Loss Functions and Backpropagation

Yao Zhang

The guy is a populace

Mostly based on Thomas Hofmann's lecture in ETH

https://zhims.github.io/

Dec 1, 2019

Reminder: Notation

- Neural networks implements map $F : \mathbb{R}^n \to \mathbb{R}^m$
- Compositional structure layers:

$$F = F^L \circ F^{L-1} \circ \cdots F^1 \tag{1}$$

- Linear + activation function

$$F^l = \sigma^l \circ \overrightarrow{F}^l, \quad \overrightarrow{F}^l(x) = W^l x + b^l, \quad l = 1, ..., L \tag{2}$$

- F minus output layer non-linearity

$$\overline{F} = \overline{F}^L \circ F^{L-1} \circ \cdots \circ F^1 \tag{3}$$

Loss Function

For learning, we need to assess the goodness-of-fit of network.

Definition 1 (Loss function)

A loss (or cost) function is non-negative function

$$\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}, \quad (y, \nu) \mapsto \ell(y, \nu) \tag{4}$$

such that $\ell(y, y) = 0$ $(\forall y \in \mathcal{Y})$ and $\ell(y, \nu) > 0$ $(\forall \nu \neq y)$.

1. here: $\mathcal{Y}$: output space
2. general convention: y is the truth and ν predicted

Loss Function: Examples

Example 1 (Squared-error)

$$\mathcal{Y} = \mathbb{R}^m,\ \ell(y,\nu) = \frac{1}{2}\|y-\nu\|_2^2 = \frac{1}{2}\sum_{i=1}^{m}(y_i-\nu_i)^2 \tag{5}$$

Example 2 (Classification error)

$$\mathcal{Y} = [1:m],\ \ell(y,\nu) = 1-\delta_{y\nu} \tag{6}$$

with Kronecker delta:

$$\delta_{ab} = \begin{cases} 1, & \text{if } a=b \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Expected Risk

Definition 2 (Expected Risk)

Assume inputs and outputs are governed by a distribution $p(x, y)$ over $\mathcal{X} \times \mathcal{Y}$, $\mathcal{X} \subseteq \mathbb{R}^n$, $\mathcal{Y} \subseteq \mathbb{R}^m$. The expected risk of F is given by

$$\mathcal{R}^\star(F) = E_{x,y}[\ell(y, F(x))] \tag{8}$$

1. as p is generally unknown, we cannot evaluate $\mathcal{R}^\star$ directly, but it serves as a point of reference in learning theory
2. $\mathcal{R}^\star$ is a functional (mapping functions to scalars)
3. parameterized functions $\{F_\theta : \ \theta \in \Theta\} \Rightarrow \mathcal{R}^\star(\theta) \triangleq \mathcal{R}^\star(F_\theta)$

Empirical Risk

Definition 3 (Empirical Risk)

Assume we have a random sample of N input-output pairs,

$$\mathcal{S}_N \triangleq \left\{ (x_i, y_i) \overset{i.i.d.}{\sim} p \ : \ 1, \ldots, N \right\}. \tag{9}$$

The empirical risk of F is defined as

$$\mathbb{R}(F, \mathcal{S}_N) = \frac{1}{N} \sum_{i=1}^{N} \ell\left(y_i, F\left(x_i\right)\right) \tag{10}$$

1. a.k.a. training risk = expected risk under the empirical distribution induced by the sample $\mathcal{S}_N$.

Empirical Risk Minimization

For a family $\mathcal{F} = \{F_\theta : \ \theta \in \Theta\}$ (e.f. neural network) and training data $\mathcal{S}_N$: find function with lowest empirical risk.

Definition 4 (Empirical risk minimization)

The empirical risk minimizer is defined as

$$\widehat{F}\left(\mathcal{S}_N\right) \in \underset{F \in \mathcal{F}}{\arg\min}\, \mathcal{R}\left(F, \mathcal{S}_N\right) \tag{11}$$

with the corresponding parameters $\widehat{\theta}\left(\mathcal{S}_N\right)$.

1. one may also add a regularizer $\Omega(F)$ or $\Omega(\theta)$ to the risk (more on that later)
2. finding $\widehat{F} \in \mathcal{F}$ amounts to solving on optimization problem

Probability Distributions as Outputs

It is often constructive to think of functions F as mappings from inputs to distribution $\mathcal{P}(\mathcal{Y})$ over outputs $y \in \mathcal{Y}$.

$$F : \mathbb{R}^n \to \mathbb{R}^m, \quad x \mapsto \nu, \quad \nu \overset{\textit{fixed}}{\mapsto} p(y, \nu) \in \mathcal{P}(\mathcal{Y}), \quad y \sim p(\cdot, \nu) \tag{12}$$

Each F effectively defines a conditional probability distribution (or conditional probability density function) via

$$p\left(y|x,\ F\right) = p\left(y,\ \nu = F\left(x\right)\right) \tag{13}$$

Example: Multivariate Normal Distribution

Example 3 (mean of a normal distribution)

$$p\left(y|x,\ F\right)=\left[\frac{1}{\sqrt{2\pi\gamma}}\right]^{m}e^{\left[-\frac{1}{2\gamma^2}\left\|y-F(x)\right\|^2\right]} \tag{14}$$

so that

$$-\log\ p\left(y|x,\ F\right)=mC\left(\gamma\right)\ +\ \frac{1}{2\gamma^2}\left\|y-F\left(x\right)\right\|^2 \tag{15}$$

which is equivalent to the squared error loss.

1. $F\left(x\right)=\nu$ and y live in same space ($\mathbb{R}^m$)

Generalized Linear Models

Definition 5 (Generalized linear model (simplified))

A generalized linear model over $y \in \mathcal{Y} \subseteq \mathbb{R}$ takes the form

$$E\left[y|x\right] = \sigma\left(w^T x\right). \tag{16}$$

where σ is invertible and σ^{-1} is called the link function.

1. can be extended to also predict variances or dispersions
2. can be extended to multidimensional outputs

Example: Logistic Regression

Example 4 (Logistic regression)

$\mathcal{Y} = \{0, 1\}, \mathcal{P} = [0, 1], \sigma = \frac{1}{1+e^{-x}}$, *then:*

$$E\left[y|x\right] = p\left(1|x\right) = \sigma\left(w^T x\right) = \frac{1}{1 + e^{-w^T x}} \tag{17}$$

Link function: logit

$$\sigma^{-1}\left(t\right) = \log\left(\frac{t}{1-t}\right), \quad t \in (0, 1) \tag{18}$$

Example: Multinomial Logistic Regression

Example 5

$\mathcal{Y} = [1 : m], \mathcal{P}(\mathcal{Y})$ *can be represented via soft-max*

$$p\left(y|x\right) = \frac{e^{z_y}}{\sum_{i=1}^{m} e^{z_i}}, \quad z \triangleq {w_i}^T x, \quad i = 1, ..., m \tag{19}$$

1. over-parametrized model: set $w_1 = 0$, s.t. $z_1 = 0$ (w.l.o.g)
2. generalizes (binary) logistic regression

Generalized Linear Units

In neural networks:

- non-linear functions replace linear functions
- output layer units implement inverse link function

Example 6 (Normal model)

Linear output layer

$$E[y|x] = \overline{F}(x) = W^L \left(F^{L-1} \circ \cdots \circ F^1 \right)(x) + b^L \tag{20}$$

Example 7 (Logistic model)

Sigmoid output layer

$$E[y|x] = \sigma \left(\overline{F}(x) \right) \tag{21}$$

Log-Likelihood

Use conditional probability distribution to define generalized loss between target value $y \in \mathcal{Y}$ and a distribution over $\mathcal{Y}$.

Definition 6 (Negative log-loss)

Canonical way of defining a generalized loss functions: negative of a log-likelihood function

$$\ell(y, \theta, x) = -\log\ p(y|x,\ \theta) \tag{22}$$

1. non-linearity of output layer is "absorbed" in loss function
2. i.e. ℓ depends on $\overline{F}$
3. provides a "template" for generalized loss/risk functions

Cross-Entropy Loss

Let us look at the (implied) risk function for the logistic function

Definition 7 (Cross-entropy Loss)

Use shorthand $z \triangleq \overline{F}(x) \in \mathbb{R}$ then the cross entropy loss over a binary response variable $y \in \{0, 1\}$ is defined as

$$\begin{aligned} -\log\ p(y|z) &= -\log \sigma((2y-1)z) \\ &= \zeta((1-2y)z) \end{aligned} \tag{23}$$

where $\zeta = log(1 + e^{(\cdot)})$ is the soft-plus function.

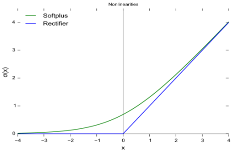

Figure 1: rectifier and softplus functions

Multinomial Log-Likelihood

Definition 8 (Multinomial cross-entropy loss)

Assume multinomial response variable $y \in [1 : m]$. Use shorthand:

$$z \triangleq \overline{F}(x) \in \mathbb{R}^m \tag{24}$$

then with the soft-max activation function

$$\begin{aligned} \ell\left(y, \overline{F}(x)\right) &= -\log\ p\left(y | \overline{F}(x)\right) = -\log\left[\frac{e^{z_y}}{\sum\limits_{i=1}^{m} e^{z_i}}\right] \\ &= -z_y + \underbrace{\log \sum_{i=1}^{m} e^{z_i}}_{\log -partition} = \log\left[1 + \sum_{i \neq y} e^{(z_i - z_y)}\right] \end{aligned} \tag{25}$$

Gradient Descent

Learning in neural networks = gradient-based optimization (with very few exceptions).

Definition 9 (Gradient)

Gradient of objective with regard to parameters θ

$$\nabla_\theta = \left(\frac{\partial \mathcal{R}}{\partial \theta_1}, ..., \frac{\partial \mathcal{R}}{\partial \theta_d} \right)^T \tag{26}$$

Definition 10 (Steepest descent and stochastic gradient decent)

Steepest descent and stochastic gradient decent

$$\theta(t+1) \leftarrow \theta(t) - \eta \nabla_\theta \mathcal{R}(\mathcal{S}) \tag{27}$$

1. here $t = 0, 1, 2, ...$ is an iteration index
2. $\mathcal{S}$ =all training data $\Rightarrow$ steepest descent
3. $\mathcal{S}$ = mini batch of data $\Rightarrow$ SGD

Gradient Computation via Backpropagation

Computational challenge: how to compute $\nabla_\theta \mathcal{R}$?
Exploit compositional structure of network = backpropagation
Basic steps:

1. perform a forward pass (for given training input x) to compute activations for all units
2. compute gradient of $\mathcal{R}$ w.r.t. output layer activations (for given target y)
3. iteratively propagate activation gradient information from outputs to inputs
4. compute local gradients of activations w.r.t weights

Backpropagation in Plain English

1. How do changes in the output layer activities change the objective?
   - depends on choice of objective
2. How does the activity of a parent unit influence the activity of each of its child units (in DAG)?
   - layer structure $\Rightarrow$ concurrently between subsequent layers
3. Propagate influence information through reverse DAG
   - details are implied by chain rule of differentiation
4. What is the effect of a change of an incoming weight on the activity of a unit?
   - can only change activities (given x) by modifying weights

Chain Rule

Compositional of functions $\Rightarrow$ use of chain rule

Proposition 1 (Chain Rule)

$$(f \circ g)' = (f' \circ g) \cdot g' \tag{28}$$

or equivalently with formal variables

$$\frac{d\,(f \circ g)}{dx}\Big|_{x=x_0} = \frac{df}{dz}\Big|_{z=g(x_0)} \cdot \frac{dg}{dx}\Big|_{x=x_0} \tag{29}$$

Jacobi Matrix

Vector-valued function (map) $F : \mathcal{R}^n \to \mathcal{R}^m$: each component function has gradient $\nabla F_i \in \mathcal{R}^n$, $i \in [1:m]$

Definition 11 (Jacobi matrix)

$$J_F \triangleq \begin{pmatrix} \nabla^T F_1 \\ \nabla^T F_2 \\ \vdots \\ \nabla^T F_m \end{pmatrix} = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1} & \frac{\partial F_m}{\partial x_2} & \cdots & \frac{\partial F_m}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{m \times n} \tag{30}$$

derivative of outputs with regard to inputs, i.e. $(J_F)_{ij} = \frac{\partial F_i}{\partial x_j}$.

Jacobin Matrix Chain Rule

Vector-valued functions $G : \mathbb{R}^n \to \mathbb{R}^q,\ H : \mathbb{R}^q \to \mathbb{R}^m, F \triangleq H \circ G$.
Componentwise rule

$$\frac{\partial F_i}{\partial x_j}\Big|_{x=x_0} = \frac{\partial (H \circ G)_i}{\partial x_j}\Big|_{x=x_0} = \sum_{k=1}^{q} \frac{\partial H_i}{\partial z_k}\Big|_{z=G(x_0)} \cdot \frac{\partial G_k}{\partial x_j}\Big|_{x=x_0} \tag{31}$$

Lemma 1 (Jacobi matrix chain rule)

$$J_{H\circ G}\big|_{x=x_0} = J_H\big|_{z=G(x_0)} \cdot J_G\big|_{x=x_0} \tag{32}$$

Function Composition

Special case: composition of a map with a function

$$G : \mathbb{R}^n \to \mathbb{R}^m, \quad h : \mathbb{R}^m \to \mathbb{R}, \quad h \circ G \ : \ \mathbb{R}^n \to \mathbb{R} \tag{33}$$

$x \in \mathbb{R}^n$, use more intuitive variable notation

$$x \overset{G}{\mapsto} y \overset{h}{\mapsto} z \in \mathbb{R} \tag{34}$$

Then

$$\nabla_x^T z = \nabla_y^T z \cdot J_G, \quad \frac{\partial z}{\partial x_i} = \sum_j \frac{\partial y_j}{\partial x_i} \frac{\partial z}{\partial y_j} \tag{35}$$

Warning: Notation!

We have a lot of indices!

- index of a layer: put as a superscript
- index of a dimension of a vector: put as a subscript
- shorthand for layer activations

$$
\begin{aligned}
&x^l \triangleq \left(F^l \circ \cdots \circ F^1\right)(x) \in \mathbb{R}^{m_l} \\
&x_i^l \in \mathbb{R} : \; activation \;\; of \;\; i - th \;\; unit \;\; in \;\; layer \;\; l
\end{aligned} \tag{36}
$$

- index of a data point, omitted where possible, rectangular brackets $(x[i], y[i])$

Deep Function Compositions

Composition of multiple maps with a final cost function

$$
\begin{aligned}
F &= F^L \circ \cdots \circ F^1 : \quad \mathbb{R}^n \to \mathbb{R}^m \\
x &= x^0 \overset{F^1}{\mapsto} x^1 \overset{F^2}{\mapsto} x^2 \mapsto \cdots \overset{F^L}{\mapsto} x^L = \nu \mapsto \ell(y, \nu)
\end{aligned} \tag{37}
$$

Proposition 2 (Activity Backpropagation)

$$
e^L \triangleq \nabla_\nu^T \mathcal{R}, \quad e^l \triangleq \nabla_{x^l}^T \mathcal{R} = e^L \cdot J_{F^L} \cdots \ J_{F^{l+1}} = e^{l+1} \cdot J_{F^{l+1}} \tag{38}
$$

Compute activity gradients is backward order via successive multiplication with Jacobians. Backpropagation of error terms e^l.
Linear nrtwork in reversed direction with "activities" e^l.

Jacobian Matrix: Ridge Functions

How does a Jacobian matrix for a ridge function look like?

$$x^l = F^l\left(x^{l-1}\right) = \sigma\left(W^l x^{l-1} + b^l\right) \tag{39}$$

Hence (assuming differentiability of σ):

$$\frac{\partial x_i^l}{\partial x_j^{l-1}} = \sigma'\left(\left\langle w_i^l, x^{l-1}\right\rangle + b_i^l\right) w_{ij}^l \triangleq \widetilde{w}_{ij}^l \tag{40}$$

and thus simply

$$J_{F^l} = \widetilde{W}^l \tag{41}$$

1. for ReLU $\widetilde{w}_{ij}^l \in \left\{0, w_{ij}^l\right\} \Rightarrow \widetilde{W}^l =$ sparsified matrix

Loss Function (Negative) Gradients

Quadratic loss

$$-\nabla_\nu \ell(y,\nu) = -\nabla_\nu \frac{1}{2}\|y-\nu\|^2 = y - \nu \tag{42}$$

Multivariate logistic loss

$$\begin{aligned}
-\frac{\partial \ell(y,\nu)}{\partial z_y} &= \frac{\partial}{\partial z_y}\left[z_{y^*} - \log \sum_i e^{z_i}\right] \\
&= \delta_{yy^*} - \frac{e^{z_y}}{\sum_i e^{z_i}} \\
&= \delta_{yy^*} - p(y|x)
\end{aligned} \tag{43}$$

From Activations to Weights

How can we get from gradients w.r.t. activations to gradients w.r.t. weights? Easily!
Need to apply chain rule one more time- locally:

$$\frac{\partial \ell}{\partial w_{ij}^l} = \frac{\partial \ell}{\partial x_i^l} \cdot \frac{\partial x_i^l}{\partial w_{ij}^l} = \underbrace{\frac{\partial \ell}{\partial x_i^l}}_{backprop} \cdot \underbrace{\sigma' \left(\left\langle w_i^l, x^{l-1} \right\rangle + b_i^l \right)}_{sensitivity \;\; of \;\; i-th \; unit} \cdot \underbrace{x_j^{l-1}}_{j-th \; unit \; activity} \tag{44}$$

$$\frac{\partial \ell}{\partial b_i^l} = \frac{\partial \ell}{\partial x_i^l} \cdot \frac{\partial x_i^l}{\partial b_i^l} = \frac{\partial \ell}{\partial x_i^l} \cdot \sigma' \left(\left\langle w_i^l, x^{l-1} \right\rangle + b_i^l \right) \cdot 1$$

- each weight/bias influences exactly one unit
- can "reshape" gradient into matrix/tensor form

Specialized Programming Languages: Theano

Symbolic representation of mathematical expressions.

Access to full computation graph (stability, optimization).

Symbolic differentiation.

[Bergstra 2015]

reference

J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio (2010)
Theano: A CPU and GPU Math Compiler in Python
9th Annual Python In Science Conference (SciPy 2010)

Thank you all of you! –Yao