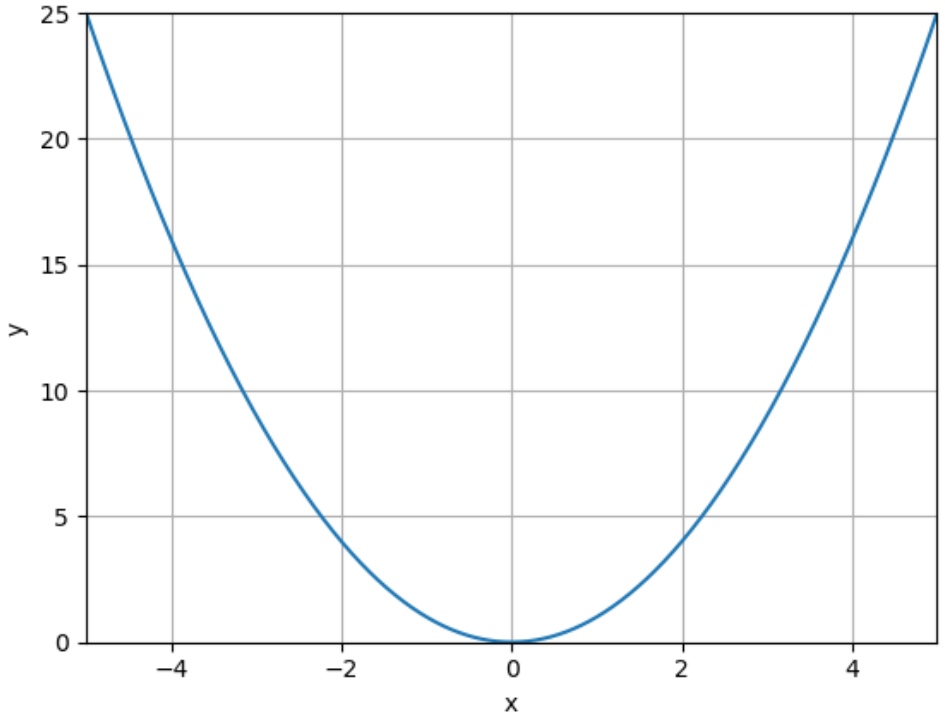
Blixtkurs i flervariabelsanalys

- 1 dimension:
  - Givet: en funktion $y = f(x)$, dvs $f: R^1 \rightarrow R^1$ (från skalär till skalär).
  - Derivataoperatorn definieras som $\frac{\delta}{\delta x} f(x) = \frac{\delta f(x)}{\delta x} = f'_x(\mathrm{x}) = \lim_{h \to 0} \frac{f(x+h)+f(x)}{h}$.

En liten, men viktig, detalj:

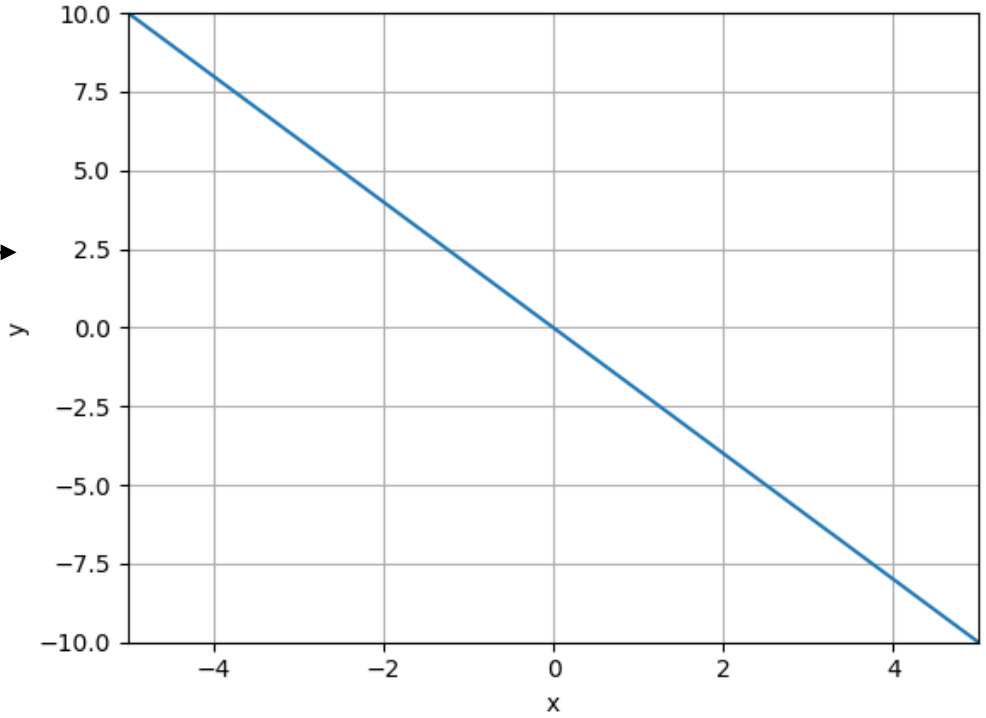
- $\left.\frac{\delta f(x)}{\delta x}\right|_{x=a} = f'_x(a)$ betecknar värdet för derivatafunktionen i punkten a. Vi kan inte skriva detta som $\frac{\delta f(a)}{\delta x}$, eftersom $\frac{\delta f(a)}{\delta x} = \frac{\delta}{\delta x} f(a) = \frac{\delta}{\delta x}(constant) = 0$.

Exempel



$$y = f(x) = x^2$$

$$\boxed{-\frac{\delta}{\delta x}}$$



$$y = -f'_x(x) = -2x$$

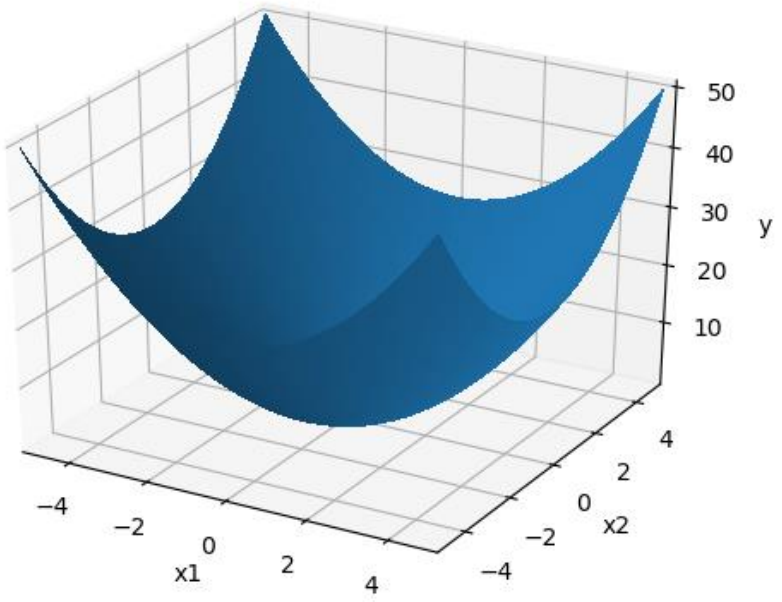
- $\frac{\delta}{\delta x} f(x) = \frac{\delta f(x)}{\delta x} = f'_x(x)$ är en funktion $f'_x: R^1 \rightarrow R^1$ (från skalär till skalär).

- Notera: den negativa derivatans tecken för en punkt a anger i vilken riktning längs x-axeln från punkten $x = a$ man bör förflytta sig för att nå ett (lokalt) minimum.

Blixtkurs i flervariabelsanalys

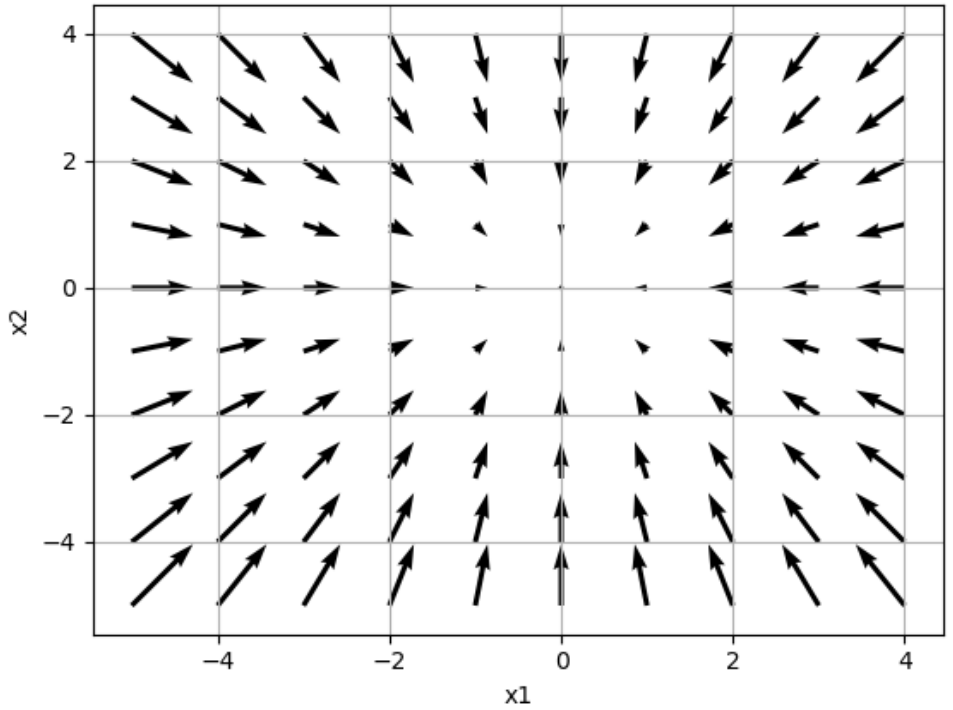
- n≥2 dimensioner:
  - Givet: en funktion $y = f(\boldsymbol{x}) = \mathrm{f}(x_1, x_2, \dots, x_n)$, dvs $f: R^n \to R^1$ (från n-vektor till skalär)
  - Gradientoperatorn $\boldsymbol{\nabla_x}$ (”nabla”) ges av $\boldsymbol{\nabla_x} = \boldsymbol{\nabla}_{x_1,x_{2,\dots}} = (\frac{\delta}{\delta x_1}, \frac{\delta}{\delta x_2}, \dots, \frac{\delta}{\delta x_n})^T$
  - Funktionen f:s gradient är $\boldsymbol{\nabla_x} f(\boldsymbol{x}) = (\frac{\delta f(\boldsymbol{x})}{\delta x_1}, \frac{\delta f(\boldsymbol{x})}{\delta x_2}, \dots)^T = (f'_{x_1}(\boldsymbol{x}), f'_{x_2}(\boldsymbol{x}), \dots)^T$
  - $\boldsymbol{\nabla_x} f(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}_{val}} = \boldsymbol{\nabla_x} f(x_1, x_2, \dots)|_{(x_1,x_2,\dots)=(a,b,\dots)}$ betecknar gradientvektorns faktiska värde i punkten $\boldsymbol{x}_{val} = (a, b, \dots)$.

Exempel



$y = f(x_1, x_2) = x_1^2 + x_2^2$

$\rightarrow$ $-\nabla_x$ $\rightarrow$



$-\nabla_x f(x_1, x_2)|_{x_1=a,\, x_2=b} = (-2a, -2b)$

- Gradienten är en funktion $\nabla_x f: R^n \rightarrow R^n$ (från n-vektor till n-vektor)

- Gradienten i en punkt pekar i den riktning vartåt funktionen ökar snabbast. Des storlek representerar hur stor ökningen är. Den negativa gradienten anger därför hur man bör förflytta sig i rymden av möjliga input $\mathbf{x} = (x_1, x_2, \dots)$ för att nå ett (lokalt) minimum.

Grundläggande optimering

- Givet: En cost/loss-funktion $C = L(\boldsymbol{x}, \boldsymbol{w}) = L(x_1, x_2, \dots, w_1, w_2, \dots)$.
- Problem: Välj element i $\boldsymbol{w}$ för att göra C så litet som möjligt.
  - Dvs $\boldsymbol{x}$ är här att betrakta som konstant(er)
- Den faktiska lösningen $\boldsymbol{w}^{opt} = (w_1^{opt}, w_2^{opt}, \dots)$ finns bland lösningarna till ekvationssystemet:

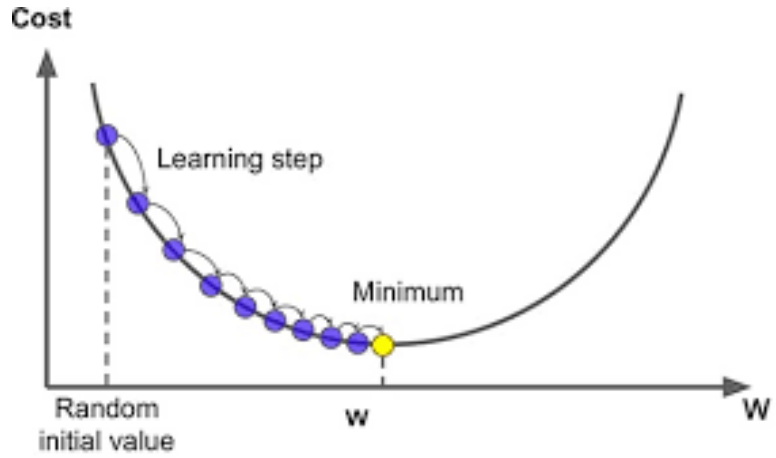
$$\boldsymbol{\nabla}_{\boldsymbol{w}} L|_{\boldsymbol{w}=\boldsymbol{w}^{opt}} = \boldsymbol{0} \leftrightarrow \begin{cases} \left.\frac{\delta L}{\delta w_1}\right|_{w_1=w_1^{opt}} = 0 \\ \left.\frac{\delta L}{\delta w_2}\right|_{w_2=w_2^{opt}} = 0 \\ \vdots \end{cases}$$

, ty gradienten är noll i alla minima/maxima.

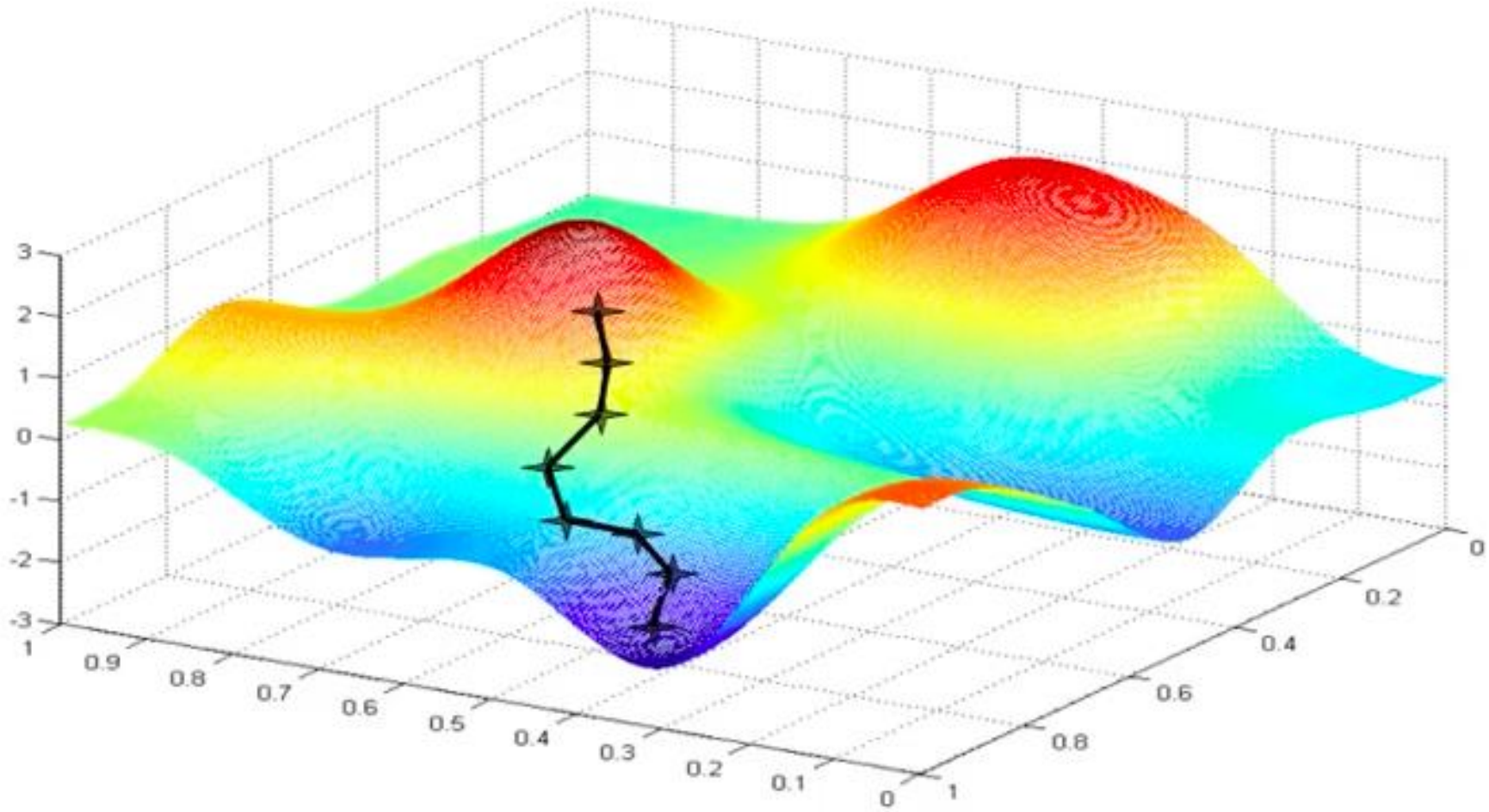
- (Vi antar här att funktionen L är kontinuerlig och deriverbar överallt, dvs inga “hopp” eller “kanter” i funktionsytan).

Grundläggande optimering

- Gradient Descent:
  - Hur hittar man ner från ett berg om man är blind och har minnesförlust? Följ lutningen.
  - Ett iterativt alternativ för hitta en lösning utan att behöva lösa ekvationssystem.
    - För komplicerade kostnadsfunktioner som beror olinjärt på miljontals parametrar kan det helt saknas metoder för att lösa $\boldsymbol{\nabla_w} L = \mathbf{0}$ analytiskt.
    - Att hitta gradienten i en given punkt är dock alltid en (jämförelsevis) enkel operation.



  - Algoritmen:
    1. Initialisera $\boldsymbol{w}_0$ (exempelvis slumpmässigt).
    2. Uppdatera $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \alpha \cdot \boldsymbol{\nabla_w} L(\boldsymbol{x}, \boldsymbol{w})|_{\boldsymbol{w}=\boldsymbol{w}_t}$, (där α är det vi kallar learning rate eller step size).
    3. Repetera steg 2 ovan n gånger (n stort). $\boldsymbol{w}_n$ är (approximativt) lösningen.
  - Metoden är matematiskt garanterad hamna godtyckligt nära ett lokalt minimum efter ett ändligt antal n steg givet tillräckligt litet α!


3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
0.8
0.6
0.4
0.2
0

Grundläggande optimering

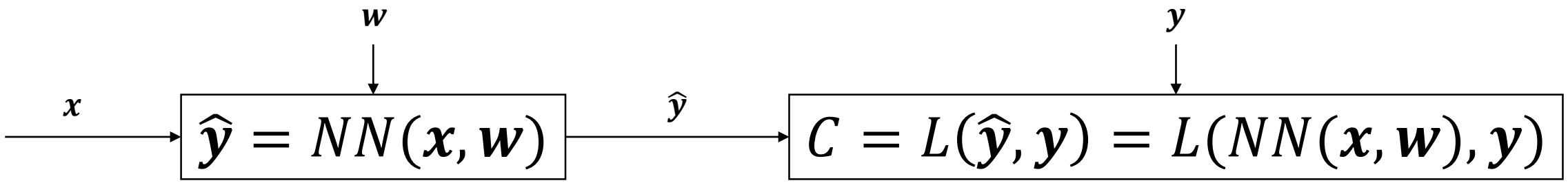
- Stochastic Gradient Descent, SGD:
  - Problem: Kostnadsfunktionen $L(\boldsymbol{x}, \boldsymbol{w})$ kanske inte är ”känd” i bemärkelsen att den beror på input $\boldsymbol{x}$ som inte är konstant, utan dragen från en (okänd) slumpfördelning.
  - Lösning: För $I \geq 1$ observationer av $\boldsymbol{x}$ (skrivet $\boldsymbol{x}^{obs}$), beräkna medelvärdet av kostnaden:

$$\hat{L}\left(\boldsymbol{x}^{obs}, \boldsymbol{w}\right) = \frac{\sum_{i=1}^{I} L(\boldsymbol{x}_i^{obs}, \boldsymbol{w})}{I} \approx L(\boldsymbol{x}, \boldsymbol{w})$$

  - Nu kan vi använda ungefär samma metod som innan för att minimera $\hat{L}$:
    1. Initialisera $\boldsymbol{w}_0$
    2. Generera observationer $\boldsymbol{x}^{obs}$
    3. Uppdatera $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \alpha \cdot \boldsymbol{\nabla}_{\boldsymbol{w}} \hat{L}\left(\boldsymbol{x}^{obs}, \boldsymbol{w}\right)\Big|_{\boldsymbol{w}=\boldsymbol{w}_t}$
    4. Repetera steg 2-3 ovan.
  - Detta minimerar även L över stora mängder observerade $\boldsymbol{x}$.

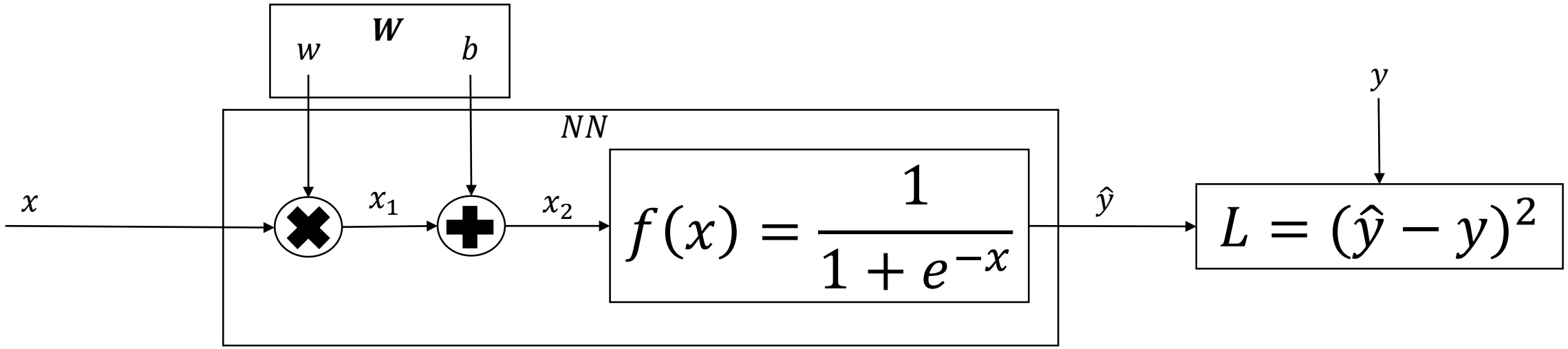
Neuronnät

- (nästan) alla modeller har följande struktur:



- Ett neuronnät $NN(\cdot)$ är en *universal function approximator* som kan lära sig efterlikna någon funktion $f: \boldsymbol{x} \rightarrow \boldsymbol{y}$. Exempelvis kan $\boldsymbol{x}$ vara en bild, $\mathrm{f}(\mathbf{x}) = \boldsymbol{y}$ är 1 om bilden föreställer en katt, 0 annars. f antas *a priori* existera och vara beräkningsbar.
- Vi vill minimera kostnaden C , som mäter hur lika $\hat{\boldsymbol{y}}$ och $\boldsymbol{y}$ är, med avseende på nätverkets justerbara parametrar $\boldsymbol{w}$. När C är minimerad lär $NN(\cdot, \boldsymbol{w})$ ”likna” f.
- Vi kan tillämpa SGD, där ”observationerna” utgörs av exempel på par $(\boldsymbol{x}, \boldsymbol{y})$. Förhoppningen är att för någon konfiguration av $\boldsymbol{w}$ kommer nätverket att beräkna den funktion som $(\boldsymbol{x}, \boldsymbol{y})$ är exempel på input resp. output från.

Exempel:



- Vi använder SGD för att uppdatera w och b:
  1. Initialisera $\boldsymbol{w}_0 = (w_0, b_0)$
  2. Generera ett exempel x med motsvarande y
  3. Uppdatera $(w_{t+1}, b_{t+1}) \leftarrow (w_t, b_t) - \alpha \cdot \boldsymbol{\nabla}_{w,b} \, (\hat{y} - y)^2 \big|_{w=w_t, b=b_t}$
  4. Repetera steg 2-3 ovan.
- För att kunna beräkna $\boldsymbol{\nabla}_{w,b} L = (\frac{\delta L}{\delta w}, \frac{\delta L}{\delta b})$ behöver vi använda kedjeregeln:
  - $\frac{\delta L}{\delta w} = \frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta x_2} \frac{\delta x_2}{\delta x_1} \frac{\delta x1}{\delta w}$ och $\frac{\delta L}{\delta b} = \frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta x_2} \frac{\delta x_2}{\delta x_b}$
- Detta kallas *backpropagation*