CSC2535: 2011
Advanced Machine Learning

Lecture 2: Variational Inference and Learning in Directed Graphical Models

Geoffrey Hinton

An apparently crazy idea

- Its hard to learn directed models in which varaibles have a large number of parents because its hard to infer (or sample from) the posterior distribution over hidden configurations (i.e. the joint distribution of the latent variables).
- Crazy idea: do inference wrong.
  - Maybe we can show that learning will still work.

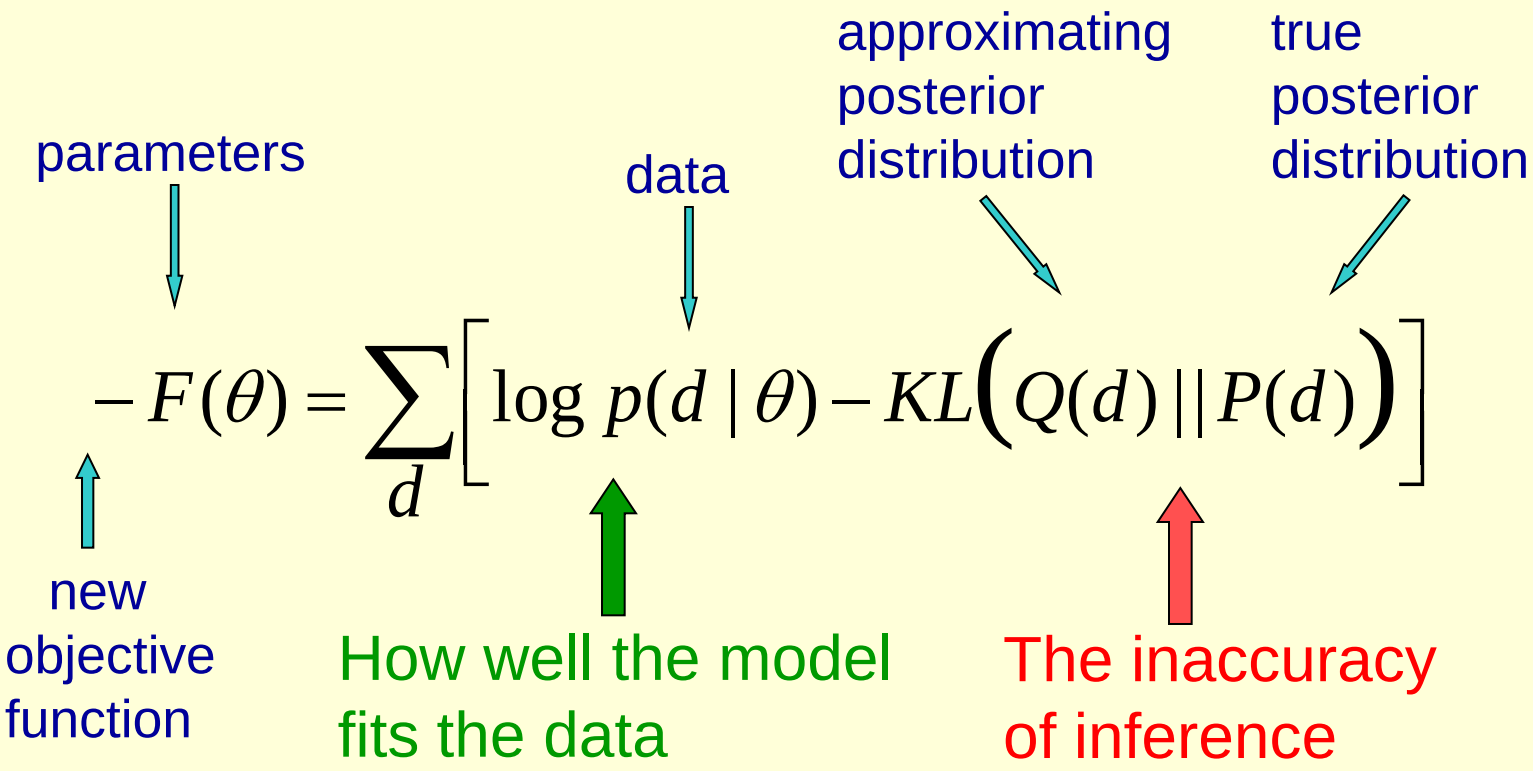
Approximate inference

- For models like sigmoid belief nets, it is intractable to compute the exact posterior distribution over hidden configurations. So what happens if we use a tractable approximation to the posterior?
  - e.g. assume the posterior over hidden configurations for each datavector factorizes into a product of distributions for each separate hidden cause.

- If we use the approximation for learning, there is no guarantee that learning will increase the probability that the model would generate the observed data.

- But maybe we can find a different and sensible objective function that is guaranteed to improve at each update of the parameters.

A trade-off between how well the model fits the data and the accuracy of inference



This makes it feasible to fit very complicated models, but the approximations that are tractable may be poor.

Two ways to derive F

- We can derive variational free energy as the objective function that is minimized by both steps of the Expectation and Maximization algorithm (EM).

- We can also derive it by using Minimum Description Length ideas.

Overview

- Clustering with K-means and a proof of convergence that uses energies.
- Clustering with a mixture of Gaussians and a proof of convergence that uses free energies.

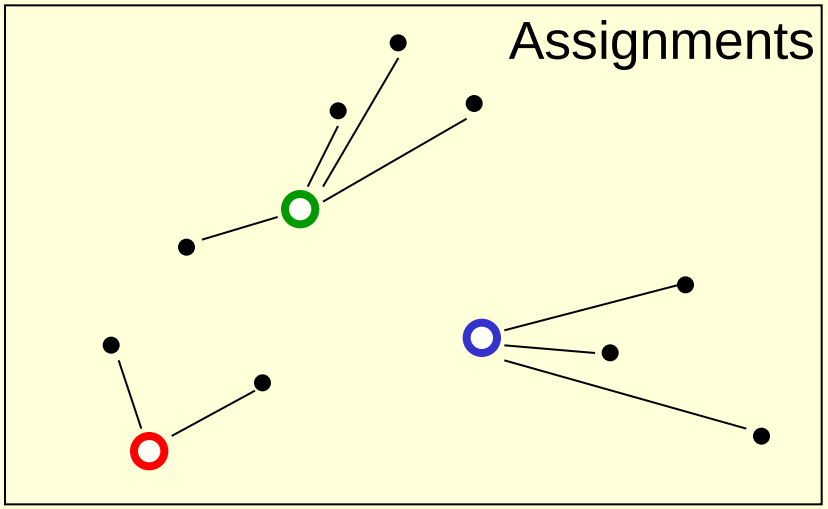
- The MDL view of clustering and the bits-back argument
- The MDL justification for incorrect inference.
- The MDL view of SBN's
- The wake-sleep algorithm.

Clustering

- We assume that the data was generated from a number of different classes. The aim is to cluster data from the same class together.
  - Why not put each datapoint into a separate class?
- What is the objective function that is optimized by sensible clusterings?

The k-means algorithm

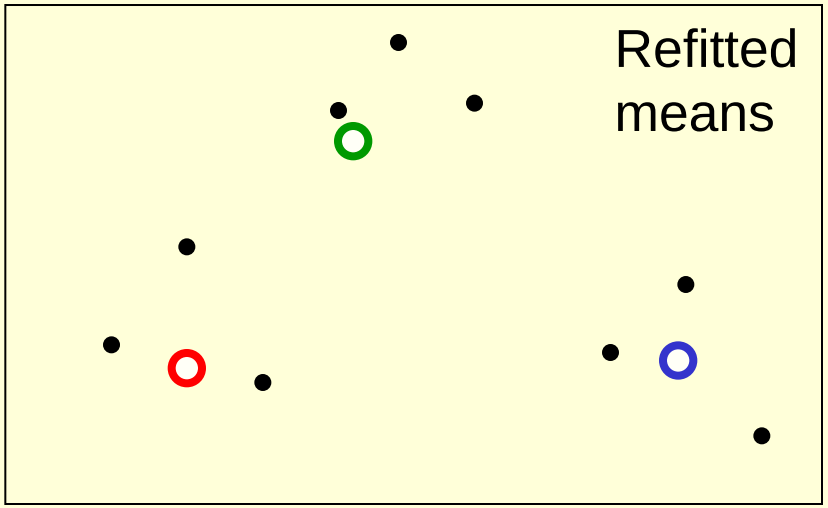
- Assume the data lives in a Euclidean space.
- Assume we want k classes.
- Assume we start with randomly located cluster centers



The algorithm alternates between two steps:

Assignment step: Assign each datapoint to the closest cluster.

Refitting step: Move each cluster center to the center of gravity of the data assigned to it.



Why K-means converges

- Whenever an assignment is changed, the sum squared distances of datapoints from their assigned cluster centers is reduced.
- Whenever a cluster center is moved the sum squared distances of the datapoints from their currently assigned cluster centers is reduced.
- Test for convergence: If the assignments do not change in the assignment step, we have converged.

A generative view of clustering

- We need a sensible measure of what it means to cluster the data well.
  - This makes it possible to judge different methods.
  - It may make it possible to decide on the number of clusters.
- An obvious approach is to imagine that the data was produced by a generative model.
  - Then we can adjust the parameters of the model to maximize the probability that it would produce exactly the data we observed.

The mixture of Gaussians generative model

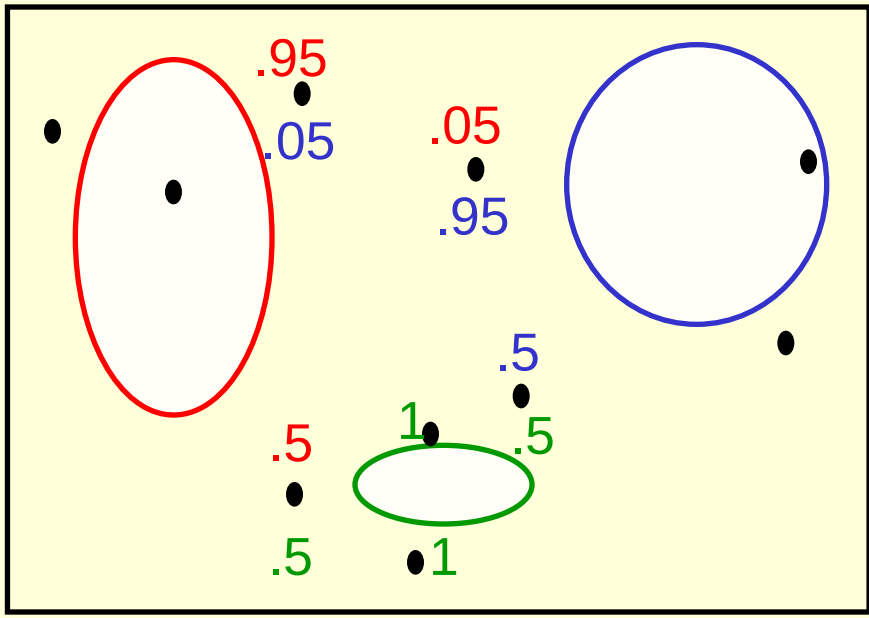
- First pick one of the k Gaussians with a probability that is called its “mixing proportion”.
- Then generate a random point from the chosen Gaussian.
- The probability of generating the exact data we observed is zero, but we can still try to maximize the probability density.
  - Adjust the means of the Gaussians
  - Adjust the variances of the Gaussians on each dimension.
  - Adjust the mixing proportions of the Gaussians.

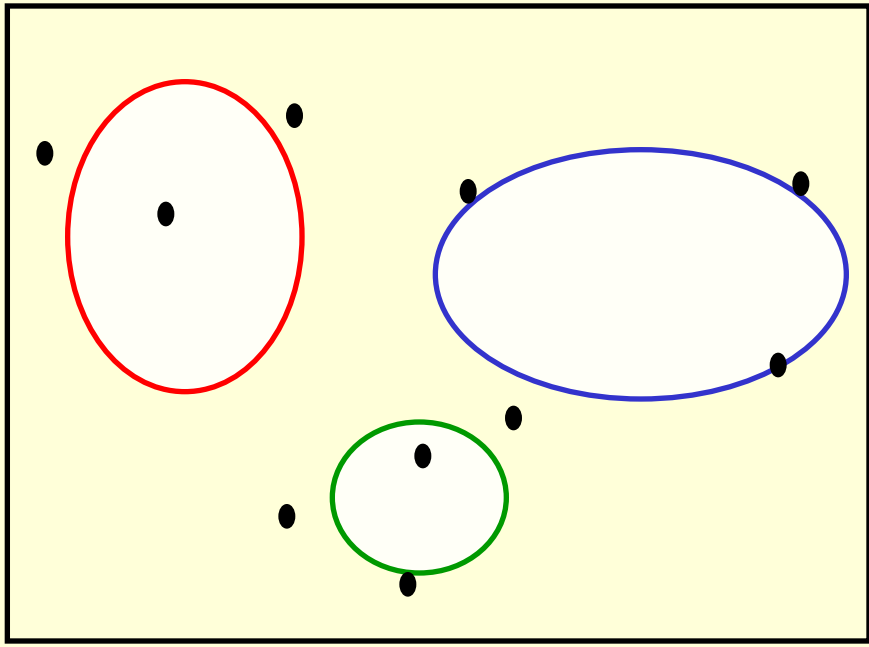
Fitting a mixture of Gaussians

The EM algorithm alternates between two steps:

E-step: Compute the posterior probability that each Gaussian generates each datapoint.

M-step: Assuming that the data really was generated this way, change the parameters of each Gaussian to maximize the probability that it would generate the data it is currently responsible for.





The E-step: Computing responsibilities

- In order to adjust the parameters, we must first solve the inference problem: Which Gaussian generated each datapoint?
  - We cannot be sure, so it's a distribution over all possibilities.
- Use Bayes theorem to get posterior probabilities for an axis aligned Gaussian.

Posterior for Gaussian i

Prior for Gaussian i

$$p(i \mid \mathbf{x}^c) = \frac{p(i)\, p(\mathbf{x}^c \mid i)}{p(\mathbf{x}^c)}$$

Bayes theorem

$$p(\mathbf{x}^c) = \sum_j p(j)\, p(\mathbf{x}^c \mid j)$$

$$p(i) = \pi_i$$

Mixing proportion

$$p(\mathbf{x}^c \mid i) = \prod_{d=1}^{d=D} \frac{1}{\sqrt{2\pi}\sigma_{i,d}} e^{-\frac{\| x_d^c - \mu_{i,d} \|^2}{2\sigma_{i,d}^2}}$$

Product over all data dimensions

The M-step: Computing new mixing proportions

- Each Gaussian gets a certain amount of posterior probability for each datapoint.
- The optimal mixing proportion to use (given these posterior probabilities) is just the fraction of the data that the Gaussian gets responsibility for.

Posterior for Gaussian i

Data for training case c

$$\pi_i^{new} = \frac{\sum_{c=1}^{c=N} p(i \mid \mathbf{x}^c)}{N}$$

Number of training cases

More M-step: Computing the new means

- We just take the center-of gravity of the data that the Gaussian is responsible for.
  - Just like in K-means, except the data is weighted by the posterior probability of the Gaussian.
  - Guaranteed to lie in the convex hull of the data
    - Could be big initial jump

$$\boldsymbol{\mu}_i^{new} = \frac{\sum_c p(i \mid \mathbf{x}^c)\, \mathbf{x}^c}{\sum_c p(i \mid \mathbf{x}^c)}$$

More M-step: Computing the new variances

- We fit the variance of each Gaussian, i, on each dimension, d, to the posterior-weighted data
  - Its more complicated if we use a full-covariance Gaussian that is not aligned with the axes.

$$\sigma_{i,d}^2 = \frac{\sum_c p(i \mid \mathbf{x}^c) \; \| \, x_d^c - \mu_{i,d}^{new} \, \|^2}{\sum_c p(i \mid \mathbf{x}^c)}$$

How do we know that the updates improve things?

- Updating each Gaussian definitely improves the probability of generating the data if we generate it from the same Gaussians after the parameter updates.
  - But we know that the posterior will change after updating the parameters.
- A good way to show that this is OK is to show that there is a single function that is improved by both the E-step and the M-step.
  - The function we need is called Free Energy.

Why EM converges

- There is a cost function that is reduced by both the E-step and the M-step.

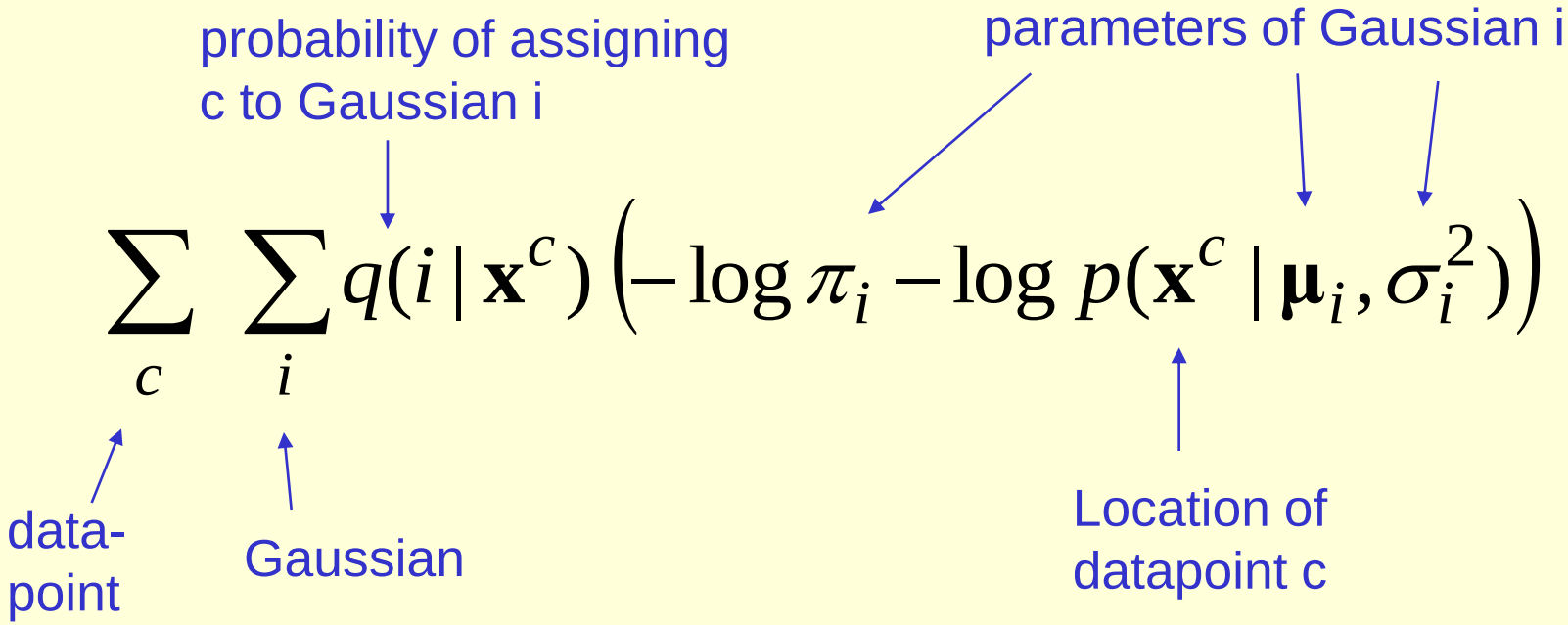
  Cost = expected energy – entropy

- The expected energy term measures how difficult it is to generate each datapoint from the Gaussians it is assigned to. It would be happiest assigning each datapoint to the Gaussian that generates it most easily (as in K-means).

- The entropy term encourages “soft” assignments. It would be happiest spreading the assignment probabilities for each datapoint equally between all the Gaussians.

The expected energy of a datapoint

- The expected energy of datapoint c is the average negative log probability of generating the datapoint
  - The average is taken using the probabilities of assigning the datapoint to each Gaussian. We can use any probabilities we like.



The entropy term

- This term wants the assignment probabilities to be as uniform as possible.
- It fights the expected energy term.

$$entropy = -\sum_c \sum_i q(i \mid \mathbf{x}^c) \log q(i \mid \mathbf{x}^c)$$

log probabilities are always negative

The E-step chooses the assignment probabilities that minimize the cost function (with the parameters of the Gaussians held fixed)

- How do we find assignment probabilities for a datapoint that minimize the cost and sum to 1?
- The optimal solution to the trade-off between expected energy and entropy is to make the probabilities be proportional to the exponentiated negative energies:

$$energy\ of\ \ assigning\ c\ to\ i\ \ =\ \ -\log \pi_i - \log\, p(\mathbf{x}^c \,|\, \boldsymbol{\mu}_i, \sigma_i^2)$$

$$optimal\ value\ of\ \ q(i \,|\, \mathbf{x}^c)\ \ \propto\ \ \exp(-energy)$$

$$\propto\ \ \pi_i\ \ p(\mathbf{x}^c \,|\, i)$$

- So using the posterior probabilities as assignment probabilities minimizes the cost function!

The M-step chooses the parameters that minimize the cost function
(with the assignment probabilities held fixed)

- This is easy. We just fit each Gaussian to the data weighted by the assignment probabilities that the Gaussian has for the data.
  - When you fit a Gaussian to data you are maximizing the log probability of the data given the Gaussian. This is the same as minimizing the energies of the datapoints that the Gaussian is responsible for.
  - If a Gaussian is assigned a probability of 0.7 for a datapoint the fitting treats it as 0.7 of an observation.

- Since both the E-step and the M-step decrease the same cost function, EM converges.

EM as coordinate descent in Free Energy

$$F(\mathbf{x}^c) = \sum_i q(i \mid \mathbf{x}^c)\left(-\log \pi_i - \log p(\mathbf{x}^c \mid i)\right) \quad - \quad \sum_i q(i \mid \mathbf{x}^c)\left(-\log q(i \mid \mathbf{x}^c)\right)$$

- Think of each different setting of the hidden and visible variables as a “configuration”. The energy of the configuration has two terms:
  - The log prob of generating the hidden values
  - The log prob of generating the visible values from the hidden ones
- The E-step minimizes F by finding the best distribution over hidden configurations for each data point.
- The M-step holds the distribution fixed and minimizes F by changing the parameters that determine the energy of a configuration.

The advantage of using F to understand EM

- There is clearly no need to use the optimal distribution over hidden configurations.
  - We can use any distribution that is convenient so long as:
    - we always update the distribution in a way that improves F
    - We change the parameters to improve F given the current distribution.
- This is very liberating. It allows us to justify all sorts of weird algorithms.

An incremental EM algorithm for fitting a mixture of Gaussians

- The idea of this algorithm is to do “online” fitting of a mixture of Gaussians
  - Look at one datapoint at a time and update the parameters after each datapoint.
- When we update the parameters of the Gaussians, the posteriors change for all datapoints.
  - The standard EM algorithm would have to update the posteriors for all datapoints after any change in the parameters
- The variational approach allows us to dispense with the updates of the posteriors for all of the other datapoints.
  - Those datapoints will now be using an out-of-date posterior, but that’s OK. We can do whatever we like to the posteriors so long as it does not increase F.

Initialization

- We can start with any initial distribution we like across the Gaussians for each datapoint.
  - But it would make sense to do one pass through all the data setting the distribution across Gaussians to be the posterior for each datapoint.
- We also need to initialize some sums over all datapoints (lets ignore the variance update).

$$\sum_c p^{init}(i \mid \mathbf{x}^c), \quad \sum_c p^{init}(i \mid \mathbf{x}^c)\,\mathbf{x}^c, \quad \boldsymbol{\mu}_i^{init} = \frac{\sum_c p^{init}(i \mid \mathbf{x}^c)\,\mathbf{x}^c}{\sum_c p^{init}(i \mid \mathbf{x}^c)}$$

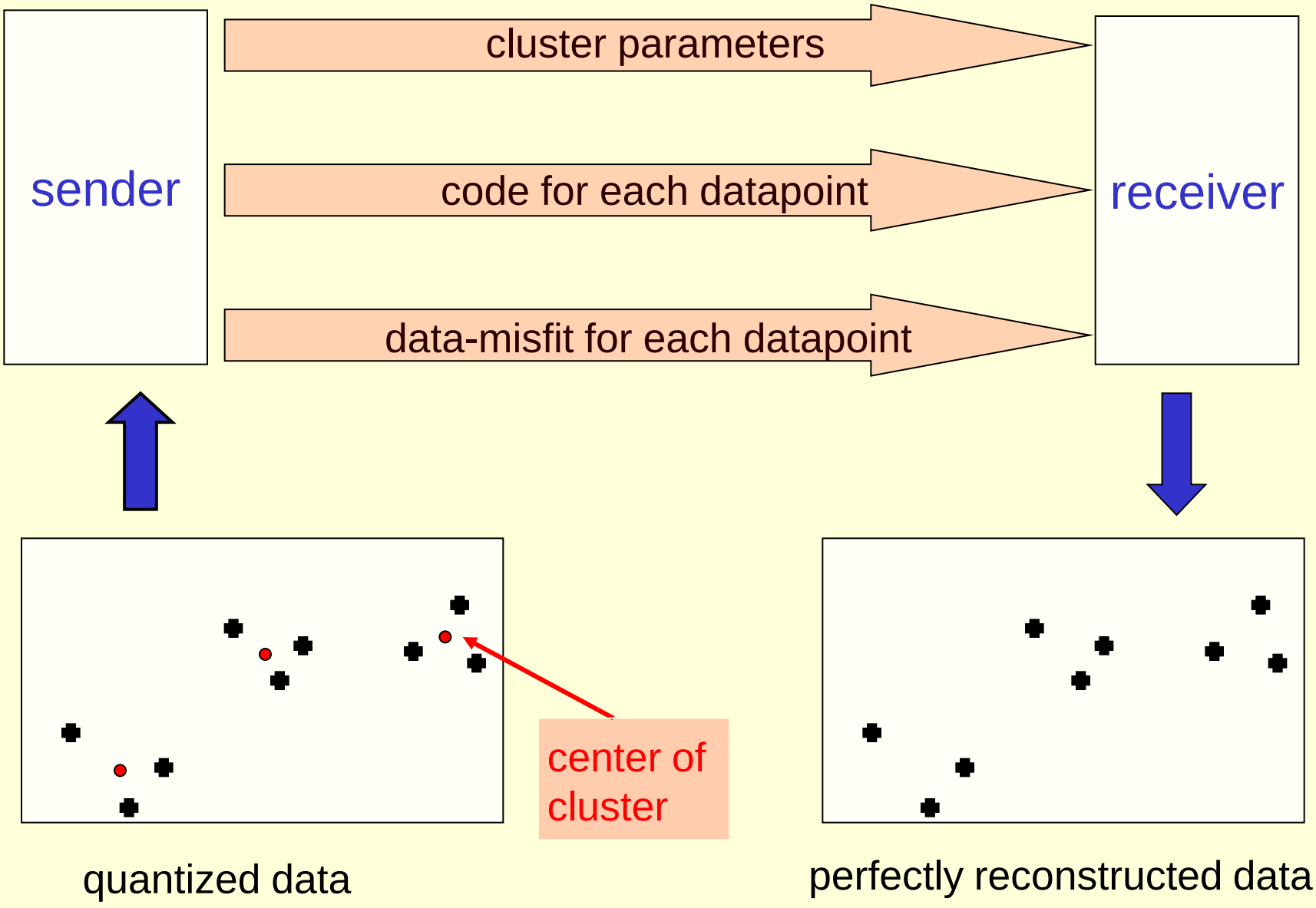
Updating the parameters

- Partial E-step: Look at a single datapoint, d, and compute the posterior distribution $p^{new}(i \mid \mathbf{x}^d)$ for d given the current parameters.
- M-step: Compute the effect on the parameters of changing the distribution for d, whilst keeping all the other approximate posteriors for all other datapoints fixed.

stored sum

$$\boldsymbol{\mu}_i^{new(d)} = \frac{p^{new}(i \mid \mathbf{x}^d)\,\mathbf{x}^d - p^{old}(i \mid \mathbf{x}^d)\,\mathbf{x}^d + \sum_c p^{old}(i \mid \mathbf{x}^c)\,\mathbf{x}^c}{p^{new}(i \mid \mathbf{x}^d) - p^{old}(i \mid \mathbf{x}^d) + \sum_c p^{old}(i \mid \mathbf{x}^c)}$$

An MDL approach to clustering



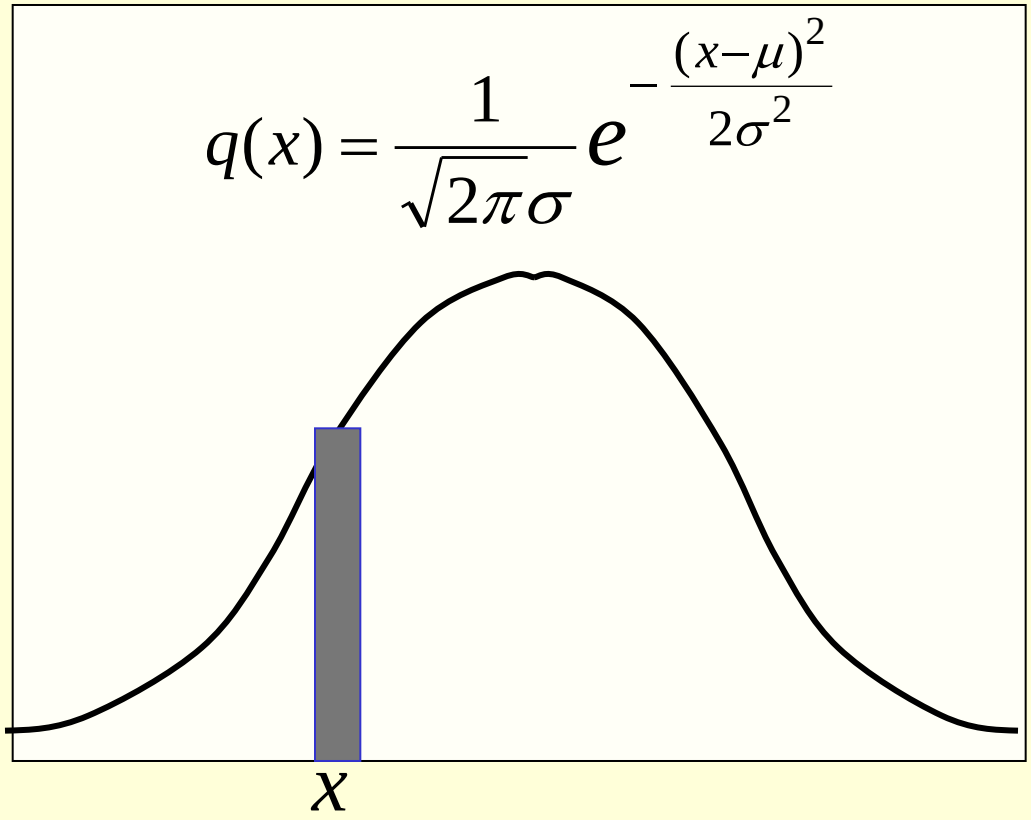
How many bits must we send?

- Model parameters:
  - It depends on the priors and how accurately they are sent.
  - Lets ignore these details for now
- Codes:
  - If all n clusters are equiprobable, log n
    - This is extremely plausible, but wrong!
  - We can do it in less bits
    - This is extremely implausible but right.
- Data misfits:
  - If sender & receiver assume a Gaussian distribution within the cluster, -log[p(d)|cluster] which depends on the squared distance of d from the cluster center.

Using a Gaussian agreed distribution

- Assume we need to send a value, x, with a quantization width of t
- This requires a number of bits that depends on

$$\frac{(x-\mu)^2}{2\sigma^2}$$



$$-\log(\ prob.mass) \approx -\log(t\ q(x))$$

$$\approx -\log(t) + \log(\sqrt{2\pi}\sigma) + \frac{(x-\mu)^2}{2\sigma^2}$$

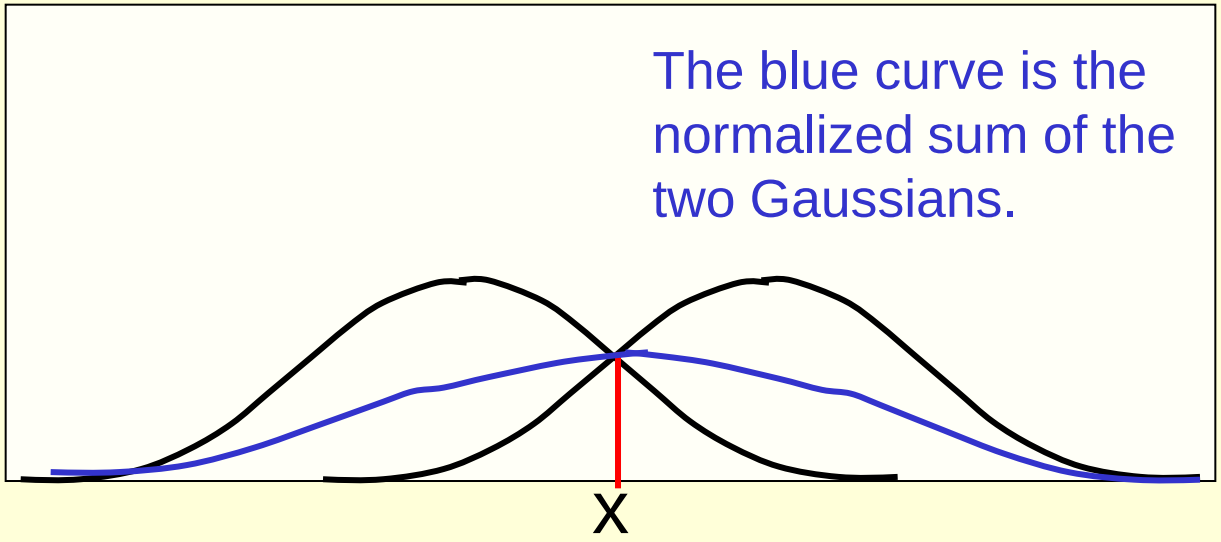
What is the best variance to use?

$$C \approx \sum_{c=1}^{N} -\log(t) + \log(\sqrt{2\pi}\sigma) + \frac{(x_c - \mu)^2}{2\sigma^2}$$

$$\frac{\partial C}{\partial \sigma} = \frac{N}{\sigma} - \frac{1}{\sigma^3}\sum_{c}(x_c - \mu)^2$$

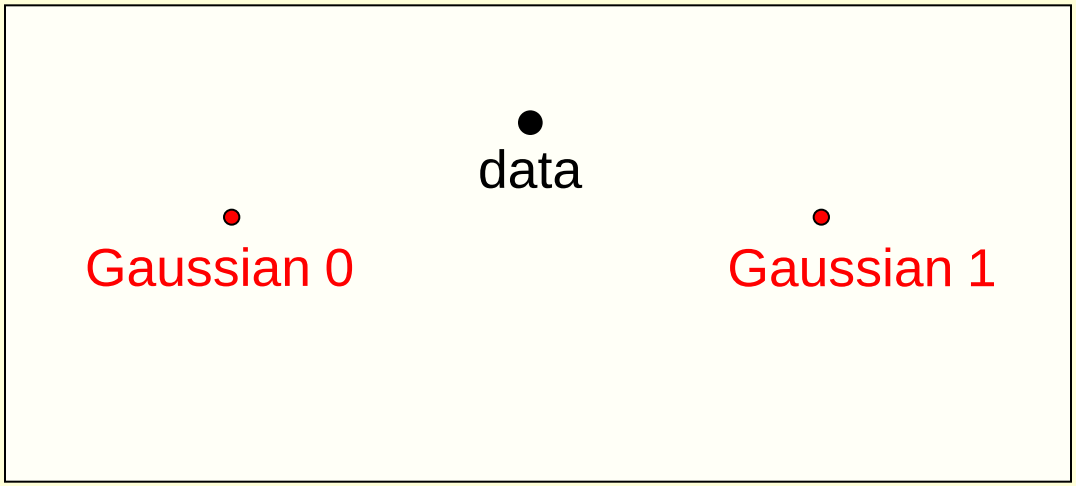
- It is obvious that this is minimized by setting the variance of the Gaussian to be the variance of the residuals.

Sending a value assuming a mixture of two equal Gaussians



- The point halfway between the two Gaussians should cost –log(p(x)) bits where p(x) is its density under the blue curve.
  - But in the MDL story the cost should be –log(p(x)) plus one bit to say which Gaussian we are using.
  - How can we make the MDL story give the right answer?

The bits-back argument

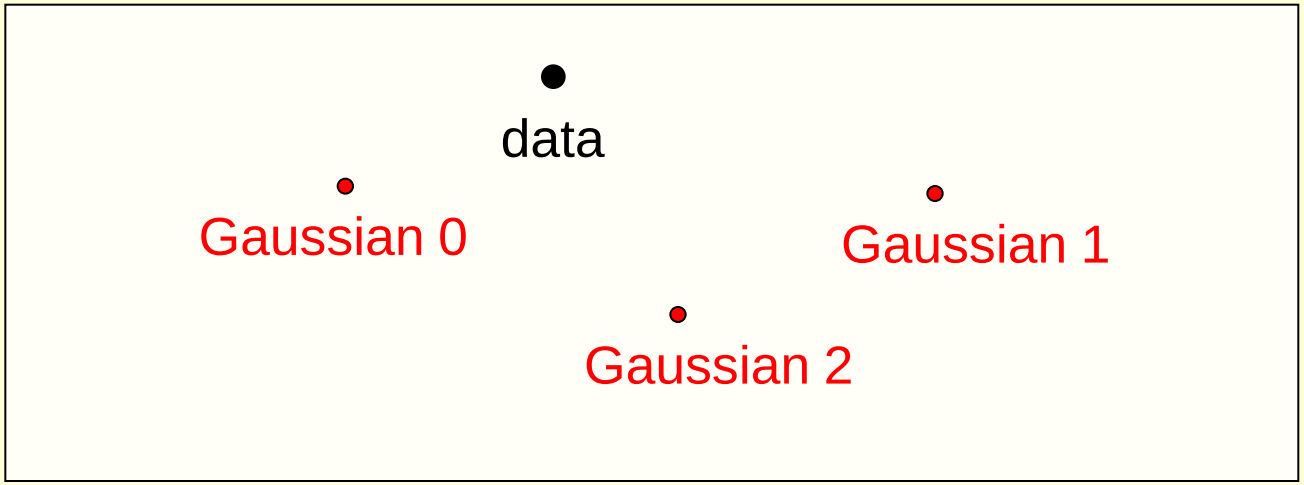


- Consider a datapoint that is equidistant from two cluster centers.
  - The sender could code it relative to cluster 0 or relative to cluster 1.
  - Either way, the sender has to send one bit to say which cluster is being used.
    - It seems like a waste to have to send a bit when you don't care which cluster you use.
    - It must be inefficient to have two different ways of encoding the same point.

Using another message to make random decisions

- Suppose the sender is also trying to communicate another message
  - The other message is completely independent.
  - It looks like a random bit stream.
- Whenever the sender has to choose between two equally good ways of encoding the data, he uses a bit from the other message to make the decision
- After the receiver has losslessly reconstructed the original data, the receiver can pretend to be the sender.
  - This enables the receiver to figure out the random bit in the other message.
- So the original message cost one bit less than we thought because we also communicated a bit from another message.

The general case



$$Expected\ \ Cost = \sum_i p_i E_i - \sum_i p_i \log \frac{1}{p_i}$$

Probability of picking cluster i

Bits required to send cluster identity plus data relative to cluster center

Random bits required to pick which cluster

What is the best distribution?

- The sender and receiver can use any distribution they like
  - But what distribution minimizes the expected message length
- The minimum occurs when we pick codes using a Boltzmann distribution:

$$p_i = \frac{e^{-E_i}}{\sum_j e^{-E_j}}$$

- This gives the best trade-off between entropy and expected energy.
  - It is how physics behaves when there is a system that has many alternative configurations each of which has a particular energy (at a temperature of 1).

Free Energy

$$\textit{Free Energy} = \sum_i p_i E_i - T \sum_i p_i \log \frac{1}{p_i}$$

Probability of finding system in configuration i

Energy of configuration i

Temperature

Entropy of distribution over configurations

The equilibrium free energy of a set of configurations is the energy that a single configuration would have to have to have as much probability as that entire set.

$$e^{-\frac{F}{T}} = \sum_i e^{-\frac{E_i}{T}}$$

A Canadian example

$$F_{ice} = < E_{ice} > - T \; H_{ice}$$

- Ice is a more regular and lower energy packing of water molecules than liquid water.
  - Lets assume all ice configurations have the same energy
- But there are vastly more configurations called water.

$$E_{ice} << E_{water}$$

$$H_{ice} << H_{water}$$

$$At \; T = 272, \quad F_{ice} << F_{water}$$

$$At \; T = 274, \quad F_{ice} >> F_{water}$$

Stochastic MDL using the wrong distribution over codes

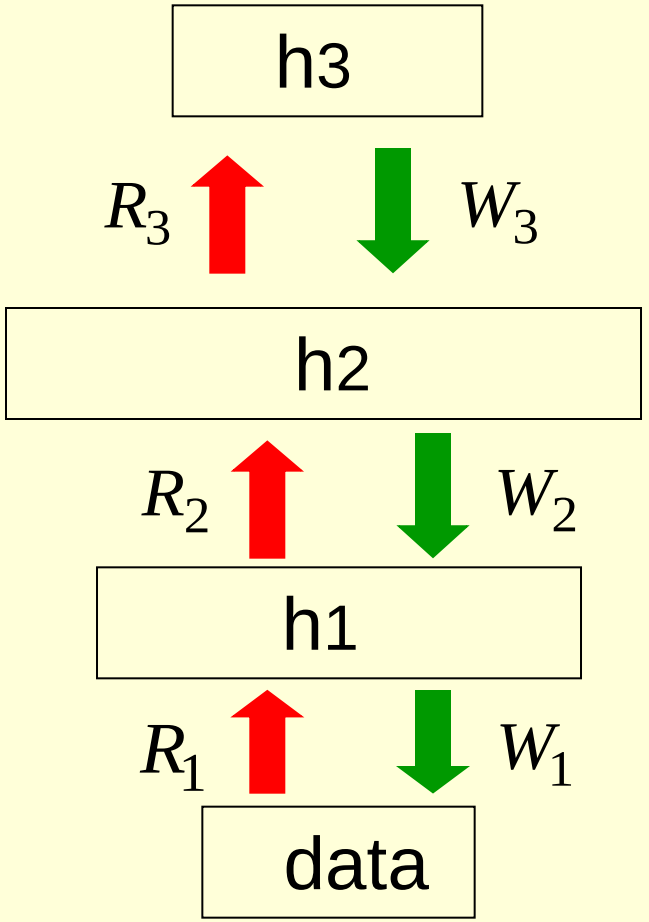
- If we want to communicate the code for a datavector, the most efficient method requires us to pick a code randomly from the posterior distribution over codes.
  - This is easy if there is only a small number of possible codes. It is also easy if the posterior distribution has a nice form (like a Gaussian or a factored distribution)
  - But what should we do if the posterior is intractable?
    - This is typical for non-linear distributed representations.
- We do not have to use the most efficient coding scheme!
  - If we use a suboptimal scheme we will get a bigger description length.
    - The bigger description length is a bound on the minimal description length.
    - Minimizing this bound is a sensible thing to do.
  - So replace the true posterior distribution by a simpler distribution.
    - This is typically a factored distribution.

Fitting a sigmoid belief net using variational inference

- For large nets, the true posterior over the hidden layers is intractable.
- The obvious variational approach is to find the best factorial approximation.
  - This requires an iterative inference process that adjusts the probability assigned to each hidden unit so as to minimize F (with the parameters held constant).
- Can we avoid this iterative inner loop?
  - What if we were willing to accept a non-optimal factorial approximation.

The wake-sleep algorithm for an SBN

- Wake phase: Use the recognition weights to perform a bottom-up pass.
  - Train the generative weights to reconstruct activities in each layer from the layer above.
- Sleep phase: Use the generative weights to generate samples from the model.
  - Train the recognition weights to reconstruct activities in each layer from the layer below.

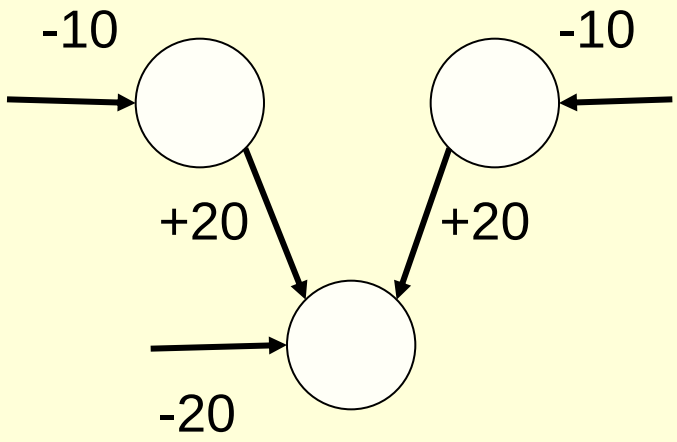


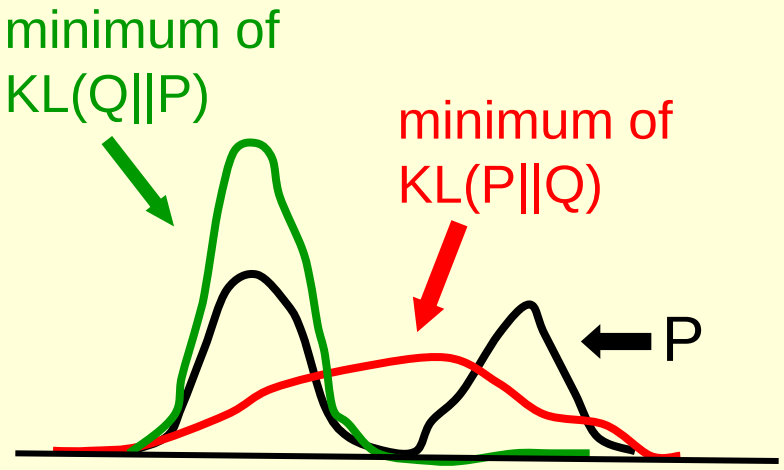
The flaws in the wake-sleep algorithm

- The recognition weights are initially trained to invert the generative model in parts of the space where there is no data.
  - This is wasteful.
- The recognition weights follow the gradient of the wrong divergence. They minimize KL(P||Q) but the variational bound requires minimization of KL(Q||P).
  - This leads to incorrect mode-averaging
- The true posterior over the top hidden layer is typically very far from independent.
  - So it is very badly modeled by a prior that assumes independence.

Mode averaging

- If we generate from the model, half the instances of a 1 at the data layer will be caused by a (1,0) at the hidden layer and half will be caused by a (0,1).
  - So the recognition weights will learn to produce (0.5,0.5)
  - This represents a distribution that puts half its mass on the very improbable hidden configurations (0,0) & (1,1)
- Its much better to just pick one mode and pay one bit for ignoring the other mode.



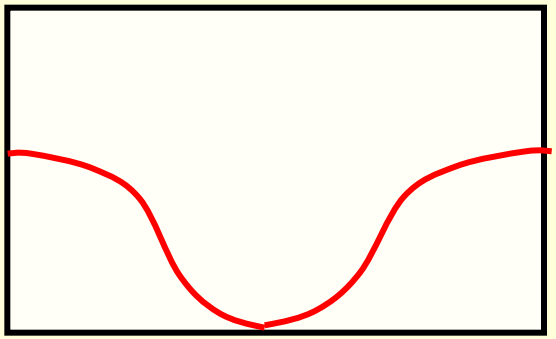


The origin of variational Bayes

- In variational Bayes, we approximate the true posterior across parameters by a much simpler, factorial distribution.
  - Since we are being Bayesian, we need a prior for this posterior distribution.
- When we use standard L2 weight decay we are implicitly assuming a Gaussian prior with zero mean.
  - Could we have a more interesting prior?

Types of weight penalty

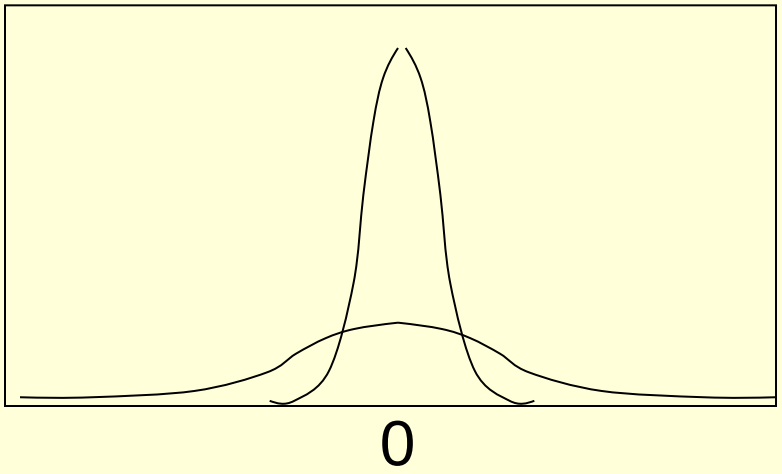
- Sometimes it works better to use a weight penalty that has negligible effect on large weights.



- 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

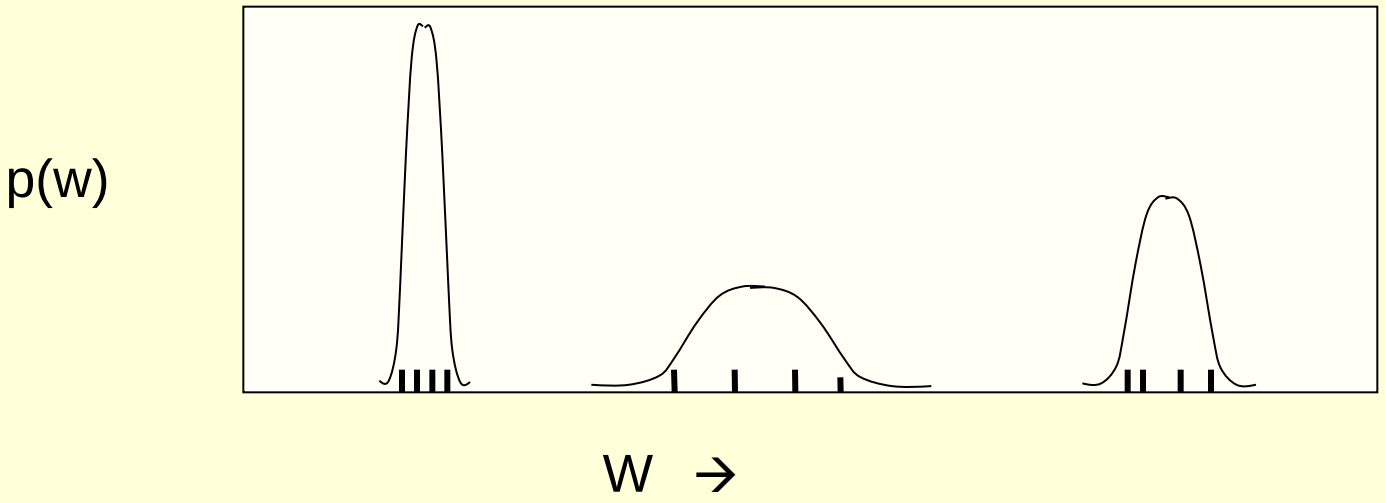


Soft weight-sharing

- 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 is called “empirical Bayes”.
- 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

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

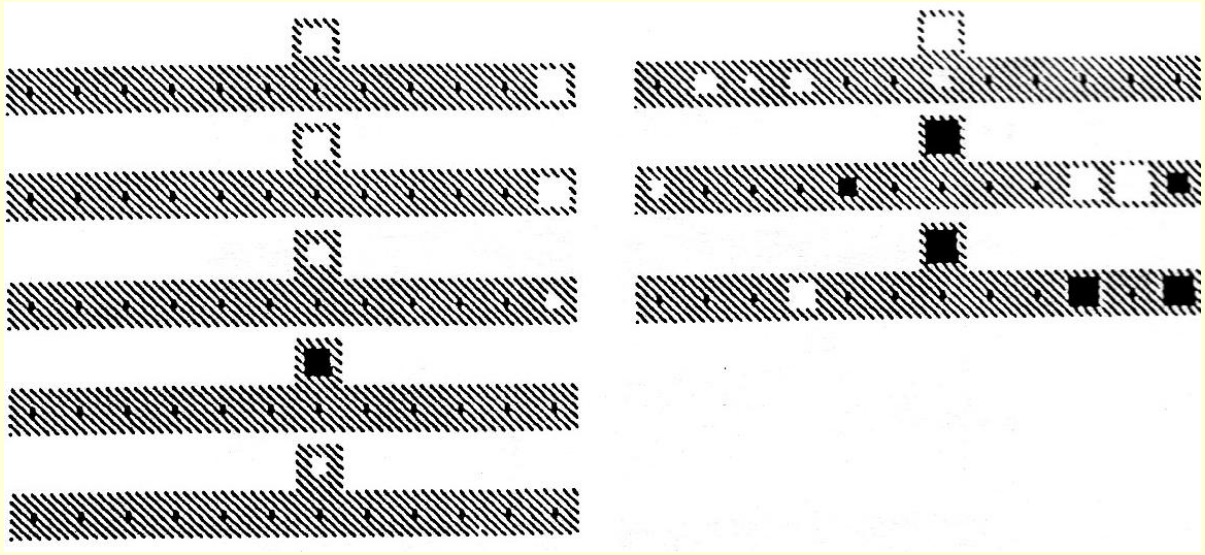
$$\frac{\partial C}{\partial w_i} = \frac{k}{\sigma_{out}^2} \sum_c (y_c - t_c) \frac{\partial y_c}{\partial w_i} \quad - \sum_j p(j \mid w_i) \frac{\mu_j - w_i}{\sigma_j^2}$$

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 has 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

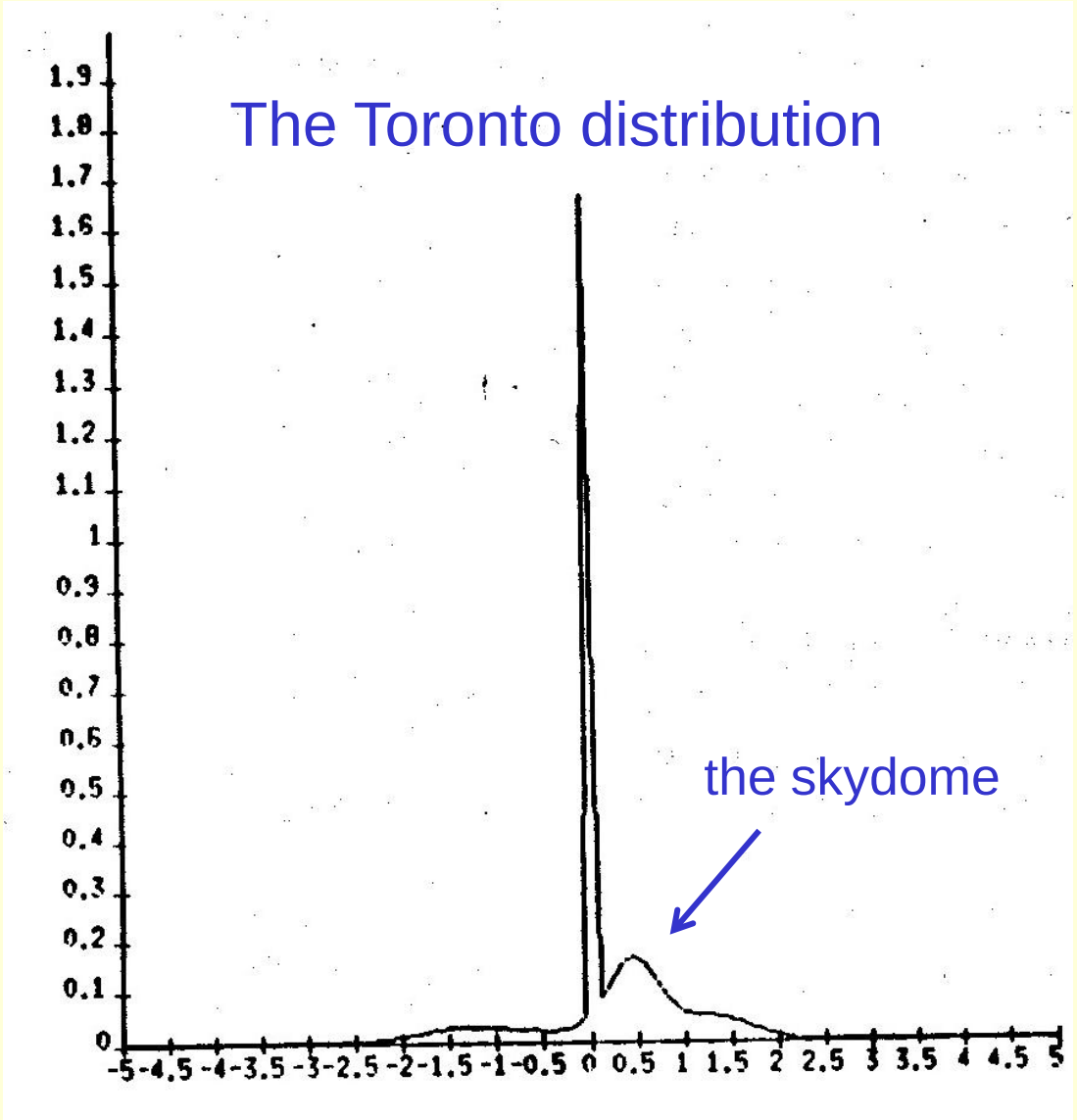


Rule 1: (uses 2 units): High if high last year.

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

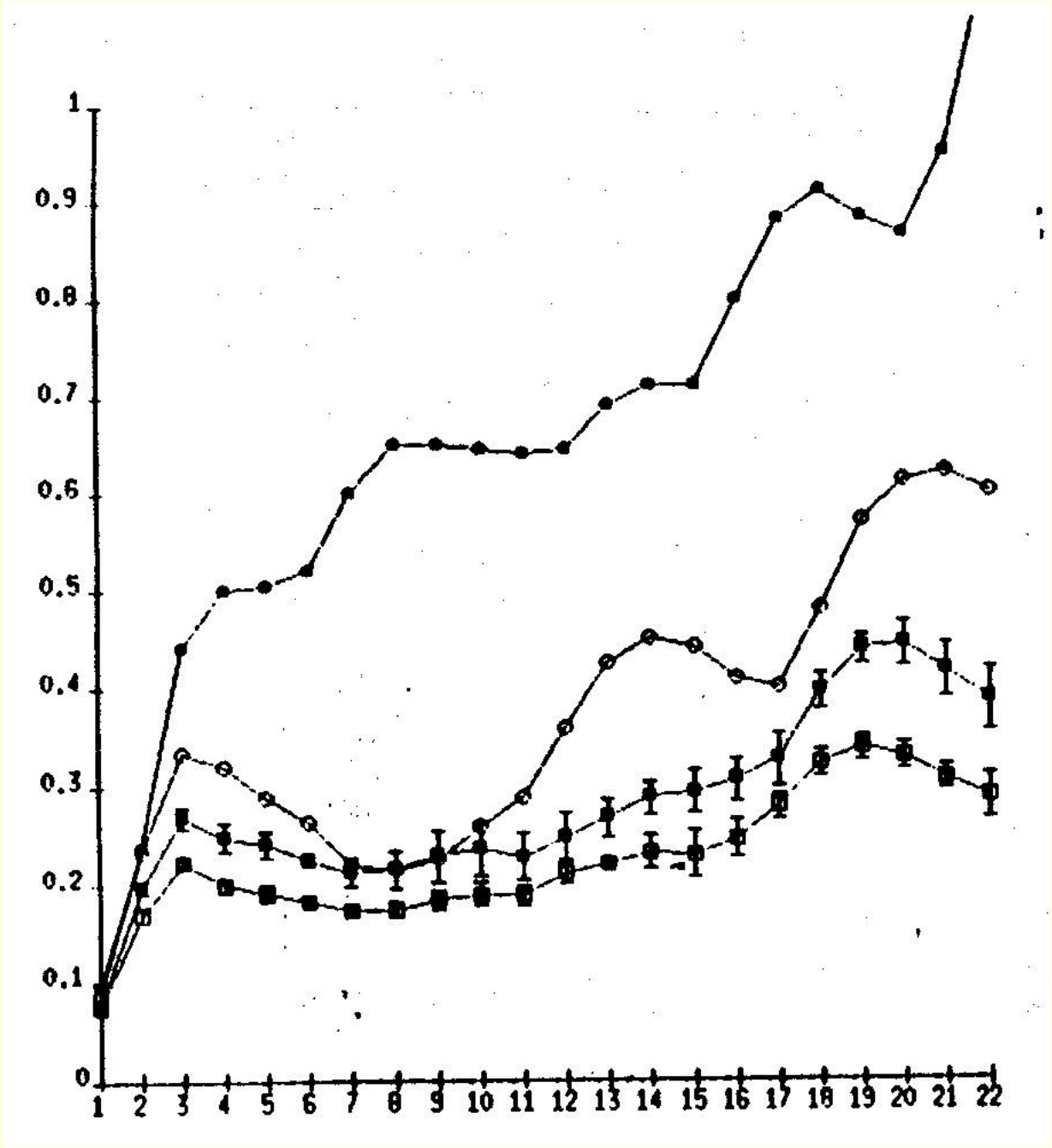
Rule 3: Low if low 1 or 8 ago & high 2 or 3 ago.

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

Stochastic output of neuron

mean of weight distribution

$$y = xw$$

$$< y > = x < w > = x\mu_w$$

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

- The noise variance for the weight gets multiplied by the squared input value and added to the squared error:

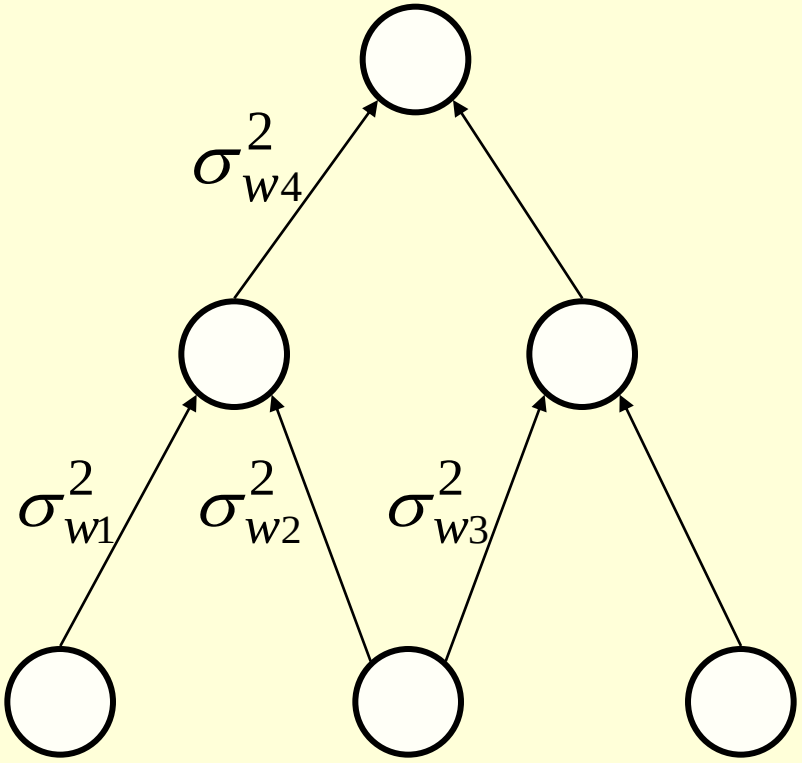
$$Error = (t - y)^2$$

$$< Error > = (t - x\mu_w)^2 + x^2\sigma_w^2$$

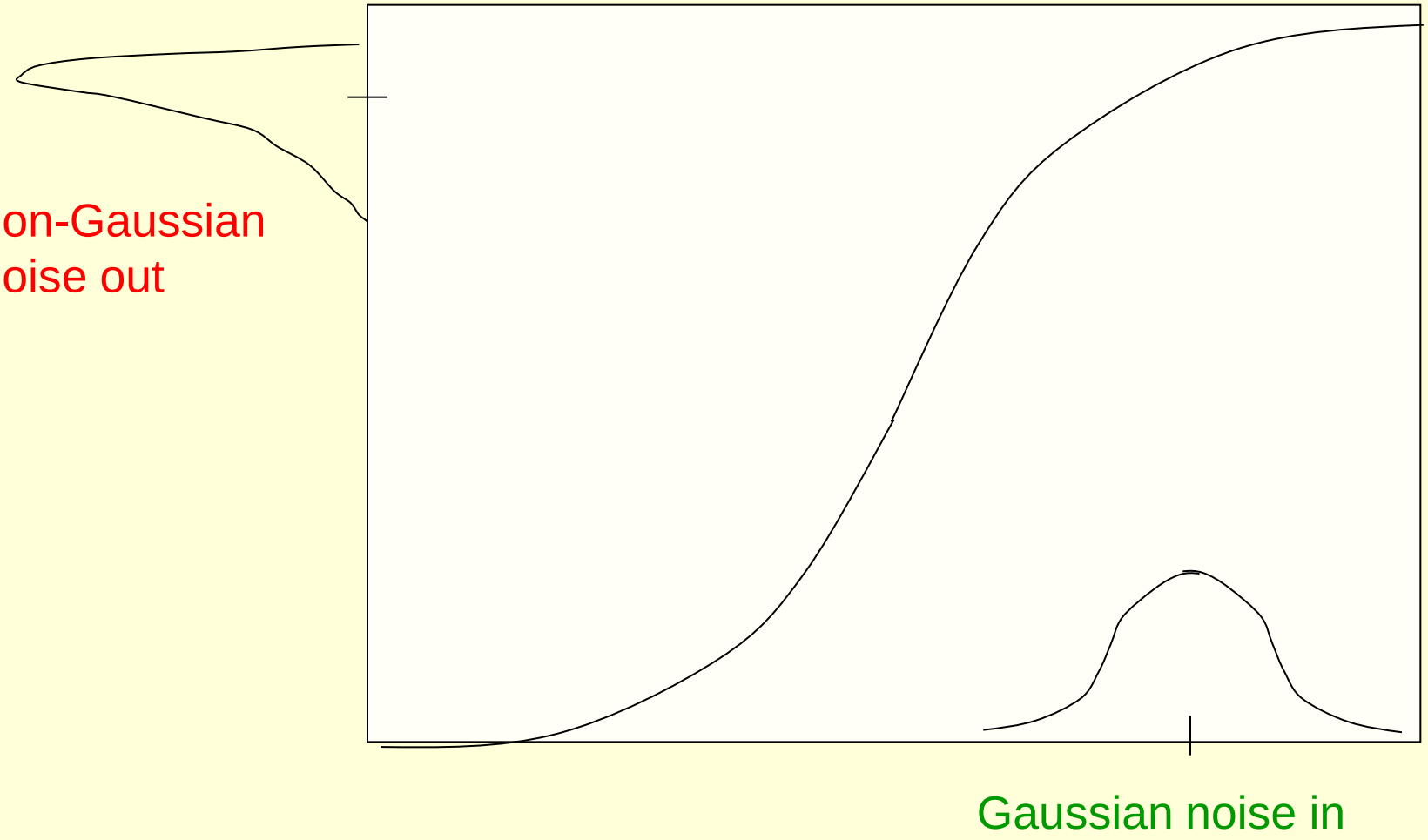
Extra squared error caused by noisy weight

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.

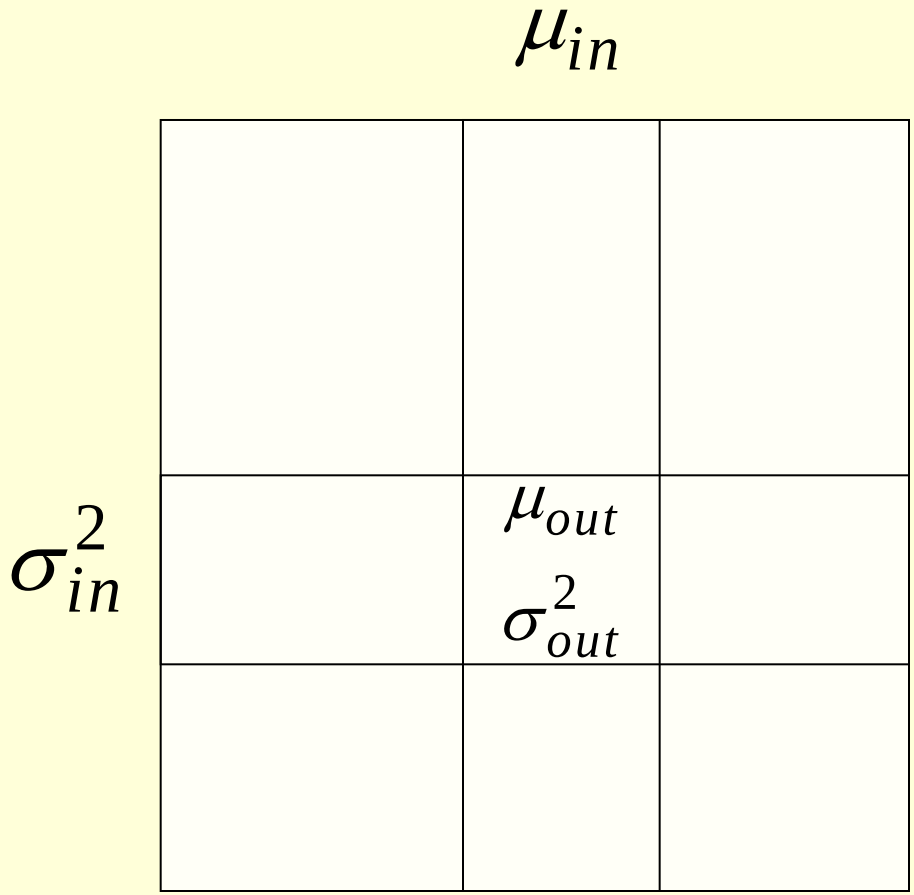


The mean and variance of the output of a logistic hidden unit



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.

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

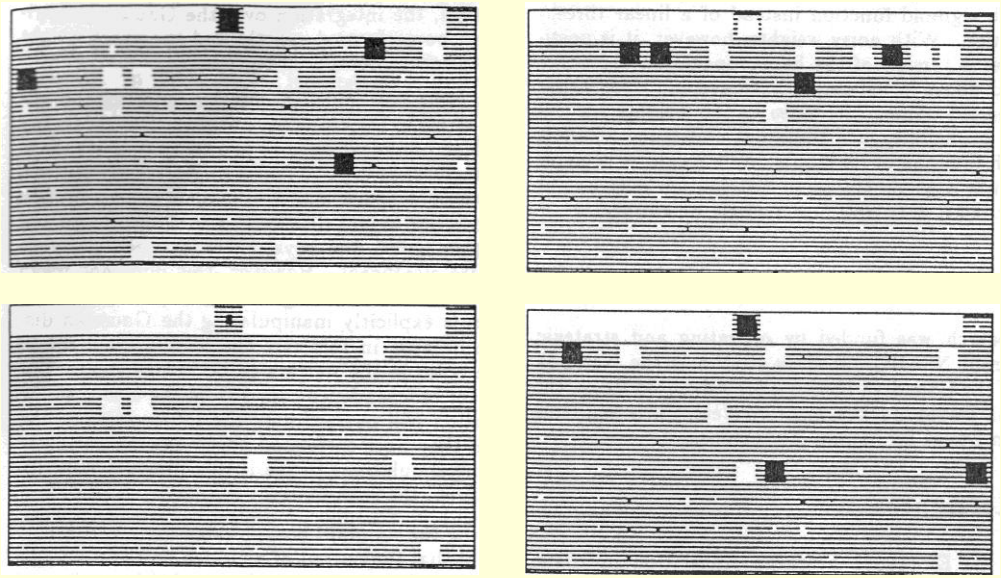
Empirical Bayes: Fitting the prior

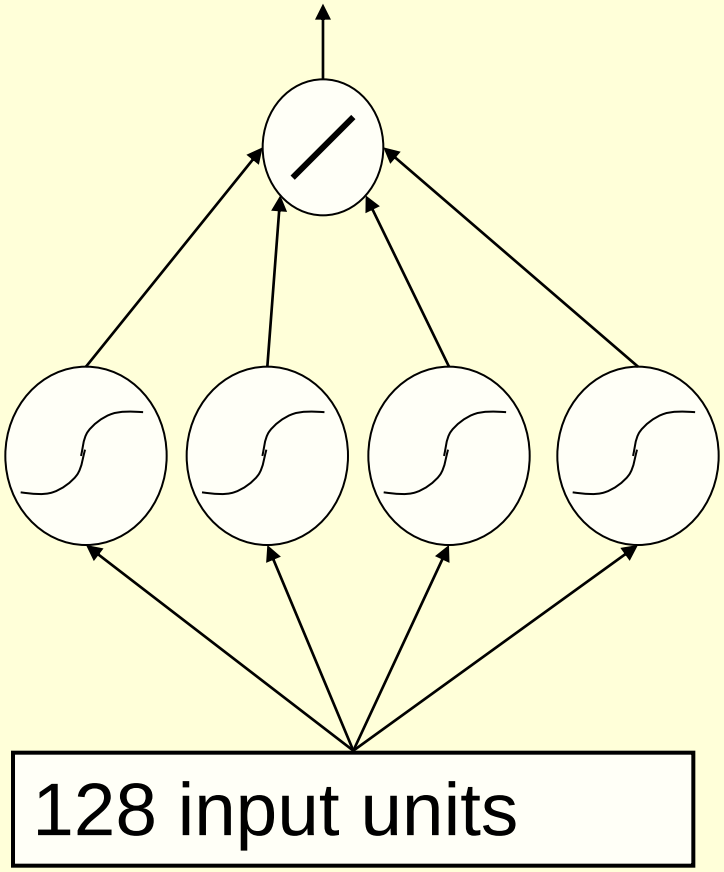
- We can now trade off the precision of the weights against the extra squared error caused by noisy weights.
  - even though the residuals are non-Gaussian we can choose to code them using a Gaussian.
- We can also learn the width of the prior Gaussian used for coding the weights.
- We can even have a mixture of Gaussians prior
  - 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

output weight

bias



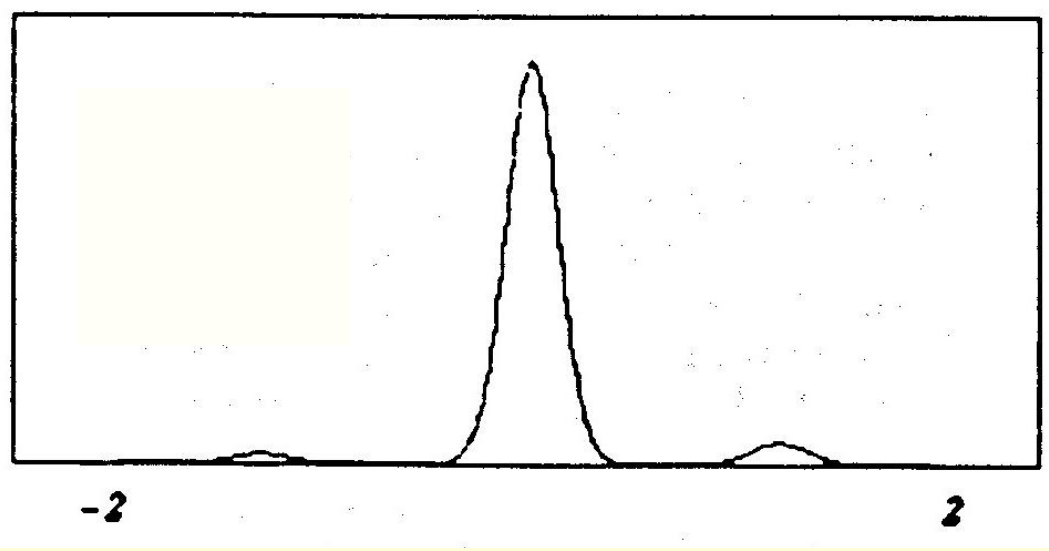


- It learns a few big positive weights, a few big negative weights, and lots of zeros. It has found four “rules” that work well.

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 with 14 parameters