

Basic Feedforward Network

Yingming Li
yingming@zju.edu.cn

Data Science & Engineering Research Center, ZJU

15th April 2018

Linear Classifier and the Perceptron Algorithm

- $f(x) = \sigma(w^T x + b)$
- Note: vector or matrix can be judged by context, e.g., here w and x is vector.
- σ : Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$
- The connection to logistic regression:
- Assume binomial distribution with parameter \hat{p}
- Assume the logit transform is linear:

$$\log \frac{\hat{p}}{1 - \hat{p}} = w^T x + b$$

$$\Rightarrow \hat{p} = \sigma(f(x))$$

Maximum Log-Likelihood

- MLE of the binomial likelihood:

$$\sum_{i=1}^n y_i^* \log \hat{p} + (1 - y_i^*) \log(1 - \hat{p})$$

where $y_i^* \in \{0, 1\} = \frac{1+y_i}{2}$

$$\log \hat{p} = -\log(1 + e^{-f(x)})$$

$$\log(1 - \hat{p}) = -\log(1 + e^{f(x)})$$

$$y_i^* \log \hat{p} + (1 - y_i^*) \log(1 - \hat{p}) = -\log(1 + e^{-yf(x)})$$

Gradient descent optimization

- Optimize w, b with gradient descent

$$\min_{w,b} \sum_i \log(1 + e^{-y_i(w^T x_i + b)})$$

$$\nabla w = \sum_i \frac{-y_i e^{-y_i(w^T x_i + b)}}{1 + e^{-y_i(w^T x_i + b)}} x_i = \sum_i -y_i^* (1 - \hat{p}(x_i)) - (1 - y_i^*) \hat{p}(x_i) x_i$$

$$\nabla b = \sum_i \frac{-y_i e^{-y_i(w^T x_i + b)}}{1 + e^{-y_i(w^T x_i + b)}}$$

XOR problem and linear classifier

- 4 points: $X = [(-1, -1), (-1, 1), (1, -1), (1, 1)]$
- $Y = [-1, 1, 1, -1]$
- Try using binomial log-likelihood loss:

$$\min_w \sum_i \log(1 + e^{-y_i(w^T x_i + b)})$$

- Gradient:

$$\nabla w = \sum_i \frac{-y_i e^{-y_i(w^T x_i + b)}}{1 + e^{-y_i(w^T x_i + b)}} x_i$$

$$\nabla b = \sum_i \frac{-y_i e^{-y_i(w^T x_i + b)}}{1 + e^{-y_i(w^T x_i + b)}}$$

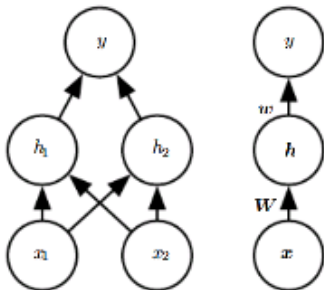
- Try $w = 0, b = 0$, what do you see?

With 1 hidden layer

- A hidden layer makes a nonlinear classifier

$$f(x) = w^T g(W^T x + c) + b$$

- g needs to be nonlinear -
- Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$ -
- RELU: $g(x) = \max(0, x)$



Taking gradient

$$\min_{W,w} E(f) = \sum_i L(f(x_i), y_i)$$

$$f(x) = w^T g(W^T x + c) + b$$

- What is $\frac{\partial E}{\partial W}$?
- Consider chain rule: $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$

Note: Vectorized Computations

- The computations performed by a network.

$$z_i = \sum_j W_{ij} x_j$$

$$h_i = \sigma(z_i)$$

$$y = \sum_i v_i h_i$$

- Write them in terms of matrix and vector operations.
- Note: judge vector or matrix by context

$$z = Wx$$

$$h = \sigma(z)$$

$$y = v^T h$$

- Where $\sigma(v)$ denote the logistic sigma function applied elementwise to a vector v . Let W be a matrix where the (i, j) entry is the weight from visible unit j to hidden unit i .

Backpropagation

- Save the gradients and the gradient products that have already been computed to avoid computing multiple times

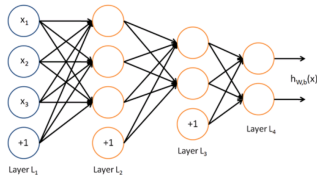
- In a multiple layer network (Ignore constant terms)

$$f(x) = w_n^T g(W_{n-1}^T g(W_{n-2}^T g(W_1^T g(x))))$$

$$\begin{aligned}\frac{\partial E}{\partial W_k} &= \frac{\partial E}{\partial f_k} g(f_{k-1}(x)) \\ &= \frac{\partial E}{\partial f_{k+1}} \frac{\partial f_{k+1}}{\partial f_k} g(f_{k-1}(x))\end{aligned}$$

- Define:

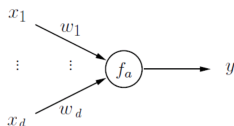
$$f_k(x) = w_k^T g(f_{k-1}(x)), f_0(x) = x$$



Modules

- Each layer can be seen as a module
- Given input, return

- Output $f_a(x)$
- Network gradient $\frac{\partial f_a}{\partial x}$
- Gradient of module parameters $\frac{\partial f_a}{\partial w_a}$



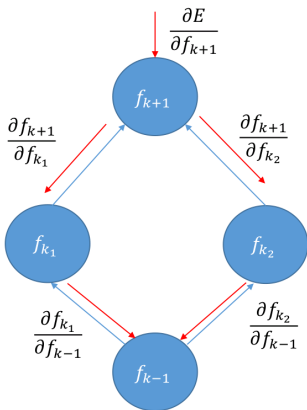
- During backprop, propagate/update
- Backpropagated gradient $\frac{\partial E}{\partial f_a}$

$$\frac{\partial E}{\partial W_k} = \frac{\partial E}{\partial f_k} g(f_{k-1}(x)) = \frac{\partial E}{\partial f_{k+1}} \frac{\partial f_{k+1}}{\partial f_k} g(f_{k-1}(x))$$

- Three term above are respectively Backprop signal; Network Gradient; gradient of parameters
- Note: $\frac{\partial E}{\partial f_k} = \frac{\partial E}{\partial f_{k+1}} \frac{\partial f_{k+1}}{\partial f_k}$

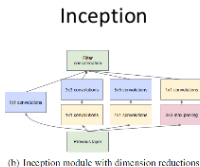
Multiple Inputs and Multiple Outputs

$$\frac{\partial E}{\partial f_{k-1}} = \frac{\partial E}{\partial f_{k+1}} \frac{\partial f_{k+1}}{\partial f_{k_1}} \frac{\partial f_{k_1}}{\partial f_{k-1}} + \frac{\partial E}{\partial f_{k+1}} \frac{\partial f_{k+1}}{\partial f_{k_2}} \frac{\partial f_{k_2}}{\partial f_{k-1}}$$

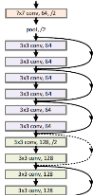


Different DAG structures

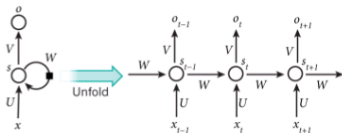
- The backpropagation algorithm would work for any DAGs
- So one can imagine different architectures than the plain layerwise one



Residual



RNN

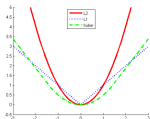


Loss functions

- Regression:
- Least squares $L(f) = (f(x) - y)^2$
- L1 loss $L(f) = |f(x) - y|$
- Huber loss

$$L(f) = \begin{cases} \frac{1}{2}(f(x) - y)^2 & , |f(x) - y| \leq \delta \\ \delta(|f(x) - y| - \frac{1}{2}\delta) & , \text{otherwise} \end{cases}$$

- Binary Classification
- Hinge loss $L(f) = \max(1 - yf(x), 0)$
- Binomial log-likelihood $L(f) = \ln(1 + \exp(-2yf(x)))$
- Cross-entropy $L(f) = -y^* \ln \sigma(f) - (1 - y^*) \ln(1 - \sigma(f))$,
- $y^* = (y + 1)/2$



Multi-class: Softmax layer

- Multi-class logistic loss function

$$P(y = j|x) = \frac{e^{x^T w_j}}{\sum_{k=1}^K e^{x^T w_k}}$$

- {Log}-likelihood:
- Loss function is minus log-likelihood

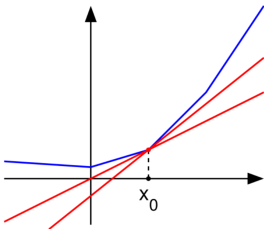
$$-\log P(y = j|x) = -x^T w_j + \log \sum_k e^{x^T w_k}$$

Subgradients

- What if the function is non-differentiable?
- Subgradients:
- For convex $f(x)$ at x_0 :
- If for any y

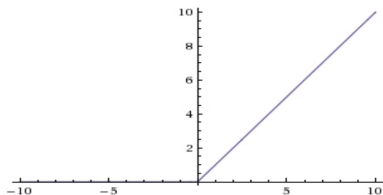
$$f(y) \geq f(x) + g^T(y - x)$$

- g is called a subgradient
- Subdifferential: ∂f : set of all subgradients
- Optimality condition: $0 \in \partial f$



The RELU unit

- $f(x) = \max(x, 0)$
- Convex
- Non-differentiable
- Subgradient: $\frac{\partial f}{\partial x} = \begin{cases} 1 & , x > 0 \\ [0, 1] & , x = 0 \\ 0 & , x < 0 \end{cases}$



Subgradient descent

- Similar to gradient descent

$$x^{(k+1)} = x^{(k)} - \alpha_k g^{(k)}$$

- Step size rules:
- Constant step size: $\alpha_k = \alpha$
- Square summable: $\alpha_k \geq 0, \sum_{k=1}^{\infty} \alpha_k^2 < \infty, \sum_{k=1}^{\infty} \alpha_k = \infty$
- Usually, a large constant that drops slowly after a long while . e.g. $\frac{100}{100+k}$

Universal Approximation Theorems

- Many universal approximation theorems proved in the 90s
- Simple statement: for every continuous function, there exist a function that can be approximated by a 1-hidden layer neural network with arbitrarily high precision

Formal statement [\[edit \]](#)

The theorem^{[2][3][4][5]} in mathematical terms:

Let $\varphi(\cdot)$ be a nonconstant, [bounded](#), and [monotonically-increasing](#) [continuous](#) function. Let I_m denote the m -dimensional [unit hypercube](#) $[0, 1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, given any function $f \in C(I_m)$ and $\varepsilon > 0$, there exists an integer N and real constants $v_i, b_i \in \mathbb{R}$, where $i = 1, \dots, N$ such that we may define:

$$F(x) = \sum_{i=1}^N v_i \varphi(w_i^T x + b_i)$$

as an approximate realization of the function f where f is independent of φ ; that is,

$$|F(x) - f(x)| < \varepsilon$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are [dense](#) in $C(I_m)$.

It obviously holds replacing I_m with any compact subset of \mathbb{R}^m .

Universal Approximation Theorems

- The approximation does not need many units if the function is kinda nice. Let

$$C_f = \int_{R_d} ||\omega|| |\tilde{f}(\omega)| d\omega$$

- Then for a 1-hidden layer neural network with n hidden nodes, we have for a finite ball with radius r ,

$$\int_{B_r} (f(x) - f_n(x))^2 d\mu(x) \leq \frac{4r^2 C_f^2}{n}$$

Thank you!