Feedforward Autoassociative Memory

Centre for Theoretic Neuroscience

January 28, 2009

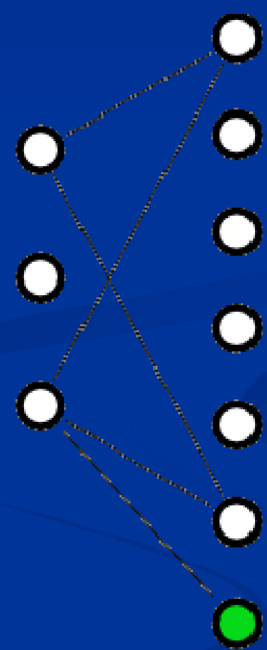
Yichuan Tang
Terry Stewart

Feedforward Autoassociative Memory

- Content based memory retrieval
- Fault tolerance/Noise invariance
- Fast Retrieval
- Associative abilities
- Can be folded to have dynamics
- Focused on feedforward networks

Linear Associators

- Linear mapping between input and output
- Same as NEF optimum decoding with 1 additional const rate hidden unit to serve as 'DC gain', or in other words allows function to be offset from origin



Linear Associators

Let X be the matrix of input data NxD_i,
N is the number of samples and D_i
is number of dimensions of input;
Let Y be matrix of output NxD_o,
where D_o is the dimensions of output

$W^T \tilde{X}^T = Y^T$ Model

Various Solutions

Linear Associator

$$\tilde{X}^T = USV^T$$
$$W^T = Y^T V S^+ U^T$$

Solving using SVD

$$\tilde{X}W = Y$$
$$pinv(\tilde{X})\tilde{X}W = pinv(\tilde{X})Y$$

Produce smallest norm of W

$$W = \tilde{X} \backslash Y$$

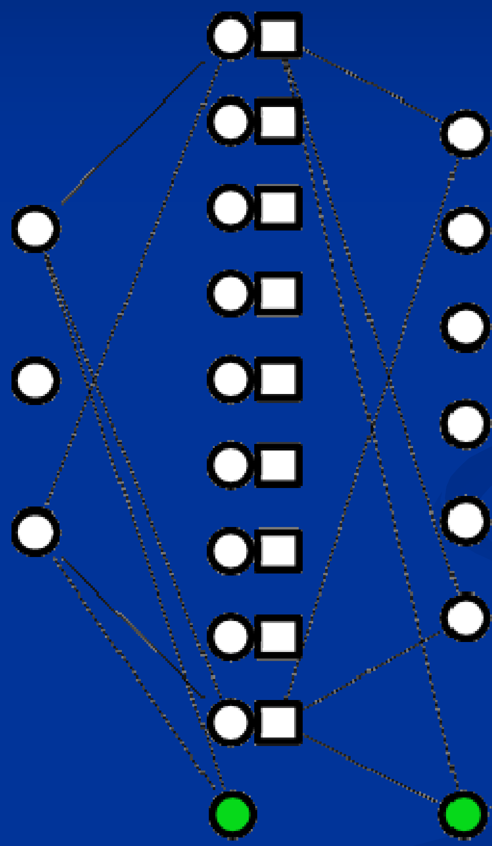
In Matlab, produces most sparse solution

$$\tilde{X}^T \tilde{X} W = \tilde{X} Y$$

$$W = pinv(\tilde{X}^T \tilde{X}) \tilde{X} Y$$

Useful if $N > D_i$

Two Layer Feedforward



Two Layer Feedforward

Let activities of the l-th layer before nonlinear activation be $\mathbf{a}^l$

Let activities of the l-th layer after nonlinear activation be $\mathbf{x}^l$

The weight matrix W^l defines transformation between the $l-1$ and l layers

σ is the non linear sigmoid activation function, i.e. $\tanh(\cdot)$

$a_i^l = \sum_j W_{i,j}^l x_j^{l-1} + b_i^l$ $\qquad x_i^l = \sigma(a_i^l)$

$\mathbf{a}^l = \tilde{W}^l \tilde{\mathbf{x}}^{l-1}$ $\qquad \tilde{\mathbf{x}}^l = [\mathbf{x}^l; 1]$ $\qquad$ Much cleaner!

$$E = \frac{1}{K} \sum_{k=1}^{K} (\mathbf{x}^L)^T (\mathbf{x}^L)$$

Minimize the Error

We want to find $\dfrac{\partial E}{\partial W_{ij}}$

$$\mathbf{a}_i^l = W_{(i,:)}^l \mathbf{x}^{l-1}$$

$$\frac{\partial E}{\partial W_{i,j}} = \frac{\partial \mathbf{a}_i^l}{\partial W_{i,j}} \frac{\partial E}{\partial \mathbf{a}_i^l}$$

Let $\boldsymbol{\phi}^l$ be $\dfrac{\partial E}{\partial \mathbf{a}^l}$, $d_l \times 1$

$$\frac{\partial E}{\partial W_{i,j}} = \mathbf{x}_j^{l-1} \phi_i^l$$

So all we need is $\boldsymbol{\phi}^l$ and $\mathbf{x}^{l-1}$

Backpropagation

- Doesn't live up to the hype of the name
- Just plain old simple matrix calculus

Since $\mathbf{a}^{l+1} = W^{l+1}\sigma(\mathbf{a}^l)$

$$\boldsymbol{\phi}^l = \frac{\partial \mathbf{x}^l}{\partial \mathbf{a}^l}\frac{\partial \mathbf{a}^{l+1}}{\partial \mathbf{x}^l}\frac{\partial E}{\partial \mathbf{a}^{l+1}}$$

$$\boldsymbol{\phi}^l = D^l (W^{l+1})^T \boldsymbol{\phi}^{l+1}$$

D^l is a diagonal matrix.
For tanh sigmoids, $D_{ii} = 1 - (x_i^l)^2$

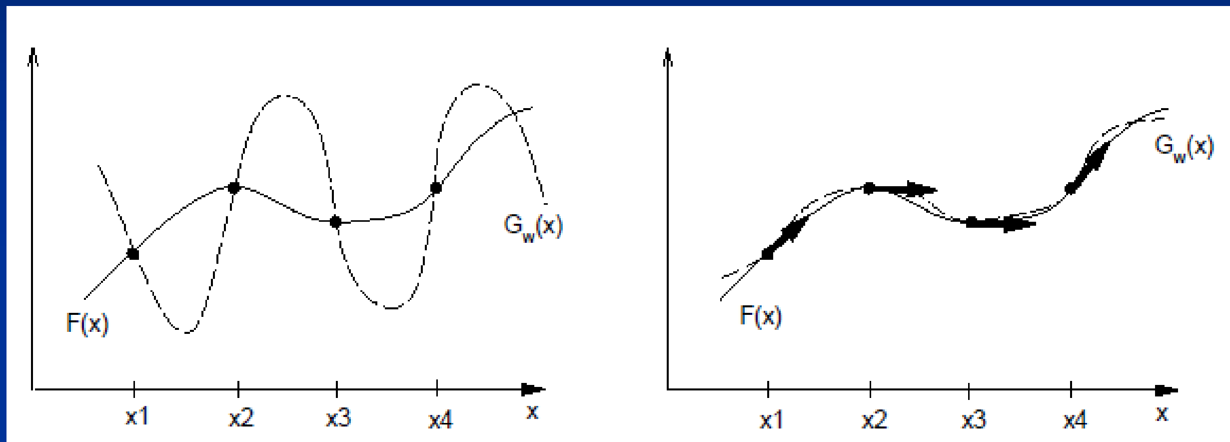
Tips and Tricks

- Whiten or Sphere the data!!
- Weight initialization is key
- "Stochastic" gradient descent
- Conjugate gradient methods
- Regularization, i.e. weight decay
- Bayesian prior
- Adaptive gradient descent cons
- Hacks such as momentum

Tangent Prop

- The idea is to not only fit to train->model pairs of point, but also to (*training set* partial derivative)->(*model* partial derivative) pairs at specific points
- The name "Tangent Prop" came from the fact that for classification, the tangent to the class specific manifold have *training set* partial derivative of zero at various points

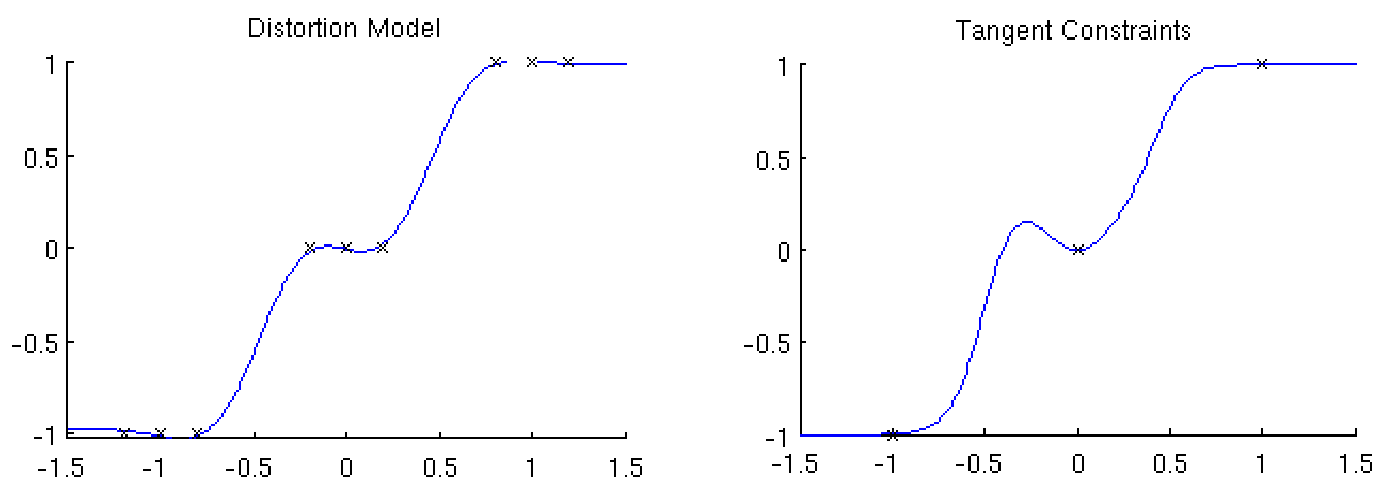
Tangent Prop



Simard et al. 2001

Let $G_w(\mathbf{x}^0)$ be the function ANN computes
As the input $\mathbf{x}^0$ changes due to noise, transformationes etc..
We want $G_w(\mathbf{x}^0)$ to change according to the training set,
Which is often zero for classification or denoising

Tangent Prop



NEF

$$a_i(x) = \frac{1}{\tau_i^{ref} - \tau_i^{RC} \ln\left(1 - \frac{J_i^{threshold}}{\alpha_i x + J^{bias}}\right)}$$

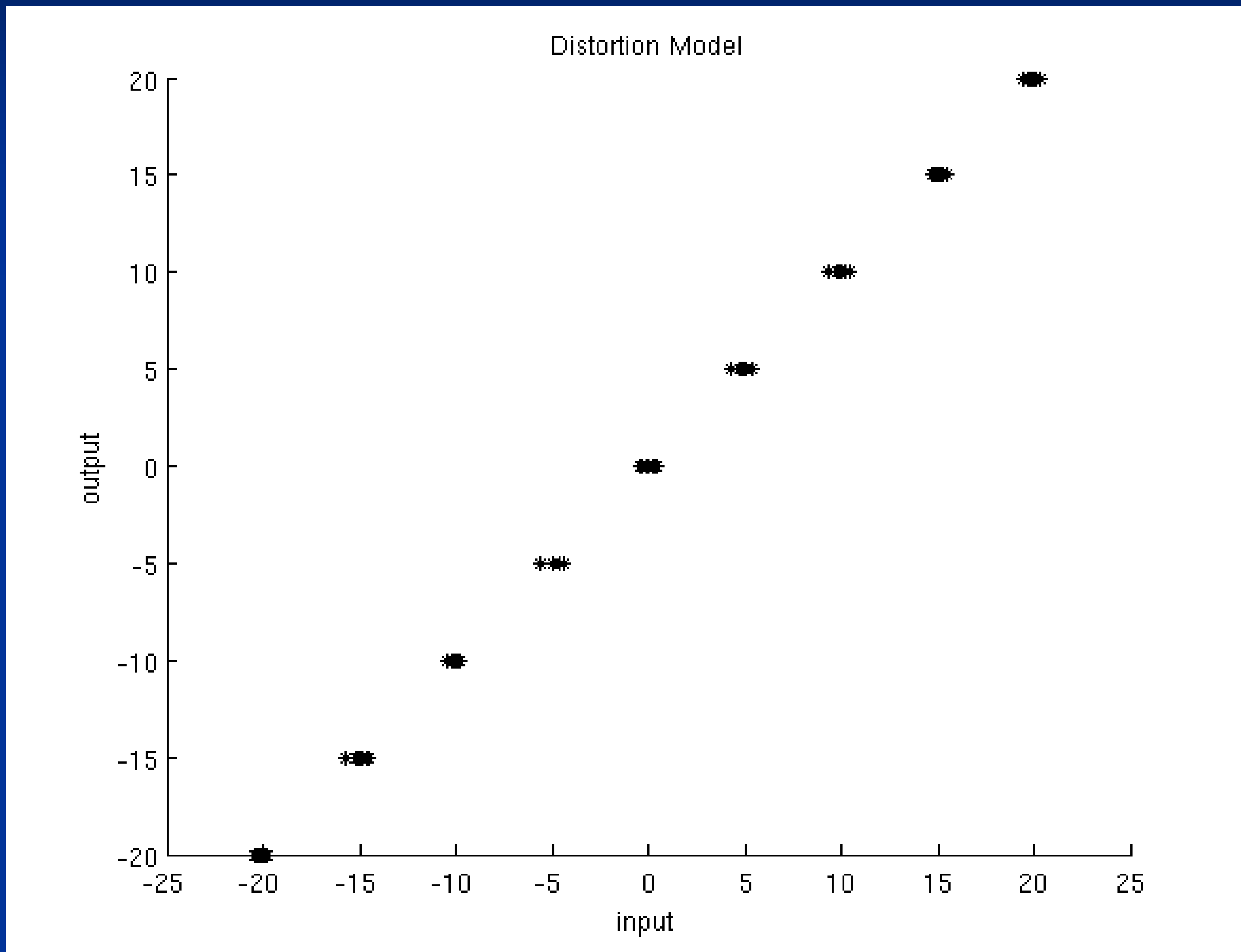
$$\hat{x} = \sum_i a_i(x)\phi_i$$

$x = \hat{x} + ´$ $\quad$ $´ \sim N(0, \sigma)$ $\quad$ noise

OR

$´ = S(\hat{x}, \alpha)$ $\quad$ transformations

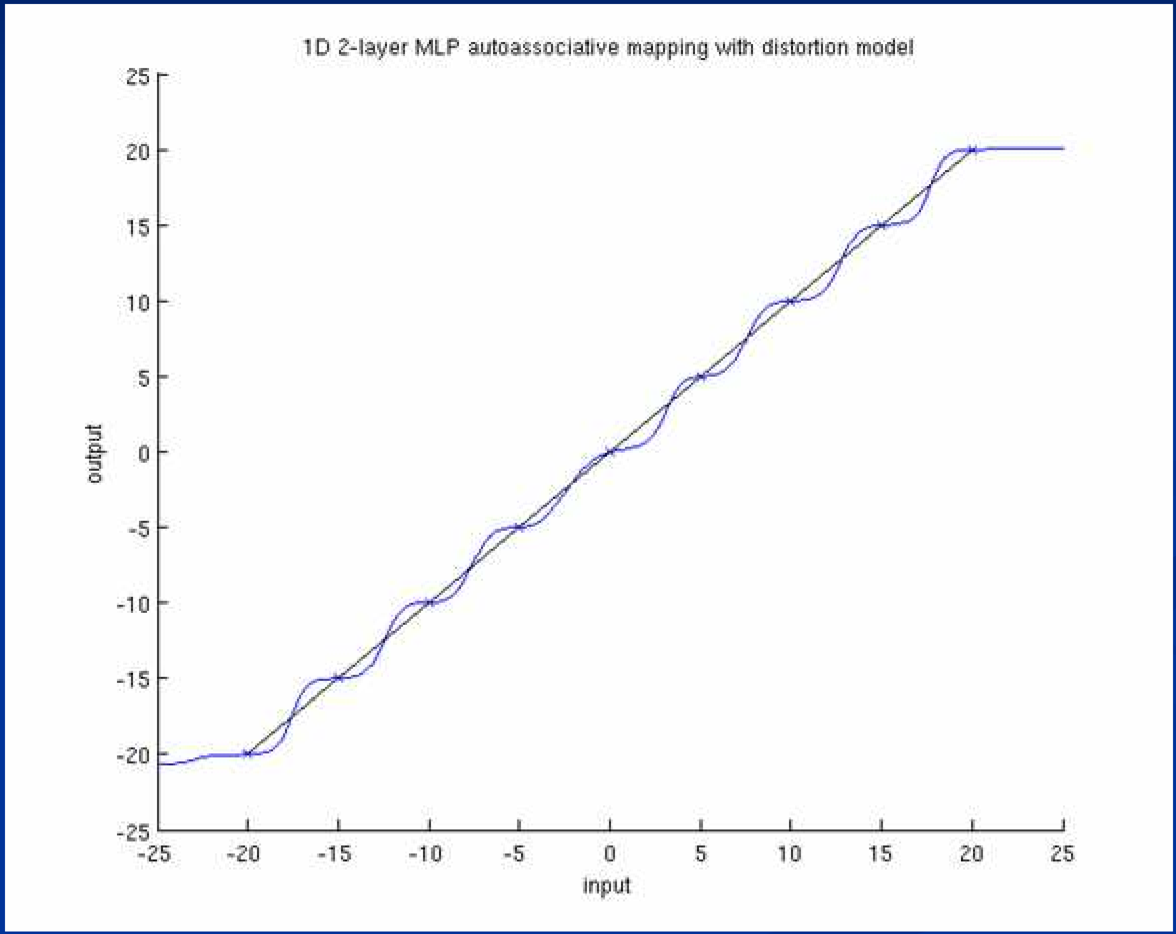
Distortion Model



Dynamics by Rolling FF networks

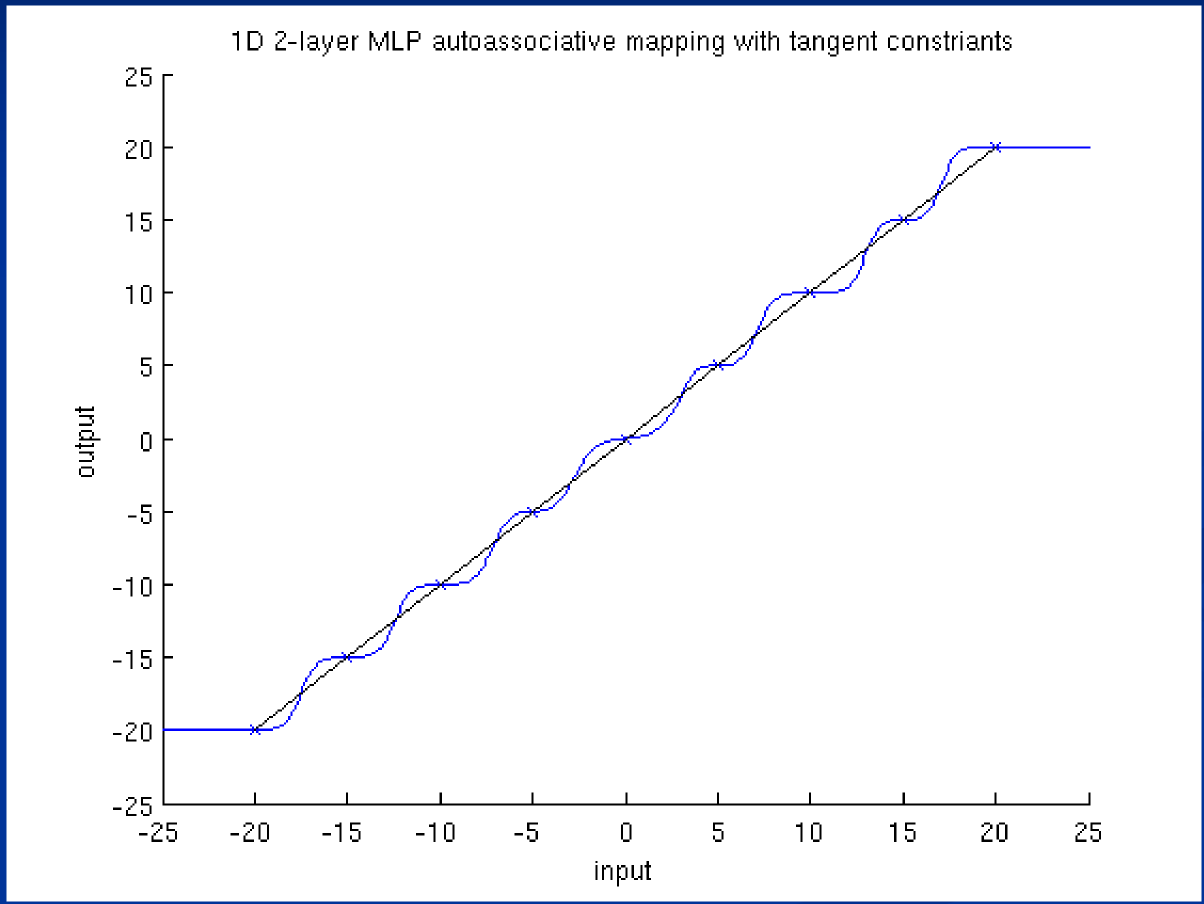
- Locations of convergence know as attractors
- Desire ‘flat areas’ in autoassociative function near the attractors

1D Two Layer Feedforward



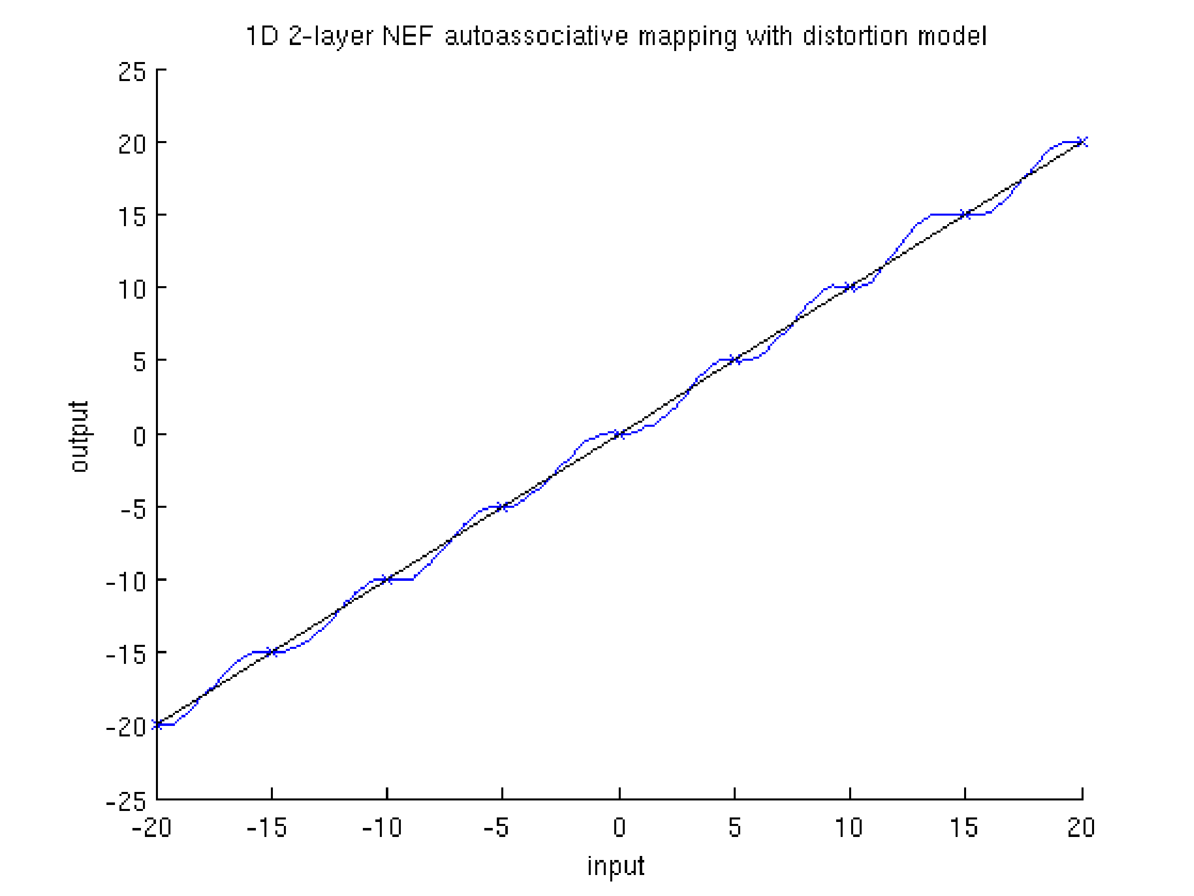
K = 30

1D Two Layer Feedforward Tangent Constraints



K = 30

1D NEF



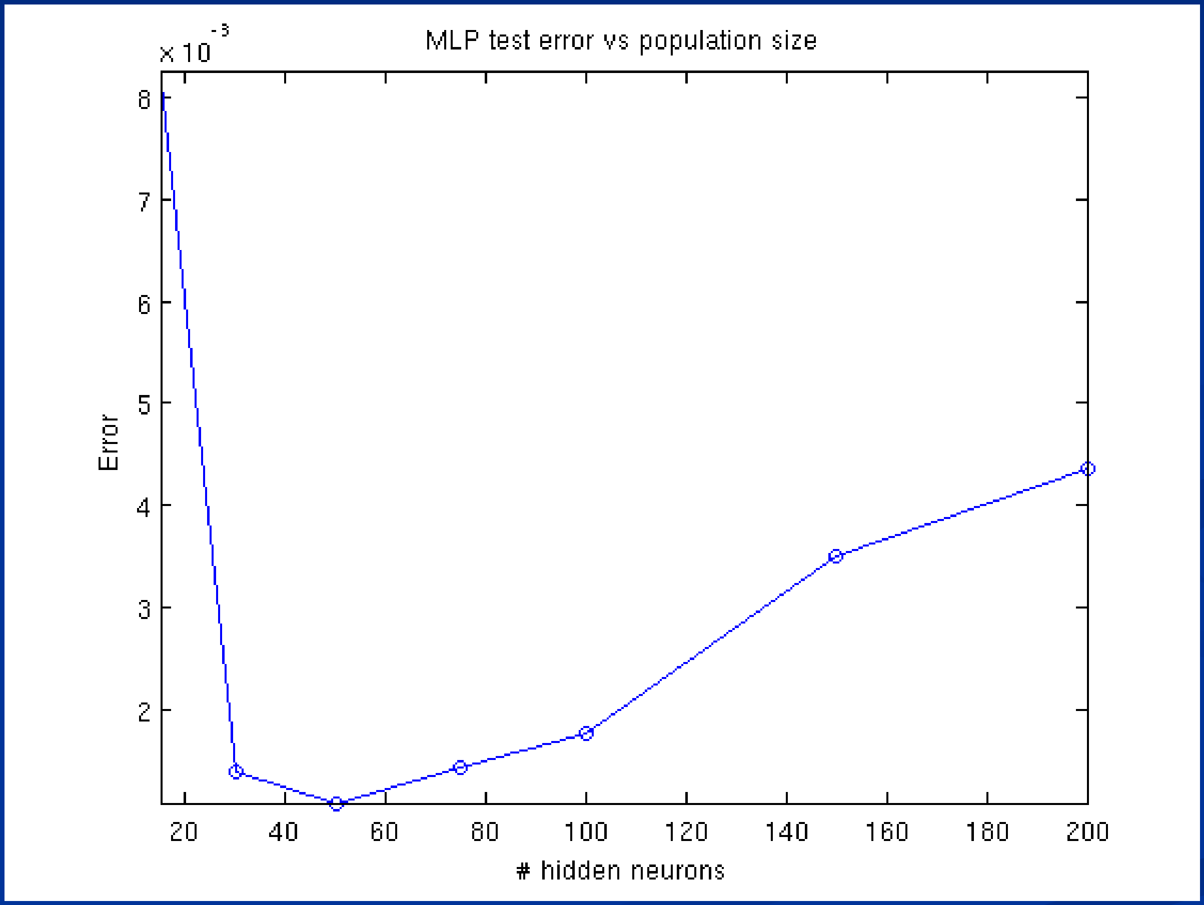
1D 2-layer NEF autoassociative mapping with distortion model
output
input
25
20
15
10
5
0
-5
-10
-15
-20
-25
-20
-15
-10
-5
0
5
10
15
20

K = 500

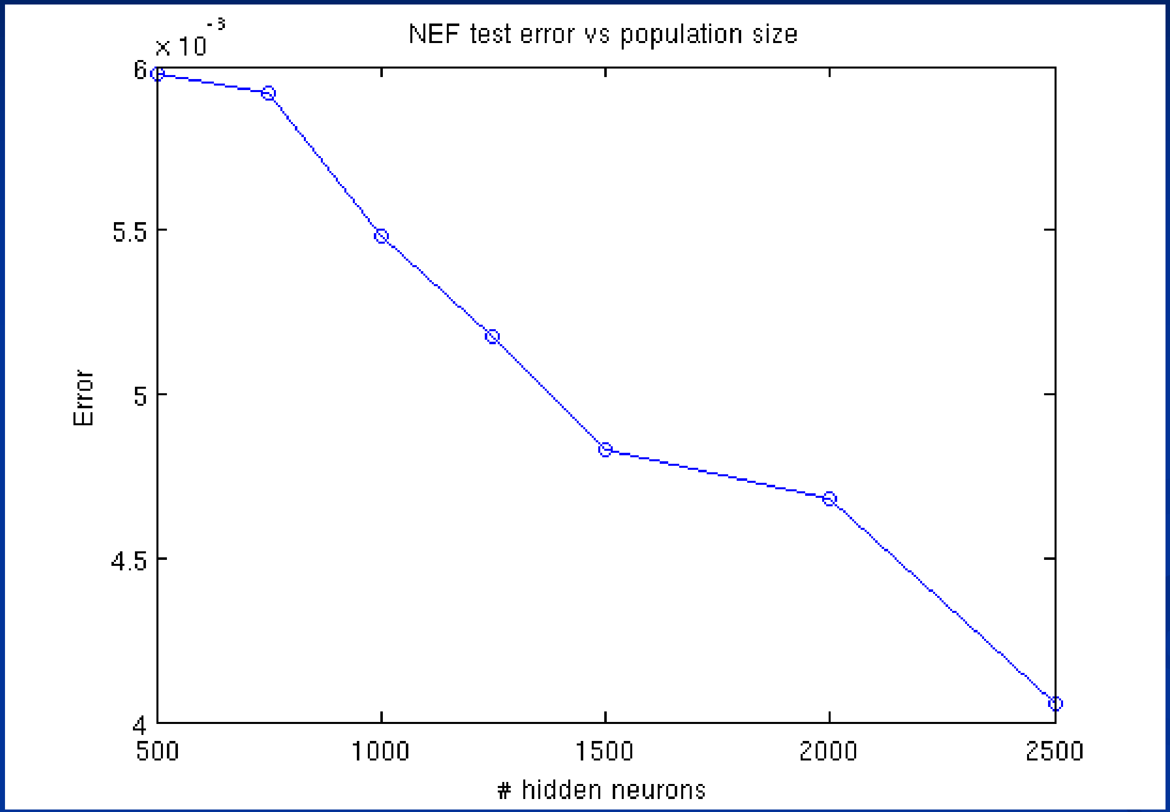
Overfitting

- Comes from the fact that higher complexity in model doesn't generalize onto unseen examples
- In biological terms, how many or what is the process behind recruiting # of neurons for representation?

Overfitting MLP



Overfitting NEF



- Why?
- 1. Due to the fact that less parameter OR
- 2. Due to the fact that hidden neuron's tuning curves are uniformly distributed
- 3. Will we see this behavior with sigmoids?

Autoassociation as Cleanup Memory

- Common task in cognitive models
  - Given a noisy version of something, output the clean original version
  - Declarative memory models, etc.
- Needed for symbolic reasoning too
  - VSAs: represent symbols as vectors
  - Combine vectors, extract original
  - Process introduces noise

Symbol Manipulation

- Represent a symbol tree:
  chase(dog,cat) $\quad chase \otimes verb + dog \otimes subj + cat \otimes obj$

  Extract the object:

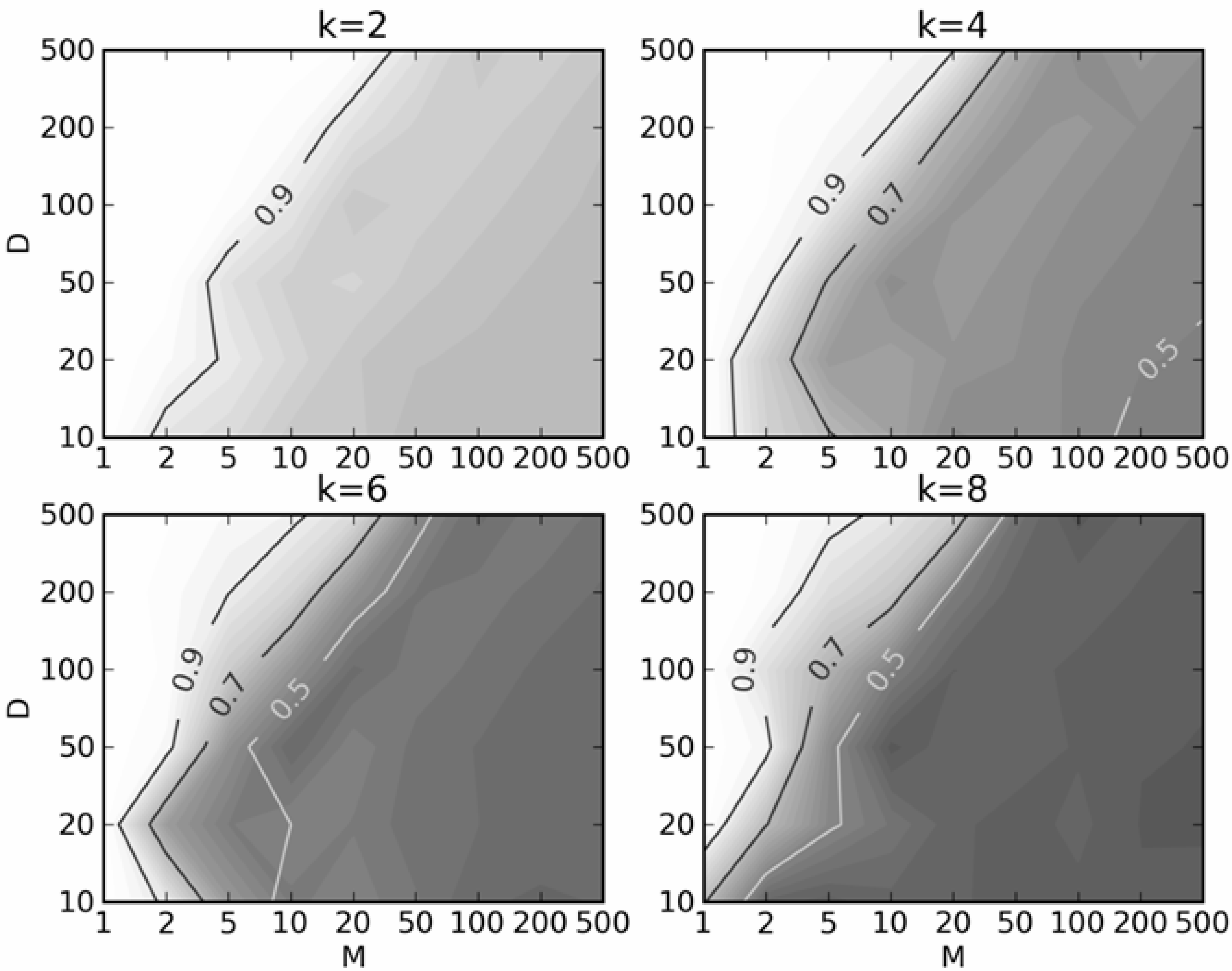
$$
\begin{aligned}
& (chase \otimes verb + dog \otimes subj + cat \otimes obj) \otimes \underline{obj} \\
= \; & chase \otimes verb \otimes \underline{obj} + dog \otimes subj \otimes \underline{obj} + cat \otimes obj \otimes \underline{obj} \\
\approx \; & cat + chase \otimes verb \otimes \underline{obj} + dog \otimes subj \otimes \underline{obj}
\end{aligned}
$$

- Result is a noisy version of cat

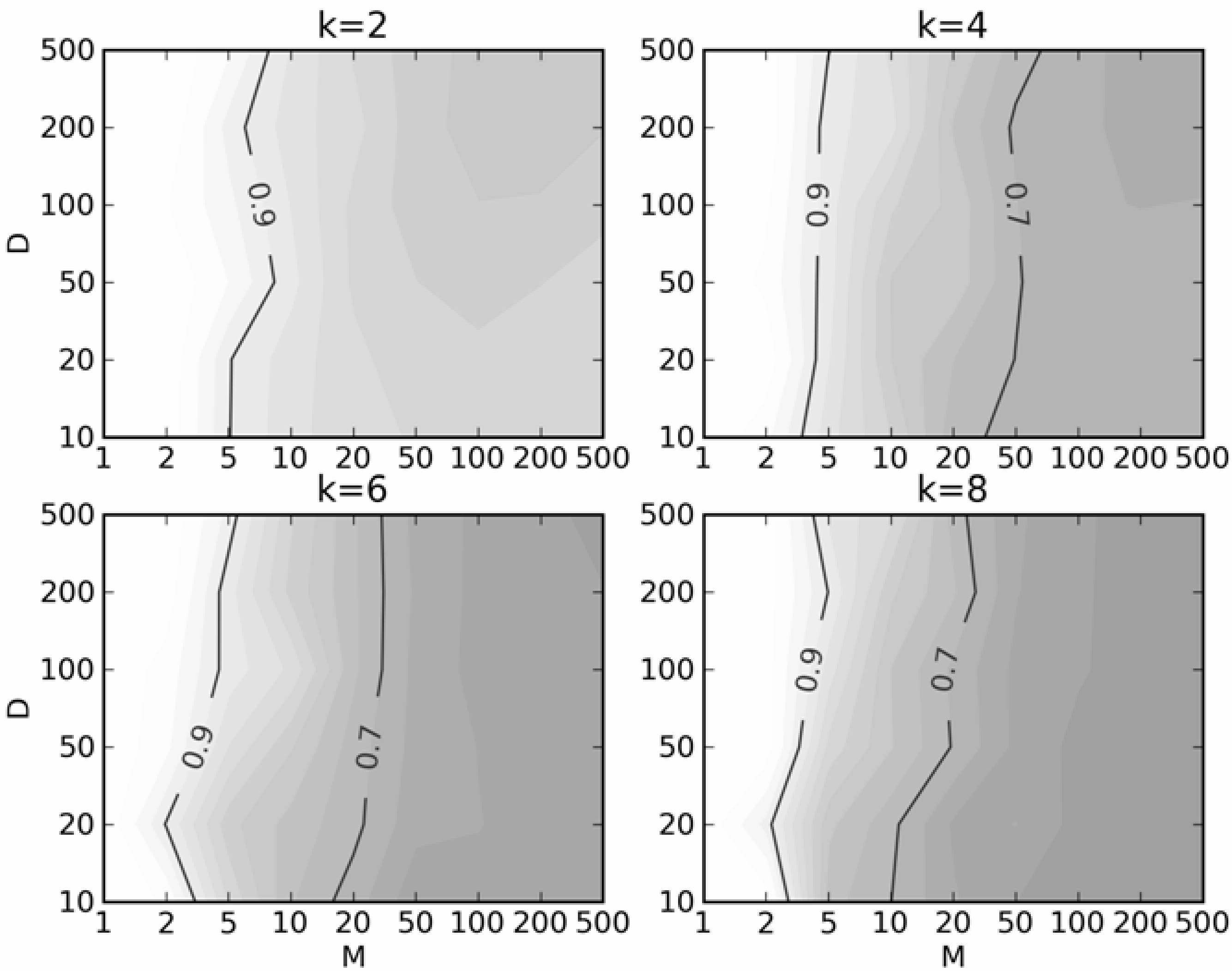
Signal and Noise

- Signal
  - Each symbol is a randomly chosen unit vector of high dimensionality D
  - Fixed set of M symbols
- Noise
  - Consists of k other randomly distributed unit vectors added to the original value
  - How large k can be limits the complexity of the symbol structures

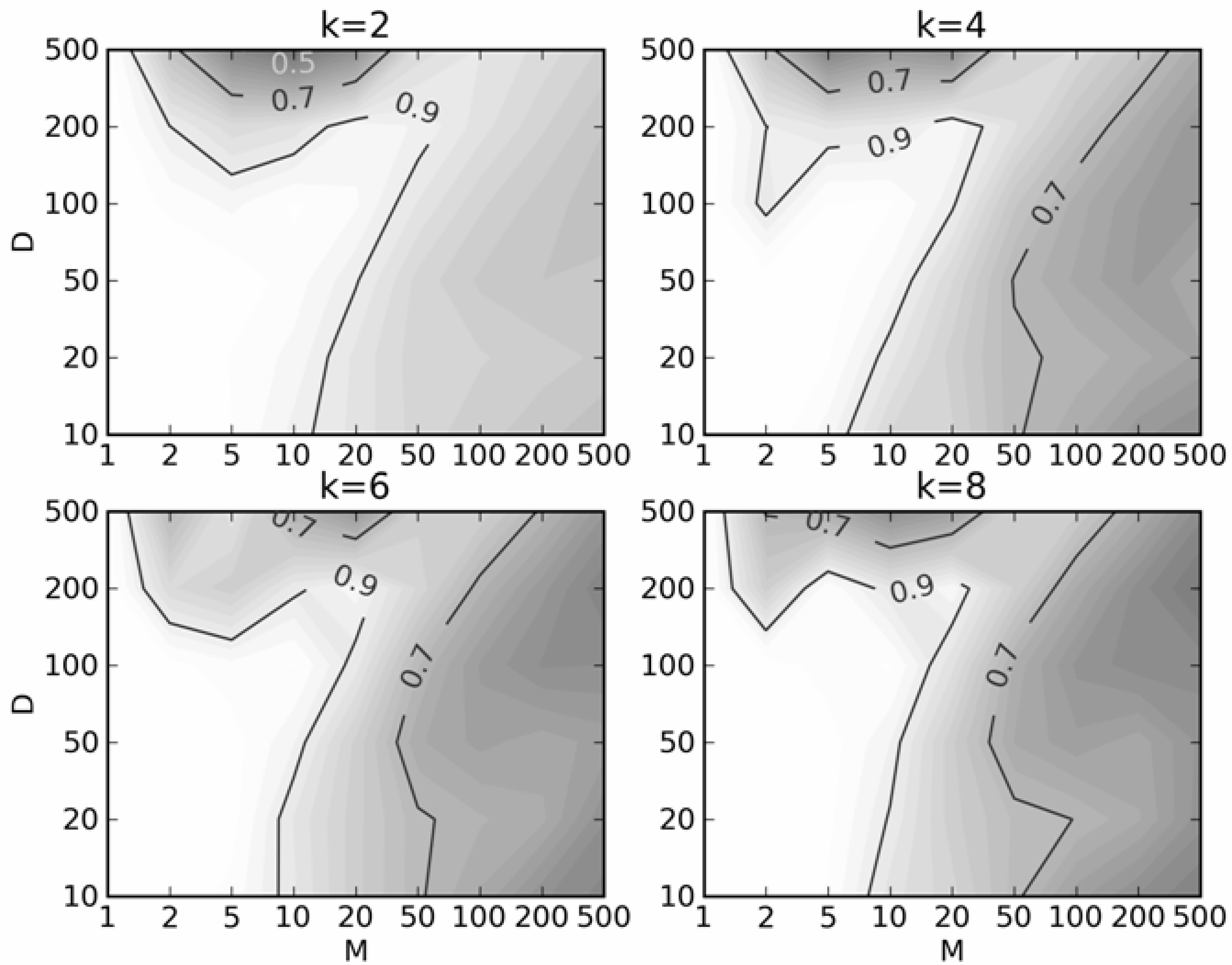
Linear Autoassociation



NEF Decoding



Multilayer Perceptron



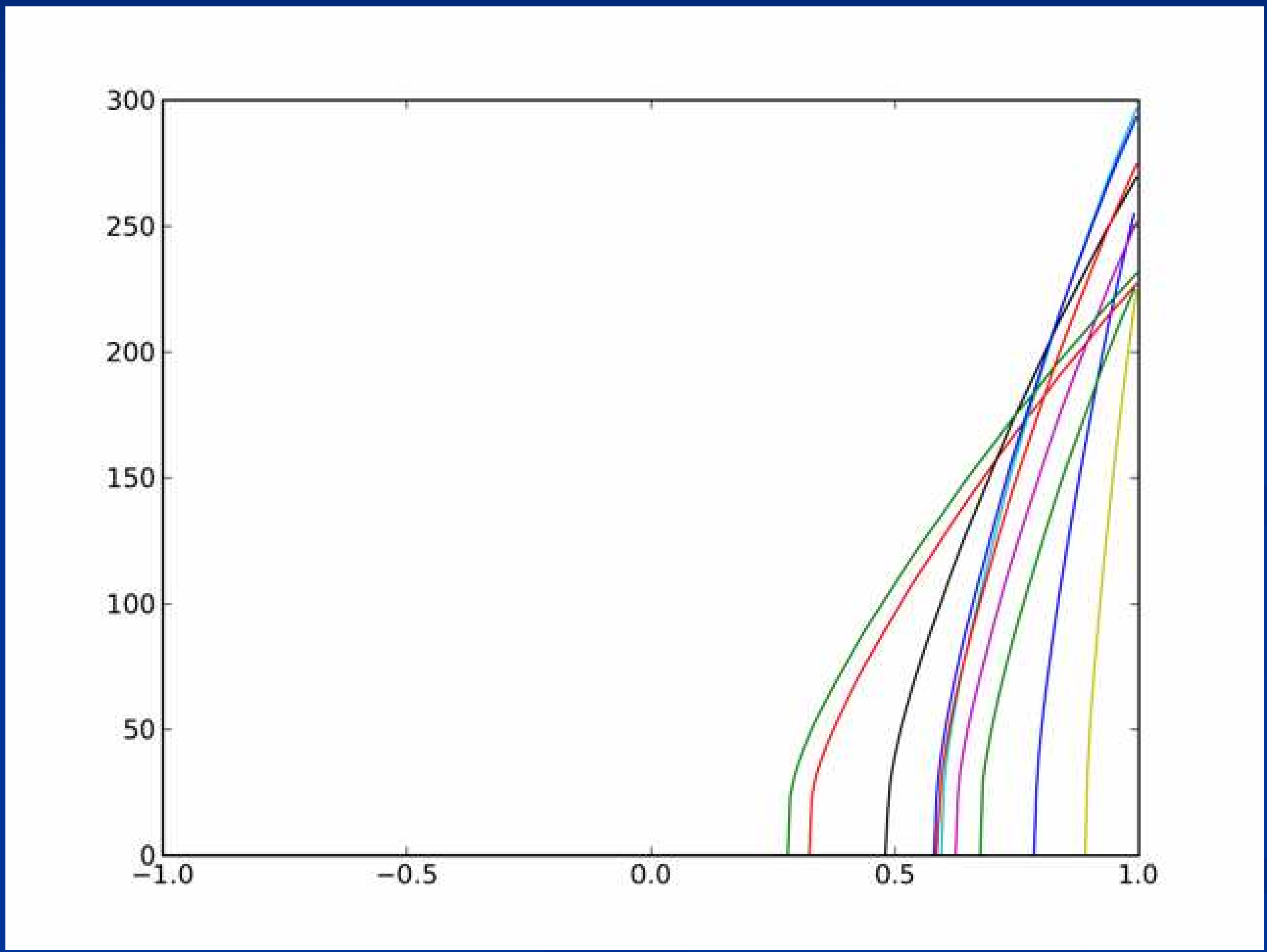
Scaling Problems

- Can't handle large M
  - Ideal VSAs handle M=10,000 at k=10 D=500
- MLP could probably do this, if we trained it for long enough
- Is there a better way?

Middle Layer

- There is no structure in the symbols
  - No better way of representing them
  - MLP worked best if H=M
- Align encoding vectors with symbols
  - ~10 neurons per symbol
  - What should their alpha and Jbias be?
    - We want it to be accurate for large values, but inaccurate for small values

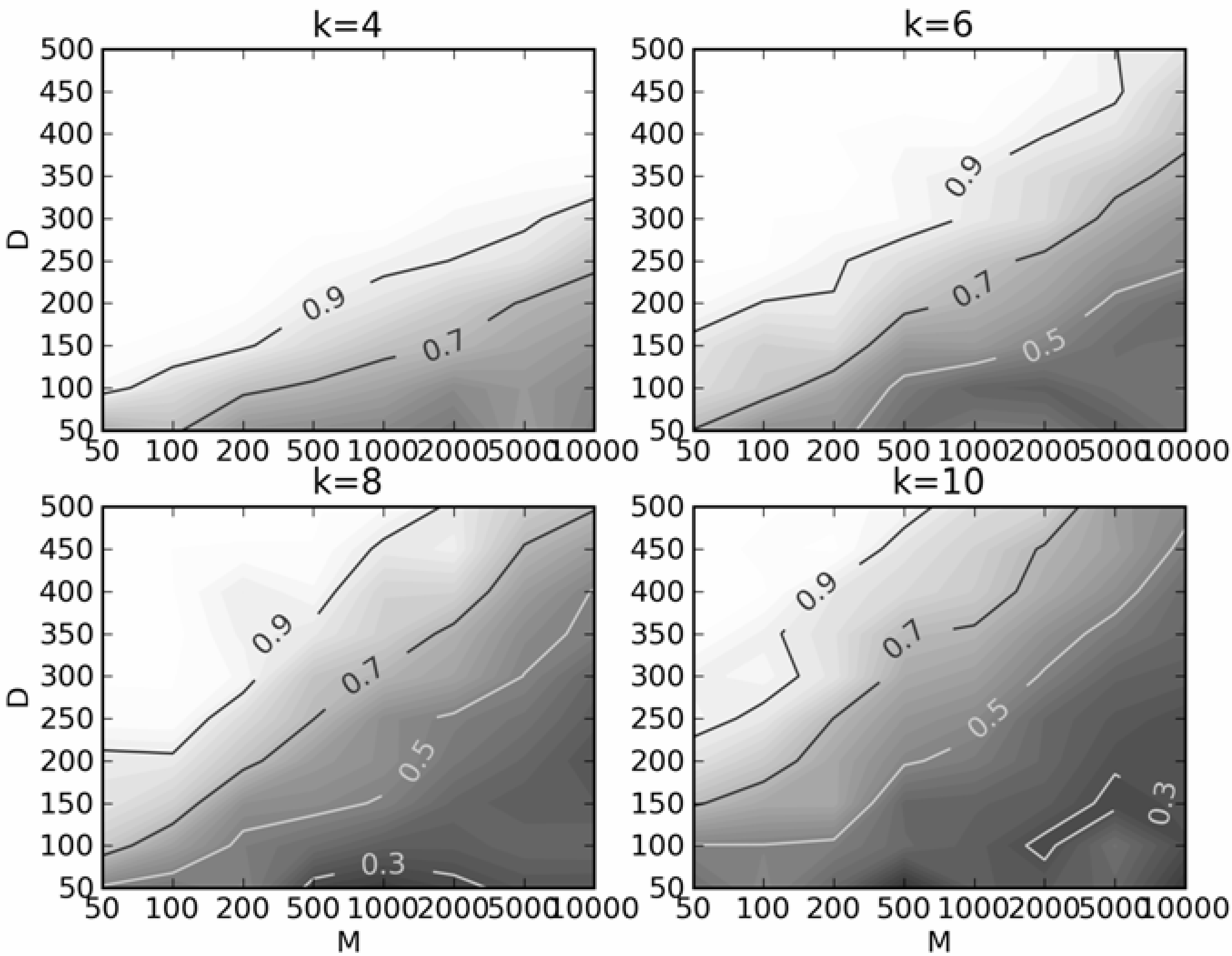
Middle Layer Encoding



Output

- Middle layer is bad at representing vectors that aren't in M
- After passing through this layer, signal should be cleaner
  - Can also apply a bit of function decoding that acts as a threshold (output 1 if $x>0.3$, otherwise 0)

Realistic Cleanup Memory



Cleanup Timing