Point-Estimate Neural Networks

Juan Maroñas
jmaronasm@gmail.com

PRHLT Research Center,
Universitat Politècnica de València

April 2019

1 Introduction

In this document I expose the mathematical derivation of point estimate Neural Networks following the same assumptions I will be taking for the rest of the models. When talking about probabilistic modelling, people always talk about classification, regression, supervised/unsupervised learning, parametric/unparametric models and so no. I do not like that classificaton as, in my opinion, it does not include all the possibilities. I always talk about point-estimate and Bayesian models.

If you are familiar with Bayesian model just skip this and the next paragraphs. As these series of documents are far from being technical descriptions of the methods I am implementing, I will only give the necessary details to justify and understand the provided implementations. Basically If you have studied this models, then understanding this short pdfs should be straightforward and you could use them to loop over the different models.

To start with, I will give a short illustration on Bayesian and point-estimate modelling. Basically in Bayesian inference your predictions on unobserved data are based on making a weighted average of the likelihood models under the posterior distribution of the parameters. This means that predictions considers each possible set of parameters that can parameterize your likelihood, and raise the importance of each depending on the data. On the other hand point-estimate approaches optimize a training criteria to recover a unique set of parameters denote by $\hat{\theta}$. This is why I like to talk about point-estimate as we get a unique point in the parameter space to make decisions.

In this document I will cover the two typical approaches for point-estimate techniques, so called maximum likelihood and maximum posterior. They are pretty much the same that is why I include them in the same document. I will now expose the models with the assumptions taken.

1.1 Common assumptions

A common assumption (when dealing at least with non-sequence inputs) is to assume that the data has been drawn i.i.d from a true never known distribution (unless you draw data using numpy.randn() :D). Following this and given a set of observed data points $\mathcal{O} = \{(x_i, t_i)\}_{i=1}^N$ we can decompose the joint distribution in the next way:

$$p(x_1, ..., x_N, t_1, ...t_N, \theta) = \prod_{i=1}^{N} p(t_i|x_i, \theta) \tag{1}$$

where apart from assuming the aforementioned independence we have also discard some dependence between our random variables and also assume some conditional independence. Just decompose $p(t_1, t_2, x_1, x_2)$ using product rule and you will easily check that. The above equation is commonly known as the likelihood function.

2 Maximum Likelihood

Well, it is straightforward to derive a training criteria in this setting, just maximize the likelihood function w.r.t θ and recover the $\hat{\theta}$ that optimizes the parameters. Formally:

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \prod_{i=1}^{N} p(t_i|x_i, \theta) \tag{2}$$

And this criteria just relies in computing derivatives of the likelihood and apply step-gradient optimization (unless it can be compute a closed form, for instance if you use Gaussian Likelihood with linear relations).

However, computing this funcion when dealing with large datasets have several problems, basically: it is costly and very numerically unstable. However, here comes the only thing I like from point-estimate models (really it is the only fancy thing from them): it is straighforward to scale them to large datasets. The only thing we require is to be able to compute an unbiased estimate of the gradient of this likelihood. With that, we can then make random estimates of the gradients, and apply the aforementioned step-gradient algorithms. To achieve this property, just apply the log to the criteria and minimize:

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \sum_{i=1}^{N} \log p(t_i|x_i, \theta) \tag{3}$$

In my set of examples I will assume a categorical likelihood distribution yielding our final training criteria:

$$\hat{\theta} = \underset{\theta}{\text{argmin}}\, \text{CE}(t, f_\theta(x)) \tag{4}$$

where CE is the common cross entropy criteria and f_θ is the function that predicts the parameters of the categorical distribution taking as input x.

3 Maximum Posterior

It is straightforward to extend the above training criteria to use a maximum posterior estimation by placing a prior over the parameters $p(w)$. In this case we recover the set of optimal parameters that maximize the posterior distribution of the parameters given the observations. By applying Bayes theorem we have:

$$\begin{aligned}\hat{\theta} &= \underset{\theta}{\operatorname{argmax}}\, p(\theta|\mathcal{O}) \\ &= \underset{\theta}{\operatorname{argmax}}\, \frac{\prod_{i=1}^{N} p(t_i|x_i,\theta)\cdot p(\theta)}{Z}\end{aligned} \tag{5}$$

where Z is just a normalization factor which does not depend on θ and can be take out from the optimization process. This normalization factor is just the marginatlization of the numerator over w. Just note that each parameter is scaled by Z so the optimal remains the same. For the same reasons as above (stability and large scale problems) we take the logarithm, yielding to our final training criteria.

$$\underset{\theta}{\operatorname{argmax}}\, \log \prod_{i=1}^{N} p(t_i|x_i,\theta) + \log p(\theta) \tag{6}$$

By setting a factorized Standard Normal prior on the parameters (as I will be doing for the rest of the models) we end up with:

$$\log p(\theta) = \log \mathcal{N}(w|0,I) = \sum_{i=1}^{|W|} -0.5\log 2\pi - 0.5 w_i^2 w_i \tag{7}$$

By removing the constant terms from the optimization (for the same reason as we remove Z) we end up with our final criteria.

$$\underset{\theta}{\operatorname{argmin}}\, \mathrm{CE}(t, f_\theta(x)) + \sum_{i=1}^{|W|} 0.5 w_i^2 \tag{8}$$

Note that this is just a formalization of the L_2 regularization. With the two criterias (maximum likelihood and maximum posterior) I will be using naive stochastic gradient descent with momentum.