3.2. Probabilistic deep learning

As per the usual supervised learning setup, suppose we have a labelled dataset $(x_1, y_1), \ldots, (x_n, y_n)$, where x_i is the input and y_i is the label for record i.

The probabilistic modelling approach is to say "the labels are noisy, and we should *model their distribution*". To be precise, we'll fit a parametric probability distribution $\Pr_Y(y_i \,;\, f(x_i))$ whose parameters are computed by a neural network f. Fitting a probability model means maximizing the log likelihood of the dataset, and so that's the objective we can use to train the neural network.

We'll see that, for simple probability distributions at least, maximum likelihood estimation does the same thing as the conventional "minimize prediction loss" procedure, as it's used by non-probabilistis who think the job of a neural network is just to make predictions.

It's good machine learning practice to split the dataset into a training set and a holdout set, and use the former for training and the latter for evaluation. Exactly the same is true for probabilistic deep learning. The only difference is that evaluation means "evaluate how good a fit my model is", and so the evaluation metric is the log likelihood according to the fitted model of the holdout dataset.

LOG LIKELIHOOD AND PREDICTION ACCURACY

For simple probability models, fitting with maximum likelihood estimation boils down in the end to minimizing prediction loss, for a suitable prediction loss function. Another way of saying this is: we can use all the standard tools for training neural networks, but now we have a principled way of picking sensible loss functions.

To see how this works, we'll look at two examples. The first example is a regression with Gaussian errors, a generalization of our model for iris petal length from section 2.4,

$$\mathsf{Petal.Length}_i \sim \alpha + \beta\, \mathsf{Sepal.Length}_i + \gamma\, (\mathsf{Sepal.Length}_i)^2 + N(0, \sigma^2).$$

Example 3.2.1 (Regression with Gaussian errors).
Consider a regression model with Gaussian errors

$$Y_i \sim f(x_i \,;\, \theta) + N(0, \sigma^2)$$

where Y_i is a random variable modelling the label, x_i includes all relevant features, and f is a neural network with parameters θ. Suppose we've observed values $y_1, \ldots, y_n$ for the labels. Describe how to find maximum likelihood estimates $\hat{\theta}$ and $\hat{\sigma}$.

The log likelihood of the entire dataset is

$$\log \Pr(y_1, \ldots, y_n \,;\, \theta, \sigma) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} \big(y_i - f(x_i; \theta)\big)^2.$$

We want to maximize this over θ and σ. We can do the maximization over θ first: for this we need to solve

$$\textit{minimize} \quad \sum_{i=1}^{n} \big(y_i - f(x_i; \theta)\big)^2 \quad \textit{over } \theta$$

This is exactly Gauss's calculation from section 2.4

which is the same thing as training the neural network to mimimize the average loss for the prediction loss function

$$L(\textit{label}, \textit{pred}) = (\textit{label} - \textit{pred})^2.$$

It's then simple calculus to find the maximum likelihood estimate for the noise parameter $\hat{\sigma}$—though this last step doesn't have a 'prediction loss' interpretation.

∎

The next example is classification, along the lines of the MNIST image classification problem in example 3.1.2 above. In classification problems, the labels come from a discrete set. For our probability model we'll use a Categorical random variable. Let $\vec{p}$ be a probability vector, with an entry p_k for every possible outcome k, and let $Y \sim \text{Cat}(\vec{p})$. This simply means $\mathbb{P}(Y = k) = p_k$. It is the simplest most general model possible for a random variable with discrete outcomes.

other uses of Categorical: exercise 1.6.3 page 18.

Example 3.2.2 (Classification).
In classification, the labels come from a finite set, call it $\{1, \dots, K\}$. Consider a probability model based on a Categorical random variable,

$$Y_i \sim \text{Cat}\big(\vec{p}(x_i; \theta)\big)$$

Here $\vec{p}(x\,;\,\theta)$ is some arbitrary function that takes the predictor variables x, as well as parameters θ, and returns a probability vector

$$\vec{p}(x_i; \theta) = \big[p_1(x_i; \theta), \dots, p_K(x_i; \theta)\big].$$

Suppose we've observed values $y_1, \dots, y_n$ for the labels. Describe how to find the maximum likelihood estimator for θ.

When trying to maximize the likelihood, it's awkward to maximize over a constrained domain—here we'd have to ensure we only pick θ that results in a valid probability vector i.e. each probability ≥ 0 and all summing to one. It's easier to transform to an unconstrained space, using the softmax transform from section 1.2 page 8. Thus, let's seek a different function $\vec{f}(x\,;\,\theta) \in \mathbb{R}^K$, and then let

$$p_k(x\,;\,\theta) = \frac{e^{f_k(x;\theta)}}{e^{f_1(x;\theta)} + \cdots + e^{f_K(x;\theta)}} \quad \textit{for } k \in \{1, \dots, K\}.$$

Then we're free to choose any θ at all, and we're guaranteed to end up with a probability vector $\vec{p}$.

To train this model we follow the standard maximum likelihood procedure: write out the log likelihood, and maximize it. The likelihood of an individual observation y is

$$\Pr_Y(y\,;\,x, \theta) = p_y(x\,;\,\theta)$$

So the log likelihood of the whole dataset is

$$\log \Pr(y_1, \dots, y_n\,;\,\theta) = \sum_{i=1}^{n} \log p_{y_i}(x_i\,;\,\theta) = \sum_{i=1}^{n} \sum_{k=1}^{K} 1_{y_i=k} \log p_k(x_i\,;\,\theta).$$

We want to pick θ to maximize this. Equivalently, sticking a minus sign in front and scaling by n, we want to pick θ to minimize

$$\text{loss} = \frac{1}{n} \sum_{i=1}^{n} \text{crossentropy}\Big(\text{onehot}(y_i), \text{softmax}\big(\vec{f}(x_i\,;\,\theta)\big)\Big)$$

where

$$\begin{aligned}
\text{onehot}(y) &= \big[1_{y=1}, \dots, 1_{y=K}\big] \\
\text{softmax}(\vec{f}) &= \Big[\frac{e^{f_1}}{e^{f_1} + \cdots + e^{f_K}}, \dots, \frac{e^{f_K}}{e^{f_1} + \cdots + e^{f_K}}\Big] \\
\text{crossentropy}(\vec{q}, \vec{p}) &= -\big(q_1 \log p_1 + \cdots + q_K \log p_K\big).
\end{aligned}$$

∎

This is known as "minimizing softmax cross-entropy loss with onehot coding". It's a sesquipadelian way of saying "I'm modelling this data with a Categorical random variable, the simplest possible model for discrete outcomes; I'm using the simplest possible transform to make the optimization easy; and I'm fitting using maximum likelihood estimation, the bog standard way of estimating parameters."

Interestingly, the decomposition into crossentropy and onehot and softmax makes it clear that the real prediction task here isn't "predict the label y_i", it's "predict the one-hot coded version of y_i".

WHY PROBABILISTIC IS BEST

The 'prediction accuracy' mindset doesn't involve any explicit probability modelling. Someone who doesn't believe in probability theory could perfectly well formulate a task as a problem of minimizing prediction loss; they might even claim that deep learning is entirely about prediction and loss functions, and doesn't need any modelling at all.

Moreover, the probability interpretation takes some mental gymnastics. The probabilist can look at a picture of a cat and say "The label for this picture is a random variable, and it takes value 'cat' with probability 92%", when any normal person would say "What do you mean, random? it's a cat, for goodness sake!" The interpretation of probability is mind bending, and even experts get it wrong, as the final example of this section will show.

In the author's opinion, the probabilistic interpretation of deep learning is much better than the 'prediction accuracy' mindset …

- Without a probability model, different loss functions are just formulae that we have to memorize. With a probability model, we still have to design a model, but the loss functions don't look like a laundry list of mystery.

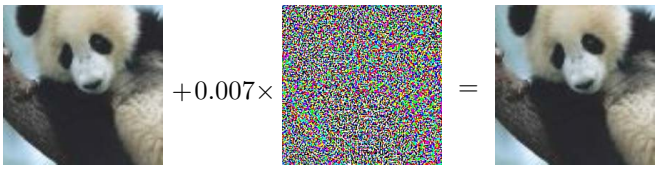
- If we face a new type of dataset, it's fairly intuitive to design a probability model for it, perhaps in the form of simulation code. We can then derive the corresponding loss function, and since it comes from our intuitive probability model, it should be well-behaved. On the other hand, if we only think in terms of prediction loss, we might design a loss function that makes the learning go haywire. Arguably, any sane loss function has a corresponding probability model.

- There are useful models that don't have a natural interpretation as minimizing prediction loss, but do have a natural probabilistic interpretation. For example, suppose we think that the noise term σ in the iris petal-length model should depend on features of the iris, $\sigma = \sigma(x_i\,;\,\phi)$ for some parameter ϕ to be learnt. It's trivial to put this into the probability model and fit it, unclear how to frame it as 'minimize prediction loss'.

- If we think in terms of probability models, it's easy to see how to train and evaluate unsupervised neural network models, which we'll turn to in the next section. If we think in terms of prediction and loss, it's hard to even begin to formulate what unsupervised learning is meant to achieve.

- Without the probabilistic perspective we can fall into traps of interpretation. The next example, a summary of some ingenious research into adversarial attacks against neural network classifiers, illustrates.

Example 3.2.3 (The adversarial panda[23]).



panda, 57.7% confidence | nematode, 8.2% confidence | gibbon, 99.3% confidence

A neural network was trained to classify images. When shown the leftmost image, it reports "`panda, 57.7% confidence`". The center image is carefully chosen noise. By blending the original panda with noise at 0.7% opacity, we obtain the rightmost image, which the neural network interprets as "`gibbon, 99.3% confidence`".

When we train a neural network classifier, we're really just fitting a probability model. Given an input image x, the neural network outputs a probability vector $\vec{p}(x\,;\,\theta)$, where θ contains all the fitted parameters of the network.

A thought experiment right at the beginning of our study of maximum likelihood, on page 1.1, told us that estimated probabilities do not measure confidence. The thought experiment was this: If

[23] I. J. Goodfellow, J. Shlens, and C. Szegedy. "Explaining and Harnessing Adversarial Examples". In: *ArXiv e-prints* (Dec. 2014). arXiv: 1412.6572 [stat.ML]

we toss 4 coins and get 3 heads, the estimated probability of heads is $3/4$. If we toss 4 million coins and get 3 million heads, it's still $3/4$. We should surely be more confident in the latter case. Thus, estimated probabilities do not measure confidence.

The neural network did not report a confidence, it reported a probability, and we don't know how confident it was in its guess. The tools for measuring confidence, which we'll study in part III, are all based on probability models, and without a probabilistic view of neural networks we will not be able to extract confidence measures. (Confidence measures for neural networks are, however, still a topic of active research.)

∎