sigma(v))$$

  - symmetry:

$\sigma(v)$ is rotationally symmetric about $(0, 0.5)$

EXAMPLE: XOR PROBLEM

- Suppose we have four data points

$$X = \{(0,0)^\top, (0,1)^\top, (1,0)^\top, (1,1)^\top\}$$

- The XOR gate (exclusive or) returns true, when an odd number of inputs are true:

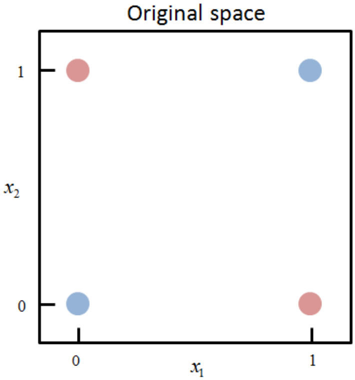
x_1	x_2	**XOR** $= y$
0	0	0
0	1	1
1	0	1
1	1	0

- Can you learn the target function with a logistic regression model?

EXAMPLE: XOR PROBLEM

- Logistic regression can not solve this problem. In fact, any model using simple hyperplanes for separation can not (including a single neuron).

- A small neural net can easily solve the problem by transforming the space!



EXAMPLE: XOR PROBLEM

- Consider the following model:

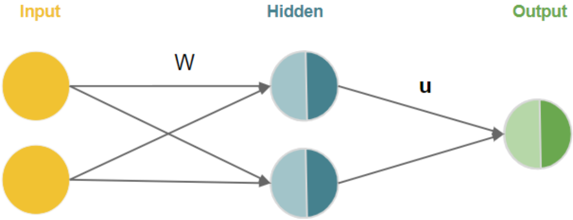


Figure: A neural network with two neurons in the hidden layer. The matrix **W** describes the mapping from **x** to **z**. The vector **u** from **z** to y.

EXAMPLE: XOR PROBLEM

- Let use ReLU $\sigma(z) = \max\{0, z\}$ as activation function and a simple thresholding function $\tau(z) = [z > 0] = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$ as output transformation function. We can represent the architecture of the model by the following equation:

$$\begin{aligned} f(\mathbf{x} \mid \boldsymbol{\theta}) &= f(\mathbf{x} \mid \mathbf{W}, \mathbf{b}, \mathbf{u}, c) = \tau\left(\mathbf{u}^\top \sigma(\mathbf{W}^\top \mathbf{x} + \mathbf{b}) + c\right) \\ &= \tau\left(\mathbf{u}^\top \max\{0, \mathbf{W}^\top \mathbf{x} + \mathbf{b}\} + c\right) \end{aligned}$$

- So how many parameters does our model have?
    - In a fully connected neural net, the number of connections between the nodes equals our parameters:

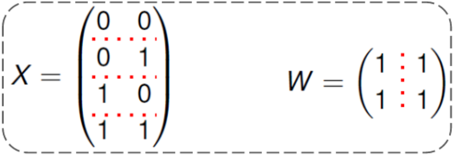
$$\underbrace{(2 \times 2)}_{W} + \underbrace{(2 \times 1)}_{\mathbf{b}} + \underbrace{(2 \times 1)}_{\mathbf{u}} + \underbrace{(1)}_{c} = 9$$

EXAMPLE: XOR PROBLEM

$$\text{Let } \mathbf{W} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \ \mathbf{b} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \ \mathbf{u} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}, \ c = -0.5$$

$$\mathbf{X} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}, \ \mathbf{XW} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{pmatrix}, \ \mathbf{XW} + \boldsymbol{B} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

Note: $\mathbf{X}$ is a $(n \times p)$ design matrix in which the *rows* correspond to the data points. $\mathbf{W}$, as usual, is a $(p \times m)$ matrix where each *column* corresponds to a single (hidden) neuron. $\boldsymbol{B}$ is a $(n \times m)$ matrix with $\mathbf{b}$ duplicated along the rows.



$$X = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \qquad W = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

EXAMPLE: XOR PROBLEM

$$\text{Let } \mathbf{W} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix},\ \mathbf{b} = \begin{pmatrix} 0 \\ -1 \end{pmatrix},\ \mathbf{u} = \begin{pmatrix} 1 \\ -2 \end{pmatrix},\ c = -0.5$$

$$\mathbf{X} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix},\ \mathbf{XW} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{pmatrix},\ \mathbf{XW} + \boldsymbol{B} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

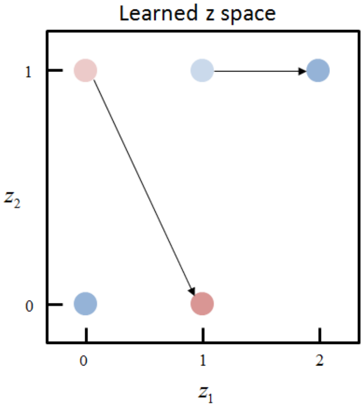
$$\boldsymbol{Z} = \max\{0, \mathbf{XW} + \boldsymbol{B}\} \quad = \quad \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

- Note that we computed all examples at once.

EXAMPLE: XOR PROBLEM

- The input points are mapped into transformed space to

$$\boldsymbol{Z} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

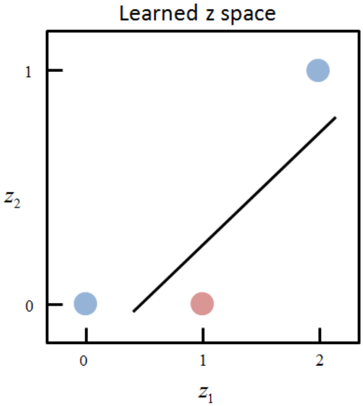


EXAMPLE: XOR PROBLEM

- The input points are mapped into transformed space to

$$\boldsymbol{Z} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

which is easily separable.



EXAMPLE: XOR PROBLEM

- In a final step we have to multiply the activated values of matrix $\boldsymbol{Z}$ with the vector $\mathbf{u}$ and add the bias term c:

$$f(\mathbf{x} \mid \mathbf{W}, \mathbf{b}, \mathbf{u}, c) = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \end{pmatrix} + \begin{pmatrix} -0.5 \\ -0.5 \\ -0.5 \\ -0.5 \end{pmatrix} = \begin{pmatrix} -0.5 \\ 0.5 \\ 0.5 \\ -0.5 \end{pmatrix}$$

- And then apply the step function $\tau(z) = [z > 0]$. This solves the XOR problem perfectly!

x_1	x_2	**XOR** = y
0	0	0
0	1	1
1	0	1
1	1	0

NEURAL NETWORKS: OPTIMIZATION

- In this simple example we actually “guessed” the values of the parameters for **W**, **b**, ***u*** and *c*.
- That will not work for more sophisticated problems!
- To learn the right weights (and biases), we have to rely on iterative algorithms like gradient descent.
- An added complication is that the loss function is no longer convex. Therefore, there might not exist a single minimum.
- An extremely efficient method to compute gradients called backpropogation will be covered in the next lecture.

Universal Approximation Property

UNIVERSAL APPROXIMATION PROPERTY

Theorem. Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a continuous, non-constant, bounded, and monotonically increasing function. Let $C \subset \mathbb{R}^p$ be compact, and let $\mathcal{C}(C)$ denote the space of continuous functions $C \to \mathbb{R}$. Then, given a function $g \in \mathcal{C}(C)$ and an accuracy $\varepsilon > 0$, there exists a hidden layer size $m \in \mathbb{N}$ and a set of coefficients $\mathbf{W}_j \in \mathbb{R}^p$, $u_j, b_j \in \mathbb{R}$ (for $j \in \{1, \dots, m\}$), such that

$$f_m : C \to \mathbb{R}\,; \quad f_m(\mathbf{x}) = \sum_{j=1}^{m} u_j \cdot \sigma\left(\mathbf{W}_j^\top \mathbf{x} + b_j\right)$$

is an ε-approximation of g, that is,

$$\|f_m - g\|_\infty := \max_{x \in C} |f_m(\mathbf{x}) - g(\mathbf{x})| < \varepsilon \ \ .$$

The theorem extends trivially to multiple outputs.

UNIVERSAL APPROXIMATION PROPERTY

Corollary. Neural networks with a single sigmoidal hidden layer and linear output layer are universal approximators.

- This means that for a given target function *g* there exists a sequence of networks $\left(f_m\right)_{m\in\mathbb{N}}$ that converges (pointwise) to the target function.
- Usually, as the networks come closer and closer to *g*, they will need more and more hidden neurons.
- A network with fixed layer sizes can only model a subspace of all continuous functions. Its dimensionality is limited by the number of weights.
- The continuous functions form an infinite dimensional vector space. Therefore arbitrarily large hidden layer sizes are needed.

UNIVERSAL APPROXIMATION PROPERTY

- Why is universal approximation a desirable property?
- Recall the definition of a Bayes optimal hypothesis $h^* : \mathcal{X} \rightarrow \mathcal{Y}$. It is the best possible hypothesis (model) for the given problem: it has minimal loss averaged over the data generating distribution.
- So ideally we would like the neural network (or any other learner) to approximate the Bayes optimal hypothesis.
- Usually we do not manage to learn h^*.
- This is because we do not have enough (infinite) data. We have no control over this, so we have to live with this limitation.
- But we do have control over which model class we use.

UNIVERSAL APPROXIMATION PROPERTY

- Universal approximation $\Rightarrow$ approximation error tends to zero as hidden layer size tends to infinity.
- Positive approximation error implies that no matter how good the data, we cannot find the optimal model.
- This bears the risk of systematic under-fitting, which can be avoided with a universal model class.

UNIVERSAL APPROXIMATION PROPERTY

- As we know, there are also good reasons for restricting the model class.

- This is because a flexible model class with universal approximation ability often results in over-fitting, which is no better than under-fitting.

- Thus, “universal approximation $\Rightarrow$ low approximation error”, but at the risk of a substantial generalization error.

- In general, models of intermediate flexibility give the best predictions. For neural networks this amounts to a reasonably sized hidden layer.

NNS AS (NON-)PARAMETRIC MODELS

Question: Are NNs parametric or non-parametric models?

NNS AS (NON-)PARAMETRIC MODELS

Question: Are NNs parametric or non-parametric models?

- Parametric models assume some finite set of model parameters. So the complexity of the model is bounded even if the amount of training data is unbounded. This makes them not very flexible.
  Example: Linear Model.
- A model is non-parametric, if the number of parameters is not fixed. Complexity of the model can grow as the amount of data grows. This makes them very flexible.
  Example: k-NN.

NNS AS (NON-)PARAMETRIC MODELS

Question: Are NNs parametric or non-parametric models?

- Parametric models assume some finite set of model parameters. So the complexity of the model is bounded even if the amount of training data is unbounded. This makes them not very flexible.
  Example: Linear Model.
- A model is non-parametric, if the number of parameters is not fixed. Complexity of the model can grow as the amount of data grows. This makes them very flexible.
  Example: k-NN.

If the architecture (here: the size of the hidden layer) is fixed, the number of trainable parameters is fixed and NNs are to be seen as parametric models.
However, if we do not restrict the hidden layer size prior to training but integrate optimizing the hidden layer size into training, NNs can also be seen as non-parametric models.

NEURAL NETS: REGRESSION/CLASSIFICATION

- Let us look at a few examples of the types of functions and decisions boundaries learnt by neural networks (with a **single** hidden layer) of various sizes.
- “size” here refers to the number of neurons in the hidden layer.
- The number of “iterations” in the following slides is the number of steps of the applied iterative optimization algorithm (stochastic gradient descent).

REGRESSION: 1000 TRAINING ITERATIONS

```
## Error in library("mlr"): there is no package called ’mlr’
## Error in makeRegrTask("sine function example", data = df, target = "y"): could
not find function "makeRegrTask"
## Error in plotLearnerPrediction("regr.nnet", tsk, size = 1L, maxit = 1000): could
not find function "plotLearnerPrediction"
```

REGRESSION: 1000 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("regr.nnet", tsk, size = 2L, maxit = 1000): could
not find function "plotLearnerPrediction"
```

REGRESSION: 1000 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("regr.nnet", tsk, size = 3L, maxit = 1000): could
not find function "plotLearnerPrediction"
```

REGRESSION: 1000 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("regr.nnet", tsk, size = 4L, maxit = 1000): could
not find function "plotLearnerPrediction"
```

REGRESSION: 1000 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("regr.nnet", tsk, size = 5L, maxit = 1000): could
not find function "plotLearnerPrediction"
```

REGRESSION: 1000 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("regr.nnet", tsk, size = 6L, maxit = 1000): could
not find function "plotLearnerPrediction"
```

REGRESSION: 1000 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("regr.nnet", tsk, size = 7L, maxit = 1000): could
not find function "plotLearnerPrediction"
```

REGRESSION: 1000 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("regr.nnet", tsk, size = 8L, maxit = 1000): could
not find function "plotLearnerPrediction"
```

REGRESSION: 1000 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("regr.nnet", tsk, size = 9L, maxit = 1000): could
not find function "plotLearnerPrediction"
```

REGRESSION: 1000 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("regr.nnet", tsk, size = 10L, maxit = 1000):
could not find function "plotLearnerPrediction"
```

CLASSIFICATION: 500 TRAINING ITERATIONS

```
## Error in library("mlr"): there is no package called ’mlr’
## Error in library("mlbench"): there is no package called ’mlbench’
## Error in mlbench.spirals(500, 1.5, 0.05): could not find function
"mlbench.spirals"
## Error in cbind(spirals$x, spirals$classes): object ’spirals’ not found
## Error in colnames(spirals) = c("x1", "x2", "class"): object ’spirals’ not found
## Error in is.factor(x): object ’spirals’ not found
## Error in makeClassifTask(data = spirals, target = "class"): could not find
function "makeClassifTask"
## Error in makeLearner("classif.nnet"): could not find function "makeLearner"
## Error in plotLearnerPrediction("classif.nnet", task, size = 1L, maxit = 500):
could not find function "plotLearnerPrediction"
```

CLASSIFICATION: 500 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("classif.nnet", task, size = 2L, maxit = 500):
could not find function "plotLearnerPrediction"
```

CLASSIFICATION: 500 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("classif.nnet", task, size = 3L, maxit = 500):
could not find function "plotLearnerPrediction"
```

CLASSIFICATION: 500 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("classif.nnet", task, size = 5L, maxit = 500):
could not find function "plotLearnerPrediction"
```

CLASSIFICATION: 500 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("classif.nnet", task, size = 10L, maxit = 500):
could not find function "plotLearnerPrediction"
```

CLASSIFICATION: 500 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("classif.nnet", task, size = 30L, maxit = 500):
could not find function "plotLearnerPrediction"
```

CLASSIFICATION: 500 TRAINING ITERATIONS

```
## Error in plotLearnerPrediction("classif.nnet", task, size = 50L, maxit = 500):
could not find function "plotLearnerPrediction"
```

Single Hidden Layer Networks for Multi-Class Classification

MULTI-CLASS CLASSIFICATION

- We have only considered regression and binary classification problems so far.
- How can we get a neural network to perform multiclass classification?

MULTI-CLASS CLASSIFICATION

- The first step is to add additional neurons to the output layer.
- Each neuron in the layer will represent a specific class (number of neurons in the output layer = number of classes).

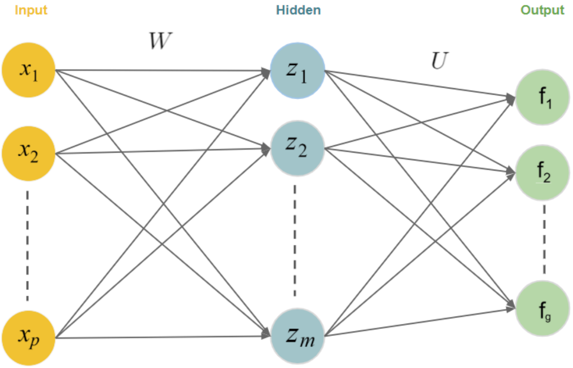


Figure: Structure of a single hidden layer, feed-forward neural network for g-class classification problems (bias term omitted).

MULTI-CLASS CLASSIFICATION

Notation:

- For g-class classification, g output units:

$$\mathbf{f} = (f_1, \dots, f_g)$$

- m hidden neurons $z_1, \dots, z_m$, with

$$z_j = \sigma(\mathbf{W}_j^\top \mathbf{x}), \quad j = 1, \dots, m.$$

- Compute linear combinations of derived features z:

$$f_{in,k} = \boldsymbol{U}_k^\top \mathbf{z}, \quad \mathbf{z} = (z_1, \dots, z_m)^\top, \quad k = 1, \dots, g$$

MULTI-CLASS CLASSIFICATION

- The second step is to apply a softmax activation function to the output layer.

- This gives us a probability distribution over g different possible classes:

$$f_{out,k} = \tau_k(f_{in,k}) = \frac{\exp(f_{in,k})}{\sum_{k'=1}^{g} \exp(f_{in,k'})}$$

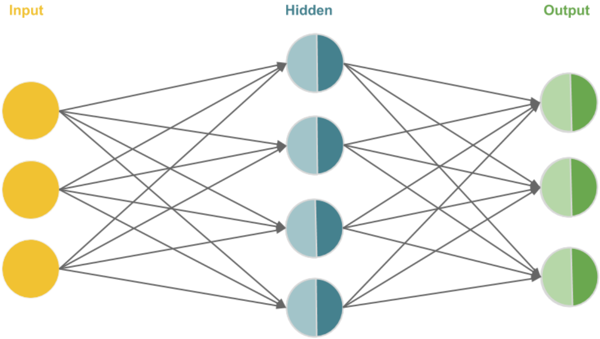
- This is the same transformation used in softmax regression!

- Derivative $\frac{\delta \tau(\mathbf{f}_{in})}{\delta \mathbf{f}_{in}} = \text{diag}(\tau(\mathbf{f}_{in})) - \tau(\mathbf{f}_{in})\tau(\mathbf{f}_{in})^\top$

- It is a “smooth” approximation of the argmax operation, so $\tau((1, 1000, 2)^\top) \approx (0, 1, 0)^\top$ (picks out 2nd element!).

MULTI-CLASS CLASSIFICATION: EXAMPLE

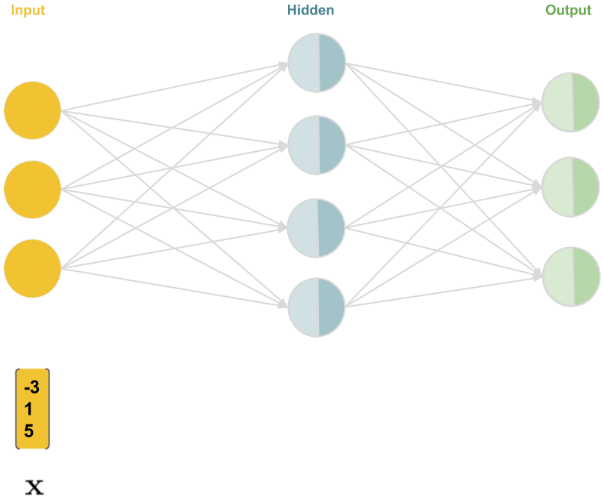
Forward pass (Hidden: Sigmoid, Output: Softmax).



$$\underbrace{\begin{pmatrix} 3 & -9 & 2 \\ 11 & -2 & 7 \\ -6 & 3 & -4 \\ 6 & -1 & 5 \end{pmatrix}}_{W^T} \underbrace{\begin{bmatrix} 5 \\ 2 \\ -1 \\ 1 \end{bmatrix}}_{\mathbf{b}} \qquad \underbrace{\begin{pmatrix} 3 & -12 & 8 & 1 \\ 2 & -3 & 9 & 1 \\ -5 & 1 & -1 & 7 \end{pmatrix}}_{U^T} \underbrace{\begin{bmatrix} 6 \\ 0 \\ -8 \end{bmatrix}}_{\mathbf{c}}$$

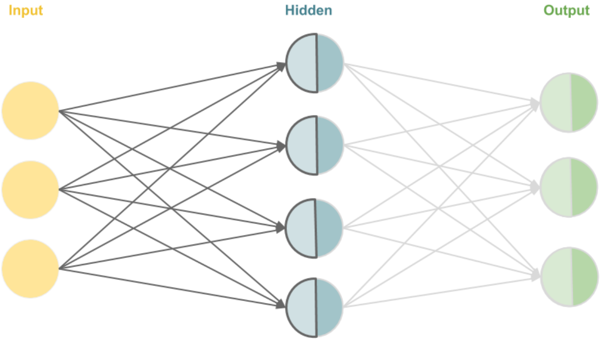
MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).

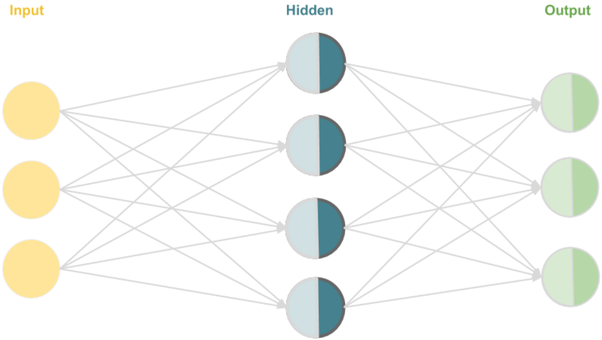


$$\begin{pmatrix}-3\\1\\5\end{pmatrix} \qquad \begin{pmatrix}(-3)*3 + 1*(-9) + 5*2 + 5\\ (-3)*11 + 1*(-2) + 5*7 + 2\\ (-3)*(-6) + 1*3 + 5*(-4) + (-1)\\ (-3)*6 + 1*(-1) + 5*5 + 1\end{pmatrix}$$

$$\mathbf{x} \qquad \mathbf{z}_{in} = \mathbf{w}^T\mathbf{x} + \mathbf{b}$$

MULTI-CLASS CLASSIFICATION: EXAMPLE

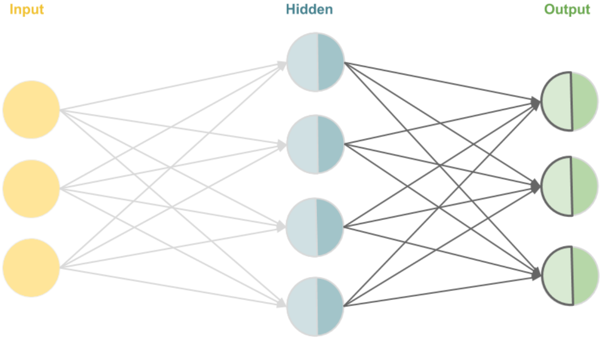
Forward pass (Hidden: Sigmoid, Output: Softmax).



$$\underset{\mathbf{x}}{\begin{pmatrix} -3 \\ 1 \\ 5 \end{pmatrix}} \qquad \underset{\mathbf{z}_{in}}{\begin{pmatrix} -3 \\ 2 \\ 0 \\ 7 \end{pmatrix}} \qquad \underset{\mathbf{z} = \mathbf{z}_{out} = \sigma(\mathbf{z}_{in})}{\begin{pmatrix} 1/\ (1 + \exp(-(-3)) \\ 1/\ (1 + \exp(-2) \\ 1/\ (1 + \exp(-0) \\ 1/\ (1 + \exp(-7) \end{pmatrix}}$$

MULTI-CLASS CLASSIFICATION: EXAMPLE

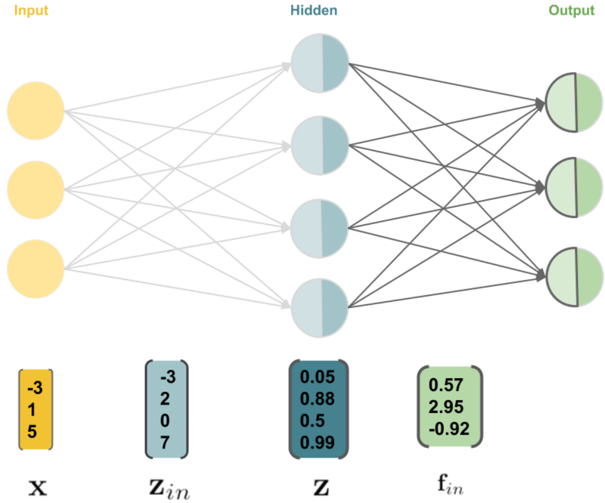
Forward pass (Hidden: Sigmoid, Output: Softmax).



$$\underbrace{\begin{pmatrix} -3 \\ 1 \\ 5 \end{pmatrix}}_{\mathbf{x}} \qquad \underbrace{\begin{pmatrix} -3 \\ 2 \\ 0 \\ 7 \end{pmatrix}}_{\mathbf{z}_{in}} \quad \underbrace{\begin{pmatrix} 0.05 \\ 0.88 \\ 0.5 \\ 0.99 \end{pmatrix}}_{\mathbf{z}} \quad \underbrace{\begin{pmatrix} (0.05)*3 + (0.88)*(-12) + (0.5)*8 + (0.99)*1 + 6 \\ (0.05)*2 + (0.88)*(-3) + (0.5)*9 + (0.99)*1 + 0 \\ (0.05)*(-5) + (0.88)*1 + (0.5)*(-1) + (0.99)*7 + -8 \end{pmatrix}}_{\mathbf{f}_{in} = U^T\mathbf{z} + \mathbf{c}}$$

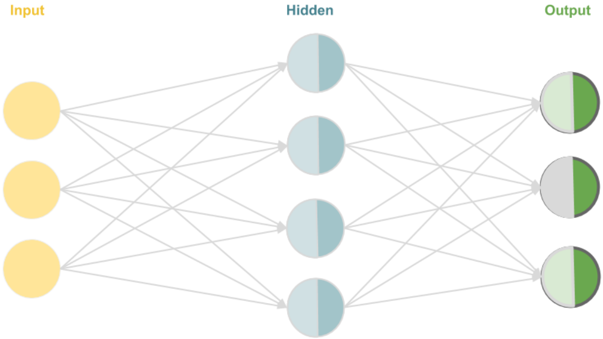
MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



$$\begin{pmatrix}0.57\\2.95\\-0.92\end{pmatrix}$$

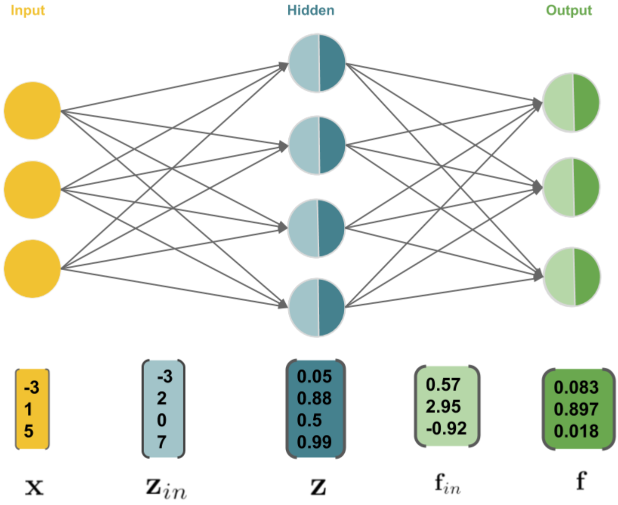
$\mathbf{f}_{in}$

$$\begin{pmatrix}\frac{\exp(0.57)}{\exp(0.57)+\exp(2.95)+\exp(-0.92)}\\ \frac{\exp(2.95)}{\exp(0.57)+\exp(2.95)+\exp(-0.92)}\\ \frac{\exp(-0.92)}{\exp(0.57)+\exp(2.95)+\exp(-0.92)}\end{pmatrix}$$

$$\mathbf{f} = \mathbf{f}_{out} = \tau(\mathbf{f}_{in})$$

MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



SOFTMAX LOSS

- The loss function for a softmax classifier is

$$L(y, f(\mathbf{x})) = -\sum_{k=1}^{g} [y = k] \log \left(\frac{\exp(f_{in,k})}{\sum_{k'=1}^{g} \exp(f_{in,k'})} \right)$$

where $[y = k] = \begin{cases} 1 & \text{if } y = k \\ 0 & \text{otherwise} \end{cases}$.

- This is equivalent to the cross-entropy loss when the label vector $\boldsymbol{y}$ is one-hot coded (e.g. $\mathbf{y} = (0, 0, 1, 0)^\top$).
- Optimization: Again, there is no analytic solution.

SINGLE HIDDEN LAYER NETWORKS: SUMMARY

- We have seen that neural networks are far more flexible than linear models. Neural networks with a single hidden layer are able to approximate any continuous function.
- Yet, in reality, there is no way to make full use of the universal approximation property. The learning algorithm will usually not find the best possible model. At best it finds a locally optimal model.
- The XOR example showed us how neural networks extract features to transform the space and actually learn a kernel (learn a representation).
- Neural networks can perfectly fit noisy data. Thus, neural networks are endangered to over-fit. This is particularly true for a model with a huge hidden layer.
- Fitting neural networks with sigmoidal activation function is nothing else but fitting many weighted logistic regressions!

Multi-Layer Feedforward Neural Networks

FEEDFORWARD NEURAL NETWORKS

- We will now extend the model class once again, such that we allow an arbitrary amount of l (hidden) layers.

- The general term for this model class is (multi-layer) **feedforward networks** (inputs are passed through the network from left to right, no feedback-loops are allowed)

FEEDFORWARD NEURAL NETWORKS

- We can characterize those models by the following chain structure:

$$f(\mathbf{x}) = \tau \circ \phi \circ \sigma^{(l)} \circ \phi^{(l)} \circ \sigma^{(l-1)} \circ \phi^{(l-1)} \circ \ldots \circ \sigma^{(1)} \circ \phi^{(1)}$$

  where $\sigma^{(i)}$ and $\phi^{(i)}$ are the activation function and the weighted sum of hidden layer i, respectively. τ and ϕ are the corresponding components of the output layer.

- Each hidden layer has:
  - an associated weight matrix $\mathbf{W}^{(i)}$, bias $\mathbf{b}^{(i)}$ and activations $\mathbf{z}^{(i)}$ for $i \in \{1 \ldots l\}$
  - $\mathbf{z}^{(i)} = \sigma^{(i)}(\phi^{(i)}) = \sigma^{(i)}(\mathbf{W}^{(i)T}\mathbf{z}^{(i-1)} + \mathbf{b}^{(i)})$, where $\mathbf{z}^{(0)} = \mathbf{x}$.

- Again, without non-linear activations in the hidden layers, the network can only learn linear decision boundaries.

FEEDFORWARD NEURAL NETWORKS

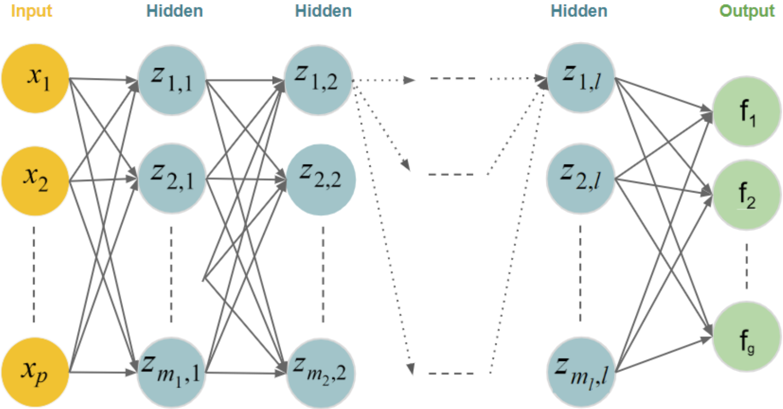
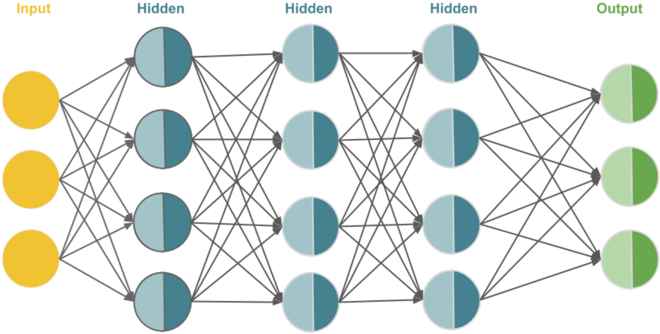


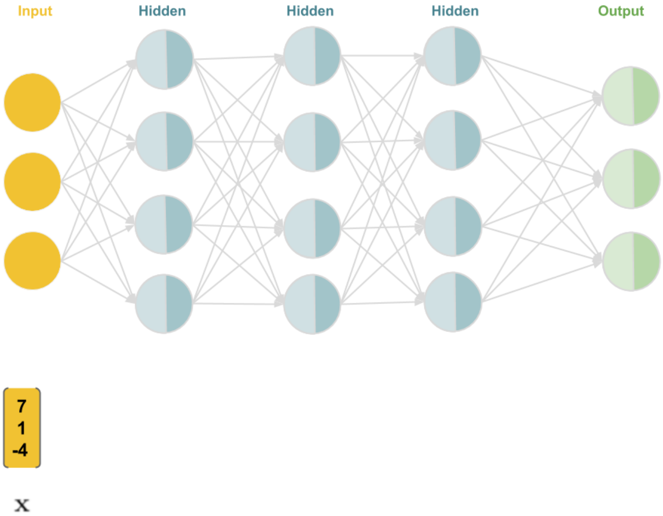
Figure: Structure of a deep neural network with *l* hidden layers (bias terms omitted).

FEEDFORWARD NEURAL NETWORKS: EXAMPLE

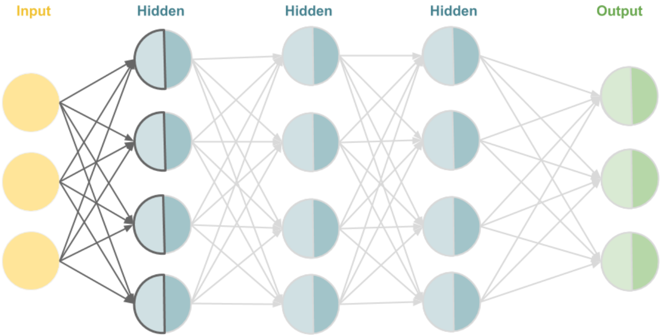


$$\underset{(W^{(1)})^T}{\begin{pmatrix} 13 & -9 & 2 \\ -8 & 0 & 3 \\ 4 & -1 & 5 \\ -3 & 12 & 7 \end{pmatrix}} \underset{\mathbf{b}^{(1)}}{\begin{pmatrix} 5 \\ -2 \\ 2 \\ 11 \end{pmatrix}} \underset{(W^{(2)})^T}{\begin{pmatrix} 1 & 0 & -4 & 1 \\ 0 & 11 & 2 & -14 \\ -1 & 5 & -2 & 16 \\ 0 & -9 & -3 & 4 \end{pmatrix}} \underset{\mathbf{b}^{(2)}}{\begin{pmatrix} -5 \\ 3 \\ 1 \\ -8 \end{pmatrix}} \underset{(W^{(3)})^T}{\begin{pmatrix} 1 & -2 & -18 & -7 \\ 3 & -4 & 8 & 0 \\ -2 & 1 & 21 & 5 \\ 2 & -2 & 11 & -13 \end{pmatrix}} \underset{\mathbf{b}^{(3)}}{\begin{pmatrix} 4 \\ -6 \\ 1 \\ -17 \end{pmatrix}} \underset{U^T}{\begin{pmatrix} 9 & 3 & -1 & -4 \\ -8 & -2 & 14 & 3 \\ 13 & 2 & -9 & -1 \end{pmatrix}} \underset{\mathbf{c}}{\begin{pmatrix} -1 \\ -4 \\ -30 \end{pmatrix}}$$

FEEDFORWARD NEURAL NETWORKS: EXAMPLE



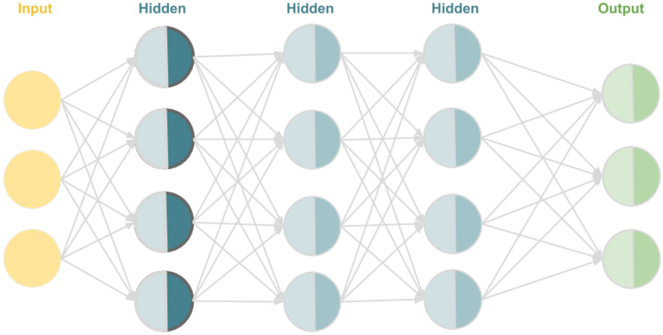
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\begin{bmatrix} 7 \\ 1 \\ -4 \end{bmatrix} \quad \begin{pmatrix} 7*13 + 1*(-9) + (-4)*2 + 5 \\ 7*(-8) + 1*0 + (-4)*3 + (-2) \\ 7*4 + 1*(-1) + (-4)*5 + 2 \\ 7*(-3) + 1*12 + (-4)*7 + 11 \end{pmatrix}$$

$$\mathbf{x} \qquad \mathbf{z}_{in}^{(1)} = W^{(1)T}\mathbf{x} + \mathbf{b}^{(1)}$$

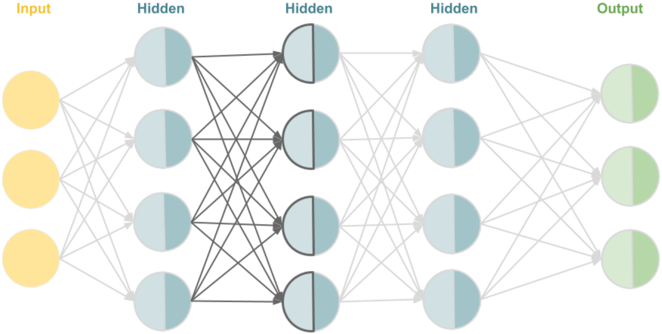
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\begin{bmatrix}7\\1\\-4\end{bmatrix} \quad \begin{bmatrix}79\\-70\\9\\-26\end{bmatrix} \quad \begin{pmatrix}\max(0, 79)\\ \max(0, -70)\\ \max(0, 9)\\ \max(0, -26)\end{pmatrix}$$

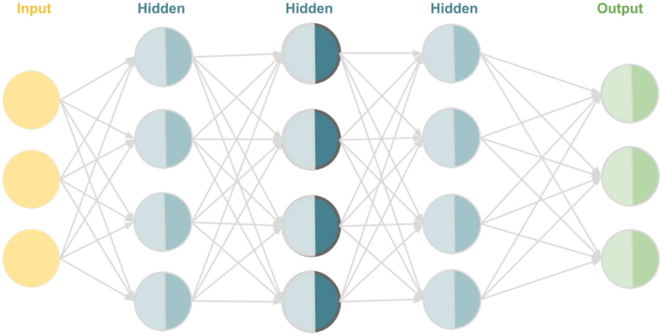
$$\mathbf{x} \quad \mathbf{z}^{(1)}_{in} \quad \mathbf{z}^{(1)} = \mathbf{z}^{(1)}_{out} = \sigma(\mathbf{z}^{(1)}_{in})$$

FEEDFORWARD NEURAL NETWORKS: EXAMPLE



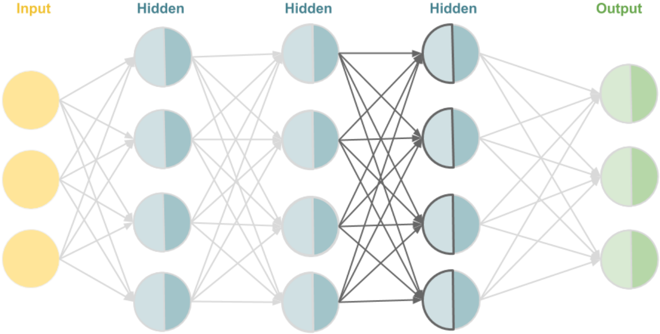
$$\begin{array}{cccc}
\begin{pmatrix}7\\1\\-4\end{pmatrix} & \begin{pmatrix}79\\-70\\9\\-26\end{pmatrix} & \begin{pmatrix}79\\0\\9\\0\end{pmatrix} & \begin{pmatrix}79*1 + 0*0 + 9*(-4) + 0*1 + (-5)\\ 79*0 + 0*11 + 9*2 + 0*(-14) + 3\\ 79*(-1) + 0*5 + 9*(-2) + 0*16 + 1\\ 79*0 + 0*(-9) + 9*(-3) + 0*4 + (-8)\end{pmatrix} \\
\mathbf{x} & \mathbf{z}_{in}^{(1)} & \mathbf{z}^{(1)} & \mathbf{z}_{in}^{(2)} = W^{(2)T}\mathbf{z}^{(1)} + \mathbf{b}^{(2)}
\end{array}$$

FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\begin{array}{ccccc}
\begin{pmatrix}7\\1\\-4\end{pmatrix} & \begin{pmatrix}79\\-70\\9\\-26\end{pmatrix} & \begin{pmatrix}79\\0\\9\\0\end{pmatrix} & \begin{pmatrix}38\\21\\-96\\-35\end{pmatrix} & \begin{pmatrix}\max(0, 38)\\\max(0, 21)\\\max(0, -96)\\\max(0, -36)\end{pmatrix} \\
\mathbf{x} & \mathbf{z}_{in}^{(1)} & \mathbf{z}^{(1)} & \mathbf{z}_{in}^{(2)} & \mathbf{z}^{(2)} = \mathbf{z}_{out}^{(2)} = \sigma(\mathbf{z}_{in}^{(2)})
\end{array}$$

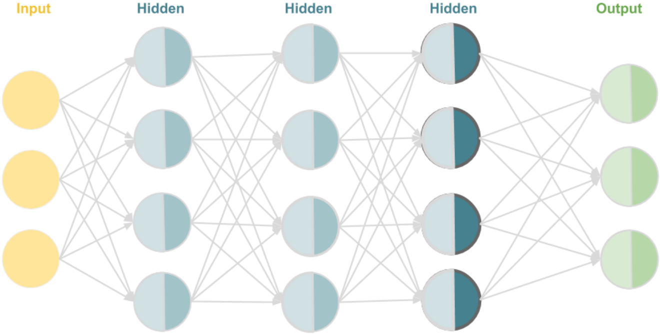
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\begin{pmatrix}7\\1\\-4\end{pmatrix} \quad \begin{pmatrix}79\\-70\\9\\-26\end{pmatrix} \quad \begin{pmatrix}79\\0\\9\\0\end{pmatrix} \quad \begin{pmatrix}38\\21\\-96\\-35\end{pmatrix} \quad \begin{pmatrix}38\\21\\0\\0\end{pmatrix} \quad \begin{pmatrix}38*1 + 21*(-2) + 0*(-18) + 0*(-7) + 4\\38*3 + 21*(-4) + 0*8 + 0*0 + (-6)\\38*(-2) + 21*1 + 0*21 + 0*5 + 1\\38*2 + 21*(-2) + 0*11 + 0*(-13) + (-17)\end{pmatrix}$$

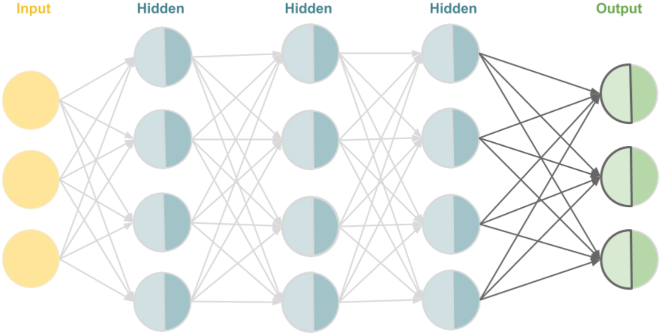
$$\mathbf{x} \quad \mathbf{z}_{in}^{(1)} \quad \mathbf{z}^{(1)} \quad \mathbf{z}_{in}^{(2)} \quad \mathbf{z}^{(2)} \quad \mathbf{z}_{in}^{(3)} = W^{(3)T}\mathbf{z}^{(2)} + \mathbf{b}^{(3)}$$

FEEDFORWARD NEURAL NETWORKS: EXAMPLE



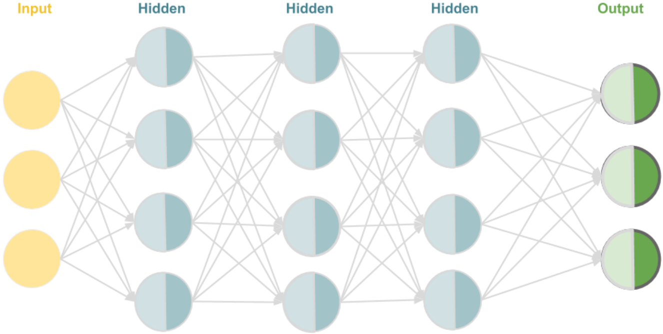
$$\begin{array}{ccccccc}
\begin{pmatrix}7\\1\\-4\end{pmatrix} & \begin{pmatrix}79\\-70\\9\\-26\end{pmatrix} & \begin{pmatrix}79\\0\\9\\0\end{pmatrix} & \begin{pmatrix}38\\21\\-96\\-35\end{pmatrix} & \begin{pmatrix}38\\21\\0\\0\end{pmatrix} & \begin{pmatrix}0\\24\\-54\\17\end{pmatrix} & \begin{pmatrix}\max(0, 0)\\\max(0, 24)\\\max(0, -54)\\\max(0, 17)\end{pmatrix} \\
\mathbf{x} & \mathbf{z}_{in}^{(1)} & \mathbf{z}^{(1)} & \mathbf{z}_{in}^{(2)} & \mathbf{z}^{(2)} & \mathbf{z}_{in}^{(3)} & \mathbf{z}^{(3)} = \mathbf{z}_{out}^{(3)} = \sigma(\mathbf{z}_{in}^{(3)})
\end{array}$$

FEEDFORWARD NEURAL NETWORKS: EXAMPLE



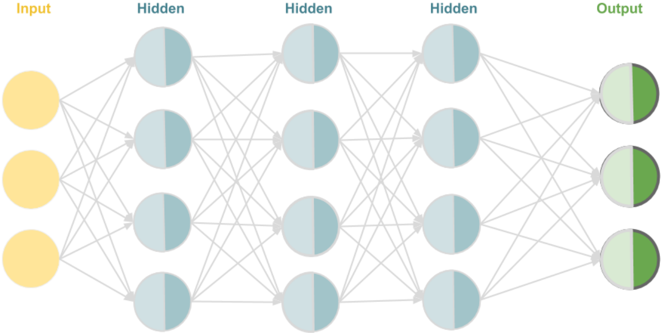
$$\underset{\mathbf{x}}{\begin{pmatrix}7\\1\\-4\end{pmatrix}} \quad \underset{\mathbf{z}_{in}^{(1)}}{\begin{pmatrix}79\\-70\\9\\-26\end{pmatrix}} \quad \underset{\mathbf{z}^{(1)}}{\begin{pmatrix}79\\0\\9\\0\end{pmatrix}} \quad \underset{\mathbf{z}_{in}^{(2)}}{\begin{pmatrix}38\\21\\-96\\-35\end{pmatrix}} \quad \underset{\mathbf{z}^{(2)}}{\begin{pmatrix}38\\21\\0\\0\end{pmatrix}} \quad \underset{\mathbf{z}_{in}^{(3)}}{\begin{pmatrix}0\\24\\-54\\17\end{pmatrix}} \quad \underset{\mathbf{z}^{(3)}}{\begin{pmatrix}0\\24\\0\\17\end{pmatrix}} \quad \underset{\mathbf{f}_{in} = U^T\mathbf{z}^{(3)} + \mathbf{c}}{\begin{pmatrix}24*3 + 17*(-4) + (-1)\\24*(-2) + 17*3 + (-4)\\24*2 + 17*(-1) + (-30)\end{pmatrix}}$$

FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\underset{\mathbf{x}}{\begin{pmatrix}7\\1\\-4\end{pmatrix}} \quad \underset{\mathbf{z}_{in}^{(1)}}{\begin{pmatrix}79\\-70\\9\\-26\end{pmatrix}} \quad \underset{\mathbf{z}^{(1)}}{\begin{pmatrix}79\\0\\9\\0\end{pmatrix}} \quad \underset{\mathbf{z}_{in}^{(2)}}{\begin{pmatrix}38\\21\\-96\\-35\end{pmatrix}} \quad \underset{\mathbf{z}^{(2)}}{\begin{pmatrix}38\\21\\0\\0\end{pmatrix}} \quad \underset{\mathbf{z}_{in}^{(3)}}{\begin{pmatrix}0\\24\\-54\\17\end{pmatrix}} \quad \underset{\mathbf{z}^{(3)}}{\begin{pmatrix}0\\24\\0\\17\end{pmatrix}} \quad \underset{\mathbf{f}_{in}}{\begin{pmatrix}3\\-1\\1\end{pmatrix}}$$

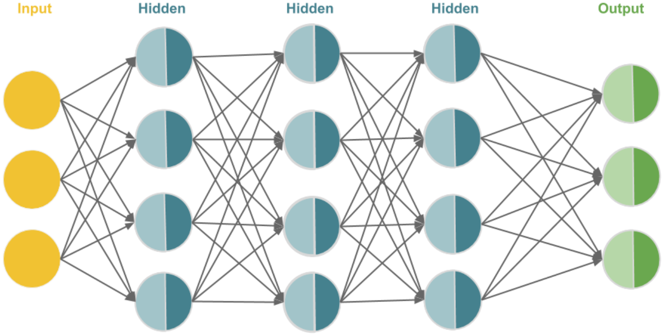
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \quad \begin{pmatrix} \frac{\exp(3)}{\exp 3 + \exp(-1) + \exp(1)} \\ \frac{\exp(-1)}{\exp 3 + \exp(-1) + \exp(1)} \\ \frac{\exp(1)}{\exp 3 + \exp(-1) + \exp(1)} \end{pmatrix}$$

$$\mathbf{f}_{in} \qquad \mathbf{f} = \mathbf{f}_{out} = \tau(\mathbf{f}_{in})$$

FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$\mathbf{x}$	$\mathbf{z}_{in}^{(1)}$	$\mathbf{z}^{(1)}$	$\mathbf{z}_{in}^{(2)}$	$\mathbf{z}^{(2)}$	$\mathbf{z}_{in}^{(3)}$	$\mathbf{z}^{(3)}$	$\mathbf{f}_{in}$	$\mathbf{f}$
7	79	79	38	38	0	0	3	0.866
1	-70	0	21	21	24	24	-1	0.015
-4	9	9	-96	0	-54	0	1	0.117
	-26	0	-35	0	17	17		

WHY ADD MORE LAYERS?

- Multiple layers allow for the extraction of more and more abstract representations.

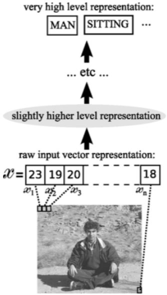

very high level representation:
MAN
SITTING
...
... etc ...
slightly higher level representation
raw input vector representation:
x =
23
19
20
18
x_1
x_2
x_3
x_n

Figure: Y. Bengio, Learning Deep Architectures for AI, Foundations and trends® in Machine Learning, 2009

WHY ADD MORE LAYERS?

- Each layer in a feed-forward neural network adds its own degree of non-linearity to the model.

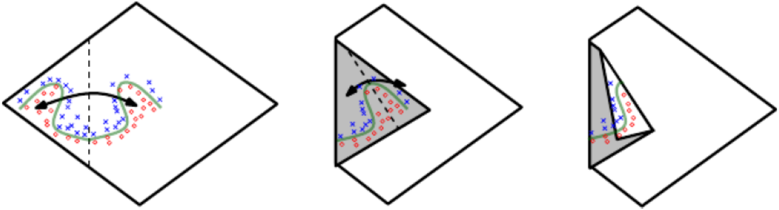


Figure: An intuitive, geometric explanation of the exponential advantage of deeper networks formally (Montúfar et al. (2014)).

Deep Learning

DEEP NEURAL NETWORKS

- Neural networks today can have dozens or even hundreds of hidden layers. The greater the number of layers, the "deeper" the network.
- Historically, deep neural networks were very challenging to train for several reasons.
- For one thing, the use of sigmoid activations (such as logistic sigmoid and tanh) significantly slowed down training due to a phenomenon known as “vanishing gradients”. The introduction of the ReLU activation largely solved this problem.
- Additionally, training deep neural networks on CPUs was too slow to be practical. Switching over to GPUs (Graphics Processing Units) cut down training time by more than an order of magnitude.
- Another reason neural networks were not popular until the late ’00s is that when dataset sizes are small, other models (such as SVMs) and techniques (such as feature engineering) outperform them.

DEEP NEURAL NETWORKS

- Therefore, the availability of large datasets (such as ImageNet) and novel architectures that are capable to handle even complex tensor-shaped data (e.g. CNNs for image data), significantly faster hardware, and equally better optimization and regularization methods made it feasible to successfully implement deep neural networks in the last decade.
- An increase in depth often translates to an increase in performance on a given task.
- State-of-the-art neural networks, however, are much more sophisticated than the simple architectures we have encountered so far. (Stay tuned!)
- The term "**deep learning**" encompasses all of these developments and refers to the field as a whole.

References

REFERENCES

Guido Montúfar, Razvan Pascanu, Kyunghyun Cho and Yoshua Bengio (2014)
On the Number of Linear Regions of Deep Neural Networks
https://arxiv.org/pdf/1402.1869.pdf

Yann LeCun and Corinna Cortes (2010)
MNIST handwritten digit database
http://yann.lecun.com/exdb/mnist/

Yann Lecun, Leon Bottou, Genevieve B. Orr and Klaus-Robert Muller (1998)
Efficient BackProp
http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

Geoffrey Hinton and Ruslan Salakhutdinov (2006)
Reducing the Dimensionality of Data with Neural Networks
https://www.cs.toronto.edu/%7Ehinton/science.pdf

Ian Goodfellow, Yoshua Bengio and Aaron Courville (2016)
Deep Learning
http://www.deeplearningbook.org/

REFERENCES

Henry W. Lin, Max Tegmark, and David Rolnick (2016)
Why does deep and cheap learning work so well?
https://arxiv.org/pdf/1608.08225.pdf