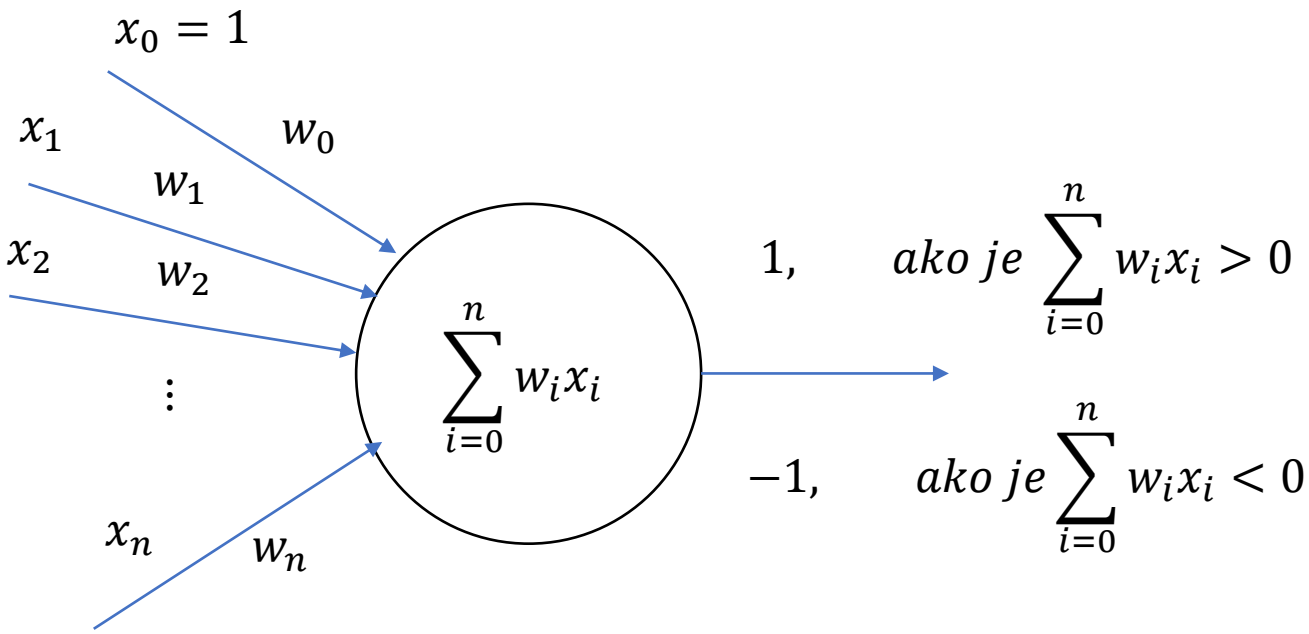
Neuronske mreže

Osnovni pojmovi

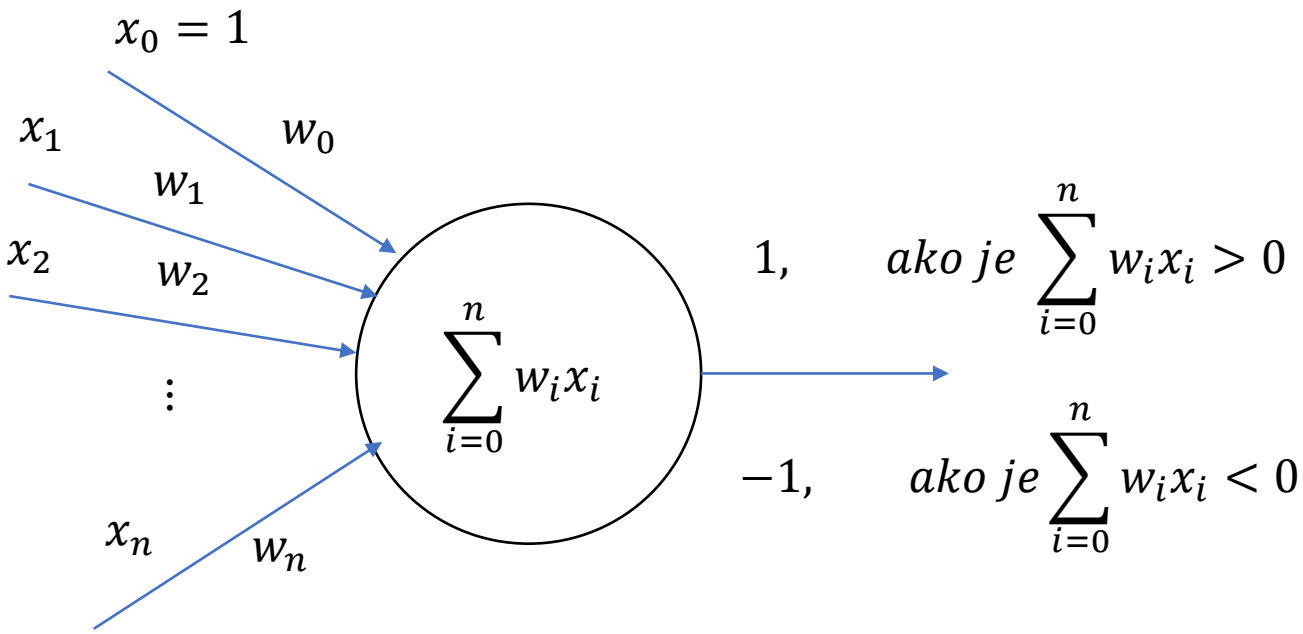
Perceptron

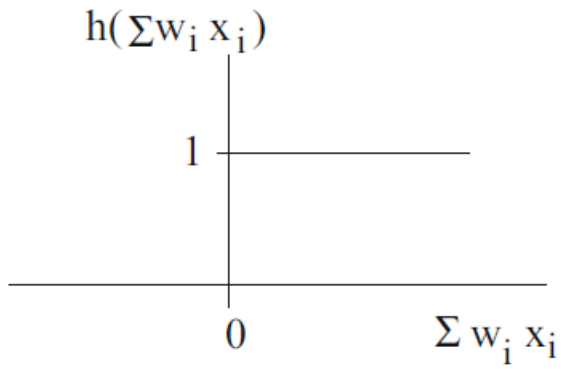


$$1, \quad ako\ je \sum_{i=0}^{n} w_i x_i > 0$$

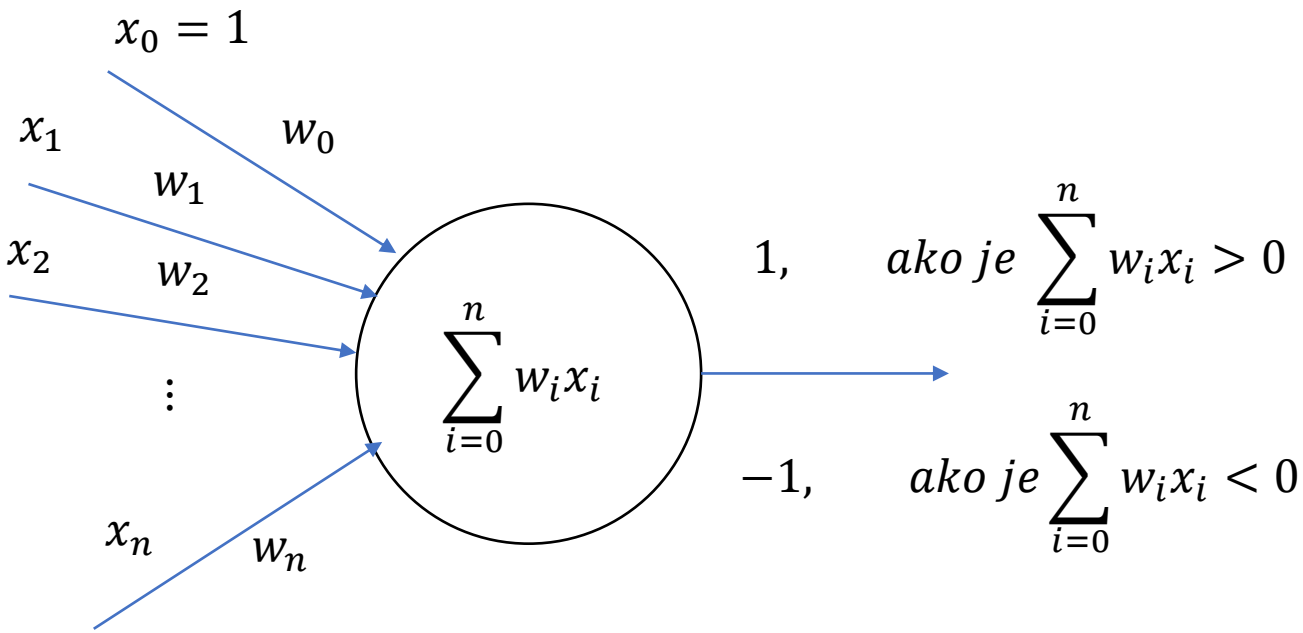
$$-1, \quad ako\ je \sum_{i=0}^{n} w_i x_i < 0$$

Perceptron


$x_0 = 1$
x_1
x_2
x_n
w_0
w_1
w_2
w_n
$\sum_{i=0}^{n} w_i x_i$
$1, \quad ako\ je \sum_{i=0}^{n} w_i x_i > 0$
$-1, \quad ako\ je \sum_{i=0}^{n} w_i x_i < 0$


h(Σw_i x_i)
1
0
Σ w_i x_i

Perceptron

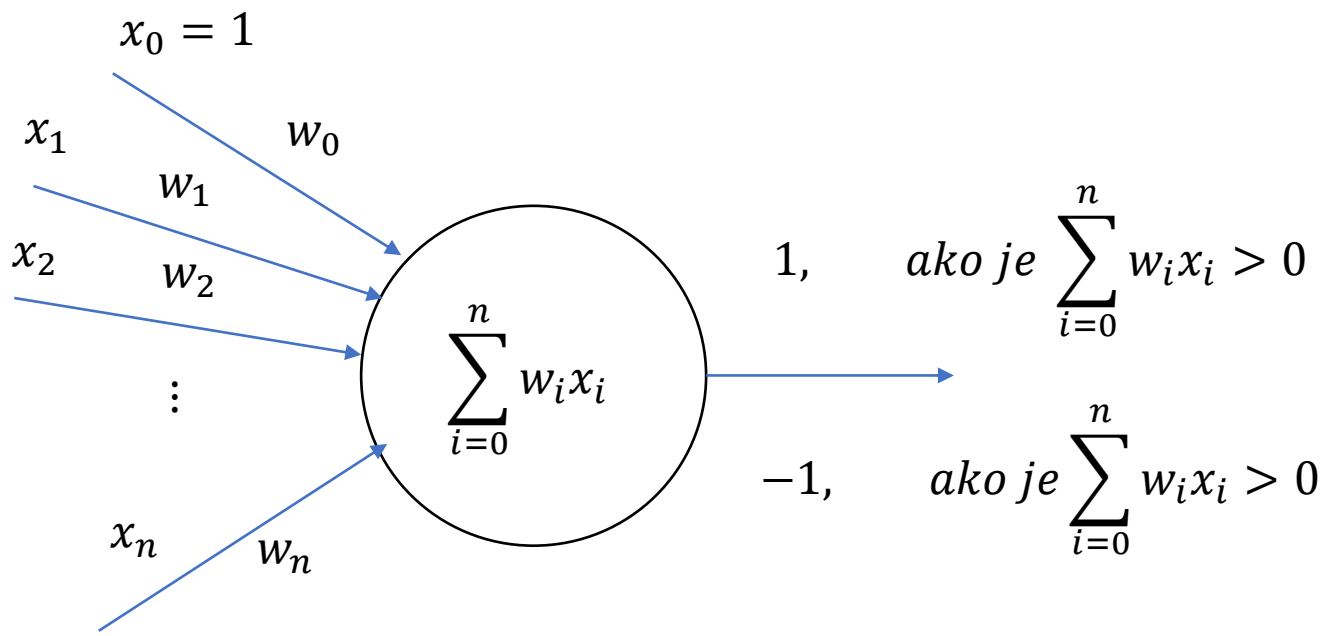


$$1, \quad ako\ je \sum_{i=0}^{n} w_i x_i > 0$$

$$-1, \quad ako\ je \sum_{i=0}^{n} w_i x_i < 0$$

$$a = \sum_{i=0}^{n} w_i x_i$$

Perceptron



$$1, \quad ako\ je \sum_{i=0}^{n} w_i x_i > 0$$

$$-1, \quad ako\ je \sum_{i=0}^{n} w_i x_i > 0$$

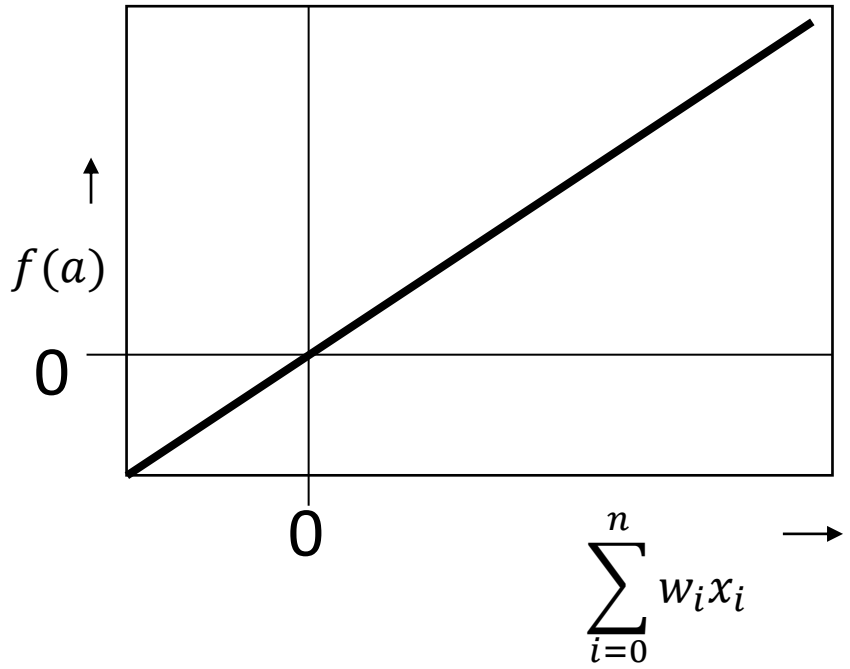
$$a = \sum_{i=0}^{n} w_i x_i$$

$$f(a)$$

Linearna aktivacionа funkcija

- Jednostavni, ali ograničenih mogućnosti

$$f(a) = \sum_{i=0}^{n} w_i x_i$$



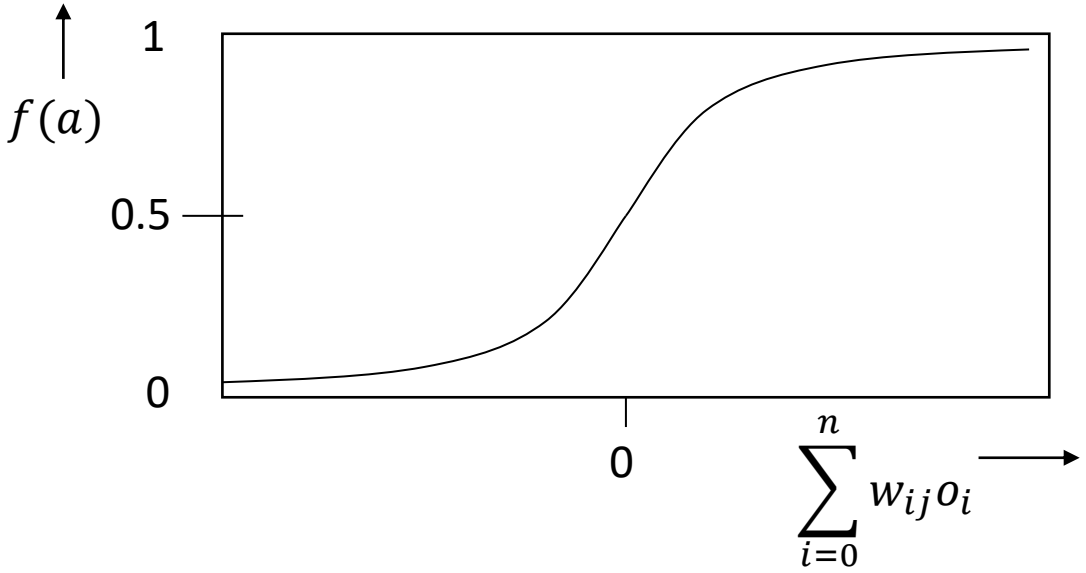
Sigmoid aktivacionom funkcijom

- Glatka, neprekidna funkcija.
- Izlaz je realan broj.
- Lepi izvodi.

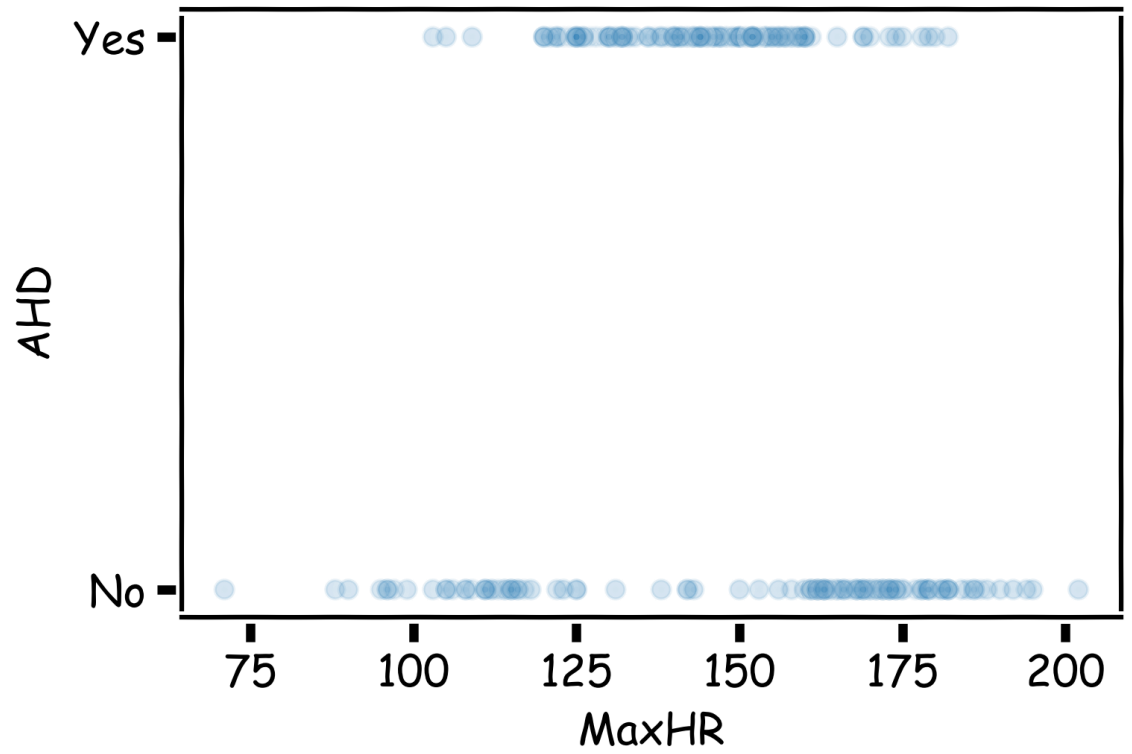
$$f(a) = \frac{1}{1 + e^{-a}}$$

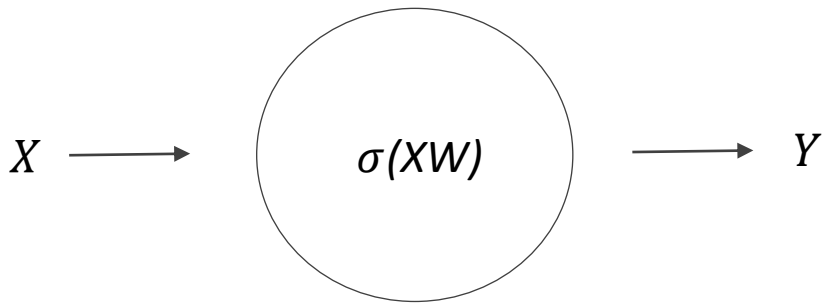
$$f' = f(1 - f)$$

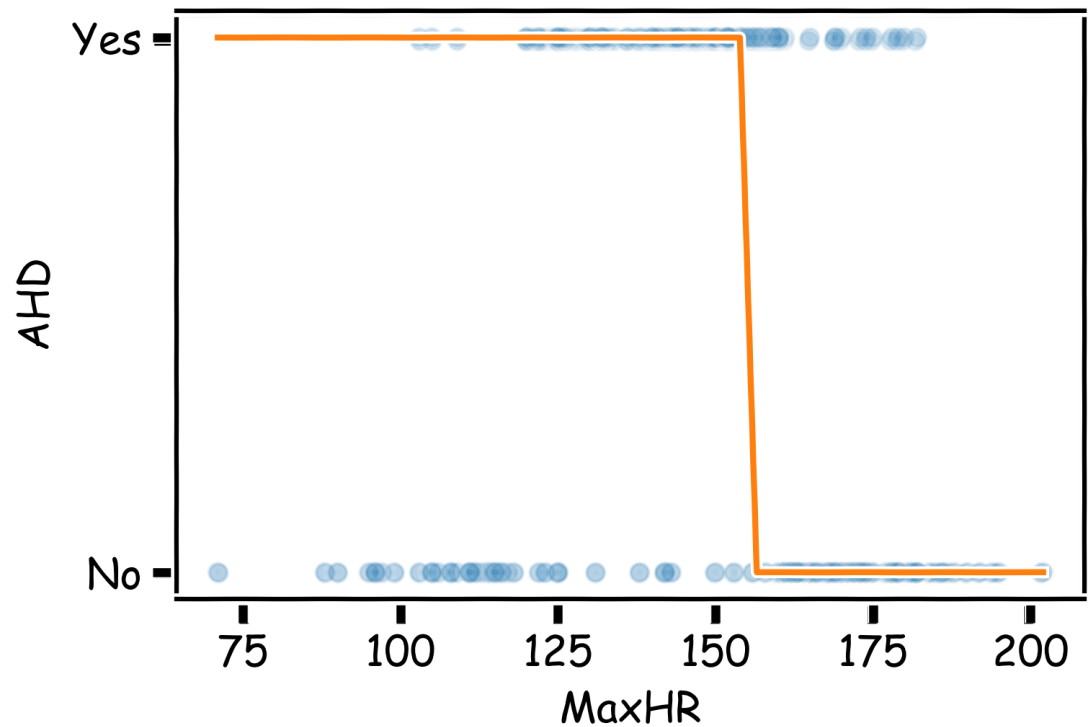
$$a = \sum_{i=0}^{n} w_i x_i$$

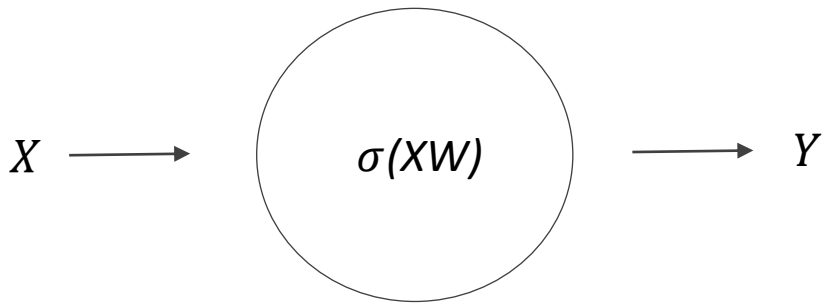


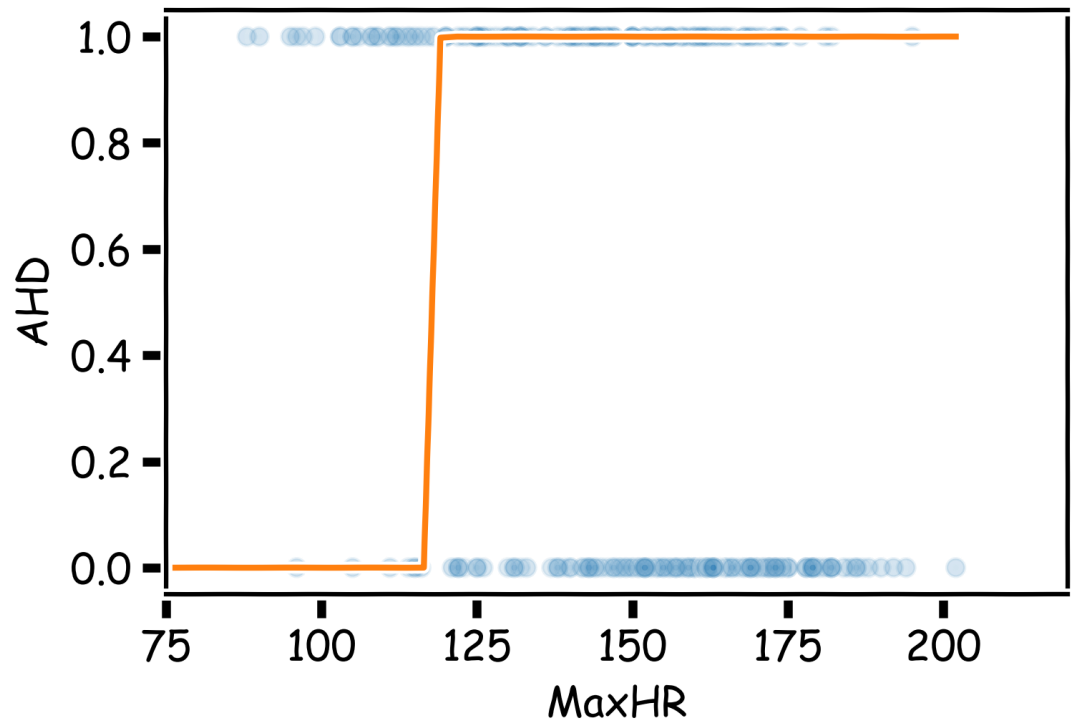
Primer podataka

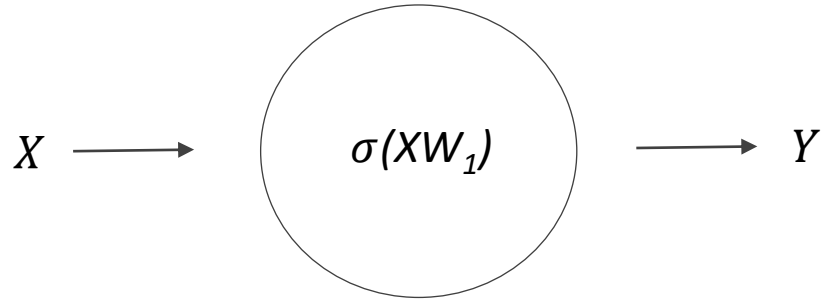


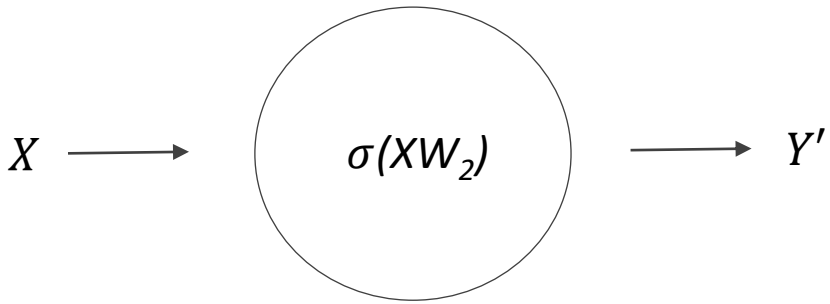

X
σ(XW)
Y

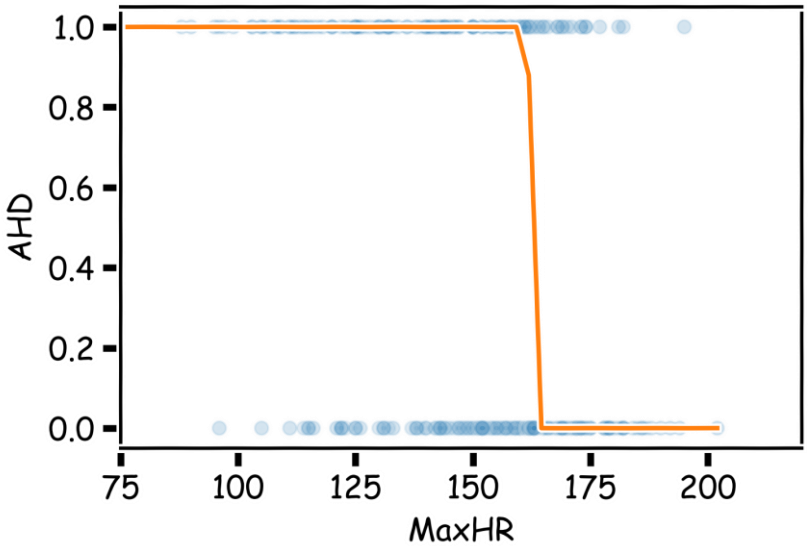

Yes
No
AHD
75
100
125
150
175
200
MaxHR

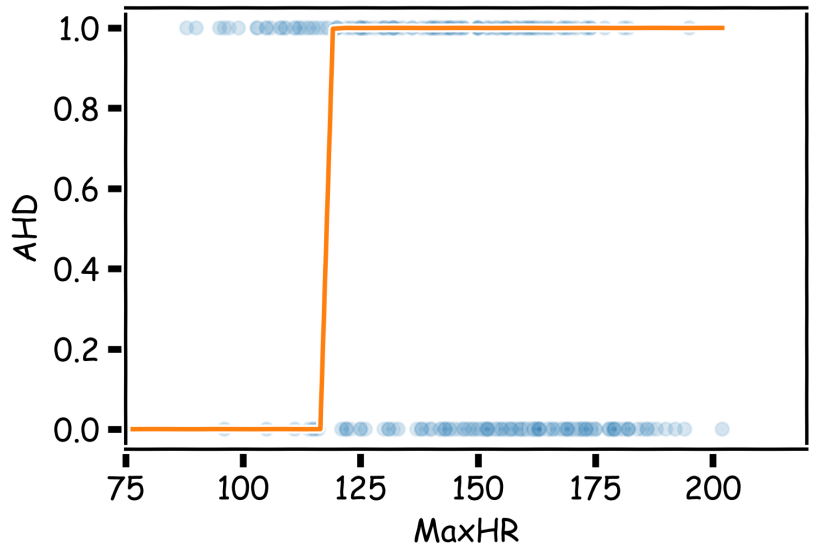

X
σ(XW)
Y

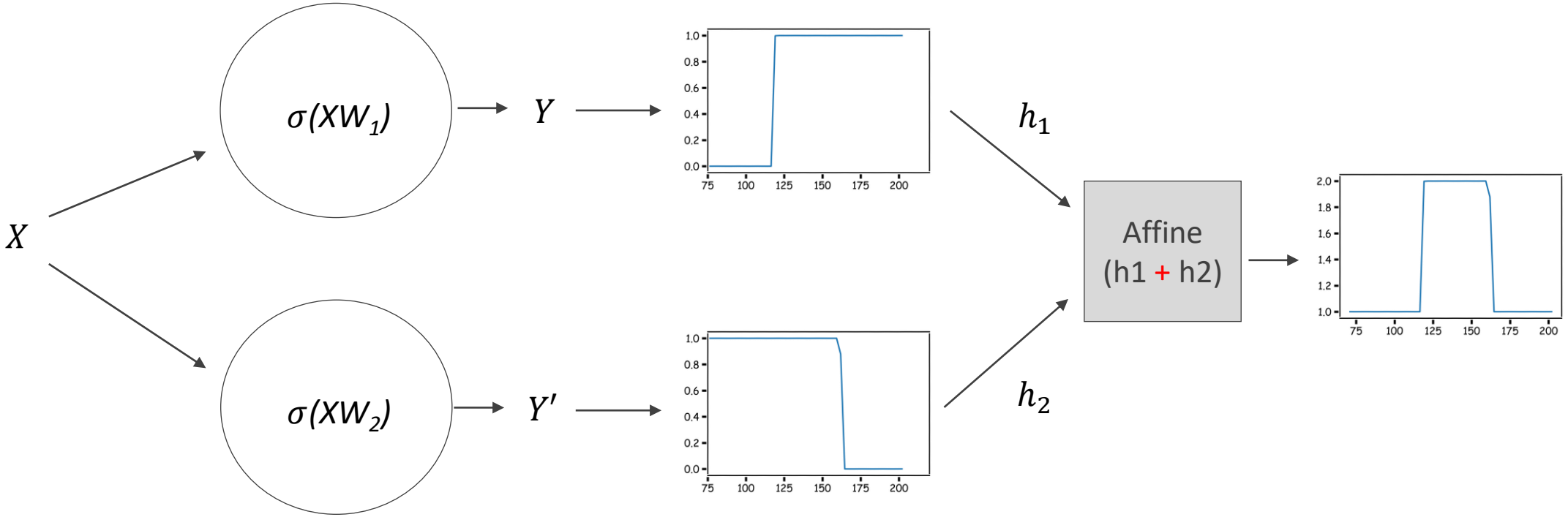

1.0
0.8
0.6
0.4
0.2
0.0
AHD
75
100
125
150
175
200
MaxHR

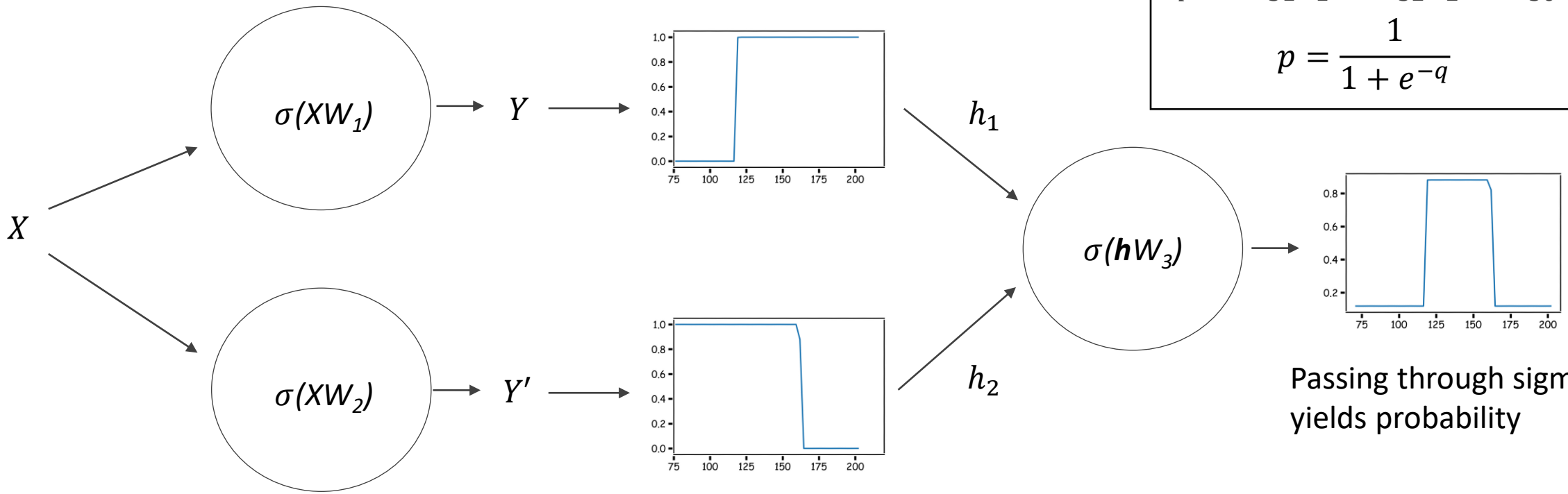

X
σ(XW1)
Y


X
σ(XW2)
Y′


AHD
1.0
0.8
0.6
0.4
0.2
0.0
75
100
125
150
175
200
MaxHR


AHD
1.0
0.8
0.6
0.4
0.2
0.0
75
100
125
150
175
200
MaxHR


X
$\sigma(XW_1)$
$\sigma(XW_2)$
Y
Y'
h_1
h_2
Affine
(h1 + h2)



$$q = W_{31}h_1 + W_{32}h_2 + W_{30}$$

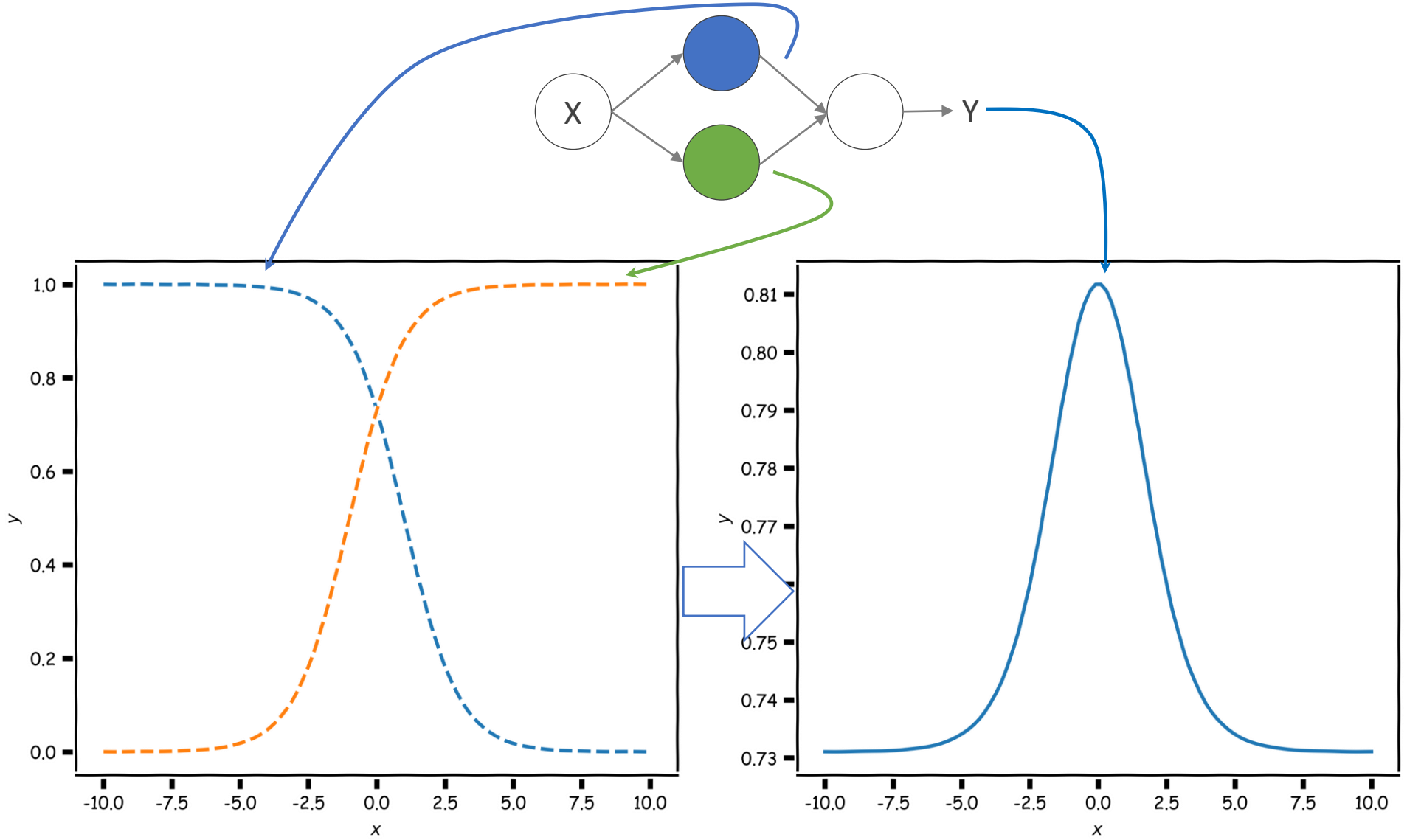
$$p = \frac{1}{1 + e^{-q}}$$

Passing through sigmoid yields probability

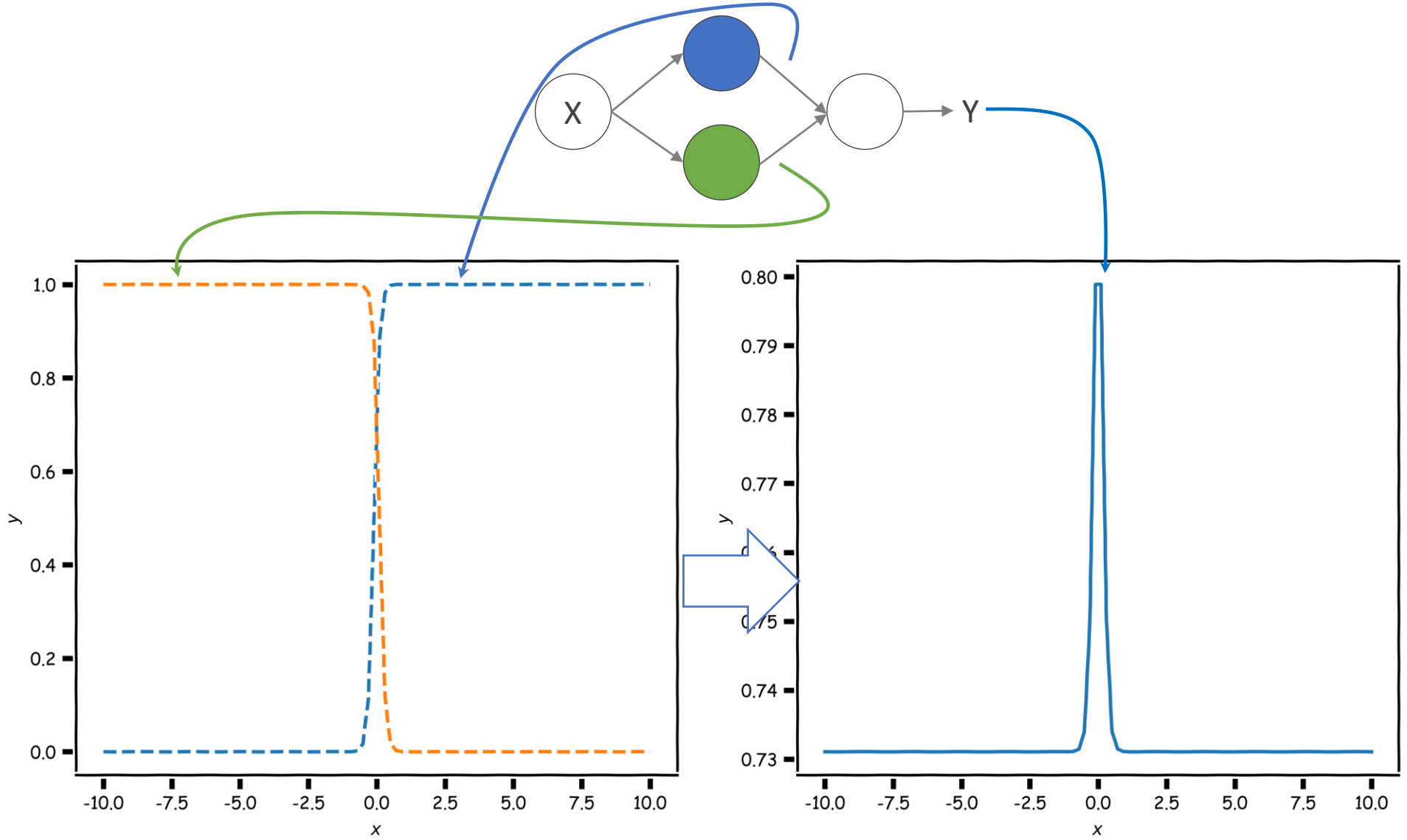
$$L = -y\ln(\mathrm{p}) - (1 - \mathrm{y})\ln(1 - \mathrm{p})$$

Need to learn W1, W2 and W3.

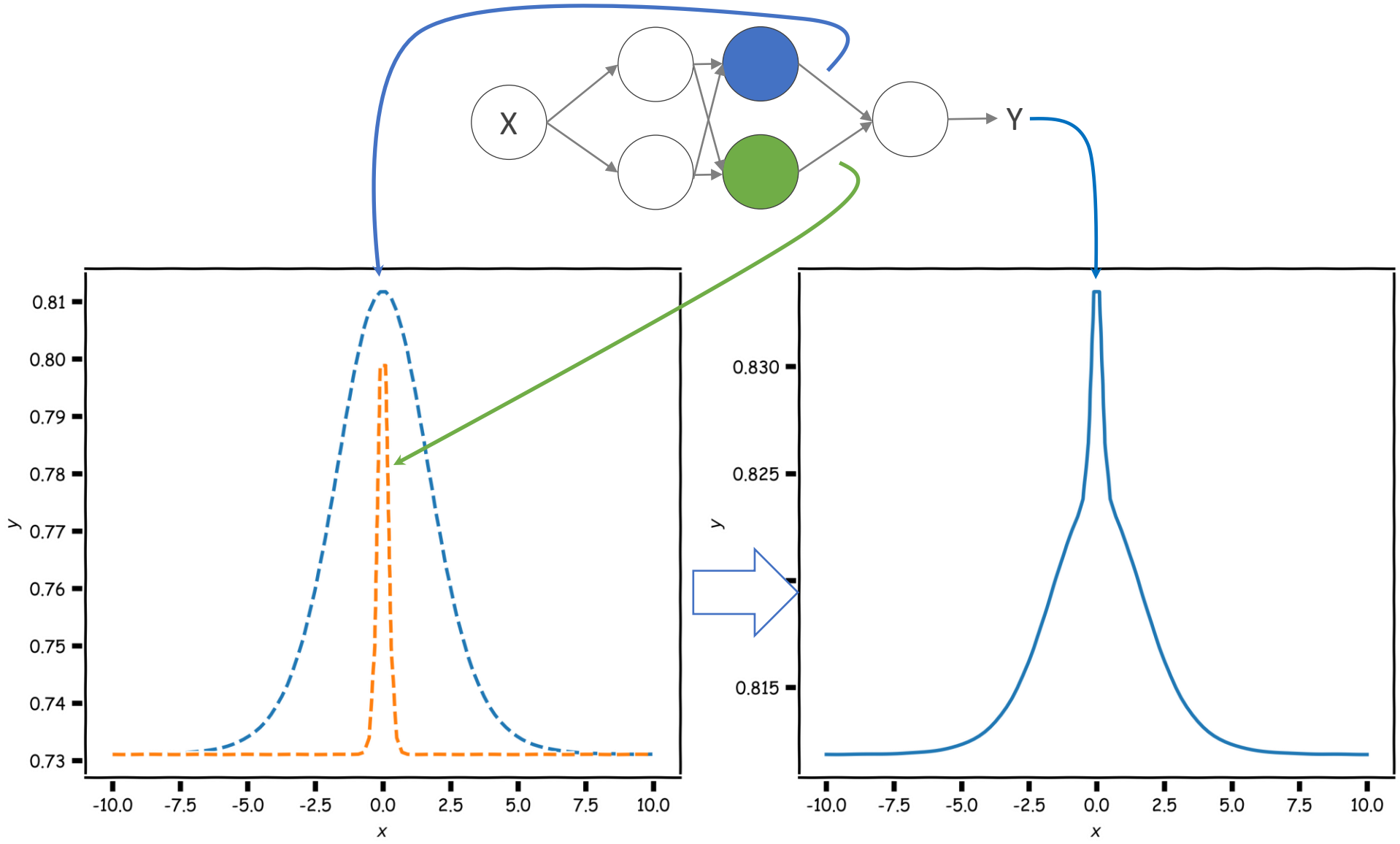
Combining neurons allows us to model interesting functions



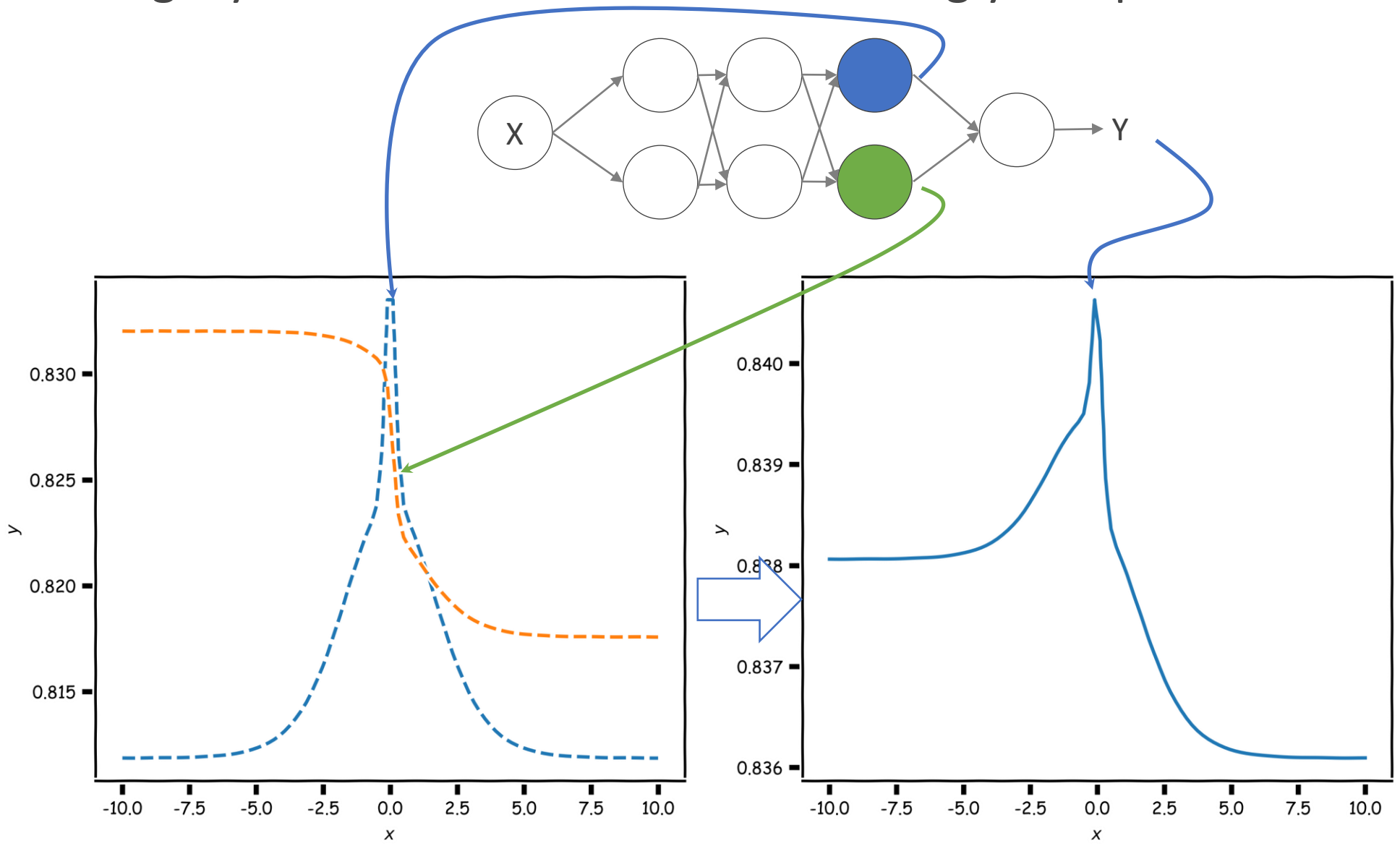
Different weights change the shape and position



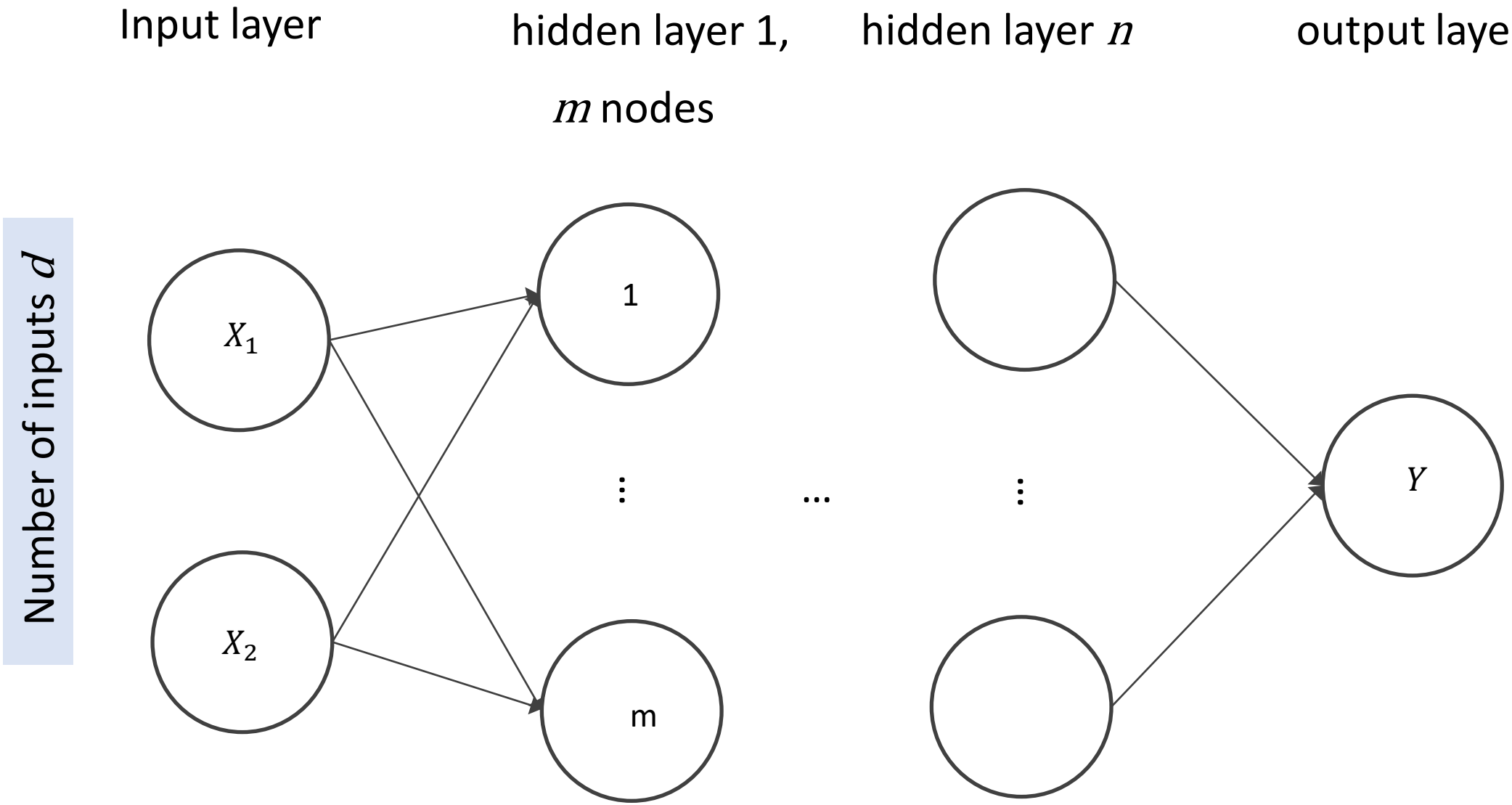
Neural networks can model *any* reasonable function



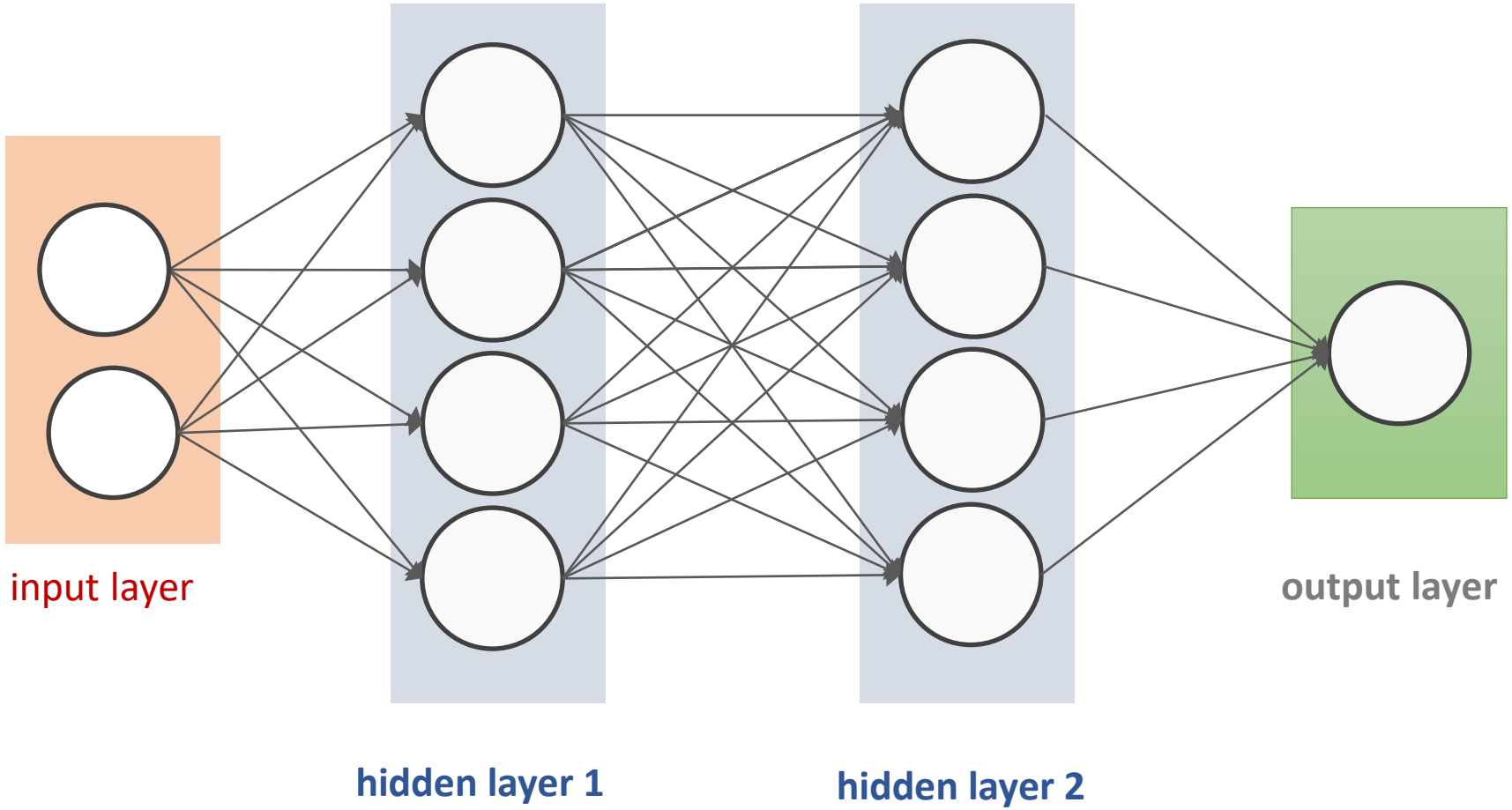
Adding layers allows us to model increasingly complex functions



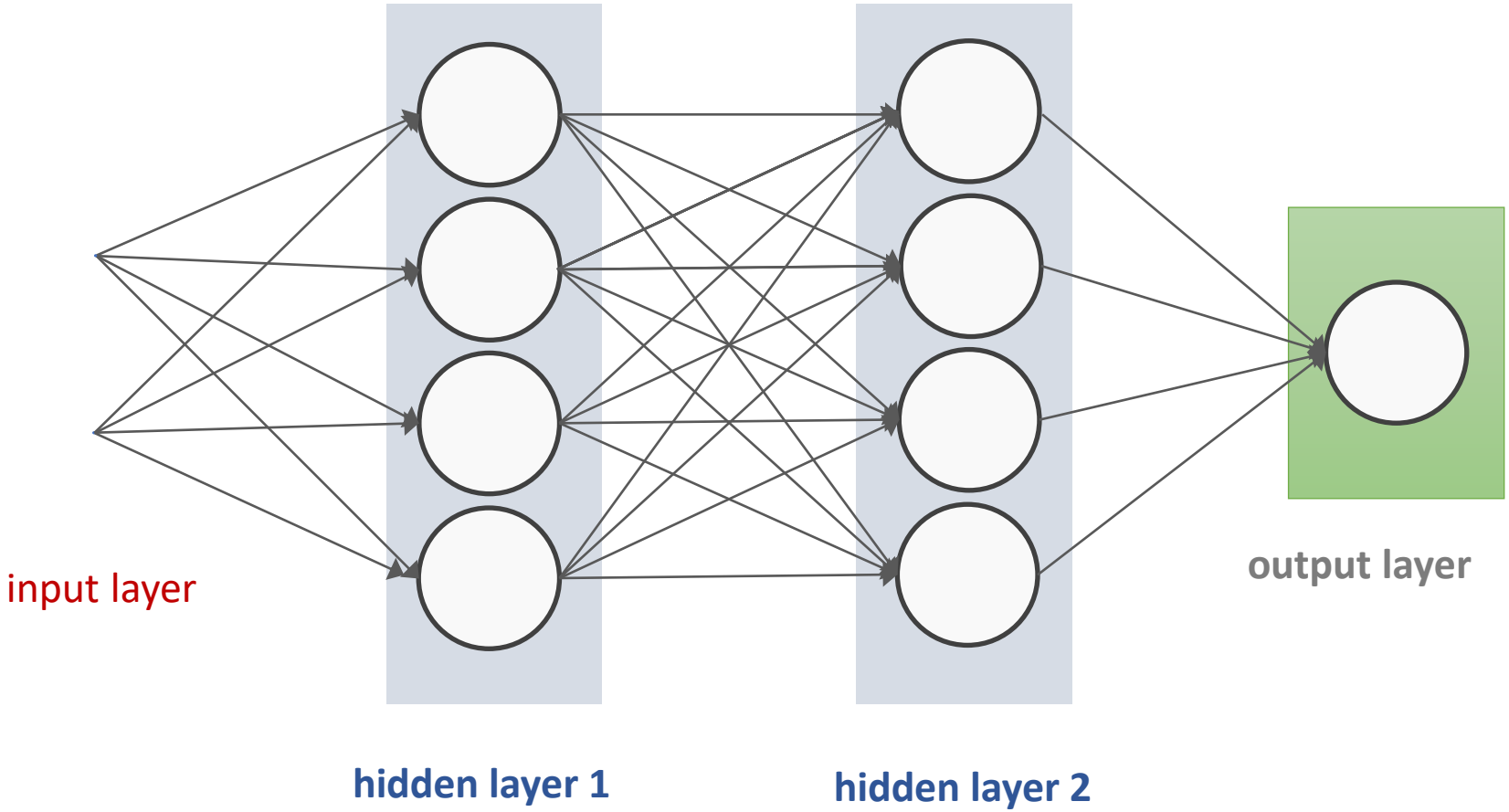
Artificial neural network (ANN)



Artificial neural network (ANN)



Artificial neural network (ANN)



Opcije za dizajniranje ANN

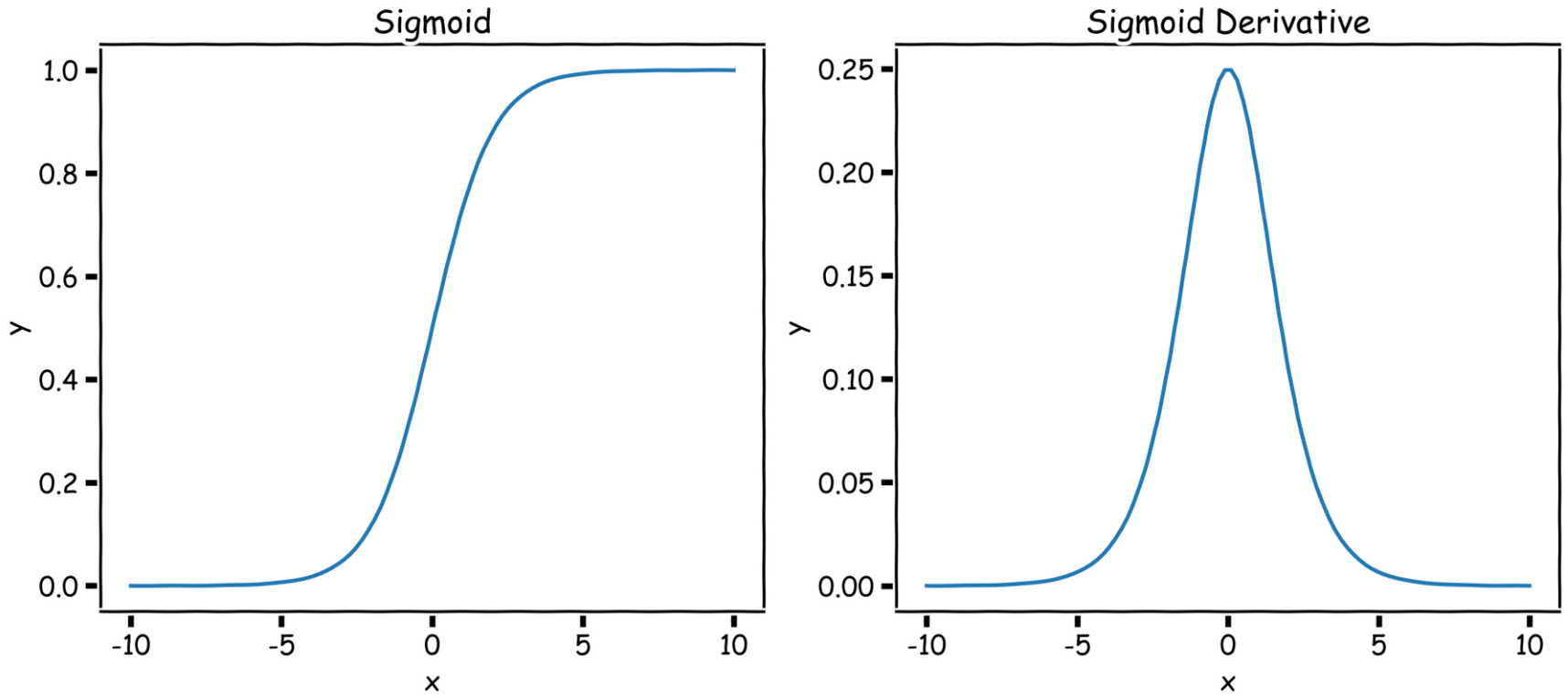
- Izbor aktivacione funkcije (Activation function)
- Izbor funkcije gubitka (Loss function)
- Izlazni neuroni (Output units)
- Arhitektura

Opcije za dizajniranje ANN

- **Izbor aktivacione funkcije**
  - **Nelinearnost**
  - **Nenula gradijenti u skrivenim čvorovima**
- Izbor funkcije gubitka (Loss function)
- Izlazni neuroni (Output units)
- Arhitektura

Sigmoid

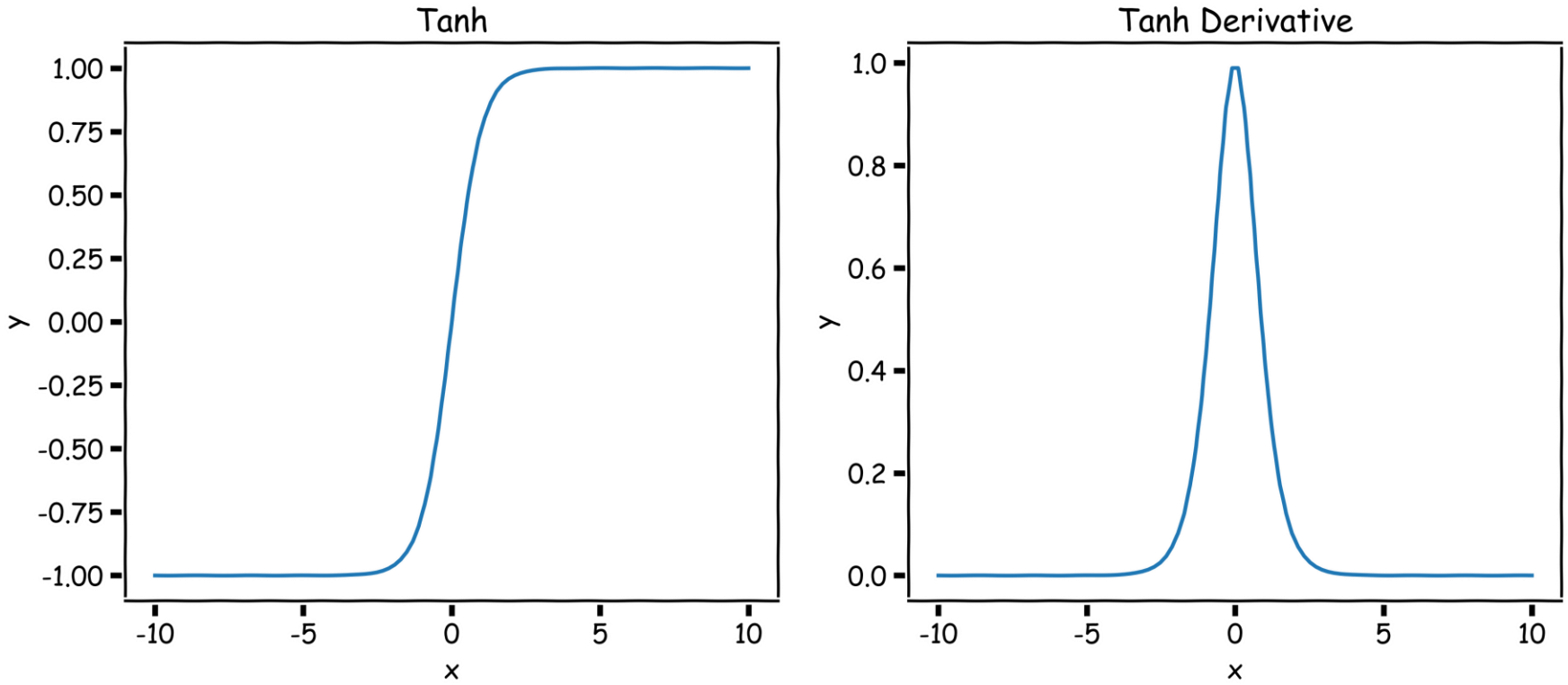
$$y = \frac{1}{1 + e^{-x}}$$



Derivative is **zero** for much of the domain. This leads to “vanishing gradients” in backpropagation.

Tangens hiperbolični (Tanh)

$$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

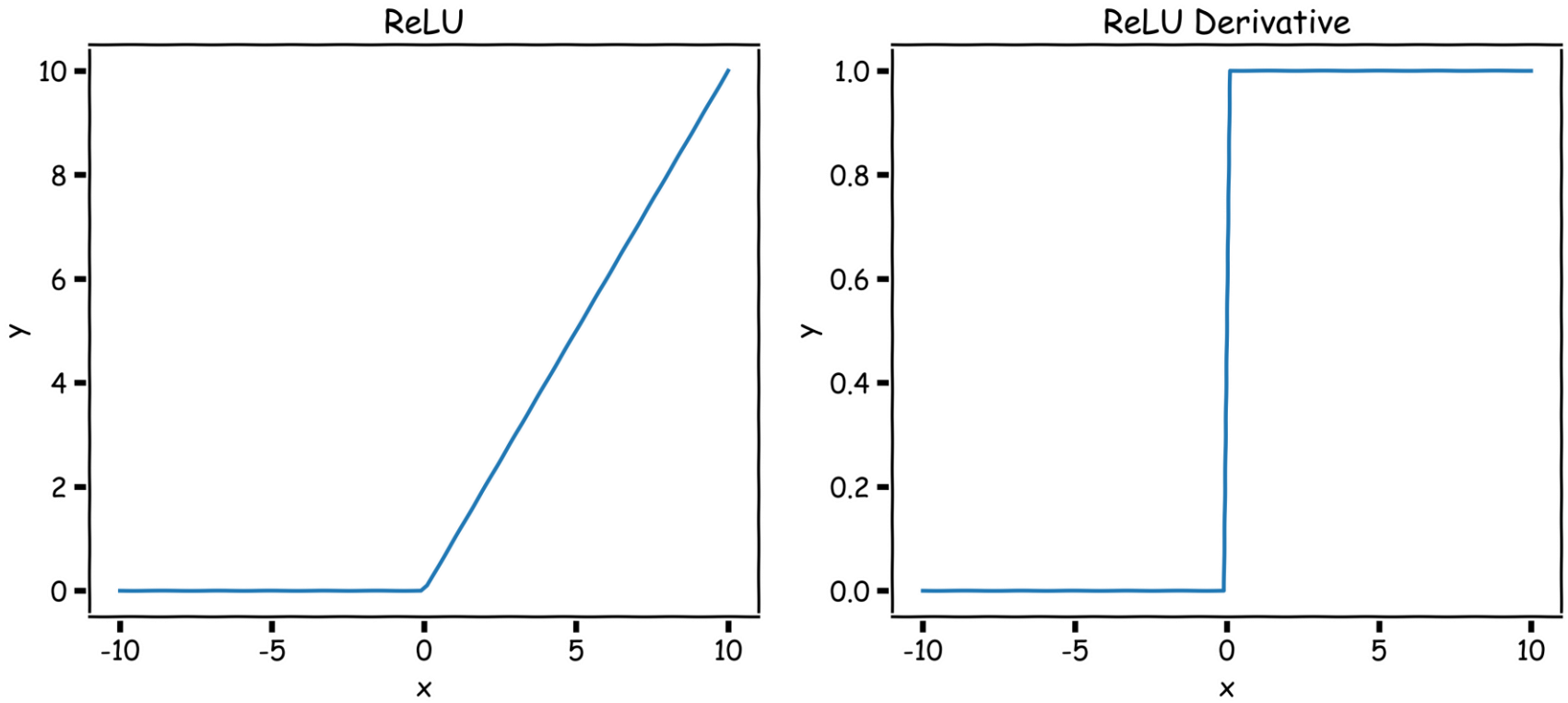


$y' = 1 - y^2$

Same problem of “vanishing gradients” as sigmoid.

Rectified Linear Unit (ReLU)

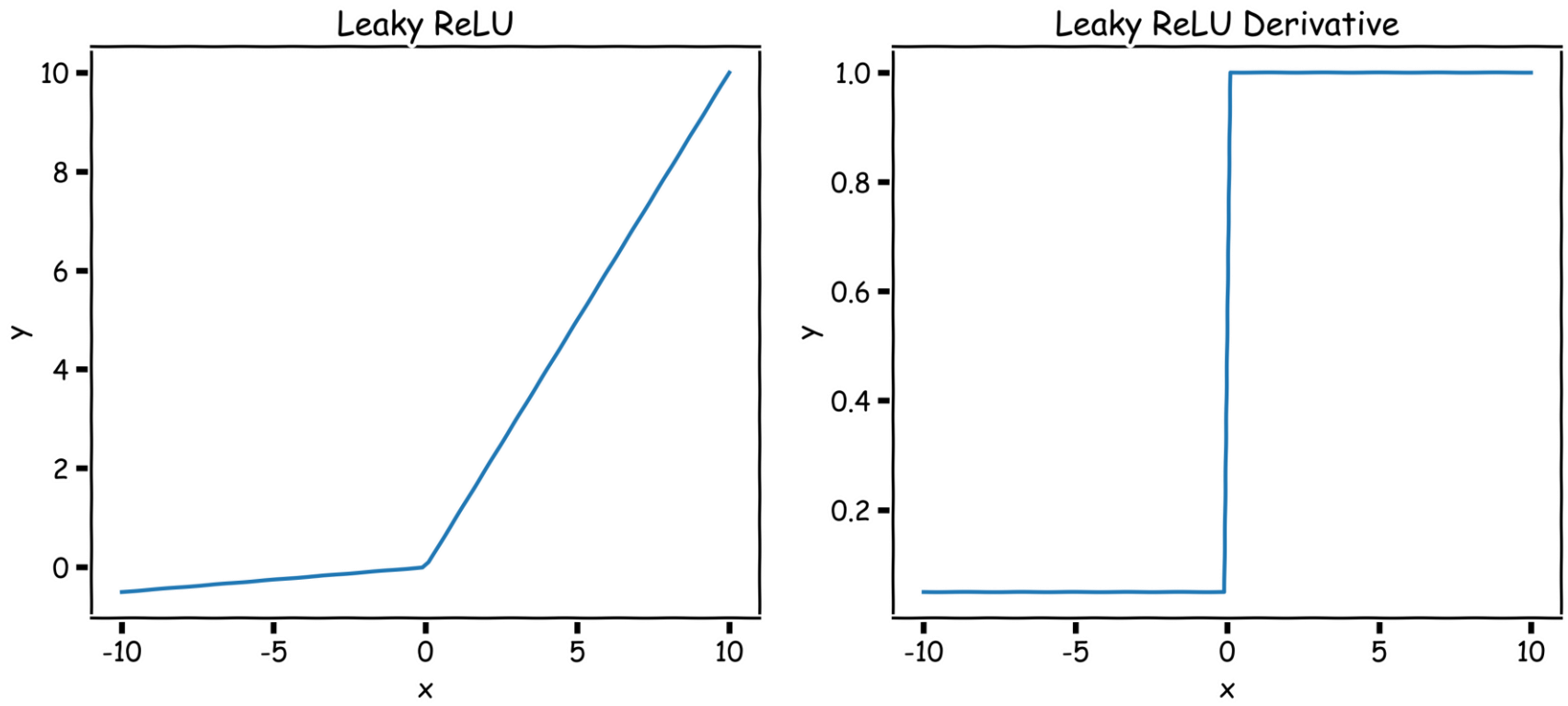
$$y = \max(0, x)$$



No vanishing gradient when x > 0

Leaky ReLU

$$y = \max(0, x) + \alpha \min(0,1)$$
where α takes a small value



- Tries to fix "dying ReLU" problem: derivative is non-zero everywhere.
- Some people report success with this form of activation function, but the results are not always consistent

Opcije za dizajniranje ANN

- Izbor aktivacione funkcije (Activation function)
- Izbor funkcije gubitka (Loss function)
  - Zavisi od tipa problema
    - Binarna klasifikacija – Binary cross entropy
    - Višeklasna klasifikacija – Cross entropy
    - Regresija - MSE
- Izlazni neuroni (Output units)
- Arhitektura

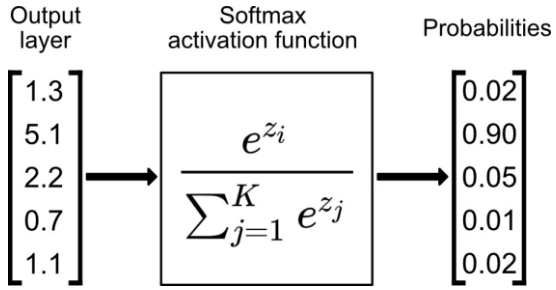
$$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log(\hat{y}_i)$$

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Opcije za dizajniranje ANN

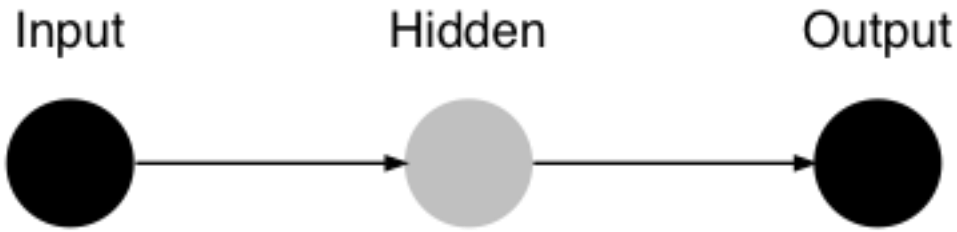
- Izbor aktivacione funkcije (Activation function)
- Izbor funkcije gubitka (Loss function)
- Izlazni neuroni (Output units)
  - Zavisi od tipa problema
    - Binarna klasifikacija – Sigmoid
    - Višeklasna klasifikacija – Softmax
    - Regresija - Linear
- Arhitektura

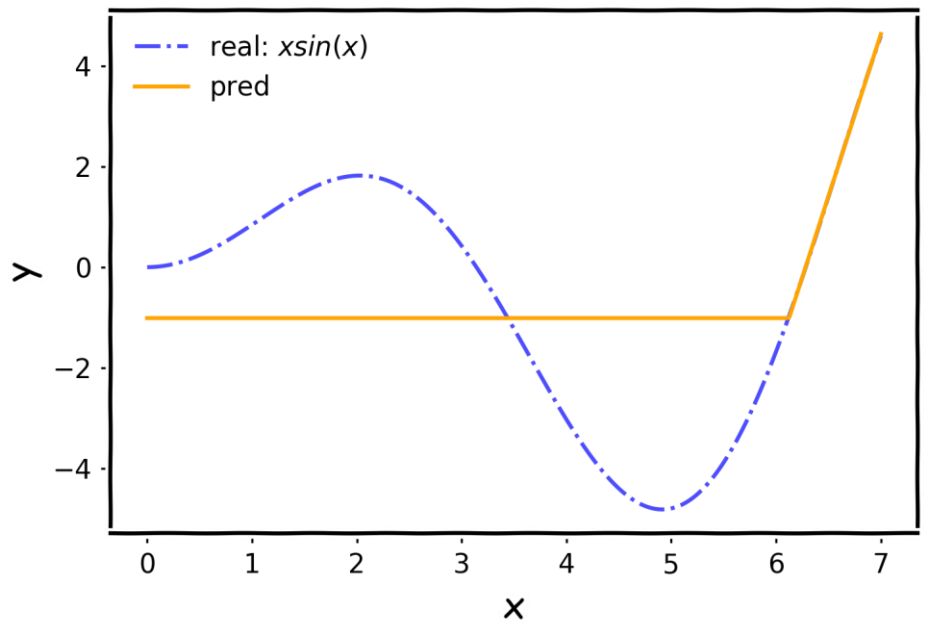
$$y = \frac{e^{z_k}}{\sum_{i=1}^{K} e^{z_i}}$$

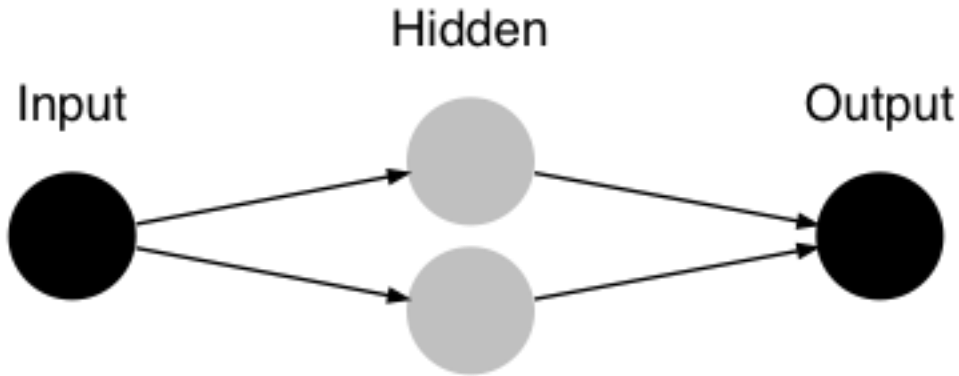


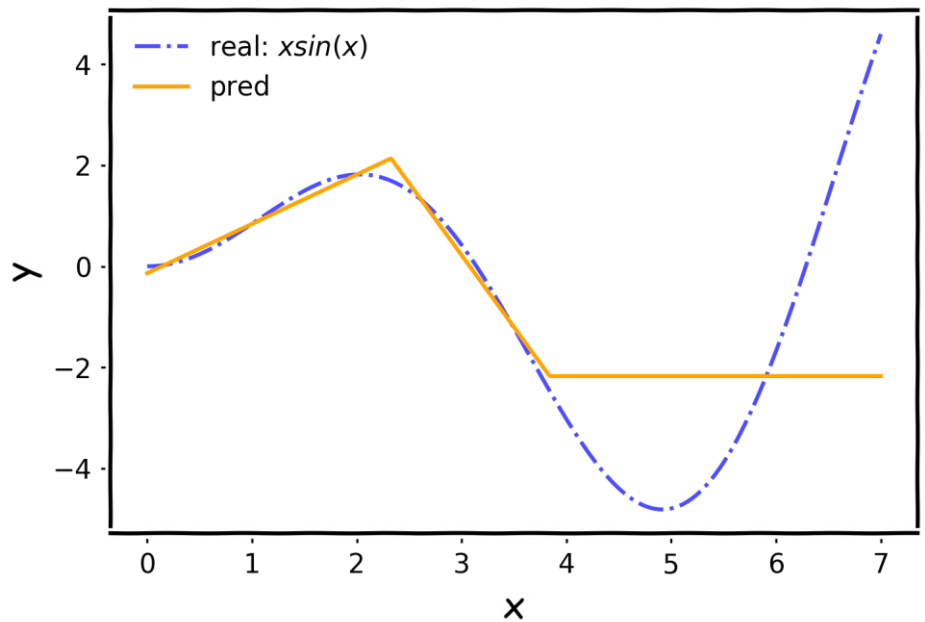
Opcije za dizajniranje ANN

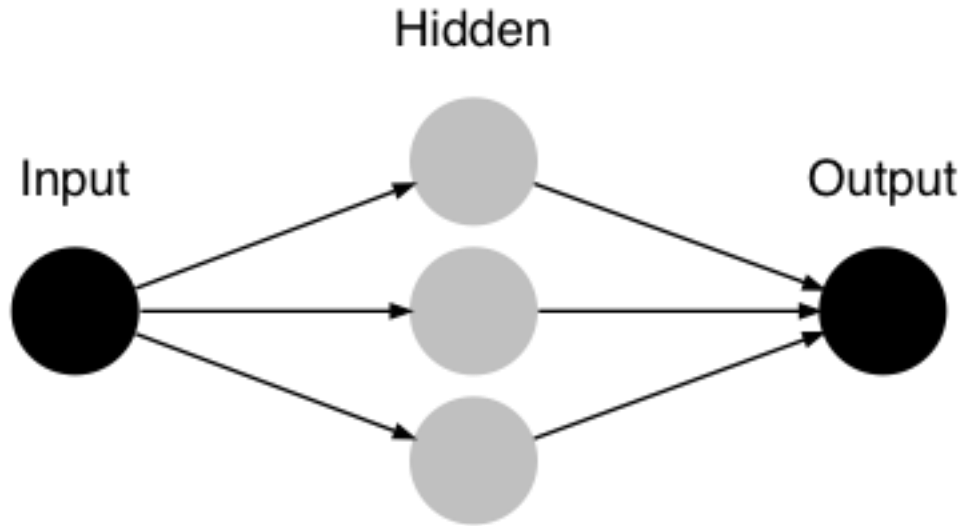
- Izbor aktivacione funkcije (Activation function)
- Izbor funkcije gubitka (Loss function)
- Izlazni neuroni (Output units)
- **Arhitektura**

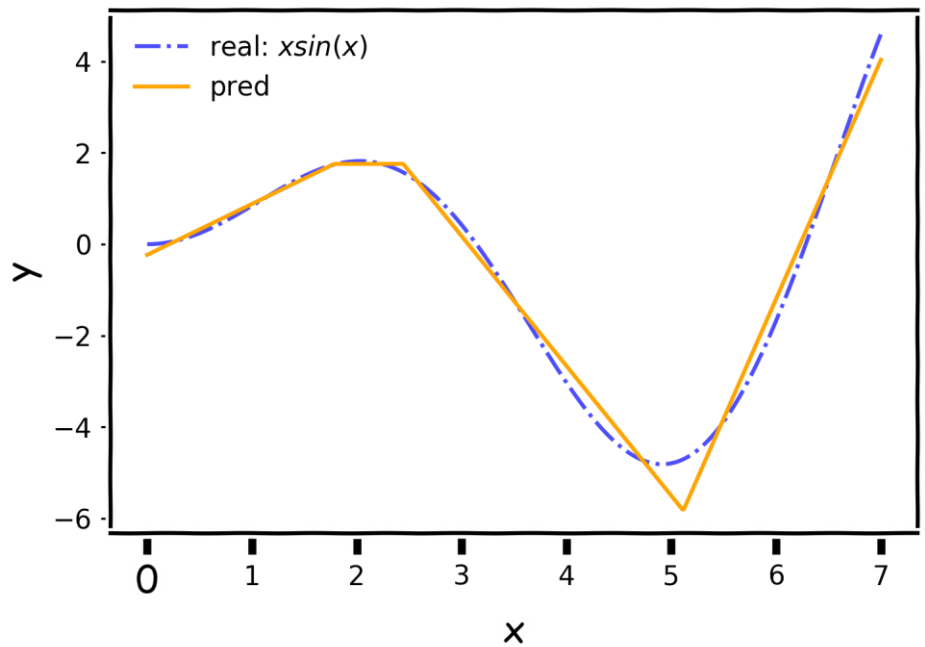

Input
Hidden
Output

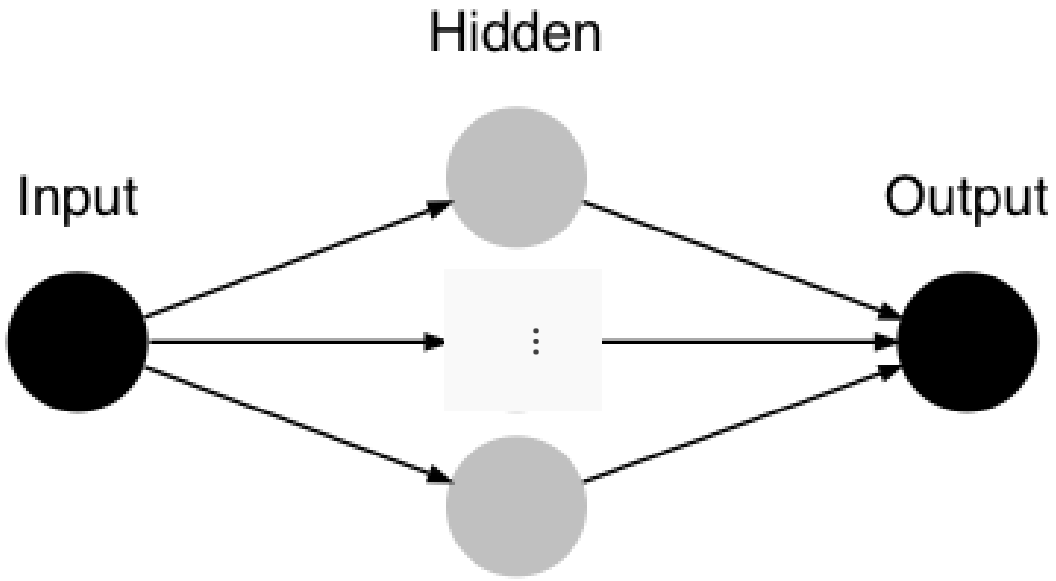

real: xsin(x)
pred
4
2
0
−2
−4
y
0
1
2
3
4
5
6
7
x

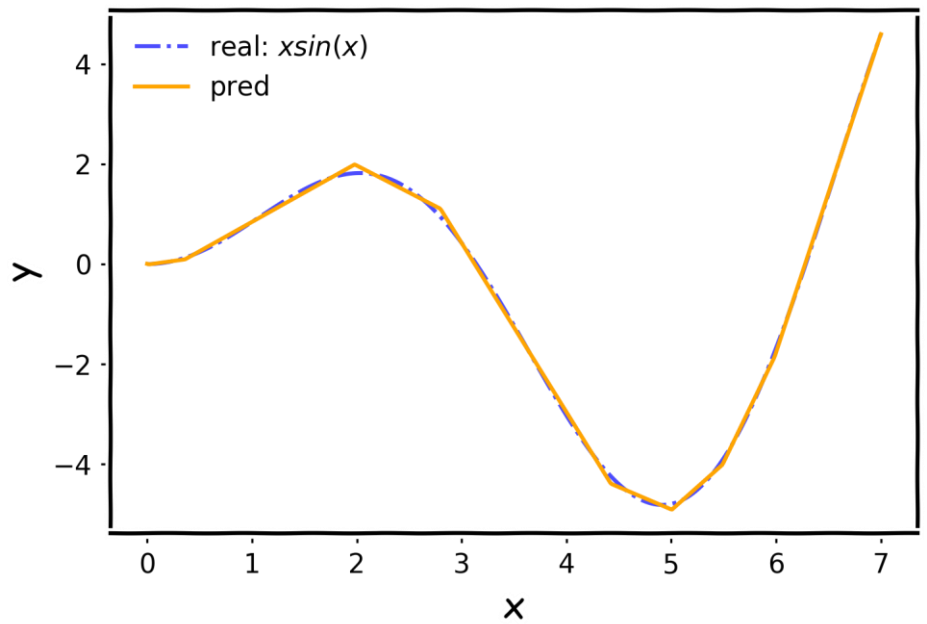

Hidden
Input
Output

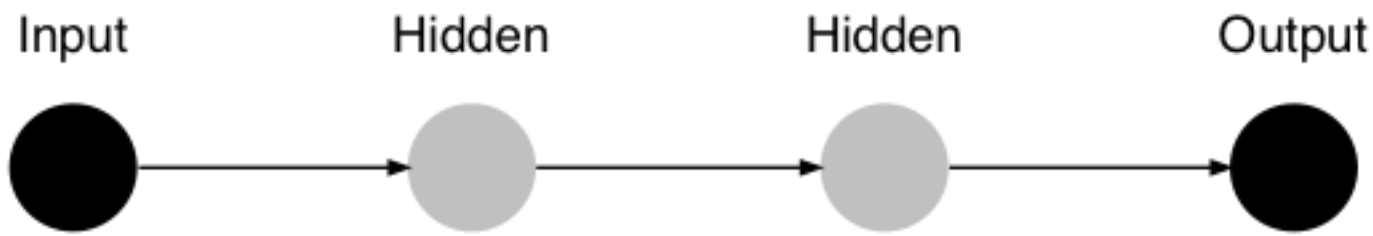

real: xsin(x)
pred
4
2
0
−2
−4
y
0
1
2
3
4
5
6
7
x

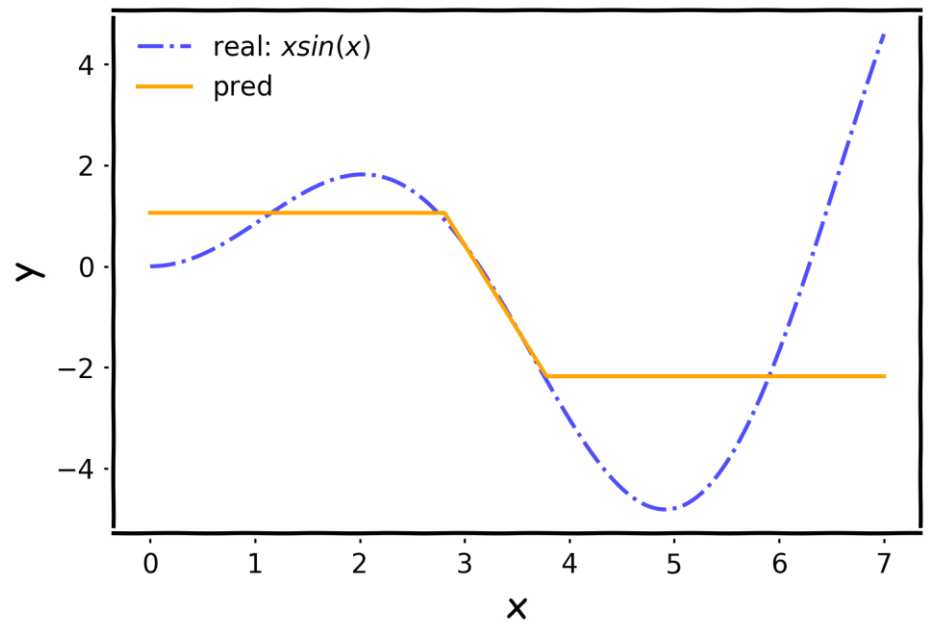

Hidden
Input
Output

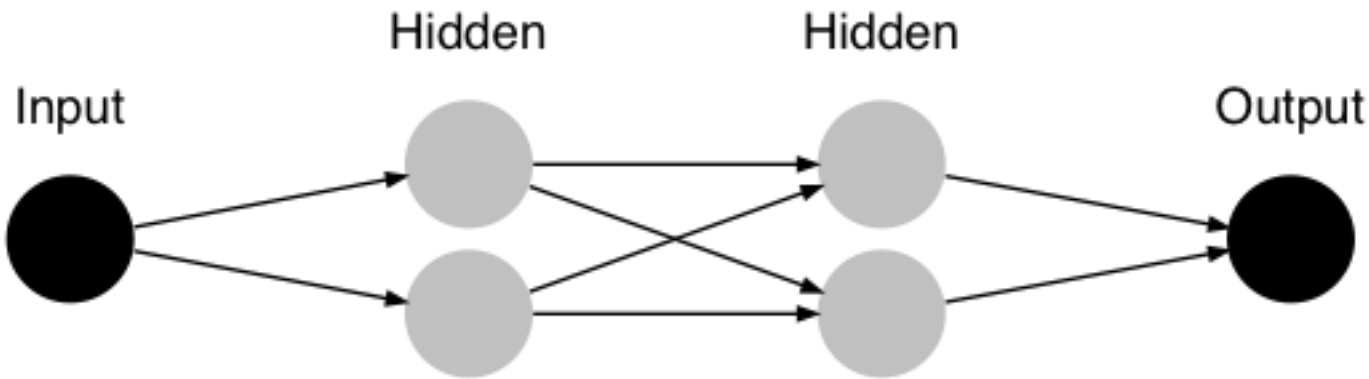

real: xsin(x)
pred
4
2
0
−2
−4
−6
y
0
1
2
3
4
5
6
7
x

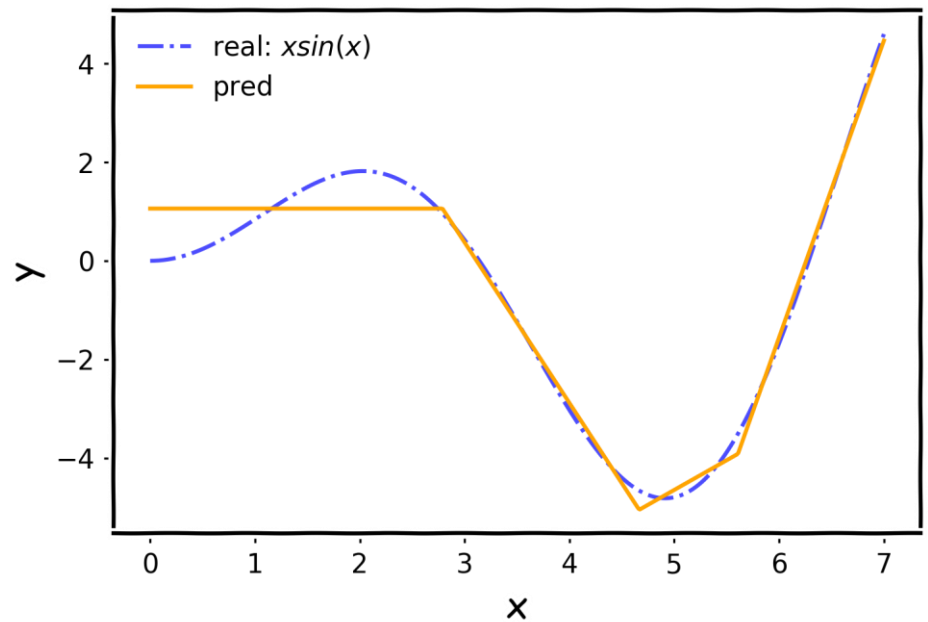

Hidden
Input
Output

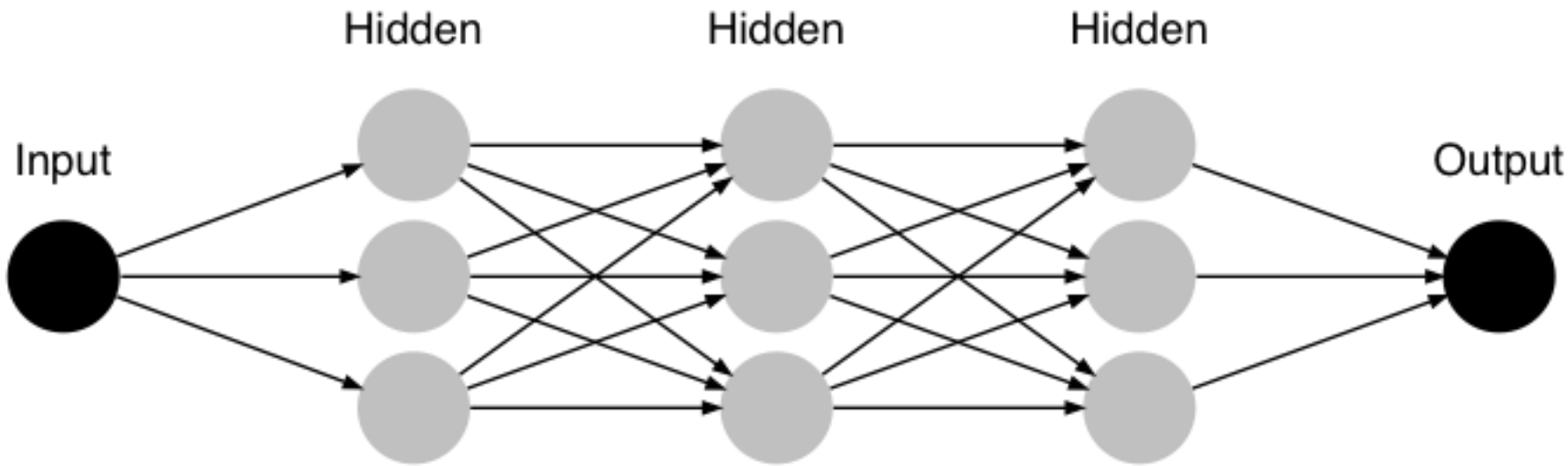

real: xsin(x)
pred
4
2
0
−2
−4
y
0
1
2
3
4
5
6
7
x

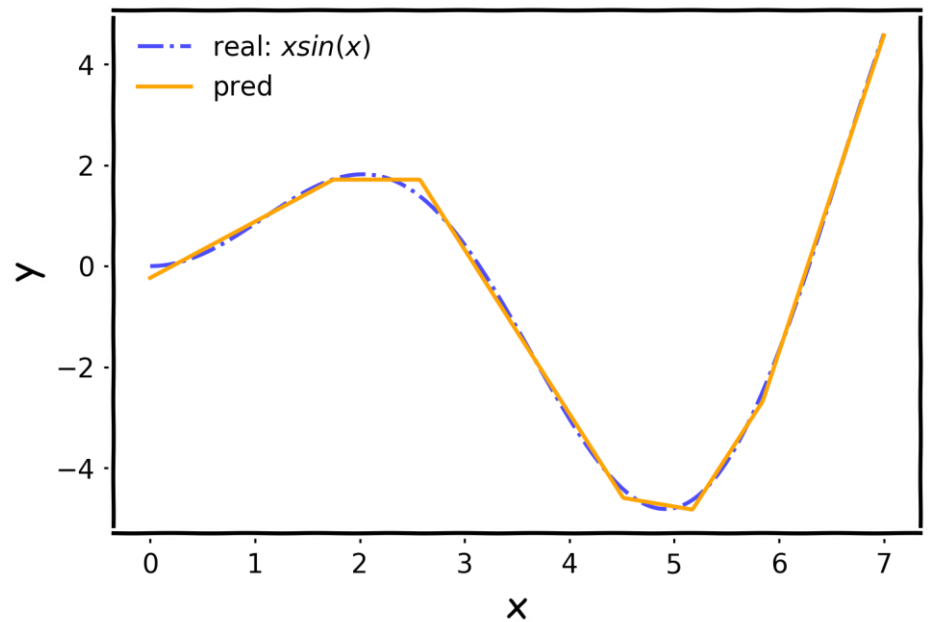

Input
Hidden
Hidden
Output


real: xsin(x)
pred
4
2
0
−2
−4
y
0
1
2
3
4
5
6
7
x


Input
Hidden
Hidden
Output

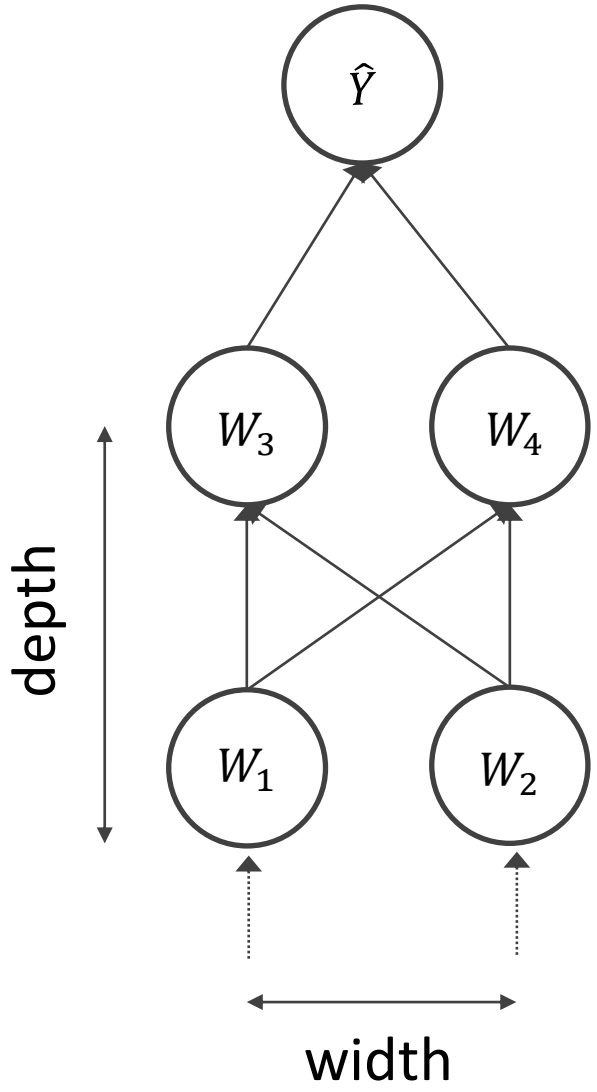

real: xsin(x)
pred
y
x
4
2
0
−2
−4
0
1
2
3
4
5
6
7


Hidden
Hidden
Hidden
Input
Output


real: xsin(x)
pred
4
2
0
−2
−4
y
0
1
2
3
4
5
6
7
x

Universal Approximation

- **Think of a Neural Network as function approximation.**
- $Y = f(x) + \epsilon$
- $Y = \hat{f}(x) + \epsilon$
  - NN: $\Longrightarrow \hat{f}(x)$
- One hidden layer is enough to *represent* an approximation of any function to an arbitrary degree of accuracy
- So why deeper?
  - Shallow net may need (exponentially) more width
  - Shallow net may overfit more



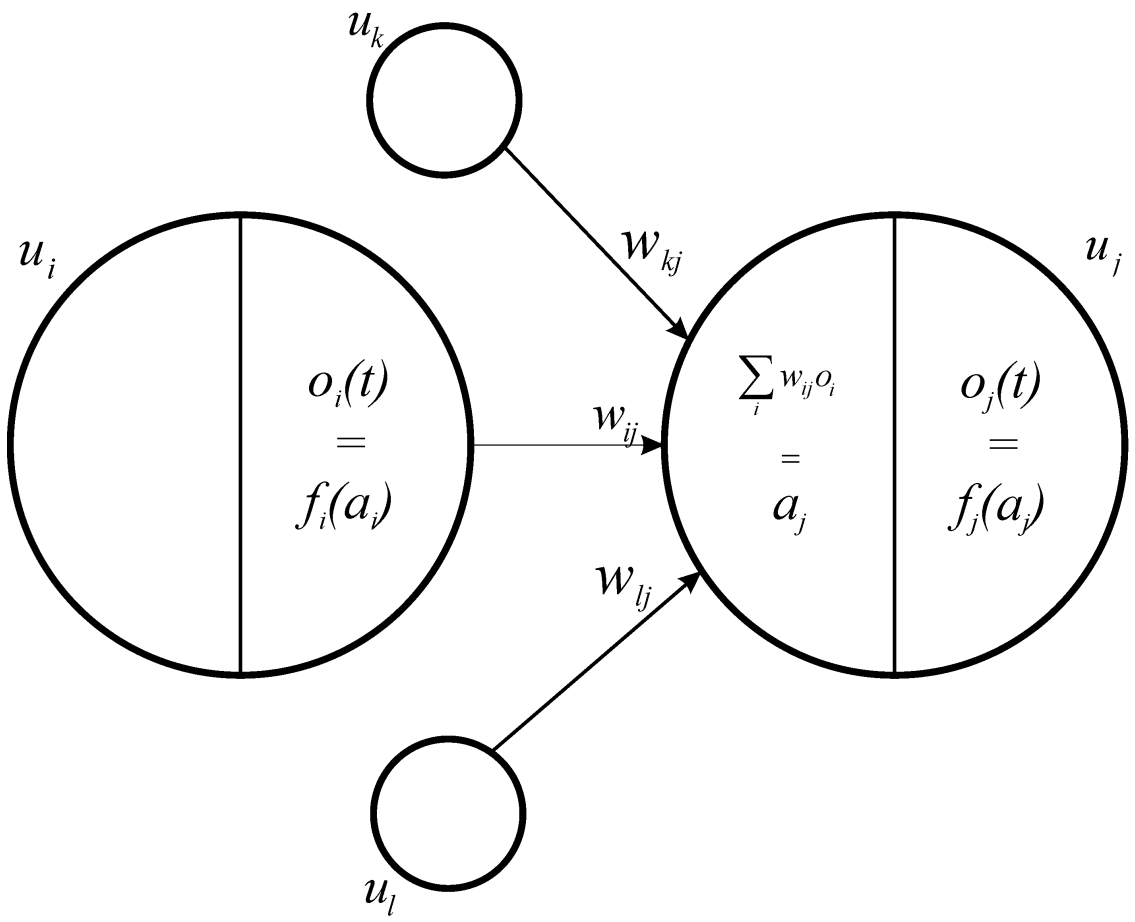
Forward propagation

- Neuron j u skrivenom sloju dobija tešinsku sumu ulaza u mrežu.

$$a_j^{(1)} = \sum_{i=0}^{n} w_{ij}^{(1)} x_i$$

- Izlaz iz neurona u skrivenom sloju

$$o_j^{(1)} = f(a_j^{(1)}) = f\left(\sum_{i=0}^{n} w_{ij}^{(1)} x_i\right)$$



Forward propagation

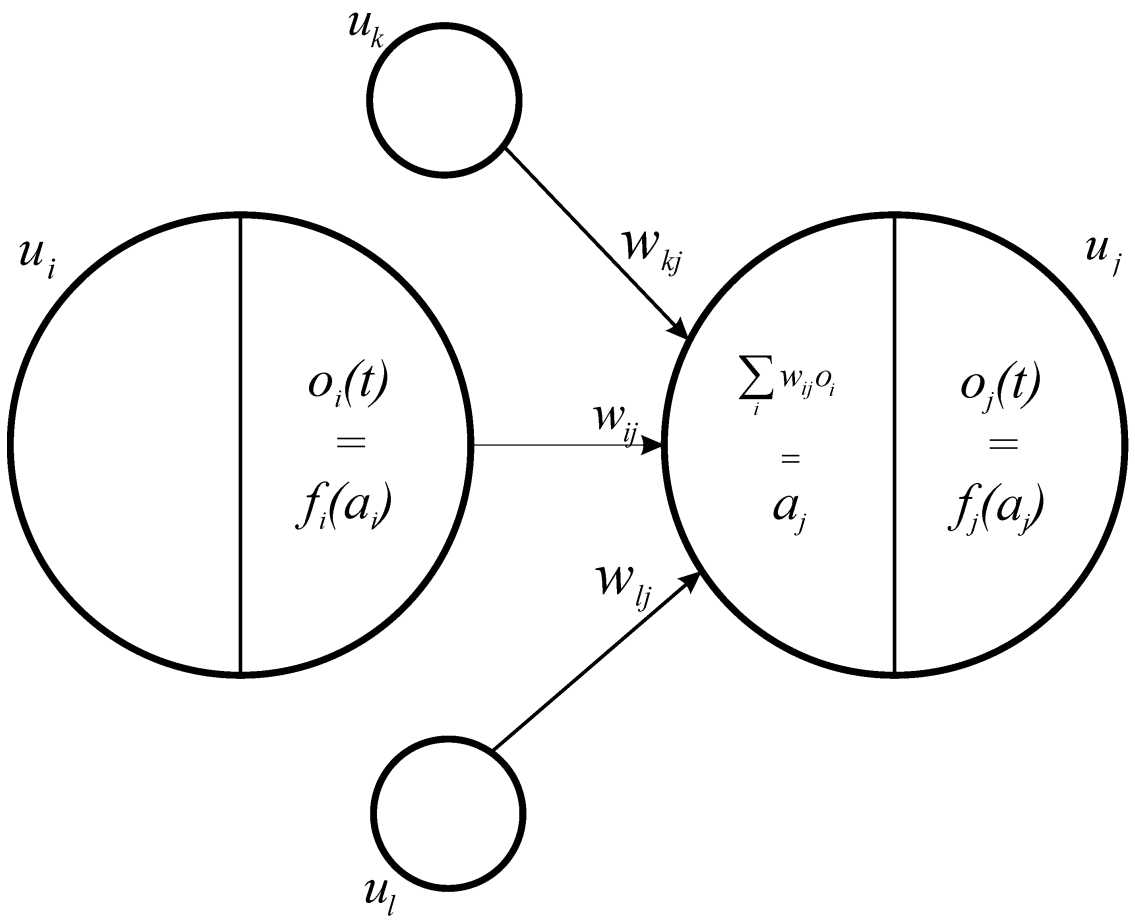
- Ako imamo samo jedan skriveni sloj, neuron *j* u izlaznom sloju dobijana ulaz težinsku sumu izlaza iz svih neurona u skrivenom sloju.

$$a_j^{(2)} = \sum_{i=0}^{n} w_{ij}^{(2)} o_i^{(1)}$$

- Izlaz iz neurona u izlaznom sloju

$$o_j^{(2)} = f(a_j^{(2)}) = f\left(\sum_{i=0}^{n} w_{ij}^{(2)} o_i^{(1)}\right)$$

$$o_j^{(2)} = f\left(\sum_{i=0}^{n} w_{ij}^{(2)} f\left(\sum_{i=0}^{n} w_{ij}^{(1)} x_i\right)\right)$$



Učenje težina

- Mreže uče podešavanjem težina u cilju smanjivanja razlike između izlaza koji daje mreža o i očekivanog izlaza y.

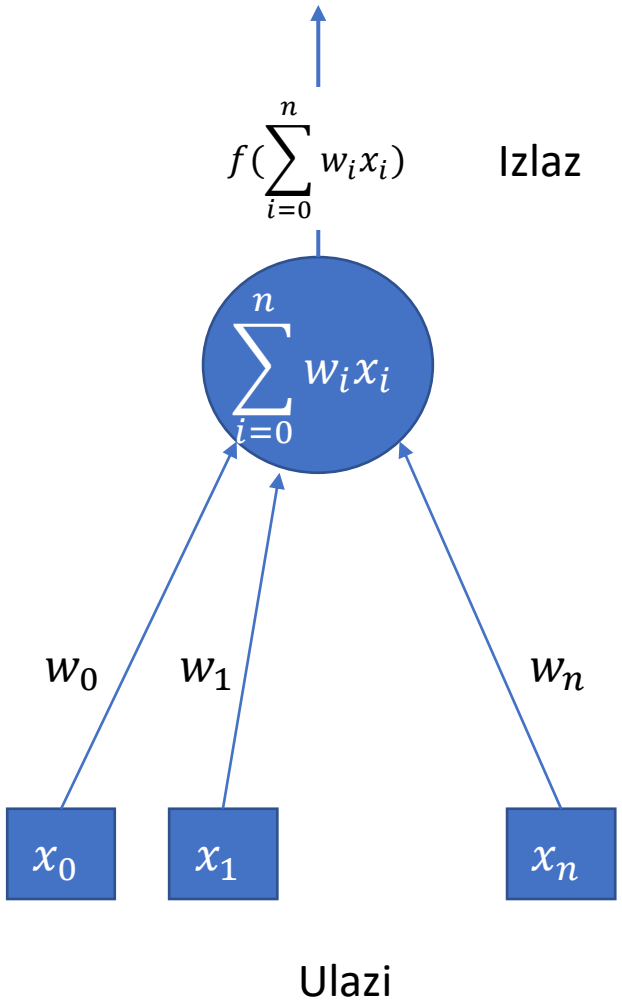
- Minimizacija razlike?

- Gradijentni spust.

Minimizacija greške

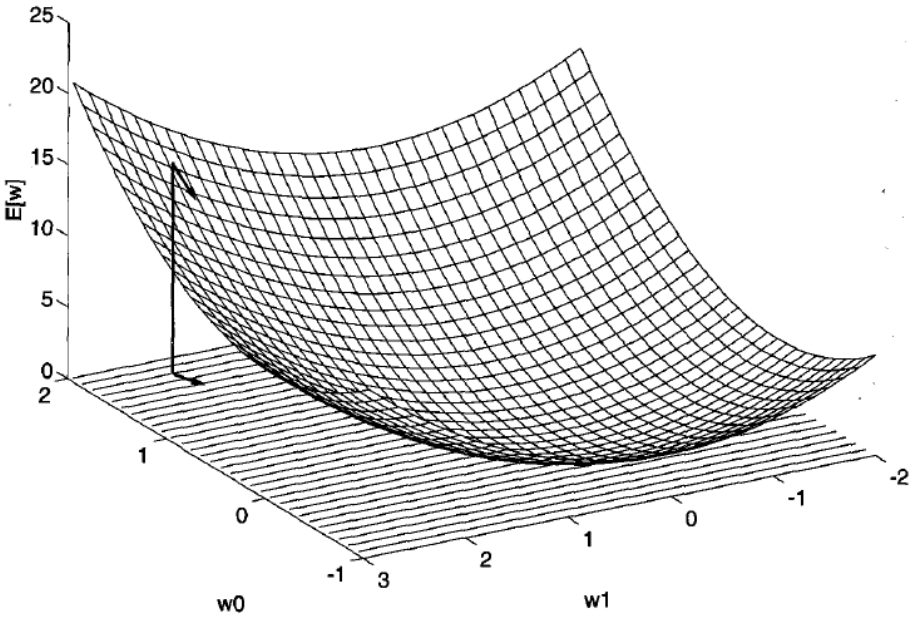
- Krenimo od mreže koja nema skriveni sloj i ima samo jedan izlaz. Neka radi jednostavnosti imamo samo jedan trening primer $(\vec{x}, y)$
- Dobijeni izlaz iz mreže o je neka funkcija $f(a)$, gde je a težinska suma ulaza u neuron $a = \sum_{i=0}^{n} w_i x_i$.
- Greška izlaza je razlika između dobijenog i željenog izlaza.
- Uobičajeni oblik funkcije greške na jednom izlaznom neuronu za jedan trening primer:

$$E = \frac{1}{2}(y - o)^2$$



Minimizacija greške

- Grešku mreže možemo smanjiti menjanjem težina veza.
- Pretražujemo prostor težina u potrazi za takvom kombinacijom težina za koju je greška mreže jednaka ili jako bliska nuli.
- Gradijentnom metodom određuje se vektor težina koji minimizuje grešku tako što se započinje s proizvoljnim početnim težinama, a zatim one menjaju u koracima. Na svakom koraku, vektor težine se menja u pravcu koji predstavlja najstrmiji silazak po površi greške. Ovaj proces se nastavlja sve dok se ne dostigne globalna minimalna greška.
- Da bismo odredili pravac najstrmijeg spusta, potrebno je da odredimo parcijalni izvod greške po svakoj od težina $\nabla E = (\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n})$. Smer najstrmijeg spusta je suprotan od smera vektora ∇E



Promena težina

- Svaka težina će biti promenjena za vrednost gradijenta i to u meri određenoj stopom učenja:

$$w_j \leftarrow w_j + \Delta w_j$$

Gde je

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$$

- η je pozitivna konstanta (learning rate) koja određuje veličinu koraka učenja

Izvod greške izlaza po težini veze

$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial o}\frac{\partial o}{\partial w_j} = \frac{\partial E}{\partial o}\frac{\partial o}{\partial a}\frac{\partial a}{\partial w_j}$$

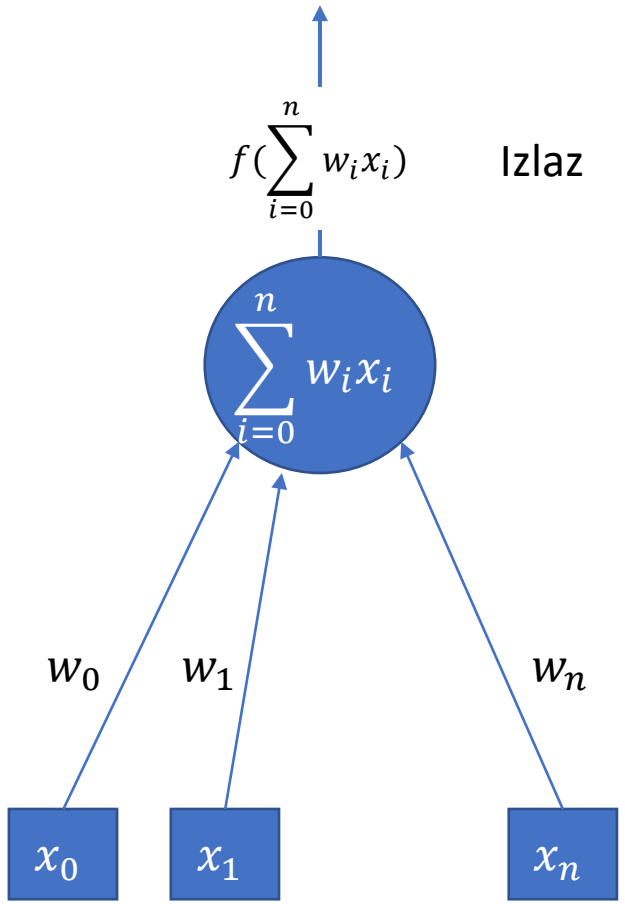
$$\frac{\partial a}{\partial w_j} = \frac{\partial}{\partial w_j}\left[\sum_{i=0}^{n} w_i x_i\right] = x_j \qquad (1)$$

$$\frac{\partial E}{\partial o} = \frac{\partial}{\partial o}\left[\frac{1}{2}(y-o)^2\right] = (o-y) \qquad (2)$$

(1)

(2)

$$\frac{\partial E}{\partial w_j} = (o-y)\cdot x_j \cdot o' \qquad (**)$$



Težina se menja po pravilu: $\Delta w_j = -\eta(o-y)\cdot x_j \cdot o'$

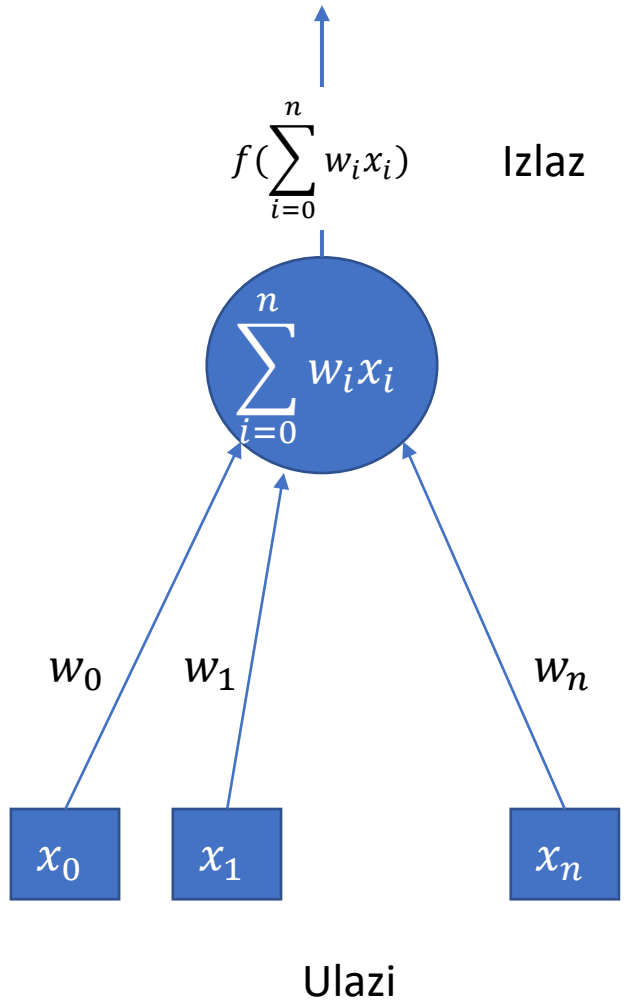
o' je izvod konkretne aktivacione funkcije po ukupnom ulazu u neuron.

Kada imamo L trening primera:

$$E = \frac{1}{2}\sum_{l \in L}(y_l - o_l)^2$$

$$\frac{\partial E}{\partial w_j} = \sum_{l \in L}(o_l - y_l)\, x_j^l o_l'$$

Težina se menja po pravilu: $\Delta w_j = -\eta \sum_{l \in L}(o_l - y_l)\, x_j^l o_l'$



Algoritam korekcije težina korišćenjem metode gradijentnog spusta

- Odabrati slučajne vrednosti težina.
- Izračunati izlaz o_l iz mreže za svaki ulaz $\overrightarrow{\mathrm{x}_l}$ iz trening primera
- Izračunati Δw_j po pravilu $\Delta w_j = -\eta \sum_{l \in L} (o_l - y_l)\, x_j^l o_l'$, za svako j
- Uvećati svaku težinu za odgovarajuće Δw_j.
- Ponoviti ovaj postupak do postizanja željene preciznosti mreže.

Linearni neuron - Pravilo za promenu težina veza

- Polazimo od opšteg pravila: $\Delta w_j = -\eta \sum_{l \in L} (o_l - y_l)\, x_j^l o_l'$
- $o(a) = a = \sum_{i=1}^{n} w_i x_i$
- $\frac{do}{da} = 1$

$$\Delta w_j = -\eta \sum_{l \in L} (o_l - y_l)\, x_j^l$$

Nedostaci primene gradijenta na učenje neuronskih mreža

- Konvergencija ka lokalnom minimumu može teći jako sporo
- Ako je površ greške sa višestrukim lokalnim minimumima, nema garancije da će procedura pronaći globalni minimum.
- Uobičajena varijanta gradijentnog spusta kojom se ovi nedostaci rešavaju je stohastički gradijentni spust.

Stohastički gradijentni spust

- Kod običnog gradijentnog spusta, promena težina se vrši nakon prolaska kroz sve trening primere.

- Stohastički gradijentni spust podrazumeva inkrementalnu promenu težina, nakon svakog izračuvanja greške za određeni trening primer.

- Posle propuštanja trening primera l kroz mrežu, korigujemo težine:

$$w_j \leftarrow w_j - \eta(o_l - y_l)x_j^l$$

Delta pravilo

A toy example to illustrate the iterative method

- Each day you get lunch at the cafeteria.
  - Your diet consists of fish, chips, and ketchup.
  - You get several portions of each.
- The cashier only tells you the total price of the meal
  - After several days, you should be able to figure out the price of each portion.
- The iterative approach: Start with random guesses for the prices and then adjust them to get a better fit to the observed prices of whole meals.

Solving the equations iteratively

- Each meal price gives a linear constraint on the prices of the portions:

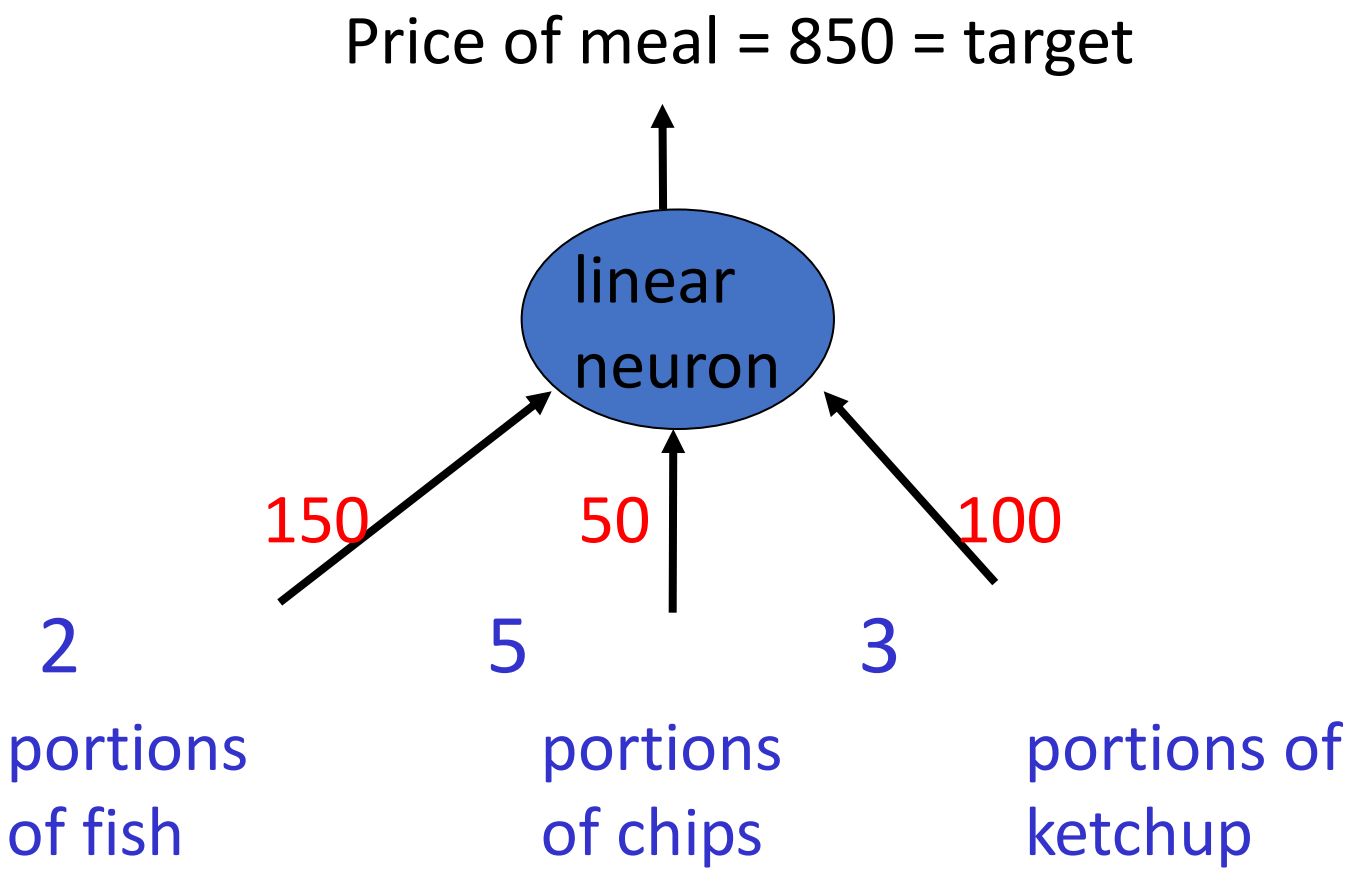
$$price = x_{fish} w_{fish} + x_{chips} w_{chips} + x_{ketchup} w_{ketchup}$$

- The prices of the portions are like the weights in of a linear neuron.

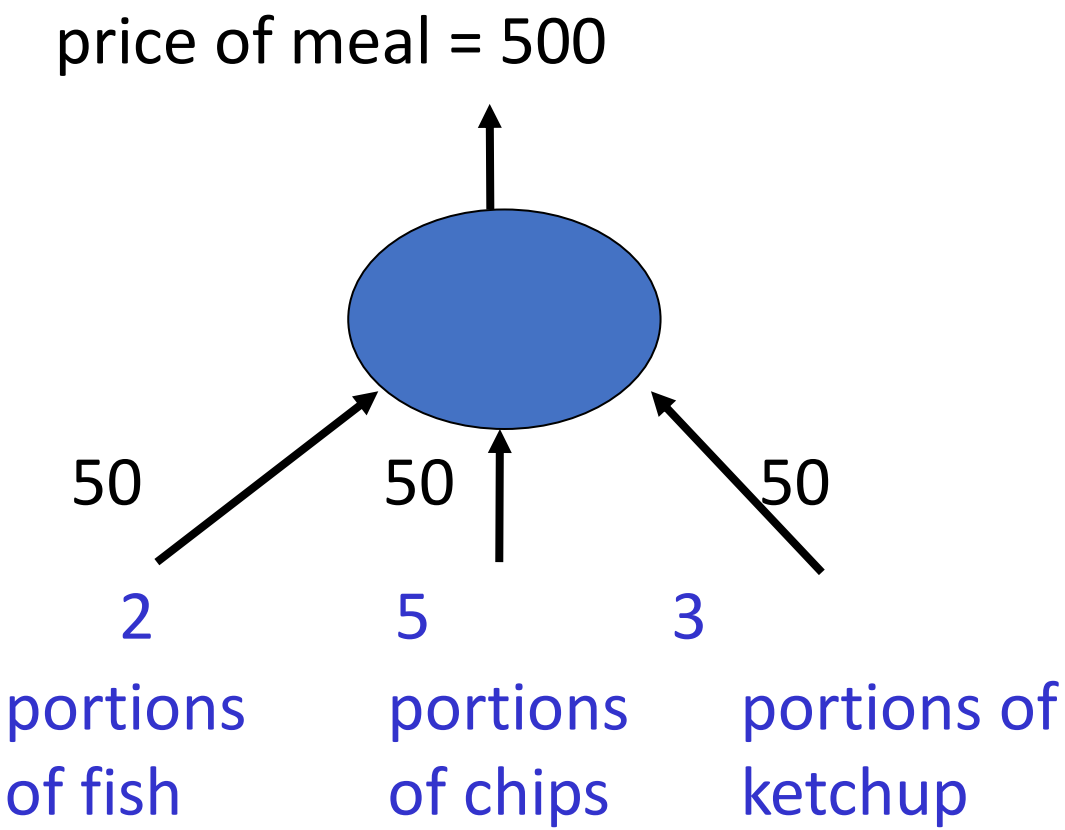
$$\mathbf{w} = (w_{fish}, w_{chips}, w_{ketchup})$$

- We will start with guesses for the weights and then adjust the guesses slightly to give a better fit to the prices given by the cashier.

The true weights used by the cashier



Primena delta pravila za učenje

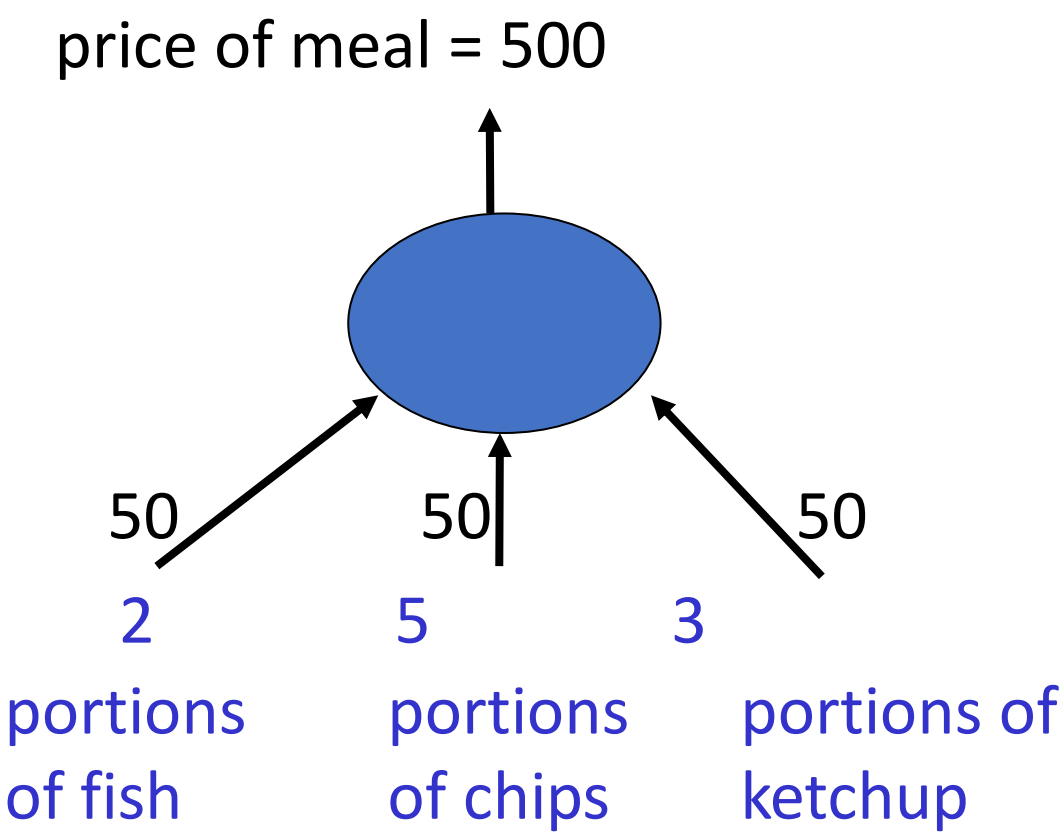


- Delta pravilo za učenje:

$$\Delta w_i = \eta \cdot x_i \cdot (y - \mathrm{o})$$

- Δw_i promena težine i-te veze
- η stopa učenja
- x_i vrednost i-tog ulaza u neuron (broj porcija)
- y očekivana vrednost izlaza (850)
- o dobijena vrednost izlaza (500)

A model of the cashier with arbitrary initial weights



- With a learning rate η of $\frac{1}{35}$, the weight changes are +20, +50, +30
- This gives new weights of 70, 100, 80.
  - Notice that the weight for chips got worse!
  - Ne poboljšavaju se težine, nego se smanjuje razlika između očekivanog i dobijenog izlaza.

Sigmoid neuron - Pravilo za promenu težina veza

- Polazimo od opšteg pravila: $\Delta w_j = -\eta \sum_{l \in L} (o_l - y_l)\, x_j^l o_l'$
- $o(a) = \frac{1}{1+e^{-a}}$
- Treba odrediti $\frac{do}{da}$

$$f(z) = (1 + e^{-z})^{-1}$$

$$\frac{do}{da} = -(1 + e^{-a})^{-2}(-1)e^{-a} = \frac{e^{-a}}{(1+e^{-a})^2} = \frac{1}{(1+e^{-a})}\frac{e^{-a}}{(1+e^{-a})} =$$

$$= \frac{1}{(1 + e^{-a})}\frac{1 + e^{-a} - 1}{(1 + e^{-a})} = \frac{1}{(1 + e^{-a})}\left(1 - \frac{1}{(1 + e^{-a})}\right)$$

$$\boldsymbol{\frac{do}{da} = o(1 - o)}$$

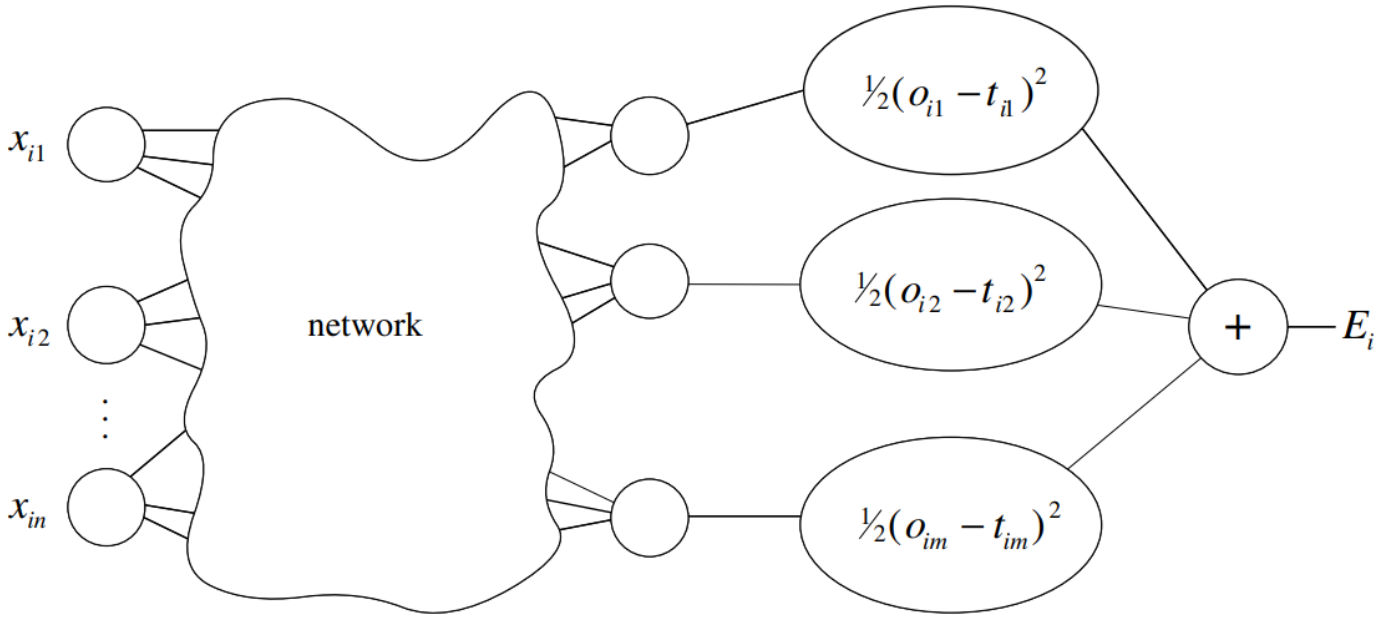
Sigmoid neuron - Pravilo za promenu težina veza

$$\Delta w_j = -\eta \sum_{l \in L} (o_l - y_l)\, x_j^l o_l'$$

$$\frac{\boldsymbol{do}}{\boldsymbol{da}} = \boldsymbol{o}(\boldsymbol{1} - \boldsymbol{o})$$

$$\Delta w_j = -\eta \sum_{l \in L} (o_l - y_l)\, x_j^l o_l (1 - o_l)$$

Mreže sa više slojeva i algoritam propagacije unazad



- Mreže sa više slojeva, koje koriste sigmoidnu aktivacionu funkciju, obučavaju se algoritmom propagacije unazad – **backpropagation algorithm**.

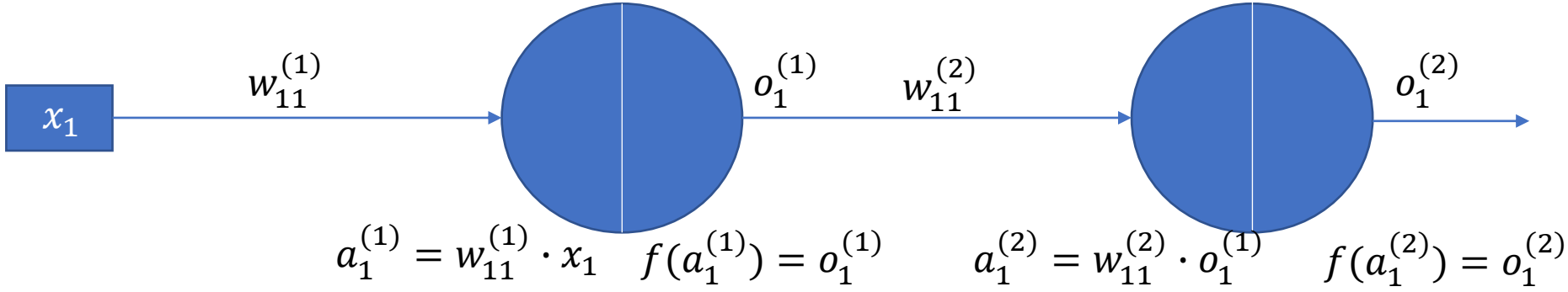
- Algoritam koristi gradijentni spust za minimizaciju kvadrata razlike izlaza koji daje mreža i očekivanog izlaza.

- Ako imamo skup od L trening primera, onda ćemo grešku mreže računati po formuli:

$$E = \frac{1}{2}\sum_{l \in L}(y_l - o_l)^2$$

Algoritam propagacije unazad

- Krenimo od najjednostavnijeg primera mreže koja ima 1 ulaz, 1 neuron u skrivenom sloju i 1 izlaz.



$w_{ij}^{(k)}$- težina veze od neurona i u sloju $k-1$ ka neuronu j u sloju k

$a_j^{(k)}$- ukupna aktivacija neurona j u sloju k

$o_j^{(k)}$- izlaz iz neurona j u sloju k

$f(a_j^{(k)}) = o_j^{(k)}$– vrednost aktivacione f-je neurona j u sloju k

Računanje promene težine za izlazni sloj

$$E = \frac{1}{2}\left(y - o_1^{(2)}\right)^2$$

$$\frac{\partial E}{\partial w_{11}^{(2)}} = \frac{\partial E}{\partial o_1^{(2)}}\frac{\partial o_1^{(2)}}{\partial w_{11}^{(2)}} = \frac{\partial E}{\partial o_1^{(2)}}\frac{\partial o_1^{(2)}}{\partial a_1^{(2)}}\frac{\partial a_1^{(2)}}{\partial w_{11}^{(2)}}$$

$$\frac{\partial E}{\partial w_{11}^{(2)}} = \underbrace{o_1^{(2)}(1 - o_1^{(2)})(o_1^{(2)} - \mathrm{y})}_{\text{Backpropagation error izlaznog sloja}} o_1^{(1)}$$

$$\Delta w_{11}^{(2)} = -\eta o_1^{(2)}(1 - o_1^{(2)})(o_1^{(2)} - \mathrm{y})o_1^{(1)}$$

Računanje promene težine za skriveni sloj

$$\frac{\partial E}{\partial w_{11}^{(1)}} = \frac{\partial E}{\partial o_1^{(2)}} \frac{\partial o_1^{(2)}}{\partial w_{11}^{(1)}} = \frac{\partial E}{\partial o_1^{(2)}} \frac{\partial o_1^{(2)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial w_{11}^{(1)}} = \frac{\partial E}{\partial o_1^{(2)}} \frac{\partial o_1^{(2)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial o_1^{(1)}} \frac{\partial o_1^{(1)}}{\partial a_1^{(1)}} \frac{\partial a_1^{(1)}}{\partial w_{11}^{(1)}}$$

$$\frac{\partial E}{\partial w_{11}^{(1)}} = o_1^{(2)} \left(1 - o_1^{(2)}\right) \left(o_1^{(2)} - \mathrm{y}\right) o_1^{(1)} \left(1 - o_1^{(1)}\right) x_1 \, w_{11}^{(2)}$$

Backpropagation error izlaznog sloja

$$\Delta w_{11}^{(1)} = -\eta o_1^{(2)} (1 - o_1^{(2)}) (o_1^{(2)} - \mathrm{y}) o_1^{(1)} \left(1 - o_1^{(1)}\right) x_1 \, w_{11}^{(2)}$$

Backpropagation algoritam

Ovo je algoritam zasnovan na stohastičkom gradijentnom spustu i to za mrežu koja ima samo jedan skriveni sloj.

Detaljnije objašnjenje algoritma naći ćete u dokumentu BPA.pdf

Ovo je algoritam koji smo primenili na času za rešavanje konkretnog primera:

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, where $\vec{x}$ is the vector of network input values, and $\vec{t}$ is the vector of target network output values.

η is the learning rate (e.g., .05). n_{in} is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.

The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between $-.05$ and .05).
- Until the termination condition is met, Do
  - For each $\langle \vec{x}, \vec{t} \rangle$ in $training\_examples$, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{T4.3}$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \tag{T4.4}$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

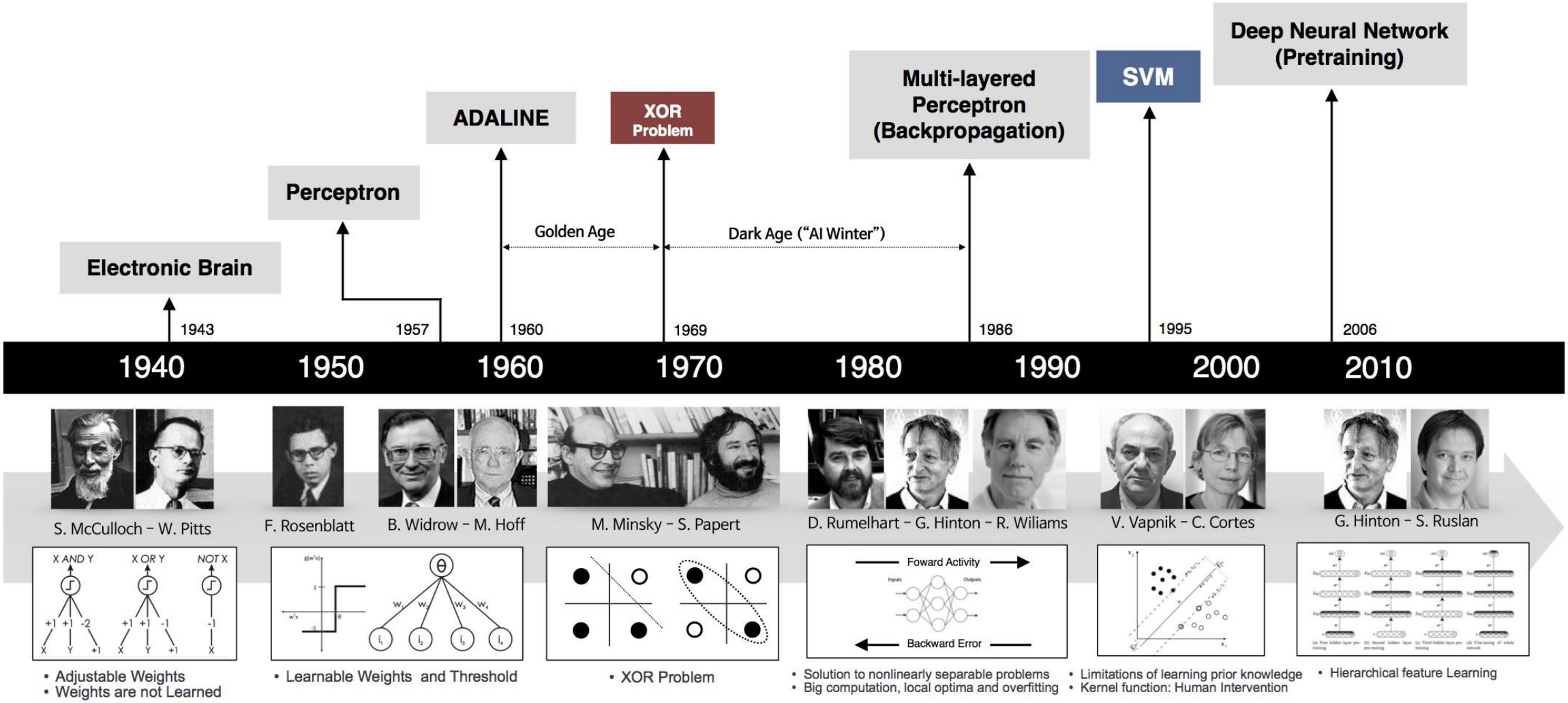
where

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji} \tag{T4.5}$$

Dodatni izvor

- Kompletno izvođenje Backpropagation algoritma za proizvoljan broj skrivenih slojeva:

https://brilliant.org/wiki/backpropagation/


Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
XOR Problem
ADALINE
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

Geoffrey Hinton,
The persistent

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

Neural Information Processing Systems, 2012

Uspeh Hintonove duboke konvolucione neuronske mreže

Neuspešne klasifikacije