Tutorial on basic neural network

① Supervised Learning

- Learn a function from a family $\mathcal{F} = \{f_\theta : \theta \in \Theta\}$.

$$f_\theta : \mathcal{X} \mapsto \mathcal{Y} \qquad \text{E.g. } \mathcal{X} = \mathbb{R}^d,\ \mathcal{Y} = \{0,1\} \,/\, \mathbb{R}^{d_2} \,/ \dots$$

- Loss function $L(f_\theta(x), y)$:

$$\text{E.g. } L(f_\theta(x), y) = -y \log f_\theta(x) - (1-y)\log(1 - f_\theta(x))$$
$$L(f_\theta(x), y) = \| y - f_\theta(x) \|_2^2 .$$

- ERM:

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^n L(f_\theta(x_i), y_i) .$$

$\mathcal{F}$ = all the linear functions, i.e. $\{a^T x + b : a \in \mathbb{R}^d, b \in \mathbb{R}\}$

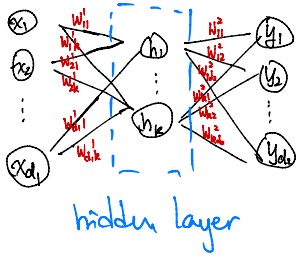
$L(f_\theta(x), y) = \| y - f_\theta(x) \|_2^2$

→ Linear regression

② Neural Networks — Multi-layer Perceptron (MLP)

- What is the $\mathcal{F}$?

$$f_\theta : \mathbb{R}^{d_1} \mapsto \mathbb{R}^{d_2}$$



For some nonlinear function $\sigma(\cdot) : \mathbb{R} \to \mathbb{R}$.

$$h_1 = \sigma\Big(\sum_{i=1}^{d_1} w_{i1}^1 x_i + b_1^1\Big), \dots, \quad h_k = \sigma\Big(\sum_{i=1}^{d_1} w_{ik}^1 x_i + b_k^1\Big),$$

$$y_1 = \sum_{j=1}^{k} w_{j1}^2 h_j + b_1^2 \cdots, \qquad y_{d_2} = \sum_{j=1}^{k} w_{jd_2}^2 h_j + b_{d_2}^2$$

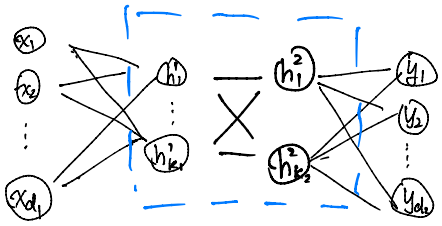
OR

$$h = \sigma(W^1 x + b^1), \qquad y = W^2 h + b^2 .$$

Popular choice of σ { sigmoid: $\sigma(x) = (1+e^{-x})^{-1}$
RELU: $\sigma(x) = \max\{0, x\}$.

$$\theta = (W^1, W^2, b^1, b^2) .$$

We can have more than one hidden layer:



- If we are dealing with a d_2-class classification problem, we can apply a softmax function to the output $y_1, \cdots, y_{d_2}$, i.e.

$$p_i = \frac{\exp(y_i)}{\sum_{j=1}^{d_2} \exp(y_j)}$$

Then, $(p_1, \cdots, p_{d_2})$ satisfies.

a). $p_i \geq 0$ for all $i \in \{1, \cdots, d_2\}$.

b). $\sum_{i=1}^{d_2} p_i = 1$

and therefore is a valid probability distribution.

What are the choices of L?

- Cross entropy loss.
(Maximum Likelihood).

$$L = \begin{cases} -\sum_{i=1}^{n} \left(y_i \log p_i + (1-y_i) \log (1-p_i) \right) & \text{(Binary).} \\ -\sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log p_{ik}. & \text{(General)} \end{cases}$$

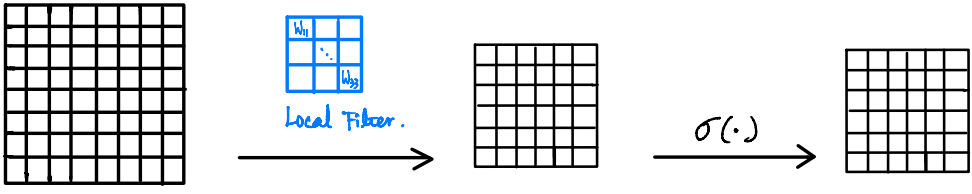
- Hinge Loss: $L = \sum_{i=1}^{n} \max(0, y_i - (1-2y_i) \cdot p_i)$.
- L_1 Loss, L_2 Loss, ...

③. Neural net – Convolution Neural Nets. (CNN).

Consider image input, with resolution 100 x 100 pixels.
If we use MLP with one hidden layer of dimension 20.
What is the number of parameter to learn?

- Slow Training
- Overfitting

For image data, local information is some time more useful.
Instead of training the full weight matrix: $\mathbb{R}^{10000} \to \mathbb{R}^{20}$,
we consider the local linear filter as follows.



a Convolutional layer

Following that, one can:

- apply another conv layer directly; or
- flatten the output to a vector and apply a fully connect layer.
- ...

There are small tricks to finetune the conv layer:

Zero padding, striding, ...

④. Train neural nets

Target: $\min_{\theta \in \mathcal{H}} L(f_\theta(x), y)$, $\theta = ((W^1, b^1), (W^2, b^2), \dots)$.

Method: Gradient descent or other optimization algorithms.

Implementation: PyTorch / Tensorflow.

They have great tutorials with codes.

Use pretrained model as feature extracters?

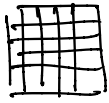


image → Pretrained model (e.g. ResNet, VGG-16) → $x \in \mathbb{R}^{d_1}$ → Your own model → $y \in \mathbb{R}^{d_2}$.

Pretrained model: Tons of Params. Very complex.

Your own model: much simpler.

Why is neural nets popular?

- Universal Approximation. MLP with a single hidden layer can approximate any continuous function in any compact set.
- Good empirical performance in many applications.

What are the shortcomings?

- Poor interpretability
- High computational costs.