Neural Computing

- Restricted Boltzmann Machines (RBMs)
- Deep networks
- Convolutional Networks
- Auto-encoders and Variational AEs
- Recurrent networks
- Backpropagation through Time (BPTT)
- Generative Adversarial Networks
- Etc, etc…

Markov chain Monte Carlo

- Markov chain Monte Carlo methods like Gibbs sampling provide a way of approximating the value of an integral…
- Monte Carlo = random sampling i.e. from an uniform distribution
- Markov chain = walking the right way (sampling from a distribution P(x) that is not uniform); i.e. make the likelihood of visiting a point x proportional to P(x) so that x^{t+1} depends only on x^t

Gibbs sampling

Given $x=x_1,\ldots,x_k$, a probabilistic choice is made for each of the k dimensions

For iterations $t=1,2,\ldots,T$, the approximated value for any x_i is $1/T\ \Sigma_t\ x_i^t$

Each choice of x_i depends on the other k-1 dimensions; the probabilistic walk goes like this:

Start with random $x_1,\ldots,x_k$

For each t, for each i, do:

$x_i^{t+1} \sim P(x|\ x_1^{t+1},\ldots,\ x_{i-1}^{t+1},\ x_{i+1}^{t},\ldots,\ x_k^{t})$

Gibbs sampling = neuro-dynamics

$$P(x|\ x_1^{t+1},\ldots,\ x_{i-1}^{t+1},\ x_{i+1}^{t},\ldots,\ x_k^{t}) = P(x_1^{t+1},\ldots,\ x_{i-1}^{t+1},\ x_i^{t},\ x_{i+1}^{t},\ldots,\ x_k^{t}) / P(x_1^{t+1},\ldots,\ x_{i-1}^{t+1},\ x_{i+1}^{t},\ldots,\ x_k^{t})$$

In a nutshell:

We want $x,y \sim P(x,y)$

But $P(x,y)$ is not known, so:

$x^{t+1} \sim P(x \mid y^t)$

$y^{t+1} \sim P(y \mid x^{t+1})$

Systematic sampling:
1. pseudo-random starting point
2. Elements selected at regular intervals according to some ordering

Combining probability models

Mixture: Take a weighted average of the distributions.
It can never be sharper than the individual distributions

Product: Multiply the distributions at each point and then renormalize (this is how an RBM combines the distributions defined by the hidden units)

More powerful than a mixture, but the normalization makes maximum likelihood learning difficult; but approximations allow learning

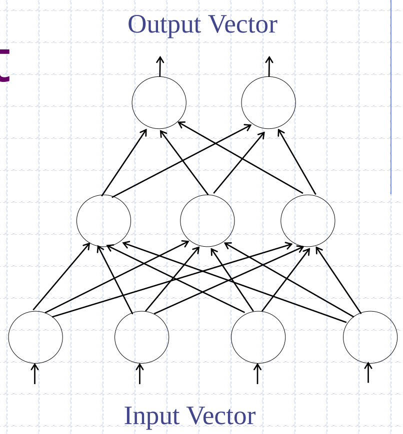
Composition: Use the values of the latent variables of one model as the data for the next model.

Works well for learning multiple layers of representation, but only if the individual models are undirected (i.e. symmetrical networks)

Generative neural networks

- If we connect binary stochastic neurons in a directed acyclic graph we get a Sigmoid Belief Net (Neal 1992)
- If we connect binary stochastic neurons using symmetric connections we get a Boltzmann Machine (Hinton & Sejnowski 1983)
  - If we restrict the connectivity in a special way, we get a Restricted Boltzmann Machine, which is easy to learn

Sigmoid Belief Net



x in {-1,1}

$P(X_j=x|parents(X_j))=$

$h(x/\mu\ \Sigma_i(W_{ij}x_i + \theta_j))$

where h is sigmoid function

Learning:

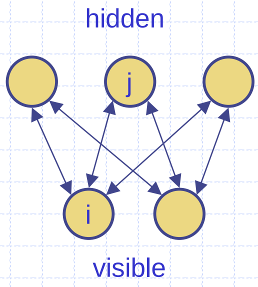
Clamp input vector

Perform Gibbs sampling on the hidden layer

Long enough and the values of the neurons will follow the distribution of the training data

Restricted Boltzmann Machines

(Smolensky 1986)



Restrict the connectivity of the Boltzmann Machine to make learning easier:

- Only one layer of hidden units (more layers later)
- No connections between hidden units, or between visible units

Thus, in an RBM, the hidden units are conditionally independent given the visible states!

The Energy of an RBM

(ignoring terms to do with biases)

binary state of visible unit i

binary state of hidden unit j

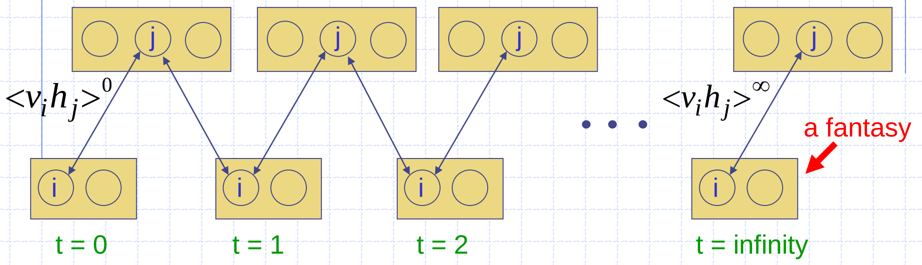
$$E(v,h) \;=\; -\sum_{i,j} v_i h_j w_{ij}$$

Energy with configuration v on the visible units and h on the hidden units

weight between units i and j

$$-\frac{\partial E(v,h)}{\partial w_{ij}} \;=\; v_i h_j$$

RBM Learning

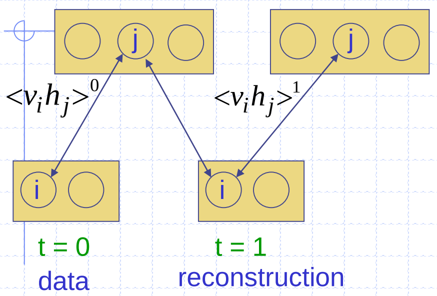


Start with a training vector on the visible units.

Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

$$\frac{\partial \log P(v)}{\partial w_{ij}} \;=\; \langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^\infty$$

A quick way to learn an RBM



Start with a training vector on the visible units

Update all the hidden units in parallel

Update all the visible units in parallel to get a reconstruction

Update the hidden units again

$$\Delta w_{ij} = \eta \, (<v_i h_j>^0 - <v_i h_j>^1)$$

This is not following the gradient of the log likelihood. But it works well... It is called Contrastive Divergence (CD1)

Calculating h^0, v^1, h^1...

$P(h_j=1|v)= \sigma(\Sigma_i W_{ij}v_i + \theta_j)$

Sample from uniform distribution in [0,1]

If $\sigma(\Sigma_i W_{ij}v_i + \theta_j) > U[0,1]$ Then set $h_j=1$

Else set $h_j=0$

This gives h^0 from v^0

Repeat now down to obtain v^1 from h^0

Repeat up to obtain h^1 from v^1

Persistent Contrastive Divergence

The main worry with CD is that there will be deep minima of the energy function far away from the data.

- To find these we need to run the Markov chain for a long time (maybe thousands of steps).
- But we cannot afford to run the chain for too long for each example (hence, we use CD-1 or CD-n).

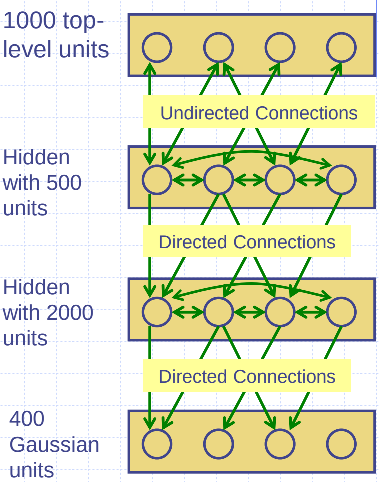
Maybe we can run the same Markov chain over many examples (persistent CD): For each mini-batch of examples, initialize the Markov chain at the state it ended at the previous iteration

if the learning rate is very small, this should be the same as running the chain for many steps and then doing a bigger weight update. In practice, the learning interacts with the Markov chain.

"Wherever the fantasies outnumber the positive data, the free-energy surface is raised. This makes the fantasies rush around hyperactively…" G. Hinton

Deep networks:
a stack of RBMs

- Stack of RBMs learned one at a time
- Unsupervised feature extraction
- Modular
- Levels of abstraction
- Efficient learning



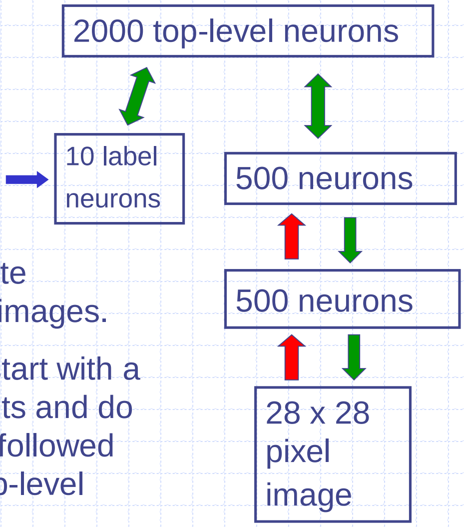
Training a deep network

- First train a layer of features that receive input directly from the pixels.
- Then treat the activations of the trained features as if they were pixels and learn features of features in a second hidden layer.
- It can be proved that each time we add another layer of features we improve a lower bound on the log probability of the training data.
  - The proof is based on a neat equivalence between an RBM and a deep directed model (described next)
- Fine-tuning: use backpropagation on deep directed network (regards all of the above as pre-training)

A neural model of digit recognition

The top two layers form an associative memory

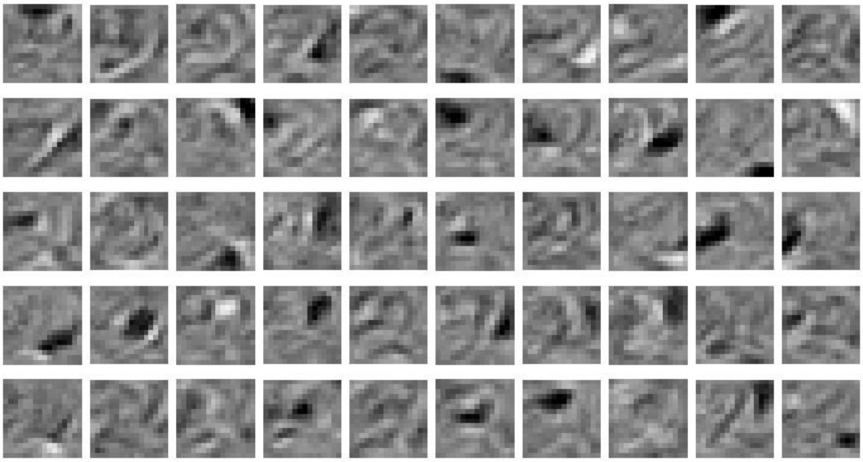
The energy valleys have names



The model learns to generate combinations of labels and images.

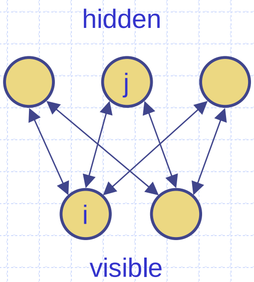
To perform recognition we start with a neutral state of the label units and do an up-pass from the image followed by a few iterations of the top-level associative memory.

The final weight vectors…



Each neuron grabs a different feature…

Sparsity



- Strong activation of a relatively small set of neurons
- To avoid co-adaptation of hidden neurons (possible even with random initial set of weights): use sparse coding

Popular extensions:

- Dropout (selects subsets of hidden layer to train) http://arxiv.org/abs/1207.0580
- Gaussian RBMs: Use a Gaussian as activation function of input layer (can be difficult to train)
- Rectified Linear Units (ReLUs) use activation function:

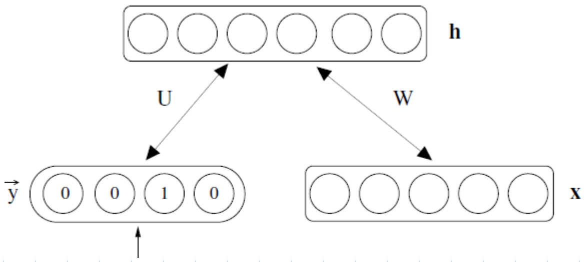
$$f(x) = \max(0,x)$$

Variations and Extensions

- RBMs with Gaussian input units (for integers or real numbers, instead of {0,1}, but can take much longer to train)
- Alternative: Make many copies of a binary unit; all copies have the same weights and the same adaptive bias, *b*, but they have different fixed offsets to the bias: *b - 0.5*, *b - 1.5*, *b - 2.5*,… (similar to using ReLUs)
- RBMs with lateral connections at visible layer: Conditional RBMs
- Deep Belief Networks (DBNs) trained with fine-tuning (uses Backprop), Deep Boltzmann Machines (DBMs)
- Discriminative RBMs, Auto-encoders, Recurrent Temporal RBMs (RTRBMs)

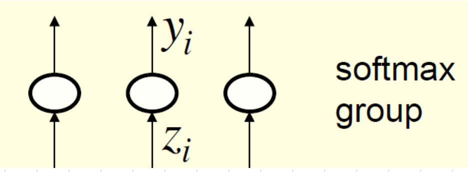
Discriminative RBMs

- Optimize directly p(y|x,h) instead of p(x,y,h)



- y is called a one-hot layer: forced to represent a probability distribution across discrete alternatives

Softmax



- Activate node with highest probability

$$y_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

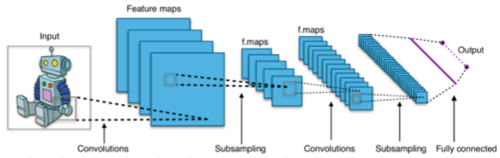
- The cost function to use with softmax is cross entropy:

$$C = -\sum_j t_j \log y_j$$

target value

Convolutional Neural Networks

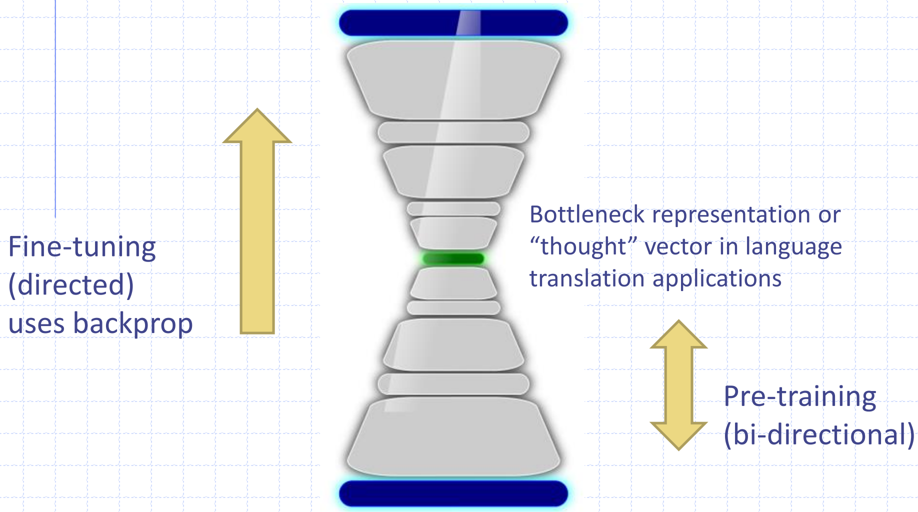
- Inspired by the visual cortex: multiple layers with overlapping patches of neurons (grey squares in the picture) mapped onto feature maps
- Very successful at image and video recognition
- Seeks to achieve translation (shift), rotation, size and illumination invariance...
- Convolution + Subsampling steps followed by single-hidden layer classifier trained with backprop.



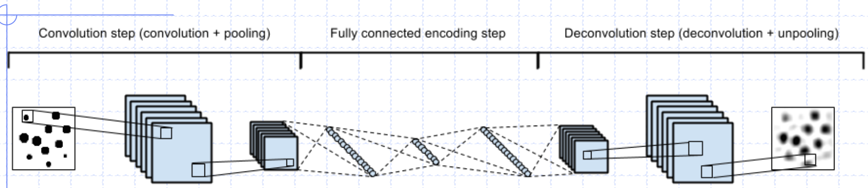
CNNs (cont.)

- MLPs do not scale well to higher-resolution pictures
- Convolution: overlapping input neuron areas with weight sharing
- Produces an activation map (similar to SOM) for each filter (set of weights)
- The activation map is then reduced (subsampling) by partitioning it into a set of now non-overlapping areas, and keeping only the neuron with the highest activation value in each area (max pooling)

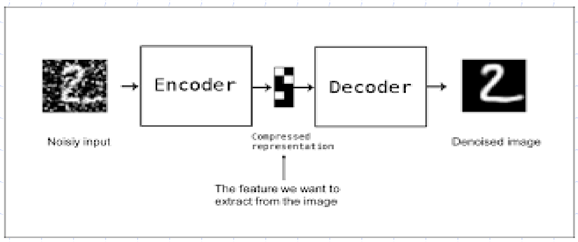
Auto-encoders



Convolutional Autoencoders



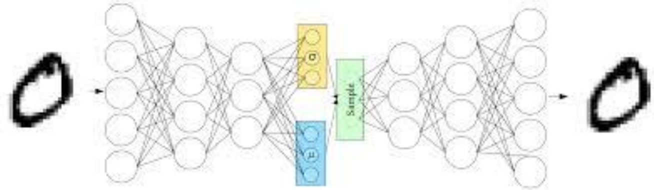
Denoising Autoencoders



Variational Autoencoders

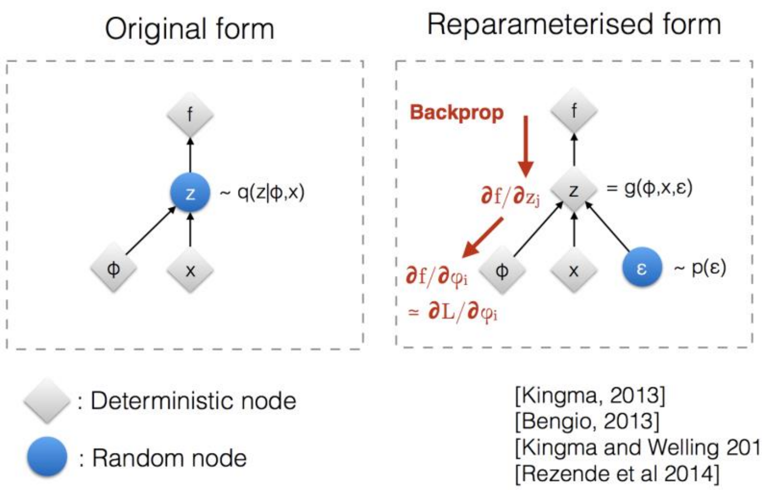
Instead of encoding into a bottleneck vector, encode into a distribution, e.g. two vectors: mean and variance.

To decode, sample from the distribution



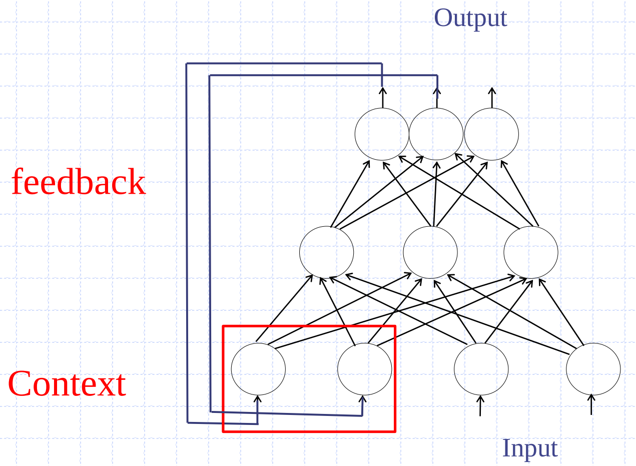
https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

Sampling still using backprop



[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

Recurrent networks (RNNs)

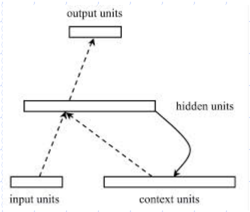


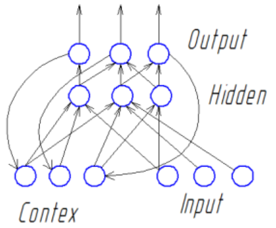
Used for time-series, sequence models!

Recurrent Neural Networks

Hopfield, Boltzmann, Echo state, Long-Short Term Memory (LSTMs), Nonlinear AutoRegressive network with eXogenous inputs (NARX), etc.

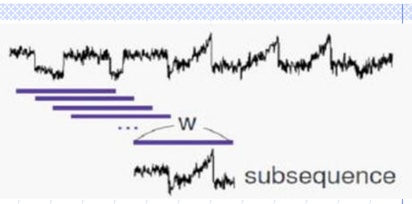
But also: Elman and Jordan Networks





Can be trained by standard backprop. and variations; e.g. input/output vectors are mapped to input+context/output (possibly using a sliding window approach)…

Sequence models



Sliding window vs. context units

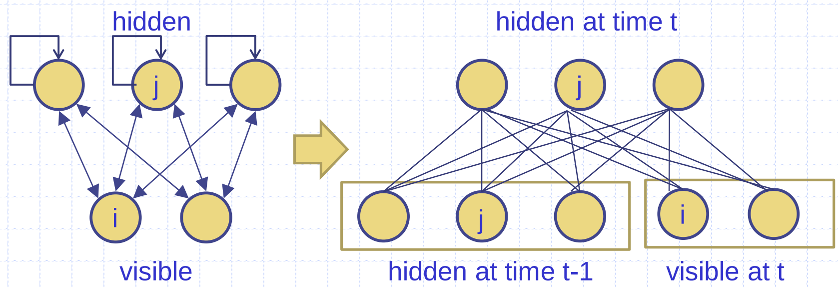
E.g. language modelling

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do…

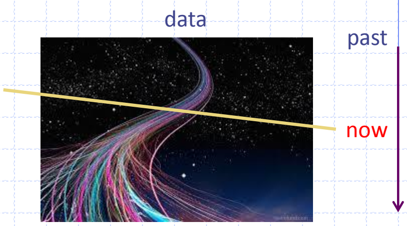
P(of|tired)=1

P(get|to)=1/2

Recurrent Temporal RBMs



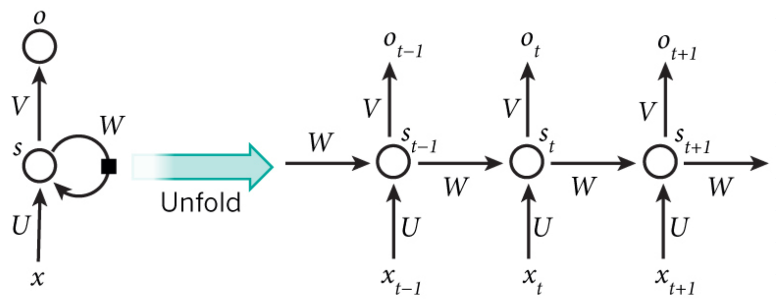
Online learning



- Streaming data
- No chance of cross-validation
- Given input (past) data, predict now
- Compare prediction with actual now, i.e. update the weights without a batch
- Repeat for tomorrow; now is gone i.e. it is in the past…
- Hope that model will do better when it sees a situation similar to now in the future (history repeats itself)

Backprop. through Time (1)

- Given data sequence x with target values t: (x_0,t_0), (x_1,t_1), (x_2,t_2),…
- Unfold a recurrent network through time (e.g. k=3 below)



Backprop. through Time (2)

- Train unfolded net with backprop. but in order, i.e. obtaining o_0, o_1, o_2…
- s_0 is normally a vector of zeros
- Each training example is of the form $(s_{t-1}, x_t, s_t, x_{t+1}, s_{t+1}, x_{t+2}, t_{t+2})$
- Typically use online learning
- After each example, average the weights to get the same U, V, W.

Time series models

- Inference is difficult in directed models of time series if we use non-linear distributed representations in the hidden units
  - It is hard to fit Dynamic Bayes Nets to high-dimensional sequences (e.g. motion capture data)
- So people tend to avoid distributed representations and use simpler methods (e.g. Hidden Markov Models)

Time series models (cont.)

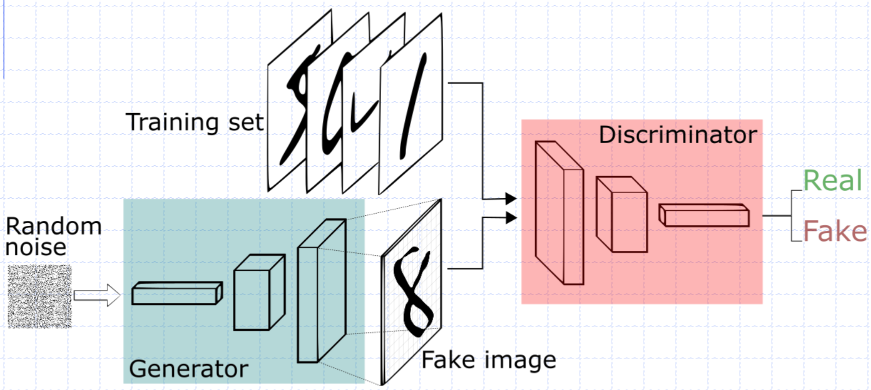
- If we need distributed representations, we can make inference simpler by:
  - Using an RBM for the interactions between hidden and visible variables.
  - Modeling short-term temporal information by allowing several previous frames to provide input to the hidden units and to the visible units.
- This leads to a temporal model that can be stacked
  - So we can use greedy learning to learn deep models of temporal structure…
  - Why does greedy work?

Generative Adversarial Nets

Learn to compare data distribution with random

The *generator* can be a decoder

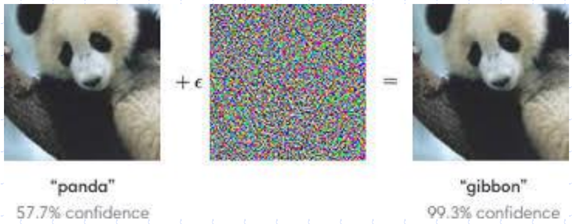
The discriminator uses backprop (also end-to-end)



Networks can be fooled

Adversarial perturbations

https://arxiv.org/abs/1312.6199



Distillation as a defence:

https://arxiv.org/abs/1511.04508


(a) Original, 0.9412 for 'wolf'

Current Research in Deep Learning

* Capsule Networks:
https://arxiv.org/abs/1710.09829
* Limitations of LSTMs:
https://papers.nips.cc/paper/8203-learning-to-reason-with-third-order-tensor-products.pdf
* Logic Tensor Networks:
https://arxiv.org/abs/1705.08968
* Neural Turing Machine and Differentiable Neural Computers:
https://arxiv.org/abs/1410.5401
* Memory networks:
https://arxiv.org/abs/1410.3916
* Gated Recurrent Units (GRUs):
https://qmro.qmul.ac.uk/xmlui/handle/123456789/22173
* Deep Belief Networks (DBN):
http://deeplearning.cs.cmu.edu/pdfs/Hinton_Osindero_fast.pdf
* Deep Boltzmann Machines (DBM):
http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS09_SalakhutdinovH.pdf
* Discriminative RBM:
http://machinelearning.org/archive/icml2008/papers/601.pdf
* Binarized Neural Networks:
https://arxiv.org/abs/1602.02830

Current Research (cont.)

* Temporal RBM:
http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS07_SutskeverH.pdf
* Recurrent Temporal RBM:
http://www.csri.utoronto.ca/~hinton/absps/rtrbm.pdf
* RNN-RBM:
http://machinelearning.wustl.edu/mlpapers/paper_files/ICML2012Boulanger-Lewandowski_590.pdf
* Factored RBM:
http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS2010_RanzatoKH10.pdf
* Factored Conditional RBM:
http://www.uoguelph.ca/~gwtaylor/publications/icml2009/fcrbm_supplementary.pdf
* Convolutional RBM/DBN:
http://www.cs.toronto.edu/~rgrosse/icml09-cdbn.pdf
* Auto-encoders:
http://www.iro.umontreal.ca/~lisa/publications2/index.php/publications/show/640
* Deep generative stochastic network:
http://www.iro.umontreal.ca/~lisa/publications2/index.php/publications/show/625
* Generative Adversarial Networks:
https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf
* Graph Neural Networks and Convolutional Graph Networks:
https://arxiv.org/pdf/1901.00596.pdf