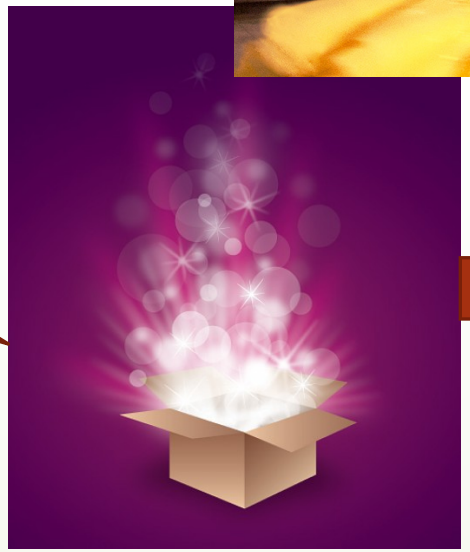
An Odyssey of Representation Learning

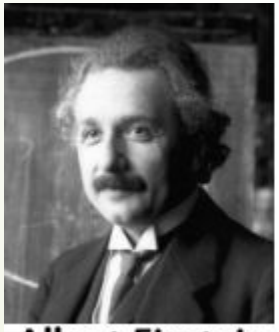
Yuan YAO
HKUST

Machine Learning has a Dream…



First principle for "Data Scientists"

- You have to know what you are doing …
- Principle of Parsimony (Occam's Razor): Everything should be kept as simple as possible, but no simpler. --Albert Einstein

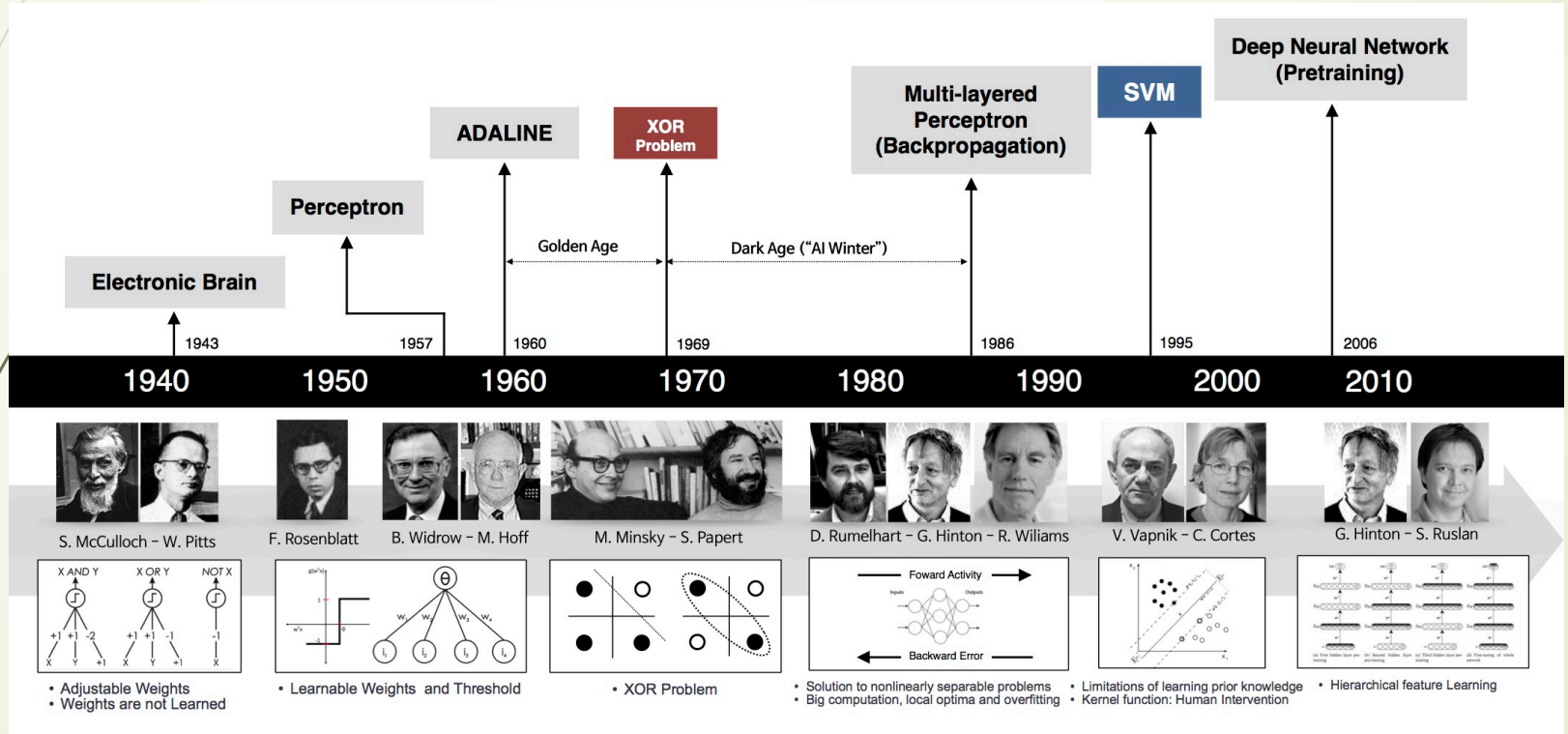


Albert Einstein

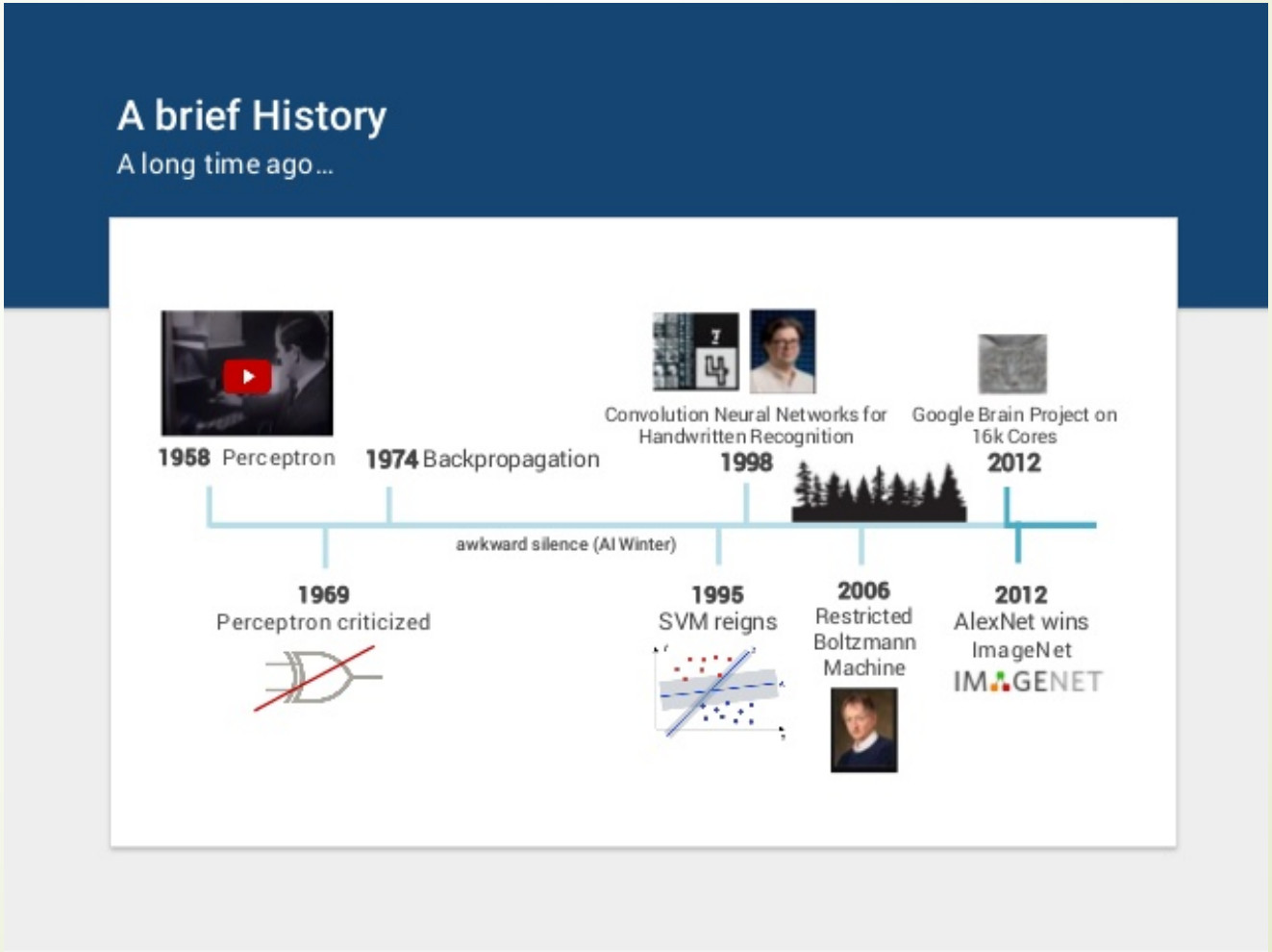
$$E = mc^2$$

Roger Sessions

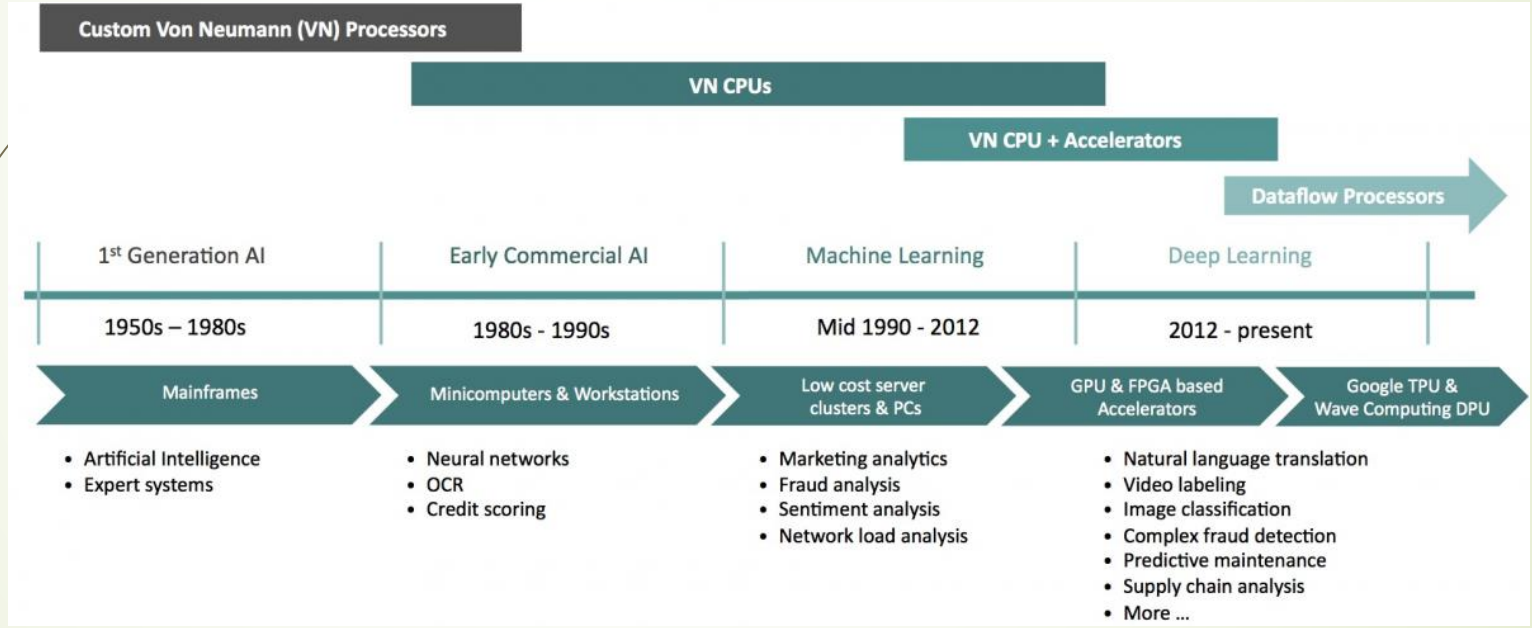
A Brief History of Neural Networks



Return of "Deep Representation" Learning ~ 2012

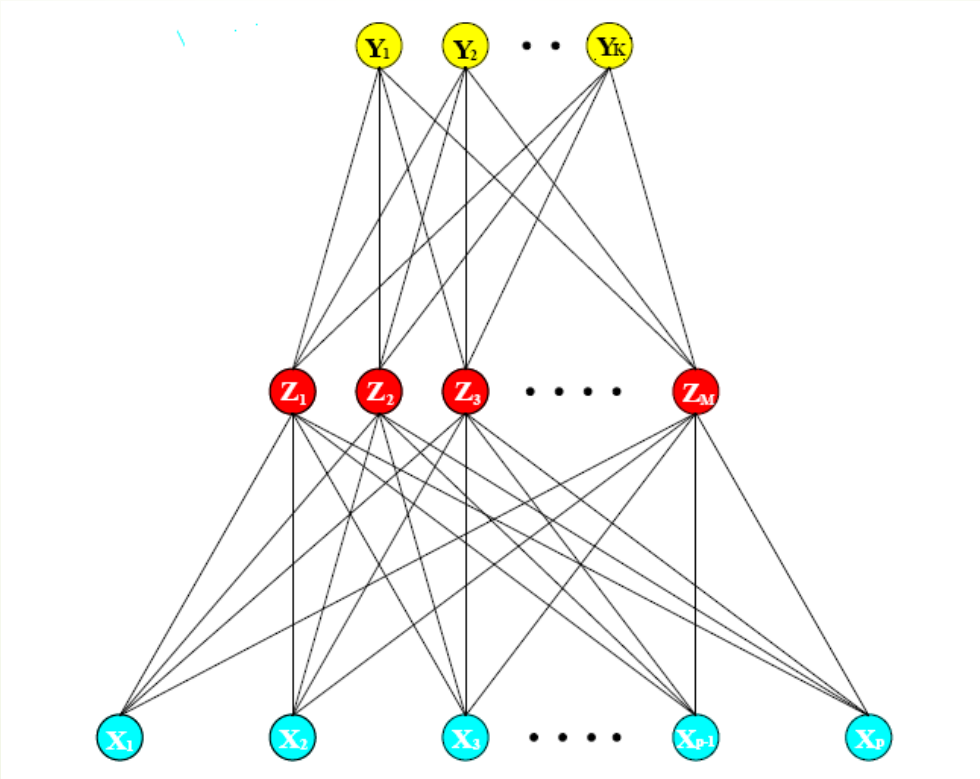


Hardware waves



1-Hidden Layer Neural Networks

- A two-stage regression or classification model



Single-Hidden Layer NN

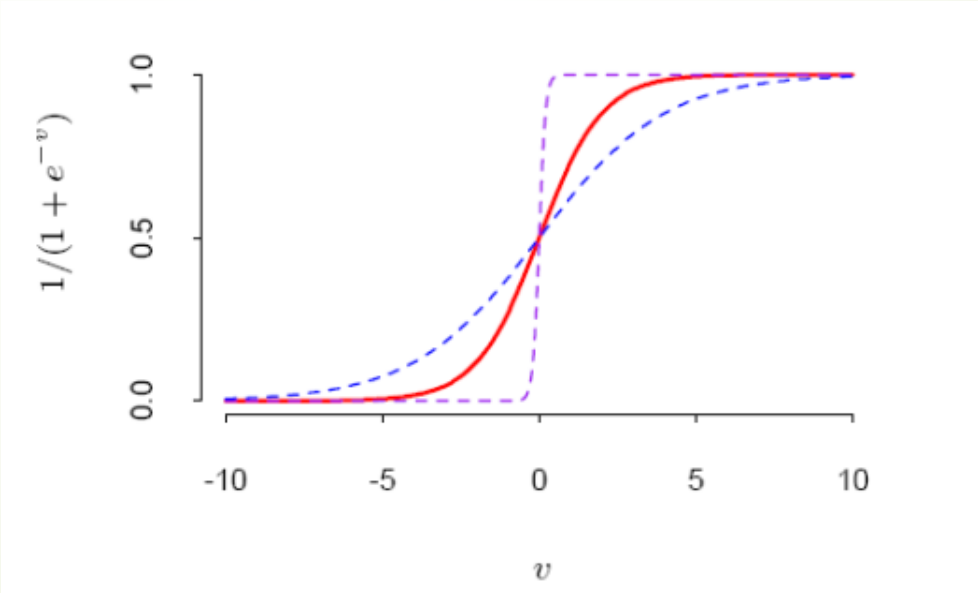
- Linear combination of X + sigmoid -> Z hidden layer
- Linear combination of Z + softmax -> Y

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X),\ m = 1, \ldots, M,$$
$$T_k = \beta_{0k} + \beta_k^T Z,\ k = 1, \ldots, K,$$
$$f_k(X) = g_k(T),\ k = 1, \ldots, K,$$

$$Z = (Z_1, Z_2, \ldots, Z_M), \text{ and } T = (T_1, T_2, \ldots, T_K).$$

Sigmoid Hidden Layer

- Sigmoid σ（y）=exp(y)/(1+exp(y))=1/(1+exp(-y))



Softmax Output

- Softmax output $g_k(T)$

$$g_k(T) = \frac{e^{T_k}}{\sum_{\ell=1}^{K} e^{T_\ell}}.$$

Projection Pursuit Regression

- Ridge function $g_m(\langle w,X \rangle)$: projection of X along w, then use that as features for regression
- Universal approximation of continuous functions when M grows to arbitrarily large numbers

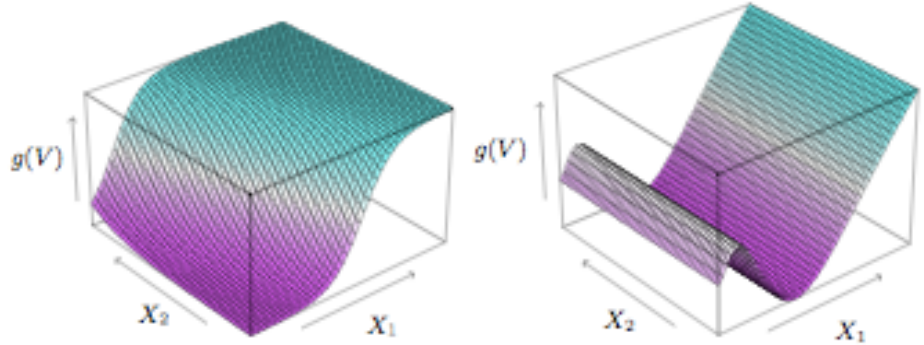


FIGURE 11.1. *Perspective plots of two ridge functions.*
(Left:) $g(V) = 1/[1+\exp(-5(V-0.5))]$, *where* $V = (X_1+X_2)/\sqrt{2}$.
(Right:) $g(V) = (V+0.1)\sin(1/(V/3+0.1))$, *where* $V = X_1$.

$$f(X) = \sum_{m=1}^{M} g_m(\omega_m^T X).$$

Least Square Fitting for PPR

- Objective function

$$\sum_{i=1}^{N}\left[y_i - \sum_{m=1}^{M} g_m(\omega_m^T x_i)\right]^2$$

- M and g can be tuned to avoid over-fitting, e.g. cross-validation

Weighted Least Square

- M=1, choose initial w_0, estimate g, e.g. splines
- Given g, estimate w by minimizing the least square loss

- A second order Taylor expansion:

$$g(\omega^T x_i) \approx g(\omega_{\text{old}}^T x_i) + g'(\omega_{\text{old}}^T x_i)(\omega - \omega_{\text{old}})^T x_i$$

- Least square regression leaves the following weighted least square:

$$\sum_{i=1}^{N} [y_i - g(\omega^T x_i)]^2 \approx \sum_{i=1}^{N} g'(\omega_{\text{old}}^T x_i)^2 \left[\left(\omega_{\text{old}}^T x_i + \frac{y_i - g(\omega_{\text{old}}^T x_i)}{g'(\omega_{\text{old}}^T x_i)} \right) - \omega^T x_i \right]^2$$

$$(\omega_m, g_m)$$

Training of Neural Networks

- Weights are unknown parameteres, denoted by θ
- Loss: sum-of-square error for regression and cross-entropy (deviance) for classification

$$\{\alpha_{0m}, \alpha_m;\ m = 1, 2, \ldots, M\}\ \ M(p+1) \text{ weights},$$
$$\{\beta_{0k}, \beta_k;\ k = 1, 2, \ldots, K\}\ \ K(M+1) \text{ weights}.$$

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.$$

$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i),$$

Least Square Fitting: Backpropogation (BP)

- Sum-of-Square error

$$R(\theta) \equiv \sum_{i=1}^{N} R_i$$
$$= \sum_{i=1}^{N}\sum_{k=1}^{K}(y_{ik} - f_k(x_i))^2,$$

- Derivative: Chain Rule for Composite Functions

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)z_{mi},$$
$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{i\ell}.$$

Gradient Descent

- Weights are updated by gradient descent

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}},$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}},$$

BP Equation

- Gradients can be rewritten as

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi},$$
$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = s_{mi} x_{i\ell}.$$

- where δ_{ki} and s_{mi} are errors in the output layer and hidden layer, satisfying the Backpropogation Equation

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki},$$

BP algorithm: Foreward and Backward

- Forward: given weights, compute predicted value $\hat{f}_k(x_i)$
- Backward: compute error δ_{ki} , then s_{mi} by BP equation
- Upgrade gradients using these errors
- Stochastic gradient descent algorithm for big data by taking a small batch of samples for every gradient update

11.4 Learning rates

- Step size γ is called Learning rates
- Batch learning: take all samples in the data to compute the gradients, when learning rates are often constants
- Online learning: take one (or mini-batch) sample after another to compute the gradients, when learning rates are
  - decay for numerical convergence
  - constants for fast tracking of data flow

Some challenges in NN training

- Initial choice
  - Can't be 0, which is a stationary point of BP algorithm
  - Random intialization
  - pretraining
- Local Optima
  - Regularization
  - Weight sharing
  - Perturbation of weights: drop out hidden units or parameters
- Architecture selection

神经网络训练的一些问题

- 过分拟合？
  - 权衰减是一种更加直接的正则化方法
  - 将惩罚项加入误差函数得到

$$R(\theta) + \lambda J(\theta)$$

  - Ridge/LASSO etc.

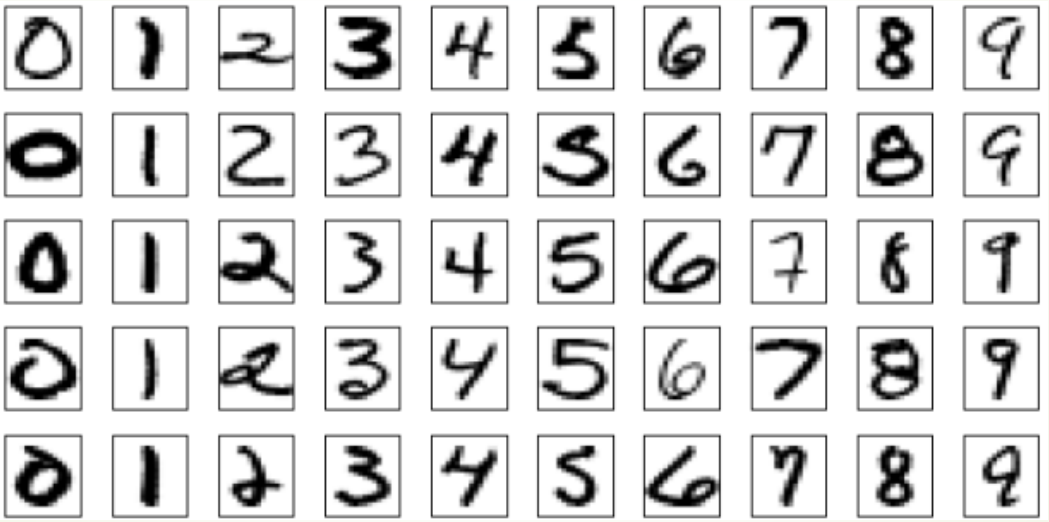
$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{m\ell} \alpha_{m\ell}^2$$

  - λ是大于0的调整参数，较大的值使权值向0收缩。 值由交叉验证估计
  - 还可以weight sharing (restricted)

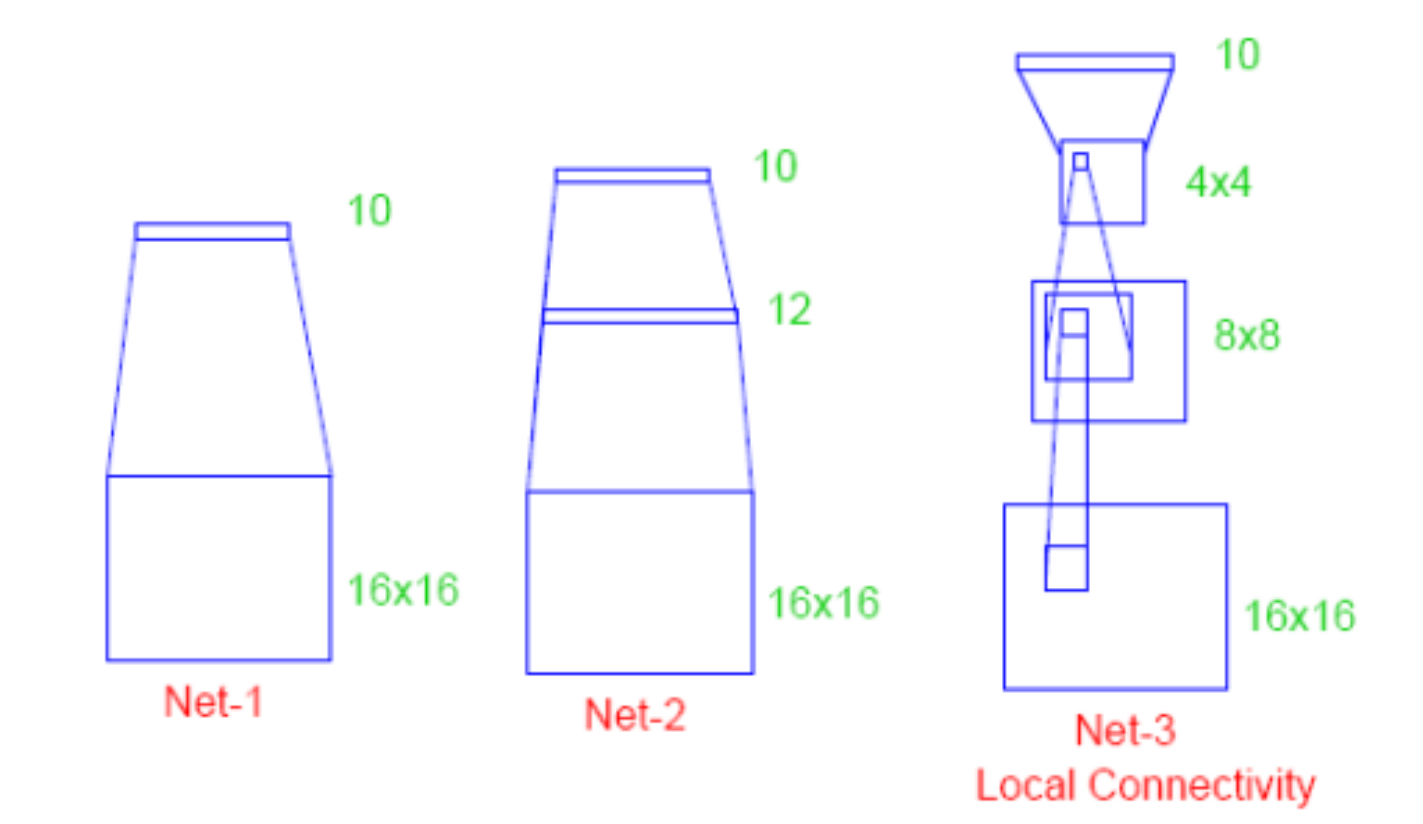
神经网络训练的一些问题

- 输入的scale对于结果的影响？
  - 最好对于所有的输入都进行标准化，这个可以保证在正则化过程中平等的对数据进行处理，而且为随机初值的选择提供一个有意义的值域
  - 一般在[-0.7, 0.7]上面随机选取均匀的权值
- 隐藏单元和层的数目：隐藏单元过少则模型可能不具备足够的灵活性，如果隐藏单元过多，则多余的收缩到0. 一般来说隐藏单元的数量在5到100之间，可以取合理大的数量，在用正则化加以训练，使得多余的变作0.
  - Deep networks?

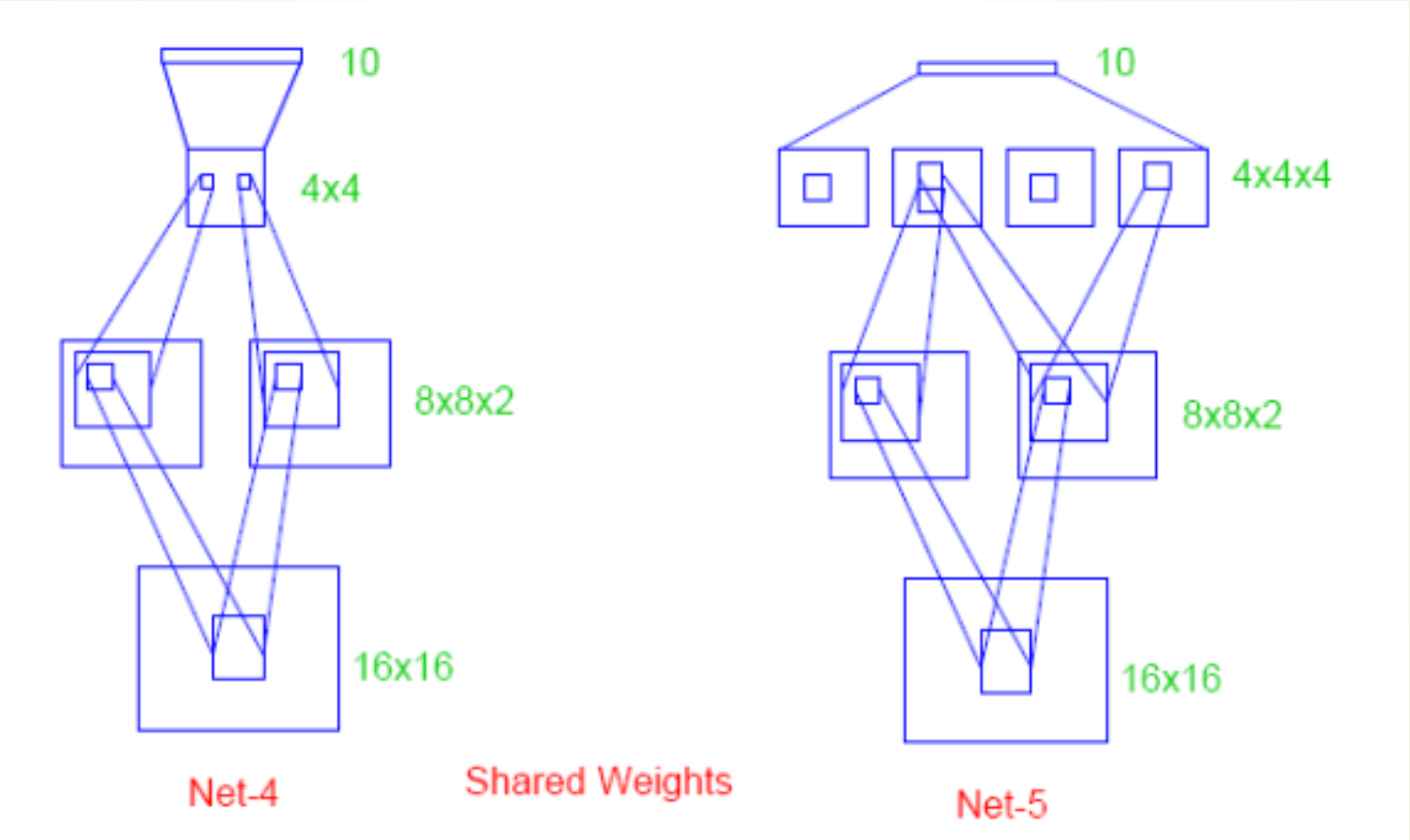
Example: Hand-written Digits



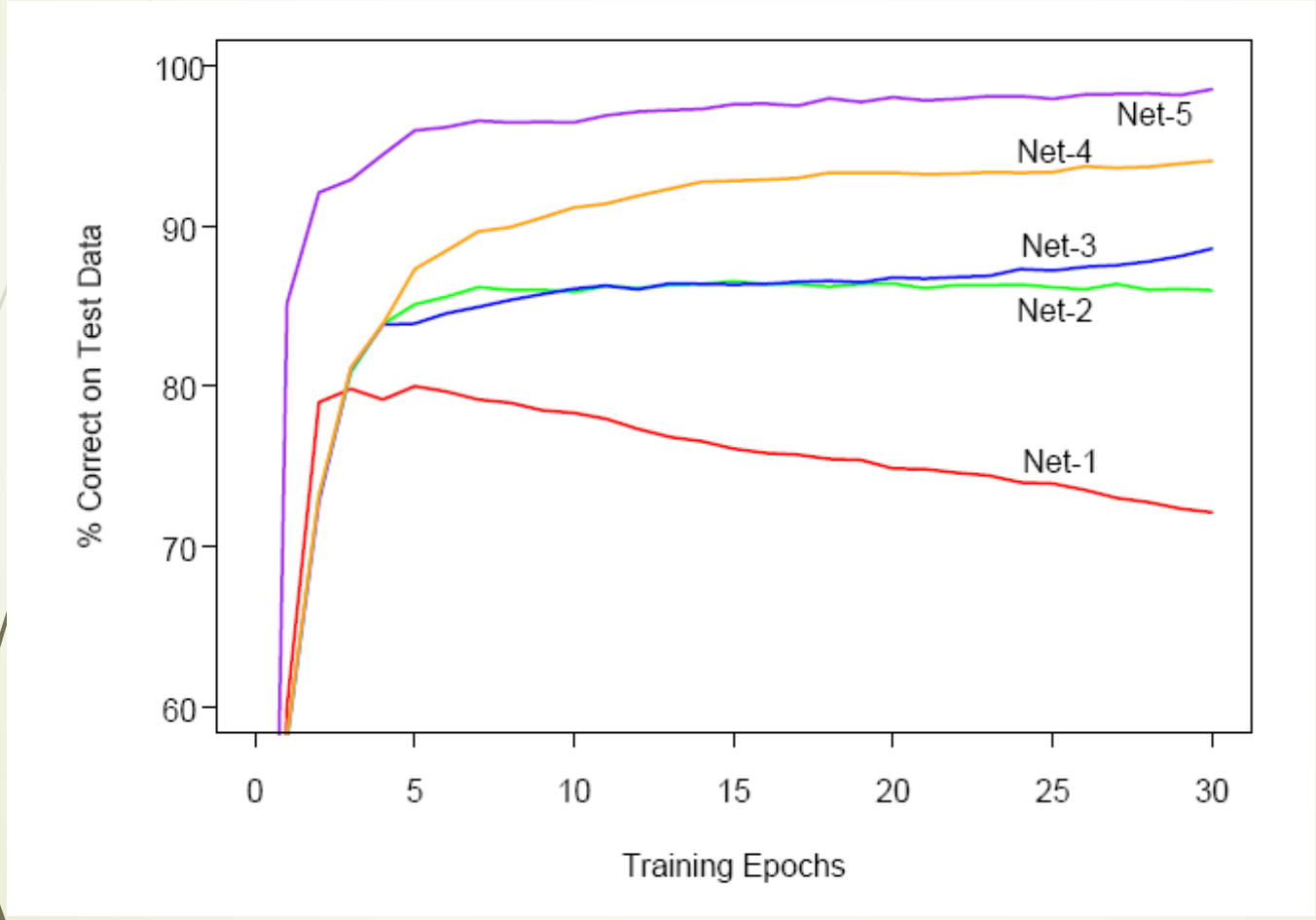
LeNets: five networks by LeCun



LeNet-4/5: weight sharing



Test Performance



Test Performance

	Network Architecture	Links	Weights	% Correct
Net-1:	Single layer network	2570	2570	80.0%
Net-2:	Two layer network	3214	3214	87.0%
Net-3:	Locally connected	1226	1226	88.5%
Net-4:	Constrained network 1	2266	1132	94.0%
Net-5:	Constrained network 2	5194	1060	98.4%

Deep Learning?

- Multilayer Perceptron tutorial: *Yue JIANG*
- Tensorflow Tutorial: *Lizhang MIAO*
- Reinforcement learning: *Akhan ISMAILOV*
- Some References
  - IPAM-UCLA 2012 Summer School: Deep Learning, Feature Learning:http://www.ipam.ucla.edu/programs/gss2012/ contains various video streams and slides
  - Andrew Ng's course at Stanford, Unsupervised Feature Learning and Deep Learning: http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial
  - Simons Institute Representation Learning 2017: https://simons.berkeley.edu/workshops/schedule/3750

Sparse Dictionary Learning

From Shallow Learning to Deep Learning

Unsupervised Learning as matrix factorization

- Shallow Learning of ***X***: $X \cong BC$
  - SVD/PCA: orthogonal B, C
  - Topic models: nonnegative B, C
  - Sparse coding/dictionary learning: B is dictionary (basis, frame, etc.), C is sparse
  - Conditional independence:

$$X = \sum_i b_i c_i^T, \quad b = [b_i], \quad C^T = [c_i^T]$$

$$b_{ik} \perp c_{ik} \mid k$$

Sparse Coding

- The aim is to find a set of basis vectors (dictionary) ϕ_i such that we can represent an input vector **x** as a linear combination of these basis vectors:

$$\mathbf{x} = \sum_{i=1}^{k} a_i \phi_i$$

- PCA: a complete basis
- Sparse coding: an *overcomplete* basis to represent $\mathbf{x} \in \mathbb{R}^n$ (i.e. such that $k > n$)
  - The coefficients a_i are no longer *uniquely* determined by the input vector **x**
  - Need additional criterion of **sparsity** to resolve the degeneracy introduced by over-completeness.

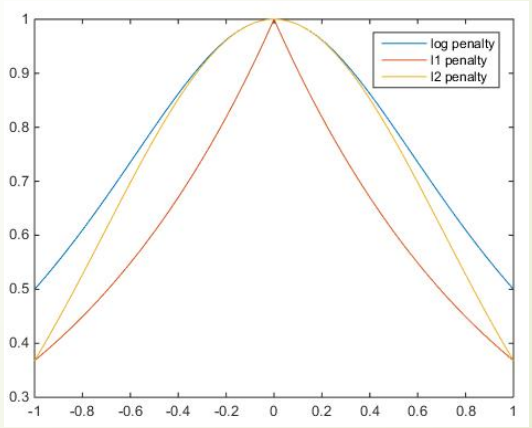
Sparsity Penalty

- We define the sparse coding cost function on a set of *m* input vectors as

$$\text{minimize}_{a_i^{(j)},\phi_i} \sum_{j=1}^{m} \left\| \mathbf{x}^{(j)} - \sum_{i=1}^{k} a_i^{(j)} \phi_i \right\|^2 + \lambda \sum_{i=1}^{k} S(a_i^{(j)})$$

where $S(.)$ is a sparsity cost function which penalizes a_i for being far from zero.

- "L_0-norm": $S(a_i) = \log(1 + a_i^2)$
- L_1 penalty: $S(a_i) = |a_i|_1$
- log penalty: $S(a_i) = \mathbf{1}(|a_i| > 0)$



Scale freedom

- In addition, it is also possible to make the sparsity penalty arbitrarily small by scaling down a_i and scaling ϕ_i up by some large constant.
- To prevent this from happening,

$$\begin{aligned} \text{minimize}_{a_i^{(j)},\phi_i} \quad & \sum_{j=1}^{m} \left|\left| \mathbf{x}^{(j)} - \sum_{i=1}^{k} a_i^{(j)} \phi_i \right|\right|^2 + \lambda \sum_{i=1}^{k} S(a_i^{(j)}) \\ \text{subject to} \quad & ||\phi_i||^2 \leq C, \forall i = 1, ..., k \end{aligned}$$

Olshausen and Field 1996

Identifiability: Scale

- One can remove the scale degree of freedom either in constraint form

$$\begin{aligned} \text{minimize} \quad & \|As - x\|_2^2 + \lambda \|s\|_1 \\ \text{s.t.} \quad & A_j^T A_j \leq 1 \; \forall j \end{aligned}$$

- Or Lagrangian form:

$$J(A, s) = \|As - x\|_2^2 + \lambda \|s\|_1 + \gamma \|A\|_2^2$$

where $\|A\|_2^2 := trace(A^T A)$ is simply the sum of squares of the entries of A (squared Frobenius norm).

Nonlinear Optimization

$$J(A,s) = \|As - x\|_2^2 + \lambda\|s\|_1 + \gamma\|A\|_2^2$$

- Bi-variate cost: not jointly convex, not differentiable
- But $J(A,s)$ is convex in A fixed s, and convex in s fixed A.

Alternating Optimization

- Initialize A randomly
- Repeat until convergence
  - Find the s that minimizes $J(A,s)$ for the A found in the previous step (LASSO)
  - Solve for the A that minimizes $J(A,s)$ for the s found in the previous step (Ridge Regression -> SVD)

Smoothing

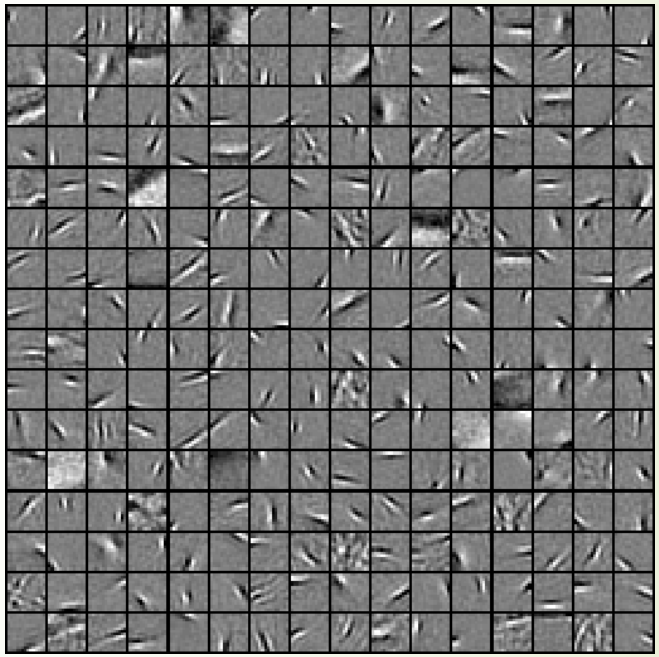
- So our final objective function:

$$J(A,s) = \|As - x\|_2^2 + \lambda\sqrt{s^2+\epsilon} + \gamma\|A\|_2^2$$

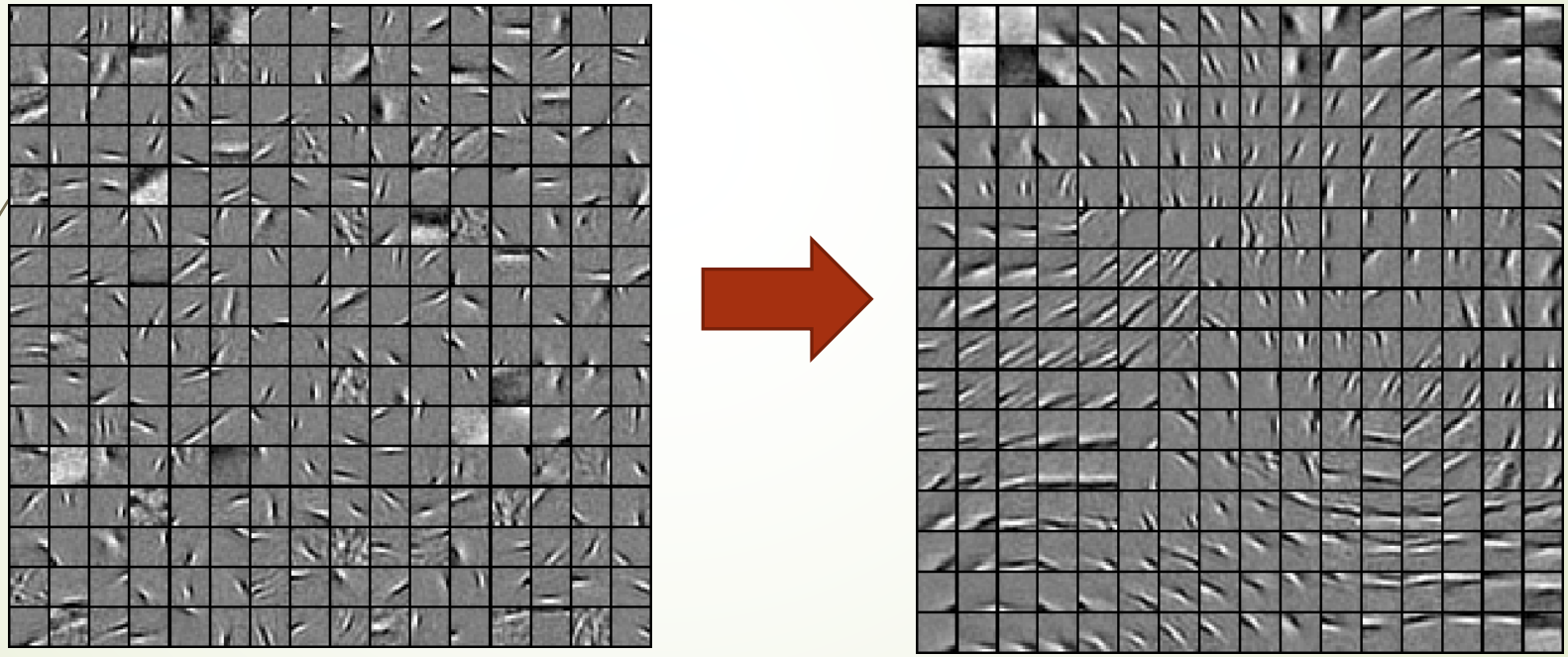
  - where $\sqrt{s^2+\epsilon}$ is shorthand for $\sum_k \sqrt{s_k^2+\epsilon}$
  - the third term $\|A\|_2^2$ is simply the sum of squares of the entries of A (squared Frobenius norm)
- Then you have a smooth objective function, restricted convex in *A* and *s*
- Gradient descent such as BP algorithm can be applied here

Dictionary Learned from natural image patches

- http://ufldl.stanford.edu/wiki/index.php/Exercise:Sparse_Coding
- http://ufldl.stanford.edu/wiki/resources/sparse_coding_exercise.zip



Adjacent features should be similar?



Topographic Sparse Coding: Group LASSO

- Group adjacent features in group LASSO norm

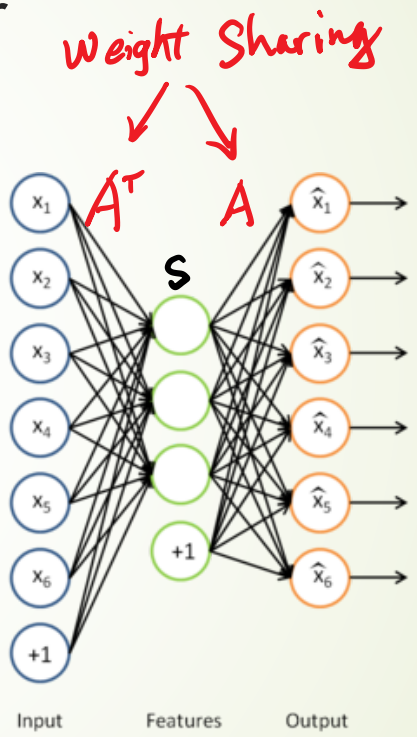
$$J(A,s) = \|As - x\|_2^2 + \lambda \sum_{\text{all groups } g} \sqrt{\left(\sum_{\text{all } s \in g} s^2\right) + \epsilon} + \gamma \|A\|_2^2$$

- Example: 3-by-3 neighborhood as 'adjacency'

$$\sqrt{s_{1,1}^2 + \epsilon} \quad \Rightarrow \quad \sqrt{s_{1,1}^2 + s_{1,2}^2 + s_{1,3}^2 + s_{2,1}^2 + s_{2,2}^2 + s_{3,2}^2 + s_{3,1}^2 + s_{3,2}^2 + s_{3,3}^2 + \epsilon}$$

A Neural Network Interpretation: Sparse Autoencoder

- Single-hidden layer NN, to learn features A to reconstruct signals x
- Encoding: $s = A^T x$
- Decoding: As
- *Sample torch codes: https://github.com/torch/tutorials/blob/master/3_unsupervised/2_models.lua*



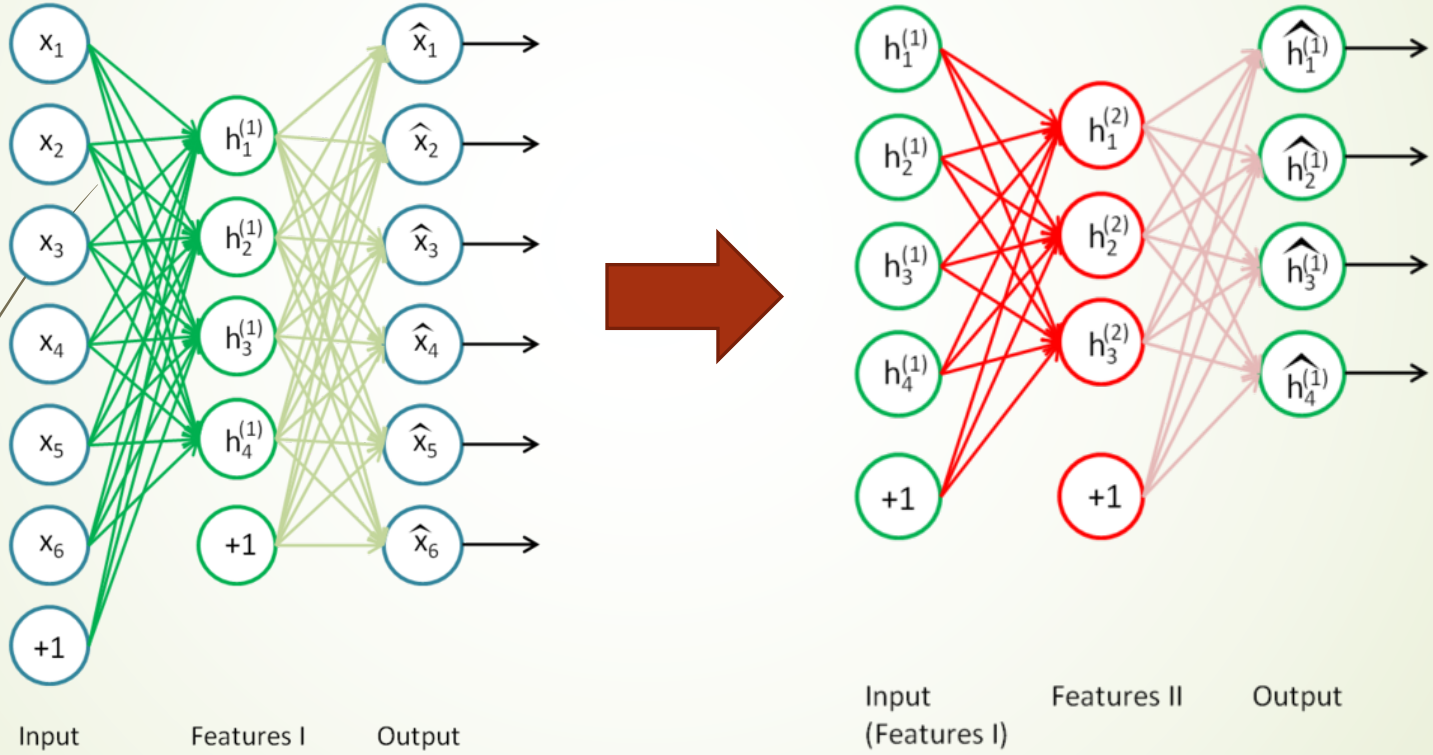

```
-- encoder
encoder = nn.Sequential()
encoder:add(nn.Linear(inputSize,outputSize))
encoder:add(nn.Tanh())
encoder:add(nn.Diag(outputSize))

-- decoder
decoder = nn.Sequential()
decoder:add(nn.Linear(outputSize,inputSize))


-- tied weights
if params.tied and not params.hessian then
-- impose weight sharing
decoder:get(1).weight = encoder:get(1).weight:t()

decoder:get(1).gradWeight = encoder:get(1).gradWeight:t()
```

Deep Networks



When does Dictionary Learning work?

- Daniel Spielman, Huan Wang, and John Wright, **Exact Recovery of Sparsely-Used Dictionaries,** arXiv:1206.5882
- Agarwal, Anandkumar, Jain, Netrapalli, **Learning Sparsely Used Overcomplete Dictionaries via Alternating Minimization**, arXiv:1310.7991
- Sanjeev Arora, Rong Ge, and Ankur Moistra, **New Algorithms for Learning Incoherent and Overcomplete Dictionaries,** arXiv:1308.6273, 2013
- Barak, Kelner, and Steurer, **Dictionary Learning and Tensor Decomposition via the Sum-of-Squares Method,** http://arxiv.org/abs/1407.1543, 2014

Application of Deep Learning: Neurostyle

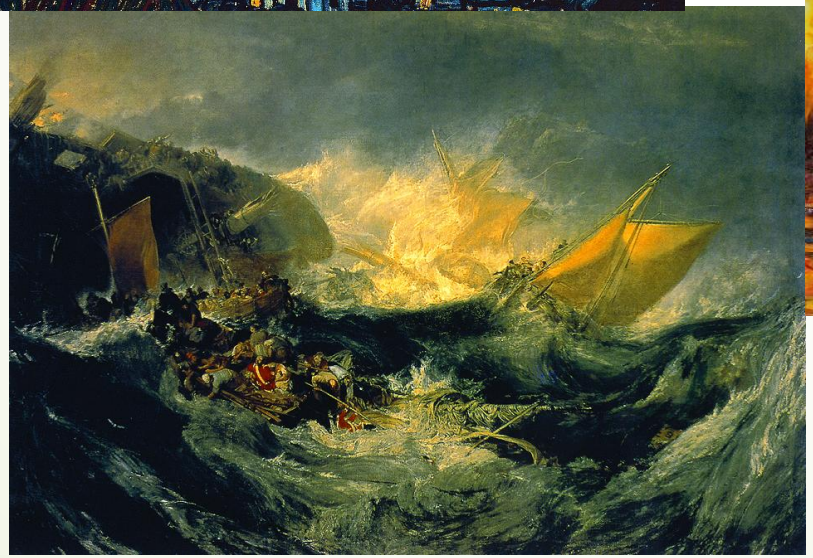
Neural Style by J C Johnson

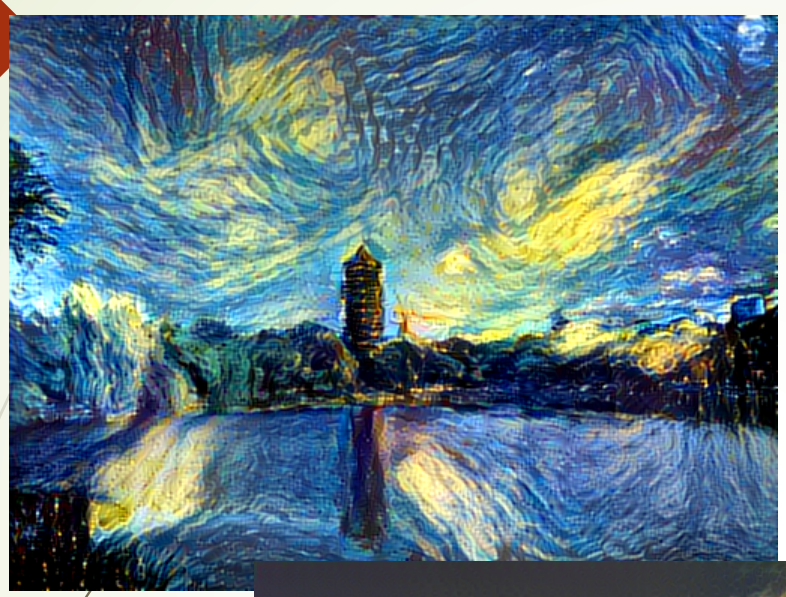
- Website: https://github.com/jcjohnson/neural-style

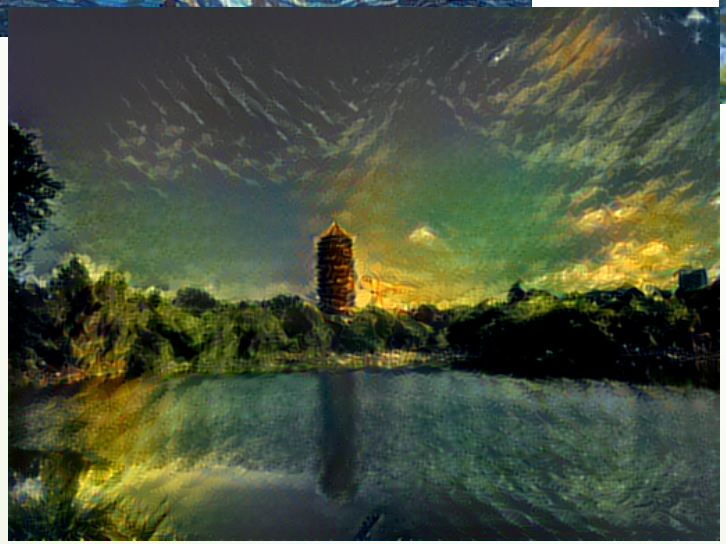
- A torch implementation of the paper
  - *A Neural Algorithm of Artistic Style*,
  - by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge.
  - http://arxiv.org/abs/1508.06576

博雅塔·未名湖

Left: *Vincent Van Gogh*, Starry Night
Right: *Claude Monet*, Twilight Venice
Bottom: *William Turner*, Ship Wreck



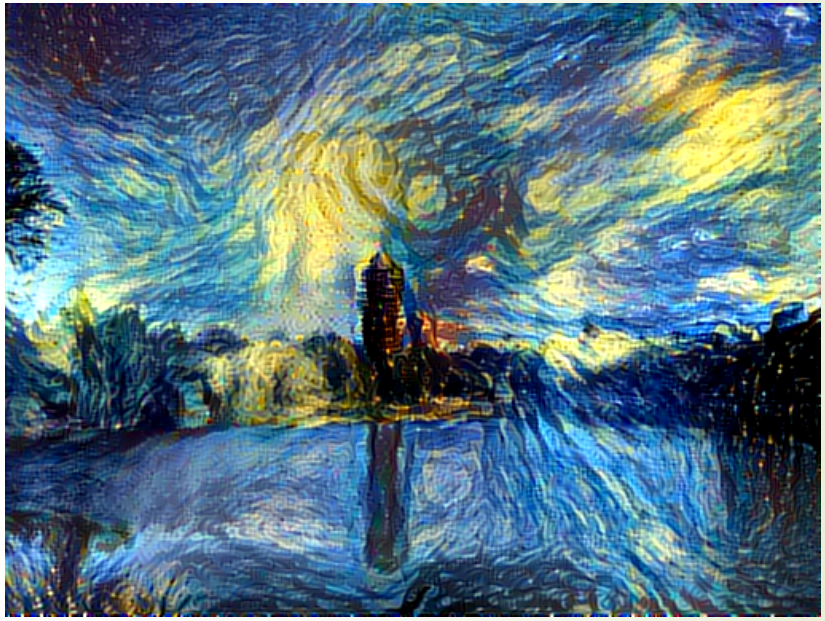




Application of Deep Learning:
Content-Style synthetic pictures
By “neural-style”

Run neural_style

- ssh bicmr@bicmr2 -p 2211
- cd neural-style/
- th neural_style.lua -style_image examples/inputs/starry_night.jpg -content_image examples/inputs/noname.jpg -output_image examples/outputs/noname_starry.png
- Different run might be different!



Style-Content Decomposition