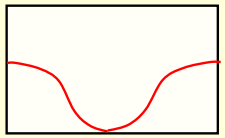
CSC 2535 Winter 2006

Lecture 6a: Improving generalization by making the weights cheap to describe.

Geoffrey Hinton

Another kind of weight penalty

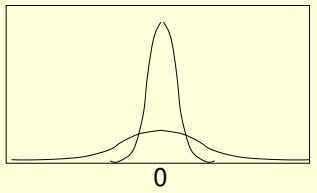
- Sometimes it works better to use a weight penalty that has negligible effect on large weights.



- We can easily make up a heuristic cost function for this.

$$C(w) = \frac{\lambda}{1 + k\, w^2}$$

- But we get more insight if we view it as the negative log probability under a mixture of two zero-mean Gaussians

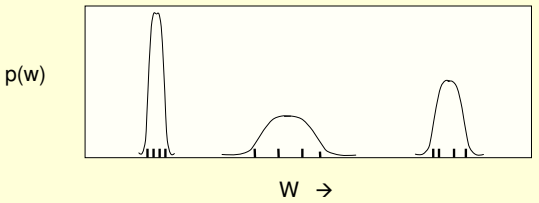


Soft weight-sharing

- Le Cun showed that networks generalize better if we constrain subsets of the weights to be equal.
  - This removes degrees of freedom from the parameters so it simplifies the model.
- But for most tasks we do not know in advance which weights should be the same.
  - Maybe we can learn which weights should be the same.

Modeling the distribution of the weights

- The values of the weights form a distribution in a one-dimensional space.
  - If the weights are tightly clustered, they have high probability density under a mixture of Gaussians model.
  - To raise the probability density move each weight towards its nearest cluster center.



Fitting the weights and the mixture prior together

- We can alternate between two types of update:
  - Adjust the weights to reduce the error in the output and to increase the probability density of the weights under the mixture prior.
  - Adjust the means and variances and mixing proportions in the mixture prior to fit the posterior distribution of the weights better.
- This automatically clusters the weights.
  - We do not need to specify in advance which weights should belong to the same cluster.

A different optimization method

- Alternatively, we can just apply conjugate gradient descent to all of the parameters in parallel (which is what we did).
  - To keep the variance positive, use the log variances in the optimization (these are the natural parameters for a scale variable).
  - To ensure that the mixing proportions of the Gaussians sum to 1, use the parameters of a softmax in the optimization.

$$\pi_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

The cost function and its derivatives

negative log probability of desired output under a Gaussian whose mean is the output of the net

Probability of weight i under Gaussian j

$$C = \frac{k}{\sigma_{out}^2} \sum_c \tfrac{1}{2}(y_c - \hat{y}_c)^2 \quad - \sum_i \log \sum_j \pi_j p(w_i \mid \mu_j, \sigma_j^2)$$

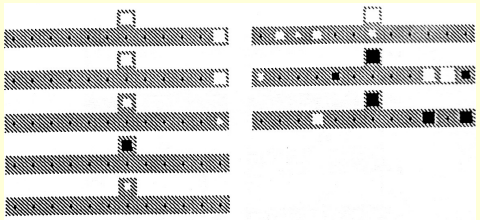
$$\frac{\partial C}{\partial w_i} = \frac{k}{\sigma_{out}^2} \sum_c (y_c - \hat{y}_c)^2 \frac{\partial \hat{y}_c}{\partial w_i} \quad - \sum_j p(j \mid w_i) \frac{\mu_j - w_i}{\sigma_j^2}$$

posterior probability of Gaussian j given weight i

The sunspot prediction problem

- Predicting the number of sunspots next year is important because they affect weather and communications.
- The whole time series only less than 400 points and there is no obvious way to get any more data.
  - So it is worth using computationally expensive methods to get good predictions.
- The best model produced by statisticians was a combination of two linear autoregressive models that switched at a particular threshold value.
  - Heavy-tailed weight decay works better.
  - Soft weight-sharing using a mixture of Gaussians prior works even better.

The weights learned by the eight hidden units for predicting the number of sunspots

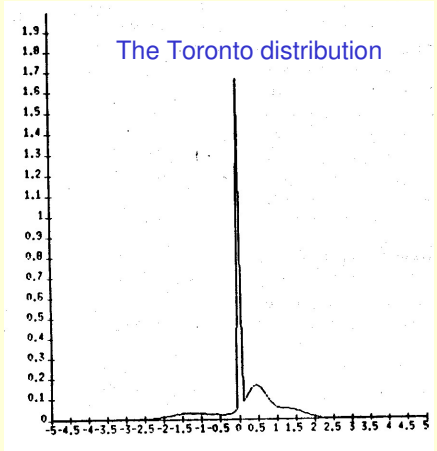


Rule 1: (uses 2 units): High if high last year.

Rule 2: High if high 6, 9, or 11 years ago.

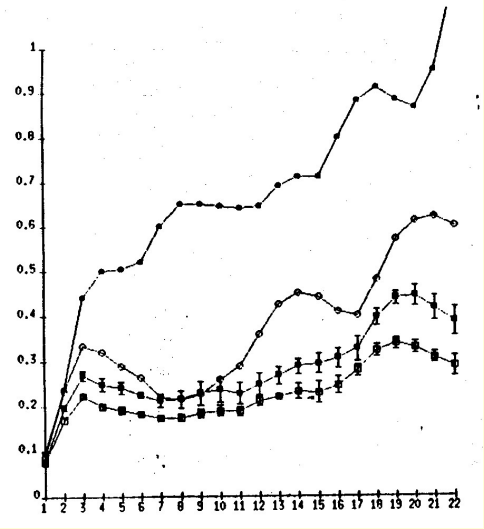
Rule 3: Low if low 1 or 8 ago & high 2 or 3 ago.

Rule 4: Low if high 9 ago and low 1 or 3 ago



The mixture of five Gaussians learned for clustering the weights.

Weights near zero are very cheap because they have high density under the empirical prior.



Predicting sunspot numbers far into the future by iterating the single year predictions.

The net with soft weight-sharing gives the lowest errors

The problem with soft weight-sharing

- It constructs a sensible empirical prior for the weights.
- But it ignores the fact that some weights need to be coded accurately and others can be very imprecise without having much effect on the squared error.
- A coding framework needs to model the number of bits required to code the value of a weight and this depends on the precision as well as the value.

Using the variational approach to make Bayesian learning efficient

- Consider a standard backpropagation network with one hidden layer and the squared error function.
- The full Bayesian approach to learning is:
  - Start with a prior distribution across all possible weight vectors
  - Multiply the prior for each weight vector by the probability of the observed outputs given that weight vector and then renormalize to get the posterior distribution.
  - Use this posterior distribution over all possible weight vectors for making predictions.
- This is not feasible for large nets. Can we use a tractable approximation to the posterior?

An independence assumption

- We can approximate the posterior distribution by assuming that it is an axis-aligned Gaussian in weight space.
  - i.e. we give each weight its own posterior variance.
  - Weights that are not very important for minimizing the squared error will have big variances.
- This can be interpreted nicely in terms of minimum description length.
  - Weights with high posterior variances can be communicated in very few bits
    - This is because we can use lots of entropy to pick a precise value from the posterior, so we get lots of "bits back".

Communicating a noisy weight

- First pick a precise value for the weight from its posterior.
  - We will get back a number of bits equal to the entropy of the weight
    - We could imagine quantizing with a very small quantization width to eliminate the infinities.
- Then code the precise value under the Gaussian prior.
  - This costs a number of bits equal to the cross-entropy between the posterior and the prior.

$$\left(-\int Q(w)\log P(w)\,dw\right) - \left(-\int Q(w)\log Q(w)\,dw\right)$$

expected number of bits to send weight — expected number of bits back

The cost of communicating a noisy weight

- If the sender and receiver agree on a prior distribution, P, for the weights, the cost of communicating a weight with posterior distribution Q is:

$$KL(Q\,\|\,P) = \int Q(w)\log\frac{Q(w)}{P(w)}\,dw$$

If the distributions are both Gaussian this cost becomes:

$$KL(Q\,\|\,P) = \log\frac{\sigma_P}{\sigma_Q} + \frac{1}{2\sigma_P^2}\left[\sigma_Q^2 - \sigma_P^2 + (\mu_P - \mu_Q)^2\right]$$

What do noisy weights do to the expected squared error?

- Consider a linear neuron with a single input.
  - Let the weight on this input be stochastic
- The noise variance for the weight gets multiplied by the squared input value and added to the squared error:

$$y = xw$$ ← Stochastic output of neuron

$$< xw > = x\mu_w$$ ← mean of weight distribution

$$< (xw)^2 > = x^2\mu_w^2 + x^2\sigma_w^2$$

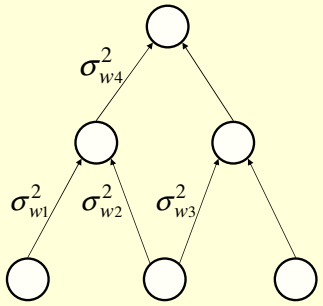
$$E = (d - y)^2 = d^2 - 2dy + y^2$$

$$< E > = (d - x\mu_w)^2 + x^2\sigma_w^2$$

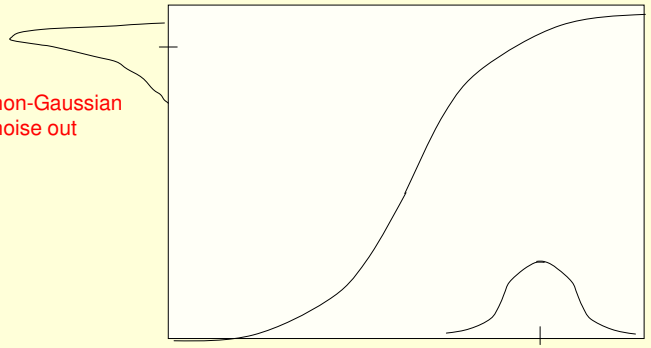
Extra squared error caused by noisy weight

How to deal with the non-linearity in the hidden units

- The noise on the incoming connections to a hidden unit is independent so its variance adds.
- This Gaussian input noise to a hidden unit turns into non-Gaussian output noise, but we can use a big table to find the mean and variance of this non-Gaussian noise.
- The non-Gaussian noise coming out of each hidden unit is independent so we can just add up the variances coming into an output unit.

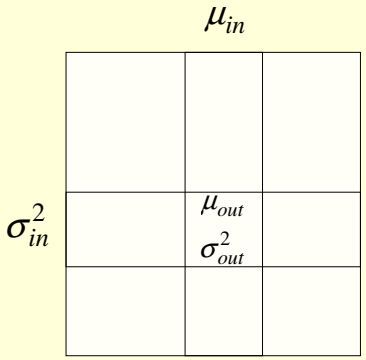


The mean and variance of the output of a logistic hidden unit



The forward table

- The forward table is indexed by the mean and the variance of the Gaussian total input to a hidden unit.
- It returns the mean and variance of the non-Gaussian output.
  - This non-Gaussian mean and variance is all we need to compute the expected squared error.



The backward table

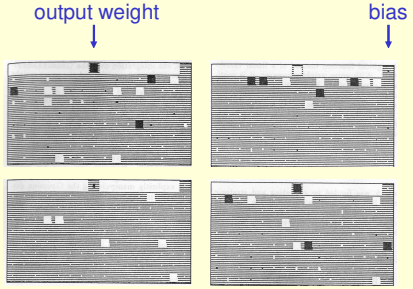
- The backward table is indexed by the mean and variance of the total input.
- It returns four partial derivatives which are all we need for backpropagating the derivatives of the squared error to the input →hidden weights.

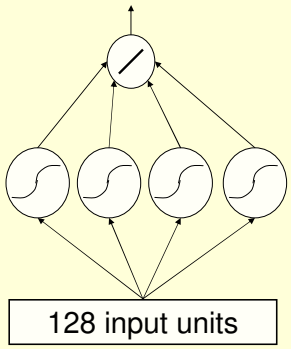
$$\frac{\partial \mu_{out}}{\partial \mu_{in}}, \quad \frac{\partial \sigma^2_{out}}{\partial \mu_{in}}, \quad \frac{\partial \mu_{out}}{\partial \sigma^2_{in}}, \quad \frac{\partial \sigma^2_{out}}{\partial \sigma^2_{in}}$$

Empirical Bayes: Fitting the prior

- We can now trade off the precision of the weights against the extra squared error caused by noisy weights.
  - even though the residuals are non-Gaussian we can choose to code them using a Gaussian.
- We can also learn the width of the prior Gaussian used for coding the weights.
- We can even have a mixture of prior Gaussians
  - This allows the posterior weights to form clusters
    - Very good for coding lots of zero weights precisely without using many bits.
    - Also makes large weights cheap if they are the same as other large weights.

Some weights learned by variational bayes



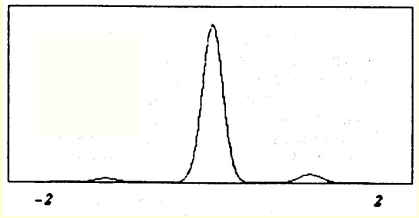


- It learns a few big positive weights, a few big negative weights, and lots of zeros. It has found four "rules" that work well.

Only 105 training cases to train 521 weights

The learned empirical prior for the weights

- The posterior for the weights needs to be Gaussian to make it possible to figure out the extra squared error caused by noisy weights and the cost of coding the noisy weights.



- The learned prior can be a mixture of Gaussians. This learned prior is a mixture of 5 Gaussians has 14 parameters