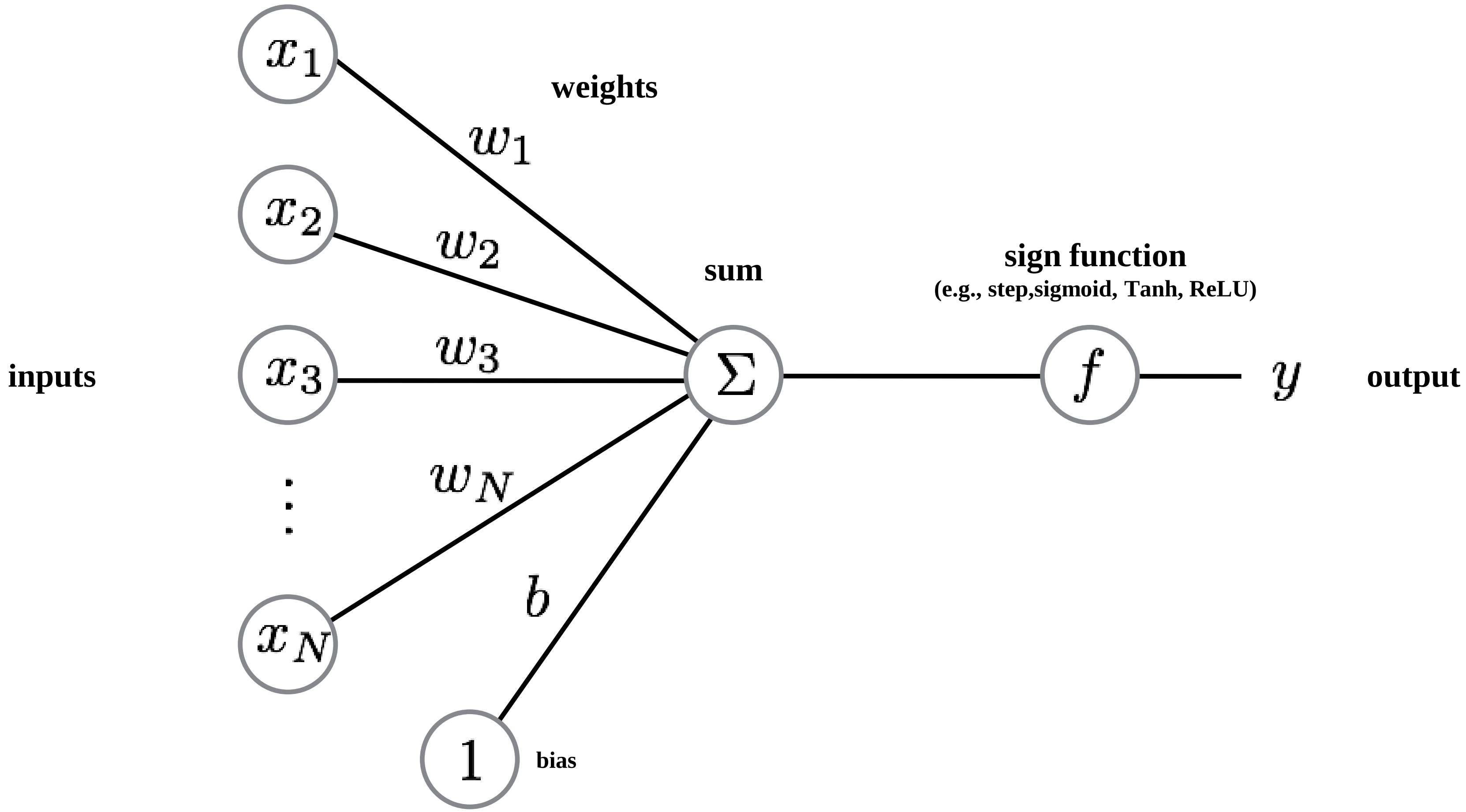
Electrical and Computer Engineering Department
Tarbiat Modares University

Deep Feedforward Networks

Foad Ghaderi, PhD

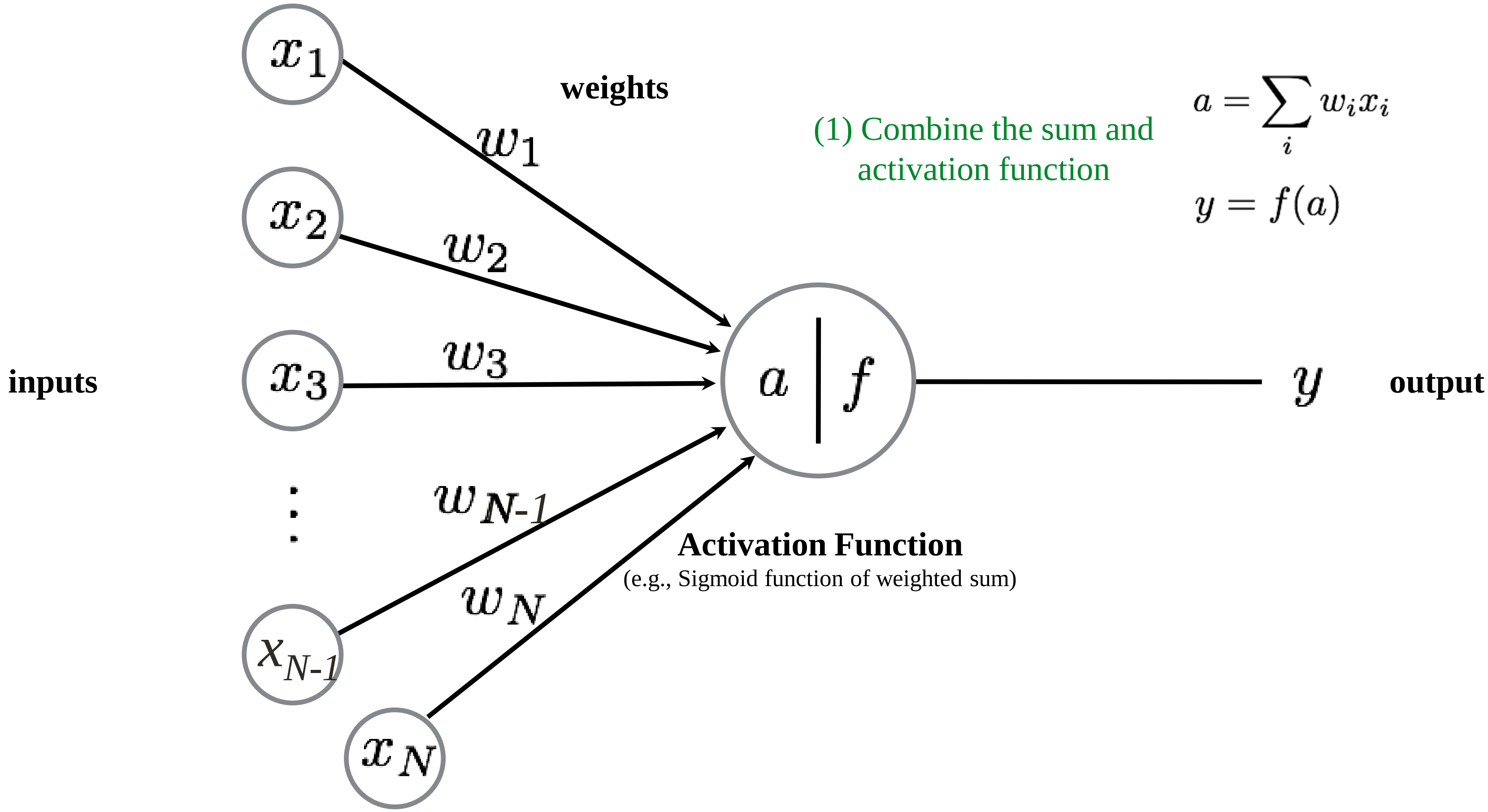
Feedforward networks

The Perceptron



Feedforward networks

The Perceptron



(1) Combine the sum and activation function

$$a = \sum_i w_i x_i$$

$$y = f(a)$$

(2) suppress the bias term (less clutter)

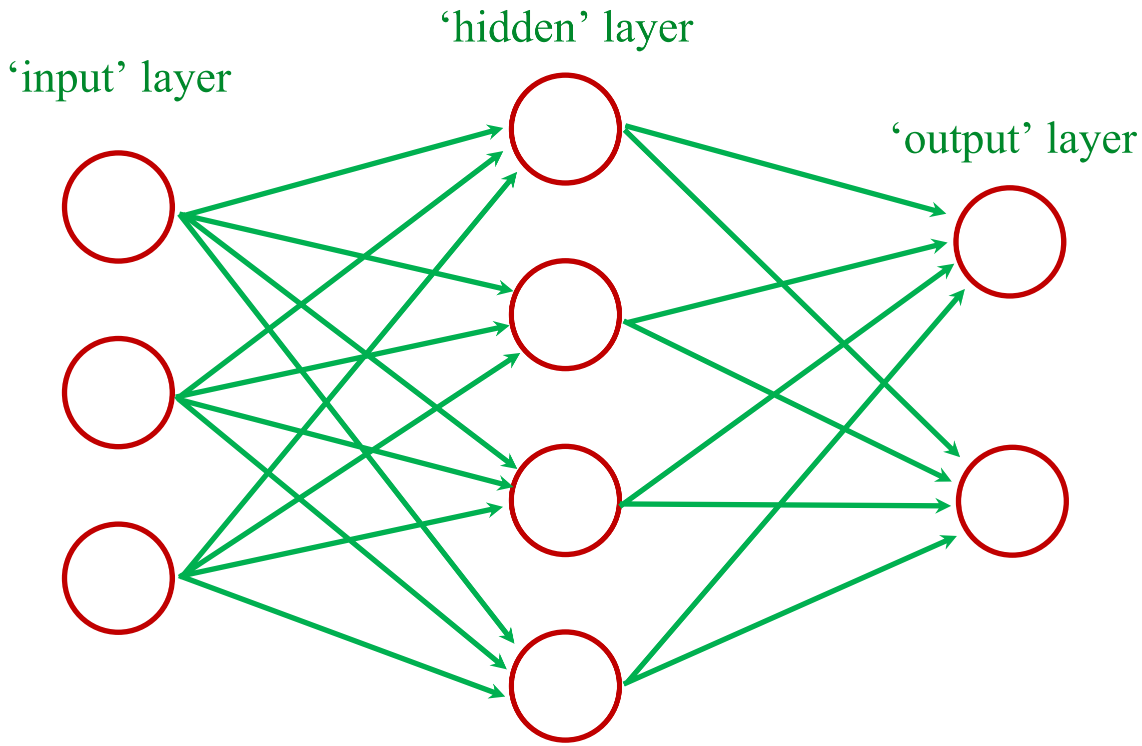
$$x_N = 1$$

$$w_N = b$$

Feedforward networks

A feedforward network (*neural networks*, or *multilayer perceptrons* (*MLPs*)) defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ that result in the best function approximation.

… a collection of connected perceptrons

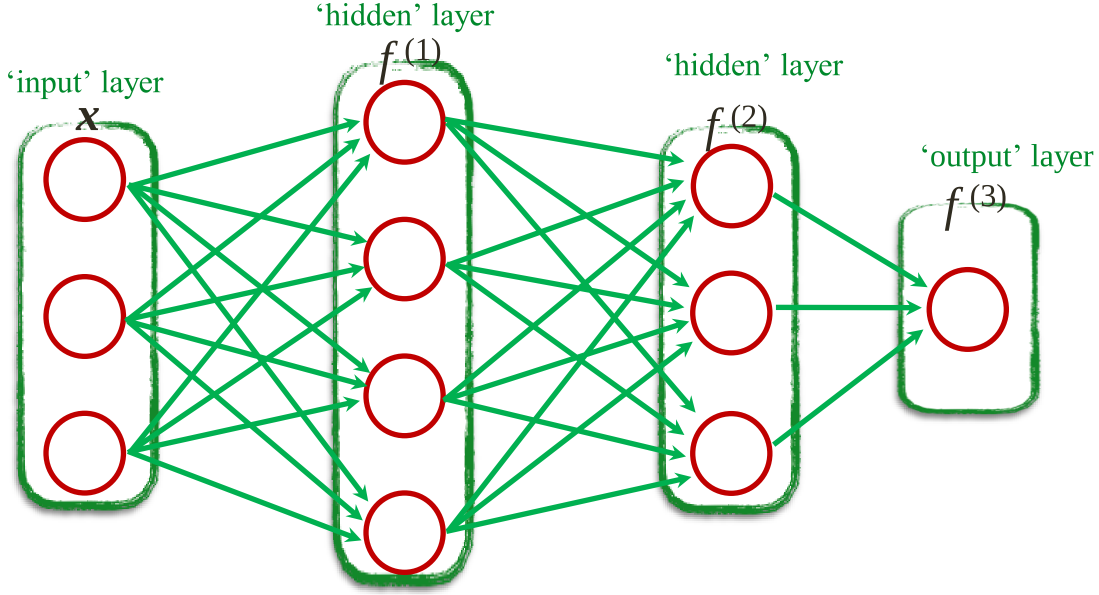


Feedforward networks

There are no *feedback* connections in which outputs of the model are fed back into itself.

The model is associated with a directed acyclic graph describing how the functions are composed together.

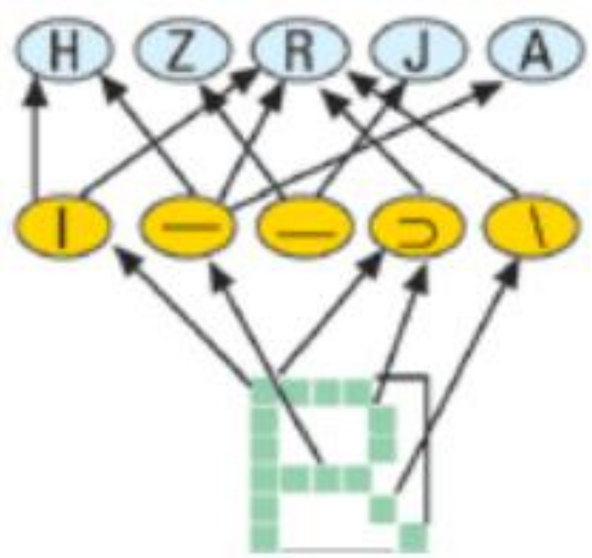
Example, we might have three functions $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$ connected in a chain, to form $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$.



Feedforward networks

Example:

Optical Character Recognition (OCR)

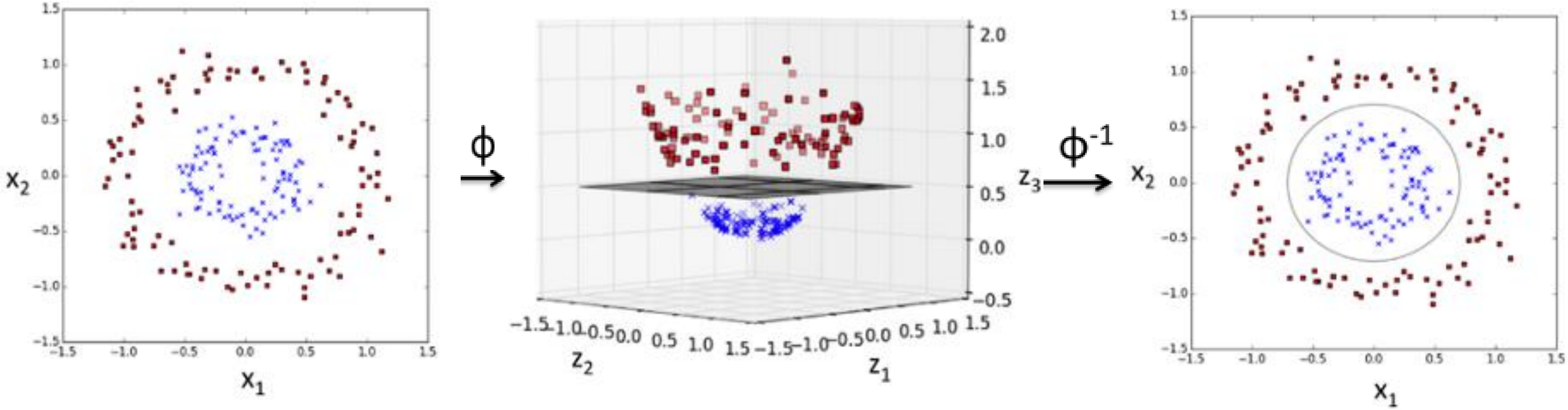


Feedforward networks

Simplest feedforward networks are linear models e.g. logistic regression and linear regression.

- ✓ May fit efficiently and reliably, either in closed form or with convex optimization.
- ✘ The model capacity is limited to linear functions.

To extend linear models to represent nonlinear functions of $\boldsymbol{x}$, we can apply the linear model not to $\boldsymbol{x}$ itself but to a transformed input $\phi(\boldsymbol{x})$, where ϕ is a nonlinear transformation.



Feedforward networks

The question is then how to choose the mapping ϕ.

1. One option is to use a very generic ϕ, such as the infinite-dimensional ϕ that is implicitly used by kernel machines based on the RBF kernel.
   - ✘ Generalization to the test set often remains poor
   - ✘ Do not encode enough prior information to solve advanced problems
2. To manually engineer ϕ
   - ✘ Domain specific
   - ✘ With little transfer between domains.
3. To learn ϕ (the strategy of deep learning)

$$y = f(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w}) = \phi(\boldsymbol{x}; \boldsymbol{\theta})^\top \boldsymbol{w}$$

- ✘ Gives up on the convexity of the training problem
- ✓ The human designer only needs to find the right general function family

Parameters:

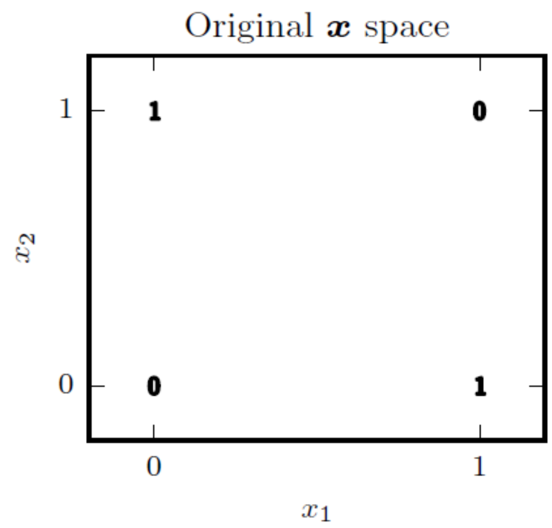
- ❑ $\boldsymbol{\theta}$ is used to learn ϕ from a broad class of functions,
- ❑ $\boldsymbol{w}$ maps from $\phi(\boldsymbol{x})$ to the desired output.

Feedforward networks

Example: Learning XOR

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

Our model provides a function $y = f(\boldsymbol{x};\theta)$ and our learning algorithm will adapt the parameters θ to make f as similar as possible to f^*.



- No concerns about statistical generalization.
- Perform correctly on X = {$[0,0]^T$, $[0,1]^T$, $[1,0]^T$, and $[1,1]^T$}.
- The only challenge is to fit the training set.

Feedforward networks

Example: Learning XOR
Solution 1: Regression (mean squared error loss function)

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

The proposed linear model for *f* is:

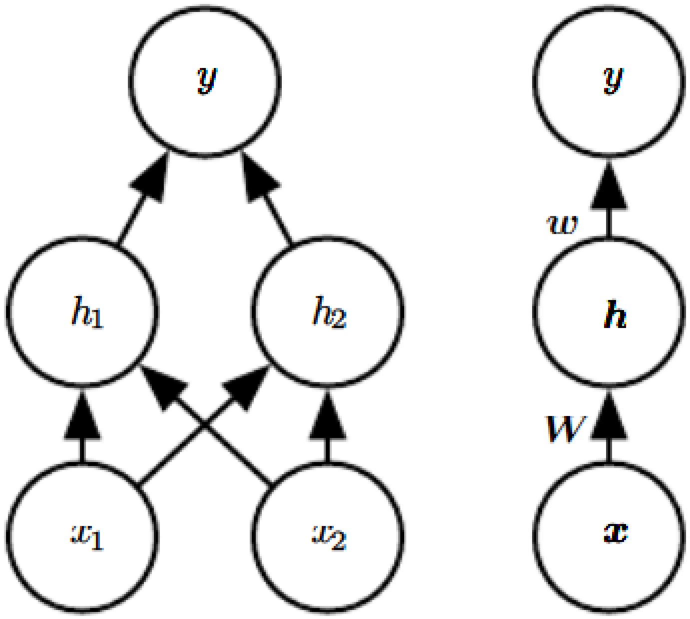
$$f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^\top \boldsymbol{w} + b$$

Using normal equations we obtain ***w*** = 0 and b = 0.5.

Feedforward networks

Example: Learning XOR
Solution 2: A feedforward network with one hidden layer

The network contains two functions chained together:
$\boldsymbol{h} = f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$ and
$y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$,
with the complete model being
$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}(f^{(1)}(\boldsymbol{x}))$.



A nonlinear function is required:
For example: an affine transformation controlled by learned parameters, followed by a fixed, nonlinear function (activation function)

$$h = g(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{c})$$

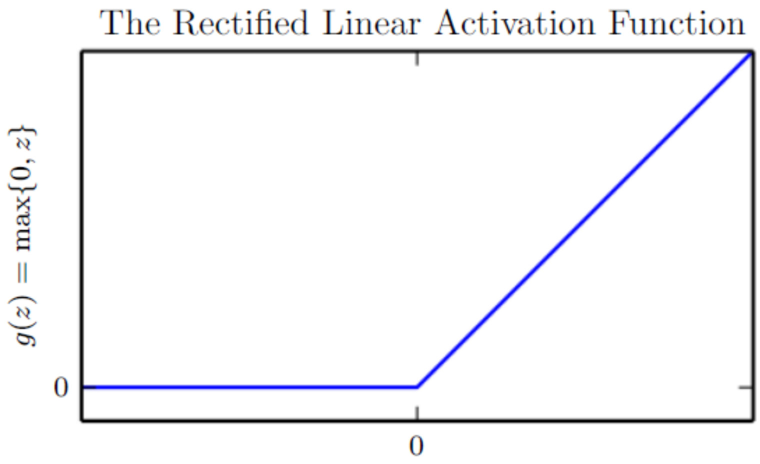
Feedforward networks

Example: Learning XOR

Solution 2: A feedforward network with one hidden layer

$h = g(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{c})$

g is the activation function (the default recommendation is to use the *rectified linear unit* or ReLU: $g(z) = max\{0, z\}$)



$$\longrightarrow \quad f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b$$

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

Feedforward networks

Example: Learning XOR

Solution 2: A feedforward network with one hidden layer

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b$$

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

Assume $\boldsymbol{X}$ is the vector of all possible inputs

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \rightarrow \boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix} \xrightarrow{+c} \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

all samples lie along a line with slope 1.

Feedforward networks

Example: Learning XOR

Solution 2: A feedforward network with one hidden layer

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b$$

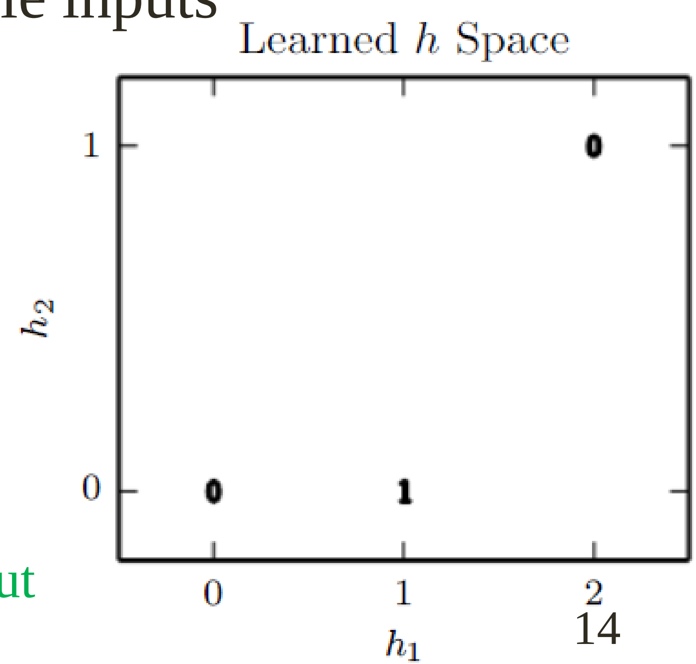
$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \quad \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \quad b = 0$$

Assume ***X*** is the vector of all possible inputs

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \xrightarrow{\text{apply ReLU}} \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

$$\xrightarrow{\text{multiply by } \boldsymbol{w}} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

For fixed h_2, output increases in h_1



Feedforward networks

Example: Learning XOR
Summary:

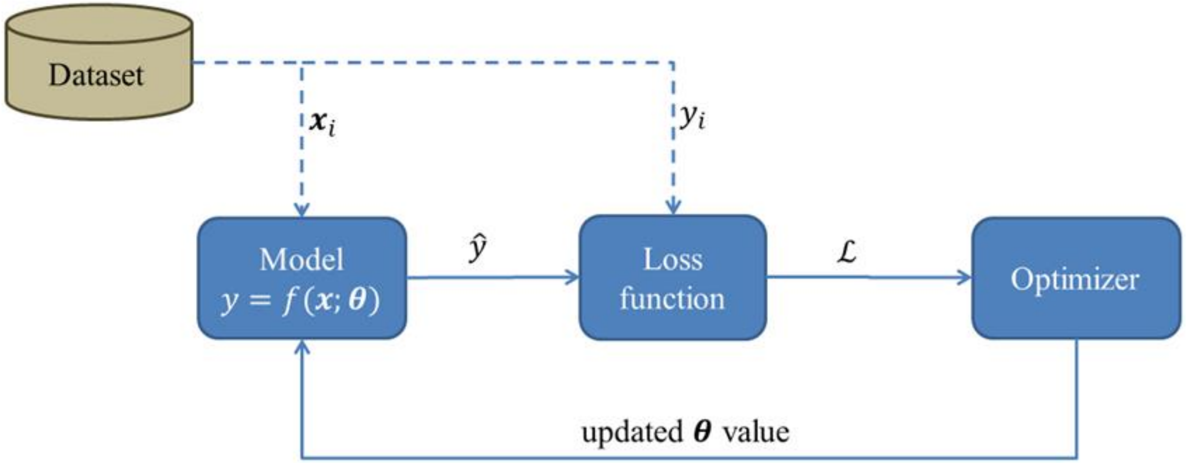
- Selecting alternative solution resulted in zero error
- In real world application there are many parameters. It is not easy to guess.
- Gradient descent optimization methods can estimate parameters with little errors

Gradient-Based Learning

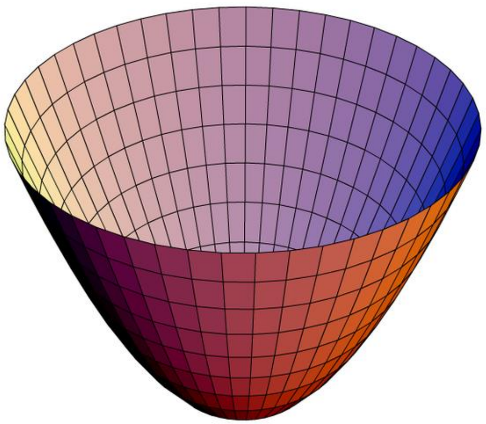
Neural Network training needs to specify:

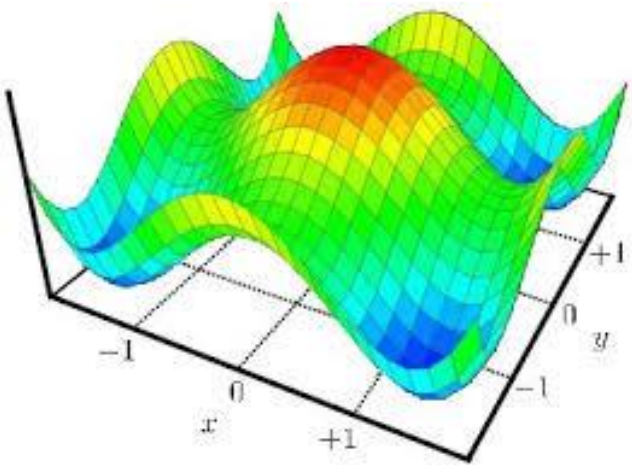
- Model family, e.g., linear with basis functions
- Cost function, e.g., MSE
- Optimization procedure, e.g., gradient descent

The largest difference between the linear models and neural networks is that the nonlinearity of a neural network causes most interesting loss functions to become non-convex.



Gradient-Based Learning





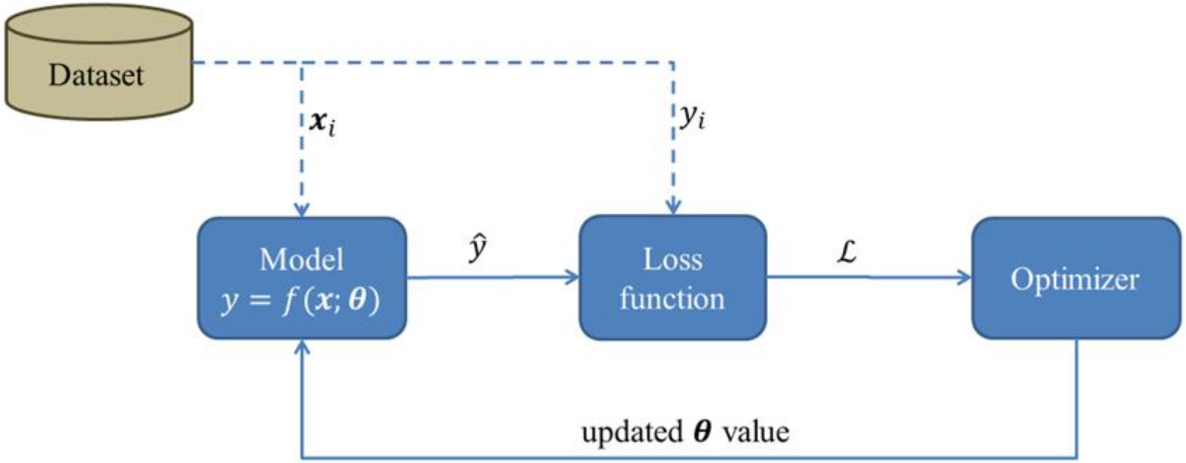
Neural networks	Linear equation solvers	Convex problems
Iterative, gradient-based optimizers	Linear regression models	Convex optimization algorithms
Merely drive the cost function to a very low value	Exact result	Global convergence guarantees

Gradient-Based Learning

Cost functions

The cost functions for neural networks are more or less the same as those for other parametric models, such as linear models.

- In most cases, our parametric model defines a distribution $p(\boldsymbol{y} \mid \boldsymbol{x};\boldsymbol{\theta})$ and we simply use the principle of maximum likelihood.
- Sometimes, we merely predict some statistic of $\boldsymbol{y}$ conditioned on $\boldsymbol{x}$.

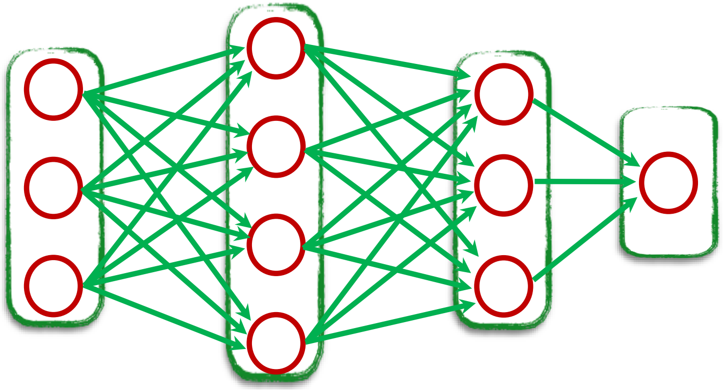


Gradient-Based Learning

Output Units

we suppose that the feedforward network provides a set of hidden features defined by $\boldsymbol{h} = f(\boldsymbol{x}; \boldsymbol{\theta})$. The role of the output layer is then to provide some additional transformation from the features to complete the task that the network must perform.

Any kind of neural network unit that may be used as an output can also be used as a hidden unit.



Gradient-Based Learning

Output Units

Linear Units for Gaussian Output Distributions

An affine transformation with no nonlinearity

Given features h, $\hat{\boldsymbol{y}} = \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b}$

Linear output layers are often used to produce the mean of a conditional Gaussian distribution:

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$$

Solving this by maximizing the log-likelihood (equivalent to minimizing the mean squared error).

Because linear units do not saturate, they pose little difficulty for gradient-based optimization algorithms.

Gradient-Based Learning

Output Units

Sigmoid Units for Bernoulli Output Distributions

Predicting the value of a binary variable *y* (Classification problems with two classes).

The neural net needs to predict only $P\,(y = 1 \mid \boldsymbol{x})$. For this number to be a valid probability, it must lie in the interval [0, 1].

This constraint needs careful design

Suppose we use:

$$P(y = 1 \mid \boldsymbol{x}) = \max\left\{0, \min\left\{1, \boldsymbol{w}^\top \boldsymbol{h} + b\right\}\right\}$$

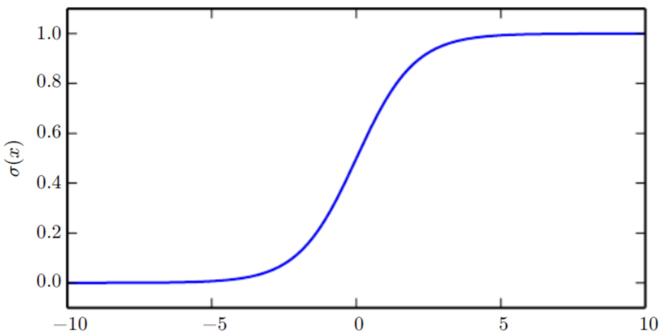
- A conditional distribution can be defined, but it cannot be effectively trained with gradient descent,
- For gradient of 0, learning algorithm cannot be guided.

Gradient-Based Learning

Output Units

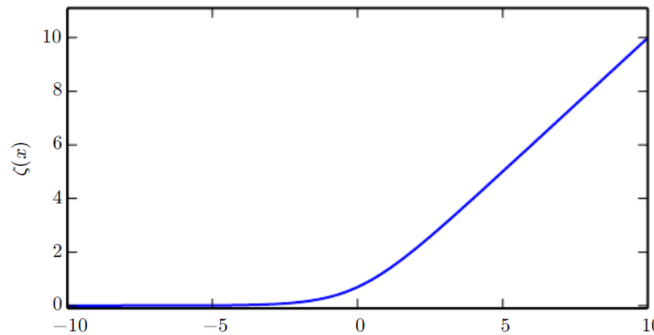
Sigmoid Units for Bernoulli Output Distributions

Using sigmoid always gives a strong gradient



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

Softplus:
The name comes from its being a smoothed or softened version of $x^+=\max(0, x)$



$$\zeta(x) = \log\left(1+\exp(x)\right)$$

$$\sigma(x) = \frac{\exp(x)}{\exp(x)+\exp(0)}$$

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x))$$

$$1-\sigma(x) = \sigma(-x)$$

$$\log\sigma(x) = -\zeta(-x)$$

$$\frac{d}{dx}\zeta(x) = \sigma(x)$$

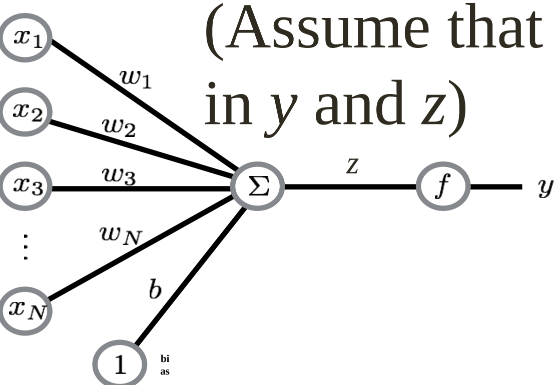
Gradient-Based Learning

Output Units

Sigmoid Units for Bernoulli Output Distributions

We define a probability distribution over *y* using the value *z*.

- Construct an un-normalized log probability distribution (Assume that the un-normalized log probabilities are linear in *y* and *z*)



$$\log \tilde{P}(y) = yz$$

$$\tilde{P}(y) = \exp(yz)$$

- We then normalize to see that this yields a Bernoulli distribution controlled by a sigmoidal transformation of *z*:

$$P(y) = \frac{\exp(yz)}{\sum_{y'=0}^{1} \exp(y'z)}$$

$$P(y) = \sigma\left((2y-1)z\right).$$

Gradient-Based Learning

Output Units

Sigmoid Units for Bernoulli Output Distributions

The loss function for maximum likelihood learning of a Bernoulli parametrized by a sigmoid is

$$\begin{aligned} J(\boldsymbol{\theta}) &= -\log P(y \mid \boldsymbol{x}) \\ &= -\log \sigma\left((2y-1)z\right) \\ &= \zeta\left((1-2y)z\right). \end{aligned}$$

y	z	
1	positive	sat
	Negative	
0	positive	
	negative	sat

This loss function saturates only when $(1-2y)z$ is very negative.

❑ Saturation thus occurs only when the model already has the right answer:

- $y = 1$ and z is very positive, or
- $y = 0$ and z is very negative.

Gradient-Based Learning

Output Units

Sigmoid Units for Bernoulli Output Distributions

$$
\begin{aligned}
J(\boldsymbol{\theta}) &= -\log P(y \mid \boldsymbol{x}) \\
&= -\log \sigma\left((2y-1)z\right) \\
&= \zeta\left((1-2y)z\right).
\end{aligned}
$$

When z has the wrong sign, the argument to the softplus function, $(1 - 2y)z$, may be simplified to $|z|$.

- As $|z|$ becomes large while z has the wrong sign, the softplus function asymptotes toward simply returning its argument $|z|$. The derivative with respect to z asymptotes to $\text{sign}(z)$, so, in the limit of extremely incorrect z, the softplus function does not shrink the gradient at all.
- This property is very useful because it means that gradient-based learning can act to quickly correct a mistaken z.

Gradient-Based Learning

Output Units

Sigmoid Units for Bernoulli Output Distributions

When using other loss functions, such as Mean squared error:

The loss can saturate anytime $\sigma(z)$ saturates.

Sigmoid activation function:

saturates to 0 when z becomes very negative and

saturates to 1 when z becomes very positive.

The gradient can shrink too small to be useful for learning whenever this happens,

Therefore, maximum likelihood is almost always the preferred approach to training sigmoid output units.