11/20/19

# 8805 Learning from Data: Lecture 25

Relevant notebooks for today: (in bayesian-methods-and-machine-learning)
(A) Bayesian_neural_networks_tif285.ipynb
(B) demo-Bayesian_neural_networks_tif285.ipynb
⇒ Mini-project-IIIb involves playing with and interpreting (B).

- Also note the Forssen_tif285_NeuralNet.ipynb
  _demo-NeuralNet.ipynb } notebooks
  - Overview of different neural net types and detailed background on backpropagation.

---

Stepping through (A)

Basic neural network

- each neuron has input $\vec{x} = \{x_i\}$ $i = 1,\ldots I$ and an output signal $y$, which depends nonlinearly on activation
$$a = w_0 + \sum_{i=1}^{I} w_i x_i \quad \text{with } \vec{w} = \{w_i\}_{i=1}^{I} \text{ the weights.}$$

- A loss function is identified for the problem; then training means feed with training data and adjust weights to minimize loss.

- Classification ⇒ single output $y$ is real number probability (so $\in [0,1]$) that input $\vec{x}$ belongs to one of two classes, $t=1$ or $t=0$:
$$\Rightarrow \quad y = p_{t=1} \equiv p(t=1 \mid \vec{w}, \vec{x})$$
$$1 - y = p_{t=0} \equiv p(t=0 \mid \vec{w}, \vec{x})$$

- A loss function for this problem could be (minimize wrt $\vec{w}$)
$$C_W(\vec{w}) = C(\vec{w}) + \alpha E_W(\vec{w})$$
"error function"

Here $C(\vec{w}) = -\sum_n \underbrace{t^{(n)} \log\left(y(\vec{x}^{(n)}, \vec{w})\right)}_{\text{if } t^{(n)}=1} + \underbrace{\left(1 - t^{(n)}\right) \log\left(1 - y(\vec{x}^{(n)}, \vec{w})\right)}_{\text{if } t^{(n)}=0}$

11/20/19

where $t^{(n)}$ is the training data (with corresponding $\vec{x}^{(n)}$) and $E_W(\vec{w}) = \frac{1}{2}\sum_i w_i^2$ is a regularizer, which inhibits overfitting by penalizing overly large weights (if they can get large, there can be fine cancellations to fit fluctuations).

We can interpret this loss function in terms of a familiar Bayesian statistical model for finding the weights given data D and a value for $\alpha$:

Bayes: $$p(\vec{w}|D,\alpha) = \frac{p(D|\vec{w})\, p(\vec{w}|\alpha)}{p(D|\alpha)}$$

where
$$p(D|\vec{w}) = e^{-G(\vec{w})}$$ tells us $G(\vec{w})$ is minus the log likelihood
$\Rightarrow p(D|\vec{w})$ is the product of the (independent) probabilities for each input-output pair

while
$$p(\vec{w}|\alpha) = \frac{1}{Z_W(\alpha)} e^{-\alpha E_W}$$
normalization

is the log prior pdf. If $E_W$ is quadratic, this is a Gaussian with variance $\sigma_W^2 = 1/\alpha$ and normalization $1/Z_W(\alpha) = (\alpha/2\pi)^{K/2}$ ($K = I+1$)

Then $$p(\vec{w}|D,\alpha) = \frac{1}{Z_M} e^{-[G(\vec{w}) + \alpha E_W(\vec{w})]} = \frac{1}{Z_M} e^{-C_W(\vec{w})}$$

The figure at this point shows some training data: 0, 2, 6, and 10 elements, and the resulting posterior.

- What do you observe? Note that $w_0$ is marginalized over.
- $N=0$ is just the prior, so what is $\alpha$?
- Note the separation of red and blue data.

- The ML approach is often to minimize $C_W(\vec{w})$ to find $\vec{w}^*$ MAP.

$\Rightarrow$ Bayesian: consider information in actual pdf.

Notation: $y$ is output from neural network. If a classification problem, $y$ is discrete categorical distributions of probabilities $p_{t=c}$ for class $c$. For regression, $y$ is continuous. If vector, then network maps (nonlinear) $y(\vec{x},\vec{w}): x \in \mathbb{R}^p \to y \in \mathbb{R}^m$.

11/20/19

Classification of uncertainties:

Epistemic — uncertainties in the model, so can be reduced with more data.

Aleatoric — from noise in the training data. E.g. Gaussian noise. More of the same data doesn't change the noise.

---

Probabilistic Model — recap 129

BNN is to infer $p(y|\vec{x},D) = \int p(y|\vec{x},\vec{w})\, p(\vec{w}|D)\, d\vec{w}$ by marginalization

new output, new input, $D = \{\vec{x}^{(i)}, y^{(i)}\}$ is a given training set

We need $p(\vec{w}|D) \propto p(D|\vec{w})\, p(\vec{w})$ by Bayes ← ordinary ML finds MAP value of $p(\vec{w}|D)$

Here $p(D|\vec{w}) = \prod_i p(y^{(i)}|\vec{x}^{(i)}, \vec{w})$ is the likelihood.

independent

- As before, $p(\vec{w})$ helps prevent overfitting by regularizing the weights.
- Calculating the marginalization integral over $\vec{w}$ is the same as averaging the predictions from an ensemble of NNs weighted by the posterior probabilities of their weights given the data (i.e. by $p(\vec{w}|D)$).

---

Now back to the binary ($t=1$ or $0$) classification problem.

The marginalization integral for the $(n+1)^{th}$ point is:

$$p(y^{(n+1)}|\vec{x}^{(n+1)}, D, \alpha) = \int d\vec{w}\; p(y^{(n+1)}|x^{(n+1)}, \vec{w}, \alpha)\; p(\vec{w}|D,\alpha)$$

could also marginalize over this. What would that look like?

The figures in the notebook show the classification in Bayesian (left panel) and regular (optimization) form. Here $\vec{x} = (x_1, x_2)$ [even if labeled wrong!] and $\alpha = 1.0$. The decision boundary is $y = 0.5 \Leftrightarrow a = 0$ with $a = \pm 1, \pm 2$ $\Rightarrow y = 0.12, 0.27, 0.73, 0.88$. Test data are pluses, training data are circles.

11/20/19

(I'm unclear about the colors on the right panel!)

- The Bayesian results is from sampling many neurons with different weights, distributed proportional to the posterior pdf. The decision boundary is from the mean of the sample predictions evaluated on a grid.
  - The next figure plots the <u>standard deviation</u> of the predictions.
  - Low sd long the \ diagonal where lots of training data (except near the origin)
  - At the origin, the prediction for class 1 is about ~0.5 but fairly certain ~0.2. In UL or LR corners, definite prediction and very small uncertainty about this certainty. UR and LL corners are about ~0.5 but very uncertain. So could be anything.

- Note that the regular binary classifier doesn't give us this kind of <u>uncertainty information</u>.

<u>Bayesian Neural Networks in practice</u>

Recap of methods for the marginalization integral:

1. Sampling, e.g. MCMC.
2. Analytic approximations, e.g. Gaussian approx. to Laplace method.
3. Variational method. This will be the method of choice for large numbers of parameters.

11/20/19

# Variational inference for Bayesian neural networks

optimize rather than sample

The basic idea is to approximate the true posterior by a parametrized posterior and adjust the parameters $\leftarrow \vec{\theta}$ to optimize the agreement.

- So use $q(\vec{w}|\vec{\theta})$ to approximate $p(\vec{w}|D)$, using $\theta = \theta^*$ as the optimal values.
- Use the Kullback-Leibler (KL) divergence as a measure for how close we are:

$q(\vec{w}|\vec{\theta}) \Rightarrow q(\vec{w})$
$p(\vec{w}|D) \Rightarrow p(\vec{w})$ here

expectation value wrt $q(\vec{w})$

$$D_{KL}(q\|p) = \int d\vec{w}\, q(\vec{w}) \log \frac{q(\vec{w})}{p(\vec{w})} \equiv \mathbb{E}_q\left[\log q(\vec{w}) - \log p(\vec{w})\right]$$

The variational property is that this quantity is $\geq 0$ and only equal to zero if $q(\vec{w}) = p(\vec{w})$.

We can prove that $D_{KL}(q\|p) \geq 0$ several ways. One of the easiest is to use that (try graphing it)

$$\log x \leq x - 1 \quad \text{for } x > 0$$

[It is easy to show this from $x \leq e^{x-1}$, considering the cases $x<1$ and $x>1$ separately.]

Then
$$-D_{KL}(q\|p) = \int d\vec{w}\, q(\vec{w}) \log\left(\frac{p(\vec{w})}{q(\vec{w})}\right)$$
$$\leq \int d\vec{w}\, q(\vec{w}) \left(\frac{p(\vec{w})}{q(\vec{w})} - 1\right)$$
because $p(\vec{w}), q(\vec{w}) \geq 0$. Handle $p(w)$ or $q(w) = 0$ separately.
$$= \int d\vec{w}\, p(\vec{w}) - \int d\vec{w}\, q(\vec{w}) = 1 - 1 = 0$$
$$\Rightarrow D_{KL}(q\|p) \geq 0 \quad \text{QED.}$$

$q(\vec{w}) = 0 \Rightarrow q \log q = 0$ or $q \log p$
$p(\vec{w}) = 0 \Rightarrow \log \frac{q}{p} \to \infty$ so $D_{KL} \to \infty$

The KL-divergence that is often seen in this context is $D_{KL}(p\|q) \neq D_{KL}(q\|p)$, but both have the variational feature.

- Here we favor $D_{KL}(q\|p)$ because the $q(\vec{w})$ distribution is known for taking expectation values.
- Note that minimizing:

$$D_{KL}(q\|p) = \int d\vec{w}\, q(\vec{w}|\vec{\theta}) \log \frac{q(\vec{w}|\vec{\theta})}{p(\vec{w}|D)} = \underbrace{-\overbrace{\int d\vec{w}\, q(\vec{w}|\vec{\theta}) \log p(\vec{w}|D)}^{\text{"cross entropy"}}}_{\text{avoid implausible parameters to minimize}} + \underbrace{\int d\vec{w}\, q(\vec{w}|\vec{\theta}) \log q(\vec{w}|\vec{\theta})}_{\text{maximize entropy of variational distribution } q}$$

11/20/19

**Evidence Lower Bound** ⇒ use Bayes Theorem on $p(\vec{w}|D)$ $= \frac{p(D|\vec{w})\,p(\vec{w})}{p(D)}$

$$D_{KL}(q\|p) = \int d\vec{w}\; q(\vec{w}|\vec{\theta})\left[\log q(\vec{w}|\vec{\theta}) - \log p(D|\vec{w}) - \log p(\vec{w}) + \log p(D)\right]$$

$$= \mathbb{E}_q[\log q(\vec{w}|\vec{\theta})] - \mathbb{E}_q[\log p(D|\vec{w})] - \mathbb{E}_q[\log p(\vec{w})] + \log p(D)$$

independent of $w$
⇒ just evidence

⇒ Evidence Lower Bound (ELBO)

$D_{KL}(q\|p) \geq 0 \Rightarrow$ $\log p(D) \geq -\mathbb{E}_q[\log q(\vec{w}|\vec{\theta})] + \mathbb{E}_q[\log p(D|\vec{w})] + \mathbb{E}_q[\log p(\vec{w})] \equiv J_{ELBO}(\vec{\theta})$

- $J_{ELBO}(\vec{\theta})$ is also called the variational free energy $F(D, \vec{\theta})$

Goal: find $\theta^*$ that maximizes $J_{ELBO}(\vec{\theta})$. Hardest term is $\mathbb{E}_q[\log p(D|\vec{w})]$
Recall $p(D|\vec{w}) = \prod_i p(y^{(i)}|\vec{x}^{(i)}; \vec{w}) \Rightarrow \sum_{i=1}^{N} \mathbb{E}_q[\log(p(y^{(i)}|\vec{x}^{(i)}, \vec{w}))]$

⇒ active research in improving how to find $\theta^*$.

---

**Bayesian neural networks in PyMC3**

do this next
- see demo-Bayesian_neural_networks_tif285.ipynb
  ⇒ uses Automatic Differentiation Variational Inference (ADVI)
- Corresponds to maximimizing the ELBO by modifying the hyperparameters in the variational distribution $q(\vec{w}|\vec{\theta})$

**Bayes by Backprop**. The terms in $J_{ELBO}$ are expectations wrt $q(\vec{w}|\vec{\theta})$.
Free energy $F(D, \vec{\theta}) \equiv -J_{ELBO}(\vec{\theta})$ is treated as a cost function ⇒ minimized.
Approximate by MC samples $\vec{w}^{(i)}$ from $q(\vec{w}|\vec{\theta})$

$$\Rightarrow F(D, \vec{\theta}) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\log q(\vec{w}^{(i)}|\vec{\theta}) - \log p(\vec{w}^{(i)}) - \log p(D|\vec{w}^{(i)})\right]$$

See blog post for example and the rest of this notebook for details.

11/20/19

(B) demo-Baysian_neural_networks_tif285.ipynb

- This should work as described using ordinary environment.yml for class if on Macs or Linux and the windows environment for Windows10.
- This demo uses ScikitLearn to set up the problem and pymc3 with theano to analyze. In other notebooks, Tensorflow and Keras are used. ⇒ focus here is on Bayesian interpretation.

we'll only minimally cover the implementation

- The introductory material is left for you to read (and ask questions about). Briefly look at highlights.

- Baysian Neural Networks in PyMC3

Generating data

- sklearn make_moons ⇒ note what noise, random_state, n_samples do
    - train_test_split to seperate into training and test data.
    - For mini-project IIIB, you'll try some different combinations. Plot training and test.
- Why is uncertainty important to have?

Art. Neural network specification

- distinguish layers from neurons in each layer (n_hidden is neuron)
- note dimensions of weight arrays (draw a picture)
- (How would you add a layer?)
- How would you add constant biases?

Running ADVI — uses automatic differentiation — no details for us!

- just note that the number of iterations is specified in pm.fit [n ⇒ iterations]
- How many iterations?
- Jump to predictions ⇒ see Mini-project IIIb

What has classifier learned?

- Focus on uncertainties as added value from BNN.