High level neural network implementations

Axel Katona

September 27, 2023

General neural network - reminder

- X,Z
  - $dim(X) = N \times (R \times C) \rightarrow dim(X') = N \times (1 \times D)$
  - $dim(Z) = N \times (1 \times F)$
- Weights
  - Number of layers (L)
  - Number of neurons in layer l ($m^{(l)}$)
  - Weight vector of the neurons ($n^{(l)}$)
  - $dim(W^{(l)}) = m^{(l)} \times n^{(l)}$
  - Match the dimensions (matrix multiplication)!
- Biases
- Define cost function
- Define optimizer

Backropagation - reminder

- Minimizing the cost function: $C(f_{w,b}(X))$
  - $\nabla C = (\nabla_w C, \nabla_b C)$
  - Hardware likes low-level operators $(\cdot, +)$
- $\nabla \rightarrow$ matrix multiplication
- Steps

$$\begin{cases} A^{(l)} = \Phi\left(W^{(l)^T} A^{(l-1)} - B^l\right) \\ \frac{\partial C}{\partial W^{(l)}} = A^{(l-1)} \delta^{(l)^T} \\ \frac{\partial C}{\partial B^{(l)}} = -\delta^{(l)^T} \\ \delta^{(L)} = \left(Y - A^{(L)}\right) \odot \Phi'(Z^L) \\ \delta^{(l-1)} = \left(W^{(l)} \delta^{(l)}\right) \odot \Phi'(Z^{l-1}) \end{cases} \tag{0.1}$$

Overfitting

- Problem: you can overtrain network
  - $C_{Unknown}(t) > C_{Unknown}(t+n), n > 1$
  - $C_{train}(t) < C_{train}(t+n), n > 1$
  - Network develops memory instead of finding patterns
  - Idea: calculate accuracy on Unknown dataset after each BP

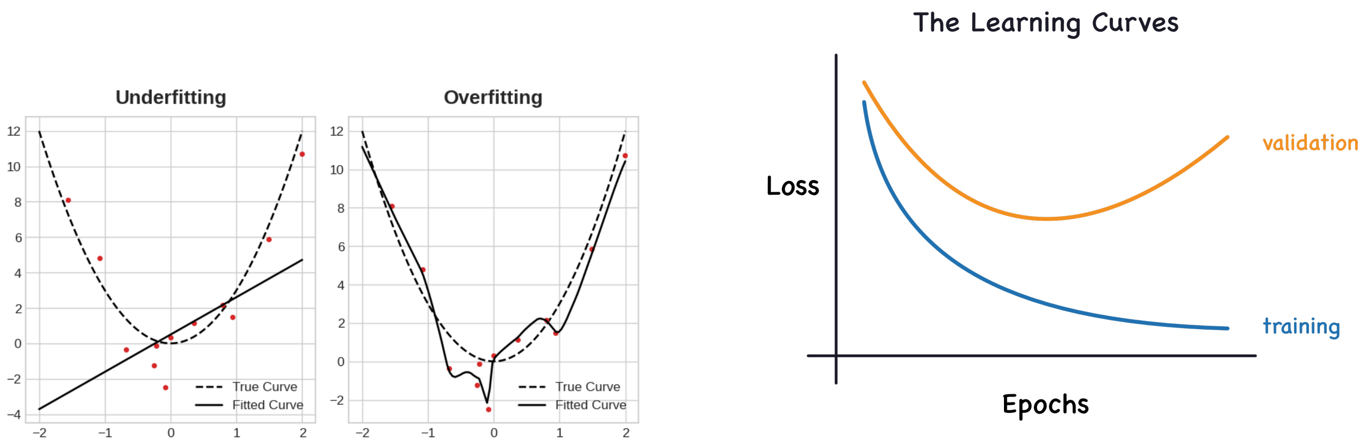

Figure 1: Source: Kaggle

Preparing data

- Always split your data into two (three) parts:
  - Training: used for weight updates (around $60 - 70\%$)
  - Validation: used for measuring accuracy (around 20%), after each BP steps (results testing accuracy)
  - Test: used after the training, unseen data (rest).
- Standardize your data

Confusion matrix

- What does accuracy mean?

Figure 2: Source: univ-angers

Cost function I

- What do we want?
  - Find the optimal w,b configuration
  - $X \to f_{w,b}(X)$ random variable
  - $\bigcup\limits_{w,b} \varepsilon_{w,b} = \varepsilon$ sigma-field
  - Best predictor?
- One candidate: MSE
  - (M,d)
  - $\langle X, Y \rangle = E(X, Y)$
  - $\{X, Y \in l^2\}$ means variance exists!
  - $d^2 = MSE$, M = Hilbert-space

Cross-entropy

- Another approach, working with pdf
- Value = surprise:

$$\text{Information} = \log\left(\frac{1}{q(\text{Event})}\right)$$

- From the prospect of $p(x)$, the information is

$$S(p,q) := -\int_{\mathbb{R}} p(x)\ln\left(q(x)\right)dx$$

  - Information assessed from a different distribution
  - $H(p) := S(p,p)$
  - $D_{KL}(p||q) := S(p,q) - H(p)$
- Minimizing KL-divergence is same as minimizing S(p,q)

Cost function II.

- How to use S(p,q) as a cost function?
- First, understand the problem in the context of neural networks.
- Input (joint) distribution: $p_{X,Y}(x,y) = \hat{p}(x,y)$
  - Control parameter: $\vartheta = \{w, b\}$
  - Output distribution: $p_\vartheta(\cdot|x)$
  - $\vartheta_{optimal} = \underset{\vartheta}{argmin}\, S(\hat{p}, p_\vartheta(y|x))$

$$C = S(\hat{p}, p_\vartheta(X,Y))$$

- For discrete variables

$$\vartheta_{opt}^{ML} = \vartheta_{opt}^{CE}$$

- Maximum Likelihood $\iff$ min(NLL) $\iff$ min(CE)
- For N=1, binary classes, $p \in \{y, 1-y\}$, $q \in \{\hat{y}, 1-\hat{y}\}$:

$$C = -\sum_i p_i \log(q_i) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

Categories

- Output (Y): categories, eg. *Cat*, *Dog*, *Horse*.
- Hardware doesn't like strings: $Y \to \mathbb{R}^n$
- n dimensional orthogonal vectors: One-hot encoding
  - $n = \#\text{categories}$ (3)
  - Every element of the vector is 0, except one index specified by the class
  - eg. Cat $= [1, 0, 0]$, Dog $= [0, 1, 0]$, Horse $= [0, 0, 1]$
- How to transform the output to probability?
- Softmax!

$$y_i(x) = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}$$

- bit.ly/3RuAODg

Regularization

- Model quickly overfits, if parameters can be arbitrarily large
- How to keep them bounded?
- Add constraint!
- L^2-regularization
  - $L_2(W) = C(W) + \lambda \|w\|_2^2$
  - λ small: weights don't count
  - λ large: weights do count, C(W) irrelevant
- L^1-regularization
  - $L_1(W) = C(W) + \lambda \|w\|_1$
  - λ small: weights don't count
  - λ large: weights do count, C(W) irrelevant
- Find λ?
  - Make it a hyper-parameter!
  - Train-test-validation split.

Optimizers I

- You want to find the local minima
- Gradient descent
- Take all values, and ”go” in the average direction
- Central Limit Theorem: Variance scales with N
- Stochastic Gradient Descent (SGD)
- Using Batches
- Momentum method
  - Add extra velocity to the system to shake out from local minima

$$\ddot{x}(t) = -\rho\dot{x}(t) - \nabla f(x(t)), \tag{0.2}$$

  - where f is the potential

Optimizers II

- Equation system:

$$\begin{cases} \dot{x}(t) = v(t) \\ \dot{v}(t) = -\rho v(t) - \nabla f(x(t)) \\ div(\dot{x}, \dot{v}) = -\rho < 0 \end{cases}$$

i.e. in phase space, the target flow shrinks and converging to the equilibrium.

- With constant time steps:

$$\begin{cases} x^{n+1} = x^n + \tilde{v}(t) \\ \tilde{v}^{n+1} = -\mu \tilde{v}^n - \eta \nabla f(x^n) \end{cases}$$

Optimizers III

- RMSProp

$$\begin{cases} x(t+1) = x(t) - \eta \frac{g_t}{\sqrt{|v(t)|}} \\ v(t) = \gamma v(t-1) + (1-\gamma) g_{t-1}^2, \end{cases}$$

  where $g_t = \nabla C(x(t))$, γ is forgetting factor. $v(t) \approx M(2)$ (second moment of gradient)
- Adam
  - Similar to RMSProp
  - Very powerful
  - Large scale models usually use this one
- Epoch = one training session

Introduction to tf, keras, torch in the notebook