**Prerequisites:**

<https://www.deeplearningbook.org/slides/18_partition.pdf>

Lower bounds as used in probability density functions:

* Lower bounds give the minimum value that a function can take on
* This may be useful because if you cannot estimate the true probability, you can at least estimate the bound and substitute it in the equation
* The tightness of the bound is useful since the more tight, the better the approximation
* You want to consider the purpose of upper bounds - minimizing an upper bound for example may be useful since you will be tightening the available values the model can take on and narrowing it down to the gith values, making it a better predictor with lower deviation
* Maximizing an upper bound however widens the available values of pmodel(x) and allows them to be further away from pdata(x)

We normalize p by dividing by a partition function Z to obtain a valid distribution:where 𝚹 presumably provides computations for the clique functions

The partition function is an integral over all continuous states and is usually intractbale



Some models avoid using p~ and other models deal with tractable partition functions, but the focus will be on intractable partition functions

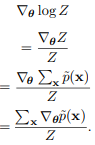
**Log-Likelihood Gradient**

When learning undirected models by maximum likelihood, the partition function depends on the parameters — the gradient of the log likelihood has a term corresponding to the gradient of the partition function

Since p = p~/z

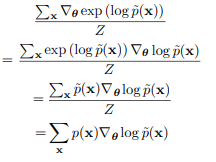
This is a decomposition into the **positive phase** and **negative phase** of learning

Looking at the gradient of log Z (1 -> 2 is possible since its treated of derivative of ln x)



The sum of the gradients of unnormalized probabilities w.r.t. 𝚹 divided by the partition

If p(x) > 0 for all x, we can substitute exp(log(p~(x)) for p(x)



By the chain rule, it simplifies to the sum of (the normalized probability of x multilied by the gradient of the log unnormalized probability of x w.r.t 𝚹)

This is the same as:



This proof is similar for integrals, where we use Leibniz's rule of integration

However this has some extra constraints such as the gradient must exist for all 𝚹 and **x** and there must be an integrable function which bounds the gradient of p~

This identity is the basis for Monte Carlo methods which approximately maximize likelihood of models with intractable partition functoins:



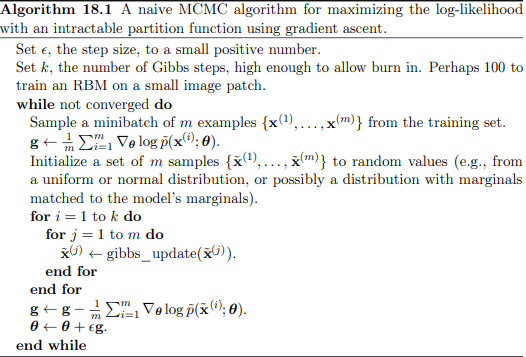
The Monte Carlo approach for undirected models provides a framework where we can think of both the positive and negative phase, where in the positive phase we increase for x drawn from the data, and in the negative phase we decrease Z by decreasing drawn from the model distribution

We can parametrize  in terms of an energy function, where the positive phase pushes down energy of training examples and the negative phase increases the energy of samples from the model

**Main idea - we can break up the learning of undirected models by max likelihood into two phases, positive and negative, where the negative phase consists of minimizing Z by decreasing the log likelihood of our model w.r.t 𝚹 while maximizing the true p log likelihood**

**Stochastic Maximum Likelihood and Contrastive Divergence**

We can implement the loss naively by burning in Markov chains from a random initialized 𝚹 when the gradient is needed, but this is infeasible since new chains would be burned every step



The initial gradient will be the gradient of the log likelihood w.r.t 𝚹 drawn from the true distribution/ training examples

Randomly initialize a set of m samples

Then, for each step until the Markov chain burns in, apply a Gibbs update to each variable

When the Markov chain burns in, calculate the gradient of log likelihood of the new states w.r.t 𝚹

Subtract this new gradient from the gradient of the training samples, and add the result to 𝚹 to ascend

This MCMC approach can be interpreted as max likelihood trying to achieve a balance between two forces — one pushing up on the model distribution where data occurs (log P, positive phase) and another pushing down on the model distribution where model samples occur (log Z)

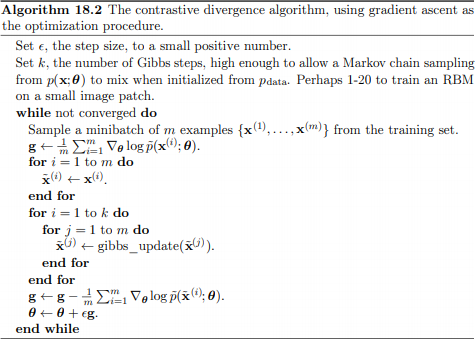
Approximations to the negative phase are computationally cheaper and can be interpreted as pushing down on the wrong/unideal locations

Since the negative phase reduces the probability of the model's points, it can be thought of as representing the model's incorrect beliefs about the world

Instead of burning in the chain from scratch every step, we can initialize the Markov chain to something close to the model distribution so that burning it in won't be hard

In **contrastive divergence**, the Markov chain is initialized from samples of p, already in data

Initially, the data distribution is far from the model distribution so the negative phase is inaccurate

However, over time, the positive phase will increase pmodel(data) which will make the model distribution closer to the data distribution making the negative phase more accurate

Basically, we have a model distribution with some parameters which we want to train

* We get the model distribution results by making the initialization close to the true data distribution and then running Gibbs sampling (using 𝚹 to burn in a fashion similar to the model distribution)
* We then use contrastive divergence

CD has been shown to be biased since it converges to a different point than the ML estimator

* It can be used for initialization and then fine-tuned better Monte Carlo methods
* The bias is a result of CD discarding the smallest terms of the MCMC update gradient

CD can be used for RBMs which can be stacked to create DBMS (deep) but CD cannot create deep models directly

* It is hard to obtain samples of the hidden units given samples of visible units since the hidden units are not in the data
* Initializing **v** from data, we need to burn in the distribution over h conditioned on v

CD can be thought of as punishing the model for having a Markov chain (remember it is intractable to directly draw from the model) which rapidly changes input from data

CD is good for pretraining shallow layers which will be stacked since early layers store latent variables which can then be fed into later layers

A different strategy, known as **stochastic maximum likelihood** or **persistent contrastive divergence**, initializes Markov chains with their states from the previous gradient step (after burning in)

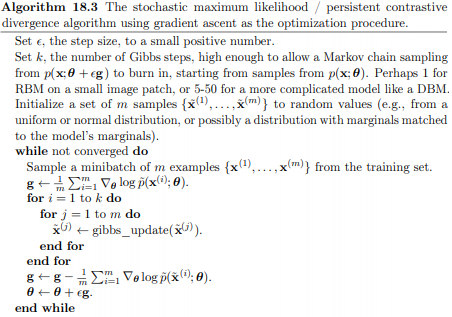
The idea of this algorithm is that if the step size is small, then the model distribution from one timestep before will be close to the model distribution now, so states from one timestep before will be easy to mix — similar to how initializing from data would be easy to mix

Since each Markov chain is constantly updated rather than restarted, the chain as a whole can wander over time and find all of the model's modes (to get a more accurate picture)

SML also provides an initialization for both v and h, while CD requires burning in for deep models which make use of latent variables

SML has been shown to be better than CD on RBMs as well as extracting features through hidden units on the RBMs

If k is too small or 𝟄 is too large, then the gradient updates might be too fast for the Markov chains to properly mix

If 𝟄 is too large, then this will be reflected in the high variance of gradient steps during the negative phase compared to the Markov chains

The only time we initialize the variables ourselves (without any stochasticity from the Markov chain) is when we start with the distribution matching the model's marginal

Remember, we still sample from the original distribution for the positive phase

When evaluating the samples from a model trained with SML, the samples must be drawn from a fresh Markov chain initialized from a random starting point after training

The samples from the negative chain at the end of training are influenced by previous versions of the model and make the model look like it has more capacity than it actually does

The variance of contrastive divergence is less than the variance of SML because CD uses the same points in both the positive and negative phase

If the negative phase is initialized from different training points than the positive phase (as in SML), then the variance rises above that of the estimator based on exact sampling

One approach relies on changing the parametrization of the model and cost function instead of the Monte Carlo approach

This is known as fast PCD, where 𝚹 is replaced with

There are twice as many parameters which are added together to provide the original model definition

The fast copy is trained with a much higher learning rate which causes it to adapt rapidly to the negative phase and push the Markov chain to new territories as it rapidly mixes (that is, when the fast weights are free to change)

Weight decay is eventually applied to the fast weights after different modes have been explored

**Main idea - using Monte Carlo methods, we can estimate the gradient of log Z, letting us decompose the problem into a function of log p and log Z. We use MCMC to estimate gradient(log Z), and any method to estimate gradient(log p)**

**Pseudolikelihood**

While MCMC directly approximate Z and the gradient of Z, other models don't compute Z at all

This is based on the assumption that the ratios are easy to calculate in undirected probabilistic models



The pseudolikelihood is based on the observation that conditional likelihoods take this ratio based form and can therefore be computed w/o knowledge of Z

Suppose we partition x into three variables **a, b, c** where **a** is the variables we want the conditional distribution over, conditioned on **b**, while **c** is ignored/not queried



This requires marginalizing out (summing over the joint distribution) **a** which is easy if a and c do not contain many variables

If there are n variables total, however, we must marginalize out a set of n - 1

The chain rule of probability:



We have made **a** maximally small (corresponding to log p(x1)) but **c** can be as large as **x**2:n

If we move **c** into **b**, this yields the **pseudolikelihood** objective function which predicts feature xi based on all other features x-i



If each random variable has k discrete variables, this requires only k x n evaluations of p~ to compute as opposed to kn evaluations to compute the partition function

Estimation by maximizing the pseudolikelihood is asymptotically consistent

The **generalized pseudolikelihood** estimator uses m different sets  of indices of variables that appear on the left side of the conditioning bar

The generalized pseudolikelihood objective function is given by:



We use m different sets of indices of variables that appear together on the left side of the conditioning bar - i ranges from 1 to m

If m = 1 and S1 = 1, … n the generalized pseudolikelihood recovers the log likelihood

If m = n (maximum number of sets), the generalized pseudolikelihood recovers the pseudolikelihood

The pseudolikelihood is not very useful for tasks like density estimation where we require the full joint distribution over p(x)

Pseudolikelihood is good for conditional distribution tasks such as filling in missing data, especially if the index sets given by Si are made to only include variables w/ strong correlations

Pseudolikelihood cannot be used with other approximations which provide a lower bound on p because p~ appears in the denominator which provides an upper bound on the whole expression, and there is no benefit to maxmizing an upper bound

Pseudolikelihood is useful for deep models using approximate inference methods

Since pseudolikelihood computes all the conditionals, it has a much greater cost than SML

However, random sampling of one conditional or generalized likelihood can bring it down to the same cost of SML

The denominators of each conditional distribution cause the learning algorithm suppressing the probability of all states that have only one variable differing from a training example

In this sense, it resembles the negative phase

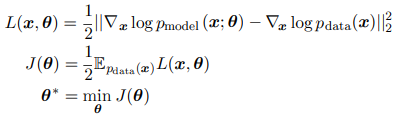
**Main idea - pseudolikelihood is a way to calculate the probability of a state without using the partition function Z — it instead uses conditional distributions being expressed as ratios to calculate which result in more expensive approximations**

**Score Matching and Ratio Matching**

Score matching consistently trains a model without estimating Z

Log density with respect to its argument are called the 

Minimize difference between derivatives of model's log density and derivative of data's log density with respect to the input



Computing the score of the data distribution requires knowing pdata which of course is not possible

However, minimizing the expected value of L(x, 𝚹) is equivalent to minimizing the expectation of

 where n is the dimensionality of **x**

Since score matching requires differentiating inputs, it is not applicable to models of discrete data; however, the latent variables in this model (generating x, not the actual data themselves) may be discrete

Since score matching requires derivatives of p~ it is not compatible with methods only providing a lower bound on p~(x)

This means score matching cannot be applied to estimating models with complicated interactions between hidden units, such as sparse coding models

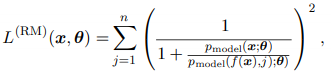
Since deeper hidden layers of deep models contain discrete variables, it is not used for pretraining those (only the first hidden layers)

Score matching can be viewed as contrastive divergence using a specific version of a Markov chain, making local moves guided by the gradient

Generalized score matching in the discrete case does not work in high dimensional discrete cases where observed probability of many events is 0

Ratio matching applies to discrete — binary — data

It consists of minimizing the average over examples of the objective function:

where f(**x**, j) returns x with bit at position j flipped

The partition function cancels out in a ratio of two probabilities, just like pseudolikelihood

**Main idea - score matching attempts to learn 𝚹 by minimizing the difference between derivatives of model's log density w.r.t input and derivatives of the data's log data w.r.t input**

**Denoising Score Matching**

We want to fit a distribution

rather than the true pdata

q(x | y) is a corruption process which forms **x** by adding noise to **y**

This represents the true training process better since we usually only have access to empirical samples

Any consistent estimator will make pmodel into a set of Dirac distributions centered on train data

Smoothing by q does not let us keep asymptotic consistency (n —> ∞) , but prevents the issue of Dirac distributions

q is typically normally distributed noise

Therefore, autoencoders basically perform denoising score matching and can be used to escape the partition function

**Main idea - denoising score matching smooths out the distribution so that pmodel is not a Dirac distribution, which may be good since we only have access to empirical data**

**Noise-Contrastive Estimation**

NCE actually estimates the partition function, rather than avoiding it or computing the gradient



Two parameters - 𝚹 and **c** where c approximates -log Z(𝚹)

They are both learnable and use the same algorithm

log pmodel(x) becomes a more valid probability distribution as the estimate of c improves

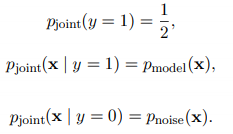
Since max likelihood does not work directly due to no upper bound on c, NCE reduces the unsupervised problem of estimating p(x) to that of learning a probabilistic binary classifier where one category corresponds to data generated by the model

Using max likelihood now, the estimator is asymptotically consistent

We introduce a second distribution, the **noise distribution** pnoise(x)

We introduce a tractable and easy to sample from noise distribution, where we construct a model over both **x** and the new binary class variable **y**

We now have a joint distribution with the following properties



y determines when we generate from the model distribution or the noise distribution

We can create a similar model for training data **ptrain**, determining whether we draw **x** from **data** or the noise distribution

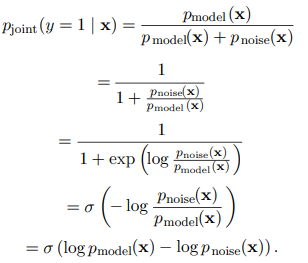
The supervised learning problem is fitting pjoint to ptrain



We draw **x** and **y** from the joint distribution, and we use 𝚹 and **c** to maximize the probability of predicting the correct **y** in the joint model given **x** which was drawn from the training or noise distribution based on the value y

Let's suppose y = 0, so we get a fake sample **x**; then we want log pjoint(y | x) to predict 0 which means if our initial guess was right, then our joint model used the noise distribution

It simplifies to a logistic regression model applied to the difference in log probabilities of the model and noise distribution



NCE works best when there are few variables, with high dimensionality

NCE does not work well when there are multiple variables since the logistic classifier rejects a noise sample by classifying any 1 variable with an unlikely value, rather than considering all

When pnoise is too simple, this can make it too obviously distinct from the data for pmodel to improve noticeably

NCE does not work if *only* a lower bound estimate is available

A lower bound estimate on pnoise for example is not useful since it provides only an upper bound on pjoint(y = 1 | x) which only appears in half the terms

In other words, a lower bound estimate on pnoise(x) is useless because it provides an upper bound on pjoint(y = 1 | x) and there is no point in maimizing the upper bound

Why would you need both a lower bound and upper bound?

When the model distribution is copied to define the new noise distribution before each gradient step, NCE defines a procedure called **self-contrastive estimation** where the expected gradient is equivalent to that of max likelihood estimator

The case where the noise is generated by the model suggests that max likelihood is a procedure which forces a model to distinguish its own beliefs from reality, while general NCE achieves a lower computational cost by forcing the model to distinguish reality from a fixed baseline of an arbitrary constant noise distribution

NCE is based on the idea that a good generative model should be able to distinguish data from noise, and also generate noise which cannot be distinguished from data

**Estimating the Partition Function**

Estimating the partition function may be required when evaluating the model

If the likelihood of an i.i.d test dataset is higher, than the model is better in the sense it has a higher test log likelihood



We must evaluate the log probability the model assigns at each point, which unfortunately requires the knowledge of the partition function

We can simplify the situation by rearranging it so that we only need to know the ratio of the partition functions (not the partitionns themselves)

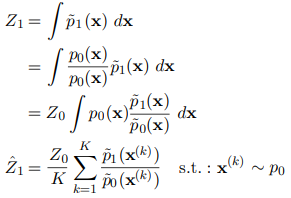


We can estimate this ratio using importance sampling provided they are similar

If we knew just 1 ratio and partition function value, we could compute the value of the other



Below we use importance sampling from tractable distributions p0~ and a tractable partition function Z0

<= This makes a Monte Carlo estimator using samples drawn from p0(**x**) and weights each sample with the ratio of the unnormalized p1 and proposal p0

Notice how p0(x) is the "given" for p1(x) and p0~(x), becoming part of their probabilities (it is now **xk**) between the last 2 steps

The ratio between the partition functions can be estimated as 

p0 must be close to p1 in order to be a high quality approximation

If p0 and p1 are not close, p0 will have low probability under p1 and make negligible contributions to the sum

* Z0/p0 is the weight by which p1~ is scaled, and if this is a small weight that is what causes negligible contributions

With only a few samples with significant weights, the estimator will be poor due to high variance



If p1 is complex, multidimensional, and/or multimodal making it harder to estimate, we introduce annealed importance sampling or bridge sampling to bridge the intermediate gap between p0 and p1

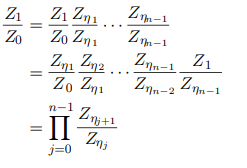
Annealed importance sampling tries to fix the gap by introducing intermediate distributions

Consider a sequence of distributions pn0 … pnn with 0 < n < 1 so that the first and last distributions in the sequence are p0 and p1

This will allow us to estimate the partition function over a complex space

Begin with a simpler model (e.g an RBM w/ learned weights) with a known partition function and estimate the ratio between the two models partition functions , based on the estimate of the ratios of a sequence of many simpler and similar distributions to the RBM

This ratio can be written as

  
We can reliably estimate each of the factors and use them to estimate Z1/Z0

The intermediate distributions are chosen by the model creator, with a popular choice being:



To sample from the intermediate d istributions, define a series of Markov chain transitions that define the conditional probability distribution of transitioning to x' given x

The transition operator T leaves pnj invariant

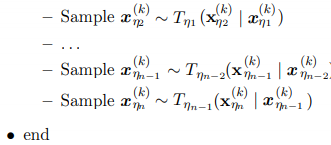


Integral over probability of each state \* probability of getting to x from that state

Obviously we do not know the Markov chains, so we can use any method to approximate them such as Monte Carlo, or Gibbs' including multiple passes through all random variables

Use transition operators to sequentially generate samples from the intermediate distribution until we arrive at samples from p1





~ T (distributed from T) means that samples are drawn when transition operator approximates sampling from the next intermediate distribution

For sample k, derive the importance weight by chaining together the importance weights from the intermediate distributions

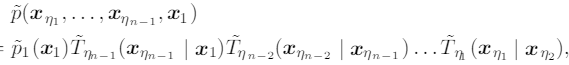
Note how x is drawn from various states of the Markov chain



Estimate of ratio of partition functions:

with k taking on different states

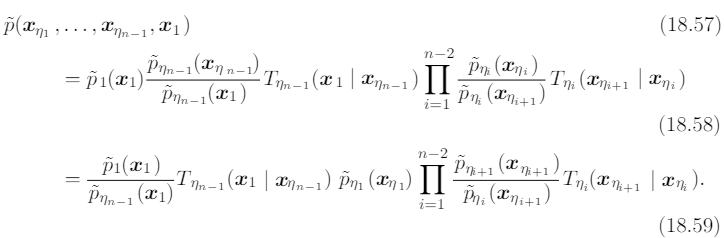
To prove this is valid annealed importance sampling, define a distribution over the extended space



where T~ is the reverse of the transition operator, obtained through Bayes' rule



Substituting this into the above equation



We can now generate samples from the joint proposal distribution q over the extended sample with the joint distribution given by

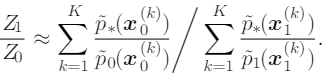


Taking q as the proposal distribution on the extended space state, the importance weights w are the same as proposed for AIS, so AIS can be interpreted as simple importance sampling applied to an extended state space

**Main idea - using importance sampling, we can make an unbiased estimate of the partition function Z1 given a different distribution Z0 and p0. However, in order to have a low variance, these two distributions must be similar which is usually not the case, so we introduce intermediate distributions (AIS) to bridge the gap and give better estimates.**

Bridge sampling relies on a single distribution p\* to interpolate between p0 and the distribution p1 which we are trying to estimate the partition function for

Bridge sampling estimates the ratio as:



If the bridge distribution is chosen carefully to have overlap with both p0 and p1 then it can deal with distributions with high distance/KL divergence values (compared to IS)

The optimal bridge distribution is given by



While this may seem circular since it requires Z1, we can start with a rough estimate of r and use the resulting bridge distribution to iteratively refine our estimate

Linked importance sampling uses bridge sampling to bridge the intermediate distributions used in AIS

AIS is too computationally intensive to use during training

Using AIS, parallel tempering, and bridge sampling, it is possible to get a low variance estimate of the partition function at every training iteration for an RBM