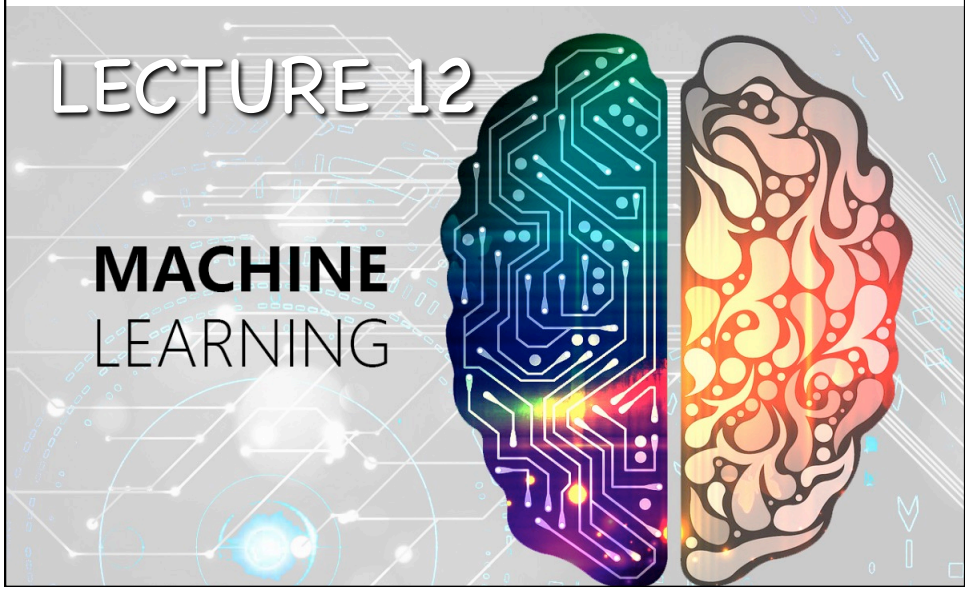
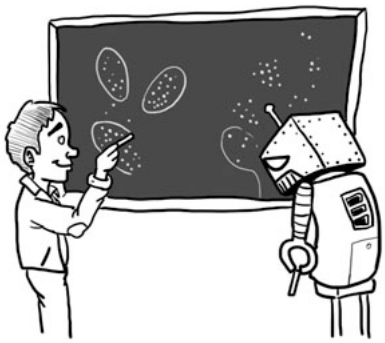
Numerical Simulation Laboratory



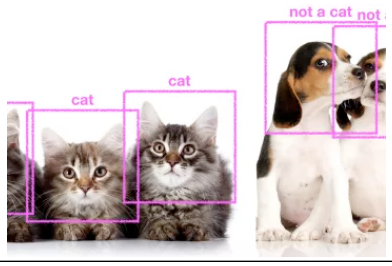
Outline

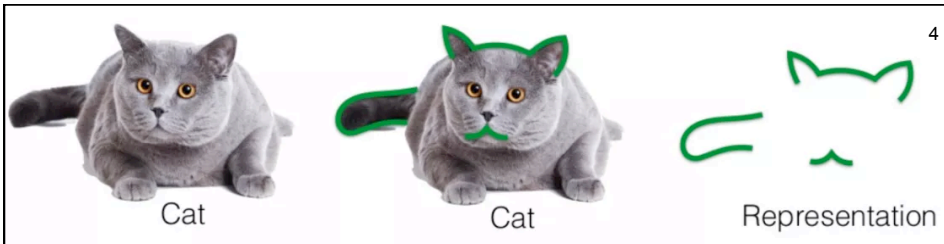
- Introduction to Convolutional Neural Networks
- Introduction to Restricted Boltzmann Machines for unsupervised learning and generative modeling
- Bayesian inference and estimators
- Training RBMs



Locality and symmetries in the dataset

- One of the core lessons of physics is that we should exploit symmetries and invariances when analyzing physical systems.
- Properties such as locality and translational invariance are often built directly into the physical laws. For example, we know that in many cases it is sufficient to consider only local couplings in our Hamiltonians, or that a model, in which local couplings depend only on the distances among the degrees of freedom, is translationally invariant and thus the total momentum is conserved, etc.
- Like physical systems, many datasets (especially datasets from Physics) and supervised learning tasks also possess additional symmetries and structure.
- For instance, consider a supervised learning task where we want to label images from some dataset as being pictures of cats or not. Our statistical procedure must first learn features associated with cats.

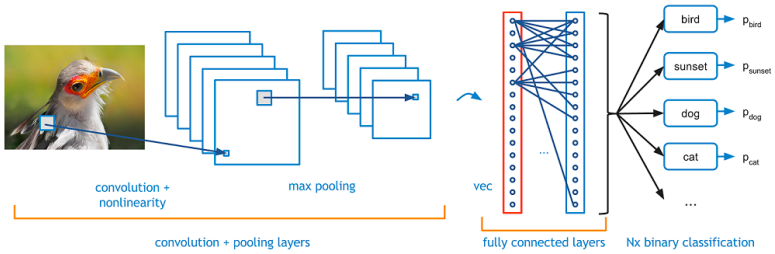




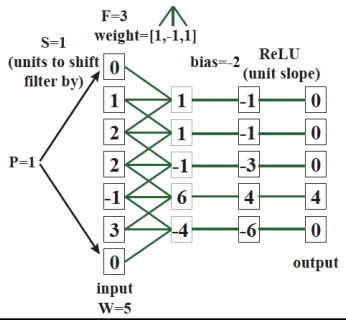
- Because a cat is a physical object, we know that these features are likely to be local (whiskers, tails, eyes, etc).
- We also know that the cat can be anywhere in the image. Thus, it does not really matter where in the picture these features occur (though relative positions of features likely do matter). This is a manifestation of translational invariance that is built into our supervised learning task.
- This example makes clear that, like many physical systems, many ML tasks (especially in the context of image processing) also possess additional structure, such as locality and translation invariance.
- The all-to-all coupled neural networks in the previous section typically fail to exploit this additional structure

Convolutional Neural Networks (CNNs)

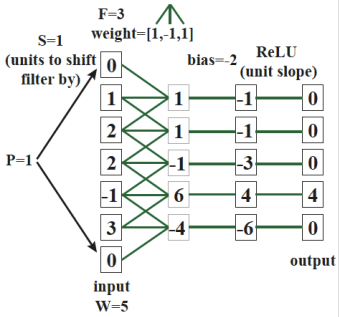
- Not surprisingly, the neural networks community realized these problems and designed a class of neural network architectures, convolutional neural networks or CNNs, that take advantage of this additional structure (locality and translational invariance)
- A convolutional neural network is a "translationally invariant" neural network that respects locality of the input data.
- There are two kinds of basic layers that make up a CNN: a convolution layer that computes the convolution of the input with a series of filters, and pooling layers that coarse-grain the input while maintaining locality and spatial structure



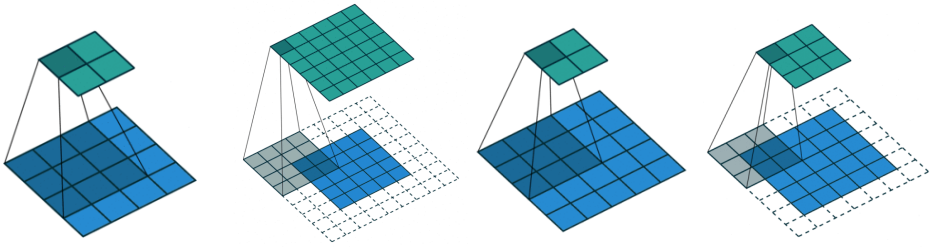
- For 2D data, a layer l is characterized by three numbers: height H_l, width W_l, and the "number of channels" D_l
- The height and width correspond to the sizes of the two-dimensional spatial (W_l;H_l)-plane (in neurons) (e.g. if l is the input layer, W_lxH_l is the number of pixels of the image), and the depth D_l to the "number of channels" in that layer (e.g. 3 channels for an RGB image).
- In general, we will be concerned with local spatial filters (often called a receptive field in analogy with neuroscience) that take as inputs a small spatial patch of the previous layer at all depths. The convolution consists of running this filter over all locations in the spatial plane
- To demonstrate how this works in practice, let us a consider the simple example consisting of a one-dimensional input of depth 1 (1 channel), shown in figure
- In this case, a filter of size Fx1x1 can be specified by a vector of weights w of length F



- The stride, S, encodes by how many neurons we translate the filter by when performing the convolution.
- In addition, it is common to pad the input with P zeros. For an input of width W, the number of neurons (outputs) in the layer is given by (W-F+2P)/S+1.
- Below I show you some convolution procedures, for a square input of unit depth:

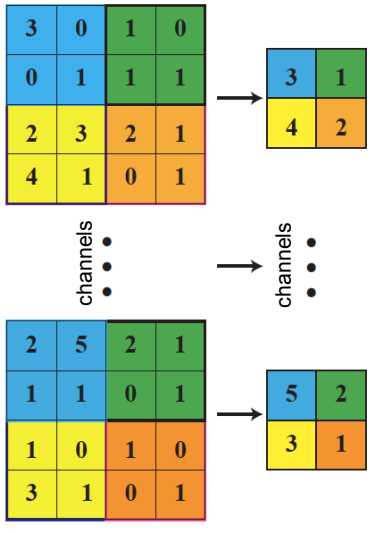


N.B.: Blue maps are inputs, and cyan maps are outputs.



No padding, no strides | Arbitrary padding, no strides | No padding, strides | Padding, strides

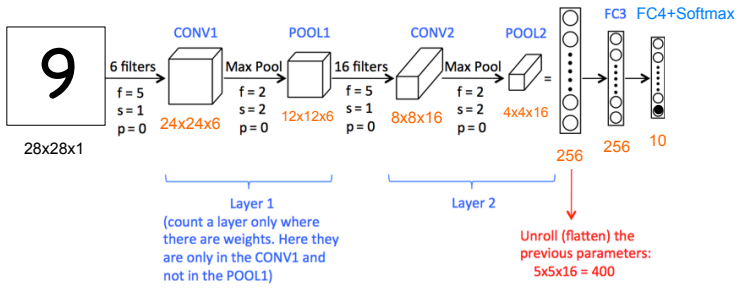
- These convolutional layers are interspersed with pooling layers that coarse-grain spatial information by performing a subsampling at each depth. One common pooling operation is the max pool. A simple example of a max-pooling operation is shown in figure
- In a max pool, the spatial dimensions are coarse-grained by replacing a small region (say 2x2 neurons) by a single neuron whose output is the maximum value of the output in the region. In physics, this pooling step is very similar to the decimation step in the Renormalization Group
- This reduces the dimension of outputs. For example, if the region we pool over is 2x2, then both the height and the width of the output layer will be halved.
- Generally, pooling operations do not reduce the "number of channels" of the convolutional layers because pooling is performed separately on each channel.



9

- In a CNN, the convolution and max-pool layers are generally followed by an all-to-all connected layer and a classifier such as a soft-max.
- CNNs are trained using the backpropagation algorithm. From a backpropagation perspective, CNNs are almost identical to fully connected NN architectures except with tied parameters …
- In fact, all neurons at a given convolutional layer represent the same filter, and hence can all be described by a single set of weights and biases. This reduces the number of free parameters by a factor of HxW at each layer. For example, for a layer with $D=10^2$ and $H=W=10^2$, this gives a reduction in parameters of nearly 10^6 … this allows for the training of much larger models!
- The immense success of CNNs for image recognition resides in the fact that filters (receptive fields) learned by the convolution layers should work well on similar tasks, not just on the ones they were originally trained for. We expect that, since images reflect the natural world, the filters learned by CNNs should transfer over to new tasks with only slight modifications and fine-tuning.

10

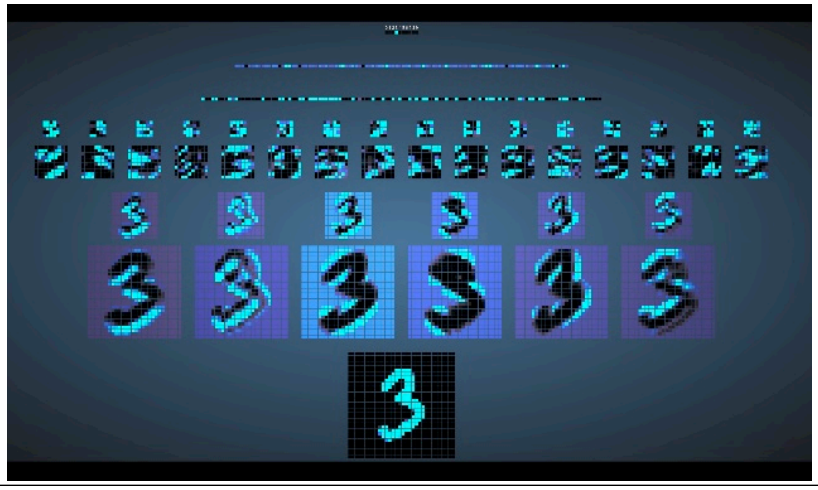


- Let's consider a concrete example of a deep convolutional neural network for digit-image classification where the input image size is 28x28x1 (i.e. grayscale)
- In the first layer, we apply the convolution operation with 6 filters of 5x5 so our output will become 24x24x6.
- Then we will apply pooling with 2x2 filter to reduce the size to 12x12x6
- In the second layer, we will apply the convolution operation with 16 filters of size 5x5. The output dimensions will become 8x8x16 …
- on which we will apply pooling layer with 2x2 filter and the size will reduce to 4x4x16.
- Finally, we will pass it through two fully connected layers to convert our image matrix into a classification matrix.

DNNs vs CNNs in action!

11

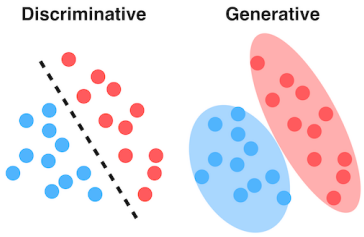
- Some fun with CNNs at:
  http://scs.ryerson.ca/~aharley/vis/conv/flat.html
- Compare with a fully connected DNN at:
  http://scs.ryerson.ca/~aharley/vis/fc/



Discriminative vs generative

12

- Many ML models are discriminative i.e. are designed to perceive differences between groups or categories of data. For example, recognizing differences between images of cats and images of dogs allows a discriminative model to label an image as "cat" or "dog".



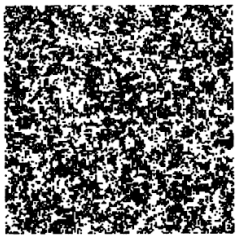
- However, discriminative methods have several limitations. First, they often require labeled data. Second, there are tasks that discriminative approaches simply cannot accomplish, such as drawing new examples from an unknown probability distribution.
- A model that can learn to represent and sample from a probability distribution is called generative.
- For example, given samples of configurations generated from one phase of a statistical complex model we may want to generate new samples from that phase. To be able to do so, we must turn to a new class of ML methods.

Generative models

- Generative models are a ML technique that allows to learn how to generate new examples similar to those found in a training dataset.
- The core idea is to learn a parametric model for the probability distribution from which the data was drawn. Once we have learned a model, we can generate new examples by sampling from the learned generative model (often via Markov Chain Monte Carlo (MCMC) methods)
- For this reason, generative models can do much more than just generate new examples. They can be used to perform a multitude of other tasks that require sampling from a complex probability distribution including "de-noising", filling in missing data etc.
- The ability to generate new examples requires models to be able to "generalize" beyond the examples they have been trained on, that is to generate new samples that are not samples of the training set.
- The models must be expressive enough to capture the complex correlations present in the underlying data distribution, but the amount of data we have is finite which can give rise to overfitting.

Energy based models

- The purpose of deep learning models is to encode dependencies between variables. In energy-based generative models this is obtained associating a scalar energy to each configuration of the variables which serves as a measure of compatibility.
- The idea is thus the usage of an energy as a metric for measurement of the model's quality: a high energy means a bad compatibility; Physics here enters for the first time.



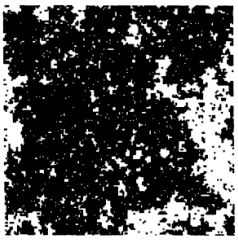
High energy: less likely

$$p(s) = e^{-E(s)} / Z$$

probability for state s in thermal equilibirum

$$Z = \sum_s e^{-E(s)}$$

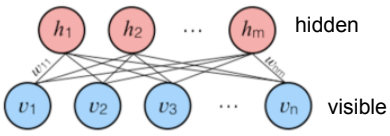
Normalization: partition function



Low energy: more likely

Hidden (latent) variables

- Latent or hidden variables are a powerful yet elegant way to encode sophisticated correlations between observable features.



- The reason for this is that marginalizing over a subset of variables – "integrating out" degrees of freedom in the language of physics – induces complex interactions between the remaining variables.
- In fact, consider a scalar energy which depends on visible, v, and hidden, h, variables. We can marginalize over the hidden units and ask about the resulting distribution over just the visible units

$$p(\vec{v}) = \int d\vec{h}\, p(\vec{v},\vec{h}) = \int d\vec{h}\, e^{-E(\vec{v},\vec{h})}/Z$$

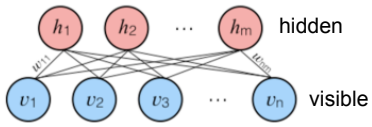
- We can also define a marginal energy, E'(**v**), using the expression

$$p(\vec{v}) = e^{-E'(\vec{v})}/Z \Rightarrow E'(\vec{v}) = -\ln\left[\int d\vec{h}\, e^{-E(\vec{v},\vec{h})}\right]$$

- Even if visible units are correlated only via hidden units, E'(v) includes all orders of interactions, i.e. among any pairs, triplets etc. of visible units.

Deep generative models: Restricted Boltzmann Machines

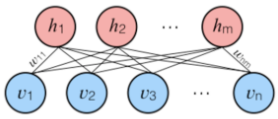
- Restricted Boltzmann Machine (RBM) are undirected graphical models that belong to so called energy based models.



- RBM are shallow, two-layer neural nets. The first layer is called the visible, or input layer, and the second is the hidden layer. Each circle represents a neuron-like unit called a node. The nodes are connected to each other across layers, but no two nodes of the same layer are linked.
- Thus, the restriction in a Restricted Boltzmann Machine is that there is no intra-layer communication. Each node is a locus of computation that processes input and begins by making stochastic decisions about whether to transmit that input or not.
- The energy function of an RBM takes the general functional form

$$E(\vec{v},\vec{h}) = -\sum_i a_i(v_i) - \sum_j b_j(h_j) - \sum_{i,j} v_i h_j w_{ij}$$

- where $a_i(\cdot)$ and $b_j(\cdot)$ are functions that we are free to choose. The most common choice is:



$$a_i(v_i) = \begin{cases} a_i v_i & \text{if } v_i \in \{0,1\} \\ v_i^2/2\sigma_i^2 & \text{if } v_i \in R \end{cases} \quad \text{and} \quad b_j(h_j) = \begin{cases} b_j h_j & \text{if } h_j \in \{0,1\} \\ b_j^2/2\sigma_j^2 & \text{if } h_j \in R \end{cases}$$

- For this choice of $a_i(\cdot)$ and $b_j(\cdot)$, layers consisting of discrete binary units are often called Bernoulli layers, and layers consisting of continuous variables are often called Gaussian layers.
- An RBM can have different properties depending on whether the hidden and visible layers are taken to be Bernoulli or Gaussian. The most common choice is to have both the visible and hidden units be Bernoulli. This is what is typically meant by an RBM.
- However, other combinations are also possible and used in the ML literature. When all the units are continuous, the RBM reduces to a multi-dimensional Gaussian with a very particular correlation structure.

17

RBMs have to be interpreted as a probabilistic models: 18

- ❑ At each time the RBM is in a certain state; this state refers to the values of neurons in the visible and hidden layers v and h
- ❑ the probability that this state of v and h can be observed is given by the following joint distribution

$$p(\vec{v},\vec{h}) = e^{-E(\vec{v},\vec{h})}/Z$$

- Here Z is called the partition function that is the summation over all possible pairs of visible and hidden vectors

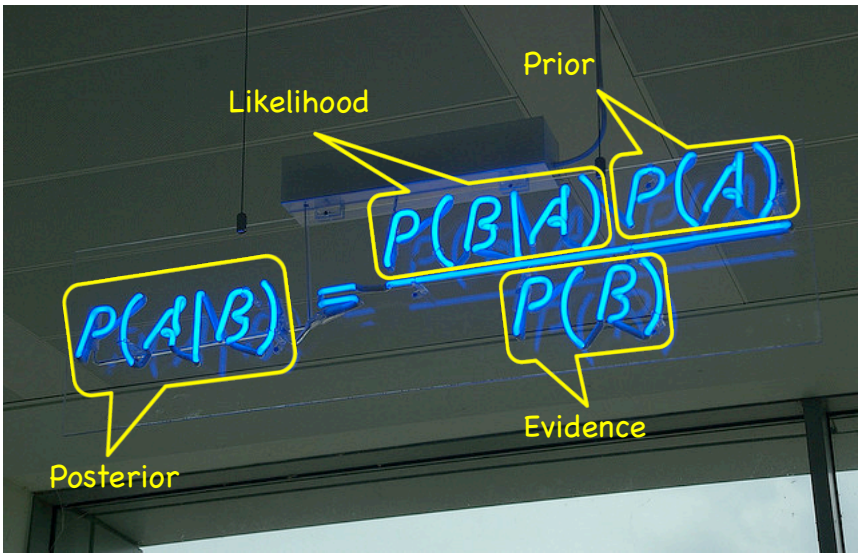
$$Z = \sum_{\vec{v},\vec{h}} e^{-E(\vec{v},\vec{h})}$$

- This is the point where RBMs meet Physics for the second time: the joint distribution is chosen to be the Boltzmann distribution
- Unfortunately it is very difficult to calculate Z; much easier is the sampling of the conditional probabilities of state h given the state v and conditional probabilities of state v given the state h
- It is easy because there are no connection among nodes in the visible layer as much as there are no connection among nodes in the hidden layer

Training RBMs

- We have understood the underlying reason for the incredible representational power of RBMs with a hidden layer: by combining many different hidden units, we can encode very complex interactions at all orders.
- Moreover, RBM can (machine) learn which order of correlations/ interactions are important directly from the data and we have not to specify them ahead of time. How to train them ?
- For any energy-based generative model, the energy function depends on some parameters that must be inferred directly from data. The goal of the training procedure is thus to use the available training data to fit these parameters.
- The most common approach is to maximize the log-likelihood (Maximum Likelihood estimate - MLE) over the training data set. Recall, that the log-likelihood characterizes the log-probability of generating the observed data using our generative model. By choosing the negative log-likelihood as the cost/loss function, the learning procedure tries to find parameters that maximize the probability of the data.

Bayesian digression



Bayesian inference

- Given two random events X and Y, Bayes's theorem says:

$$P[X,Y] = P[X|Y]P[Y] = P[Y|X]P[X]$$

where P[X,Y] is the joint probability function for X and Y, P[X] is the (marginal) probability for X and P[X|Y] is the conditional probability of X given Y.

- Given the data Y_E , where the subscript E means that data could be noisy, our *initial* criterion for a best inference *solution* will be the X that maximizes $P[X|Y_E]$, i.e. the most probable model given the data; we take

$$P[X|Y_E] = \frac{P[Y_E|X]P[X]}{P[Y_E]} \propto P[Y_E|X]P[X]$$

- $P[X|Y_E]$ is called the posterior probability, $P[Y_E|X]$ is the likelihood function, P[X] the prior probability and $P[Y_E]$ is a normalization constant called the evidence. The non-observable random variable X that is of primary interest is called the unknown.

21

- In Bayesian inference, the previous equation transfers the problem of specifying the posterior probability to the problem of specifying the likelihood function and the prior probability. These latter functions are generally ones about which we can either make reasonable assumptions or have specific knowledge
- Assume that before performing the measurement of Y , we have some information about the variable X. In Bayesian theory, it is assumed that this information can be coded into the probability density P[X]; it expresses what we know about the unknown, prior to the measurement
- The conditional probability of Y, P[Y|X], is called the likelihood function because it expresses the likelihood of different measurement outcomes with X given
- Looking at the Bayes' formula, we can say that solving an inference problem may be broken into three subtasks:

  (1) Based on all the prior information of the unknown X, find a prior probability density that reflects judiciously this prior information.

  (2) Find the likelihood function that describes the interrelation between the observation and the unknown.

  (3) Develop methods to explore the posterior probability density.

22

Estimates

- With a known posterior distribution, one can calculate different point estimates and spread or interval estimates.
- The point estimates answer to questions of the type "Given the data y and the prior information, what is the most probable value of the unknown X?"
- The interval estimates answer questions like "In what interval are the values of the unknown with 90% probability, given the prior and the data?"
- One of the most popular statistical estimates is the maximum a posteriori estimate (MAP). Given the posterior probability density P[X|Y] of the unknown X, the MAP estimate satisfies

$$X_{MAP} = \underset{X \in d(F)}{\arg\max} P[X \mid Y_E]$$

(arg max): points of the domain at which the function is maximized

⇒ the problem of finding a MAP estimate requires a solution of an optimization problem.

23

- Another common point estimate is the conditional mean (CM) of the unknown X conditioned on the data Y_E , defined as

$$X_{CM} = E[X|Y_E] = \int dX\, X P[X \mid Y_E]$$

- A technical advantage of CM is in the fact that smoothness properties of the posterior distribution are not as crucial as in the MAP estimation problem. A technical problem of CM could be that when the integration is typically over a very high-dimensional space common quadrature methods are not applicable.
- The most popular point estimate in statistics is the maximum likelihood (ML) estimate. This estimate answers the question "Which value of the unknown is most likely to produce the measured data Y_E?", is defined as

$$X_{ML} = \underset{X \in d(F)}{\arg\max} P[Y_E \mid X]$$

- From a Bayesian point of view this is equivalent to assume that the prior probability P[X] can be ignored: $P[X|Y_E] \propto P[Y_E|X]$
- In this case one can also obtain a different conditional mean:

$$X_{CM}^{(ML)} = \int dX\, X\, P[Y_E \mid X]$$

24

Computing gradients

- To continue with a general discussion, let's forget for a moment of deep models with hidden variables and denote a generative model by the probability distribution $p_w(x)$ with a Boltzmann form and its corresponding partition function Z_w. **Goal**: adapt weights such that the probability distribution of a set of training examples, $p_{data}(x)$, is approximately reproduced by $p_w(x)$
- The likelihood of the parameters given the data is:

$$P\left[Y_E \middle| X\right] = P\left[data \middle| \vec{w}\right] = \left\langle p_{\vec{w}}(x) \right\rangle_{x \in data}$$

- In MLE, the parameters of the model, $\{w_i\}$, are fit by maximizing the log-likelihood:

$$L_{\vec{w}} = \left\langle \log\left(p_{\vec{w}}\right) \right\rangle_{data} = -\left\langle E(x;\vec{w}) \right\rangle_{data} - \log Z_{\vec{w}}$$

- In writing this expression we made use of two facts:
  (i) our generative distribution is of the Boltzmann form, and
  (ii) the partition function does not depend on the data:

$$\left\langle \log Z_{\vec{w}} \right\rangle_{data} = \log Z_{\vec{w}}$$

- Performing MLE using SGD requires calculating the gradient of the log-likelihood with respect to the parameters $\{w_i\}$.
- To simplify notation and gain intuition, it is helpful to define quantities, $O_i(x)$, conjugate to the parameters w_i :

$$O_i(x) = -\frac{\partial E(x;\{w_i\})}{\partial w_i}$$

- Since the partition function Z is just the normalization for the Boltzmann distribution, we know that the usual statistical mechanics relationships between expectation values and derivatives of the log-partition function hold:

$$\left\langle O_i \right\rangle_{model} = \frac{\partial \log Z_{\vec{w}}}{\partial w_i}$$

- In terms of the $\{O_i\}$, the negative gradient of the loss function (i.e. the negative gradient of the negative of the log-likelihood) takes the form

$$-\frac{\partial}{\partial w_i}\left(-L_{\vec{w}}\right) = \frac{\partial L_{\vec{w}}}{\partial w_i} = \left\langle -\frac{\partial E(x;\vec{w})}{\partial w_i} \right\rangle_{data} - \frac{\partial \log Z_{\vec{w}}}{\partial w_i} = \left\langle O_i \right\rangle_{data} - \left\langle O_i \right\rangle_{model}$$

27

$$\Rightarrow \Delta w_i = \frac{\partial L_{\tilde{w}}}{\partial w_i} = \left\langle O_i \right\rangle_{data} - \left\langle O_i \right\rangle_{model}$$

- These equations have a simple and beautiful interpretation: The gradient of the log-likelihood with respect to a model parameter is a difference of two terms, one calculated directly from the data and one calculated from our model using the current model parameters.
- The data dependent term is known as the positive phase of the gradient and the model-dependent term is known as the negative phase of the gradient.
- This derivation also gives an intuitive explanation for likelihood-based training procedures: The gradient acts on the model to lower the energy of configurations that are near observed data points while raising the energy of configurations that are far from observed data points.
- Finally, we note that all information about the data only enters the training procedure through the expectations $\langle O_i \rangle_{data}$ and our generative model is blind to information beyond what is contained in these expectations.

28

$$\Delta w_i = \frac{\partial L_{\tilde{w}}}{\partial w_i} = \left\langle O_i \right\rangle_{data} - \left\langle O_i \right\rangle_{model}$$

- To use SGD, we must still calculate the expectation values that appear in this equation.
- The positive phase of the gradient – the expectation values with respect to the data – can be easily calculated using samples from the training dataset.
- However, the negative phase – the expectation values with respect to the model – is generally much more difficult to compute and, in almost all cases, one has to resort to either numerical or approximate methods. The fundamental reason for this is that it is impossible to calculate the partition function exactly for most interesting models in both physics and ML.
- One way to do this is draw samples from the model $p_w(x)$ and evaluate arbitrary expectation values using these samples; the samples from the model are often referred to as fantasy particles in the ML literature and can be generated using simple MCMC algorithms

Summary of the training procedure

- We now summarize the discussion above and present a general procedure for training an energy based model using SGD on the cost function.

Our goal is to fit the parameters of a model $p_w(x) = e^{-E(x;w)}/Z$

Training the model involves the following steps:

1. Read a minibatch of data, {x}
2. Generate fantasy particles {x'}, drawn from $p_w(x)$, using an MCMC algorithm (e.g., Metropolis)
3. Compute the gradient of log-likelihood using these samples and the equation:

$$\frac{\partial L_{\vec{w}}}{\partial w_i} = \left\langle O_i \right\rangle_{data} - \left\langle O_i \right\rangle_{model}$$

where the averages are taken over the minibatch of data and the fantasy particles from the model, respectively.

4. Use the gradient as input to one of the gradient based optimizers like SGD

Training RBMs

- With Bernoulli-Bernoulli RBMs, where hidden variables are present, the gradients are:

$$\Delta w_{ij} = \frac{\partial L(\{w_{ij}, a_i, b_j\})}{\partial w_{ij}} = \left\langle v_i h_j \right\rangle_{data} - \left\langle v_i h_j \right\rangle_{model}$$

$$\Delta a_i = \frac{\partial L(\{w_{ij}, a_i, b_j\})}{\partial a_i} = \left\langle v_i \right\rangle_{data} - \left\langle v_i \right\rangle_{model}$$

$$\Delta b_j = \frac{\partial L(\{w_{ij}, a_i, b_j\})}{\partial b_j} = \left\langle h_j \right\rangle_{data} - \left\langle h_j \right\rangle_{model}$$

- Calculating the negative phase of the gradient (i.e. the expectation value with respect to the model) requires that we draw samples from the model.
- "Luckily", the bipartite form of the interactions in RBMs were specifically chosen with this in mind: it makes it possible to calculate expectation values using a MCMC method known as Gibbs sampling.

Gibbs sampling and contrastive divergence

- The key reason for this is that since there are no interactions of visible units with themselves or hidden units with themselves, the visible and hidden units of an RBM are conditionally independent:

$$p(\vec{h} \mid \vec{v}) = \prod_j p(h_j \mid \vec{v}) \qquad p(\vec{v} \mid \vec{h}) = \prod_i p(v_i \mid \vec{h})$$

with:

$$p(h_j = 1 \mid \vec{v}) = \frac{1}{1 + e^{-\left(b_j + \sum_i w_{ij} v_i\right)}} = \sigma\left(b_j + \sum_i w_{ij} v_i\right) \qquad (*)$$

$$p(v_i = 1 \mid \vec{h}) = \frac{1}{1 + e^{-\left(a_i + \sum_j w_{ij} h_j\right)}} = \sigma\left(a_i + \sum_j w_{ij} h_j\right)$$

- In fact,

$$p(\vec{h} \mid \vec{v}) = \frac{p(\vec{v}, \vec{h})}{p(\vec{v})} = \frac{e^{-E(\vec{v},\vec{h})}}{Z\, p(\vec{v})} = \frac{e^{-E(\vec{v},\vec{h})}}{\sum_{\vec{h}'} e^{-E(\vec{v},\vec{h}')}}$$

thus

$$p(\vec{h} \mid \vec{v}) = \frac{e^{\sum_i a_i v_i} e^{\sum_j b_j h_j + \sum_{ij} h_j w_{ij} v_i}}{e^{\sum_i a_i v_i} \sum_{\vec{h}'} e^{\sum_j b_j h'_j + \sum_{ij} h'_j w_{ij} v_i}} = \prod_j \frac{e^{\left(b_j + \sum_{ij} w_{ij} v_i\right) h_j}}{\sum_{h'_j} e^{\left(b_j + \sum_{ij} w_{ij} v_i\right) h'_j}}$$

- The previous formula readily suggests that,

$$p(h_j \mid \vec{v}) = \frac{e^{\left(b_j + \sum_{ij} w_{ij} v_i\right) h_j}}{\sum_{h'_j} e^{\left(b_j + \sum_{ij} w_{ij} v_i\right) h'_j}}$$

- which for the binary case becomes

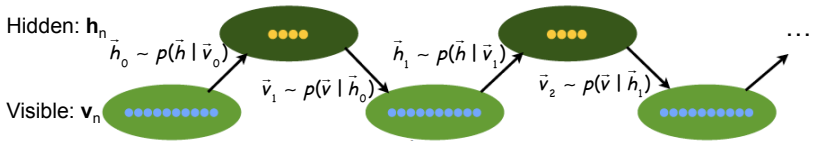
$$p(h_j = 1 \mid \vec{v}) = \frac{e^{\left(b_j + \sum_{ij} w_{ij} v_i\right)}}{1 + e^{\left(b_j + \sum_{ij} w_{ij} v_i\right)}} = \frac{1}{1 + e^{-\left(b_j + \sum_i w_{ij} v_i\right)}} = \sigma\left(b_j + \sum_i w_{ij} v_i\right)$$

c.v.d.

- Due to the symmetry involved in the defining equations, it can be similarly shown that,

$$p(v_i = 1 \mid \vec{h}) = \frac{1}{1 + e^{-\left(a_i + \sum_j w_{ij} h_j\right)}} = \sigma\left(a_i + \sum_j w_{ij} h_j\right)$$

- To calculate expectation values with respect to the model, we use (block) Gibbs sampling.

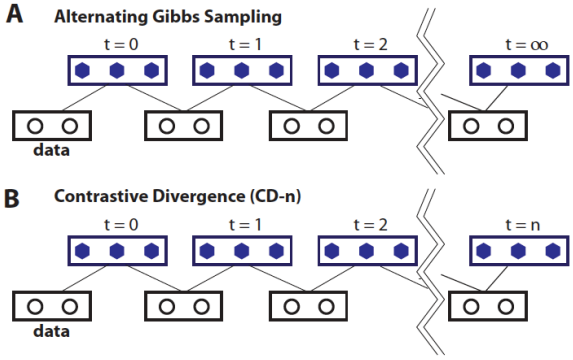


33

- The idea behind (block) Gibbs sampling is to iteratively sample from the conditional distributions:

$$\vec{h}_t \sim p(\vec{h} \mid \vec{v}_t) \qquad \vec{v}_{t+1} \sim p(\vec{v} \mid \vec{h}_t)$$

- The samples are guaranteed to converge to the equilibrium distribution of the model in the limit that $t \to \infty$. Stopping this procedure after n-steps is called Contrastive Divergence (CD-n)



- Truncating the Gibbs sampler prevents sampling far away from the starting point, which for CD-n are the data points in the minibatch. Therefore, our generative model will be much more accurate around regions of feature space close to our training data

34

- At the end of the Gibbs sampling procedure, one ends up with a minibatch of samples (fantasy particles): at convergence the model becomes generative!

- In fact, given that our generative model would be able to reproduce accurately the expectation values underlying the observed data, we should obtain that

$$\Delta w_{ij} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \cong 0$$

$$\Delta a_i = \langle v_i \rangle_{data} - \langle v_i \rangle_{model} \cong 0$$

$$\Delta b_j = \langle h_j \rangle_{data} - \langle h_j \rangle_{model} \cong 0$$

- And the parameters of the model do not substantially change anymore!

Lecture 12: Suggested books

- P. Mehta, C.H. Wang, A.G.R. Day, and C. Richardson, *A high-bias, low-variance introduction to Machine Learning for physicists*, arXiv:1803.08823 (Feb 2019)
- S. Haykin, *Neural Networks and Learning Machines*, Pearson (2009)
- S. Theodoridis, *Machine Learning. A Bayesian and Optimization Perspective*, Elsevier (2015)
- K.P. Murphy, *Machine Learning. A Probabilistic Perspective*, MIT press (2012)
- J. Kaipio, E. Somersalo, "Statistical and Computational Inverse Problems", Springer

35

After such a big learning, some fun & relax … some (suggested) movies on computers, simulation, ML and AI

- *2001: A Space Odyssey (1968)*
- *A.I. Artificial Intelligence (2001)*
- *Blade Runner (1982)*
- *Blade Runner 2049 (2017)*
- *Ex Machina (2014)*
- *Ghost in the shell (manga, 1995)*
- *Her (2013)*
- *I, Robot (2004)*
- *Matrix (1999)*
- *Moon (2009)*
- *Terminator (1984)*
- *Upgrade (2018)*
- *WarGames (1983)*

36

But remember …