Neural Network

Huy V. Vo

MASSP

Outline

- Introduction
- Multi-Layer Perceptron (MLP)
- Back-propagation.

Reminder

- Learning problem:
  - Input data: $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n) \in \mathbb{R}^d \times \mathbb{R}^k$.
  - Approximator: f_w where w is the parameters of f. Find w such that f predicts exactly the labels of unseen data, i.e., $f_w(x_t) = y_t$ for unseen data point x_t.
  - Loss function $\ell(\cdot, \cdot)$.
  - We want $f_w(x_i)$ to be as close to y_i as possible for all i. Usually, we solve the optimization problem:

$$\min_w \frac{1}{n} \sum_{i=1}^{n} \ell(f_w(x_i), y_i) + \Omega(w),$$

    where $\Omega(w)$ is a regularization term on w. Functions ℓ, f_w and Ω are usually chosen convex to ease the optimization.

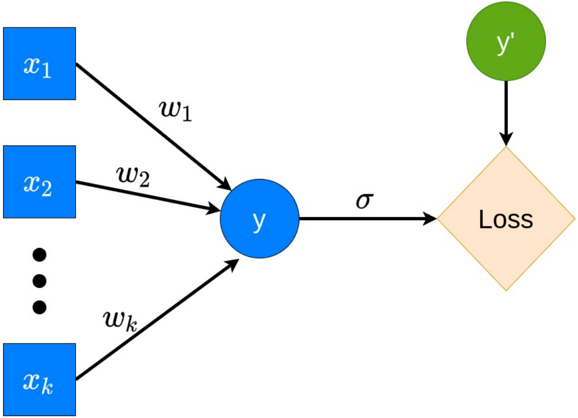
Reminder

- Linear regression:
  - $w = (w_0, w_1, ..., w_d)$ and $f_w(x) = w_1 x_1 + w_2 x_2 + ... + w_0$.
  - $l(y, y') = \|y - y'\|^2$.
- Logistic regression:
  - $w = (w_0, w_1, ..., w_d)$ and $f_w(x) = \sigma(w_1 x_1 + w_2 x_2 + ... + w_0)$ where $\sigma(x) = \frac{1}{1+\exp(-x)}$.
  - $l(y, y') = y' \log(y) + (1 - y') \log(1 - y')$.
  - .. or in another way: $f_w(x) = w_1 x_1 + w_2 x_2 + ... + w_0$ and $l(y, y') = y' \log(\sigma(y)) + (1 - y') \log(1 - \sigma(y))$.
- Support Vector Machine:
  - $w = (w_0, w_1, ..., w_d)$ and $f_w(x) = w_1 x_1 + w_2 x_2 + ... + w_0$.
  - $l(y, y') = \max(0, 1 - yy')$.

Reminder

- Linear regression, logistic regression, SVM use linear approximators $\rightarrow$ that's why they are called linear models.
- Linear models:
  - Pros: Simple, easy to optimize, guaranteed to have a unique solution.
  - Cons: Cannot approximate complex data distribution.
- Non-linearity can be obtained with feature engineering or kernel trick but they require domain specific knowledge. Even so, that is not enough in some complex domain (images, text, sound, ...).

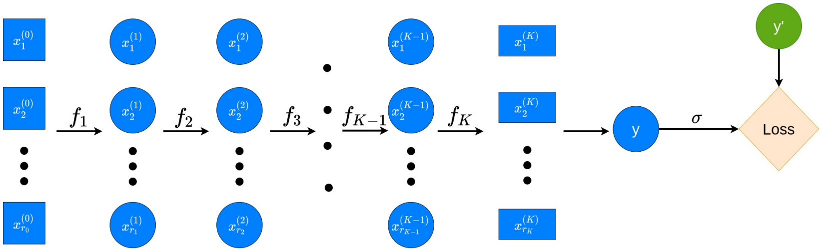
Graphical representation of linear models



Input data are transformed once.

Neural network

Input data are transformed multiple times.



Neural network

- A high-capacity model for machine learning
  - Can contain millions or billions of parameters.
  - Can approximate complicated data distribution.
  - Yielding state-of-the-art performances in many important problems.
  - Many variants: Multi-Layer Perceptron, Convolutional Neural Network, Recurrent Neural Network, ...
  - Deep learning: Branch of machine learning that studies neural networks and its applications.
- Training with Stochastic Gradient Descent using back-propagation.
- Applications: Image perception/generation, machine translation, speech recognition, autonomous driving, ...
- We investigate the simplest type of neural network, Multi-Layer Perceptron, in the next slides.

Multi-Layer Perceptron (MLP)

- Functional representation:
  - $f = f_K \circ f_{K-1} \circ ... \circ f_1$ or $f(x) = f_K(f_{K-1}(...(f_1(x))))$.
  - Denote $x^{(k)} = f_k(f_{k-1}(...(f_1(x))))$. We have:

$$x_k = f_k(x_{k-1})$$

.

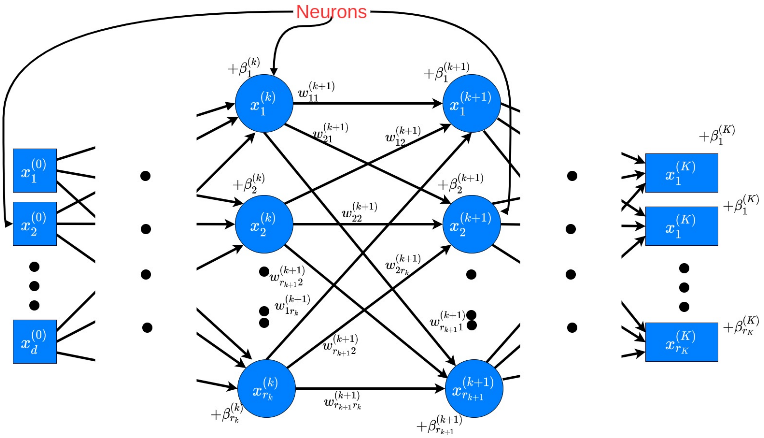
  - x_0 is the input layer, x_K is the output layer, x_k is the layer k of the neural network. x_k with $1 \leq k \leq K-1$ are **hidden layers**.
  - Usually $f_k(x) = \sigma_k(W^{(k)}x + \beta^{(k)})$ where $W^{(k)} \in \mathbb{R}^{r_k \times c_k}$ and $\beta^{(k)} \in \mathbb{R}^{r_k}$ and σ_k **is a point-wise non-linear activation function**. So:

$$x^{(k)} = \sigma(W^{(k)}x^{(k-1)} + \beta^{(k)}).$$

  - $W^{(k)}$ and $\beta^{(k)}$, $1 \leq k \leq K$, are the parameters of the neural network. K and r_k, $1 \leq k \leq K$ are hyper-parameters.
  - We must have $c_{k+1} = r_k$ for all $k \in [1..K-1]$ and $x^{(0)} \in \mathbb{R}^{c_1}$. **Why?**

Multi-Layer Perceptron (MLP)

- Graphical representation:

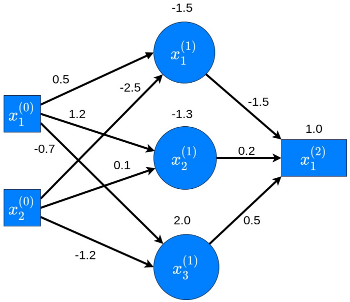


$$x^{(k+1)} = \sigma(W^{(k)}x^{(k)} + \beta^{(k+1)})$$

$$\Rightarrow x_i^{(k+1)} = \sigma(W_{i1}^{(k+1)}x_1^{(k)} + W_{i2}^{(k+1)}x_2^{(k)} + \ldots + W_{ir_k}^{(k+1)}x_{r_k}^{(k)} + \beta^{(k+1)}).$$

Multi-Layer Perceptron (MLP)

- Example: MLP with one hidden layer.



Compute the hidden layer and the output layer with $x^{(0)} = [0.5, -0.5]^T$ and $\sigma : x \mapsto \frac{1}{1+\exp(-x)}$.

Multi-Layer Perceptron (MLP)

- Example: MLP with one hidden layer.
  $x_1^{(1)} = \sigma(0.5 \times 0.5 + (-2.5) \times (-0.5) - 1.5) = 0.5$
  $x_2^{(1)} = \sigma(1.2 \times 0.5 + 0.1 \times (-0.5) - 1.3) = 0.3208$
  $x_3^{(1)} = \sigma((-0.7) \times 0.5 + (-1.2) \times (-0.5) + 2.0) = 0.9047$
  $x_1^{(2)} = \sigma((-1.5) \times 0.5 + 0.2 \times 0.3208 + 0.5 \times 0.9047 + 1.0) = 0.6828$
  Or

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} = \sigma\left(\begin{bmatrix} 0.5 & -2.5 \\ 1.2 & 0.1 \\ -0.7 & -1.2 \end{bmatrix} \begin{bmatrix} 0.5 \\ -0.5 \end{bmatrix} + \begin{bmatrix} -1.5 \\ -1.3 \\ 2.0 \end{bmatrix} \right) = \begin{bmatrix} 0.5 \\ 0.3208 \\ 0.9047 \end{bmatrix}$$

  and

$$x_1^{(2)} = \sigma\left(\begin{bmatrix} -1.5 & 0.2 & 0.5 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.3208 \\ 0.9047 \end{bmatrix} + 1.0 \right) = 0.6828.$$

Multi-Layer Perceptron (MLP)

- Training MLP:
  - Gradient descent: At iteration t, update parameters $W^{(k)}$ and $\beta^{(k)}$

$$W^{(k)} \longleftarrow W^{(k)} - \eta_w \frac{\partial l}{\partial W^{(k)}}$$

and

$$\beta^{(k)} \longleftarrow \beta^{(k)} - \eta_b \frac{\partial l}{\partial \beta^{(k)}}.$$

  - How to compute the gradients? $\longrightarrow$ Chain rule.
  - How to compute the gradients efficiently? $\longrightarrow$ Back-propagation.

Back-propagation

- Chain rule: $\dfrac{dz}{dx} = \dfrac{dz}{dy}\dfrac{dy}{dx}$.

$$\frac{\partial l}{\partial W^{(k)}} = \frac{dl}{dx_K}\frac{dx_K}{dx_{K-1}}\cdots\frac{\partial x_k}{\partial W^{(k)}} = \frac{dl}{dx_K}\frac{df_k(x_{K-1})}{dx_{K-1}}\cdots\frac{\partial f_k(x_{k-1})}{\partial W^{(k)}},$$
$$\frac{\partial l}{\partial \beta^{(k)}} = \frac{dl}{dx_K}\frac{dx_K}{dx_{K-1}}\cdots\frac{\partial x_k}{\partial \beta^{(k)}} = \frac{dl}{dx_K}\frac{df_k(x_{K-1})}{dx_{K-1}}\cdots\frac{\partial f_k(x_{k-1})}{\partial \beta^{(k)}},$$
$$\frac{df_k(x_{k-1})}{dx_{k-1}} = \frac{d\sigma(W^{(k)}x_{k-1}+\beta^{(k)}))}{dx_{k-1}} = \mathrm{diag}(\sigma'(W^{(k)}x_{k-1}+\beta^{(k)}))W^{(k)},$$
$$\frac{\partial f_k(x_{k-1})}{\partial W^{(k)}} = \frac{\partial\sigma(W^{(k)}x_{k-1}+\beta^{(k)}))}{\partial W^{(k)}} = \mathrm{diag}(\sigma'(W^{(k)}x_{k-1}+\beta^{(k)}))\mathbb{1}_{r_k}x_{k-1}^T,$$
$$\frac{\partial f_k(x_{k-1})}{\partial \beta^{(k)}} = \frac{\partial\sigma(W^{(k)}x_{k-1}+\beta^{(k)}))}{\partial \beta^{(k)}} = \mathrm{diag}(\sigma'(W^{(k)}x_{k-1}+\beta^{(k)}))\mathbb{1}_{r_k}.$$

- Compute the gradients separately is expensive $\Rightarrow$ compute from the last layer to the first layer, save intermediate results.

Back-propagation

- Example with one hidden MLP:
  - $l(x^{(2)}, y') = y' \log(x^{(2)}) + (1 - y') \log(1 - x^{(2)})$.
  - Let $y' = 1$, we have:

$$\frac{dl}{dx^{(2)}} = \frac{y'}{x^{(2)}} = \frac{1}{0.6828} = 1.4646.$$

$$\frac{dx^{(2)}}{dx^{(1)}} = \mathrm{diag}(\sigma'(W^{(2)}x_1 + \beta^{(2)}))W^{(2)} = [0.3249, -0.0433, -0.1083]$$

$$\frac{dx^{(1)}}{dx^{(0)}} = \mathrm{diag}(\sigma'(W^{(1)}x_0 + \beta^{(1)}))W^{(1)} = \begin{bmatrix} -0.125 & 0.625 \\ -0.2615 & -0.0218 \\ 0.0604 & 0.1035 \end{bmatrix}$$

Back-propagation

- Exercises:
  - Write code to reproduce the gradient computation above.
  - Compute the gradients with $\sigma : x \mapsto \max(x, 0)$.

Neural Network Applications

- Computer Vision:
- Natural Language Processing: