Maskinlæring
Kunstige nevrale nettverk

Ole Christian Eidheim

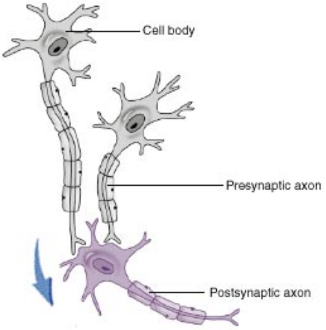
Institutt for informatikk og e-læring,
NTNU

24. august 2020

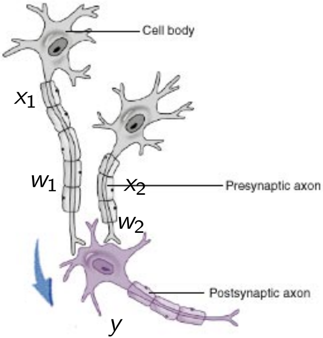
Oversikt

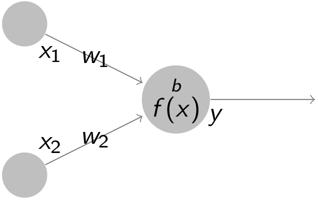
- **Kunstige nevrale nettverk**
  - Tapsfunksjon ved klassifisering
  - Klassifisering med flere klasser
  - Testing av optimalisert modell
- Øving 2

Kunstige nevrale nettverk: svært forenklet etterligning av hjernen - Interesserte kan lese mer om biologiske nevroner for eksempel her

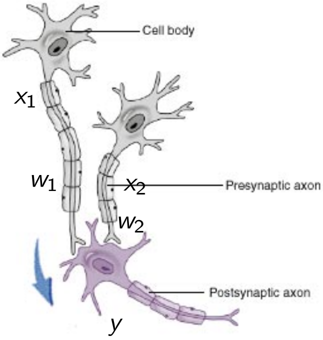


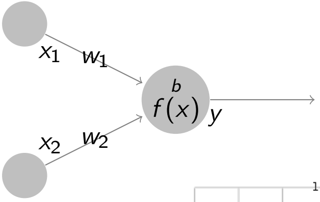
Kunstige nevrale nettverk: svært forenklet etterligning av hjernen - Interesserte kan lese mer om biologiske nevroner for eksempel her

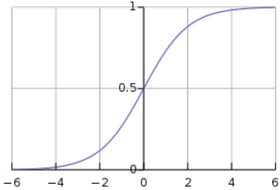




Kunstige nevrale nettverk: svært forenklet etterligning av hjernen
- Interesserte kan lese mer om biologiske nevroner for eksempel her



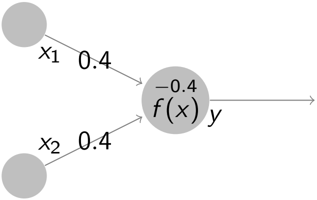




$\sigma(z) = \frac{1}{1+e^{-z}}$

$$y = f(x) = \sigma\left(\sum_j (x_j w_j) + b\right) = \sigma(xW + b) = \sigma\left(\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + b\right)$$

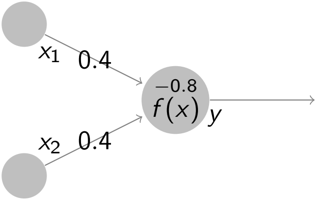
Kunstige nevrale nettverk
- OR



$$y = f(x) = \sigma(xW + b) = \sigma\left(\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 0.4 \\ 0.4 \end{bmatrix} - 0.4\right)$$

$$f(\begin{bmatrix} 0 & 0 \end{bmatrix}) = \sigma(-0.4) \approx 0.4 \approx 0$$
$$f(\begin{bmatrix} 0 & 1 \end{bmatrix}) = \sigma(0) \approx 0.5 \approx 1$$
$$f(\begin{bmatrix} 1 & 0 \end{bmatrix}) = \sigma(0) \approx 0.5 \approx 1$$
$$f(\begin{bmatrix} 1 & 1 \end{bmatrix}) = \sigma(0.4) \approx 0.6 \approx 1$$

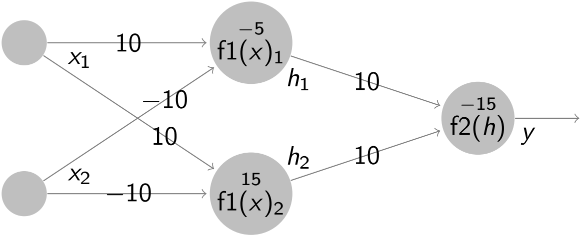
Kunstige nevrale nettverk - AND



$$y = f(x) = \sigma(xW + b) = \sigma\left(\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 0.4 \\ 0.4 \end{bmatrix} - 0.8\right)$$

$$\begin{aligned}
f(\begin{bmatrix} 0 & 0 \end{bmatrix}) &= \sigma(-0.8) \approx 0.3 \approx 0 \\
f(\begin{bmatrix} 0 & 1 \end{bmatrix}) &= \sigma(-0.4) \approx 0.4 \approx 0 \\
f(\begin{bmatrix} 1 & 0 \end{bmatrix}) &= \sigma(-0.4) \approx 0.4 \approx 0 \\
f(\begin{bmatrix} 1 & 1 \end{bmatrix}) &= \sigma(0.0) \approx 0.5 \approx 1
\end{aligned}$$

Kunstige nevrale nettverk - XOR



$$h = \mathrm{f1}(x) = \sigma(x\mathrm{W1}+\mathrm{b1}) = \sigma\left(\begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} 10 & -10 \\ 10 & -10 \end{bmatrix} + \begin{bmatrix} -5 & 15 \end{bmatrix}\right)$$

$$y = \mathrm{f2}(\mathrm{f1}(x)) = \mathrm{f2}(h) = \sigma(h\mathrm{W2}+\mathrm{b2}) = \sigma\left(\begin{bmatrix} h_1 & h_2 \end{bmatrix}\begin{bmatrix} 10 \\ 10 \end{bmatrix} + \begin{bmatrix} -15 \end{bmatrix}\right)$$

$$f(\begin{bmatrix} 0 & 0 \end{bmatrix}) \approx 0$$
$$f(\begin{bmatrix} 0 & 1 \end{bmatrix}) \approx 1$$
$$f(\begin{bmatrix} 1 & 0 \end{bmatrix}) \approx 1$$
$$f(\begin{bmatrix} 1 & 1 \end{bmatrix}) \approx 0$$

Oversikt

- Kunstige nevrale nettverk
  - **Tapsfunksjon ved klassifisering**
  - Klassifisering med flere klasser
  - Testing av optimalisert modell
- Øving 2

Tapsfunksjon ved klassifisering

- Cross Entropy brukes i stedet for Mean Squared Error
  - Når en kun har en klasse ($\hat{y}^{(i)}$ og $f(\hat{x}^{(i)})$ inneholder bare en kolonne):
    - $loss = -\frac{1}{n}\sum_{i=1}^{n}\left[\hat{y}^{(i)} log(f(\hat{x}^{(i)}) + (1-\hat{y}^{(i)}) log(1-f(\hat{x}^{(i)}))\right]$
    - Funksjon i PyTorch som kan brukes: torch.nn.functional.binary_cross_entropy_with_logits
    - Se interaktive visualiseringer av modeller med tapsfunksjoner for OR og XOR operatorene: https://gitlab.com/ntnu-tdat3025/ann/visualize
      Ikke se på den rotete kildekoden!

Tapsfunksjon ved klassifisering

- Cross Entropy brukes i stedet for Mean Squared Error
  - Når en kun har en klasse ($\hat{y}^{(i)}$ og $f(\hat{x}^{(i)})$ inneholder bare en kolonne):
    - $loss = -\frac{1}{n}\sum_{i=1}^{n}\left[\hat{y}^{(i)}log(f(\hat{x}^{(i)}) + (1-\hat{y}^{(i)})log(1-f(\hat{x}^{(i)}))\right]$

Demonstrasjon av $loss = -\frac{1}{n}\sum_{i=1}^{n}\left[\hat{y}^{(i)}log(f(\hat{x}^{(i)}) + (1-\hat{y}^{(i)})log(1-f(\hat{x}^{(i)}))\right]$:

- Like verdier $\hat{y}^{(1)} = 1$ og $f(\hat{x}^{(1)}) = 0.99$, fører til lav *loss*:
  - $1log(0.99) + (1-1)log(1-0.99) \approx -0.01$
- Like verdier $\hat{y}^{(1)} = 0$ og $f(\hat{x}^{(1)}) = 0.01$, fører til lav *loss*:
  - $0log(0.01) + (1-0)log(1-0.01) \approx -0.01$
- Ulike verdier $\hat{y}^{(1)} = 1$ og $f(\hat{x}^{(1)}) = 0.01$, fører til høy *loss*:
  - $1log(0.01) + (1-1)log(1-0.01) \approx -4.61$
- Ulike verdier $\hat{y}^{(1)} = 0$ og $f(\hat{x}^{(1)}) = 0.99$, fører til høy *loss*:
  - $0log(0.99) + (1-0)log(1-0.99) \approx -4.61$

Oversikt

- Kunstige nevrale nettverk
  - Tapsfunksjon ved klassifisering
  - **Klassifisering med flere klasser**
  - Testing av optimalisert modell
- Øving 2

Klassifisering med flere klasser

- En eller flere uavhengige klasser:
  - $\sigma(z)$ brukes i modellprediktoren $f(x)$
  - Tapsfunksjon: `torch.nn.functional.binary_cross_entropy_with_logits`
    - "Measures the probability error in discrete classification tasks in which each **class is independent and not mutually exclusive**. For instance, one could perform multilabel classification where **a picture can contain both an elephant and a dog at the same time**."
- Flere gjensidig utelukkende klasser:
  - Softmax brukes i stedet for $\sigma(z)$ i modellprediktoren $f(x)$
    - Softmax er en flerklasse-utvidelse av $\sigma(z)$
  - Tapsfunksjon: `torch.nn.functional.cross_entropy`
    - "Measures the probability error in discrete classification tasks in which the **classes are mutually exclusive** (each entry is in exactly one class). For example, each CIFAR-10 image is labeled with one and only one label: **an image can be a dog or a truck, but not both**."

Klassifisering med flere klasser
- de innebygde tapsfunksjonene

- På grunn av økt numerisk stabilitet, tar de innebygde tapsfunksjonene `torch.nn.functional.binary_cross_entropy_with_logits` og `torch.nn.functional.cross_entropy` *logits* som argument i stedet for $f(x)$
  - *logits* er modellprediktoren før normalisering ved hjelp av σ eller *softmax*
    - Ved `torch.nn.functional.binary_cross_entropy_with_logits`: $f(x) = \sigma(logits)$
    - Ved `torch.nn.functional.cross_entropy`: $f(x) = softmax(logits)$

Klassifisering med flere klasser
- de innebygde tapsfunksjonene

Eksempel modell i PyTorch:

```
class SigmoidModel:
    def __init__(self):
        # Model variables
        self.W = torch.tensor([[0.0]], requires_grad=True)
        self.b = torch.tensor([[0.0]], requires_grad=True)

    def logits(self, x):
        return x @ self.W + self.b

    # Predictor
    def f(self, x):
        return torch.sigmoid(self.logits(x))

    # Cross Entropy loss
    def loss(self, x, y):
        return torch.nn.functional.
               binary_cross_entropy_with_logits(self.logits(x), y)
        # Similar to:
        # return -torch.mean(y * torch.log(self.f(x)) +
        #                    (1 - y) * torch.log(1 - self.f(x)))
```

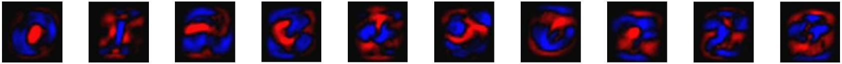
Klassifisering med flere gjensidig utelukkende klasser - MNIST

Stor database av håndskrevne tall, for eksempel:

$\hat{x}^{(1)} =$ , $\hat{x}^{(2)} =$ , $\hat{x}^{(3)} =$ , $\hat{x}^{(4)} =$ , der svart=1, hvitt=0

$$\hat{y}^{(1)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\hat{y}^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\hat{y}^{(3)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\hat{y}^{(4)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Etter optimalisering av $f(x) = softmax(xW + b)$:

$W = \Big[$  $\Big]$

, der $W_{i,j}$ er positiv/negativ

$f($  $) \approx \begin{bmatrix} 0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.1 \end{bmatrix}$

$f($  $) \approx \begin{bmatrix} 0 & 0.7 & 0 & 0 & 0.1 & 0 & 0 & 0.2 & 0 & 0 \end{bmatrix}$

Oversikt

- Kunstige nevrale nettverk
  - Tapsfunksjon ved klassifisering
  - Klassifisering med flere klasser
  - **Testing av optimalisert modell**
- Øving 2

Testing av optimalisert modell

- Vi måler hvor *nøyaktig* (engelsk: *accurate*) en optimalisert modell er gjennom et *testdatasett*
  - Testdatasettet inneholder observasjoner $(\hat{x}^{(i)}, \hat{y}^{(i)})$ som ikke inngår i treningsdatasettet (som er brukt i optimaliseringen)
  - En kan regne ut nøyaktighetet av en modell som predikerer gjensidig utelukkende klasser i PyTorch med for eksempel med metoden:
    - ```
      def accuracy(self, x, y):
          return torch.mean(torch.eq(self.f(x).argmax(1),
                                     y.argmax(1)).float())
      ```
      ```

Oversikt

- Kunstige nevrale nettverk
  - Tapsfunksjon ved klassifisering
  - Klassifisering med flere klasser
  - Testing av optimalisert modell
- **Øving 2**

Øving 2

Denne øvingen bygger videre på Øving 1

a) Lag en modell som predikerer tilsvarende NOT-operatoren. Visualiser resultatet etter optimalisering av modellen.

b) Lag en modell som predikerer tilsvarende NAND-operatoren. Visualiser resultatet etter optimalisering av modellen.

c) Lag en modell som predikerer tilsvarende XOR-operatoren. Før du optimaliserer denne modellen må du initialisere modellvariablene med tilfeldige tall for eksempel mellom -1 og 1. Visualiser både når optimaliseringen konvergerer og ikke konvergerer mot en riktig modell.

d) Lag en modell med prediktoren $f(x) = softmax(xW + b)$ som klassifiserer handskrevne tall. Se mnist for eksempel lasting av MNIST datasettet, og visning og lagring av en observasjon. Du skal oppnå en nøyaktighet på 0.9 eller over. Lag 10 .png bilder som viser W etter optimalisering.