CSC321: Neural Networks

Lecture 9: Bayesian learning continued

Geoffrey Hinton

Bayes Theorem

joint probability

conditional probability

$$p(D)\,p(W \mid D) = p(D,W) = p(W)\,p(D \mid W)$$

Prior probability of weight vector W

Probability of observed data given W

$$p(W \mid D) = \frac{p(W)\,p(D \mid W)}{p(D)}$$

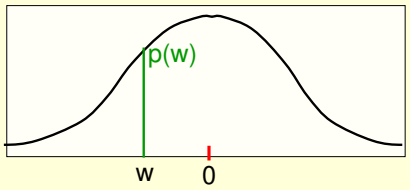
Posterior probability of weight vector W given training data D

$$p(D) = \int_W p(W)\,p(D \mid W)$$

$$Cost = -\log p(W \mid D) = -\log p(W) - \log p(D \mid W) + \log p(D)$$

Maximum A Posteriori Learning

- This trades-off the prior probabilities of the parameters against the probability of the data given the parameters. It looks for the parameters that have the greatest product of the prior term and the likelihood term.
- Minimizing the squared weights is equivalent to maximizing the log probability of the weights under a zero-mean Gaussian prior.



$$p(w) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{w^2}{2\sigma_W^2}}$$

$$-\log p(w) = \frac{w^2}{2\sigma_W^2} + k$$

The Bayesian interpretation of weight decay

$$-\log p(W \mid D) = -\log p(D \mid W) - \log p(W) + \log p(D)$$

$$C^* = \frac{1}{2\sigma_D^2}\sum_c (y_c - d_c)^2 + \frac{1}{2\sigma_W^2}\sum_i w_i^2$$

constant

assuming that the model makes a Gaussian prediction

assuming a Gaussian prior for the weights

$$C = E + \frac{\sigma_D^2}{\sigma_W^2}\sum_i w_i^2$$

So the correct value of the weight decay parameter is the ratio of two variances. Its not just an arbitrary hack.

Full Bayesian Learning

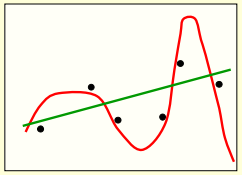
- Instead of trying to find the best single setting of the parameters (as in ML or MAP) compute the full posterior distribution over parameter settings
  - This is extremely computationally intensive for all but the simplest models (its feasible for a biased coin).
- To make predictions, let each different setting of the parameters make its own prediction and then combine all these predictions by weighting each of them by the posterior probability of that setting of the parameters.
  - This is also computationally intensive.
- The full Bayesian approach allows us to use complicated models even when we do not have much data

Overfitting: A frequentist illusion?

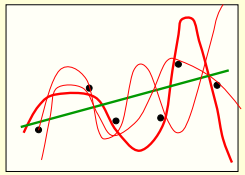
- If you do not have much data, you should use a simple model, because a complex one will overfit.
  - This is true. But only if you assume that fitting a model means choosing a single best setting of the parameters.
  - If you use the full posterior over parameter settings, overfitting disappears!
  - With little data, you get very vague predictions because many different parameters settings have significant posterior probability

A classic example of overfitting

- Which model do you believe?
  - The complicated model fits the data better.
  - But it is not economical and it makes silly predictions.



- But what if we start with a reasonable prior over all fifth-order polynomials and use the full posterior distribution.
  - Now we get vague and sensible predictions.
- There is no reason why the amount of data should influence our prior beliefs about the complexity of the model.



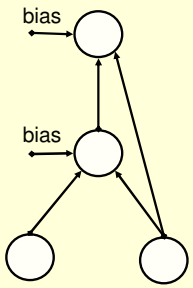
Approximating full Bayesian learning in a neural network

- If the neural net only has a few parameters we could put a grid over the parameter space and evaluate p(W | D) at each grid-point.
  - This is expensive, but it does not involve any gradient descent and there are no local optimum issues.
- After evaluating each grid point we use all of them to make predictions on test data
  - This is also expensive, but it works much better than ML learning when the posterior is vague or multimodal (this happens when data is scarce).

$$p(d_{test} \mid input_{test}) = \sum_{g \ \varepsilon \ grid} p(W_g \mid D) \;\; p(d_{test} \mid input_{test}, W_g)$$

An example of full Bayesian learning

- Allow each of the 6 weights or biases to have the 9 possible values [-2 : 0.5 : 2]
  - So there are 9^6 grid-points in parameter space.
- For each grid-point compute the probability of the observed outputs of all the training cases.
  - This is the likelihood term and is explained on the next slide
- Multiply the prior for each grid-point by the likelihood term and renormalize to get the posterior probability for each grid-point.
- Make predictions by using the posterior probabilities to average the predictions made by the different grid-points.



A neural net with 2 inputs, 1 output and 6 parameters

Computing the likelihood term for a logistic output unit

- The output of the logistic unit is the probability that the network assigns to the answer 1. It assigns the complementary probability to the answer 0.

$$y_c = f(input_c, W_g)$$

if d=1 if d=0

$$p(output = d_c \mid input_c, W_g) = d_c y_c + (1 - d_c)(1 - y_c)$$

$$p(all\ training\ outputs \mid W_g) = \prod_c p(output = d_c \mid input_c, W_g)$$

What can we do if there are too many parameters for a grid to be feasible?

- The number of grid points is exponential in the number of parameters.
  - So we cannot deal with more than a few parameters using a grid.
- If there is enough data to make most parameter vectors very unlikely, only need a tiny fraction of the grid points make a significant contribution to the predictions.
  - Maybe we can just evaluate this tiny fraction
- It might be good enough to just sample weight vectors according to their posterior probabilities.

$$p(y_{test} \mid input_{test}, D) = \sum_i p(W_i \mid D)\ p(y_{test} \mid input_{test}, W_i)$$

Sample weight vectors with this probability

One method for sampling weight vectors

- In standard backpropagation we keep moving the weights in the direction that decreases the cost
  - i.e. the direction that increases the log likelihood plus the log prior, summed over all training cases.
- Suppose we add some Gaussian noise to the weight vector after each update.
  - So the weight vector never settles down.
  - It keeps wandering around, but it tends to prefer low cost regions of the weight space.
- Amazing fact: If we use just the right amount of noise, and if we let the weight vector wander around for long enough before we take a sample, we will get a sample from the true posterior over weight vectors.
  - This is called a "Markov Chain Monte Carlo" method and it makes it feasible to use full Bayesian learning with hundreds or thousands of parameters.
  - There are related MCMC methods that are more complicated but more efficient (we don't need to let the weights wander around for so long before we get samples from the posterior).