Gaussian process classifiers and CNN uncertainty

Sebastian Borgeaud dit Avocat
spb61@cam.ac.uk

LE49 - Probabilistic Machine Learning Project

Abstract

The abstract paragraph should be indented ½ inch (3 picas) on both the left- and right-hand margins. Use 10 point type, with a vertical spacing (leading) of 11 points. The word **Abstract** must be centered, bold, and in point size 12. Two line spaces precede the abstract. The abstract must be limited to one paragraph.

1 Introduction

Model uncertainty is often of crucial importance.

Softmax output does not give model confidence, maybe show the example used by Yarin Gal.

With model uncertainty it is possible to treat uncertain inputs and special cases explicitly. Example if model uncertainty is high we might decide to pass the input to a human for classification.

Another example is in Reinforcement learning: with uncertainty a RL agent can decided when to explore and when to exploit in an environment. With principled uncertainty for the agent's Q-value function, it can learn much faster using techniques such as Thompson sampling.

Yarin Gal shows how dropout in neural networks can be interpreted as a Bayesian approximantion of a Gaussian Process.

1.1 Convolutional neural networks [1] (ADD MORE CITATIONS)

Convolutional neural networks (CNNs) were originally inspired by biological processes. They have become standard in many deep learning applications, especially in image processing or vision tasks. A convolutional neural network is a type of feedforward neural network, typically consisting of convolution layers, pooling layers and fully connected layers:

- **Convolution layers** are composed of several convolution kernels each computing a different feature map. The output feature maps are obtained by convolving the input with the convolution kernel and then applying an element-wise nonlinearity. Mathematically, the feature value $z^l_{i,j,k}$ at location (i,j) of the k^{th} feature map in the l^{th} layer is computed as:

  $$z^l_{i,j,k} = {\mathbf{w}^l_k}^T \mathbf{x}^l_{i,j} + b^l_k$$

  where $\mathbf{w}^l_k$ and b^l_k are the weight and bias vectors for the k^{th} convolution kernel in the l^{th} layer and $\mathbf{x}^l_{i,j}$ is the input patch centered around (i,j) in the l^{th} layer. The ouput value is computed by apply a nonlinearity $a(\cdot)$ point-wise:

  $$x^{(l+1)}_{i,j,k} = a(z^l_{i,j,k})$$

- **Pooling layers** aim to achieve shift-invariance and reduce the number of parameters in the network by reducing the resolution of the feature maps. The pooling layer operates on each feature map independently. Mathematically, the output of a pooling layer with pooling operation pool$(\cdot)$ is given by

$$y^l_{i,j,k} = \text{pool}(x^l_{m,n,k}), \forall (m,n) \in \mathcal{R}_{i,j}$$

where $\mathcal{R}_{i,j}$ is a local neighbourhood around (i,j). Typically, the pooling operation computes the average or the maximum.

- **Fully connected layers** connect every neuron in the previous layer to single neuron in the current layer. Fully connected layers are the layers used in standard neural networks. Mathematically, the output of a fully connected layer is given by:

$$x_i^{(l+1)} = a\Big(\big(\sum_j w^l_{i,j} x^l_j\big) + b^l_i\Big)$$

where $a(\cdot)$ is a nonlinearity, $w^l_{i,j}$ is the weight connecting neuron j in the l^{th} layer to neuron i in layer $(l+1)^{\text{th}}$, and b^l_i is the bias weight for neuron i.

Optimising a convolutional neural network is done in the same way as optimising standard neural networks. A differentiable loss function is computed for the training examples (often done in batches) and the gradients w.r.t. the weights of the network are computed. Using these gradients, the weights are updated in a gradient descent step. Typically, more complex update rule that take into account change momentum (e.g. Adam [2]) are used as they converge faster to a local minimum.

1.2 Gaussian processes

Formally, a Gaussian Process is defined as a collection of random variables, any finite number of which have (consistent) joint Gaussian distributions. A Gaussian process therefore defines a distribution over functions and is fully specified by a mean function $m(x)$ and a covariance function $k(x, x')$. Write $f \sim \mathcal{GP}(m, k)$ meaning f is distributed as a GP with mean m and covariance k.

Using the GP we can draw samples from the function for any finite number n of locations. Given locations $\mathbf{x} = [x_1, \ldots, x_n]$, first compute $\mu_i = m(x_i)$, $\Sigma_{i,j} = k(x_i, x_j)$. We can then sample a vector from this distribution: $\mathbf{f} \sim \mathcal{N}(\mu, \mathbf{\Sigma})$.

1.2.1 Regression

We can now use this GP as a prior for Bayesian inference. Let $\mathbf{f}$ be the known function values for the training examples an let $\mathbf{f}_*$ be the set of function values corresponding to the set of test input X_*. The joint distribution is given by

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} = \mathcal{N}\Big(\begin{bmatrix} \mu \\ \mu_* \end{bmatrix}, \begin{bmatrix} \mathbf{\Sigma} & \mathbf{\Sigma}_* \\ \mathbf{\Sigma}_*^T & \mathbf{\Sigma}_{**} \end{bmatrix}\Big)$$

where μ_* are the test means, $\mathbf{\Sigma}_*$ are the training-test covariances, and $\mathbf{\Sigma}_{**}$ are the test-test covariances. Since we know the training values $\mathbf{f}$, we are interested in the conditional distribution of $\mathbf{f}_*$ given $\mathbf{f}$:

$$\mathbf{f}\big|\mathbf{f}_* \sim \mathcal{N}\big(\mu_* + \mathbf{\Sigma}_*^T \mathbf{\Sigma}^{-1}(\mathbf{f} - \mu), \mathbf{\Sigma}_{**} - \mathbf{\Sigma}_*^T \mathbf{\Sigma}^{-1} \mathbf{\Sigma}_*\big)$$

This corresponds to a posterior Gaussian process, given by

$$\begin{aligned} f\big|\mathcal{D} &\sim \mathcal{GP}(m_\mathcal{D}, k_\mathcal{D}), \\ m_\mathcal{D}(x) &= m(x) + \Sigma(X, x)^T \Sigma^{-1}(\mathbf{f} - \mathbf{m}) \\ k_\mathcal{D}(x, x') &= k(x, x') - \Sigma(X, x)^T \Sigma^{-1} \Sigma(X, x') \end{aligned}$$

where $\Sigma(X, x)$ is a vector of covariances between every training case in X and x. Furthermore, it is easy to incorporate noise in the observations. Assuming i.i.d. additive Gaussian noise, every $f(x)$ now has extra covariance with itself with a magnitude equal to the noise variance σ_n^2:

$$f\big|\mathcal{D} \sim \mathcal{GP}(m_\mathcal{D}, k_\mathcal{D} + \delta_{ii}\sigma_n^2)$$

where $\delta_{ii'} = 1$ iff $i = i'$ is the Kronecker's delta.

The mean function $m(x)$ and the covariance function $k(x, x')$ are typically parametrised in terms of hyper-parameters $\boldsymbol{\theta}$. During training we find the values of the hyper-parameters which optimise the marginal likelihood:

$$L = \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) = -\frac{1}{2}|\Sigma| - \frac{1}{2}(\boldsymbol{y} - \boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{y} - \boldsymbol{\mu}) - \frac{n}{2}\log(2\pi)$$

This optimisation can be done using standard gradient methods.

1.2.2 Classification

Explain how classification is done with GPs, explain the difficulties

2 Method

2.1 Network architecture

2.2 Gaussian process

2.3 Metrics

3 Results

4 Discussion

More quantitave evalution of the results. Show examples of images that are missclassified. Discuss definition of the metrics.

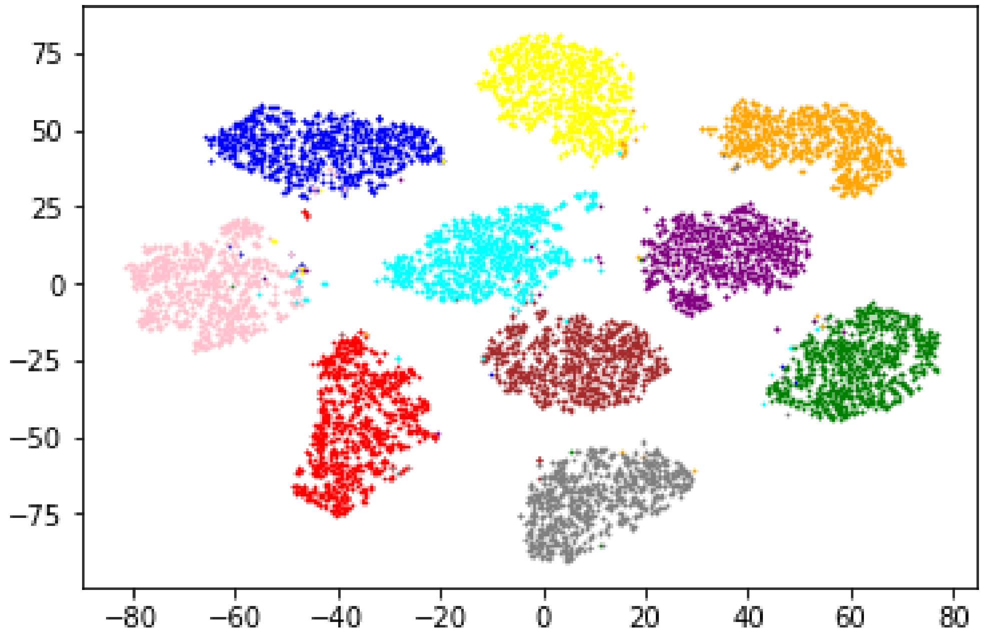

Figure 1: Test point embedding using t-SNE

References

[1] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, and Gang Wang. Recent advances in convolutional neural networks. *CoRR*, abs/1512.07108, 2015.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.