Deep Neural Networks Continued

Contents

References

- Chapters 7, 9, and 11 of https://www.deeplearningbook.org/
- These notes.

These notes are not exhaustive. They merely provide a summary of the key concepts. Please consult the book chapters for the complete details.

Loss functions for classification

Take some features $\mathbf{x}_{1:n}$ and some discrete targets $y_{1:n}$. Because the targets are discrete, we have a classification problem. What loss function should we use? Let's examine two cases: binary and multiclass classification.

Binary classification

In binary classification, we use a DNN $f(\mathbf{x};\theta)$ with parameters θ to model the probability that y take the value 1 by:

$$p(y=1|\mathbf{x},\theta) = \text{sigm}(f(\mathbf{x};\theta)) = \frac{\exp\{f(\mathbf{x};\theta)\}}{1+\exp\{f(\mathbf{x};\theta)\}}.$$

Remember that the sigmoid function takes the scalar $f(\mathbf{x};\theta)$ and maps it on $[0,1]$ so that we get a probability. From the obvious rule of probability, we get that:

$$p(y=0|\mathbf{x},\theta) = 1 - p(y=1|\mathbf{x},\theta) = 1 - \text{sigm}(f(\mathbf{x};\theta)).$$

So, for an arbitrary y (either 0 or 1), we can write:

$$p(y|\mathbf{x},\theta) = [\text{sigm}(f(\mathbf{x};\theta))]^y[1-\text{sigm}(f(\mathbf{x};\theta))]^{1-y}.$$ likelihood of a single observation

This is a nice trick because it activates the right term based on what y is.

Now that we have specified the likelihood of a single observation, the likelihood of the entire dataset is:

$$p(y_{1:n}|\mathbf{x}_{1:n},\theta) = \prod_{i=1}^n p(y_i|\mathbf{x},\theta).$$

We are almost done. The idea is to train the network by maximizing the log likelihood, which is the same as minimizing the following loss function:

\begin{split} L(\theta) &= -\log p(y_{1:n}|\mathbf{x}*{1:n},\theta)\ &= -\sum{i=1}^n \left{y_i \log \operatorname{sigm}(f(\mathbf{x}_i;\theta))

- (1-y_i)\log \left[1-\operatorname{sigm}(f(\mathbf{x}_i;\theta))\right] \right}. \end{split}

This loss function is known as the *cross entropy* loss.

Multiclass classification

Now assume that y can take K different values ranging from 0 to $K-1$. We need to model the probability that $y=k$ given $\mathbf{x}$. To do this, we introduce a DNN $\mathbf{f}(\mathbf{x};\theta)$ with parameters θ and K outputs:

$$\mathbf{f}(\mathbf{x};\theta) = (f_0(\mathbf{x};\theta), \ldots, f_{K-1}(\mathbf{x};\theta)).$$

However, $\mathbf{f}(\mathbf{x})$ is just a bunch of K scalars. We need to turn it into a K-dimensional probability vector. To achieve this, we define:

$$p(y=k|\mathbf{x},\theta) = \text{softmax}_k(\mathbf{f}(\mathbf{x};\theta)) := \frac{\exp\{f_k(\mathbf{x};\theta)\}}{\sum_{k'=0}^{K-1}\exp\{f_{k'}(\mathbf{x};\theta)\}}.$$ https://en.wikipedia.org/wiki/Softmax_function

So, the role of the softmax is dule. First, to turn the scalars to positive numbers and, second, to normalize them.
this generates a probability distribution of K possible outcomes

For an arbitrary y, we can just write:

$$p(y|\mathbf{x},\theta)) = \text{softmax}_y(\mathbf{f}(\mathbf{x};\theta)).$$

So, the likelihood of the dataset is:

$$p(y_{1:n}|\mathbf{x}_{1:n},\theta) = \prod_{i=1}^{n} p(y_i|\mathbf{x},\theta).$$

Therefore, the loss function we should be minimizing is:

\begin{split} L(\theta) &= -\log p(y_{1:n}|\mathbf{x}*{1:n},\theta)\ &= -\sum{i=1}^n \log \left[\operatorname{softmax}_{y_i}(\mathbf{f}(\mathbf{x}_i;\theta))\right]. \end{split}

This is also a called *cross entropy* loss, but for multiclass classification. Sometimes, it is called the *softmax cross entropy* loss. Now all these names are not really important. What is important is to understand how these losses are derived from maximum likelihood.

Regularization

DNNs are extremely flexible. However, this makes them prone to overfitting. As we have discussed in the past one way to avoid overfitting is to add regularization to the loss function. This works as follows. Instead of minimizing $L(\theta)$ you minimize:

$$J(\theta) = L(\theta) + \lambda R(\theta),$$

where λ is a positive parameter, and $R(\theta)$ is a term that penalizes very big weights. One possibility of $R(\theta)$ is the L2 regularization:

$$R(\theta) = \sum_j \theta_j^2.$$

But, there are many more. Also, you could add multiple regularizers - not just one. Now, the bigger λ is the more you regularize and the smaller it is the less you regularize. So, λ is a parameter you would like to tune - but more on this later.

Bayesian interpretation of regularization

As we showed during our regression lectures, the regularization can be interpreted as a maximum posterior approach to inference. To see this, assume that we have a data likelihood $p(y_{1:n}|\mathbf{x}_{1:n},\theta)$ for either a regression or a classification problem. Now, assume that you have a prior over all parameters, say $p(\theta)$. Then, write down the posterior of the parameters:

$$p(\theta|\mathbf{x}_{1:n},y_{1:n}) \propto p(y_{1:n}|\mathbf{x}_{1:n},\theta)p(\theta).$$

Instead of picking the parameters by maximizing the log likelihood, let's pick the parameters by maximizing the log posterior. This is equivalent to minimizing the following loss function: skipping a lot of steps/math:

$$J(\theta) = -\log p(y_{1:n}|\mathbf{x}_{1:n},\theta) - \log p(\theta).$$ function to be minimized

Of course, the first term is just $L(\theta)$. The second term is $\lambda R(\theta)$. For a more concrete example, assume that the prior over θ is a zero-mean Gaussian with precision α:

$$p(\theta) = N(\theta|0,\alpha^{-1}I).$$

Then, we get:

$$-\log p(\theta) = \frac{\alpha}{2\pi}\sum_j \theta_j^2 + \text{const.}$$

And this is how you get the L2 regularization term in a principled way.

Convolution neural networks

Convolutional neural networks try to mimic the operations carried out by the animal visual cortex. By construction, they satisfy certain properties (e.g., invariance to small translations and rotations) that make them particularly succesfful in image recognition tasks. It is beyond the scope of to explain the mathematical details of convolutional neural networks. The details presented in the lecture videos should be enough for this course and for building standard image classifiers. If you want to know more, you can read chapter 9 of the Deep Learning Book or take a course on deep neural networks, e.g., BME 646 Deep Learning- Theory and Practice of Deep Neural Networks.

By Ilias Bilionis (ibilion[at]purdue.edu)