March 17

update

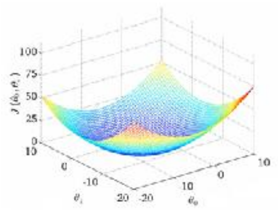
- Conference
- **Data Analytics and Management in Data Intensive Domains» (DAMDID)**
  - **Medicine, Neuro are included**
- http://damdid2017.frccsc.ru/en/conference_short.html
- PhD Workshop
  - Paper submission deadline June 23, 2017
- Regular submission date
  - Paper submission 18.06.2017

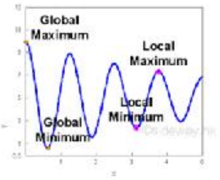
- Now: Liya and information gain
- Saturday 25th
  - First session on Matlab and psychtoolbox
  - Bring your laptop with matlab if possible

- Requests for topics?
- Wolfe and Horowitz next meeting?
  - Five factors that guide attention in visual search
  - Jeremy M. Wolfe & Todd S. Horowitz

Convexity

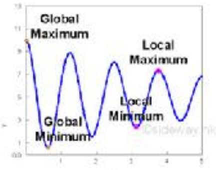
- Convex (shallow) architectures can be mathematically proven to have single global minimum
  - Remember, we want to find minimum error?
  - Logistic regression, Support Vector Machines
- The price of convexity is scale
  - Any 'shallow' algorithm can learn anything a deep one can, in theory
  - But O(n^2)…
- More detail http://videolectures.net/eml07_lecun_wia/





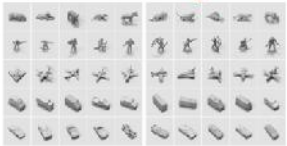
Concavity

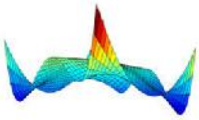
They work. Even without theoretical guarantees, the empirical evidence is overwhelming.
When empirical evidence and theory disagree, the theory is wrong.



Concave local prevents you from getting over the hill?

- Linear Classifier on raw stereo images: 30.2% error.
- K-Nearest-Neighbors on raw stereo images: 18.4% error.
- K-Nearest-Neighbors on PCA-95: 16.6% error.
- Pairwise SVM on 96x96 stereo images: 11.6% error
- Pairwise SVM on 95 Principal Components: 13.3% error.
- Convolutional Net on 96x96 stereo images: 5.8% error.



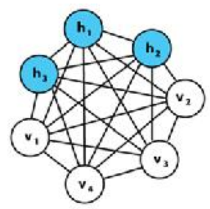


Add another dimension and go around the hill?

- The brain must use concave learning.
- Primate visual system has 10^20 layers of neurons (from retina to infero-temporal), but only 10^9 seconds in a lifetime
- Gradient descent in neural networks are very unreliable when the network is small
  - Or ‘exactly the right size for the problem’
  - So why not make the network much bigger than necessary?
  - Lots of local mimima most of them are pretty much the same/good
- Deep, concave architectures trade space (layers) for time (learning efficiency)
- Convinced? Lets look at RBM

autoencoder

Boltzman machine

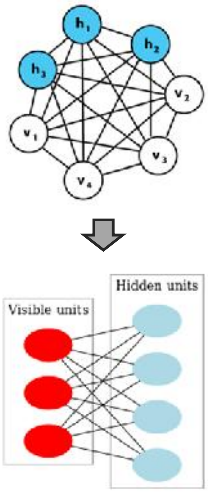


Three visible and 4 hidden units

- Like a Hopfield net, but stochastic and generative
  - Still Hebbian though
- Completely connected is theoretically interesting, but inefficient and impractical to train
  - Training examples needed increase exponentially with machine size

Restricted Boltzman machines

- Restriction: No connections within layer
  - Significantly reduces complexity
  - Still effective learning
- Extended versions can use real data, not just binary
- Successes in speech recognition software
  - Retrieve known patterns from noisy or incomplete data, remember?
- After training, the hidden layer activity can be used as 'visible' input layer to a higher level RBM
  - This is a preview of how 'Deep Learning Neural Nets' are implemented
- Hidden unit learn common 'features' of input
  - Text? -> topics
  - Images? Features, edges



Similar, but *Restricted* Boltzman machine

- Trained with 'contrastive divergence'
  - MUCH faster than gradient descent
  - Only allowed because of the new rule of no interlayer connections
  - Learning rate, momentum, weight cost, number of hidden nodes, etc is still hard to determine, but this is where the current research is being conducted

Training energy (see hopfield, 1982)

- Every Visible/Hidden pair of vectors has an energy

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i\in\text{visible}} a_i v_i - \sum_{j\in\text{hidden}} b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

- *V_i, H_j*
  - State of visible unit i and hidden unit j
- A_i, B_j are their biases
- W^{ij} is the weight between I and j

Evaluating energy of a pair

- Given that formula, is a given v/h pair energy good or not?

$$p(\mathbf{v},\mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})}$$

- Calculate the probability of a pair by comparing its energy to all other possible energies
- Z = sum of energy for all possible pairs of v,h

Evaluating energy of an input

- How much influence does this input have in training

$$p(\mathbf{v}) = \frac{1}{Z}\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$$

- Calculate the probability of a input by summing its energy over all possible hidden vectors' energy
- This can be changed by adjusting bias and weights in the original energy function

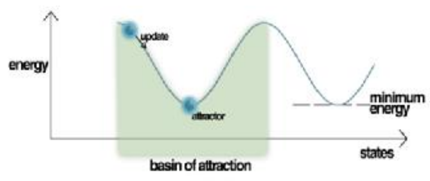
$$\Delta w_{ij} = \epsilon(\langle v_i h_j\rangle_{data} - \langle v_i h_j\rangle_{model})$$

Changing energy

- The contribution

$$p(\mathbf{v},\mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})}$$

- Z = sum of ... pairs of v,h



$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model})$$

This part is easy

This part is difficult
Hinton's big contribution was to devise a good estimate of this using contrastive divergence (2002)

We won't go any deeper than this, unless you want to construct the algorithm yourself

This is entirely *unsupervised learning*! You need lots of data, but it doesn't have to be labeled to infer what latent features are in the data

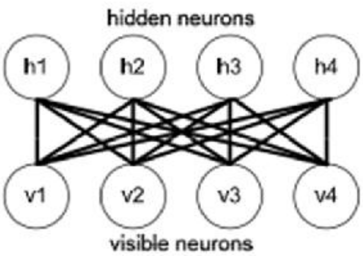
Generative

- Given a vector of hidden units, can we generate a possible/likely input?
- Gibb's sampling

classification

- The hidden units simply learn input features.
- If we want to predict something, we need another layer
- Discrete category predictions (classification) is usually done with Softmax
  - RBM + softmax won the netflix challenge for recommending movies
- The RBM layer is unsupervised, but the softmax uses supervised learning
  - Requires labeled examples of classes to predict
  - But builds on weights in the RBM layer that have previously been learned
- Want to predict what movies people will like based on previous watching habits?
  - First learn RBM layer of movie topics/features/themes by feeding in as much info on movies as you can find
    - (Reviews? Ads? Tags? Watching habits?)
    - Input = observed movies, hidden = unobserved learned features
  - Then add softmax layer and tweak weights with history – preference pairs
    - Input = observed movies,
    - hidden = unobserved learned features
    - Output = P(like movie X)

- RBM pseudocode




```
-Visible neurons initially set to a batch of training examples, denoted vis_batch_0
-Repeat until convergence {
  1) Sample hid_batch_0 from P(h|vis_batch_0)
    a) tmp_matrix_1 = vis_batch_0 * weights
    b) tmp_matrix_2 = tmp_matrix_1 + hid_biases
    c) tmp_matrix_3 = sigmoid(tmp_matrix_2)
    d) hid_batch_0 = tmp_matrix_3 > rand()
  2) Sample vis_batch_1 from P(v|hid_batch_0)
  3) Sample hid_batch_1 from P(h|vis_batch_1)
  4) Update parameters:
    a) weights += α(vis_batch_0ᵀ*hid_batch_0 - vis_batch_1ᵀ*hid_batch_1)
    b) vis_biases += α(vis_batch_0ᵀ*1 – vis_batch_1ᵀ*1)
    c) hid_biases += α(hid_batch_0ᵀ*1 – hid_batch_1ᵀ*1)
}
```