Application of the Gaussian process for designing the CNN architecture

Hardik Prabhu

April 2021

1 A shot in the dark

When it comes to an image recognition task, a simple neural network just performs poorly due the parameter blow up. There are just too many parameters to deal with and this may lead to over fitting and bad results. The Vanilla neural network reads entire image as an input. Let us consider the input being of size 200x200 pixels then the flattened input vector will in be 200x200 dimensions and then if we have 10 neurons in the first hidden layer then the network will have more than 200x200x10 number of weight parameters. If the training sample isn't large enough it will lead to over fitting. CNN architecture mimics the visual cortex. The processing is done in layers. Initially we have convolution and pooling layers. A convolution layer filters aggregate information across a small receptive field to capture features. Higher layers combine features at lower levels. Pooling layers use subsampling for dimension reduction. Within a filter layer, parameter sharing reduces number of parameters to be learned. And finally we have our vanilla neural network at the end which accepts a significantly lower dimension input.

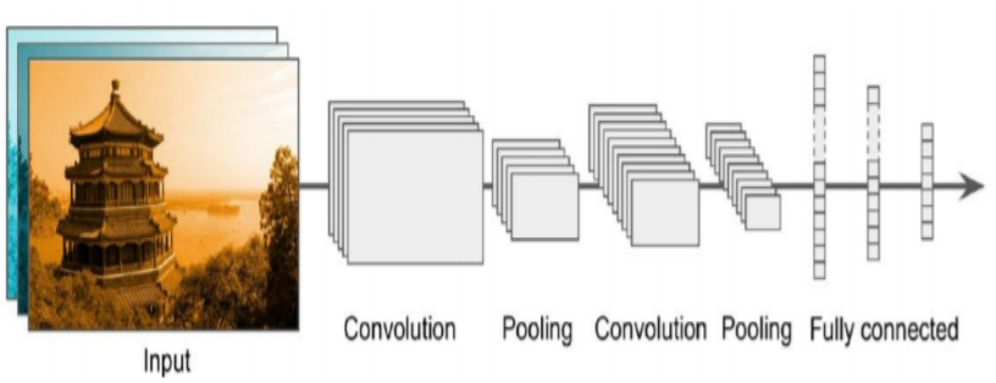

Figure 1: The CNN architecture.

Unfortunately designing an architecture of the neural network still feels like taking a shot in the dark. The tuning of hyper parameters of the model is totally left to us. We take some alternating layers of Convolution and max-pooling. The number of such layers could be a hyperparameter. Each convolution layer consists of some filters, the number of filters in a layer could be a hyperparameter. Each filter has square kernel of size nxn. "n" could be a hyperparameter. Each max-pooling layer has pool size of k. (compresses the image by 1/k) finally the lower dimension input is flattened and then fed to a neural network. Total number of hidden layers could be a hyper parameter. Neurons per layer is also another hyper parameter which we may want to tune. Simple grid search over the all such combination of parameters may turn out to be computationally costly. We don't have any analytical cost function for the performance of the model over the hyper parameter space. It is a complete black box function. In such a situation we rely on the Gaussian process. It will provide us with an algorithm to step by step pick a better combination of hyper parameters instead of just going over the entire grid of hyper parameters values and creating the network and then selecting the hyper parameter with the best performance.

1.1 Gaussian process

GP regression is a Bayesian statistical approach for modeling functions. The functions which are of interest have no mathematical formulation and the closer two input points are the closer the values the function returns (a sense of continuity over input space).
We first collect f's values at a finite collection of points $x_1, \ldots, x_k \in R^d$. We get a vector of these values $[f(x_1), ..., f(x_k)]^\top$. we suppose that it was drawn at random by nature from some prior probability distribution. This prior distribution is taken as a multivariate normal, with a particular mean vector and covariance matrix. We initalize a mean vector $\mu_0 = [\mu_0(x_1), ..., \mu_0(x_k)]^\top$ and covariance matrix Σ_0. Each element of the mean vector is calculated by a function μ_0. The covariance matrix is initialized by using a kernel function which k. Each entry i,j is given by $k(x_i, x_j)$ Since we believe the function f is such that if x is close to y then $f(x)$ is close to $f(y)$. The kernel is chosen such that points x_i, x_j that are closer in the input space have a large positive correlation. One example of such a kernel is the RBF kernel.

$$K(x, x') = e^{-\alpha||x-x'||^2}$$

Say if we were interested in predicting housing prices. It makes sense if my neighbour who has an identical house as me will have the same price as mine. On the other hand, my houses price is practically independent of what price a mansion is being sold at. The vector of a house may include features such as no. of rooms, area, locality etc. The price is the function we want to predict. It can be modelled using a Gaussian process. For now, lets stick with the task at hand. How is setting such prior is going to help us model the function over entire range of values of the input vector?

The prior distribution of $[f(x_1), ..., f(x_k)]^\top$ is,

$$f(x_{1:k}) \sim Normal(\mu_0, \Sigma_0)$$

Suppose we have n sample points and we observe a new data point x, the prior distribution for the collection of n+1 points i.e $f(x_{1:n+1})$ is the same as mentioned above, just replace k with n+1 and x_k is x.

Now we get a posterior distribution for $f(x)$ which is also a Gaussian distribution.

$$f(x)|f(x_{1:n}) \sim Normal(\mu_n(x), \sigma_n^2(x))$$

Notation: if $x \in R^n$, $x_{l:m}$ represents the partition of x in R^{m-l} from lth to the mth element. $x_{l:m} = [x_l, x_{l+1}..., x_m]^\top$. $f(x_{l:m}) = [f(x_l), f(x_{l+1})..., f(x_m)]^\top$

where,

$$\mu_n(x) = \Sigma_0(x, x_{1:n})\Sigma_0(x_{1:n}, x_{1:n})^{-1}(f(x_{1:n}) - \mu_0(x_{1:n})) + \mu_0(x)$$

$\Sigma_0(x, x_{1:n})$ is the vector of covariances of of all sample x_i with x. It is $[K(x_1, x), ...k(x_n, x)]$.
The matrix $\Sigma_0(x_{1:n}, x_{1:n})$ is $n \times n$ sub matrix of the matrix Σ_0 with the last row and column removed. Removing the rows and columns having co variances w.r.t x and the sample points.
The posterior mean $\mu_n(x)$ is a weighted average between the prior $\mu_0(x)$ and an estimate based on the data $f(x_{1:n})$, with a weight that depends on the kernel.

and

$$\sigma_n^2(x) = k(x, x) - \Sigma_0(x, x_{1:n})\Sigma_0(x_{1:n}, x_{1:n})^{-1}\Sigma_0(x, x_{1:n})^\top$$

.

The proof of the conditional distribution being Gaussian, and also the derivation of posterior mean and co variance matrix can be found in any multivariate statistics book.

For each x in the input space, we have a posterior distribution of $f(x)$. The points which are not near the sampled point has a large uncertainty in the value(large variance). We can sample a point from that region to improve our predicted function (exploration). We can get greedy with the task at hand and sample the next point in the region of optimal predicted value(exploitation). Since, the sampling isn't cheap, we rely on an acquisition function which considers both the aspects to get the best candidate x for sampling. We wont be discussing more about the acquisition

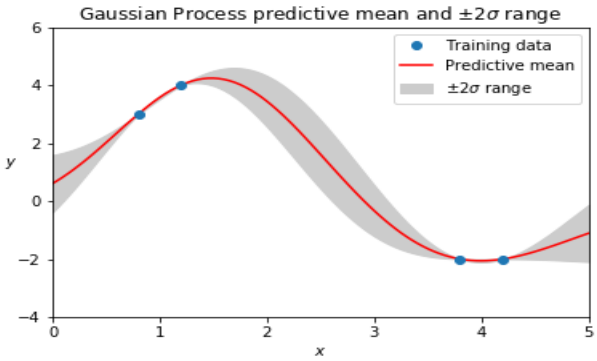

Figure 2: For $x \in R$, $f(x)$ predicted by the Gaussian process.

function.
We haven't yet discussed how we arrive at an initial μ_0. If we don't have any prior knowledge about the function behaviour, then we can set the mean values to a constant $\mu_0(x_i) = \mu$. If we assume the function may have a certain parametric structure, we may also take the mean function to be $\mu_0(x) = \mu + \Sigma\beta_i\Psi_i(x)$. where each Ψ_i is a parametric function.

1.2 Designing the neural network

- We take some alternating layers of Convolution and max-pooling. The number of such layers is a hyperparameter.
- Each convolution layer consists of some filters, the number of filters in a layer is a hyperparameter.
- Each filter has square kernel of size nxn. "n" is a hyperparameter. Each max-pooling layer has pool size of 2. (compresses the image by half)
- The image is flattened and then fed to a neural network.
- Neurons per layer is also another hyperparameter. The final layer has 10 neurons(1 per class) with softmax activation function. Every other neuron not in the last layer has Re-Lu activation function.

We use Tensorflow library to design the neural network. We use the bayesian optimization code given by Thomas Huijskens on his github page.

2 Results

The accuracy of a particular CNN over test data is our cost function and the input is the possible values of the hyper parameters. We set the bounds for different hyper parameters. Grid searching through all the possible combination is computationally very much costly. To show that indeed, its better to do a grid search over entire combination on hyper-parameter space, we use the gaussian process to evaluate after looking at only the 20 combinations of hyper parameters to estimate the cost function over entire hyper parameter space using the gaussian process. We select the hyperparameters with lowest cost. We set up two different CNNs, one with the hyperparameters obtained by the gp and chosen arbitrarily for the other. We compare the accuracy after 15 epochs of both the networks. The arbitrarily designed network got an accuracy of **0.28**. After hyper-parameter tuning using Bayesian optimization for 20 iterations, the accuracy over the test dataset is **0.92** which is significantly better than random hyper-parameter tuning.