ARISTOTLE UNIVERSITY OF THESSALONIKI

FACULTY OF ENGINEERING

Pattern Recognition & Machine Learning

Neural Networks

Panagiotis C. Petrantonakis

Assistant Professor

Dept. of Electrical and Computer Engineering

ppetrant@ece.auth.gr

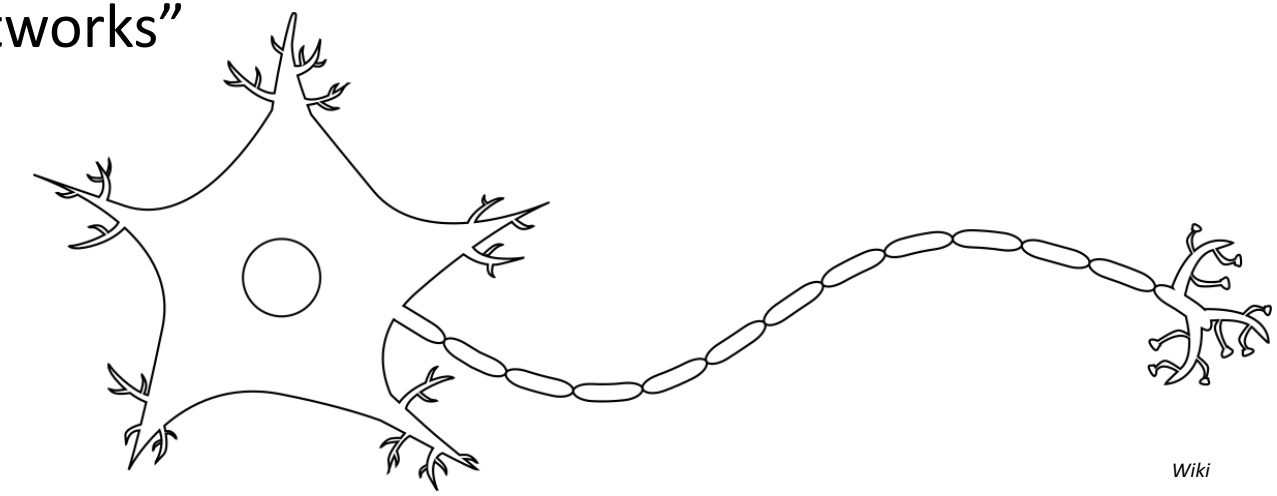
Fall Semester

Until now…

- Dataset: inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$ and targets $\boldsymbol{t} = (t_1, \dots, t_N)$, where $\boldsymbol{x}_n \in \mathbb{R}^D$
- Fixed features: $\boldsymbol{\varphi}(\boldsymbol{x}) = \big(\varphi_0(\boldsymbol{x}), \dots, \varphi_M(\boldsymbol{x})\big)$, where $\varphi_0 = 1$
- Classification: $y(\boldsymbol{x}, \boldsymbol{w}) = f(\boldsymbol{w}^T \boldsymbol{\varphi}(\boldsymbol{x}))$
  - f: nonlinear activation function (e.g., logistic sigmoid)
- In the SVM case we had basis functions centered on training samples and then selecting a number of them to define the decision rule (support vectors)
  - convex objective function
  - again fixed features $\boldsymbol{\varphi}(\boldsymbol{x})$
- What if we want adaptive basis functions?
  - Neural Networks!

Neural Network

- Artificial “Neuron” and “Neural Networks”
  - Mathematical interpretation of information processing in the brain.
  - McCulloch and Pitts (1943)
  - Widrow and Hoff (1960)
  - Rosenblatt (1962)
  - Rumelhart et al. (1986)
  - etc. etc.



Wiki

- A vast variety of different implementations and algorithms regarding the above mentioned mathematical interpretation.
  - Biological plausibility

- Here we will talk for a class of NN that are of great practical value:
  - Multilayer Perceptrons (MLP)!

Perceptron

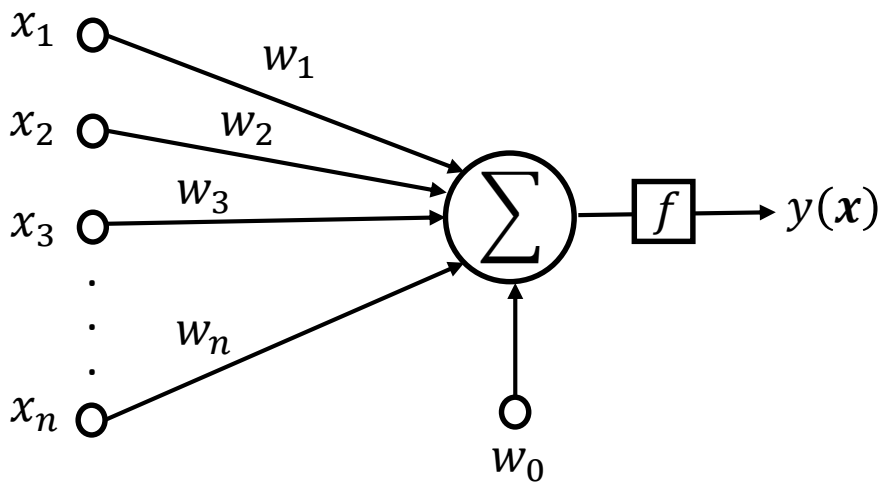
$$y(\boldsymbol{x}) = f(\boldsymbol{w}^t \boldsymbol{x} + w_0)$$

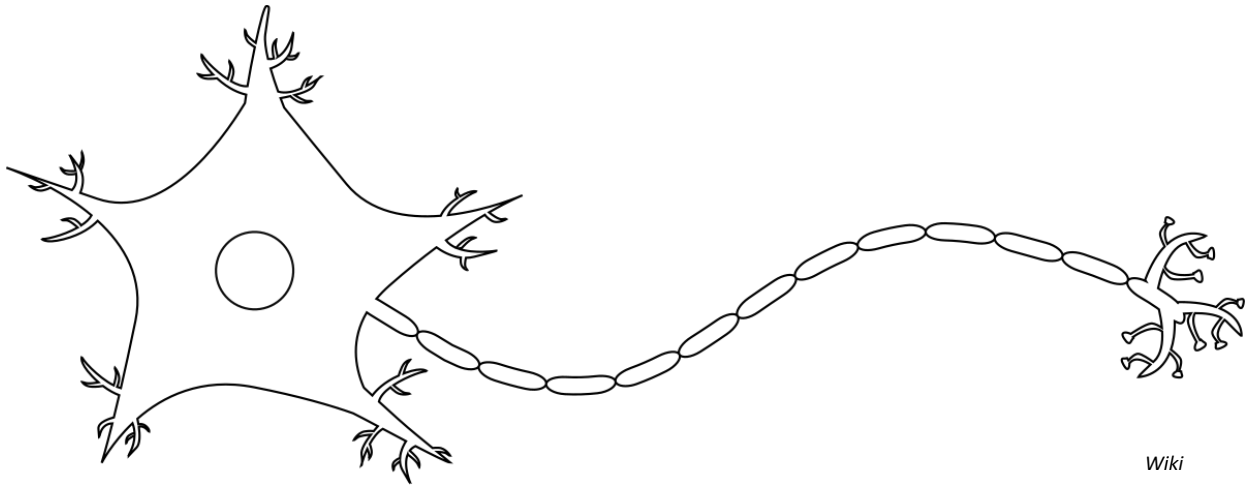
If we incorporate w_0:

$$y(\boldsymbol{x}) = f(\boldsymbol{w}^t \boldsymbol{x})$$

In the general case:

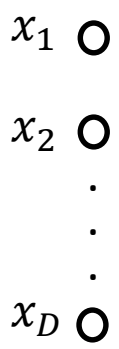
$$y(\boldsymbol{x}) = f(\boldsymbol{w}^t \boldsymbol{\varphi}(\boldsymbol{x}))$$



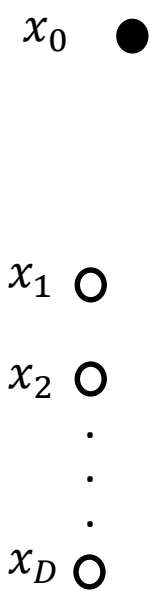


Wiki

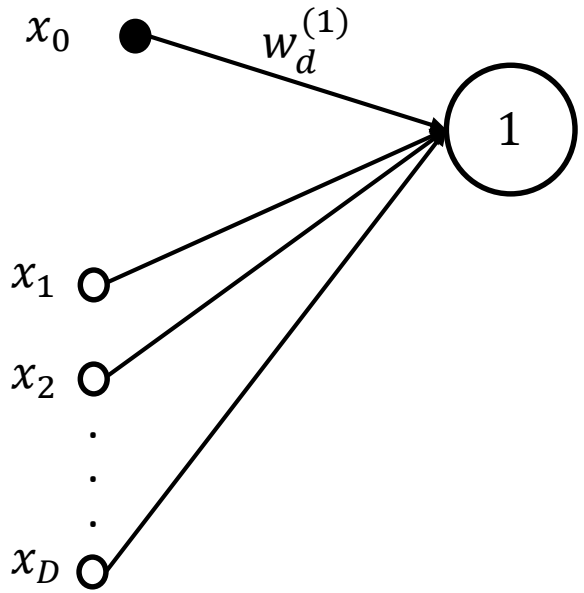
From a single perceptron to a MLP



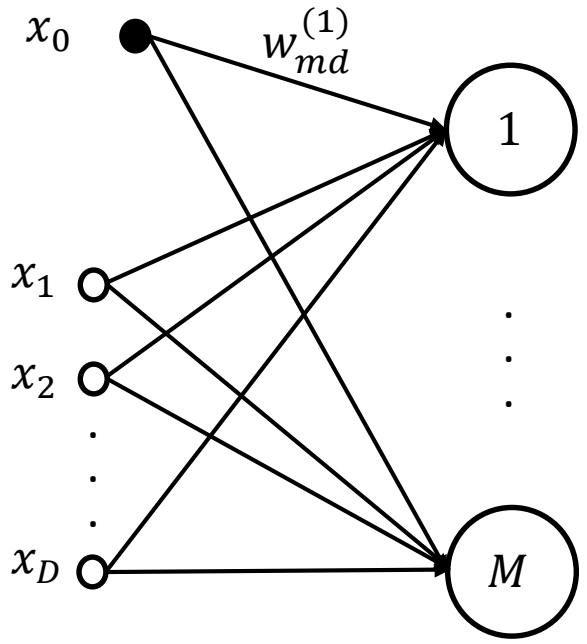
From a single perceptron to a MLP



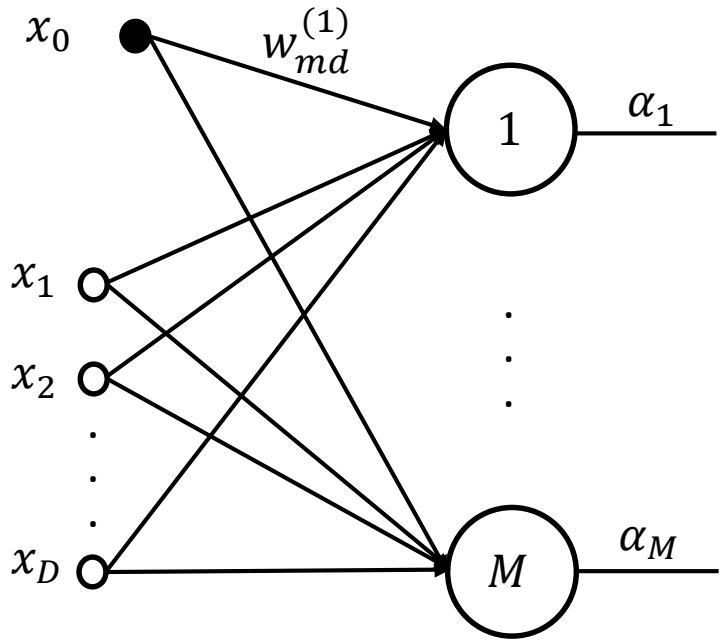
From a single perceptron to a MLP



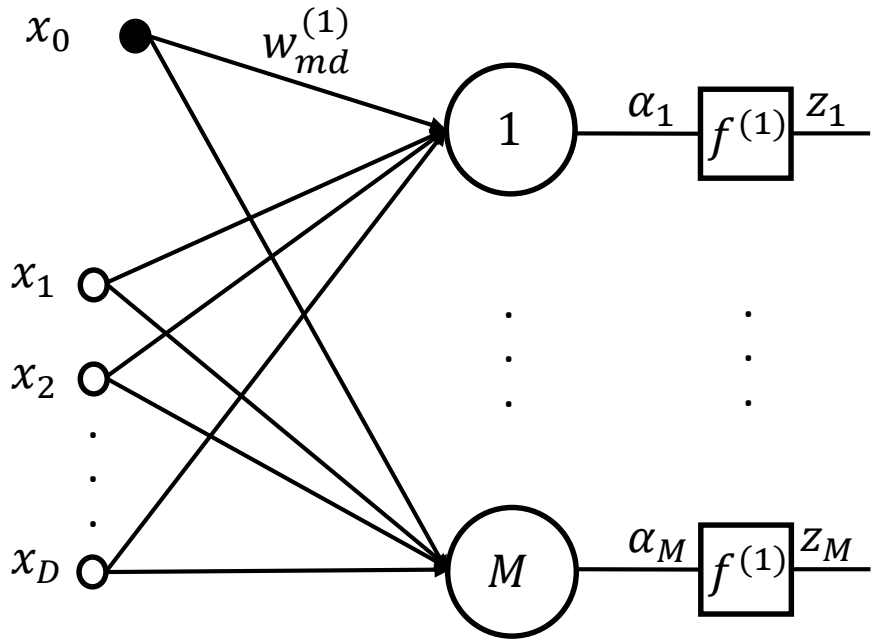
From a single perceptron to a MLP



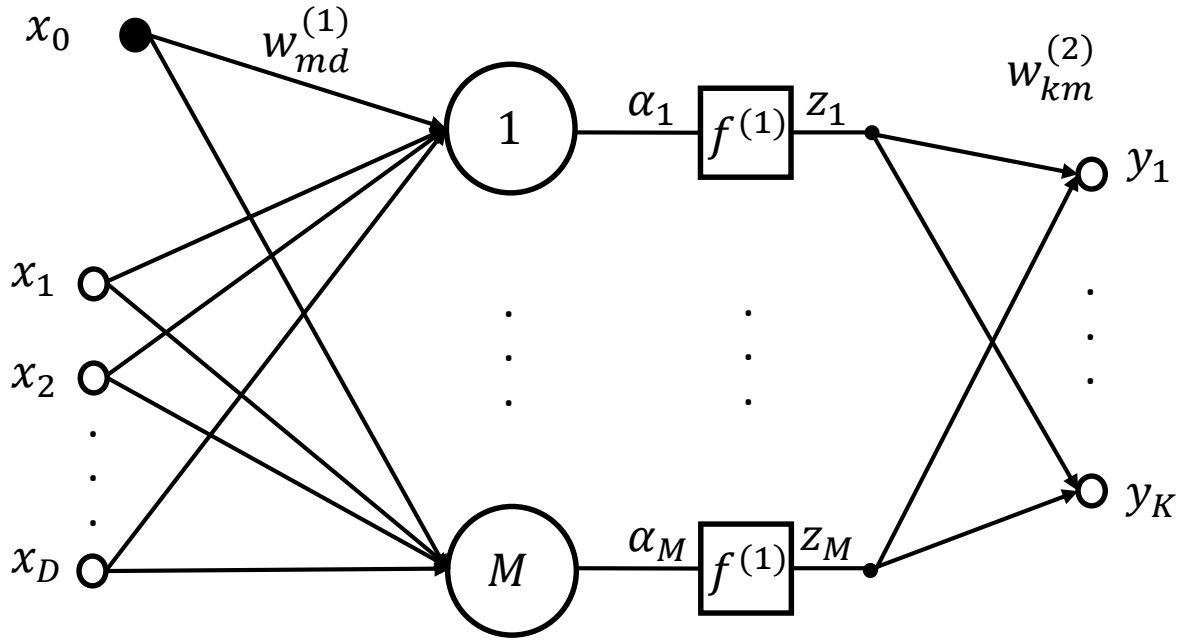
From a single perceptron to a MLP



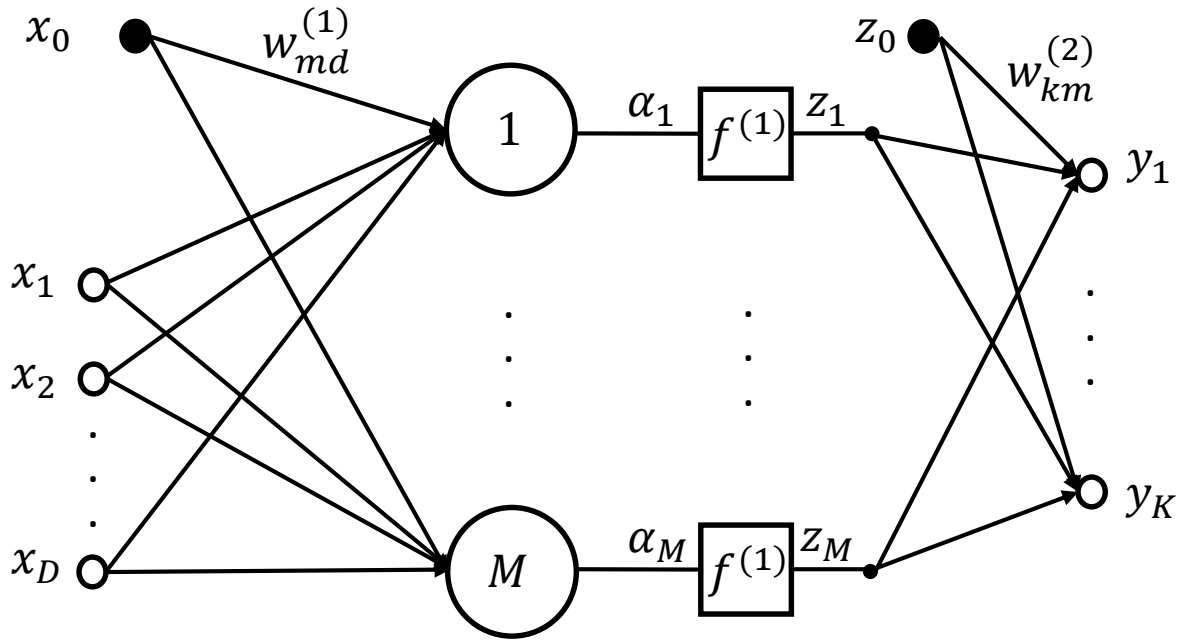
From a single perceptron to a MLP



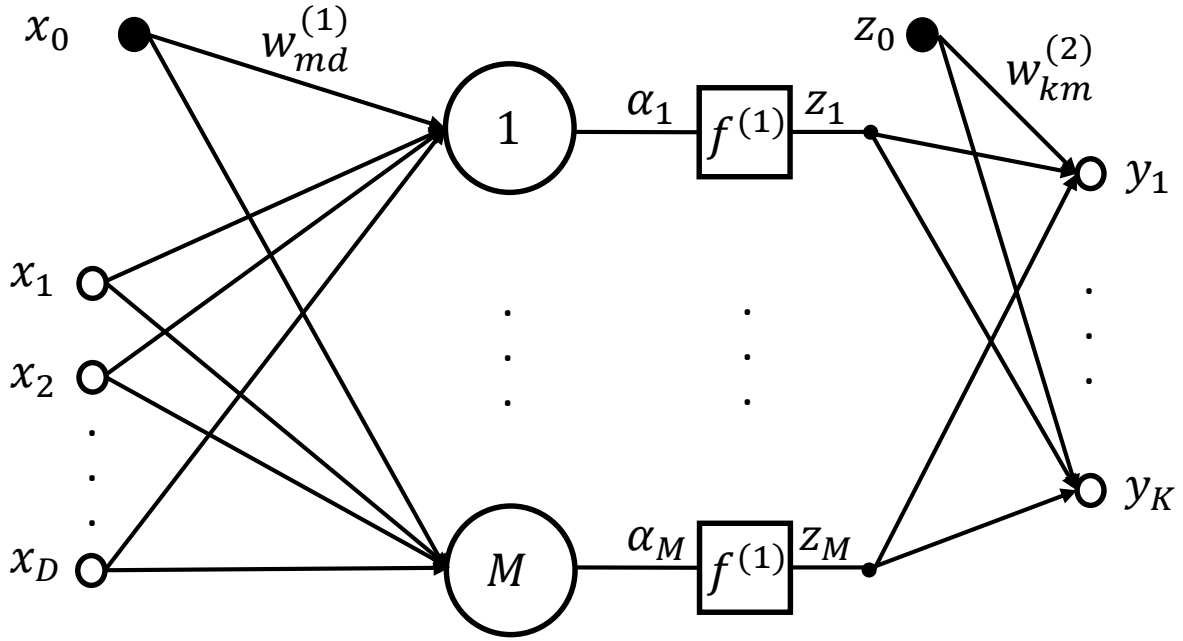
Two layer-Perceptron



Two layer-Perceptron

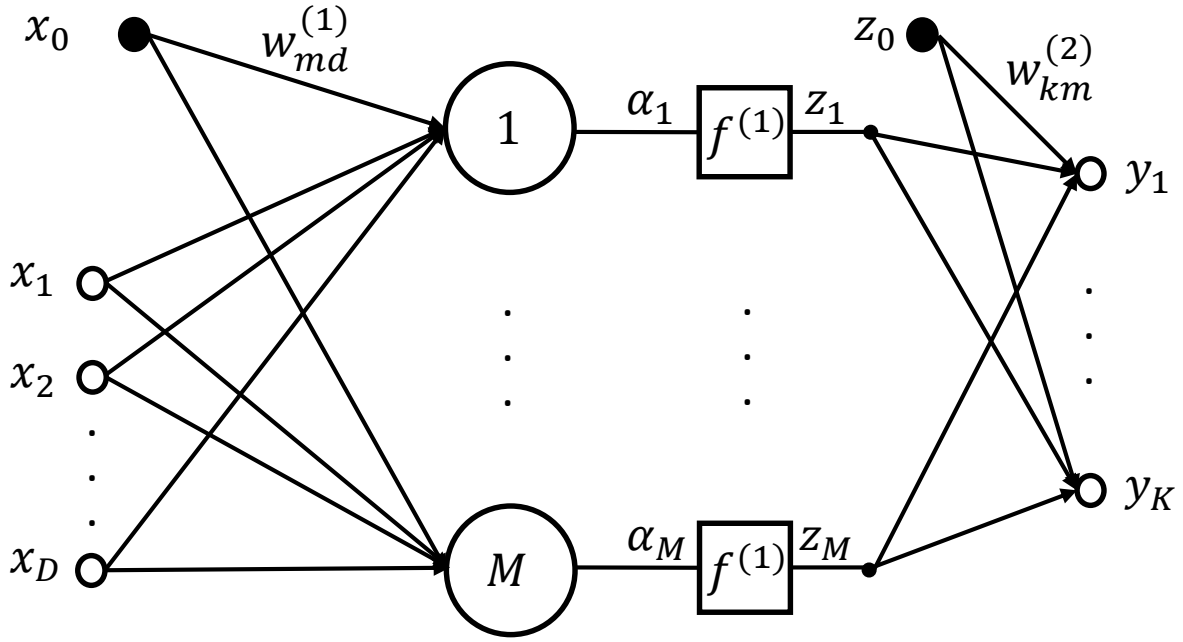


Two layer-Perceptron



Two layer-Perceptron

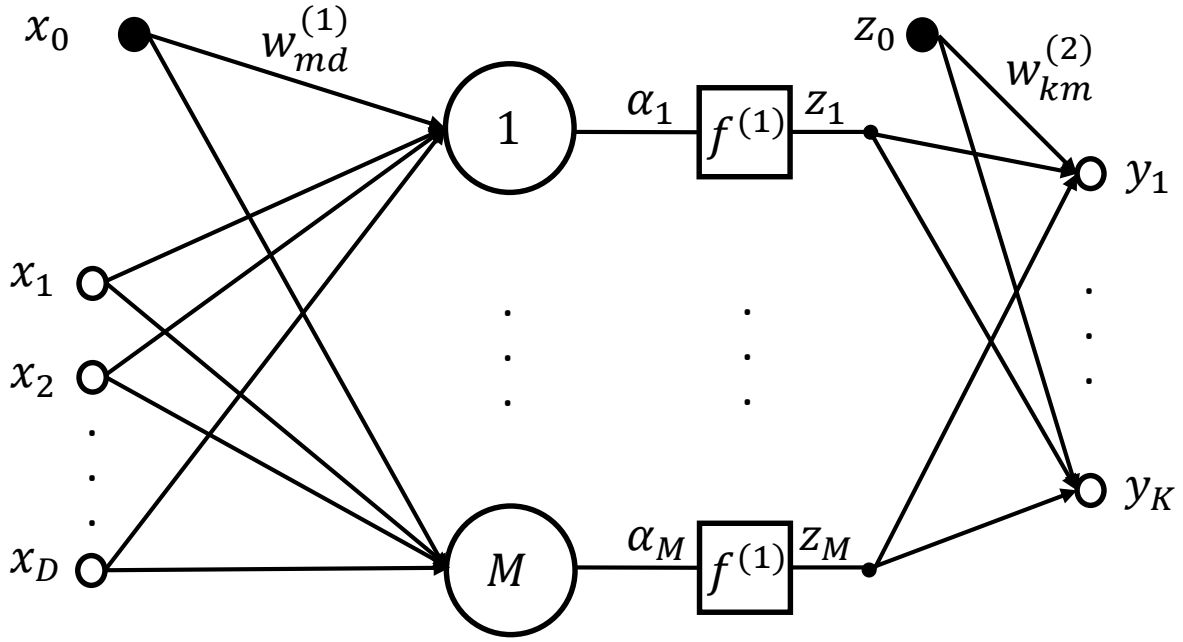
$$y_k = \sum_{m=0}^{M} w_{km}^{(2)} z_m$$



Two layer-Perceptron

$$y_k = \sum_{m=0}^{M} w_{km}^{(2)} z_m$$

$$y_k = \sum_{m=0}^{M} w_{km}^{(2)} f^{(1)}(\alpha_m)$$

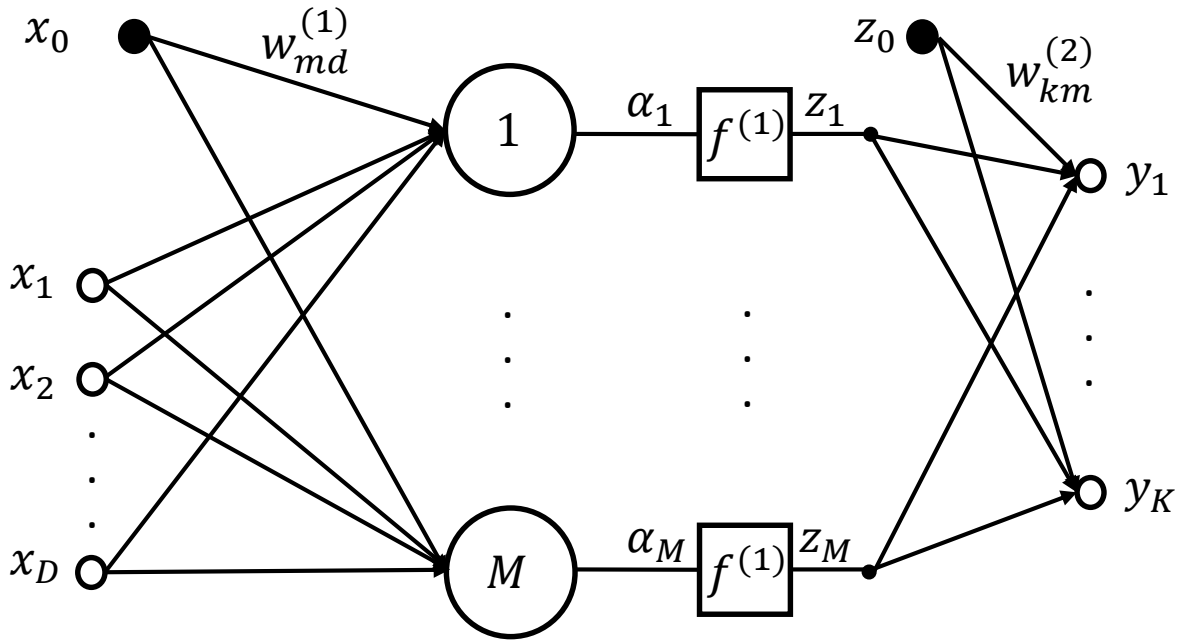


Two layer-Perceptron

$$y_k = \sum_{m=0}^{M} w_{km}^{(2)} z_m$$

$$y_k = \sum_{m=0}^{M} w_{km}^{(2)} f^{(1)}(\alpha_m)$$

$$y_k = \sum_{m=0}^{M} w_{km}^{(2)} f^{(1)}\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right)$$

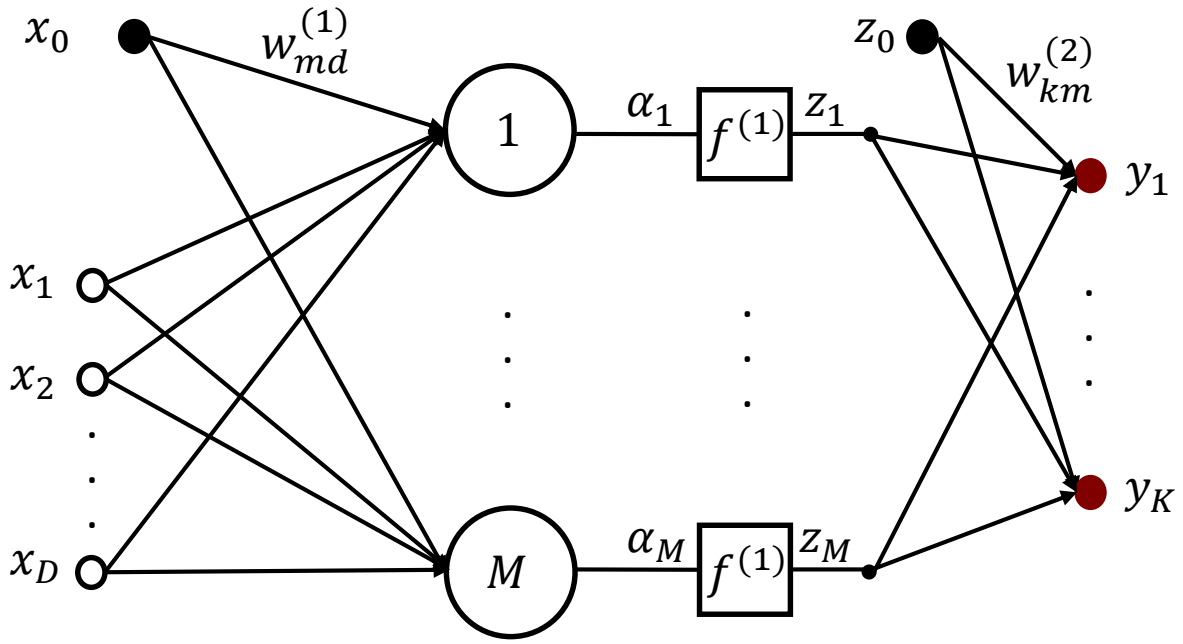


Two layer-Perceptron

$$y_k = \sum_{m=0}^{M} w_{km}^{(2)} z_m$$

$$y_k = \sum_{m=0}^{M} w_{km}^{(2)} f^{(1)}(\alpha_m)$$

$$y_k = \sum_{m=0}^{M} w_{km}^{(2)} f^{(1)}\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right)$$



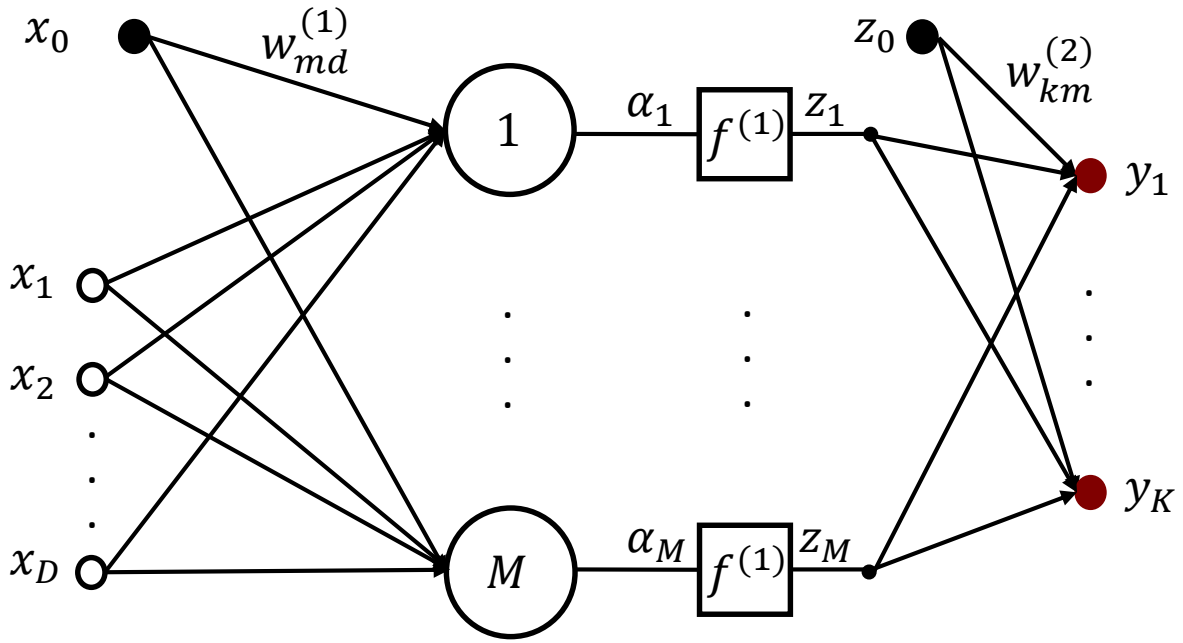
Usually: $y_k = f^{(2)}\left(\sum_{m=0}^{M} w_{km}^{(2)} f^{(1)}\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right)\right)$

Two layer-Perceptron

$$y_k = \sum_{m=0}^{M} w_{km}^{(2)} z_m$$

$$y_k = \sum_{m=0}^{M} w_{km}^{(2)} f^{(1)}(\alpha_m)$$

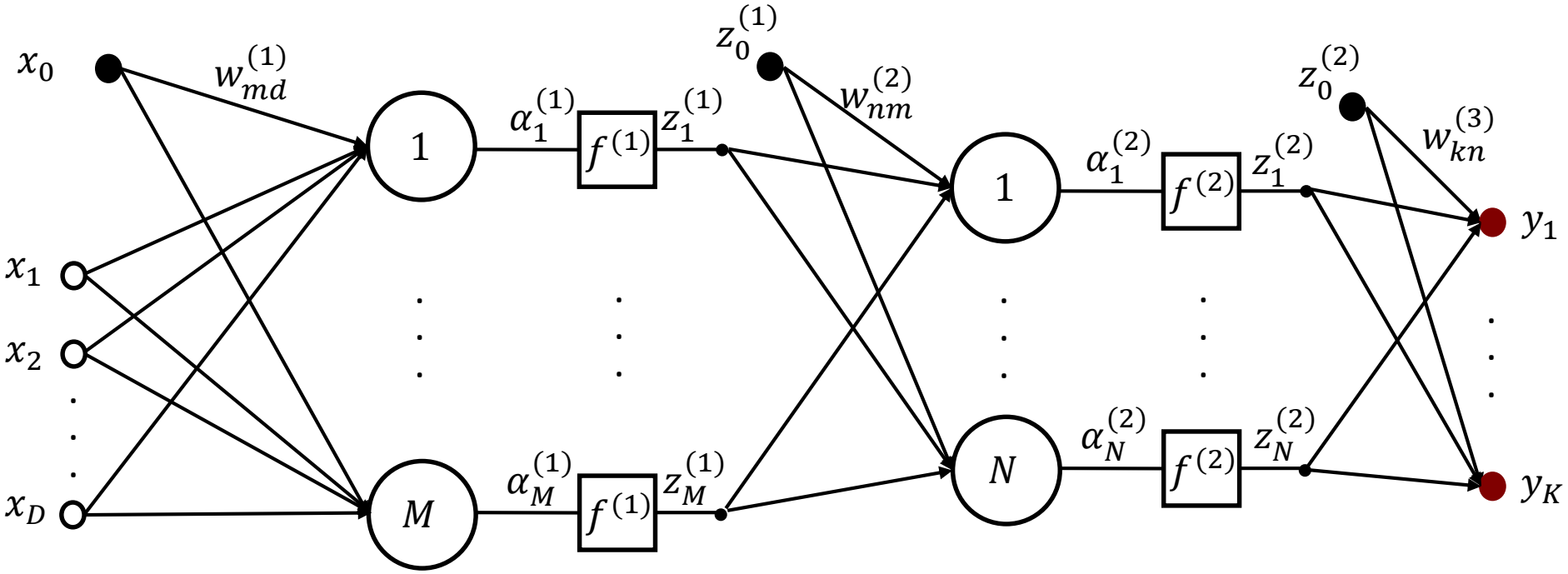
$$y_k = \sum_{m=0}^{M} w_{km}^{(2)} f^{(1)}\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right)$$



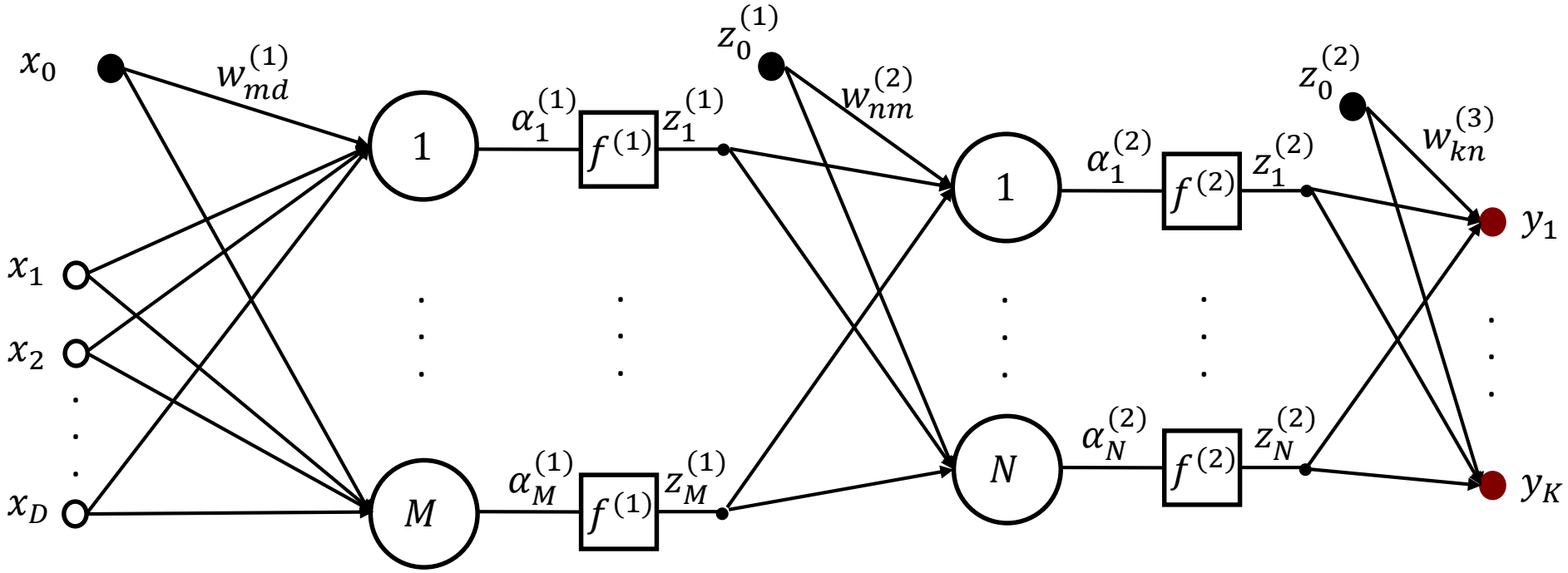
Usually: $y_k = f^{(2)}\left(\sum_{m=0}^{M} w_{km}^{(2)} f^{(1)}\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right)\right) = f^{(2)}\left(\sum_{m=0}^{M} w_{km}^{(2)} \varphi(\boldsymbol{x})\right)$

- Thus we use parametric forms of basis functions in advance and allow them to be *adaptive!*
- Adaptive: learn through training!

Three-layer Perceptron

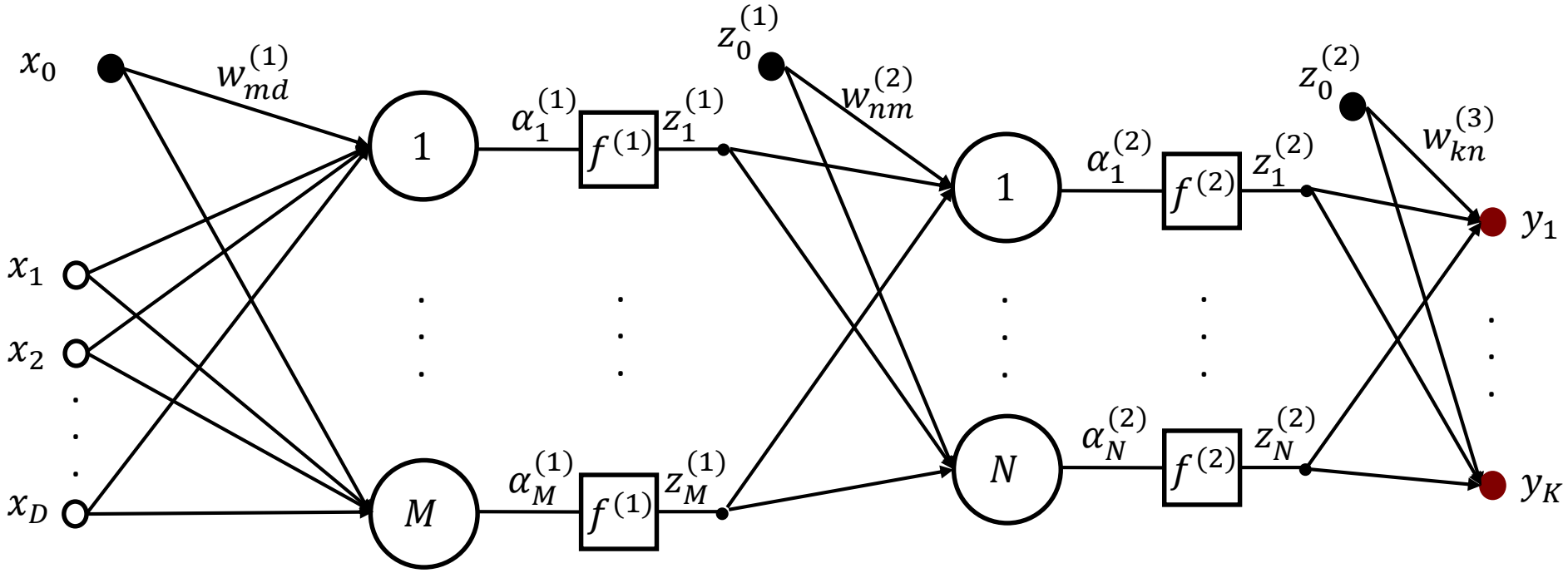


Three-layer Perceptron



$$y_k = f^{(3)}\left(\sum_{n=0}^{N} w_{kn}^{(3)} f^{(2)}\left(\sum_{m=0}^{M} w_{nm}^{(2)} f^{(1)}\left(\sum_{d=0}^{D} w_{md} x_d\right)\right)\right)$$

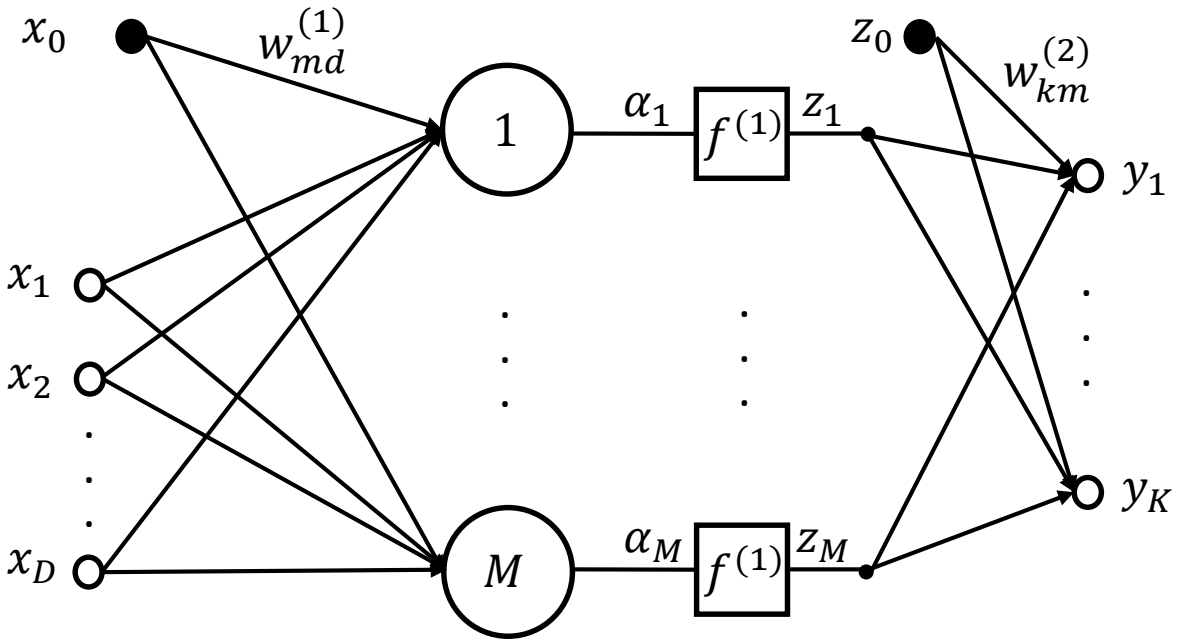
Three-layer Perceptron



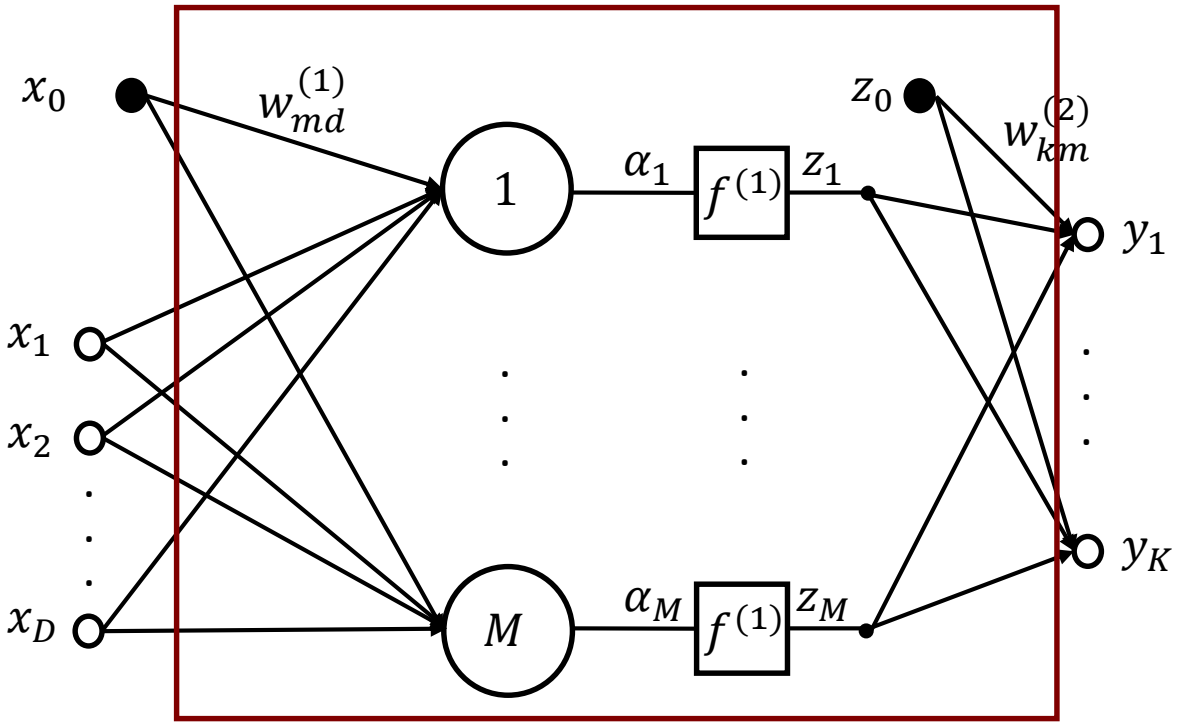
$$y_k = f^{(3)}\left(\sum_{n=0}^{N} w_{kn}^{(3)} f^{(2)}\left(\sum_{m=0}^{M} w_{nm}^{(2)} f^{(1)}\left(\sum_{d=0}^{D} w_{md} x_d\right)\right)\right) = f^{(3)}\left(\alpha^{(3)}\left(f^{(2)}\left(\alpha^{(2)}\left(f^{(1)}\left(\alpha^{(1)}(\boldsymbol{x})\right)\right)\right)\right)\right)$$

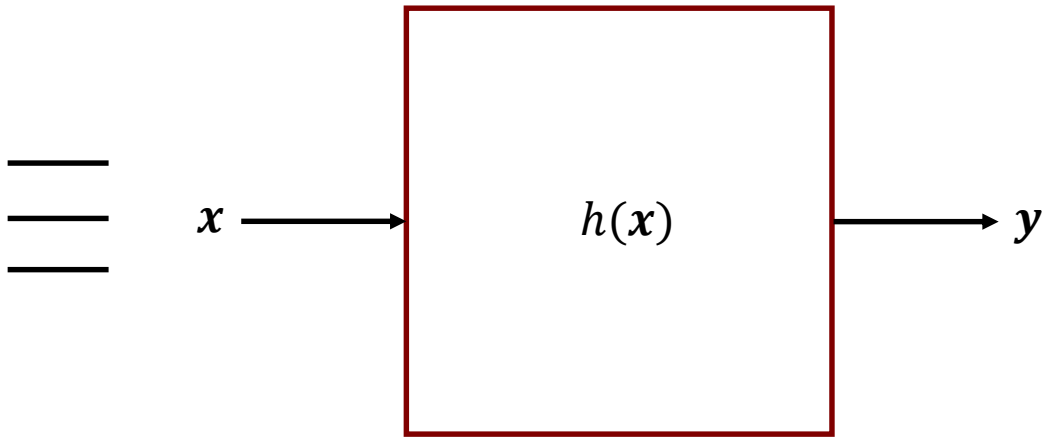
- Very complicated basis functions as functions of $\boldsymbol{x}$ (sequential mapping from layer to layer)

Function Approximators

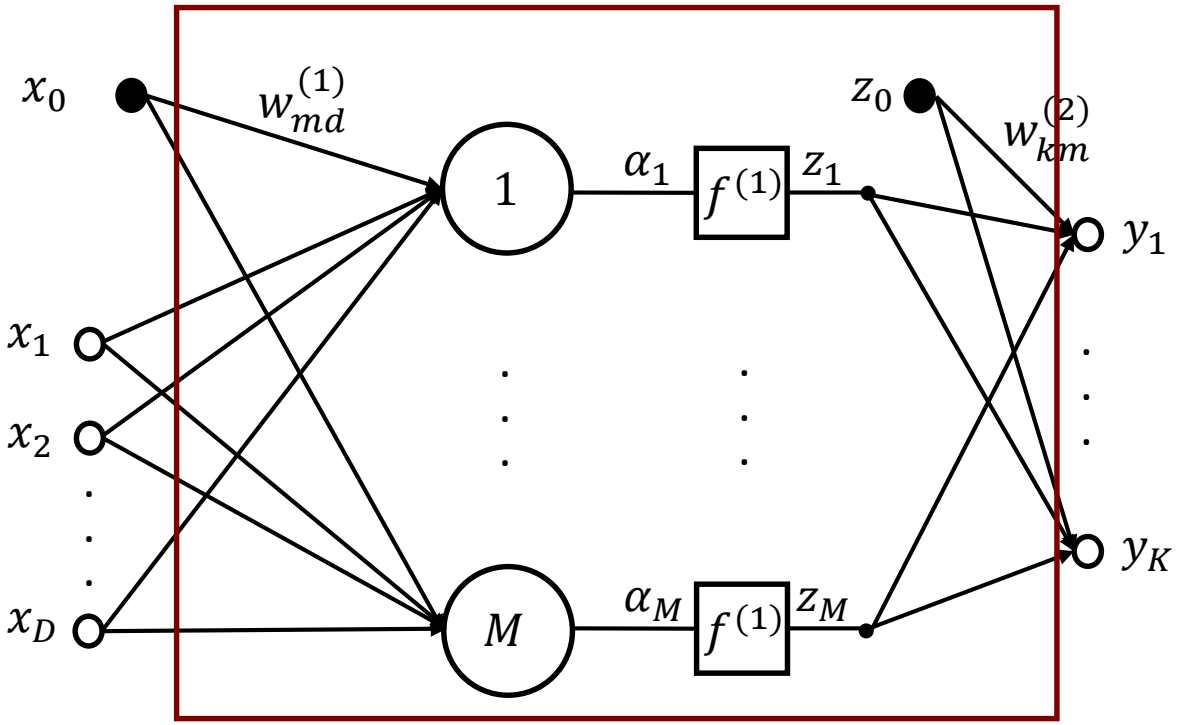


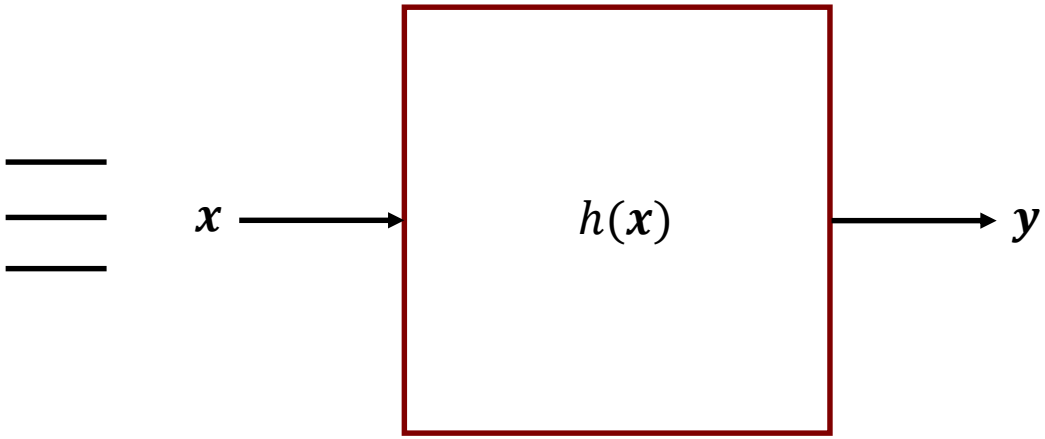
Function Approximators





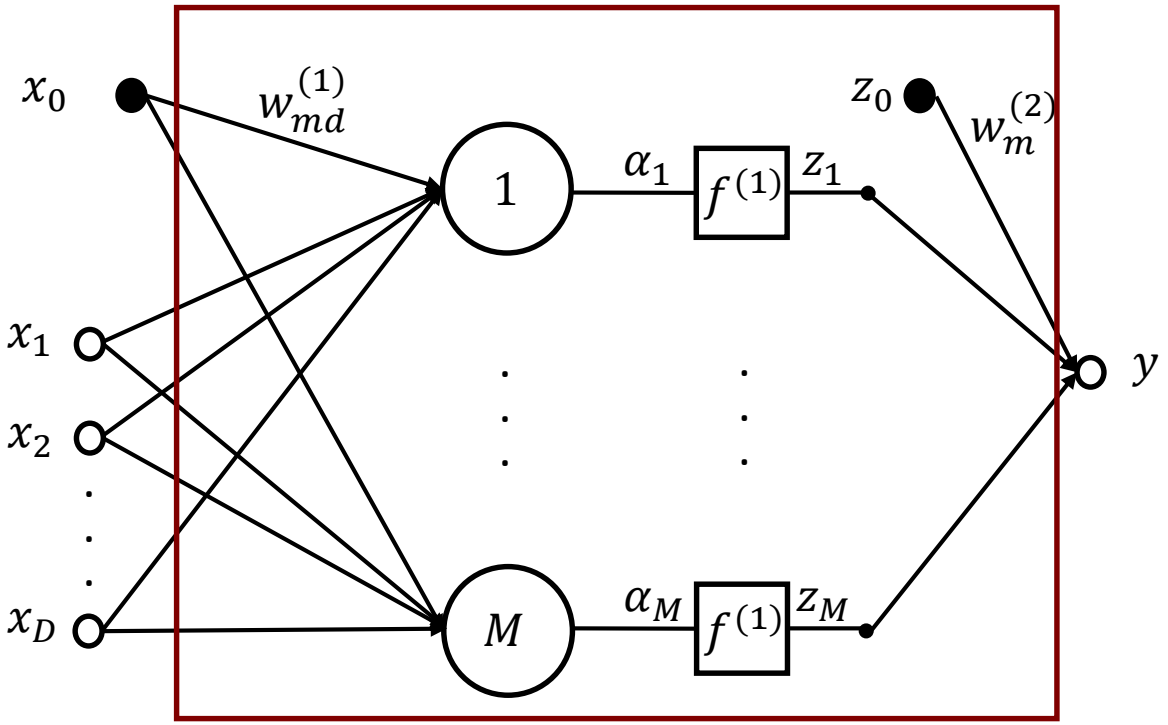
Function Approximators

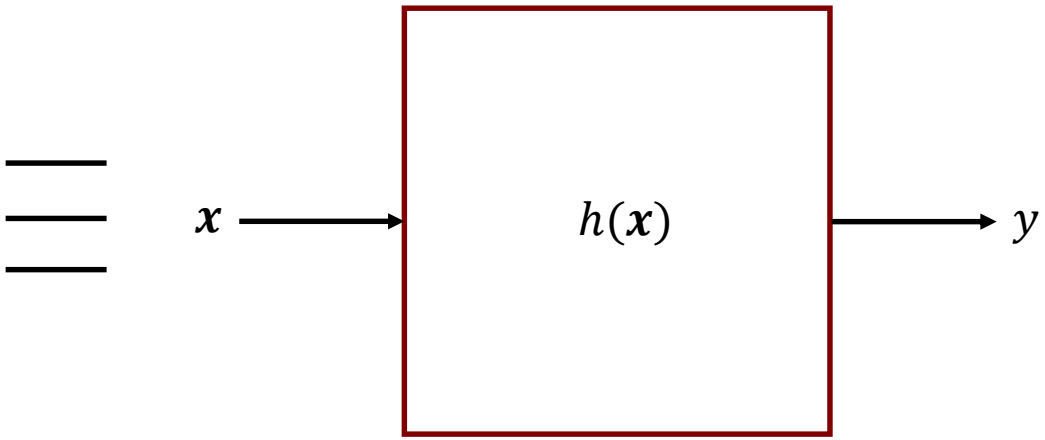




$$\boldsymbol{y}\left(\boldsymbol{x}, \boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}\right) = \sum_{m=0}^{M} w_{km}^{(2)} f^{(1)}\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right) = \sum_{m=0}^{M} w_{km}^{(2)} \varphi_m(\boldsymbol{x})$$

Function Approximators



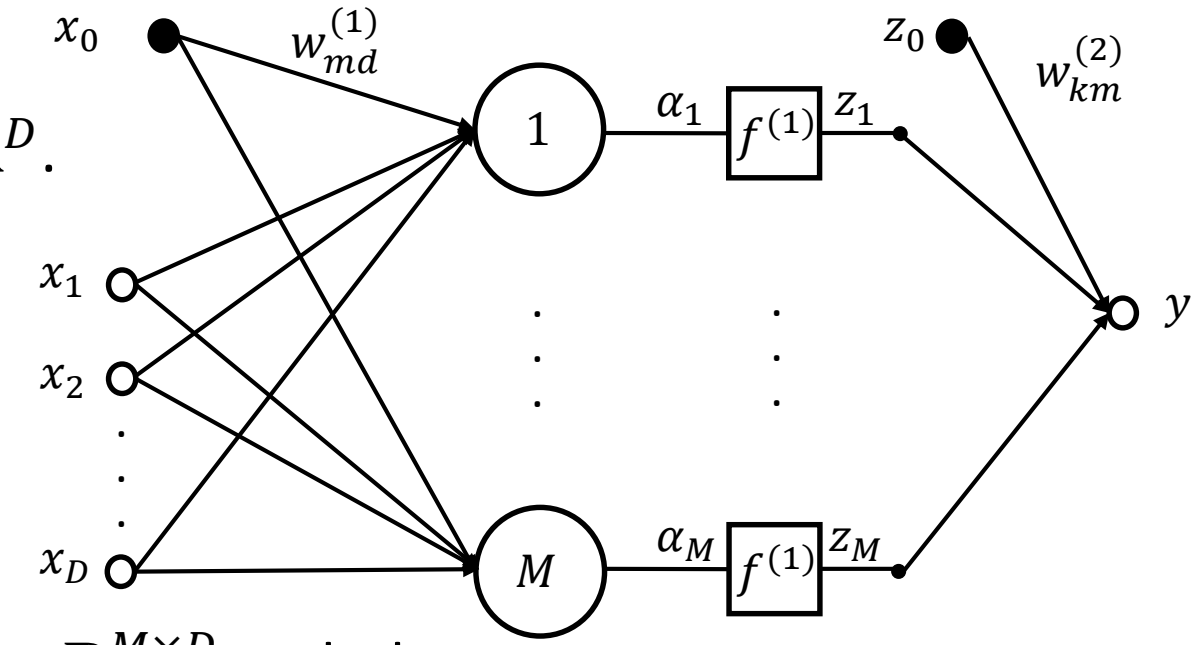


$$y\left(\boldsymbol{x}, \boldsymbol{W}^{(1)}, \boldsymbol{w}^{(2)}\right) = \sum_{m=0}^{M} w_m^{(2)} f^{(1)} \left(\sum_{d=0}^{D} w_{md}^{(2)} x_d \right) = \sum_{m=0}^{M} w_m^{(2)} \varphi_m(\boldsymbol{x})$$

Universal approximators

Theorem:

- Let $h(.)$ be a continuous function on $\mathbb{R}^D$.
- Let $f(.)$ be a fixed analytic function which is **not** polynomial.
- Given any small number $\varepsilon > 0$,we can find a number M, $\boldsymbol{w}^{(2)} \in \mathbb{R}^M$ and $\boldsymbol{W}^{(1)} \in \mathbb{R}^{M\times D}$ such that:



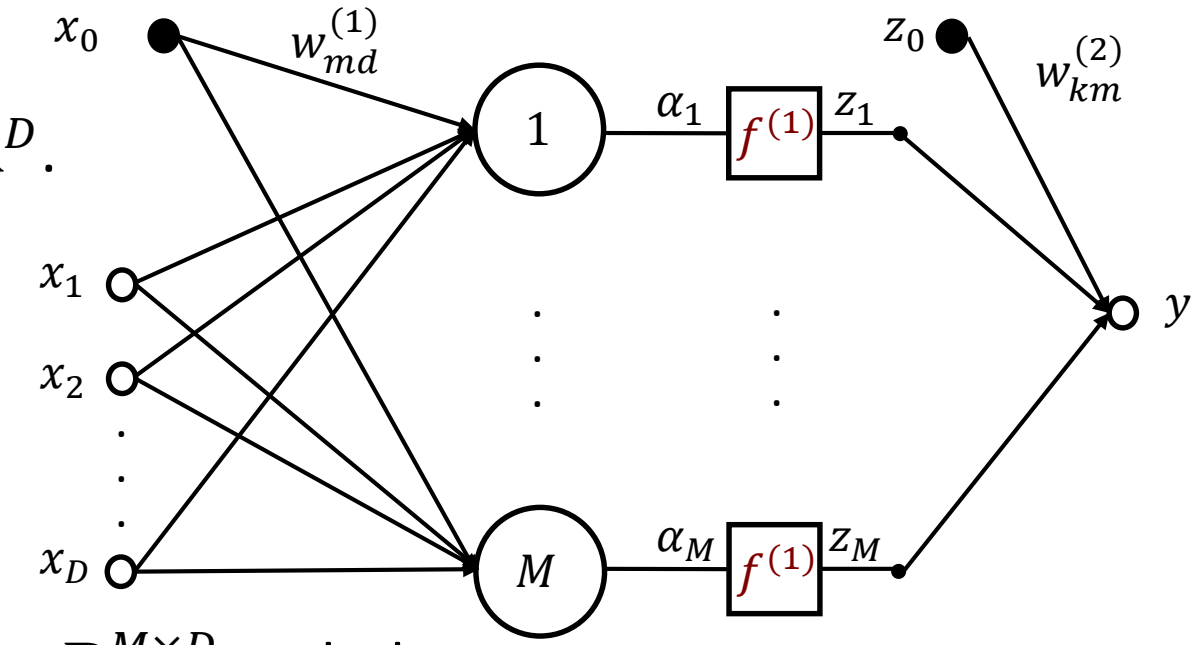
$$\left|h(x) - y\left(\boldsymbol{x}, \boldsymbol{W}^{(1)}, \boldsymbol{w}^{(2)}\right)\right| < \varepsilon$$

Note: For smaller ε we usually need larger M.

Universal approximators

Theorem:

- Let $h(.)$ be a continuous function on $\mathbb{R}^D$.
- Let $f(.)$ be a fixed analytic function which is **not** polynomial.
- Given any small number $\varepsilon > 0$,we can find a number M, $\boldsymbol{w}^{(2)} \in \mathbb{R}^M$ and $\boldsymbol{W}^{(1)} \in \mathbb{R}^{M \times D}$ such that:

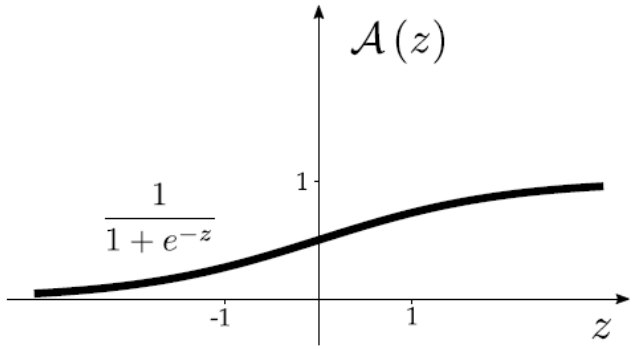


$$\left|h(x) - y\left(\boldsymbol{x}, \boldsymbol{W}^{(1)}, \boldsymbol{w}^{(2)}\right)\right| < \varepsilon$$

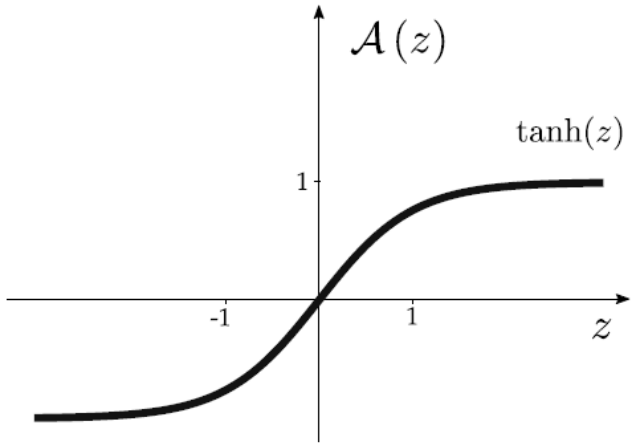
Note: For smaller ε we usually need larger M.

Activation Functions

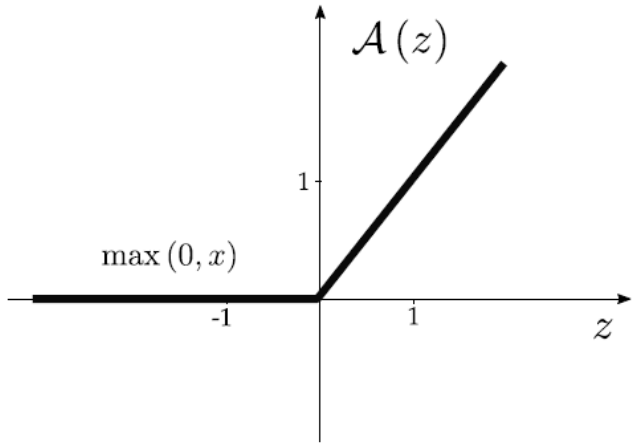
Logistic sigmoid



Tanh

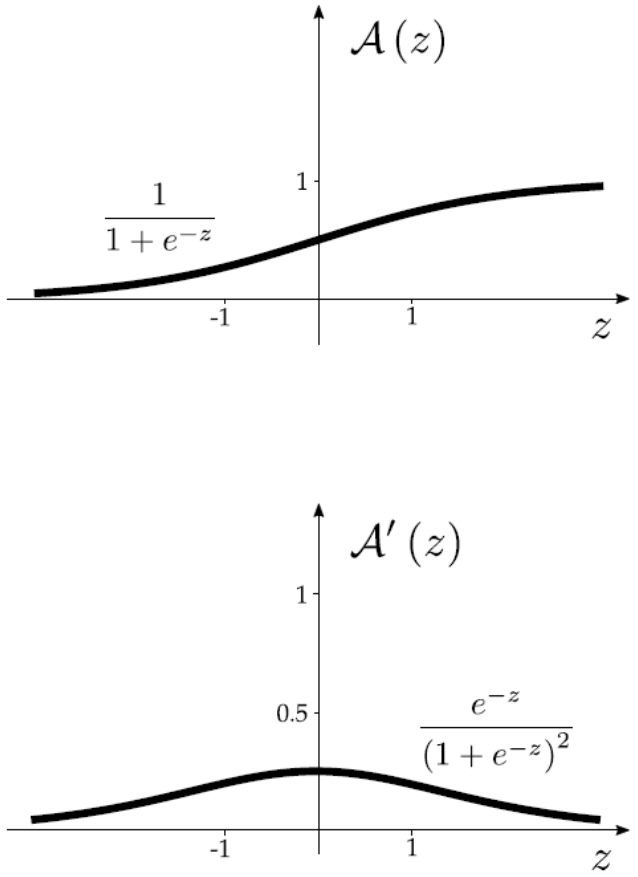


Rectified Linear Unit (RELU)

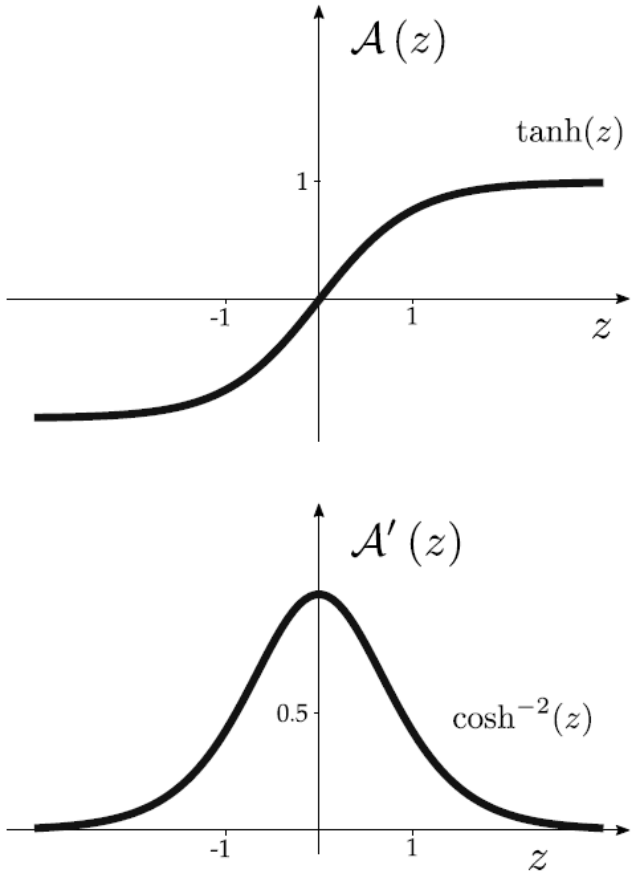


Activation Functions

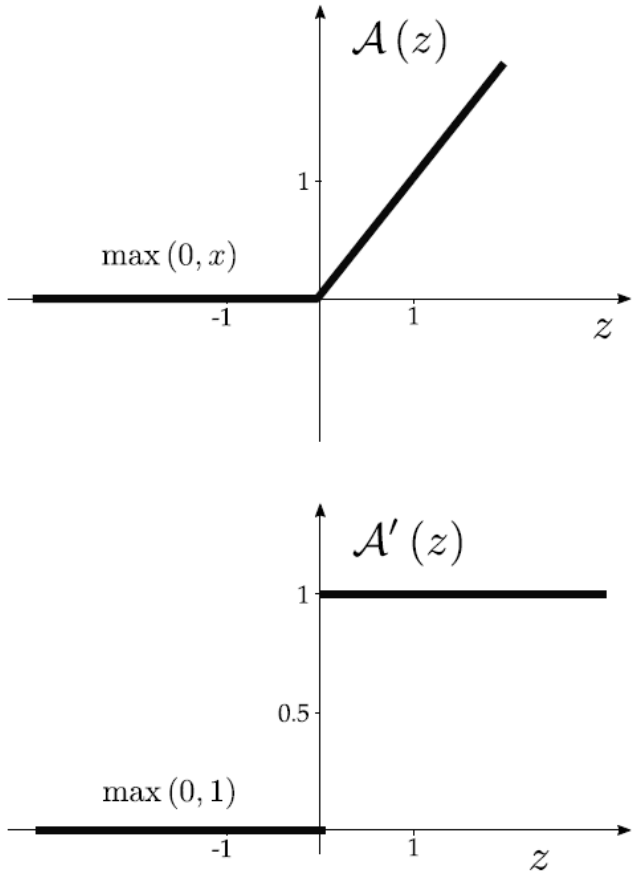
Logistic sigmoid



Tanh

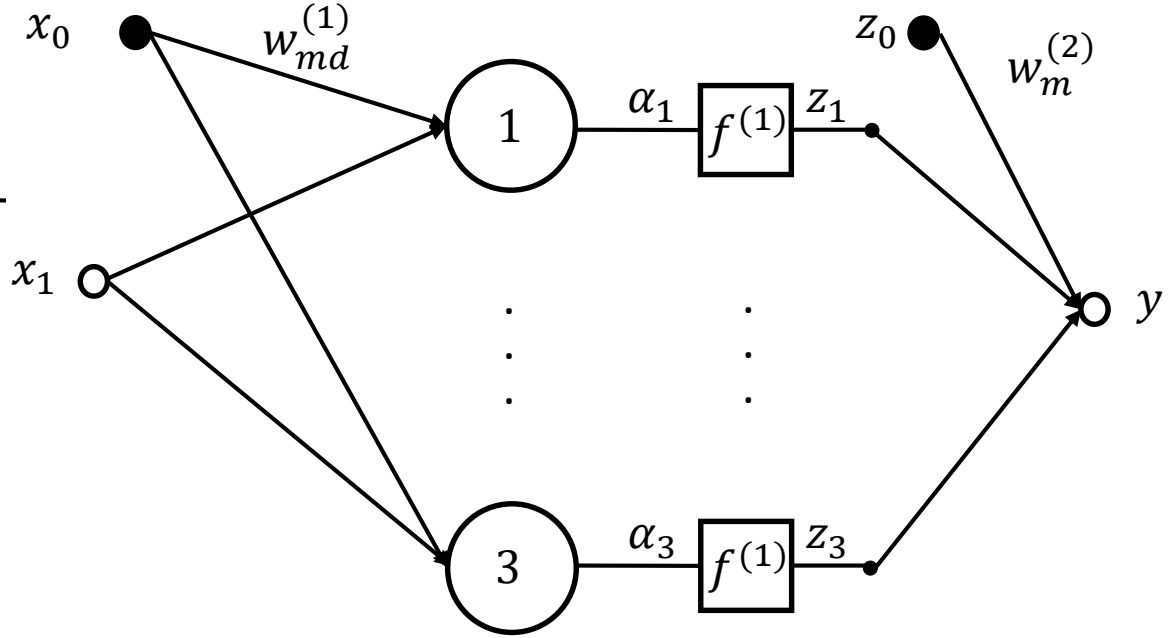


Rectified Linear Unit (RELU)



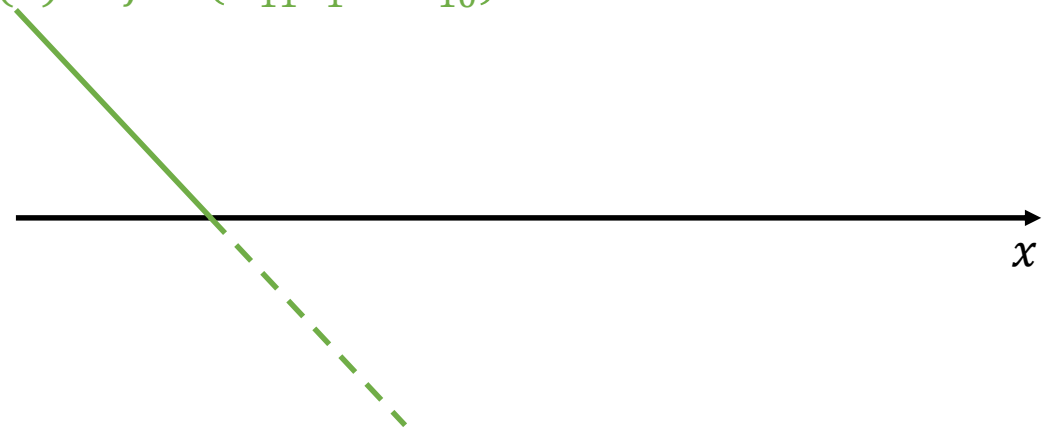
Masi et al. 2020

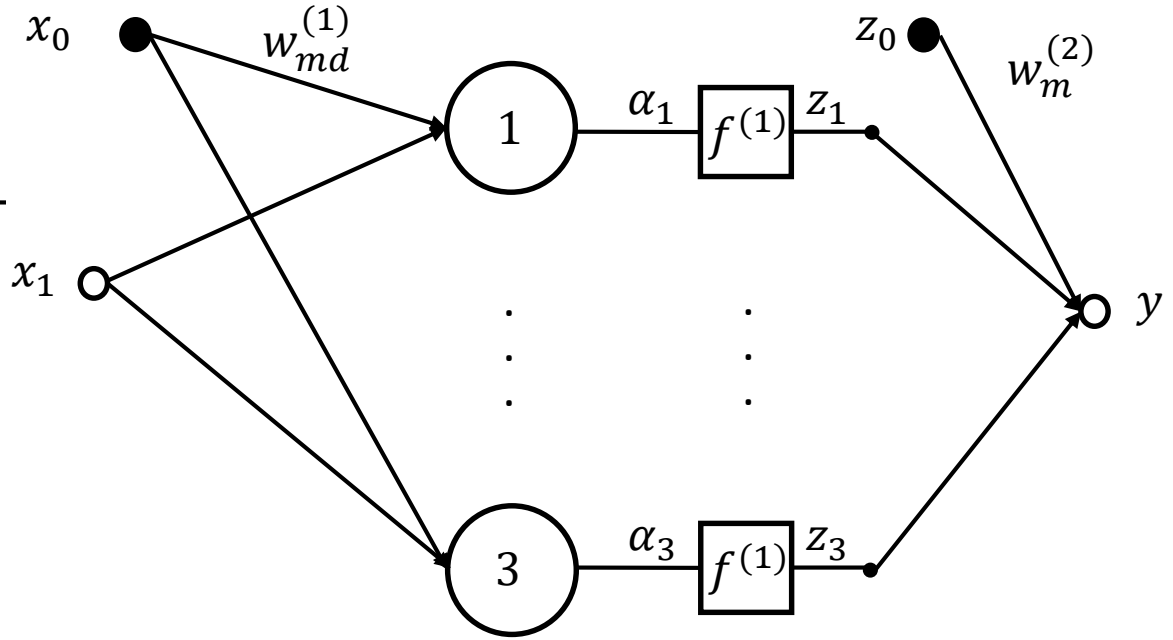
NN with ReLU



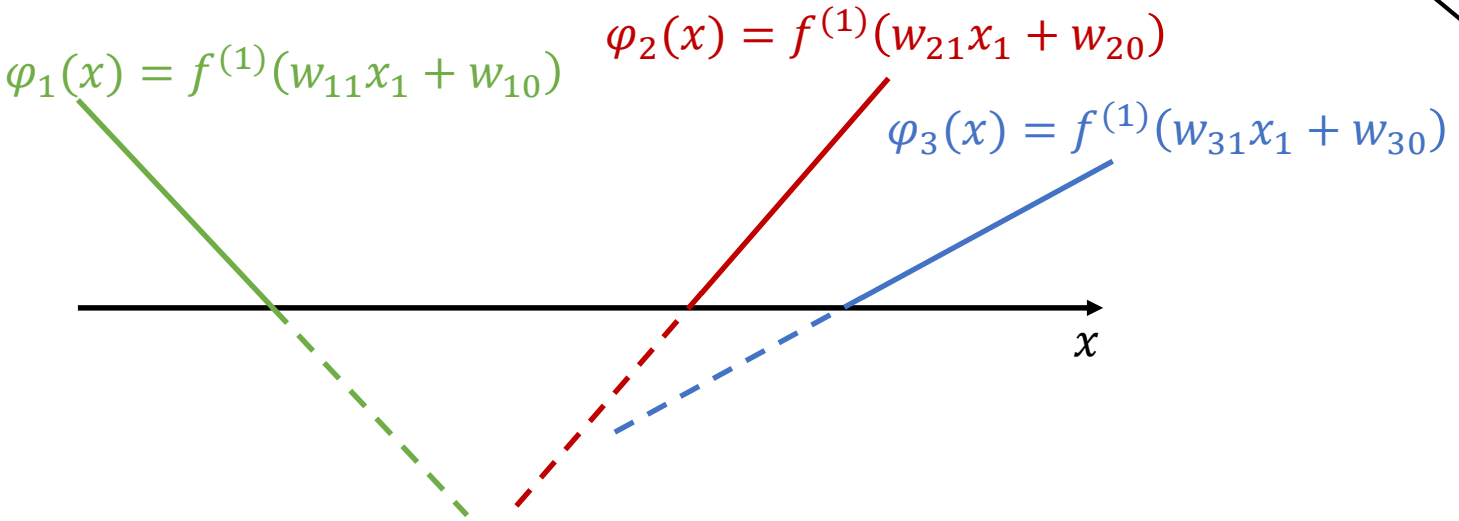
NN with ReLU

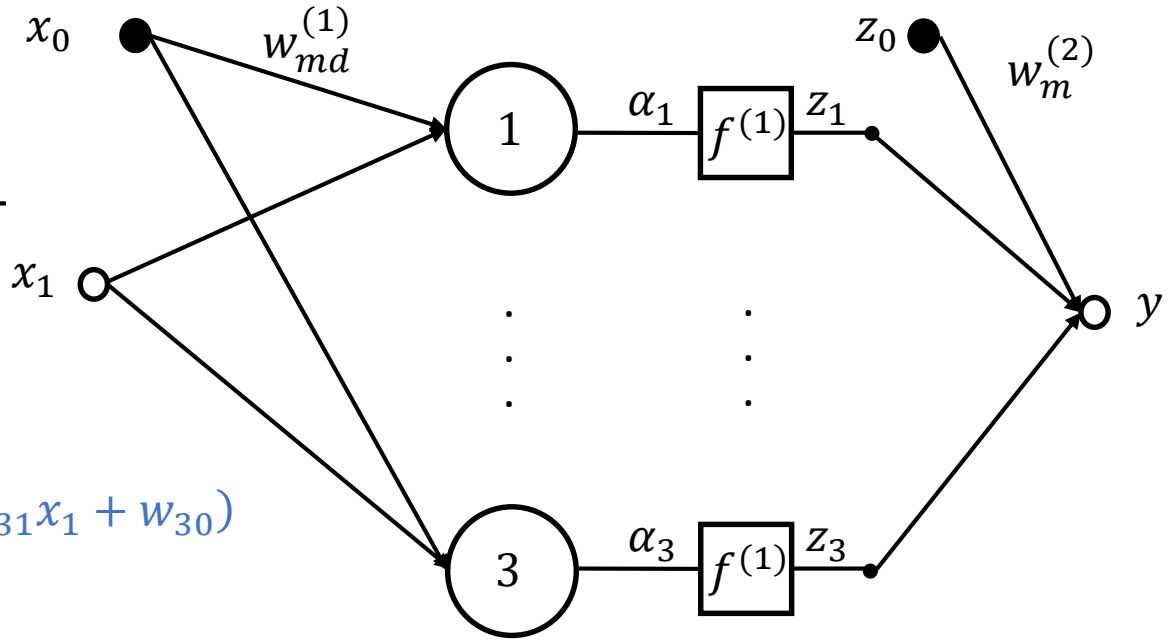
$z_1 = \varphi_1(x) = f^{(1)}(w_{11}x_1 + w_{10})$



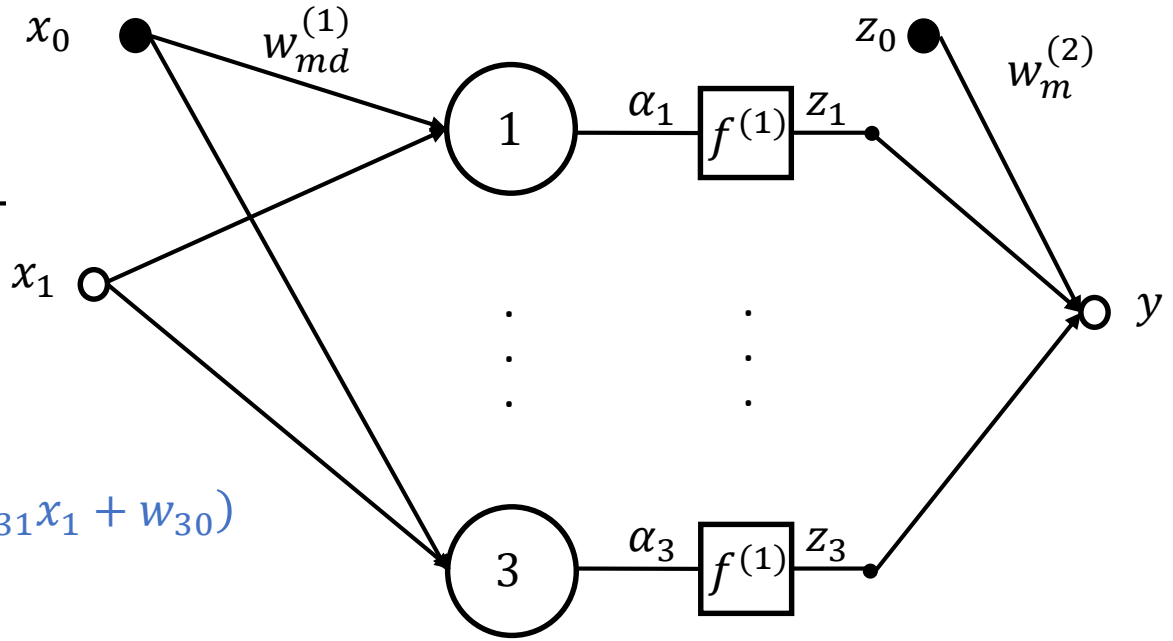


NN with ReLU


$\varphi_1(x) = f^{(1)}(w_{11}x_1 + w_{10})$
$\varphi_2(x) = f^{(1)}(w_{21}x_1 + w_{20})$
$\varphi_3(x) = f^{(1)}(w_{31}x_1 + w_{30})$
x


x_0
$w_{md}^{(1)}$
x_1
1
3
α_1
α_3
$f^{(1)}$
$f^{(1)}$
z_1
z_3
z_0
$w_m^{(2)}$
y

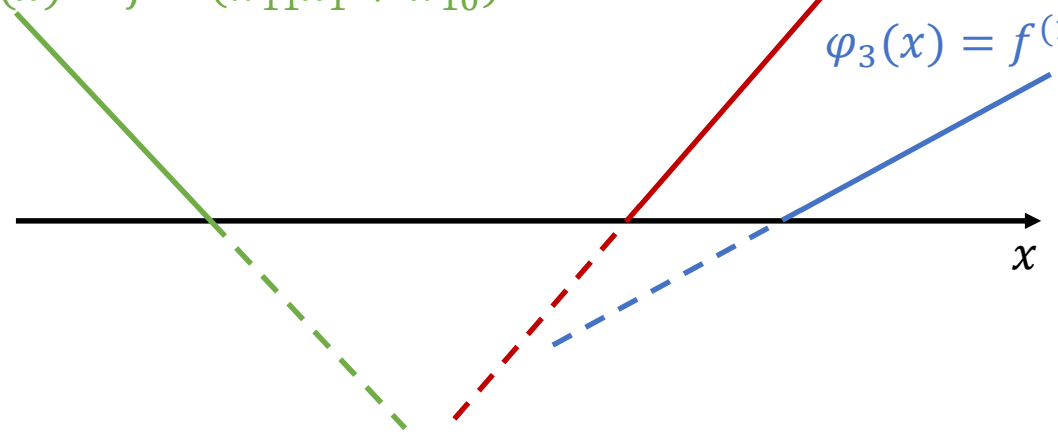
NN with ReLU



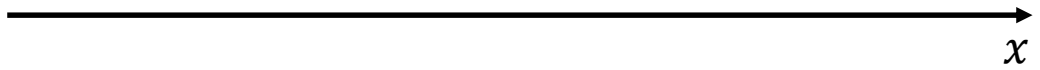
$\varphi_1(x) = f^{(1)}(w_{11}x_1 + w_{10})$

$\varphi_2(x) = f^{(1)}(w_{21}x_1 + w_{20})$

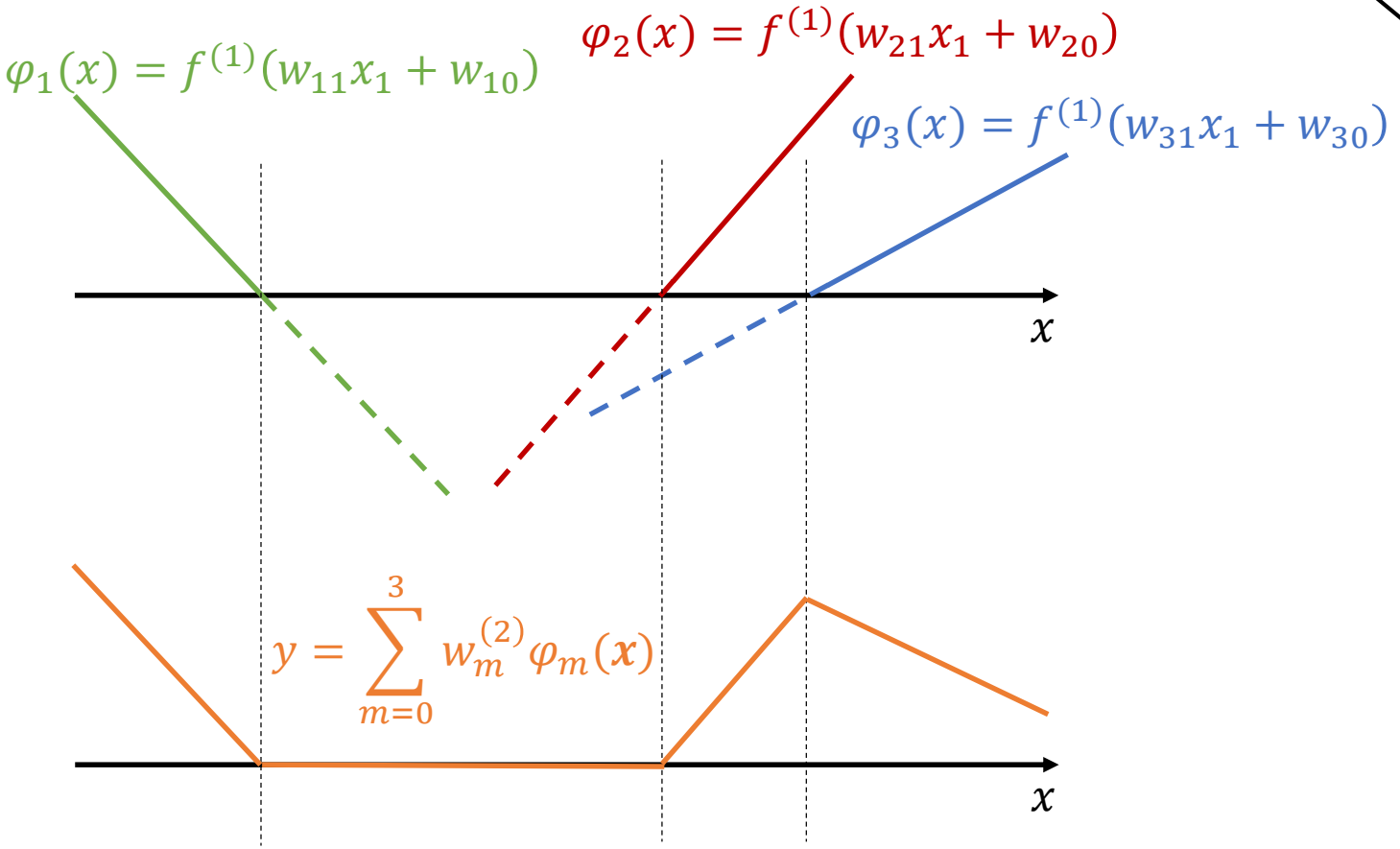
$\varphi_3(x) = f^{(1)}(w_{31}x_1 + w_{30})$

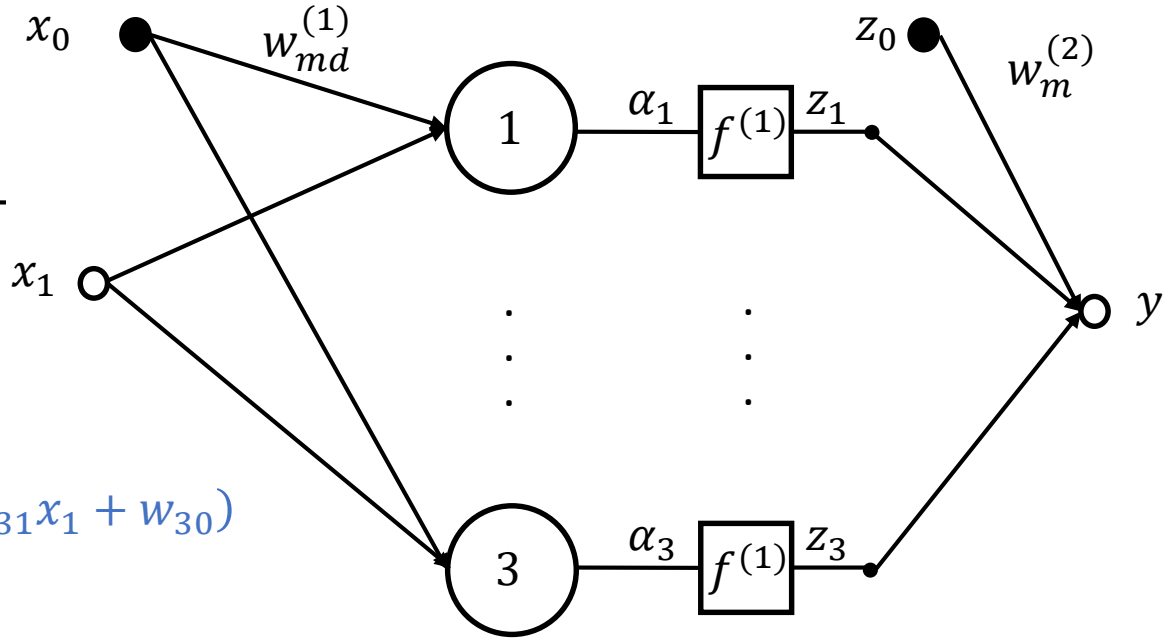


$$y = \sum_{m=0}^{3} w_m^{(2)} \varphi_m(\boldsymbol{x})$$

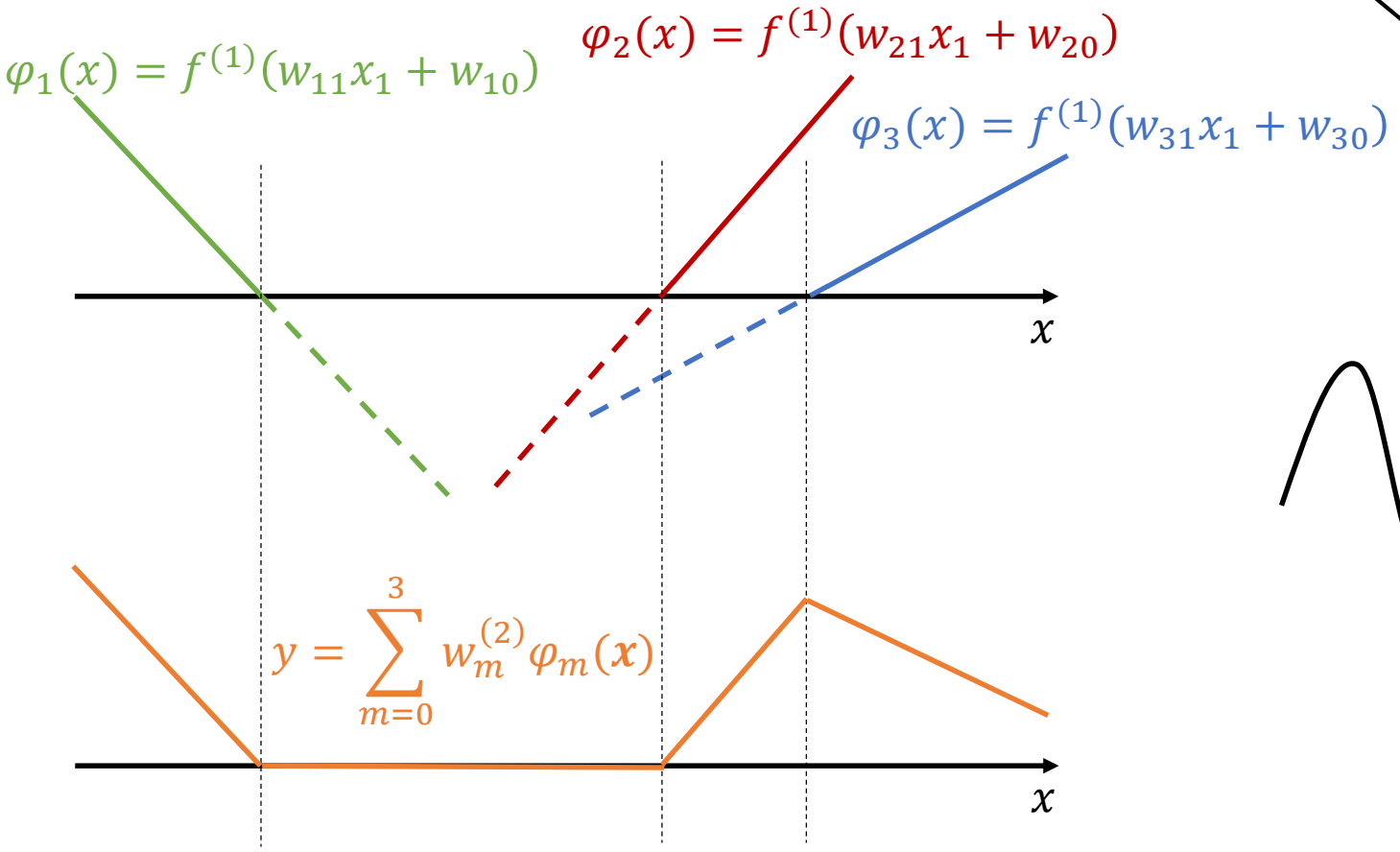


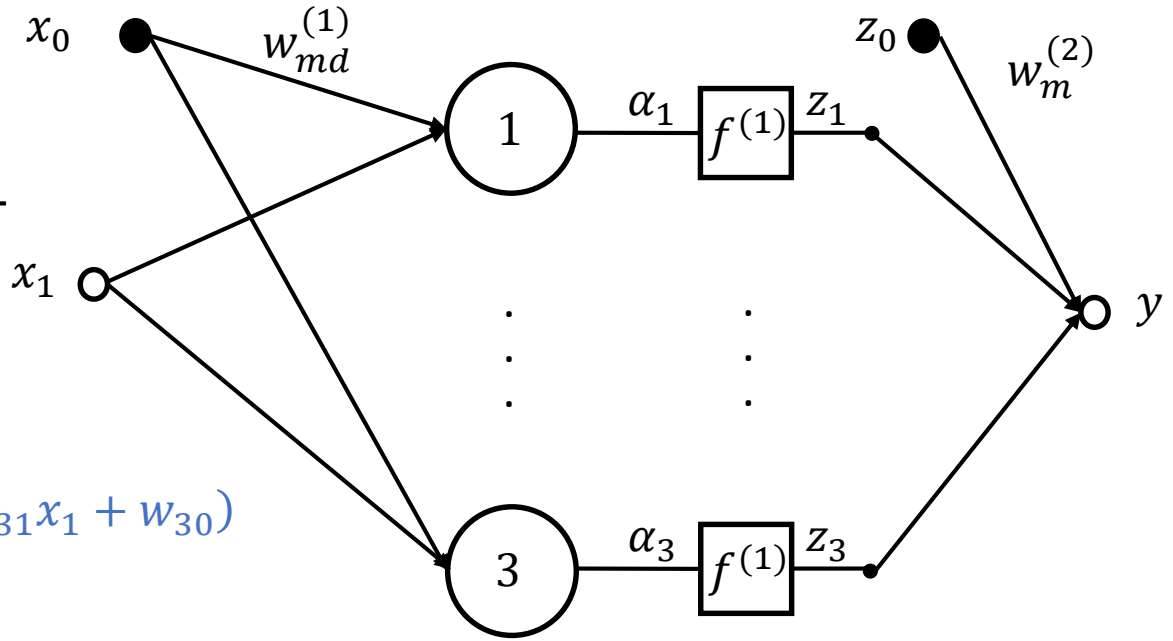
NN with ReLU

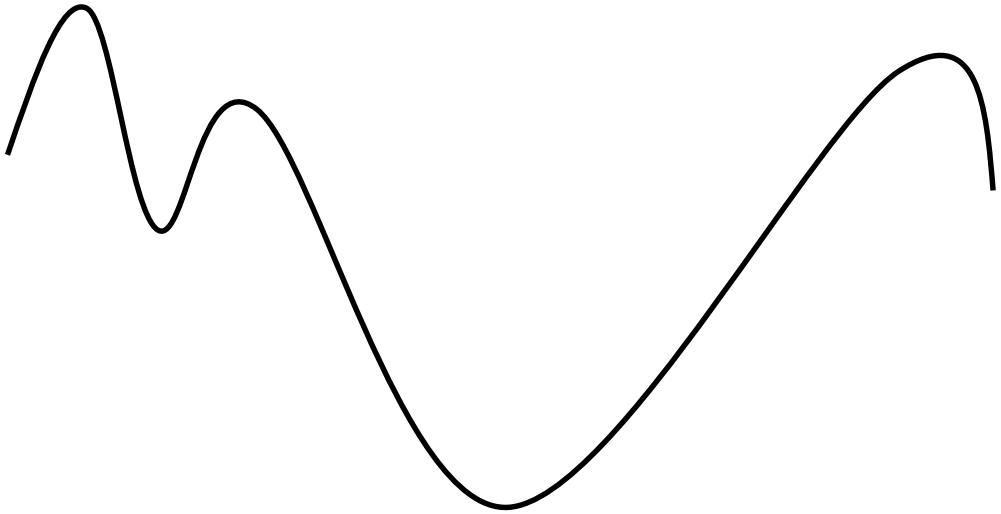

$\varphi_1(x) = f^{(1)}(w_{11}x_1 + w_{10})$
$\varphi_2(x) = f^{(1)}(w_{21}x_1 + w_{20})$
$\varphi_3(x) = f^{(1)}(w_{31}x_1 + w_{30})$
x
$y = \sum_{m=0}^{3} w_m^{(2)} \varphi_m(\boldsymbol{x})$
x


x_0
$w_{md}^{(1)}$
z_0
$w_m^{(2)}$
1
α_1
$f^{(1)}$
z_1
x_1
y
3
α_3
$f^{(1)}$
z_3

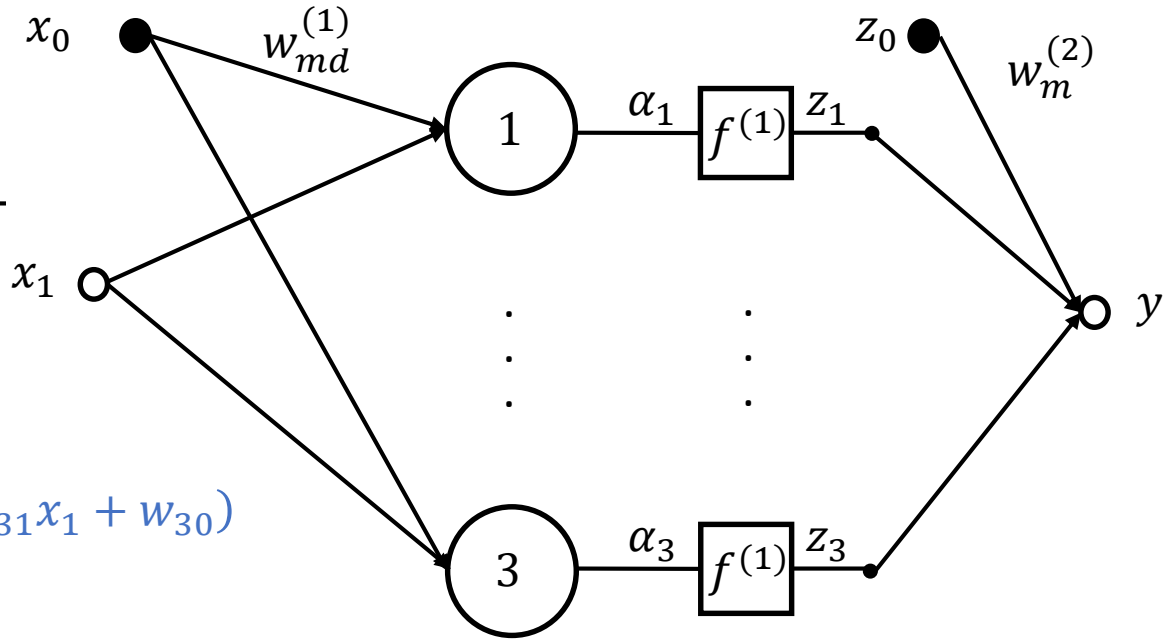
NN with ReLU

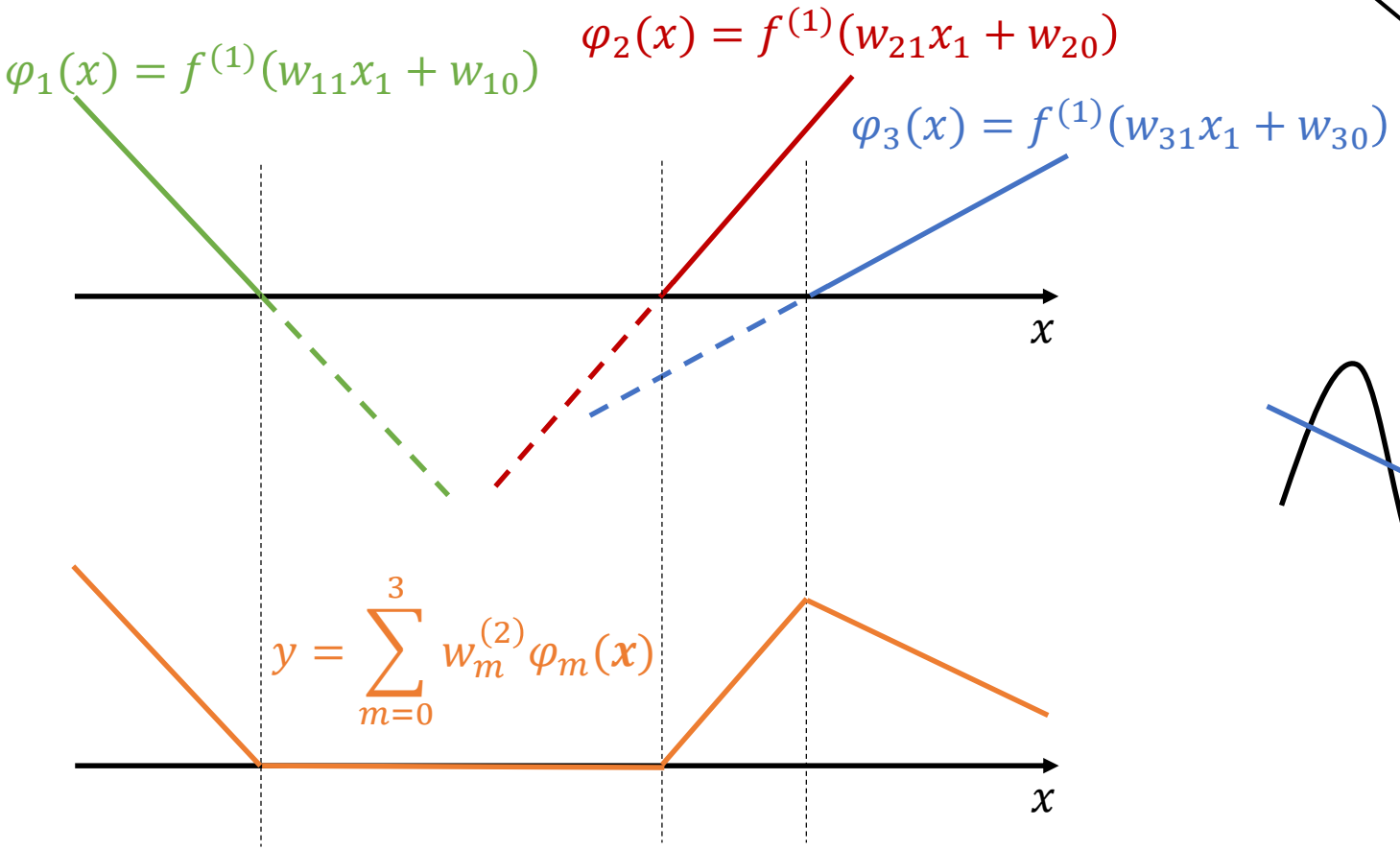

$\varphi_1(x) = f^{(1)}(w_{11}x_1 + w_{10})$
$\varphi_2(x) = f^{(1)}(w_{21}x_1 + w_{20})$
$\varphi_3(x) = f^{(1)}(w_{31}x_1 + w_{30})$
x
$y = \sum_{m=0}^{3} w_m^{(2)} \varphi_m(\boldsymbol{x})$
x

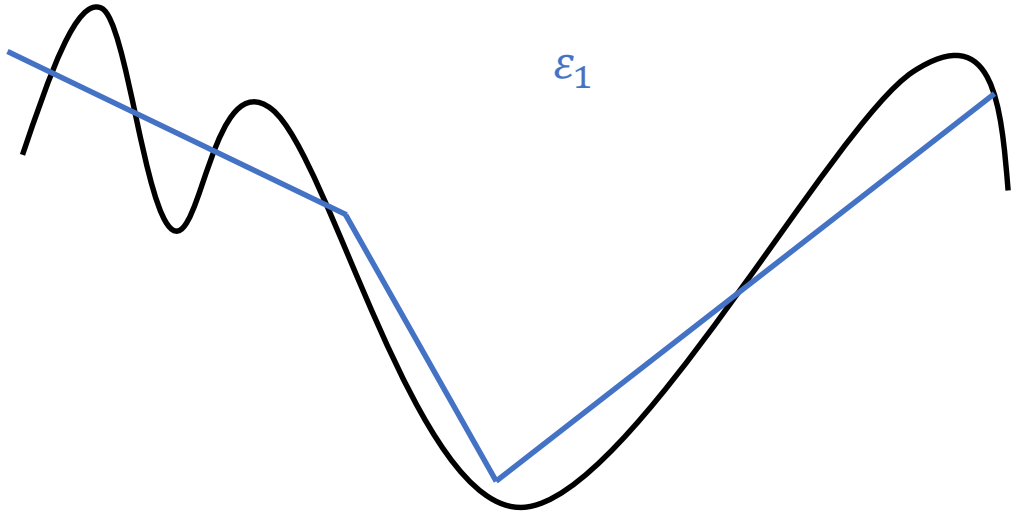

x_0
$w_{md}^{(1)}$
z_0
$w_m^{(2)}$
1
α_1
$f^{(1)}$
z_1
x_1
y
3
α_3
$f^{(1)}$
z_3



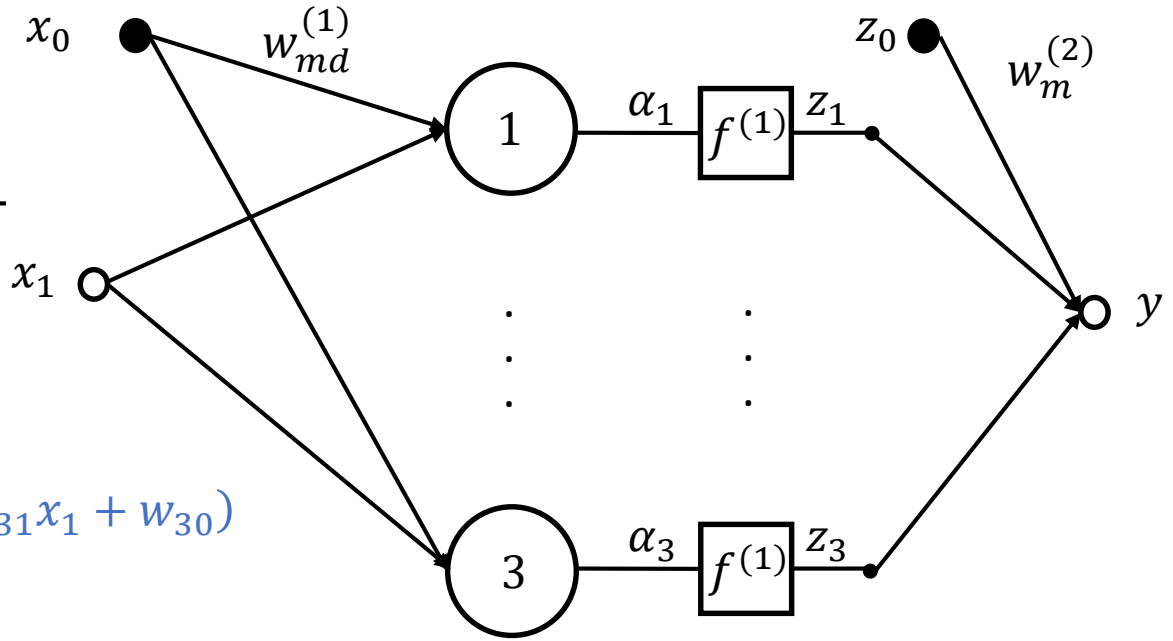
NN with ReLU

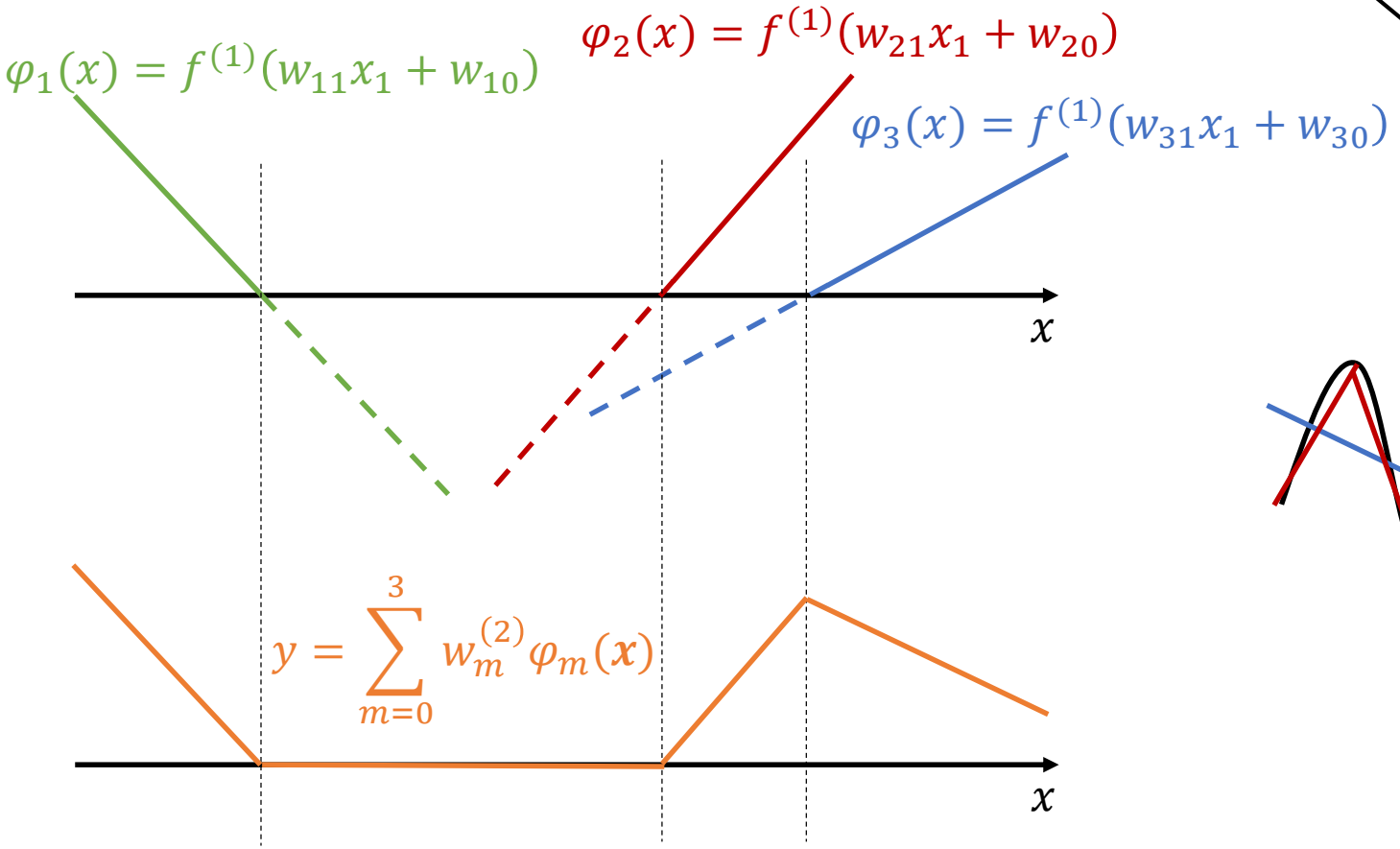

x_0
$w_{md}^{(1)}$
z_0
$w_m^{(2)}$
1
α_1
$f^{(1)}$
z_1
x_1
y
3
α_3
$f^{(1)}$
z_3

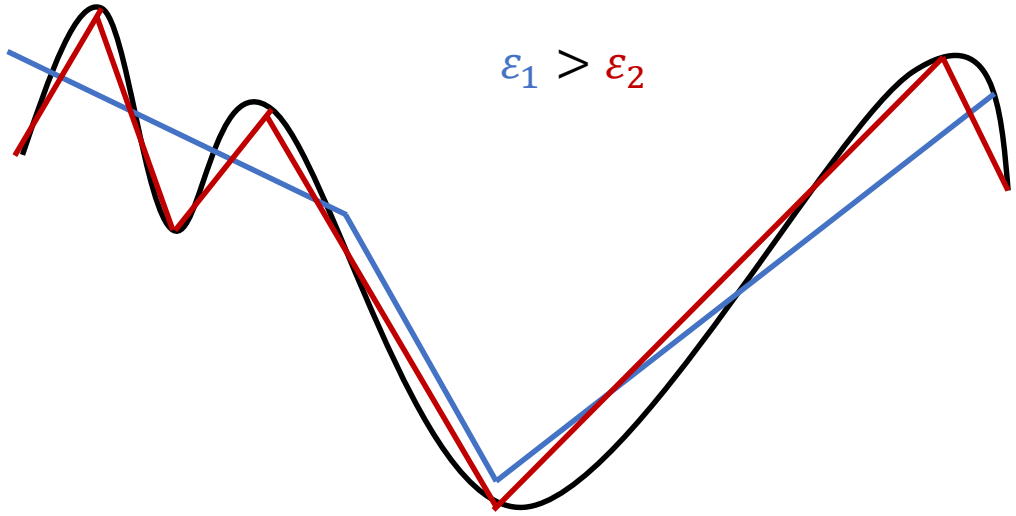

$\varphi_1(x) = f^{(1)}(w_{11}x_1 + w_{10})$
$\varphi_2(x) = f^{(1)}(w_{21}x_1 + w_{20})$
$\varphi_3(x) = f^{(1)}(w_{31}x_1 + w_{30})$
x
$y = \sum_{m=0}^{3} w_m^{(2)} \varphi_m(\boldsymbol{x})$
x


ε_1

NN with ReLU


x_0
$w_{md}^{(1)}$
z_0
$w_m^{(2)}$
1
α_1
$f^{(1)}$
z_1
x_1
y
3
α_3
$f^{(1)}$
z_3


$\varphi_1(x) = f^{(1)}(w_{11}x_1 + w_{10})$
$\varphi_2(x) = f^{(1)}(w_{21}x_1 + w_{20})$
$\varphi_3(x) = f^{(1)}(w_{31}x_1 + w_{30})$
x
$y = \sum_{m=0}^{3} w_m^{(2)} \varphi_m(\boldsymbol{x})$
x


$\varepsilon_1 > \varepsilon_2$

Expressive power of ReLU Networks

- The expressive power of ReLU-DNN networks is represented by the number of linear regions (see the example in the previous slide).
- In particular:

$$\#linear\ regions = width^{depth \times D}$$

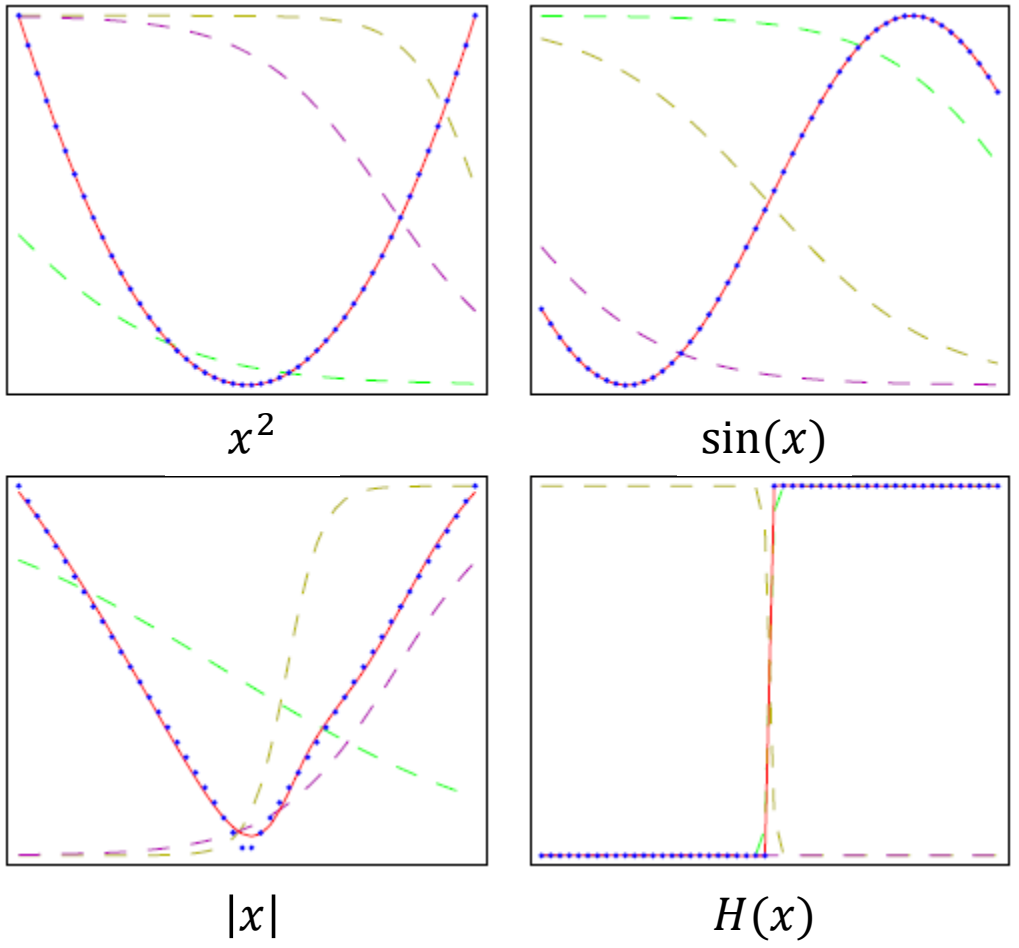
- Thus the expressive power is polynomial in width but exponential in depth.
- With fixed network capacity:

$$\#parameters = width^2 \times depth$$

- Most expressive power is gained by going deeper instead of making wider networks (per layer) and staying shallow!

Examples: Function Approximations

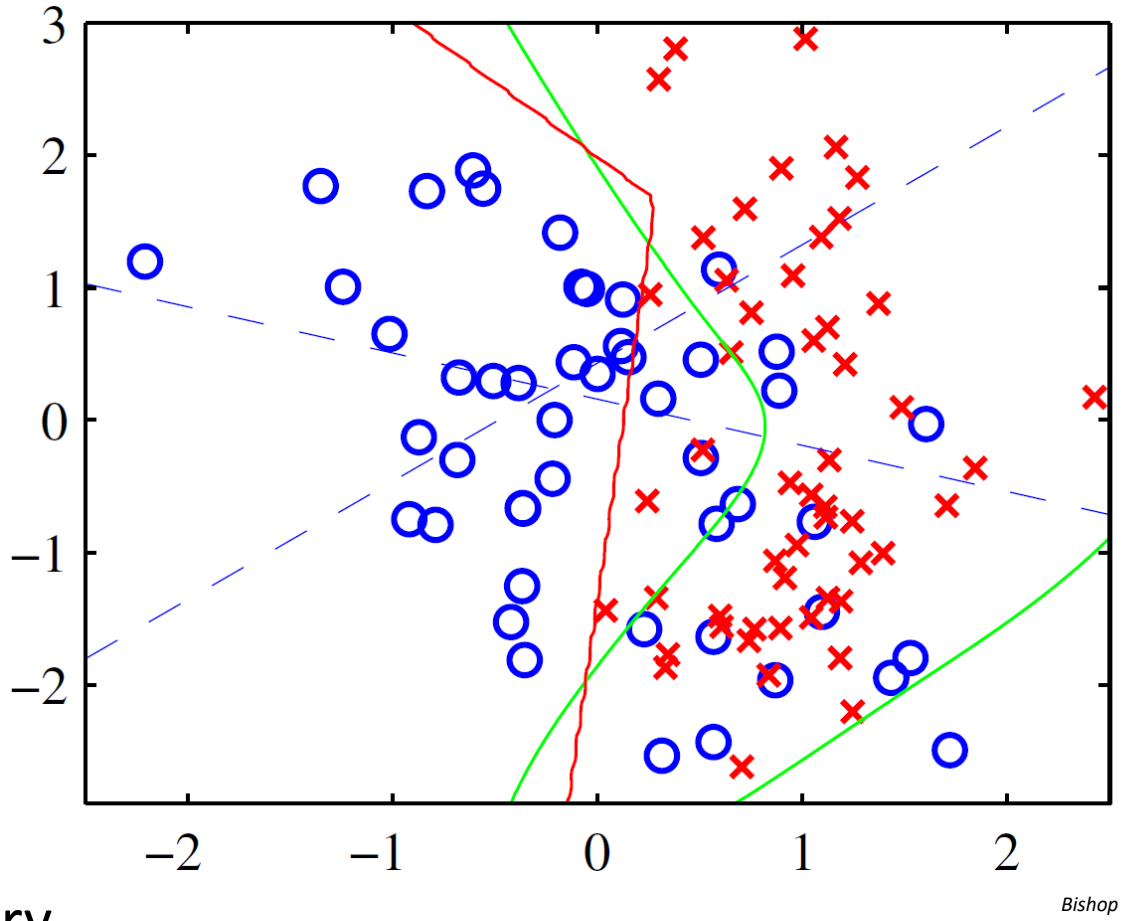
- $N = 50$
- $\#layers = 2$
- $M = 3$ (hidden units)
- Activation function: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- One linear output unit
- Dashed curves: Hidden units output



Bishop

Example: Classification

- $\#layers = 2$
- $M = 2$ (hidden units)
- Activation function: $\tanh(x)$
- One output unit with sigmoid activation
- Red line: 2-LP decision boundary
- Green line: optimal decision boundary



Bishop

Thus far…

- We have defined:
  - Number of inputs
  - Hidden layers
  - Output layers
  - Activation functions (of hidden layers)
- We will use now a probabilistic interpretation of the network outputs to choose:
  - Number of outputs
  - Output activation function
  - Loss function!

Binary classification

- Data: inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$, targets $\boldsymbol{t} = (t_1, \dots, t_N)$, where $\boldsymbol{x}_n \in \mathbb{R}^D, t \in [0,1]$.

Binary classification

- Data: inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$, targets $\boldsymbol{t} = (t_1, \dots, t_N)$, where $\boldsymbol{x}_n \in \mathbb{R}^D, t \in [0,1]$.
- For binary classification we define one output with target distribution:

$$p(t = 1|\boldsymbol{x}) = y(\boldsymbol{x}, \boldsymbol{w})$$ (output of the NN)

Binary classification

- Data: inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$, targets $\boldsymbol{t} = (t_1, \dots, t_N)$, where $\boldsymbol{x}_n \in \mathbb{R}^D, t \in [0,1]$.
- For binary classification we define one output with target distribution:

$$p(t = 1|\boldsymbol{x}) = y(\boldsymbol{x}, \boldsymbol{w}) \text{ (output of the NN)}$$

- In general:

$$p(t|\boldsymbol{x}, \boldsymbol{w}) = y(\boldsymbol{x}, \boldsymbol{w})^t \big(1 - y(\boldsymbol{x}, \boldsymbol{w})\big)^{1-t} \text{ (Bernoulli distribution)}$$

Binary classification

- Data: inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$, targets $\boldsymbol{t} = (t_1, \dots, t_N)$, where $\boldsymbol{x}_n \in \mathbb{R}^D, t \in [0,1]$.
- For binary classification we define one output with target distribution:

$$p(t = 1|\boldsymbol{x}) = y(\boldsymbol{x}, \boldsymbol{w}) \text{ (output of the NN)}$$

- In general:

$$p(t|\boldsymbol{x}, \boldsymbol{w}) = y(\boldsymbol{x}, \boldsymbol{w})^t \big(1 - y(\boldsymbol{x}, \boldsymbol{w})\big)^{1-t} \text{ (Bernoulli distribution)}$$

- Thus, we have a single output unit: $y(\boldsymbol{x}, \boldsymbol{w}) = f^L(a^{out}) = \sigma(a^{out}) \in [0,1]$.
- Maximum likelihood or Minimum Negative log likelihood:

Binary classification

- Data: inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$, targets $\boldsymbol{t} = (t_1, \dots, t_N)$, where $\boldsymbol{x}_n \in \mathbb{R}^D, t \in [0,1]$.
- For binary classification we define one output with target distribution:

$$p(t = 1|\boldsymbol{x}) = y(\boldsymbol{x}, \boldsymbol{w}) \text{ (output of the NN)}$$

- In general:

$$p(t|\boldsymbol{x}, \boldsymbol{w}) = y(\boldsymbol{x}, \boldsymbol{w})^t \big(1 - y(\boldsymbol{x}, \boldsymbol{w})\big)^{1-t} \text{ (Bernoulli distribution)}$$

- Thus, we have a single output unit: $y(\boldsymbol{x}, \boldsymbol{w}) = f^L(a^{out}) = \sigma(a^{out}) \in [0,1]$.
- Maximum likelihood or Minimum Negative log likelihood:

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N} t_n \ln\big(y(\boldsymbol{x}_n, \boldsymbol{w})\big) + (1 - t_n) \ln\big(1 - y(\boldsymbol{x}_n, \boldsymbol{w})\big) \text{ (Cross Entropy Loss)}$$

Classification with c classes

- Data: inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$, targets $\boldsymbol{T} = (\boldsymbol{t}_1, \ldots, \boldsymbol{t}_N)$, where $\boldsymbol{x}_n \in \mathbb{R}^D$, $and\ \boldsymbol{t}_n = [t_{n1}, \ldots, t_{nc}]$ (one hot encoding).

Classification with c classes

- Data: inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$, targets $\boldsymbol{T} = (\boldsymbol{t}_1, \dots, \boldsymbol{t}_N)$, where $\boldsymbol{x}_n \in \mathbb{R}^D$, $and\ \boldsymbol{t}_n = [t_{n1}, \dots, t_{nc}]$ (one hot encoding).

- For multi-class classification we define c output units and we assume target distribution:

$$p(\boldsymbol{t}_n|\boldsymbol{x}_n, \boldsymbol{w}) = \prod_{i=1}^{c} y_i(\boldsymbol{x}_n, \boldsymbol{w})^{t_{ni}}$$ (Generalized Bernoulli)

where

$$y_i(\boldsymbol{x}_n, \boldsymbol{w}) = p(\omega_i|\boldsymbol{x}_n, \boldsymbol{w})$$(output of the NN)

Classification with c classes

- Data: inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$, targets $\boldsymbol{T} = (\boldsymbol{t}_1, \dots, \boldsymbol{t}_N)$, where $\boldsymbol{x}_n \in \mathbb{R}^D$, $and\ \boldsymbol{t}_n = [t_{n1}, \dots, t_{nc}]$ (one hot encoding).

- For multi-class classification we define c output units and we assume target distribution:

$$p(\boldsymbol{t}_n|\boldsymbol{x}_n, \boldsymbol{w}) = \prod_{i=1}^{c} y_i(\boldsymbol{x}_n, \boldsymbol{w})^{t_{ni}}$$ (Generalized Bernoulli)

where

$$y_i(\boldsymbol{x}_n, \boldsymbol{w}) = p(\omega_i|\boldsymbol{x}_n, \boldsymbol{w})$$(output of the NN)

- For each output unit: $y_i(\boldsymbol{x}_n, \boldsymbol{w}) = f^L\left(a_i^{out}\right) = \frac{\exp(a_i^{out})}{\sum_{j=1}^{c} \exp\left(a_j^{out}\right)}$ (softmax activation function) with $\sum p(\omega_i|\boldsymbol{x}_n, \boldsymbol{w}) = \sum y_i(\boldsymbol{x}_n, \boldsymbol{w}) = 1$

Classification with c classes

- Data: inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$, targets $\boldsymbol{T} = (\boldsymbol{t}_1, \dots, \boldsymbol{t}_N)$, where $\boldsymbol{x}_n \in \mathbb{R}^D$, $and\ \boldsymbol{t}_n = [t_{n1}, \dots, t_{nc}]$ (one hot encoding).

- For multi-class classification we define c output units and we assume target distribution:

$$p(\boldsymbol{t}_n|\boldsymbol{x}_n, \boldsymbol{w}) = \prod_{i=1}^{c} y_i(\boldsymbol{x}_n, \boldsymbol{w})^{t_{ni}}$$ (Generalized Bernoulli)

where

$$y_i(\boldsymbol{x}_n, \boldsymbol{w}) = p(\omega_i|\boldsymbol{x}_n, \boldsymbol{w})$$(output of the NN)

- For each output unit: $y_i(\boldsymbol{x}_n, \boldsymbol{w}) = f^L\left(a_i^{out}\right) = \frac{\exp(a_i^{out})}{\sum_{j=1}^{c} exp\left(a_j^{out}\right)}$ (softmax activation function) with $\sum p(\omega_i|\boldsymbol{x}_n, \boldsymbol{w}) = \sum y_i(\boldsymbol{x}_n, \boldsymbol{w}) = 1$

- Maximum likelihood or Minimum Negative log likelihood:

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N}\sum_{i=1}^{c} t_{ni} \ln(y_i(\boldsymbol{x}_n, \boldsymbol{w}))$$(Cross Entropy Loss)

Regression

- Data: inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$, targets $\boldsymbol{t} = (t_1, \ldots, t_N)$, where $\boldsymbol{x}_n \in \mathbb{R}^D, t_n \in \mathbb{R}$.

Regression

- Data: inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$, targets $\boldsymbol{t} = (t_1, \dots, t_N)$, where $\boldsymbol{x}_n \in \mathbb{R}^D, t_n \in \mathbb{R}$.
- We assume target distribution: $p(t|\boldsymbol{x}_n, \boldsymbol{w}) = \mathcal{N}(t|y(\boldsymbol{x}_n, \boldsymbol{w}), \beta^{-1})$. Here the mean of the gaussian is $\boldsymbol{x}$ −dependent. Actually, it is the output of the NN.

Regression

- Data: inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$, targets $\boldsymbol{t} = (t_1, \dots, t_N)$, where $\boldsymbol{x}_n \in \mathbb{R}^D, t_n \in \mathbb{R}$.
- We assume target distribution: $p(t|\boldsymbol{x}_n, \boldsymbol{w}) = \mathcal{N}(t|y(\boldsymbol{x}_n, \boldsymbol{w}), \beta^{-1})$. Here the mean of the gaussian is $\boldsymbol{x}$ −dependent. Actually, it is the output of the NN.
- We adopt a single output NN. In particular, $y(\boldsymbol{x}_n, \boldsymbol{w}) = f^L(a^{out})$.
- Because the outputs are real valued we assume an identity output activation function:$y(\boldsymbol{x}_n, \boldsymbol{w}) = f^L(a^{out}) = a^{out}$.

Regression

- Data: inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$, targets $\boldsymbol{t} = (t_1, \dots, t_N)$, where $\boldsymbol{x}_n \in \mathbb{R}^D, t_n \in \mathbb{R}$.
- We assume target distribution: $p(t|\boldsymbol{x}_n, \boldsymbol{w}) = \mathcal{N}(t|y(\boldsymbol{x}_n, \boldsymbol{w}), \beta^{-1})$. Here the mean of the gaussian is $\boldsymbol{x}$ −dependent. Actually, it is the output of the NN.
- We adopt a single output NN. In particular, $y(\boldsymbol{x}_n, \boldsymbol{w}) = f^L(a^{out})$.
- Because the outputs are real valued we assume an identity output activation function:$y(\boldsymbol{x}_n, \boldsymbol{w}) = f^L(a^{out}) = a^{out}$.
- Maximum likelihood or Minimum Negative log likelihood:

$$E(\boldsymbol{w}) = -\ln p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}) = \frac{\beta}{2}\sum_{n=1}^{N}(y(\boldsymbol{x}_n, \boldsymbol{w}) - t_n)^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln 2\pi$$

- In essence: $E(\boldsymbol{w}) = \sum_{n=1}^{N}(y(\boldsymbol{x}_n, \boldsymbol{w}) - t_n)^2$

Losses overview

- Binary classification:
  - Assumed Bernoulli target distribution
  - NN makes prediction of the probability of class ω_1
  - Output activation is logistic sigmoid
  - Minimize cross entropy loss
- Multi-class classification
  - Assumed generalized Bernoulli target distribution
  - NN makes prediction of the probability of each class
  - Output activation is softmax function
  - Minimize (multi-class) cross entropy loss.
- Regression
  - Assumed gaussian target distribution
  - NN makes prediction for the mean
  - Output activation function is the identity function
  - Minimize least squares.

Training

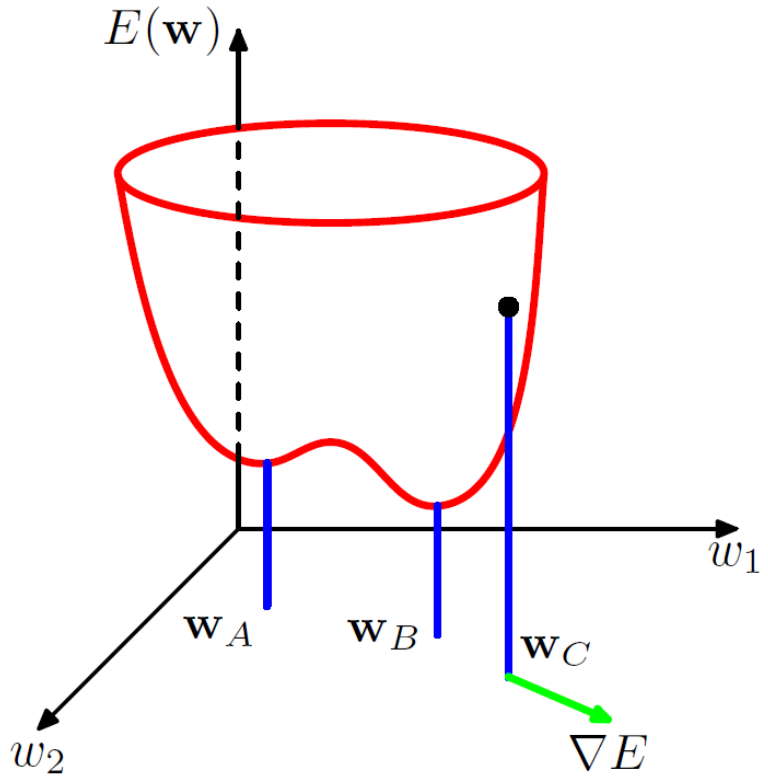
- Thus far:
  - Structure
  - different components of the NNs
  - activation functions
  - loss functions

- Next:

  Training: How can I adjust $\boldsymbol{w}$ in order to approximate the function I am interested in.

Optimization

- For each task I have a different loss function $E(\boldsymbol{w})$ that I want to minimize.
- Parameter optimization: $\boldsymbol{w_{opt}} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, E(\boldsymbol{w})$
- A big problem: $E(\boldsymbol{w})$ is not convex in $\boldsymbol{w}$, thus several local optimal exist!
- How to reach a global minimum?

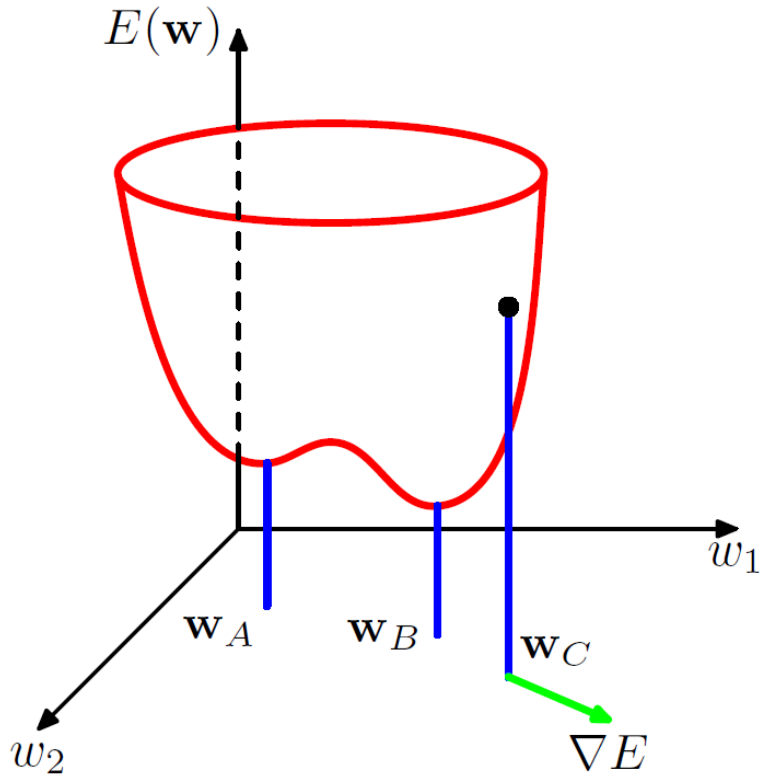


Bishop

Optimization

- For each task I have e different loss function $E(\boldsymbol{w})$ that I want to minimize.
- Parameter optimization: $\boldsymbol{w_{opt}} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, E(\boldsymbol{w})$
- A big problem: $E(\boldsymbol{w})$ is not convex in $\boldsymbol{w}$, thus several local optimal exist!
- How to reach a global minimum?

Stochastic Gradient Descent



Bishop

Stochastic gradient descent

- Gradient descent: $\boldsymbol{w}^{i+1} = \boldsymbol{w}^{i} - \eta \nabla E(\boldsymbol{w}^{i})$

  where $E(\boldsymbol{w}^{i}) = \sum_{n=1}^{N} E_n(\boldsymbol{w}^{i})$

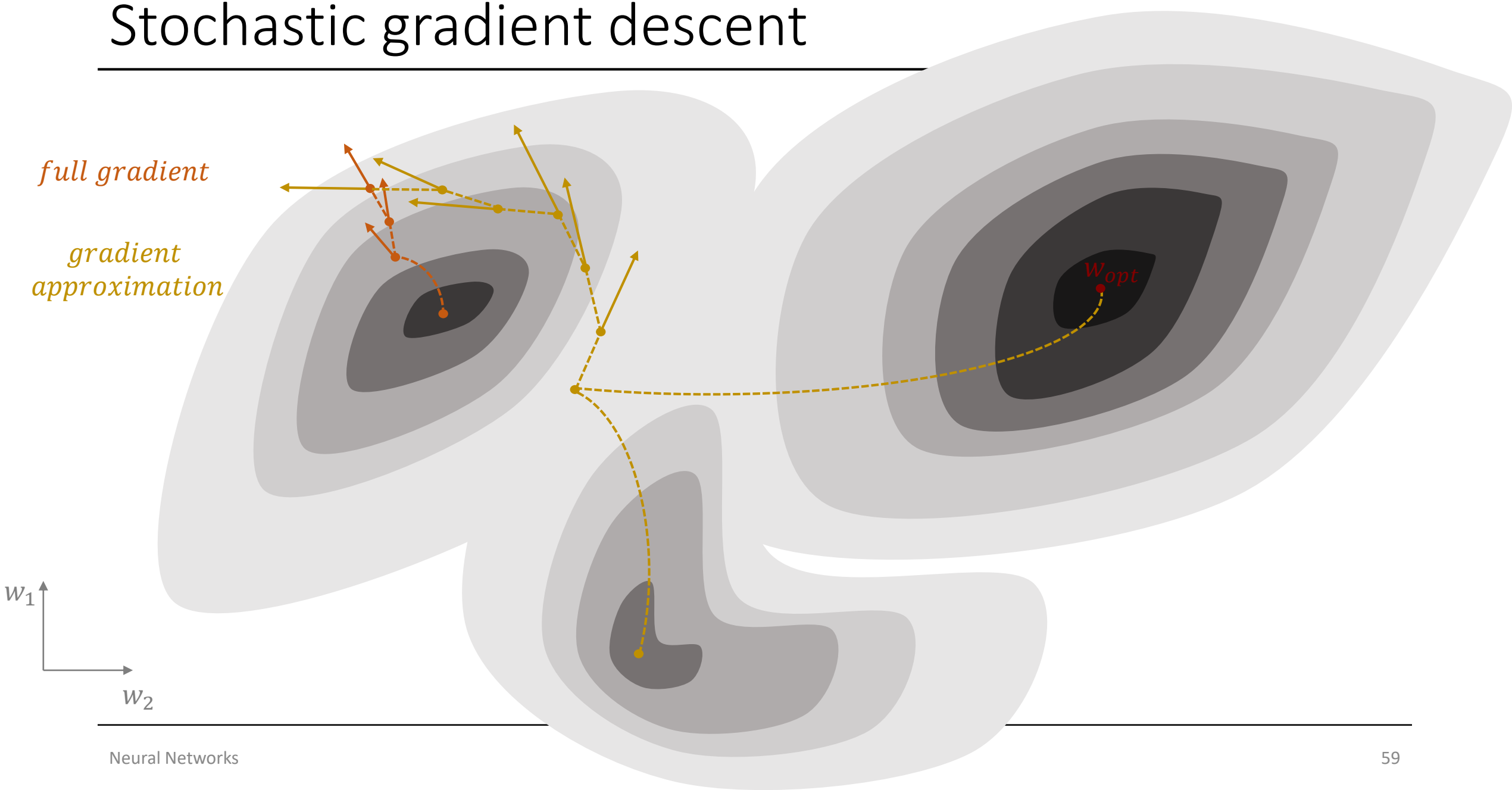
- By incorporating Stochasticity:

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^{i} - \eta \nabla E_n(\boldsymbol{w}^{i})$$

or

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^{i} - \eta \nabla \sum_{l=1}^{L} E_l(\boldsymbol{w}^{i}) \textit{ (minibatch)}$$

Stochastic gradient descent



Comments on SGD

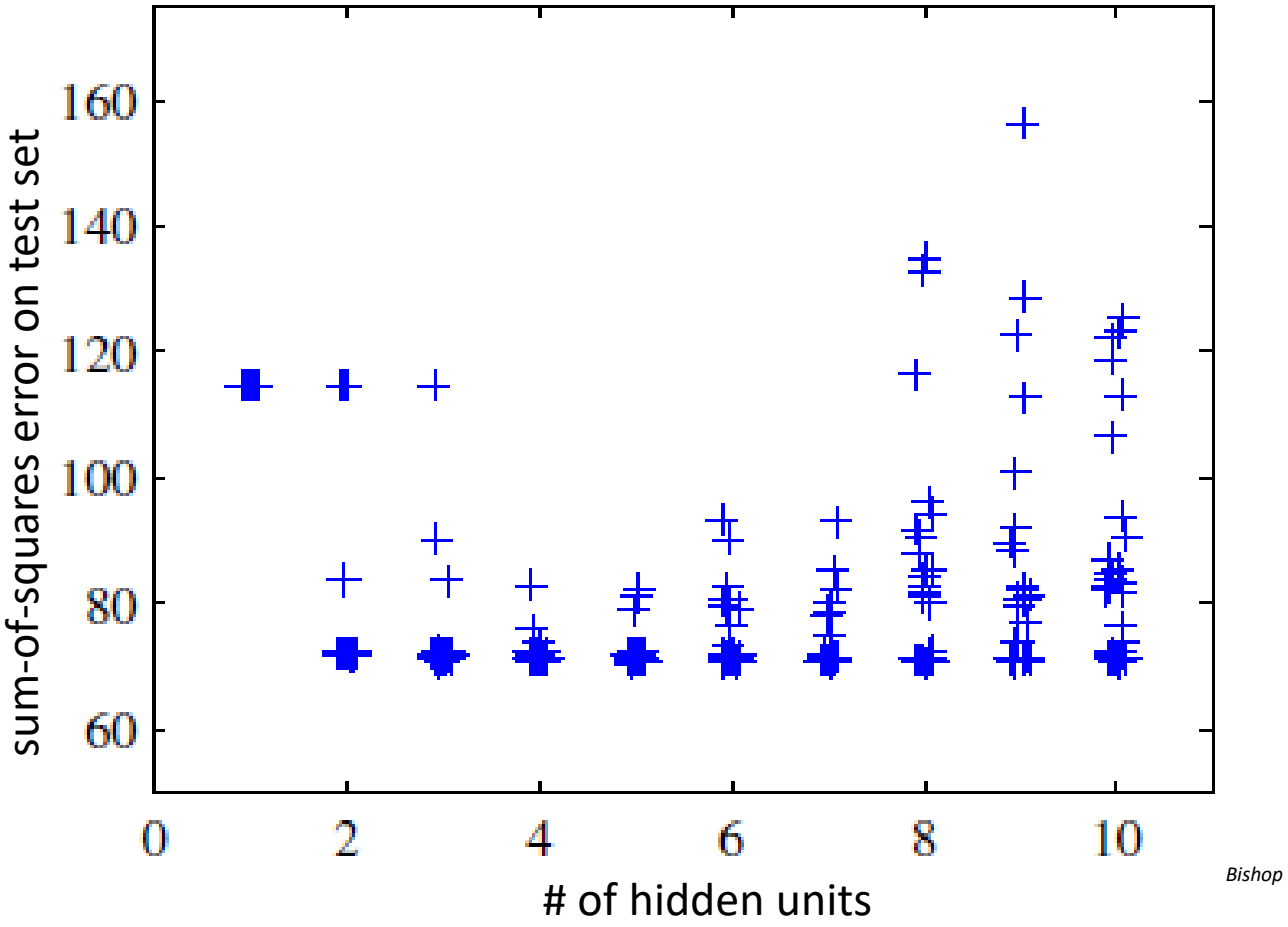
- Learning rate η too small: slow convergence
- Learning rate η too large: oscillations (no convergence)
- We most often use rate scheduling with smaller and smaller rates over time.
- SGD is more likely to escape a local minimum as:

$$\nabla E(\boldsymbol{w}) = \mathbf{0} \text{ does not necessarily imply } \nabla E_n(\boldsymbol{w}) = \mathbf{0}!$$

- As the number of hidden units increases the landscape becomes more complex and thus more initializations are needed to prove the robustness of the network

Comments on SGD

- Try several initialization of $\boldsymbol{w}$ to test uncertainty on performance



Bishop

ARISTOTLE UNIVERSITY OF THESSALONIKI

FACULTY OF ENGINEERING

Questions?

Pattern Recognition & Machine Learning

Neural Networks