CMPT412 – Computer Vision
Training Losses

Andrea Tagliasacchi

Associate Professor	Simon Fraser University
Associate Professor (status only)	University of Toronto
Staff Research Scientist	Google DeepMind

Parametric Model

Let us reason probabilistically about a (parametric) **model** f:

$$\hat{y} = f(x; \theta)$$

- with (unknown) parameters θ
- where $\hat{y}$ denotes the model prediction
- and our data is pairs (x, y) of input x and corresponding output y

Remark

As θ captures *all* the information to predict $\hat{y}$ from x, $\Rightarrow p(y|\hat{y}) = p(y|\theta)$.
Henceforth, we will use this notation interchangeably.

Bayesian inference

Recall: Bayes' theorem

$$\begin{cases} p(\theta, y) = p(\theta|y)\, p(y) \\ p(y, \theta) = p(y|\theta)\, p(\theta) \end{cases} \quad \underset{p(\theta,y)\equiv p(y,\theta)}{\Longrightarrow} \quad p(\theta|y) = \frac{p(y|\theta)\, p(\theta)}{p(y)}$$

- $p(y)$: *evidence* — normalizes to ensure probabilities sum to one
- $p(y|\theta)$: *likelihood* — how likely we observe y when model parameters are θ
- $p(\theta|y)$: *posterior* — how likely are the parameters θ given the data y

Our objective

As we are interested in finding θ, we are interested in maximizing the *posterior*.

Maximum Likelihood Estimation (MLE)

- We can recover θ by maximizing the *posterior*:

$$\underset{\theta}{\operatorname{argmax}}\ p(\theta|y) \equiv \underset{\theta}{\operatorname{argmax}}\ \frac{p(y|\theta)\, p(\theta)}{p(y)} \equiv \underset{\theta}{\operatorname{argmax}}\ \ p(y|\theta)\, p(\theta)$$
$$\equiv \underset{\theta}{\operatorname{argmax}}\ \ p(y|\theta) \equiv \underset{\theta}{\operatorname{argmin}}\ \ -log(p(y|\theta))$$

- Where (for now) above we assumed we have *no knowledge* about the optimal θ, and therefore $p(\theta)$ is a constant from out point of view.

General recipe

To optimize parameters, minimize the <u>negative log likelihood</u> $-log(p(y|\theta))$

What if we consider multiple data points?

To simplify our model, we assume the data is i.i.d.

- identically distributed: all data explained by the same distribution $p(.)$
- independently distributed: joint probability factorizes $p(a, b) = p(a) \cdot p(b)$

$$
\begin{aligned}
\theta^* &= \underset{\theta}{\operatorname{argmax}} \;\; p(y_1, ..., y_N | \hat{y}_1, ..., \hat{y}_N) \\
&= \underset{\theta}{\operatorname{argmax}} \;\; \prod_n p(y_n | \hat{y}_n) = \underset{\theta}{\operatorname{argmax}} \;\; \log \left(\prod_n p(y_n | \hat{y}_n) \right) \\
&= -\underset{\theta}{\operatorname{argmin}} \sum_n \log(p(y_n | \hat{y}_n)) = -\underset{\theta}{\operatorname{argmin}} \sum_n \log(p(y_n | \theta))
\end{aligned}
$$

Why do we use the log?

Finite precision and large cardinality products of (small) numbers is a bad idea.

The (discrete) Kullback–Leibler divergence

The Kullback–Leibler (KL) divergence (also called relative entropy denoted $D_{\mathsf{KL}}(A \parallel B)$, is a type of statistical distance: a measure of how much a model probability distribution A is different from a distribution B:

$$\mathsf{KL}\big[A \parallel B\big] = \sum_{y \in \mathcal{Y}} A(y) \, \log \left(\frac{A(y)}{B(y)} \right)$$

- Let us consider $A(y) = p(y)$ to be our empirical data distribution, and $B(y) = p(y|\theta)$ the distribution of predictions from our network
- We then seek θ that **minimizes** the divergence between the two distributions

Recall: logarithm properties

$$log(A/B) = log(A) - log(B)$$

$$log(A^b) = b \cdot log(A)$$

$$\begin{aligned}
\underset{\theta}{\operatorname{argmin}}\ \mathsf{KL}\big[p(y) \parallel p(y|\theta)\big] &\equiv \underset{\theta}{\operatorname{argmin}} \sum_{y \in \mathcal{Y}} p(y)\ \log\left(\frac{p(y)}{p(y|\theta)}\right) \\
&\equiv \underset{\theta}{\operatorname{argmin}} \sum_{y \in \mathcal{Y}} \left[p(y)\ \log\left(p(y)\right) - p(y)\ \log\left(p(y|\theta)\right)\right] \\
&\equiv \underset{\theta}{\operatorname{argmin}}\ -\sum_{x \in \mathcal{X}} p(y)\ \log\left(p(y|\theta)\right)
\end{aligned}$$

- Now let us model our data distribution as a collection of discrete elements:

$$p(y) = \frac{1}{N}\sum_n \delta(y - y_n) \quad \text{where the dirac} \quad \delta(x) = \begin{cases} 0 & \text{otherwise} \\ 1 & x = 0 \end{cases}$$

- Continuing from the previous slide:

$$\equiv \underset{\theta}{\operatorname{argmin}} \; -\frac{1}{N}\sum_n \sum_{y \in \mathcal{Y}} \delta(y - y_n)\, \log\left(p(y|\theta)\right) \equiv \underset{\theta}{\operatorname{argmin}} \; -\sum_n \log\left(p(y_n;\theta)\right)$$

Outcome

The negative log-likelihood criterion (maximizing the data likelihood) and the cross-entropy criterion (minimizing the distance between the model and data distributions) are <u>equivalent</u>.

Example: the least squares loss (LS)

Assume we have a problem where if we guess the correct solution, the residuals are then distributed with Gaussian noise:

$$p(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left[-\frac{(y-\mu)^2}{2\sigma^2}\right]$$

And now assume the job of our model is to correctly predict the mean:

$$p(y|\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left[-\frac{(y-f(x;\theta))^2}{2\sigma^2}\right]$$

Following our "recipe", let's minimize the negative log of the likelihood $p(y|\hat{y})$:

$$
\begin{aligned}
\theta^* &= \underset{\theta}{\operatorname{argmin}}\ -log(p(y|\theta)) \\
&= \underset{\theta}{\operatorname{argmin}}\ -\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\cdot\exp\left[-\frac{(y-f(x;\theta))^2}{2\sigma^2}\right]\right) \\
&= \underset{\theta}{\operatorname{argmin}}\ -\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)-\log\left(\exp\left[-\frac{(y-f(x;\theta))^2}{2\sigma^2}\right]\right) \\
&= \underset{\theta}{\operatorname{argmin}}\ \frac{(y-f(x;\theta))^2}{2\sigma^2} = \underset{\theta}{\operatorname{argmin}}\ \big(y-f(x;\theta)\big)^2
\end{aligned}
$$

Source code API

```
loss = torch.nn.MSELoss()
output = loss(model(x), y)
```

Example: binary cross-entropy loss (BCE)

A random variable x follows a Bernoulli distribution if it takes the value 1 with probability α and 0 with probability $1 - \alpha$:

$$x \sim \text{Bernoulli}(\alpha) \quad \Longrightarrow \quad \begin{cases} p(x=1) = \alpha \\ p(x=0) = 1-\alpha \end{cases} \quad \Longrightarrow \quad p(x=\beta) = \alpha^{\beta} \cdot (1-\alpha)^{1-\beta}$$

Where $\beta \in \{0, 1\}$. If we have a binary classification problem, we can use this distribution to model the predictions $\hat{y} \in [0, 1]$ vs. the ground truth labels $y \in \{0, 1\}$:

$$p(y|\hat{y}) = \hat{y}^{y} \cdot (1-\hat{y})^{1-y}$$

Following our “recipe”, let’s minimize the negative log of the likelihood $p(y|\hat{y})$:

$$
\begin{aligned}
\theta^* &= \underset{\theta}{\operatorname{argmin}} \ -log(p(y|\hat{y})) \\
&= \underset{\theta}{\operatorname{argmin}} \ -log(\hat{y}^y \cdot (1-\hat{y})^{1-y}) \\
&= \underset{\theta}{\operatorname{argmin}} \ -log(\hat{y}^y) - log((1-\hat{y})^{1-y}) \\
&= \underset{\theta}{\operatorname{argmin}} \ \underbrace{-y \cdot log(\hat{y}) - (1-y) \cdot log(1-\hat{y})}_{\text{BCE}(\hat{y},y)}
\end{aligned}
$$

Source code API

```
loss = torch.nn.BCELoss()
output = loss(model(x), y)
```

Example: cross-entropy loss (CE)

What if, rather than binary, we have a multi-class classification?

$$p(y_c|\hat{y}) = \frac{\exp[\hat{y}_c]}{\sum_{\tilde{c}} \exp[\hat{y}_{\tilde{c}}]}$$

We follow a similar trick as before:

$$p(y|\hat{y}) = \prod_c p(y_c|\hat{y})^{y_c}$$

where y is a one-hot vector, and y_c is the c-th element of y.

Following our "recipe", let's minimize the negative log of the likelihood $p(y|\hat{y})$:

$$\begin{aligned}\theta^* &= \operatorname*{argmin}_{\theta}\ -\log(p(y|\hat{y})) \\ &= \operatorname*{argmin}_{\theta}\ -\log\Big(\prod_c \exp[\hat{y}_c]^{y_c}\Big) + \log\Big(\prod_c \Big(\sum_{\tilde{c}} \exp[\hat{y}_{\tilde{c}}]\Big)^{y_c}\Big) \\ &= \operatorname*{argmin}_{\theta}\ -\sum_c \hat{y}_c \cdot y_c + \log \sum_{\tilde{c}} \exp[\hat{y}_{\tilde{c}}]\end{aligned}$$

Source code API

```
loss = torch.nn.CrossEntropyLoss()
output = loss(model(x), y)
```

Reading materials (and acknowledgments)

- Simon Prince, “Understanding Deep Learning“, Chapter 5
  `http://udlbook.com`
- Kostantinos Derpanis, “Deep Learning in Computer Vision: Multi-layer Perceptron”
  `https://www.eecs.yorku.ca/~kosta/Courses/EECS6322`