

# Deep Learning

2023-11-22 EM

Ämne: Perceptron, Fully connected neural networks, Backprop,  
Gradient descent

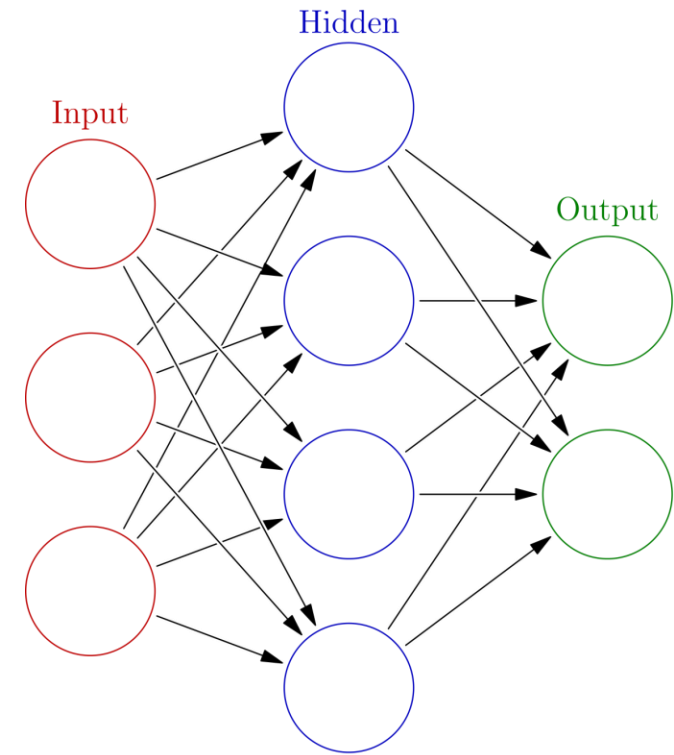
# Agenda

- Neurala nätverk
  - Perceptroner, vikter, bias
  - Gömda lager, input, output
  - Aktiveringsfunktion på hög nivå
  - Loss funktioner
  - Inferens med nätverk.
- Bakåtpropagering
  - Repetition av grundläggande koncept
  - Learning rate
  - Gradient descent
- XOR-exempel



# Neurala nätverk

- Ett enkelt fully connected nerualt närverk:
- Består av 3 typer av lager.
- Varje lager består av perceptroner.
- Kan bestå av fler gömda lager.



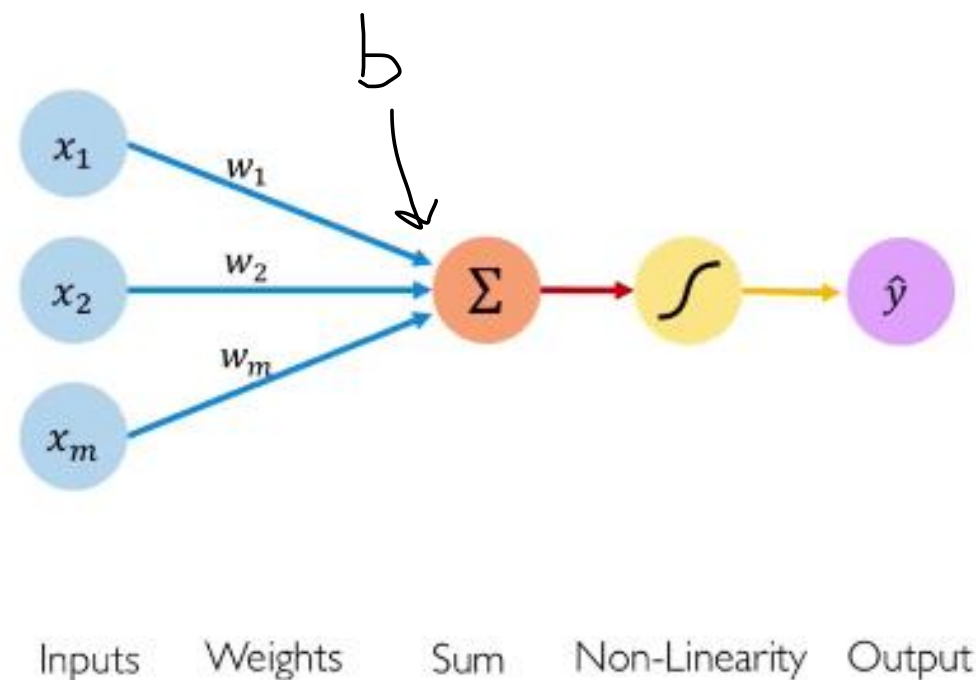
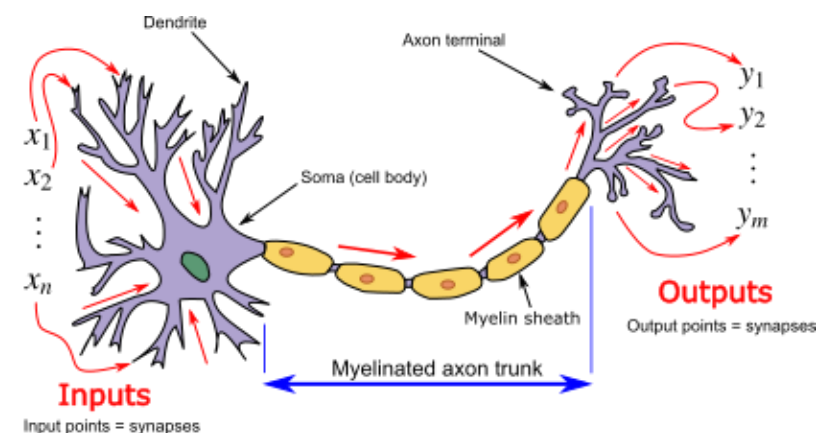
# Perceptron

- Varje perceptron tar flera input och producerar 1 output.

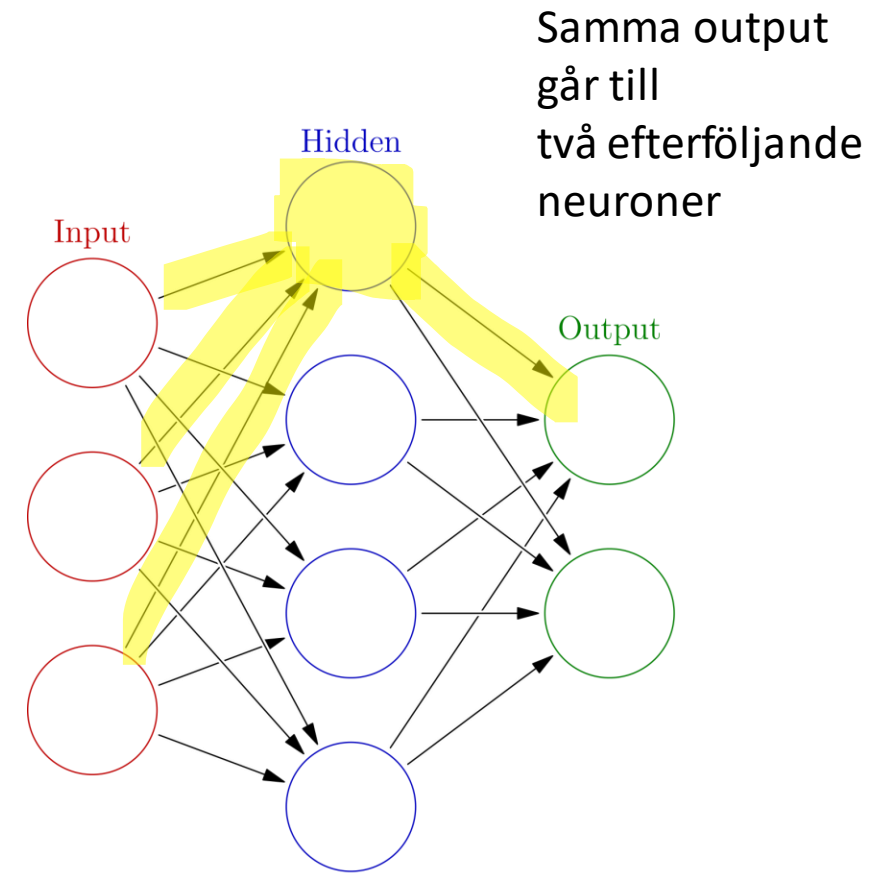
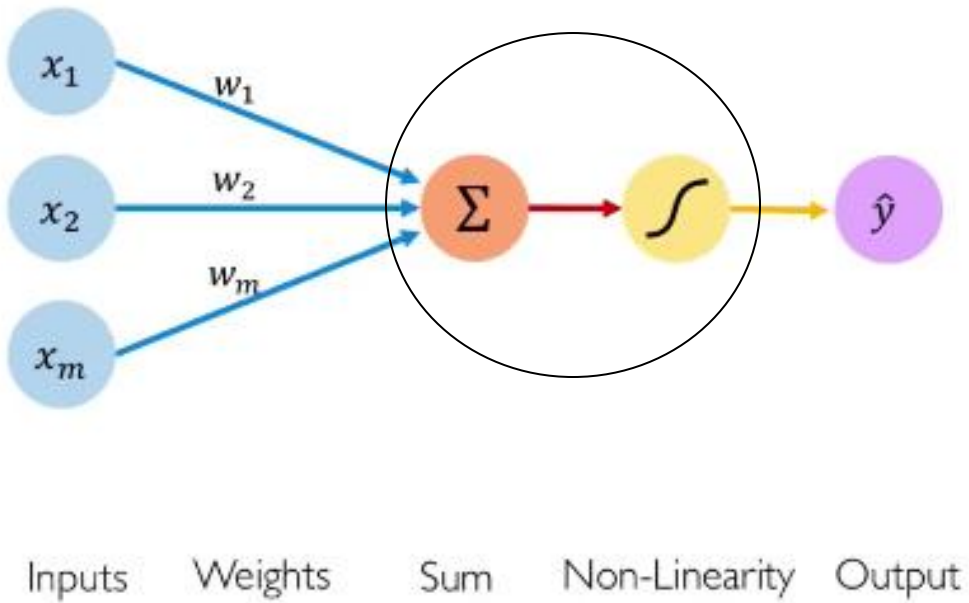
$$f\left(\sum_{i=1}^m w_i x_i + b\right) = \hat{y}$$

Funktionen  $f$  kallas  
*aktiveringsfunktion*

Inspirerad av neuroner i hjärnan - dock inte så nära längre.

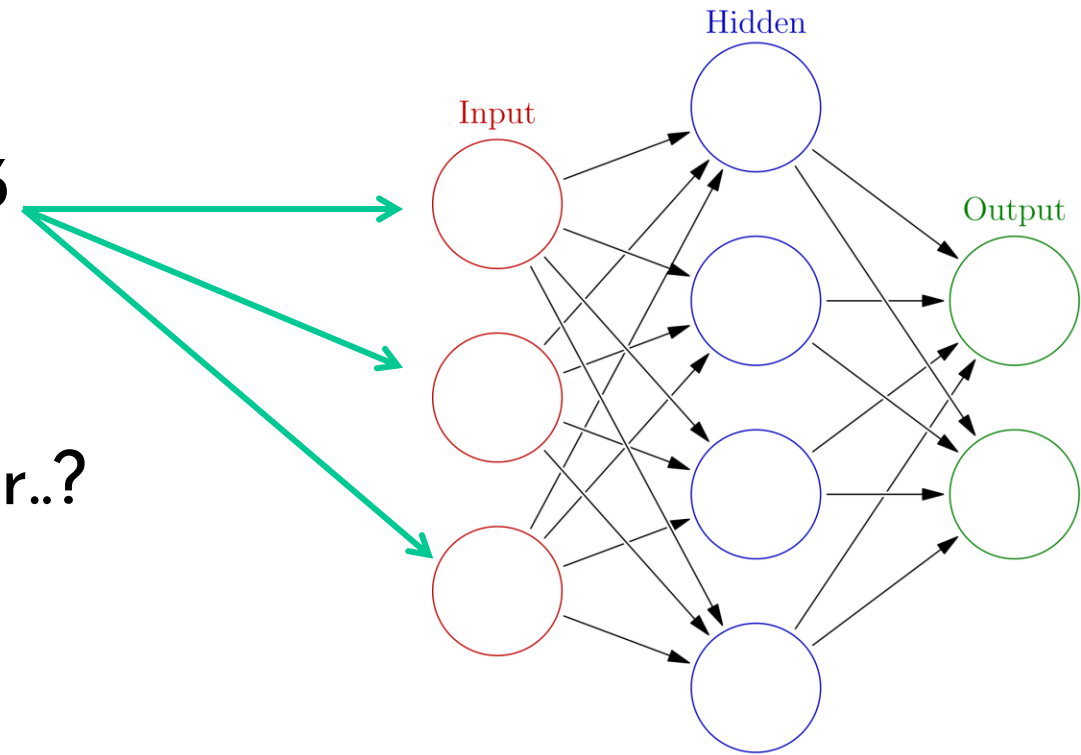


# Perceptron i ett NN



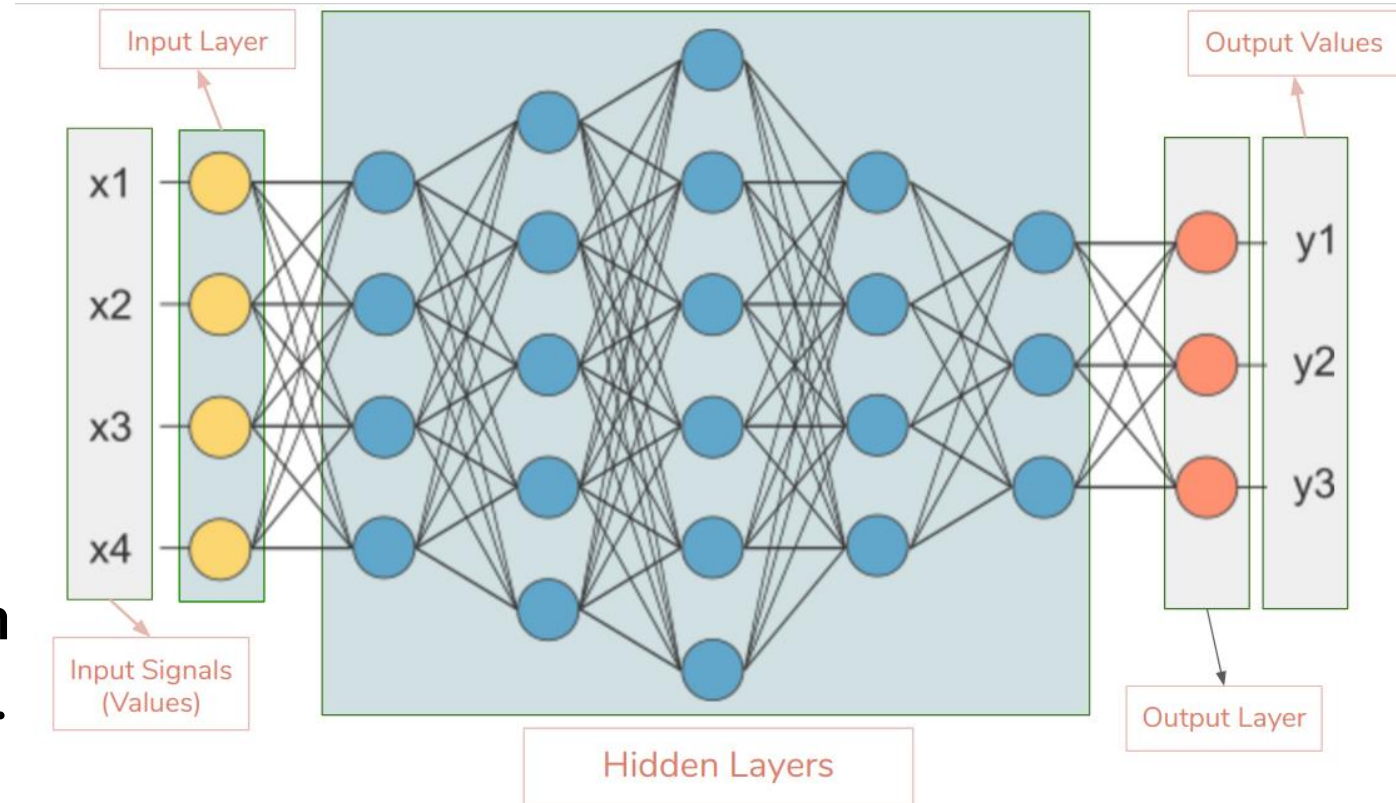
# Neurala nätverk: Input + output lager

- Input lager är din input data.
- I exemplet består inputen av 3 dimensioner
- Kan t.ex. Vara x,y,z-koordinater..?
- Outputen här är 2 dimensioner



# Neurala nätverk - Hidden layers

- De “gömda” lagren är kärnan i ett NN.
- När det finns många gömda lager kallas det “Deep Nerural network”.
- Med varje lager blir modellen mer komplex och mer olinjäritet kan fångas.









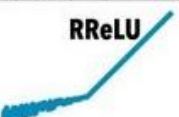

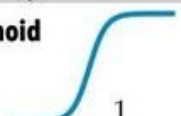
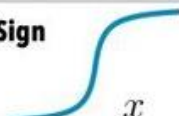

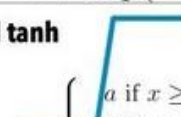
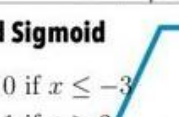
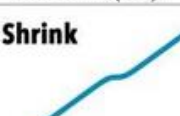
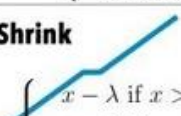
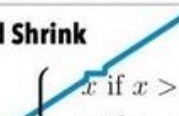


# Aktiveringsfunktioner

- Finns många olika att välja på.
- ReLU är bra go-to.
- Stor påverkan på perceptrons output.

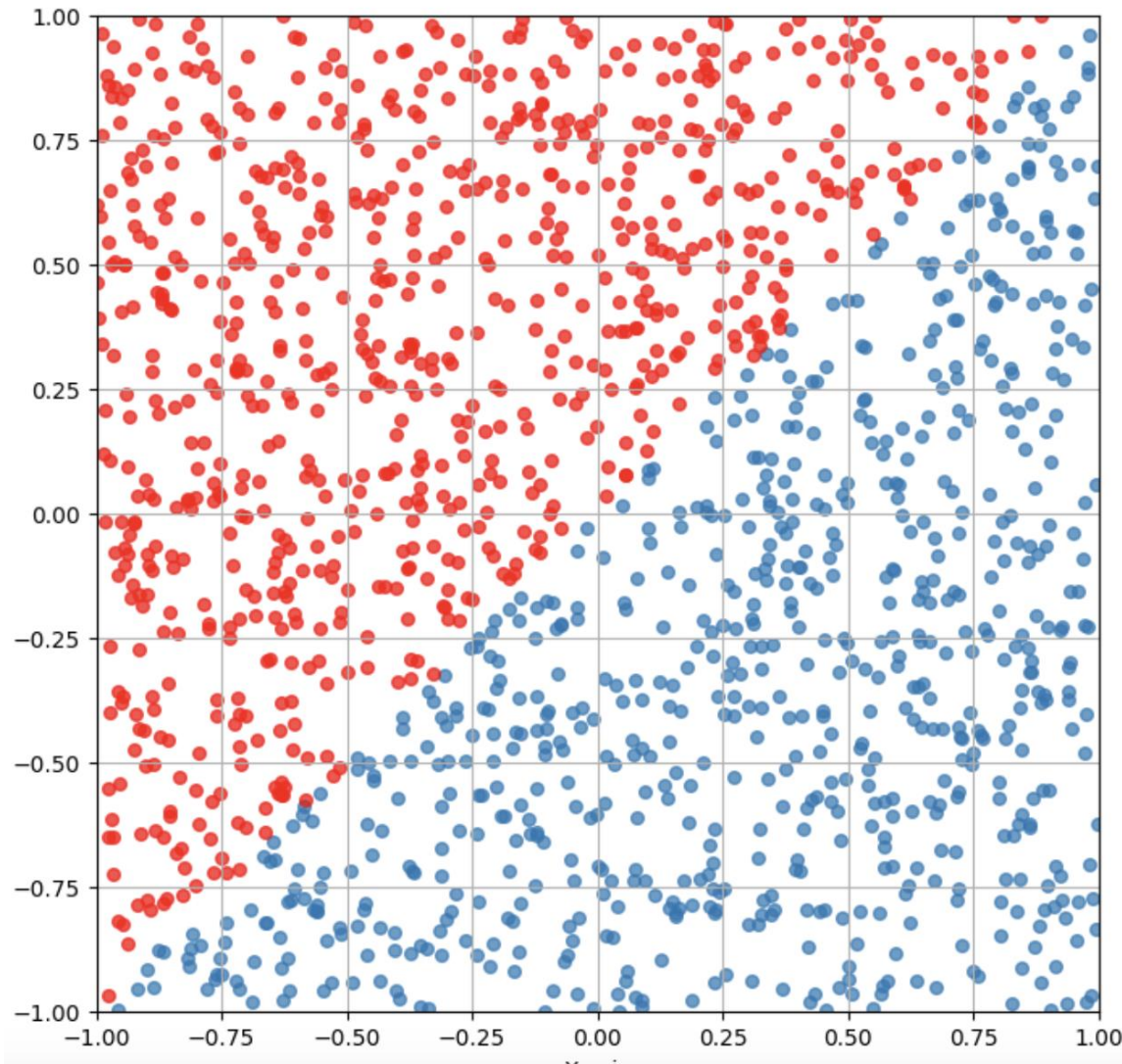
$$f\left(\sum_{i=1}^m w_i x_i + b\right) = \hat{y}$$

- Man kan använda olika aktiveringsfunktioner på olika ställen i nätverket.

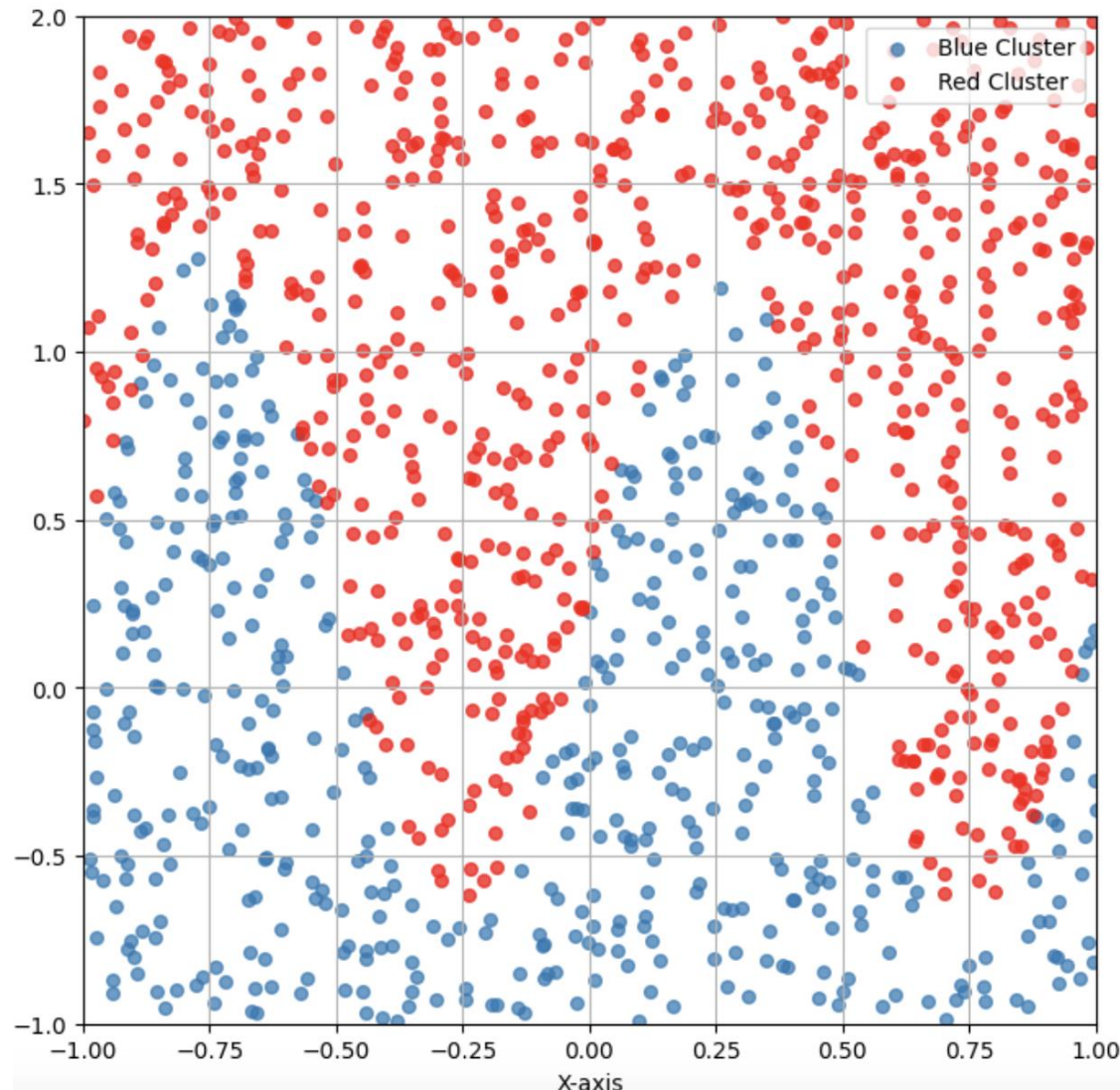
<b>ReLU</b>  $\max(0, x)$	<b>GELU</b>  $\frac{x}{2} \left( 1 + \tanh \left( \sqrt{\frac{2}{\pi}} (x + ax^3) \right) \right)$	<b>PReLU</b>  $\max(0, x)$
<b>ELU</b>  $\begin{cases} x & \text{if } x > 0 \\ \alpha(x \exp x - 1) & \text{if } x < 0 \end{cases}$	<b>Swish</b>  $\frac{x}{1 + \exp -x}$	<b>SELU</b>  $\alpha(\max(0, x) + \min(0, \beta(\exp x - 1)))$
<b>SoftPlus</b>  $\frac{1}{\beta} \log(1 + \exp(\beta x))$	<b>Mish</b>  $x \tanh \left( \frac{1}{\beta} \log(1 + \exp(\beta x)) \right)$	<b>RReLU</b>  $\begin{cases} x & \text{if } x \geq 0 \\ ax & \text{if } x < 0 \text{ with } a \sim \mathcal{R}(l, u) \end{cases}$
<b>HardSwish</b>  $\begin{cases} 0 & \text{if } x \leq -3 \\ x & \text{if } x \geq 3 \\ x(x+3)/6 & \text{otherwise} \end{cases}$	<b>Sigmoid</b>  $\frac{1}{1 + \exp(-x)}$	<b>SoftSign</b>  $\frac{x}{1 +  x }$
<b>Tanh</b>  $\tanh(x)$	<b>Hard tanh</b>  $\begin{cases} a & \text{if } x \geq a \\ b & \text{if } x \leq b \\ x & \text{otherwise} \end{cases}$	<b>Hard Sigmoid</b>  $\begin{cases} 0 & \text{if } x \leq -3 \\ 1 & \text{if } x \geq 3 \\ x/6 + 1/2 & \text{otherwise} \end{cases}$
<b>Tanh Shrink</b>  $x - \tanh(x)$	<b>Soft Shrink</b>  $\begin{cases} x - \lambda & \text{if } x > \lambda \\ x + \lambda & \text{if } x < -\lambda \\ 0 & \text{otherwise} \end{cases}$	<b>Hard Shrink</b>  $\begin{cases} x & \text{if } x > \lambda \\ x & \text{if } x < -\lambda \\ 0 & \text{otherwise} \end{cases}$



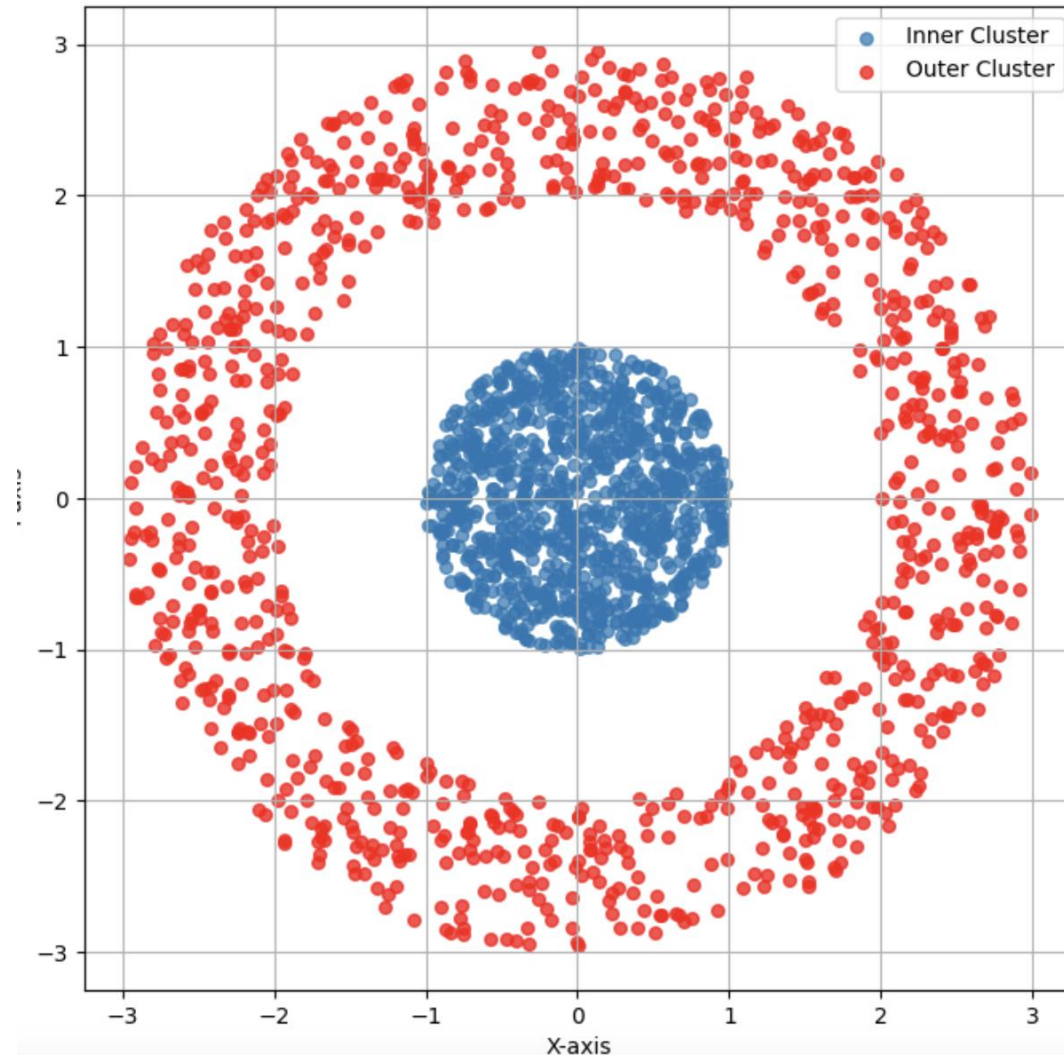
# Kan klustren separeras av ett nätverk med linjär/icke-linjär aktiveringsfunktion?



# Kan klustren separeras av ett nätverk med linjär/icke-linjär aktiveringsfunktion?

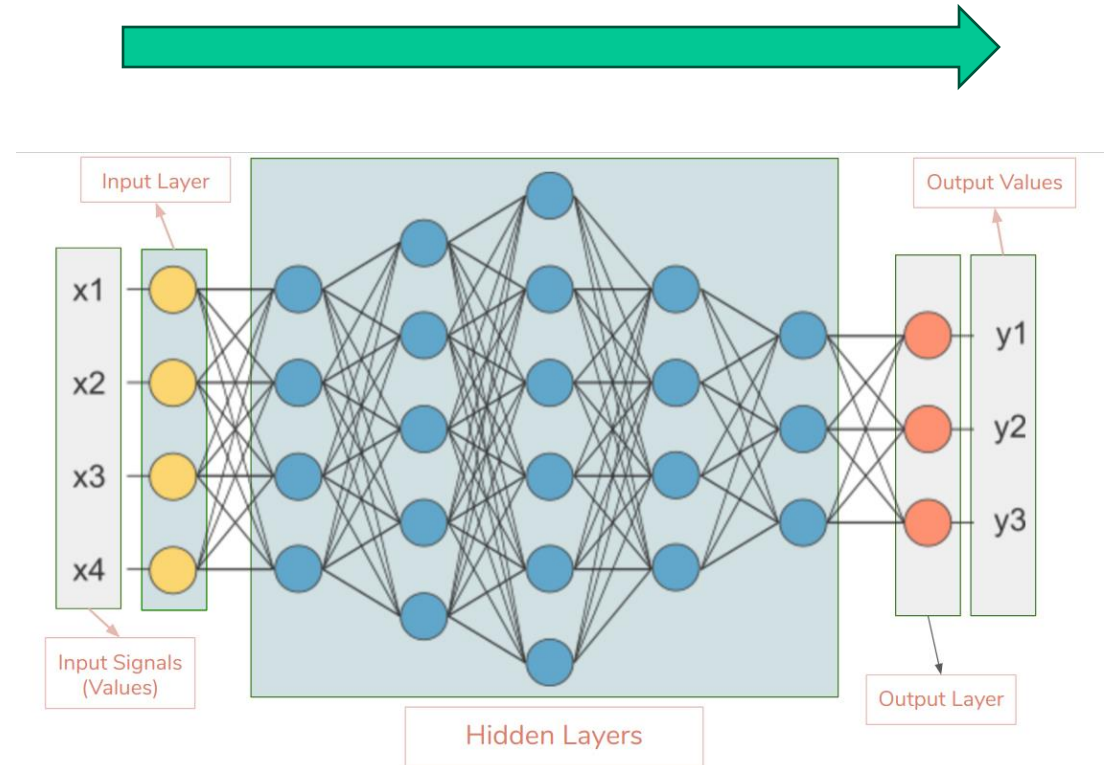


# Kan klustren separeras av ett nätverk med linjär/icke-linjär aktiveringsfunktion?



# Inferens av NN (Forward pass)

- Input en datapunkt och beräkna alla steg genom hela nätverket tills du får en output.





# Inferens av NN - matte

- $x$ : input vektor
- $y$ : mål vektor (ground truth)
- $\hat{y}$ : output vektor
- $L$ : antalet lager
- $f^l$ : activation funktion vid lager  $l$ .
- $W^l = (w_{jk}^l)$ : vikterna mellan lager  $l - 1$  och  $l$ , där  $(w_{jk}^l)$  är vikten mellan  $k$ :te noden i lager  $l - 1$  och  $j$ :te noden i lager  $l$ .

Totala ekvationen för nätverket blir då

$$g(x) := f^L(W^L f^{L-1}(W^{L-1} \dots f^1(W^1 x) \dots))$$

# Inferens av NN - matte

- $x$ : input vektor
- $y$ : mål vektor (ground truth)
- $\hat{y}$ : output vektor
- $L$ : antalet lager
- $f^l$ : activation funktion vid lager  $l$ .
- $W^l = (w_{jk}^l)$ : vikterna mellan lager  $l - 1$  och  $l$ , där  $(w_{jk}^l)$  är vikten mellan  $k$ :te noden i lager  $l - 1$  och  $j$ :te noden i lager  $l$ .

För en datapunkt i från träningsdatan  $\{(x_i, y_i)\}$ , så beräknas outputen av nätverket enligt

$$g(x_i) = \hat{y}_i$$

# Hur lär sig ett nätverk?

- Vi försöker approximera en funktion.
- Vi vet inte vad funktionen är, men när vi vet input och output och ground truth, så kan vi beräkna felet.
- Vi stävar efter att *minimera felet* mellan den riktiga, okända, funktionen och vår approximation.

# Loss functions

- Ett sätt att mäta hur “fel” vår output är jämfört med ground truth.
- Mean squared Error (MSE) är populär för regressionsproblem.
- Cross entropy loss är bra för klassificerings problem.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$



# Inferens av NN - matte

- $x$ : input vektor
- $y$ : mål vektor (ground truth)
- $\hat{y}$ : output vektor
- $L$ : antalet lager
- $f^l$ : activation funktion vid lager  $l$ .
- $W^l = (w_{jk}^l)$ : vikterna mellan lager  $l - 1$  och  $l$ , där  $(w_{jk}^l)$  är vikten mellan  $k$ :te noden i lager  $l - 1$  och  $j$ :te noden i lager  $l$ .

“Lossen” för ett nätverk blir då

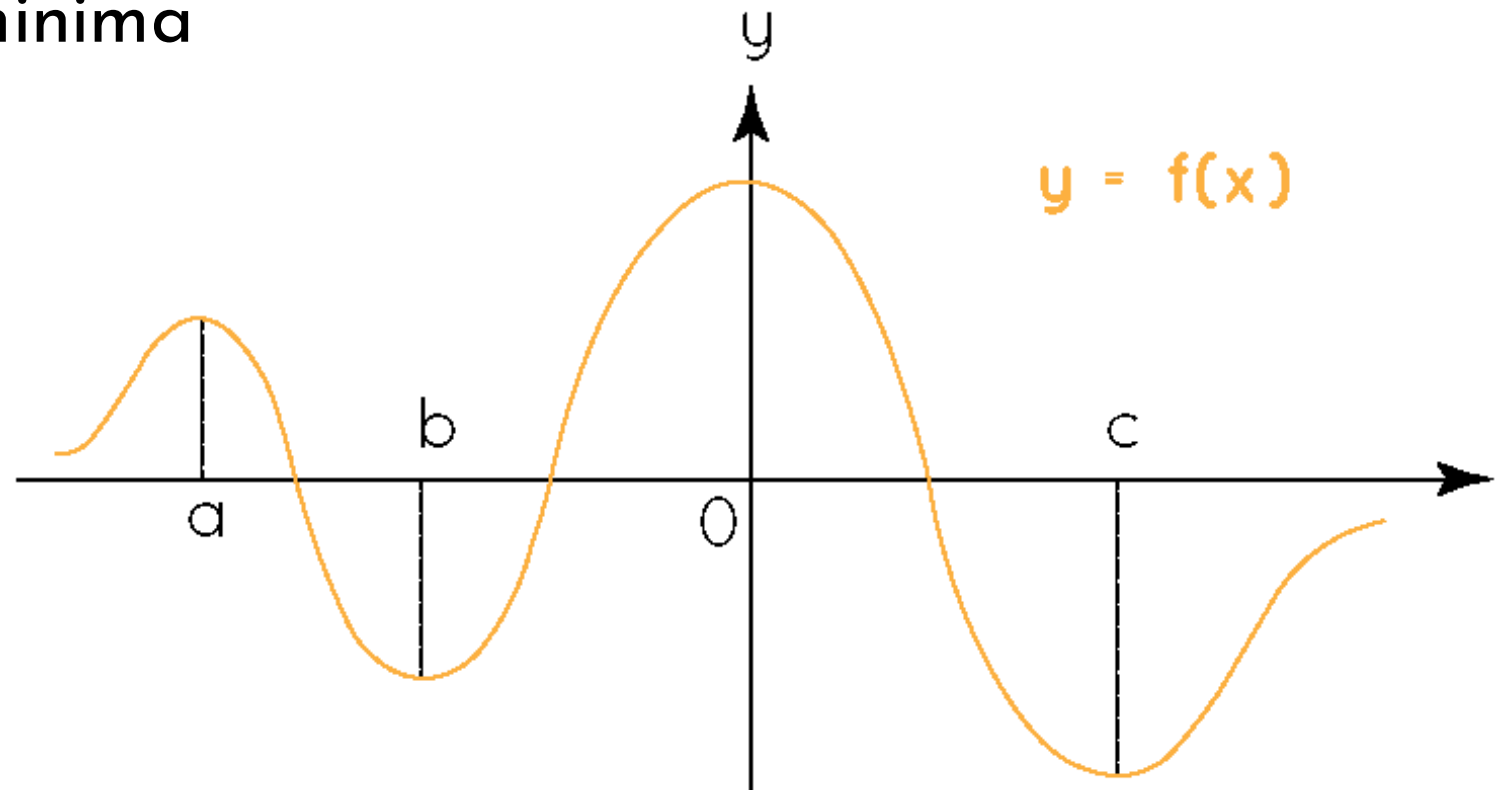
$$\mathcal{L}(\hat{y}_i, y_i) = Loss$$

# Bakåtpropagering (Back propagation)

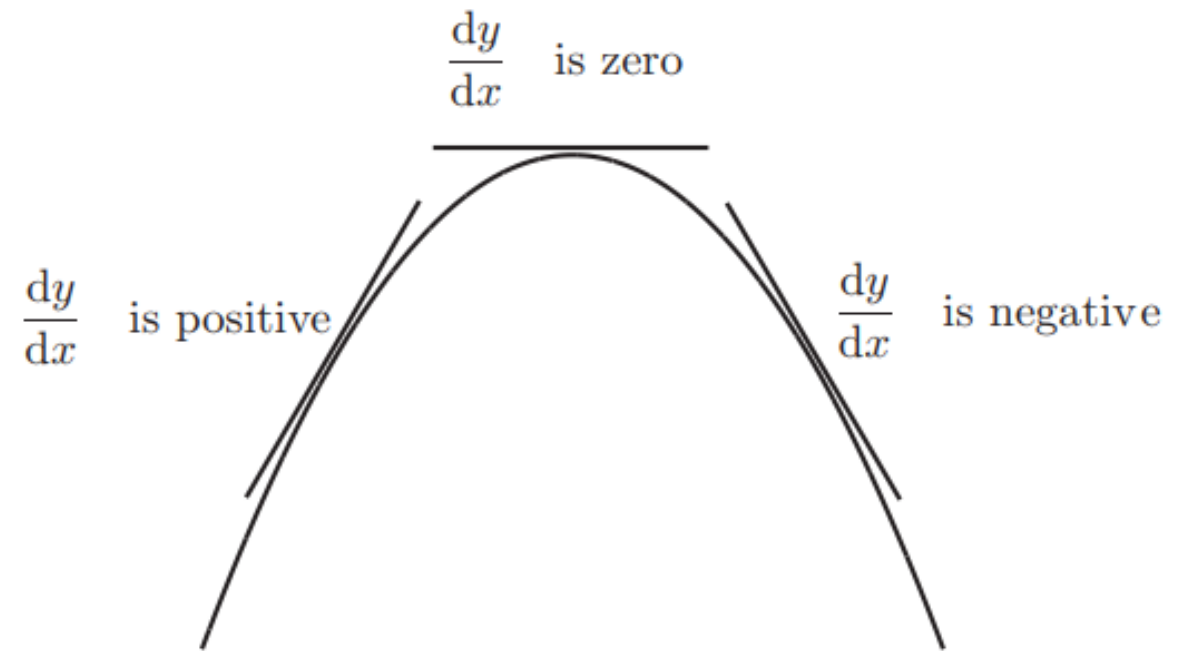
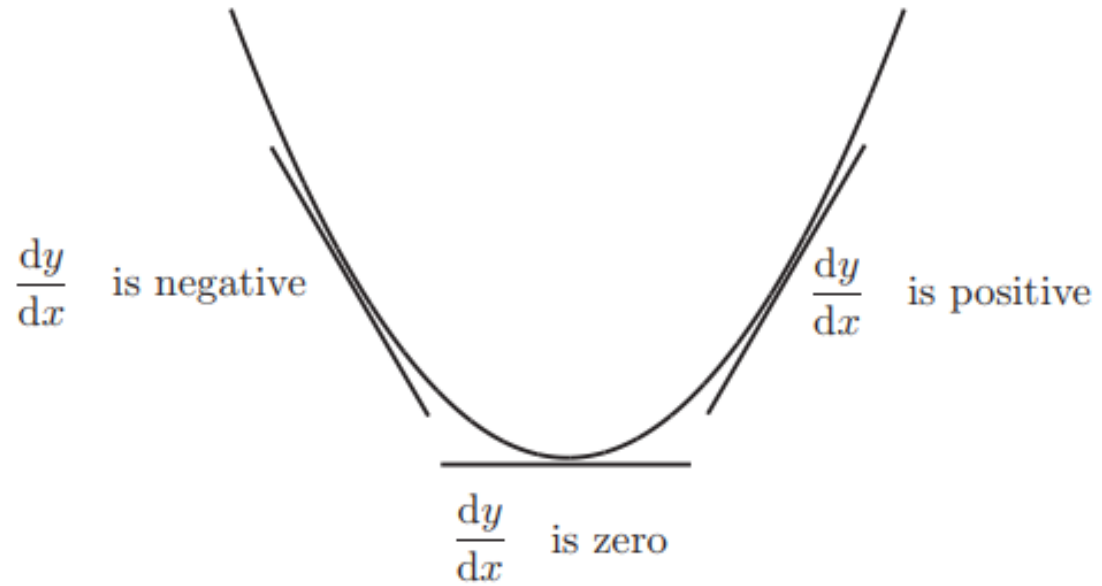
- Bakåtpropagering är hur vi uppdaterar ett nätverk med avseende på felet (vilket i sin tur tillåter nätverket att **lära sig**).
- Vikterna i ett nätverk är det som uppdateras och således det som "lärt sig".
  - Vanligt att initialisera vikterna till random värden innan träningen påbörjas.
- Innan vi går in på det behöver vi kunna...

# Minima och maxima

- Lokala maxima och minima
- Globala maxima och minima



# Derivata



# Gradient

- Gradienten är en **vektor** som pekar i riktningen av den skarpaste lutningen i en funktion.
- Ej att blanda ihop med derivata som är ett **värde** som beskriver lutningen.

# Kedjeregeln

## The Chain Rule

For  $F(x) = f(g(x))$

$$F'(x) = f'(g(x)) \cdot g'(x)$$

Derivative of outer function

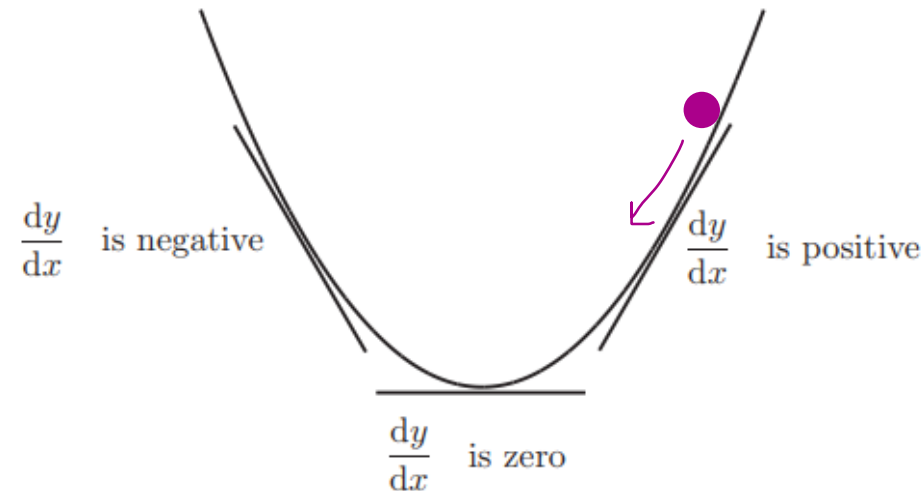
Derivative of inner function

# Bakåtpropagering (Back propagation)

- Bakåtpropagering beräknar gradienten hos vikterna givet ett fel.
- "Felet" är skillnaden i vårt NNs output och vårt rätta svar, vår ground truth. (Skillnaden mäts genom loss funktionen)
- Med bakåtpropagering får vi reda på gradienten hos varje enskild vikt.
- Vi vill att alla vikters gradient ska hamna i globalt minima,
  - det betyder att felet på varje vikt är så litet det bara kan bli.
- Minimerar vi felet på alla vikter så har vi approximerat funktionen.

# Backprop forts

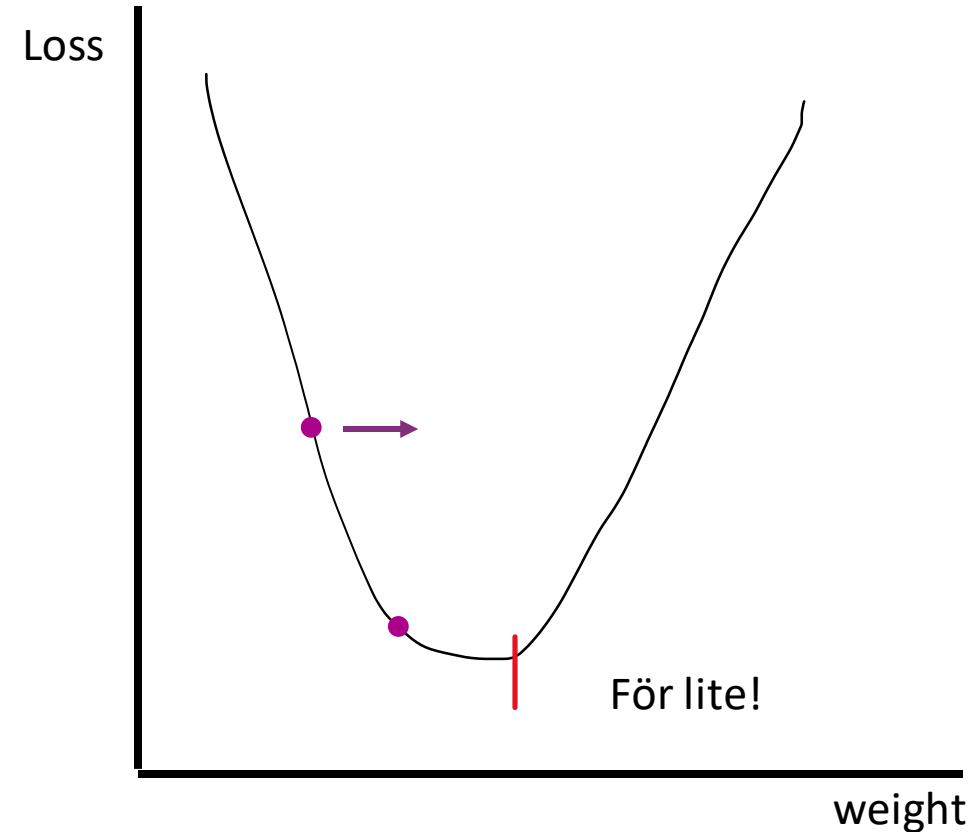
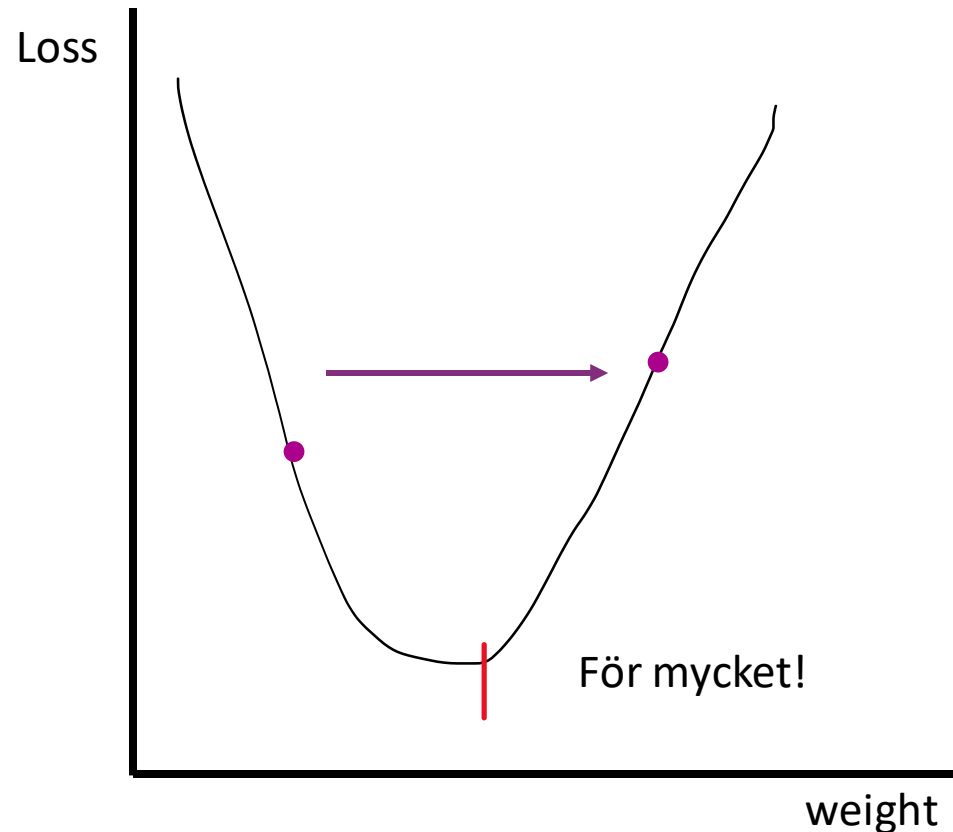
- För varje träningsdatapunkt där vi har en input  $x$  och en groundtruth  $y$ .
- Vi stoppar in inputen  $x$  i närverket och får ut outputen  $\hat{y}$ .
- Lossfunktionen beräknar hur mycket "fel" vi har.  $\mathcal{L}(y, \hat{y})$ .
- Bakåtpropagering beräknar vilken gradient varje vikt har (vi vill att alla vikter ska gå mot minima).
- Då vet vi vilket håll vi ska ändra vikten\*.





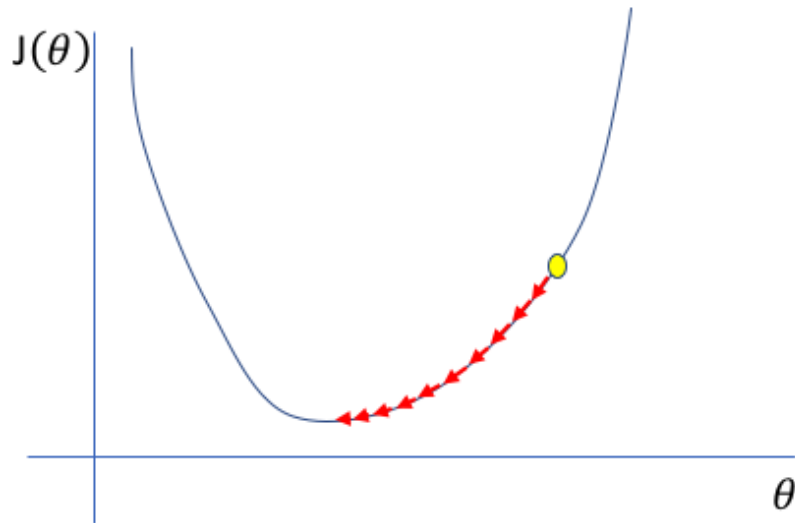
# Minimera vikterna givet felet (loss)

- Vi vet nu vilken "riktning" vi ska ändra vikten för att den ska bli mindre, men hur stort steg ska vi ta?



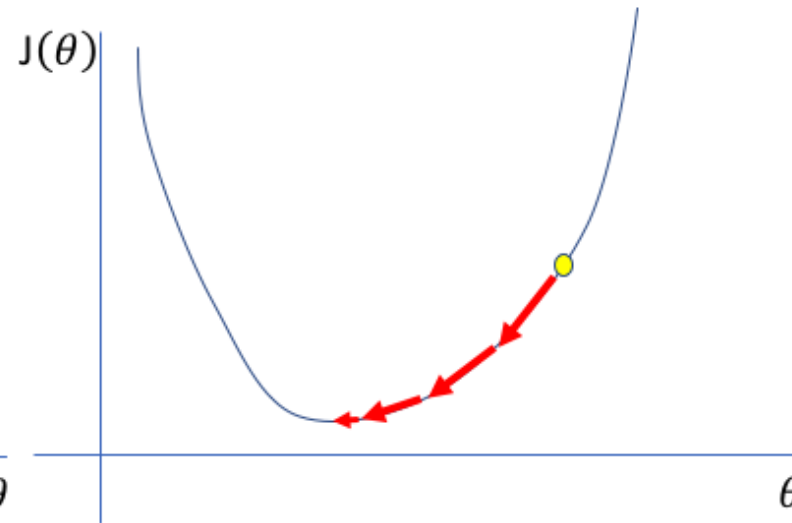
# Learning rate (hur stora steg vi tar)

Too low



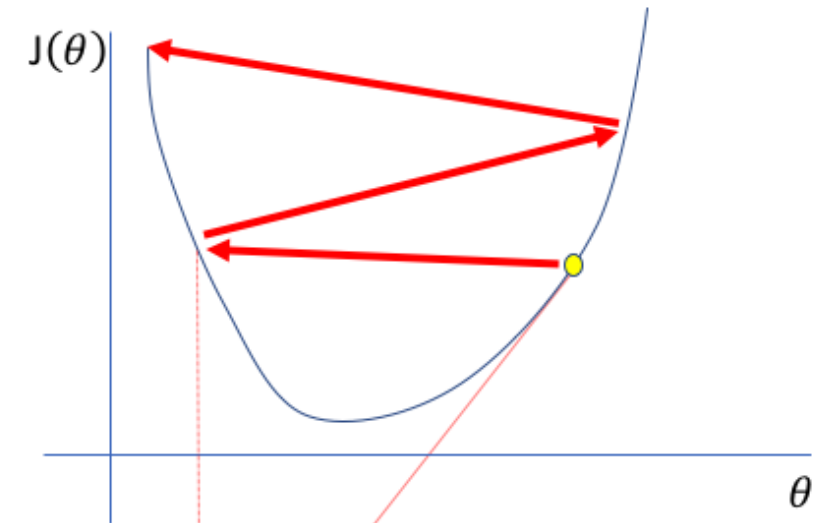
A small learning rate requires many updates before reaching the minimum point

Just right



The optimal learning rate swiftly reaches the minimum point

Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors

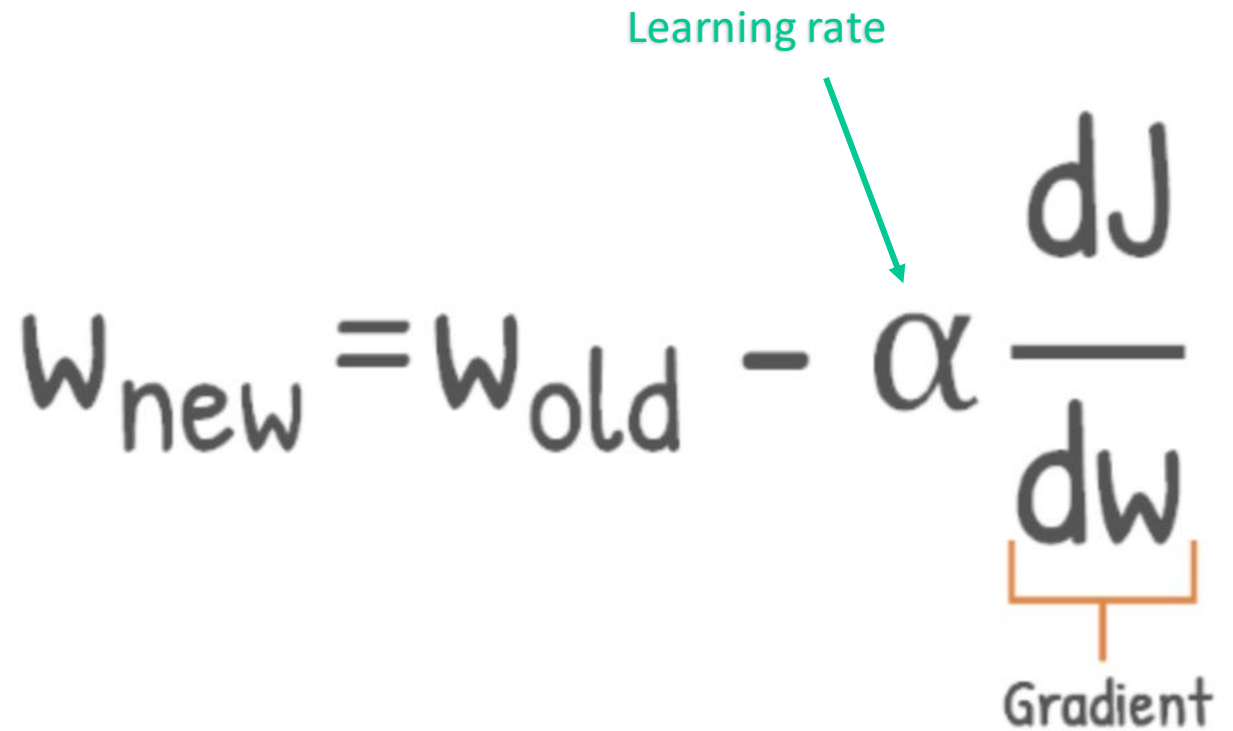
# Hur beräknar vi viktens nya värde?

- Denna ekvation kallas Gradient Descent!

$$w_{\text{new}} = w_{\text{old}} - \alpha \frac{dJ}{dw}$$

Learning rate

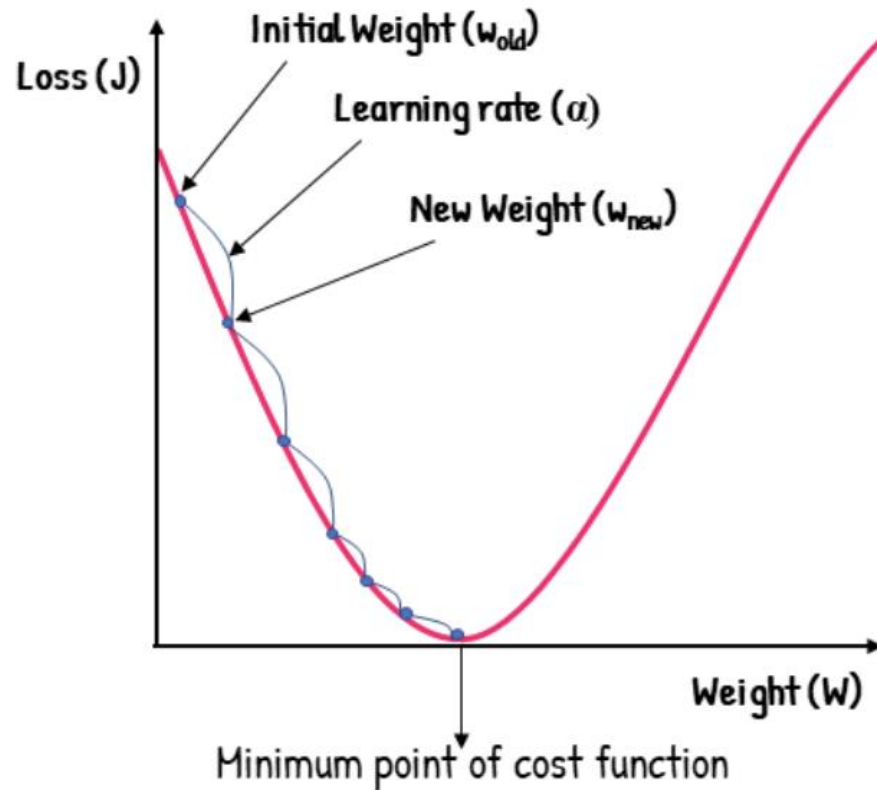
Gradient

The diagram shows the equation  $w_{\text{new}} = w_{\text{old}} - \alpha \frac{dJ}{dw}$ . A teal arrow points from the text "Learning rate" to the Greek letter  $\alpha$ . An orange bracket is placed under the fraction  $\frac{dJ}{dw}$ , with the word "Gradient" written below it.

- Descent = nerstigning, nedgång

# "Gå nerför loss funktionen"

## Gradient Descent



$$w_{new} = w_{old} - \alpha \frac{\delta J}{\delta w}$$

Learning rate

Gradient

När vi går ner för backen så *minimerar vi felet*.

# Backpropagation vs Gradient descent

	Backpropagation	Gradient Descent
Definition	An algorithm for calculating the gradients of the cost function	Optimization algorithm used to find the weights that minimize the cost function
Requirements	Differentiation via the chain rule	<ul style="list-style-type: none"><li>• Gradient via Backpropagation</li><li>• Learning rate</li></ul>
Process	Propagating the error backwards and calculating the gradient of the error function with respect to the weights	Descending down the cost function until the minimum point and find the corresponding weights

# Bakåtpropagering

- Detta lilla steget var bara 1 datapunkt från vår träningsdata.
- Vi vill upprepa detta för alla datapunkter, troligen flera gånger om.
- Målet är att approximera funktionen. Är modellen för simpel (eller för komplex) för problemet så blir approximationen inte bra trots att vi nått minima. (mer om det vid en senare lektion...)

# Att träna ett nätverk - grunderna

- Random initialisera vikterna
- Gå igenom varje datapunkt och gör backprop och uppdatera vikterna.
- Upprepa tills nätverket konvergerat ("lärt sig klart", alltså hittat funktionen vi letar efter.
- Använd nätverket på osedd data.



XOR code along!