MACHINE LEARNING
DEPARTMENT

Deep Learning

10-715 Fall 2015

Alexander Smola
alex@smola.org

Office hours - after class in my office

Marianas Labs

Carnegie Mellon University

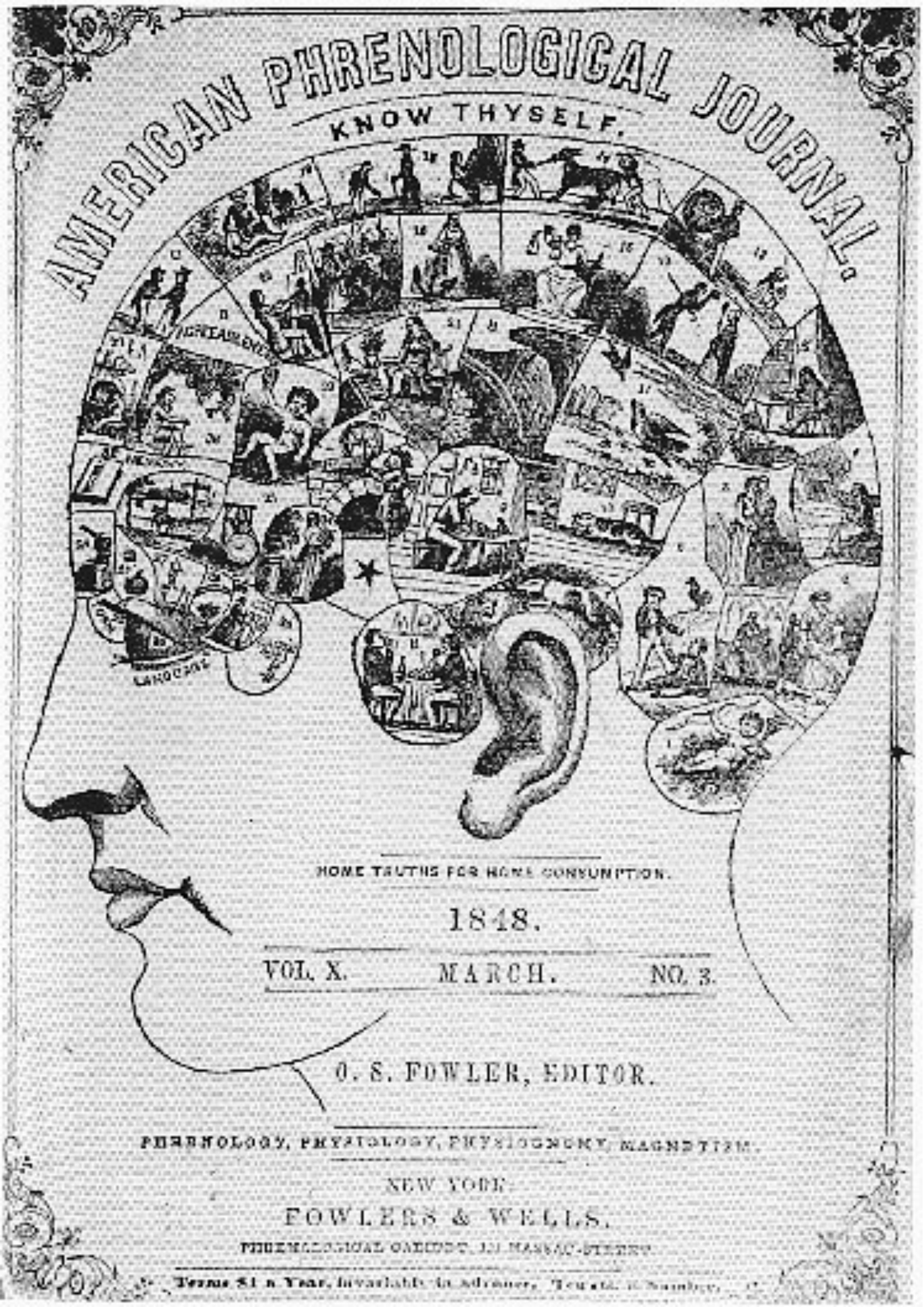
Outline

- **Basics**
  - Perceptron (and convergence rule)
  - Stochastic gradient descent
  - Loss functions and objectives
- **Deep Networks**
  - Layers and Invariances
  - Advanced objectives
- **State and Structure**
  - Optimization
  - Autoregressive Models & Hidden State
  - Toolkits

1. Perceptron

Rosenblatt

Widom



Neurons and Learning

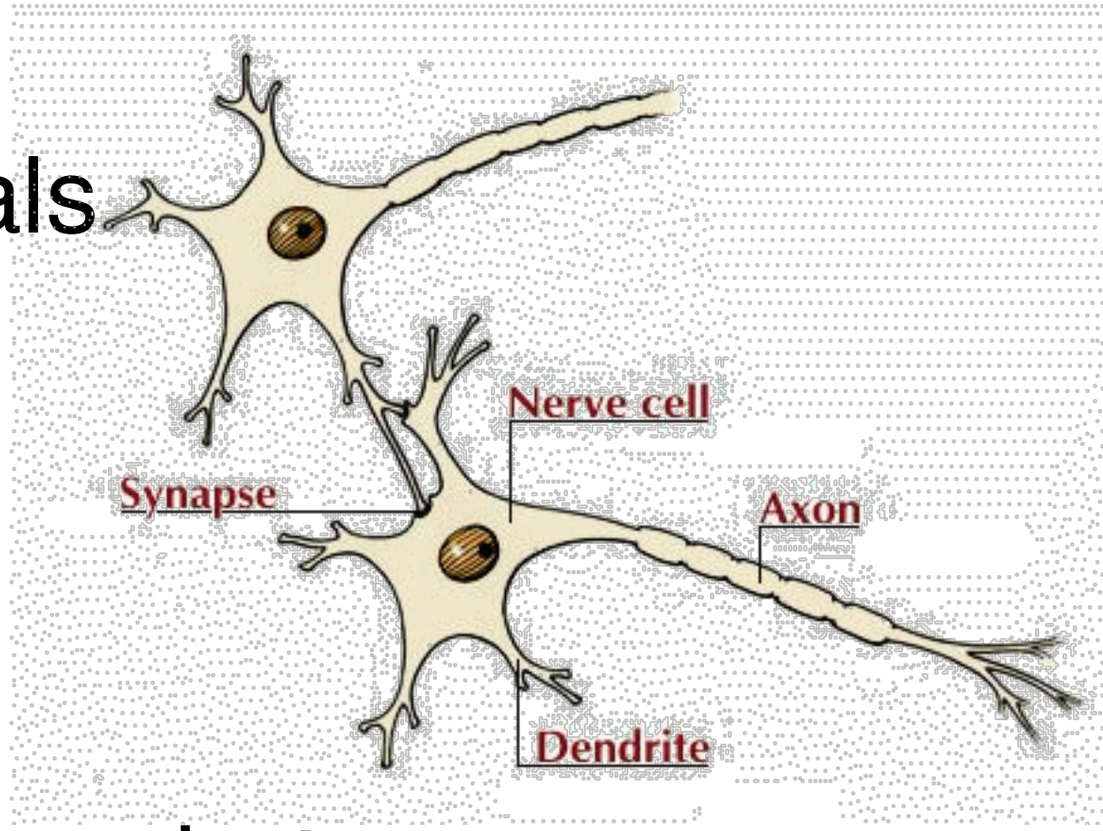
Carnegie Mellon University

Biology and Learning

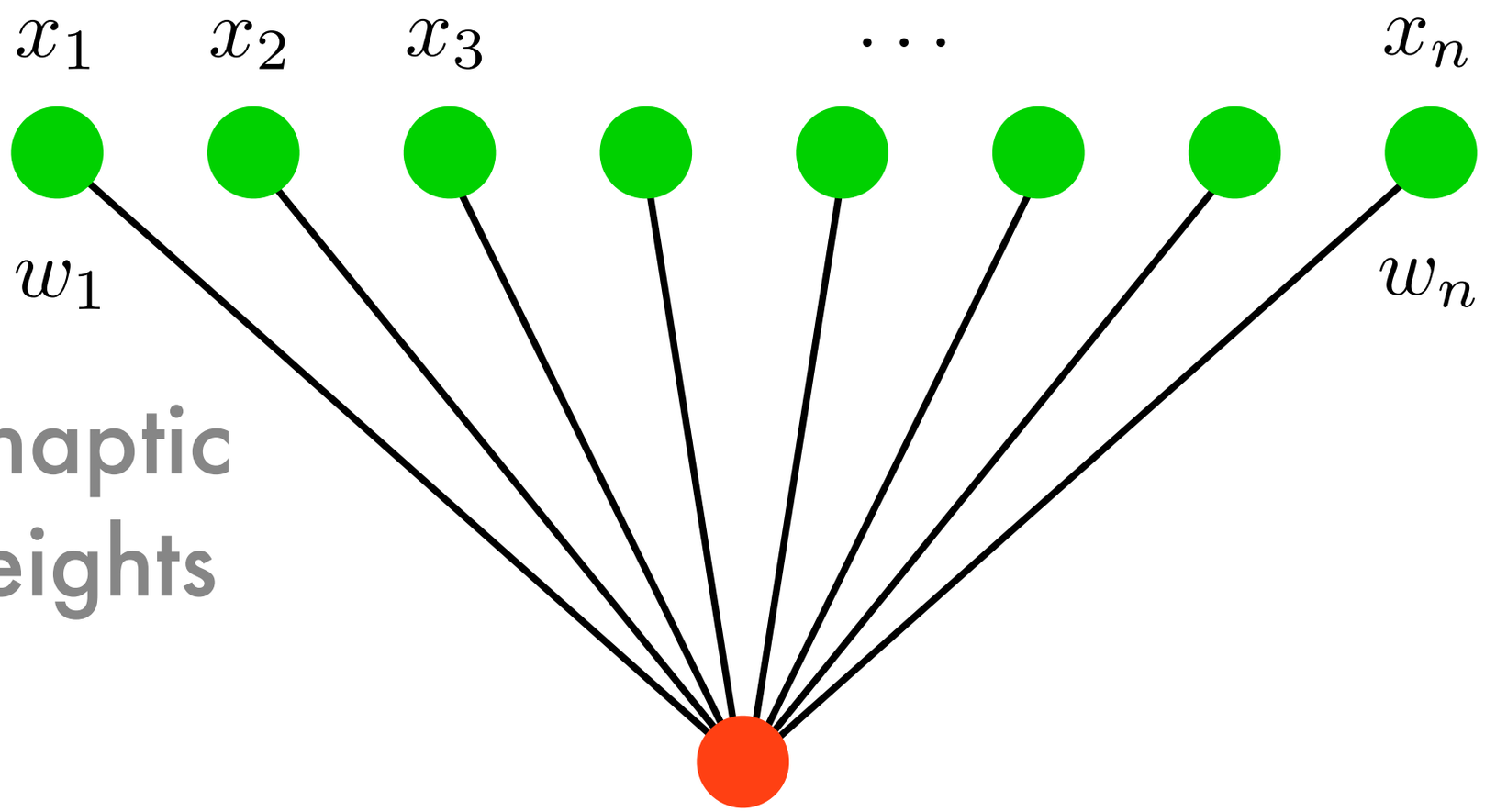
- **Basic Idea**
  - Good behavior should be rewarded, bad behavior punished (or not rewarded). This improves system fitness.
  - Killing a sabertooth tiger should be rewarded ...
  - Correlated events should be combined.
  - Pavlov's salivating dog.
- **Training mechanisms**
  - Behavioral modification of individuals (learning) Successful behavior is rewarded (e.g. food).
  - Hard-coded behavior in the genes (instinct) The wrongly coded animal does not reproduce.

Neurons

- **Soma** (CPU)
  Cell body - combines signals
- **Dendrite** (input bus)
  Combines the inputs from several other nerve cells
- **Synapse** (interface)
  Interface and parameter store between neurons
- **Axon** (cable)
  May be up to 1m long and will transport the activation signal to neurons at different locations



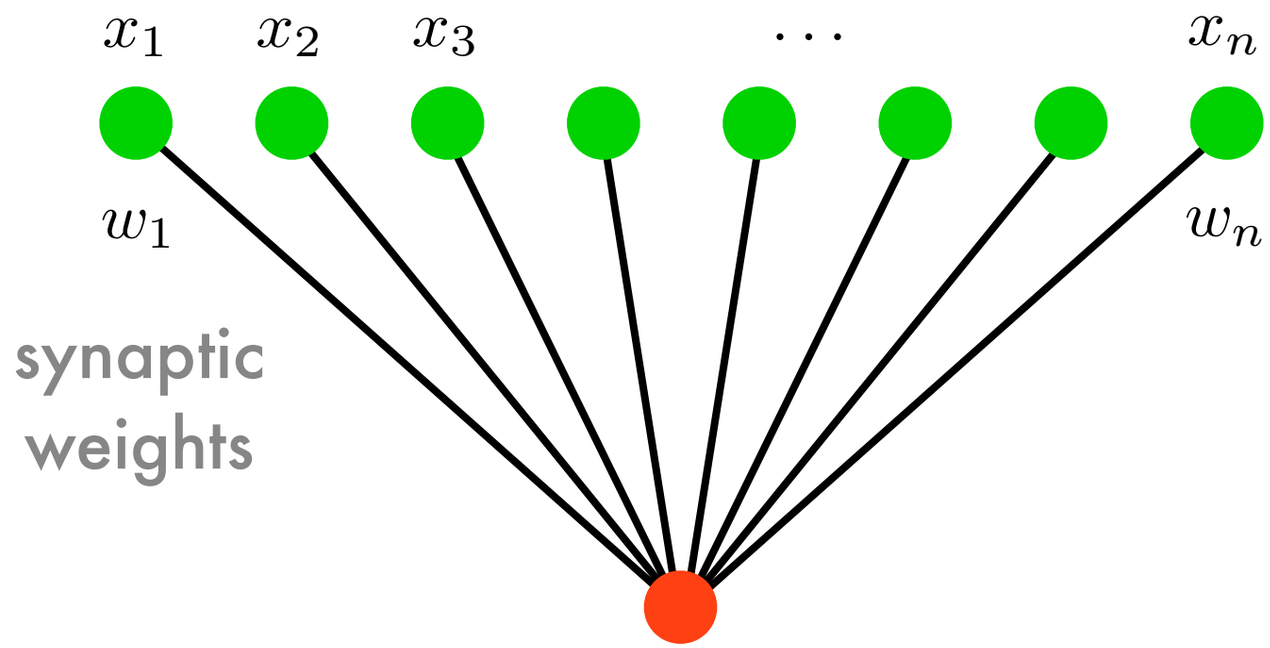
Neurons



$$f(x) = \sum_i w_i x_i = \langle w, x \rangle$$

Perceptron

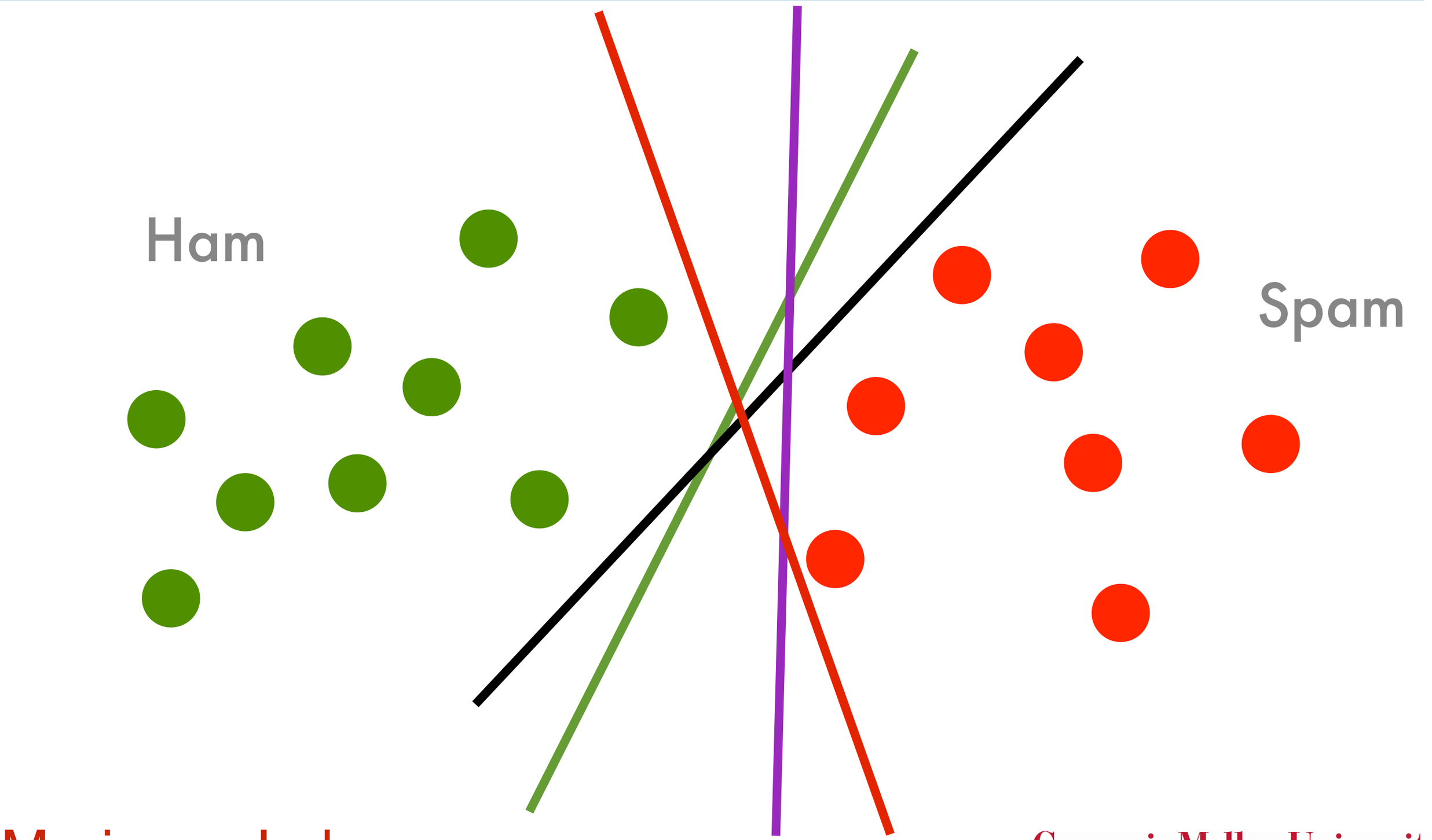
- Weighted linear combination
- Nonlinear decision function
- Linear offset (bias)



$$f(x) = \sigma\left(\langle w, x \rangle + b\right)$$

- Linear separating hyperplanes (spam/ham, novel/typical, click/no click)
- Learning - Estimating the parameters w and b

Perceptron



Perceptron Algorithm

The Perceptron

initialize $w = 0$ and $b = 0$
repeat
  if $y_i \left[\langle w, x_i \rangle + b\right] \leq 0$ **then**
    $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$
  end if
until all classified correctly

- Nothing happens if classified correctly
- Weight vector is linear combination $w = \sum_{i \in I} y_i x_i$
- Classifier is linear combination of inner products

$$f(x) = \sum_{i \in I} y_i \langle x_i, x \rangle + b$$

Convergence Theorem

- If there exists some $(w^*, b^*)$with unit length and

$$y_i \left[\langle x_i, w^* \rangle + b^*\right] \geq \rho \text{ for all } i$$

then the perceptron converges to a linear separator after a number of steps bounded by

$$\left(b^{*2} + 1\right)\left(r^2 + 1\right)\rho^{-2} \text{ where } \|x_i\| \leq r$$

- Dimensionality independent
- Order independent (i.e. also worst case)
- Scales with ‘difficulty’ of problem

Proof

Starting Point

We start from $w_1 = 0$ and $b_1 = 0$.

Step 1: Bound on the increase of alignment

Denote by w_i the value of w at step i (analogously b_i).

Alignment: $\langle (w_i, b_i), (w^*, b^*) \rangle$

For error in observation (x_i, y_i) we get

$$\begin{aligned}
&\langle (w_{j+1}, b_{j+1}) \cdot (w^*, b^*) \rangle \\
&= \langle [(w_j, b_j) + y_i (x_i, 1)], (w^*, b^*) \rangle \\
&= \langle (w_j, b_j), (w^*, b^*) \rangle + y_i \langle (x_i, 1) \cdot (w^*, b^*) \rangle \\
&\geq \langle (w_j, b_j), (w^*, b^*) \rangle + \rho \\
&\geq j\rho.
\end{aligned}$$

Alignment increases with number of errors.

Proof

Step 2: Cauchy-Schwartz for the Dot Product

$$\begin{aligned}\langle (w_{j+1}, b_{j+1}) \cdot (w^*, b^*) \rangle &\leq \|(w_{j+1}, b_{j+1})\| \, \|(w^*, b^*)\| \\ &= \sqrt{1 + (b^*)^2} \|(w_{j+1}, b_{j+1})\|\end{aligned}$$

Step 3: Upper Bound on $\|(w_j, b_j)\|$

If we make a mistake we have

$$\begin{aligned}\|(w_{j+1}, b_{j+1})\|^2 &= \|(w_j, b_j) + y_i(x_i, 1)\|^2 \\ &= \|(w_j, b_j)\|^2 + 2y_i\langle (x_i, 1), (w_j, b_j) \rangle + \|(x_i, 1)\|^2 \\ &\leq \|(w_j, b_j)\|^2 + \|(x_i, 1)\|^2 \\ &\leq j(R^2 + 1).\end{aligned}$$

Step 4: Combination of first three steps

$$j\rho \leq \sqrt{1 + (b^*)^2} \|(w_{j+1}, b_{j+1})\| \leq \sqrt{j(R^2 + 1)((b^*)^2 + 1)}$$

Solving for j proves the theorem.

Consequences

- Only need to store errors.
  This gives a compression bound for perceptron.
- This is stochastic gradient descent on hinge loss

$$l(x_i, y_i, w, b) = \max\left(0, 1 - y_i\left[\langle w, x_i\rangle + b\right]\right)$$

- Fails miserably with noisy data

do NOT train your avatar with perceptrons

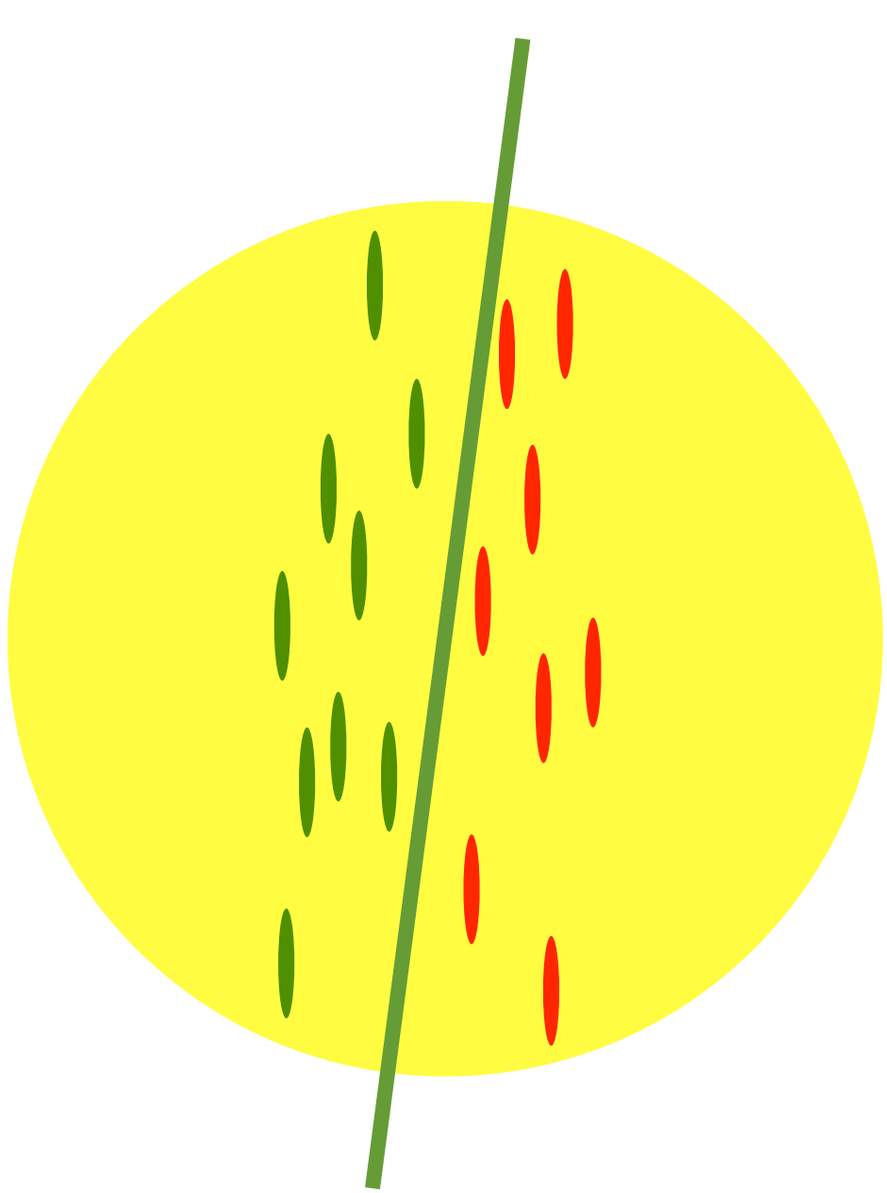
Stochastic Gradient with Hinge Loss

$$l(x_i, y_i, w, b) = \max\left(0, 1 - y_i\left[\langle w, x_i\rangle + b\right]\right)$$

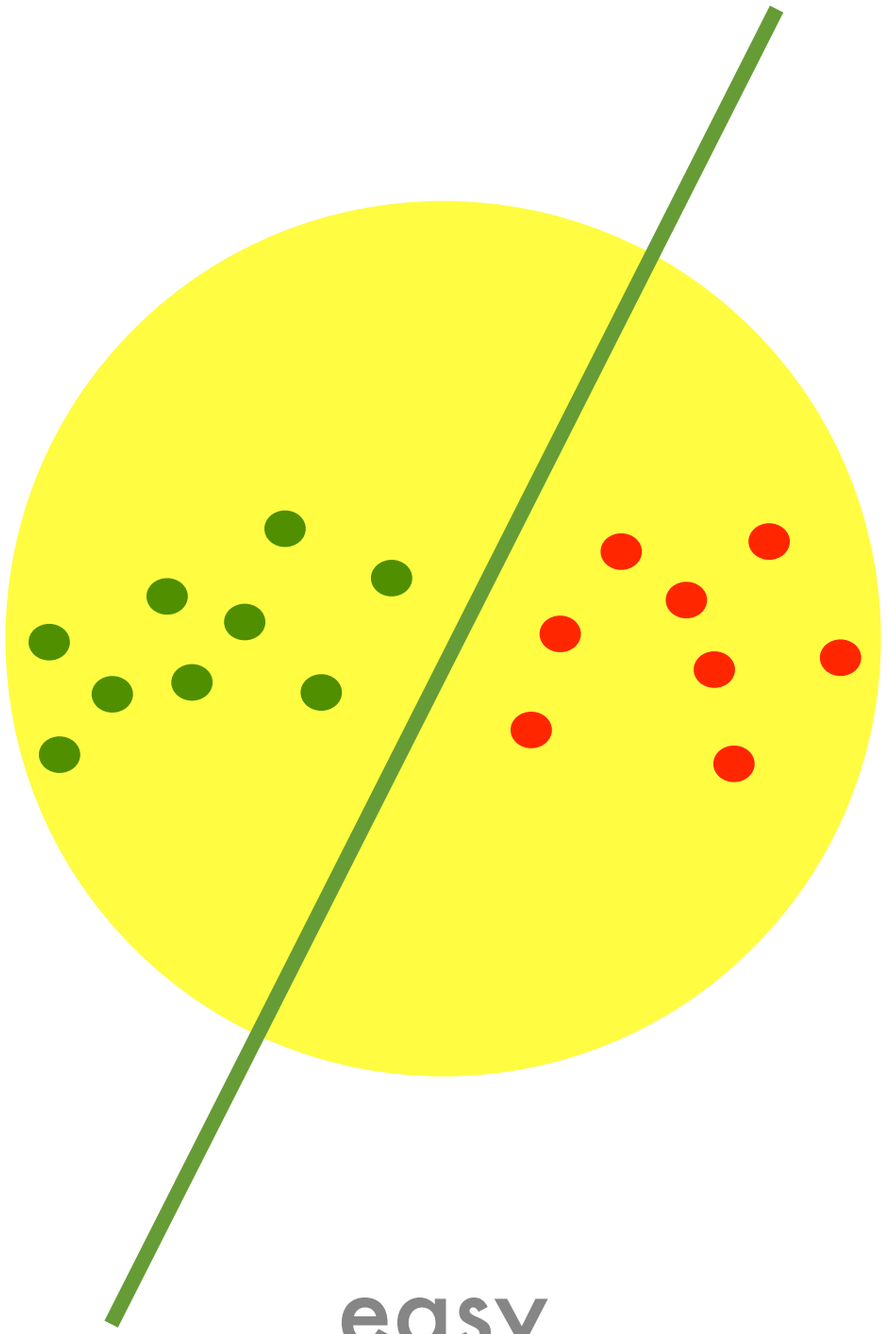
Marianas Labs

Carnegie Mellon University

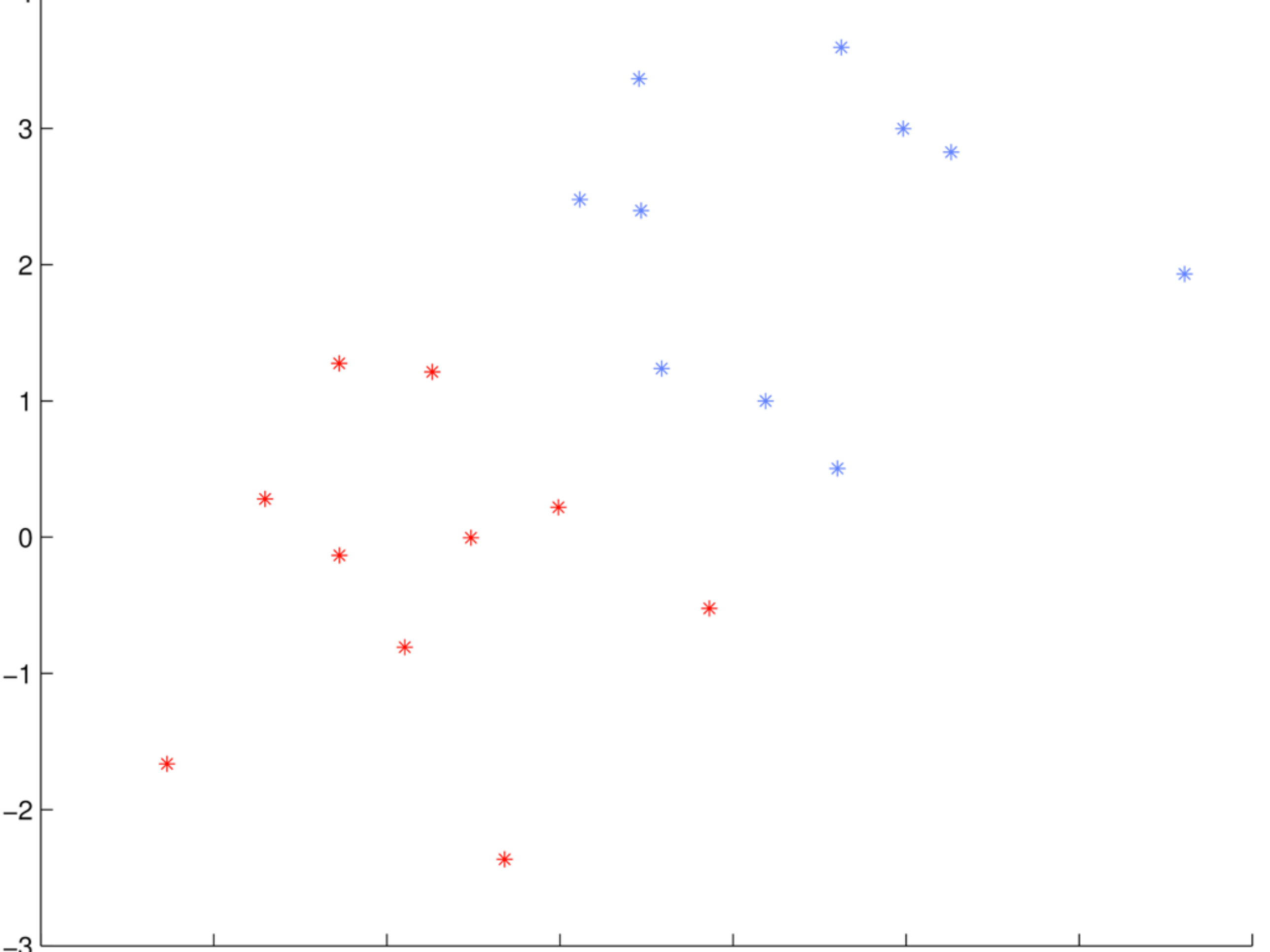
Hardness



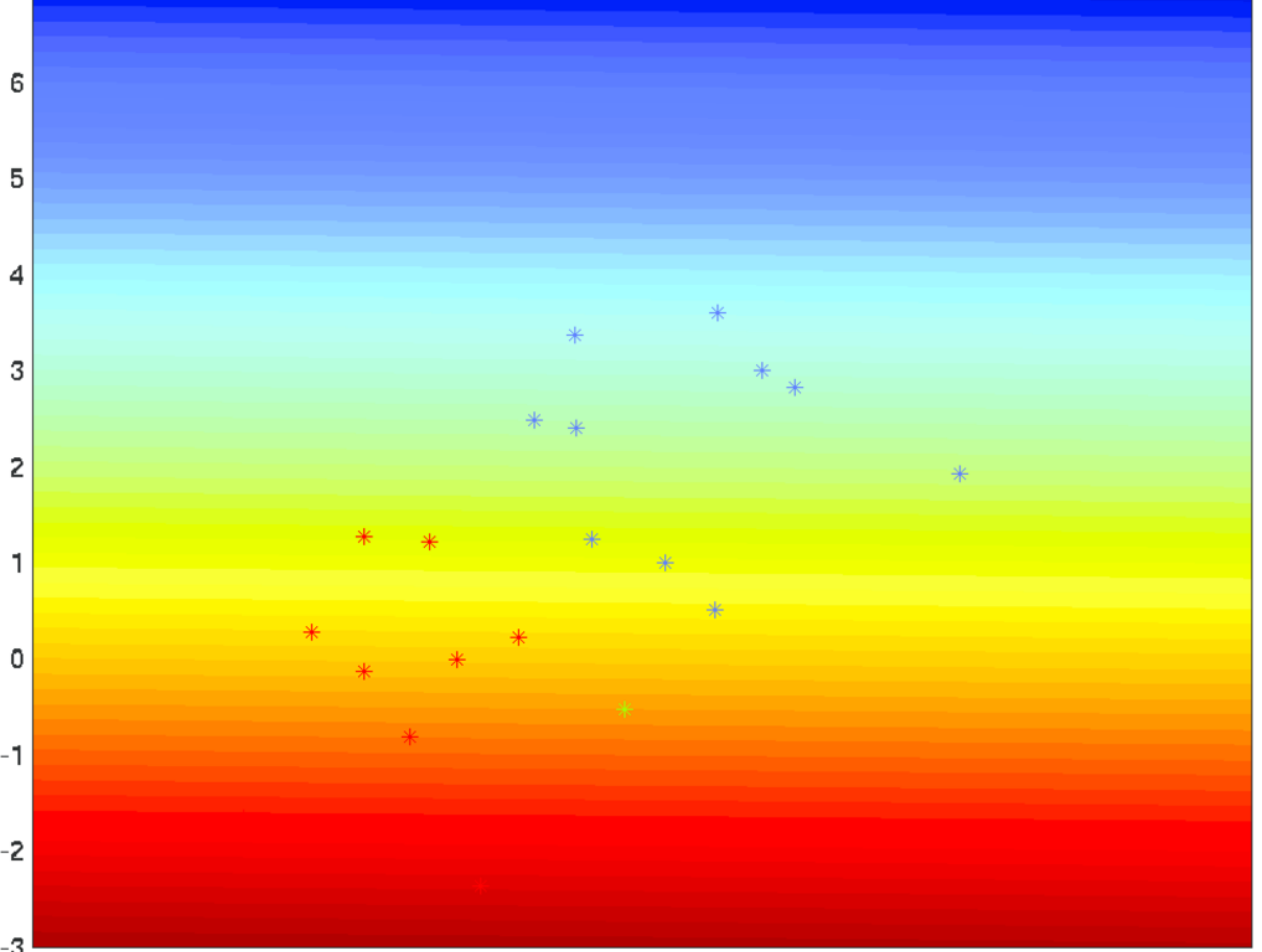
hard

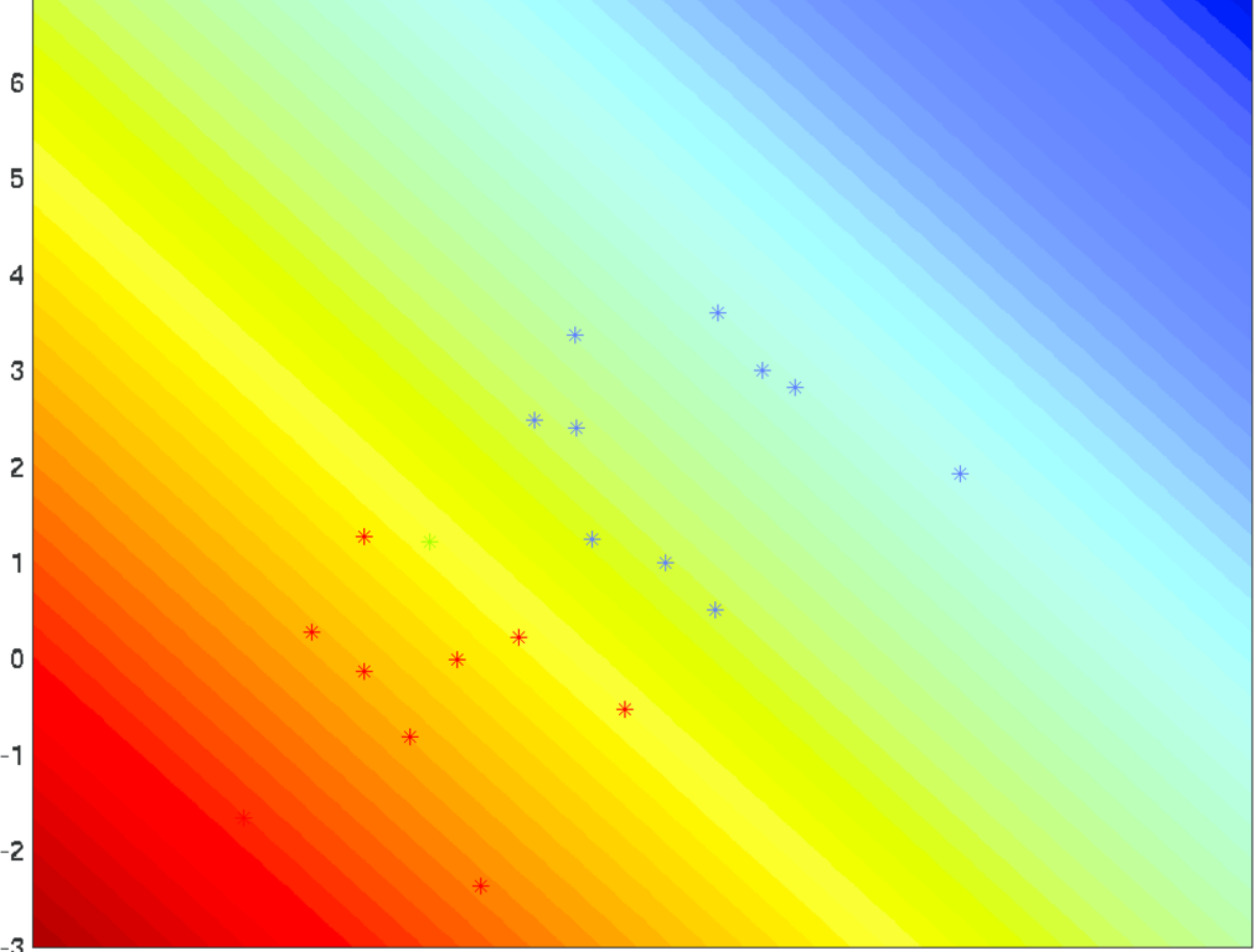


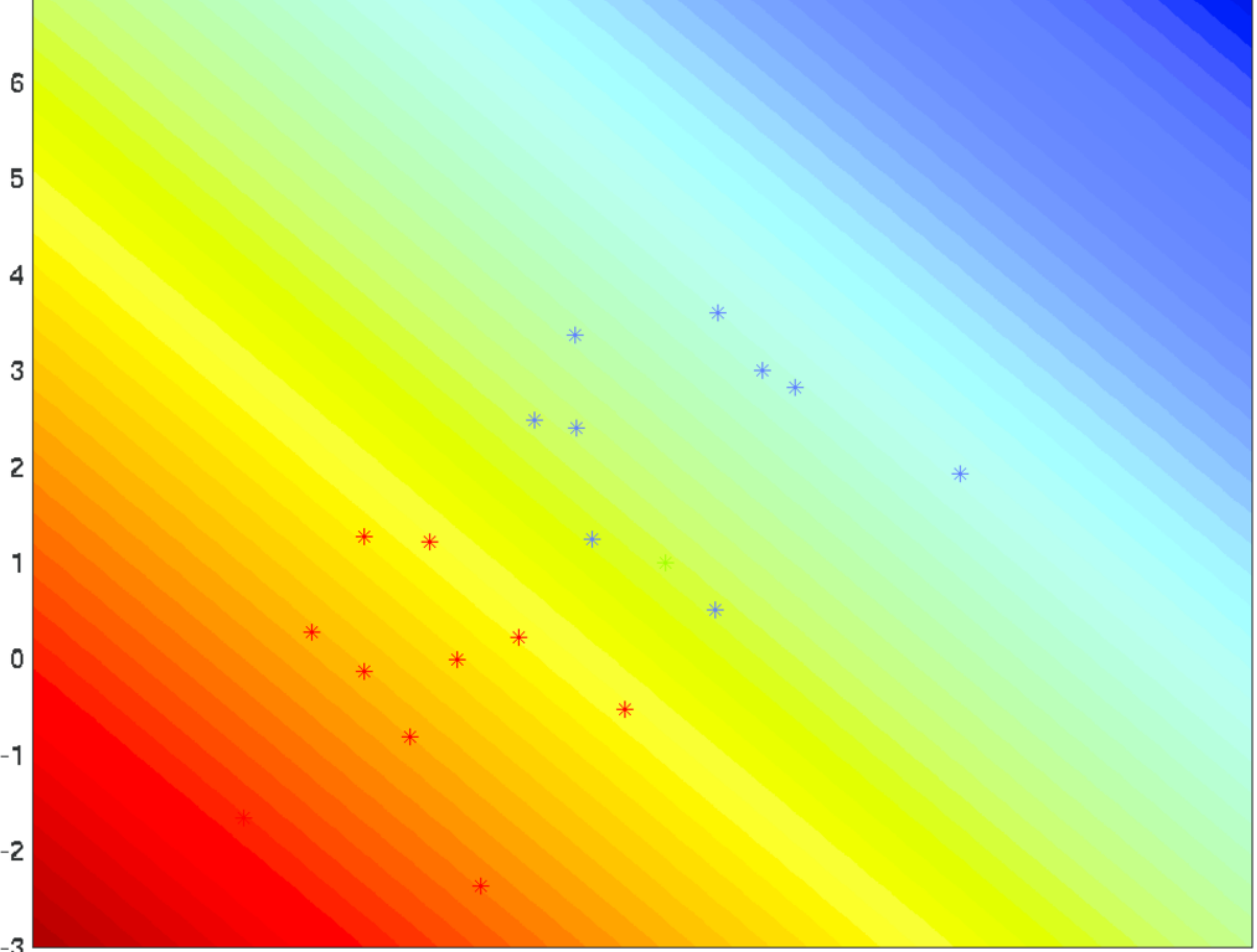
easy


3
2
1
0
−1
−2
−3

6
5
4
3
2
1
0
-1
-2
-3

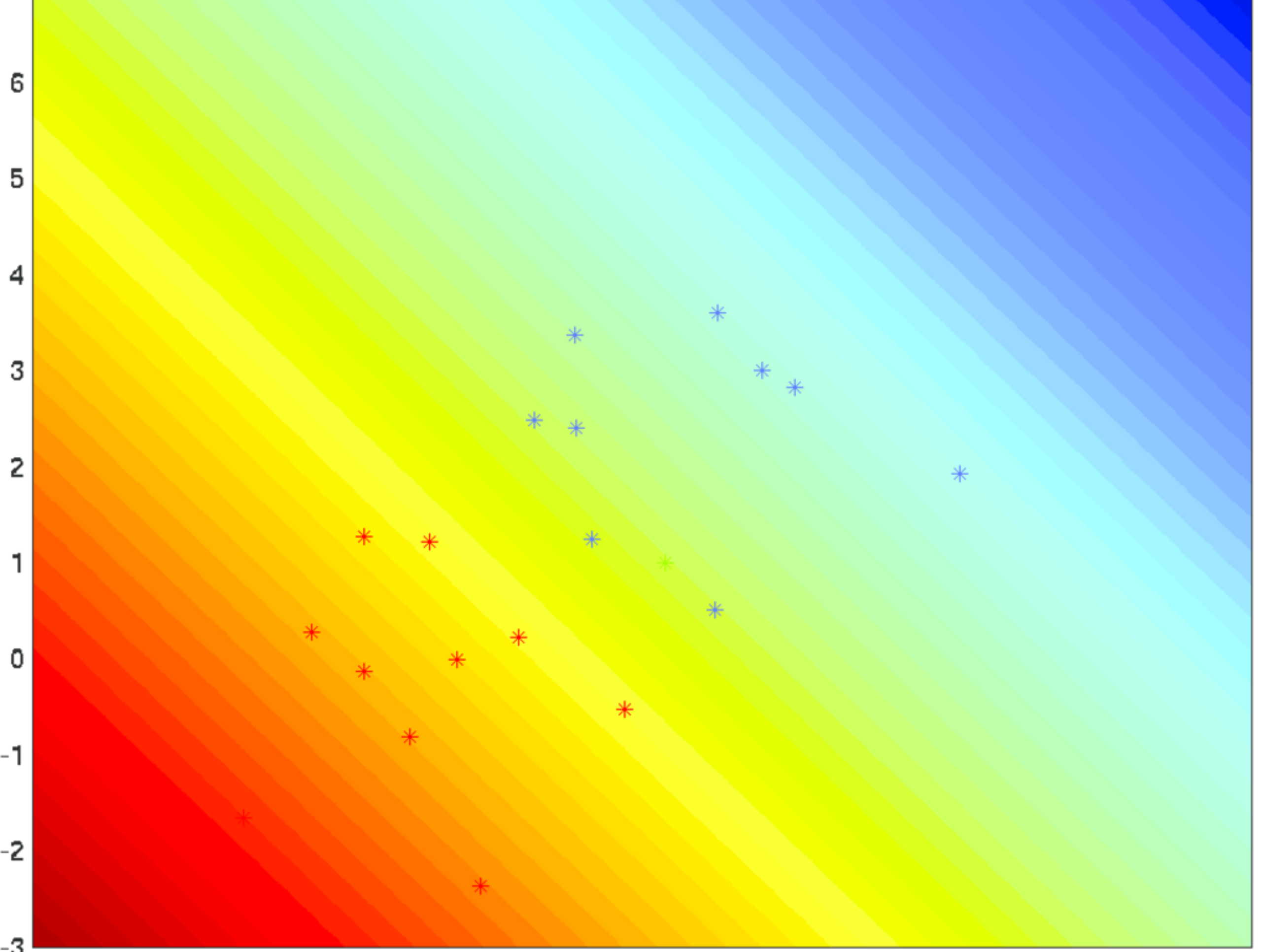

6
5
4
3
2
1
0
-1
-2
-3


6
5
4
3
2
1
0
-1
-2
-3

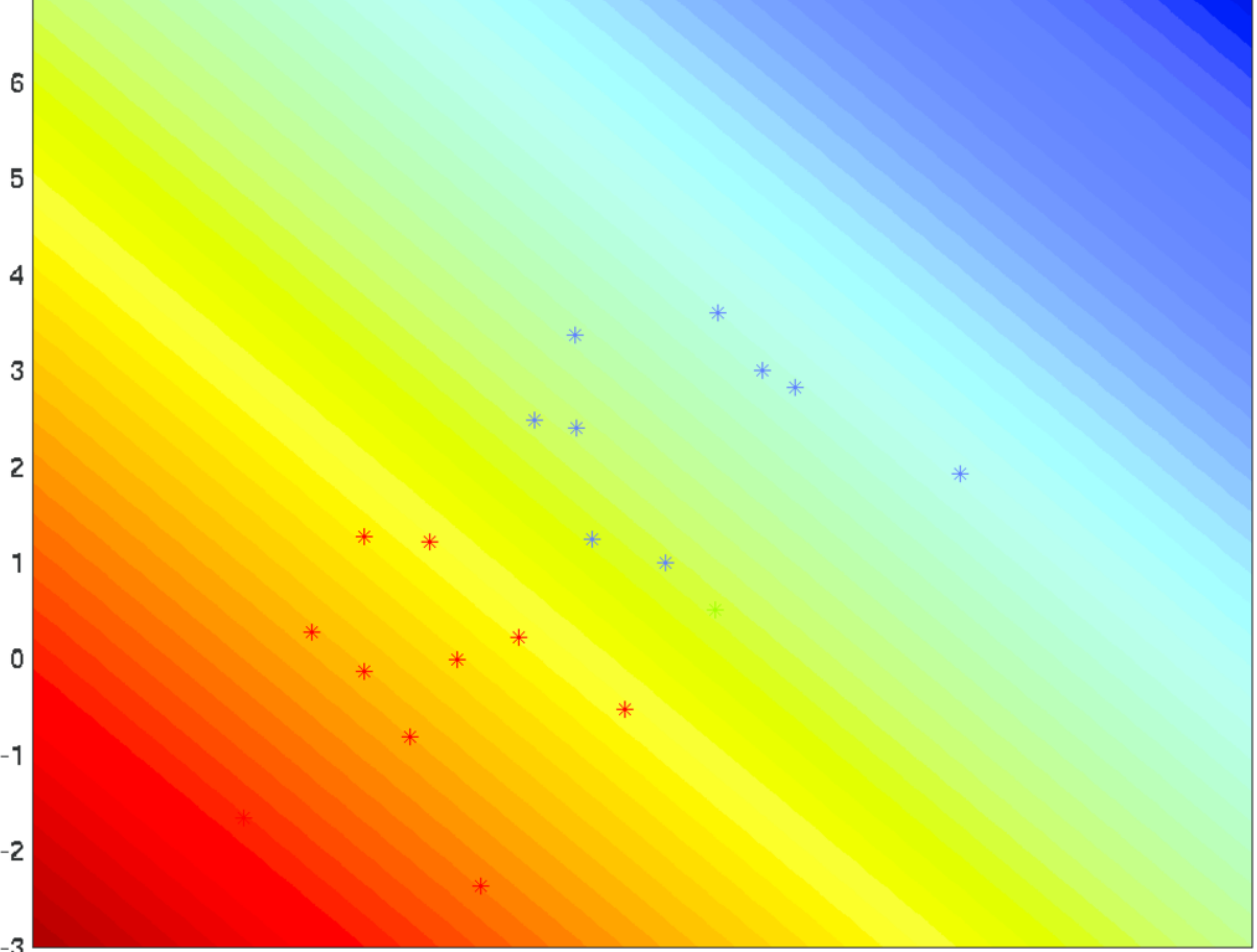

6
5
4
3
2
1
0
-1
-2
-3

6
5
4
3
2
1
0
-1
-2
-3

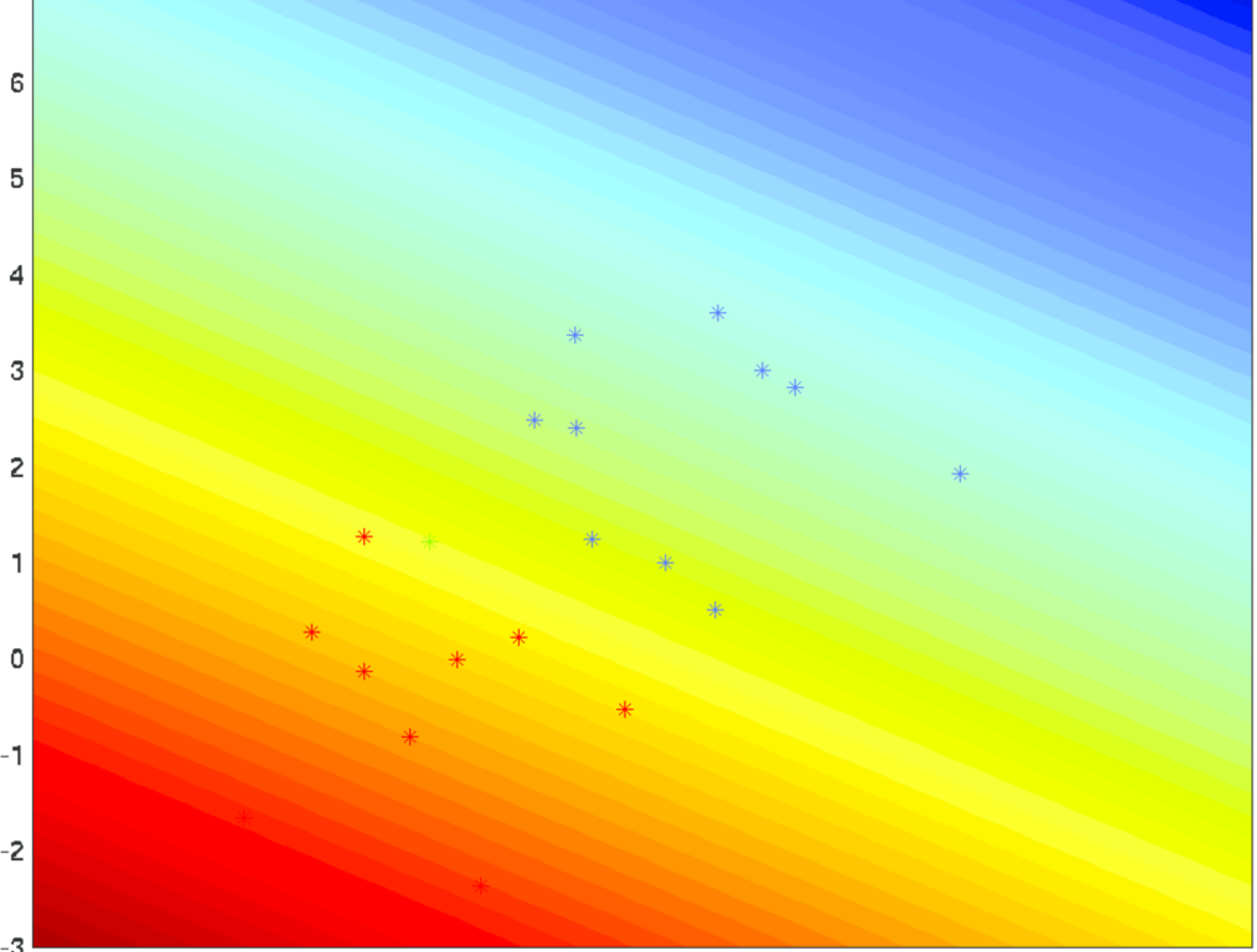
6
5
4
3
2
1
0
-1
-2
-3


6
5
4
3
2
1
0
-1
-2
-3

6
5
4
3
2
1
0
-1
-2
-3


6
5
4
3
2
1
0
-1
-2
-3

6
5
4
3
2
1
0
-1
-2
-3


6
5
4
3
2
1
0
-1
-2
-3

Gradient Descent

2. Stochastic

Objectives

Objectives

- (Linear) **Function**

$$f(x) = \langle w, x \rangle + b$$

We need to define what to **do** with it

- **Regression**

Real valued y, goodness measured by y - f(x)

$$l(y, f(x)) = |y - f(x)|$$ absolute value

$$l(y, f(x)) = \frac{1}{2}(y - f(x))^2$$ least mean squares

$$l(y, f(x)) = \max(0, |y - f(x)|)$$ ϵ-insensitive

$$l(y, f(x)) = \begin{cases} |y - f(x)| - 0.5 & \text{if } |y - f(x)| > 1 \\ 0.5(y - f(x))^2 & \text{otherwise} \end{cases}$$ Huber

$$l(y, f(x)) = |y - f(x)|$$

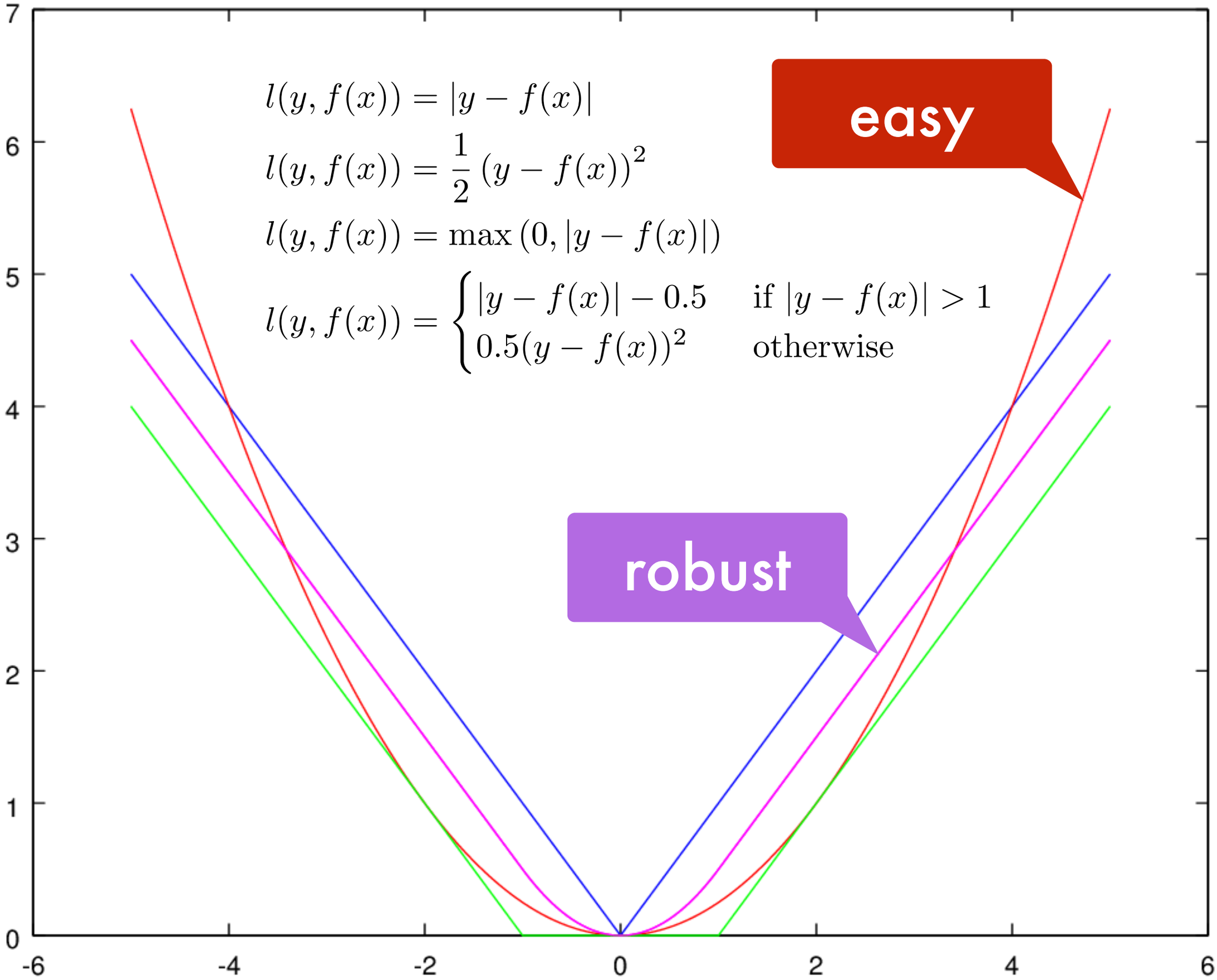
$$l(y, f(x)) = \frac{1}{2}(y - f(x))^2$$

$$l(y, f(x)) = \max(0, |y - f(x)|)$$

$$l(y, f(x)) = \begin{cases} |y - f(x)| - 0.5 & \text{if } |y - f(x)| > 1 \\ 0.5(y - f(x))^2 & \text{otherwise} \end{cases}$$

easy

robust



Regression applications

- **Stock market prediction**
  (often better to convert to log-price)
- **Image Superresolution**
  (regress from lower dimensional to higher dimensional image - Laplacian pyramid)
- **Recommendation and rating**
  (Netflix, Spotify, Yelp, Amazon)
- **Embeddings**
  Feature vectors for words, profiles, images

Objectives

- **Classification**
  - Binary y, e.g. {apples, oranges}
  - Multiple categories, e.g. {red, green, blue}
  - Ordinal relationship, e.g. {A, B, C, D, Fail}

$$l(y, f(x)) = \log\left(1 + e^{-yf(x)}\right) \qquad \text{logistic}$$

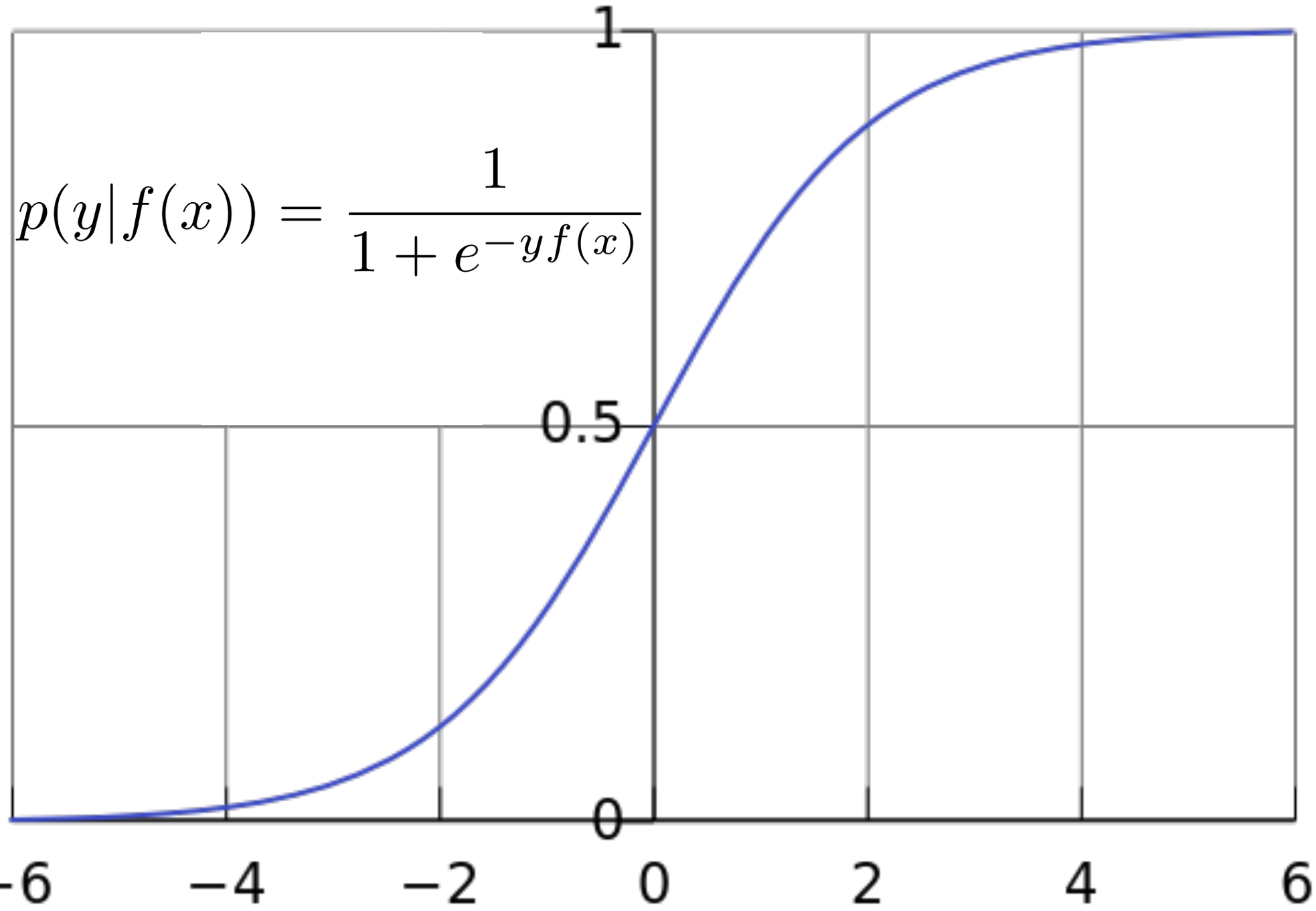
$$l(y, f(x)) = \max(0, 1 - yf(x)) \qquad \text{soft-margin}$$

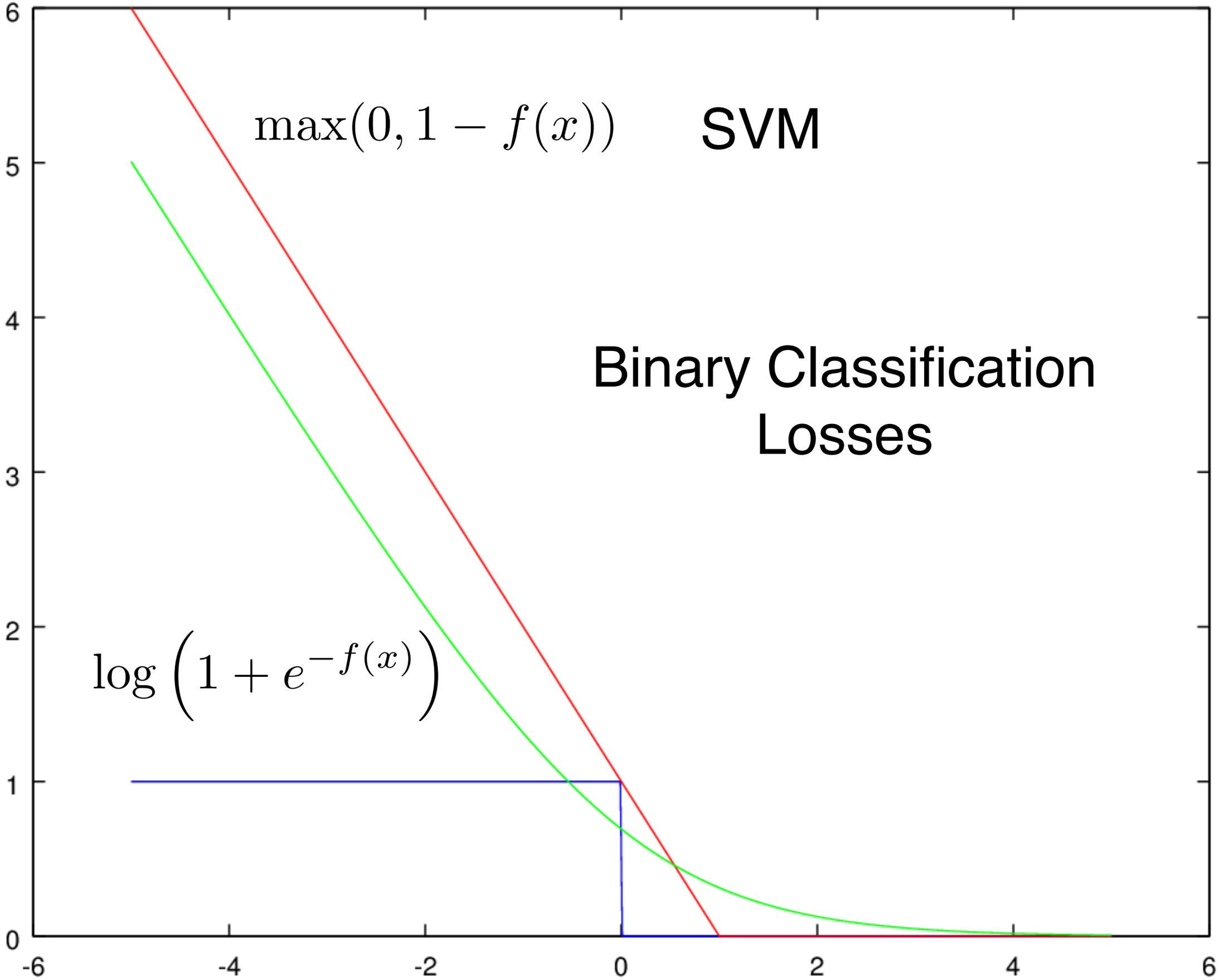
$$l(y, f(x, \cdot)) = \log \sum_{y'} e^{f(x,y')} - f(x, y) \qquad \text{multiclass logistic}$$

$$l(y, f(x, \cdot)) = \max_{y'} \left[f(x, y') + \Delta(y, y')\right] - f(x, y)$$

Logistic Regression

$$p(y|f(x)) = \frac{1}{1 + e^{-yf(x)}}$$





Marianas Labs

Carnegie Mellon University

Risk

- **Ideal Case**
  Minimize expected risk

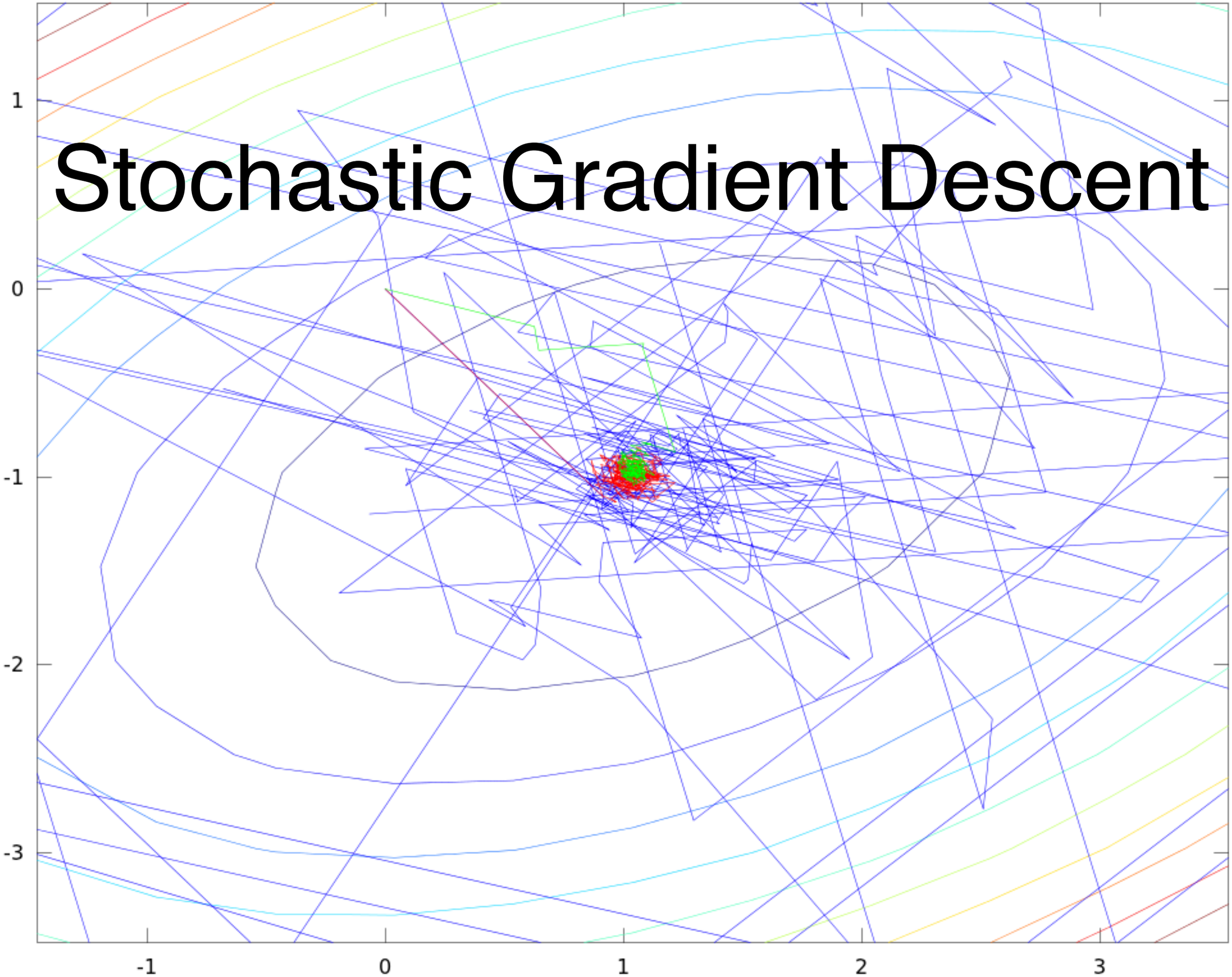
$$R[f] := \mathbf{E}_x \left[\mathbf{E}_{y|x} \left[l(y, f(x)) \right] \right]$$

- **Reality**
  We only have samples (x,y) drawn from distribution. Hence minimize empirical risk

$$R_{\mathrm{emp}}[f] := \frac{1}{m} \sum_{i=1}^{m} l(y_i, f(x_i))$$

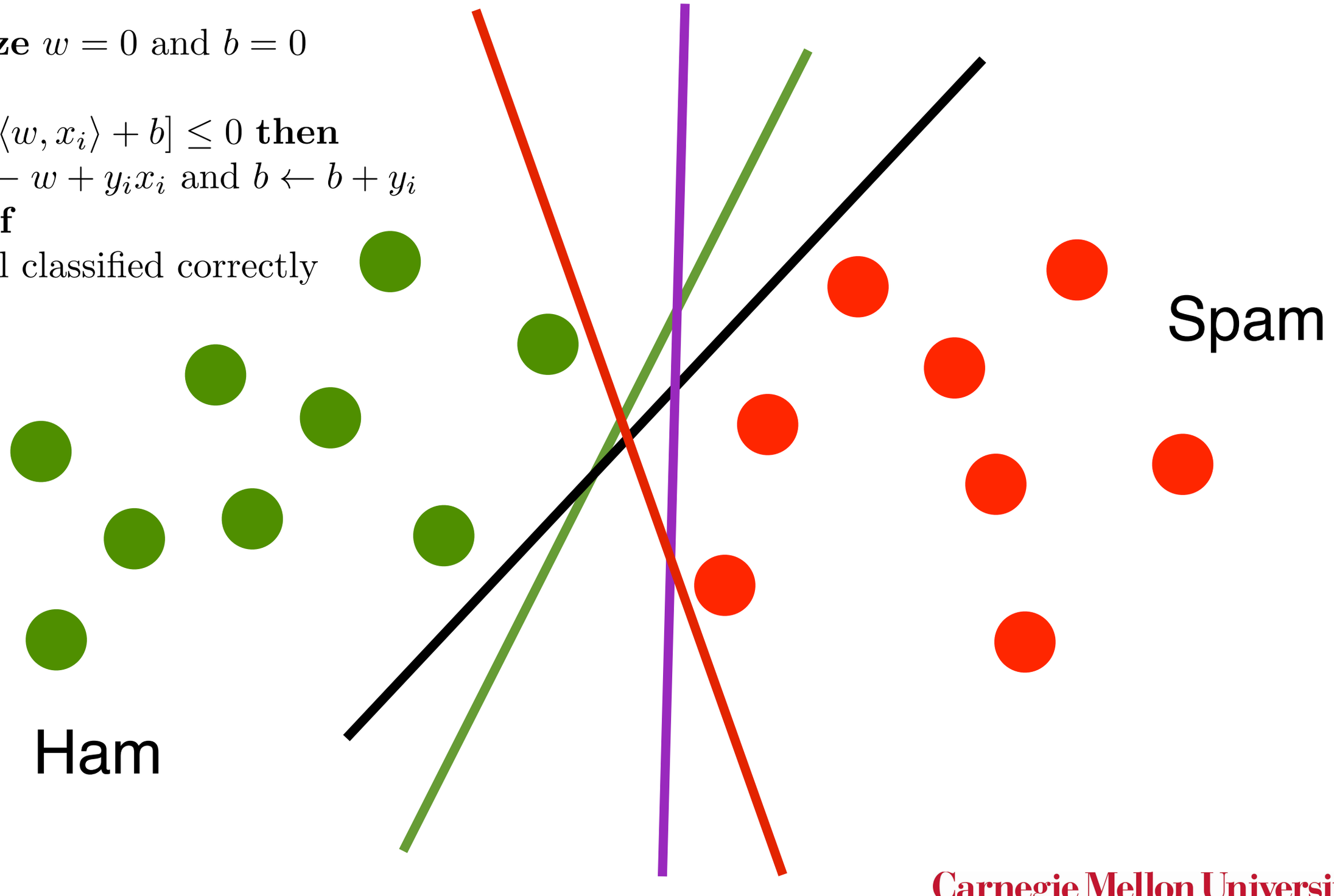
- **Challenges**
  Overfitting (regularization), Redundancy (SGD)

Stochastic Gradient Descent



Recall … Perceptron

initialize $w = 0$ and $b = 0$
repeat
  if $y_i \left[\langle w, x_i \rangle + b\right] \leq 0$ **then**
    $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$
  end if
until all classified correctly



Stochastic gradient descent

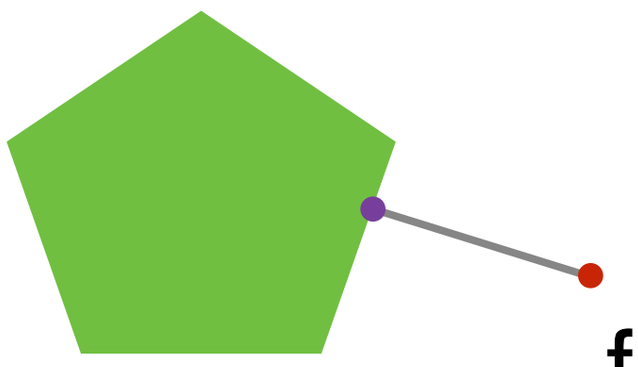
- Empirical risk

$$R_{\mathrm{emp}}[f] := \frac{1}{m}\sum_{i=1}^{m} l(y_i, f(x_i))$$

- Stochastic gradient descent (pick random x,y)

$$w_{t+1} \leftarrow w_t - \eta_t \partial_{f(x_i)} l(y_i, f(x_i)) \partial_w f(x_i)$$

- Often we require that parameters are restricted to some convex set X, hence we project on it



$$\text{Rate is } O\left(t^{-\frac{1}{2}}\right)$$

faster for strong convexity

Convergence in Expectation

initial loss

$$\mathbf{E}_{\bar{\theta}}\left[l(\bar{\theta})\right] - l^* \leq \frac{R^2 + L^2 \sum_{t=0}^{T-1} \eta_t^2}{2 \sum_{t=0}^{T-1} \eta_t} \text{ where}$$

$$l(\theta) = \mathbf{E}_{(x,y)}\left[l(y, \langle \phi(x), \theta \rangle)\right] \text{ and } l^* = \inf_{\theta \in X} l(\theta) \text{ and } \bar{\theta} = \frac{\sum_{t=0}^{T-1} \theta_t \eta_t}{\sum_{t=0}^{T-1} \eta_t}$$

expected loss

parameter average

- Proof
  Show that parameters converge to minimum

$$\theta^* \in \operatorname*{argmin}_{\theta \in X} l(\theta) \text{ and set } r_t := \|\theta^* - \theta_t\|$$

Nesterov and Vial 10.1016/j.automatica.2008.01.017

Proof

$$
\begin{aligned}
r_{t+1}^2 &= \left\|\pi_X[\theta_t - \eta_t g_t] - \theta^*\right\|^2 \\
&\leq \left\|\theta_t - \eta_t g_t - \theta^*\right\|^2 \\
&= r_t^2 + \eta_t^2 \left\|g_t\right\|^2 - 2\eta_t \left\langle \theta_t - \theta^*, g_t \right\rangle \\
\text{hence } \mathbf{E}\left[r_{t+1}^2 - r_t^2\right] &\leq \eta_t^2 L^2 + 2\eta_t \left[l^* - \mathbf{E}[l(\theta_t)]\right] \\
&\leq \eta_t^2 L^2 + 2\eta_t \left[l^* - \mathbf{E}[l(\bar{\theta})]\right]
\end{aligned}
$$

by convexity

- Summing over inequality for t proves claim
- This yields randomized algorithm for minimizing objective functions (try log times and pick the best / or average median trick)

Rates

- Guarantee

$$\mathbf{E}_{\bar{\theta}}\left[l(\bar{\theta})\right] - l^* \leq \frac{R^2 + L^2 \sum_{t=0}^{T-1} \eta_t^2}{2 \sum_{t=0}^{T-1} \eta_t}$$

- If we know R, L, T pick constant learning rate

$$\eta = \frac{R}{L\sqrt{T}} \text{ and hence } \mathbf{E}_{\bar{\theta}}[l(\bar{\theta})] - l^* \leq \frac{R[1+1/T]L}{2\sqrt{T}} < \frac{LR}{\sqrt{T}}$$

- If we don't know T pick $\eta_t = O(t^{-\frac{1}{2}})$
  This costs us an additional log term

$$\mathbf{E}_{\bar{\theta}}[l(\bar{\theta})] - l^* = O\left(\frac{\log T}{\sqrt{T}}\right)$$

Strong Convexity

$$l_i(\theta') \geq l_i(\theta) + \langle \partial_\theta l_i(\theta), \theta' - \theta \rangle + \frac{1}{2}\lambda \left\| \theta - \theta' \right\|^2$$

- Use this to bound the expected deviation

$$\begin{aligned} r_{t+1}^2 &\leq r_t^2 + \eta_t^2 \left\| g_t \right\|^2 - 2\eta_t \left\langle \theta_t - \theta^*, g_t \right\rangle \\ &\leq r_t^2 + \eta_t^2 L^2 - 2\eta_t \left[l_t(\theta_t) - l_t(\theta^*) \right] - 2\lambda\eta_t r_k^2 \end{aligned}$$

$$\text{hence } \mathbf{E}[r_{t+1}^2] \leq (1 - \lambda h_t)\mathbf{E}[r_t^2] - 2\eta_t \left[\mathbf{E}\left[l(\theta_t) \right] - l^* \right]$$

- Exponentially decaying averaging

$$\bar{\theta} = \frac{1-\sigma}{1-\sigma^T} \sum_{t=0}^{T-1} \sigma^{T-1-t} \theta_t$$

and plugging this into the discrepancy yields

$$l(\bar{\theta}) - l^* \leq \frac{2L^2}{\lambda T} \log \left[1 + \frac{\lambda R T^{\frac{1}{2}}}{2L} \right] \text{ for } \eta = \frac{2}{\lambda T} \log \left[1 + \frac{\lambda R T^{\frac{1}{2}}}{2L} \right]$$

More variants

- Adversarial guarantees

$$\theta_{t+1} \leftarrow \pi_x \left[\theta_t - \eta_t \partial_\theta \left(y_t, \langle \phi(x_t), \theta_t \rangle\right)\right]$$

  has low regret (average instantaneous cost) for arbitrary orders (useful for game theory)

- Ratliff, Bagnell, Zinkevich
  $O(t^{-\frac{1}{2}})$ learning rate
- Shalev-Shwartz, Srebro, Singer (Pegasos)
  $O(t^{-1})$ learning rate (but need constants)
- Bartlett, Rakhlin, Hazan
  (add strong convexity penalty)

In a Nutshell

Marianas Labs

Carnegie Mellon University

Key Components

- **Data & distribution**
  (image, ‘cat’), (email, ‘spam’), ((user,movie), ★★★)
- **Loss Function**
  least mean squares, logistic, multiclass, …
- **Function class**
  linear, nonlinear, kernels, deep
- **Optimization procedure**
  Stochastic gradient descent, batch, sampling
- **Capacity control**
  Norm of coefficients, operator, dropout

Key Components

- **Data & distribution**

( , **‘cat’)**

- **Loss Function**

$$\log\left(1 + e^{-yf(x)}\right)$$

- **Function class**

$$f(x) = \langle w, x \rangle$$

- **Optimization procedure**

$$w \leftarrow w + \eta \frac{yx}{1 + e^{yf(x)}}$$

- **Capacity control**

$$w \leftarrow (1 - \eta\lambda)w + \eta \frac{yx}{1 + e^{yf(x)}}$$

Model Selection

- **Simple model**
  Linear function in R, n observations, no problem
  (e.g. estimating voltage in sockets)
- **High dimensional model**
  Linear function in R^d, $n < d$ observations
  The problem is underdetermined
- **Even for d = O(n) not enough data to handle noise**
  - Only small number of nonzeroes (sparsity)
  - Only small coefficients (norm)
  - Insensitive to local changes (dropout)

Outline

- **Basics**
  - Perceptron (and convergence rule)
  - Stochastic gradient descent
  - Loss functions and objectives
- **Deep Networks**
  - Layers and Invariances
  - Advanced objectives
- **State and Structure**
  - Optimization
  - Autoregressive Models & Hidden State
  - Toolkits

3. Backprop

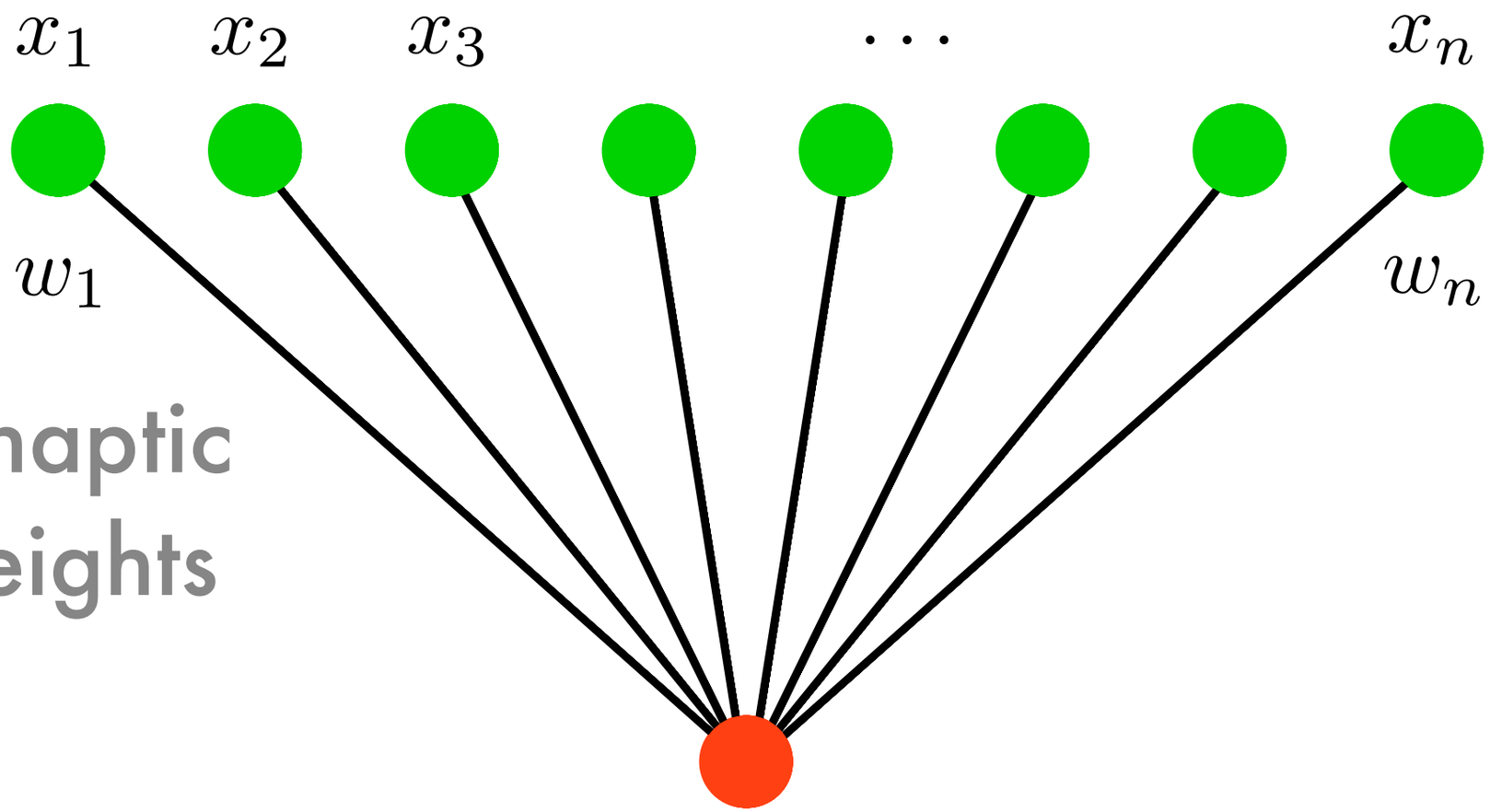
A brief history of computers

	1970s	1980s	1990s	2000s	2010s
Data	10^2	10^3	10^5	10^8	10^{11}
RAM	?	1MB	100MB	10GB	1TB
CPU	?	10MF	1GF	100GF	1PF GPU

deep nets | kernel methods | deep nets

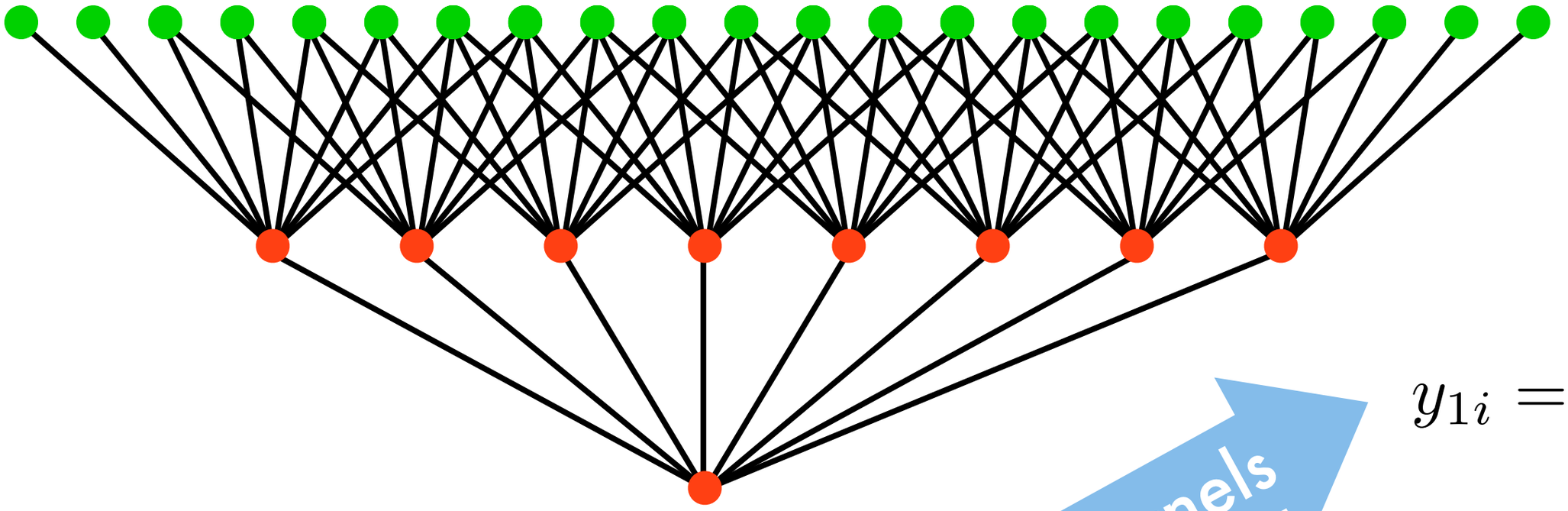
- Data grows at higher exponent
- Moore's law (silicon) vs. Kryder's law (disks)
- Early algorithms data bound, now CPU/RAM bound

Perceptron



$$y(x) = \sigma(\langle w, x \rangle)$$

Nonlinearities via Layers



$$y_{1i} = k(x_i, x)$$

Kernels

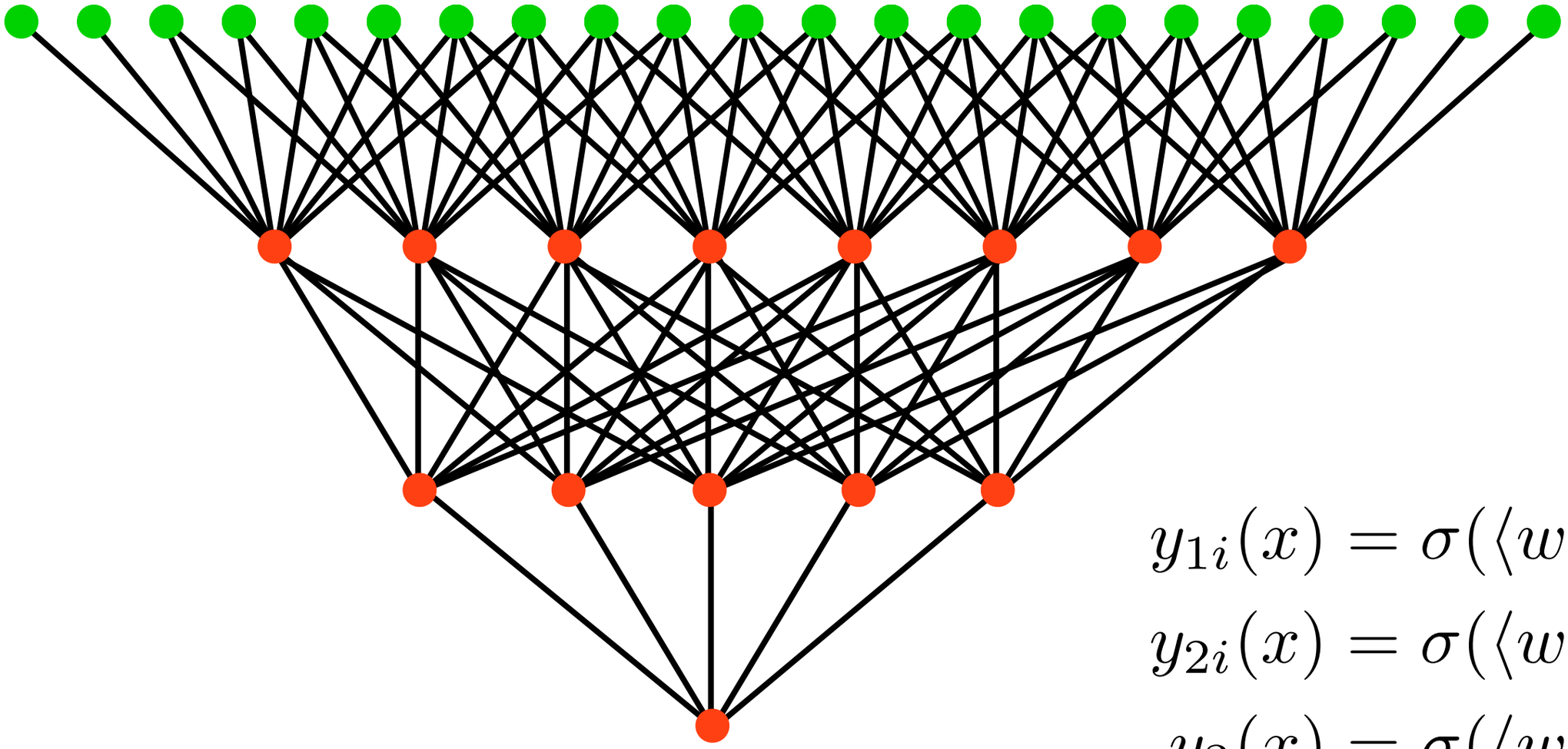
$$y_{1i}(x) = \sigma(\langle w_{1i}, x \rangle)$$

$$y_2(x) = \sigma(\langle w_2, y_1 \rangle)$$

Deep Nets

optimize
all weights

Nonlinearities via Layers



$$y_{1i}(x) = \sigma(\langle w_{1i}, x \rangle)$$
$$y_{2i}(x) = \sigma(\langle w_{2i}, y_1 \rangle)$$
$$y_3(x) = \sigma(\langle w_3, y_2 \rangle)$$

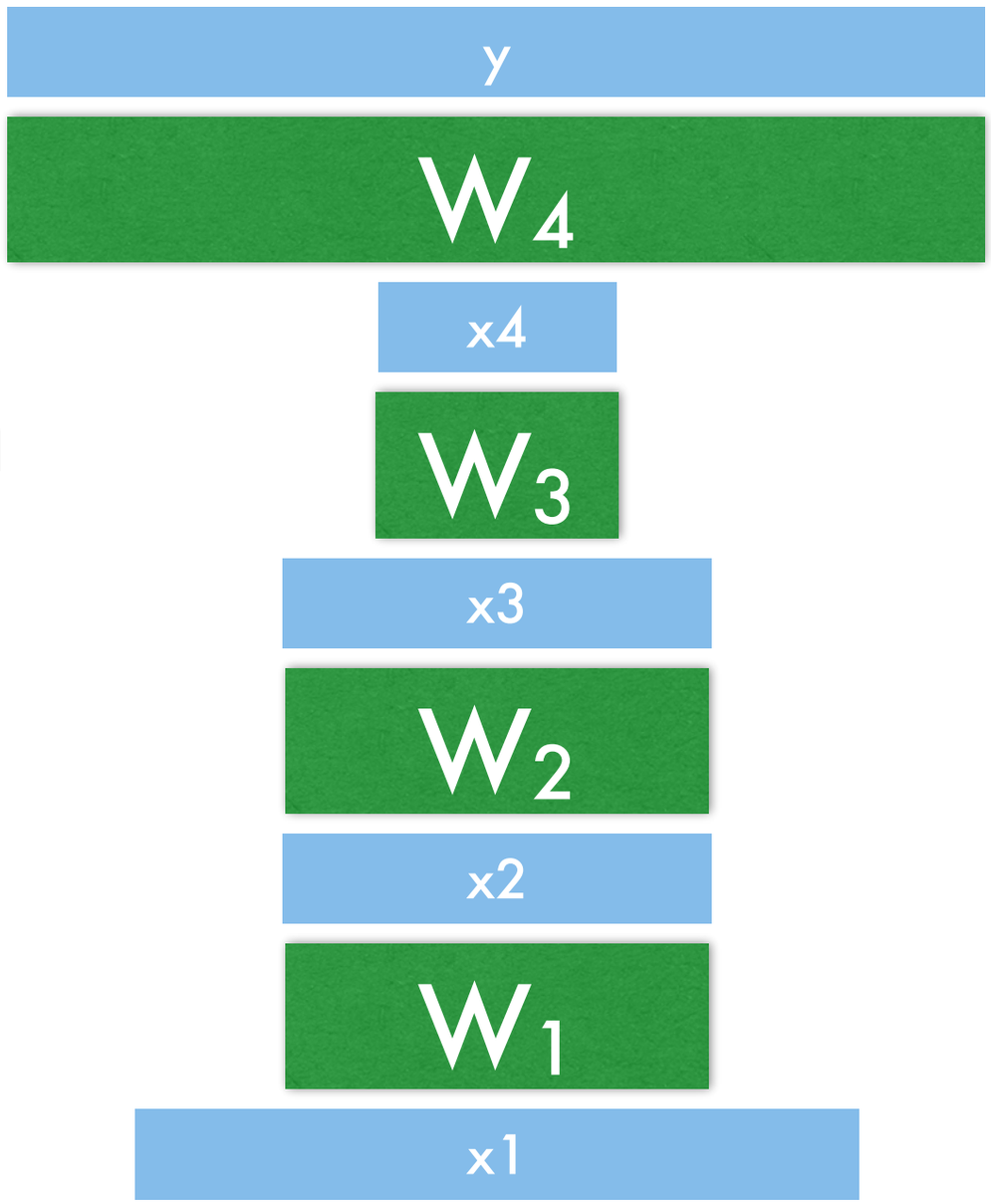
Multilayer Perceptron

- **Layer Representation**

$$y_i = W_i x_i$$

$$x_{i+1} = \sigma(y_i)$$

- (typically) iterate between linear mapping Wx and nonlinear function
- **Loss function** $l(y, y_i)$ to measure quality of estimate so far

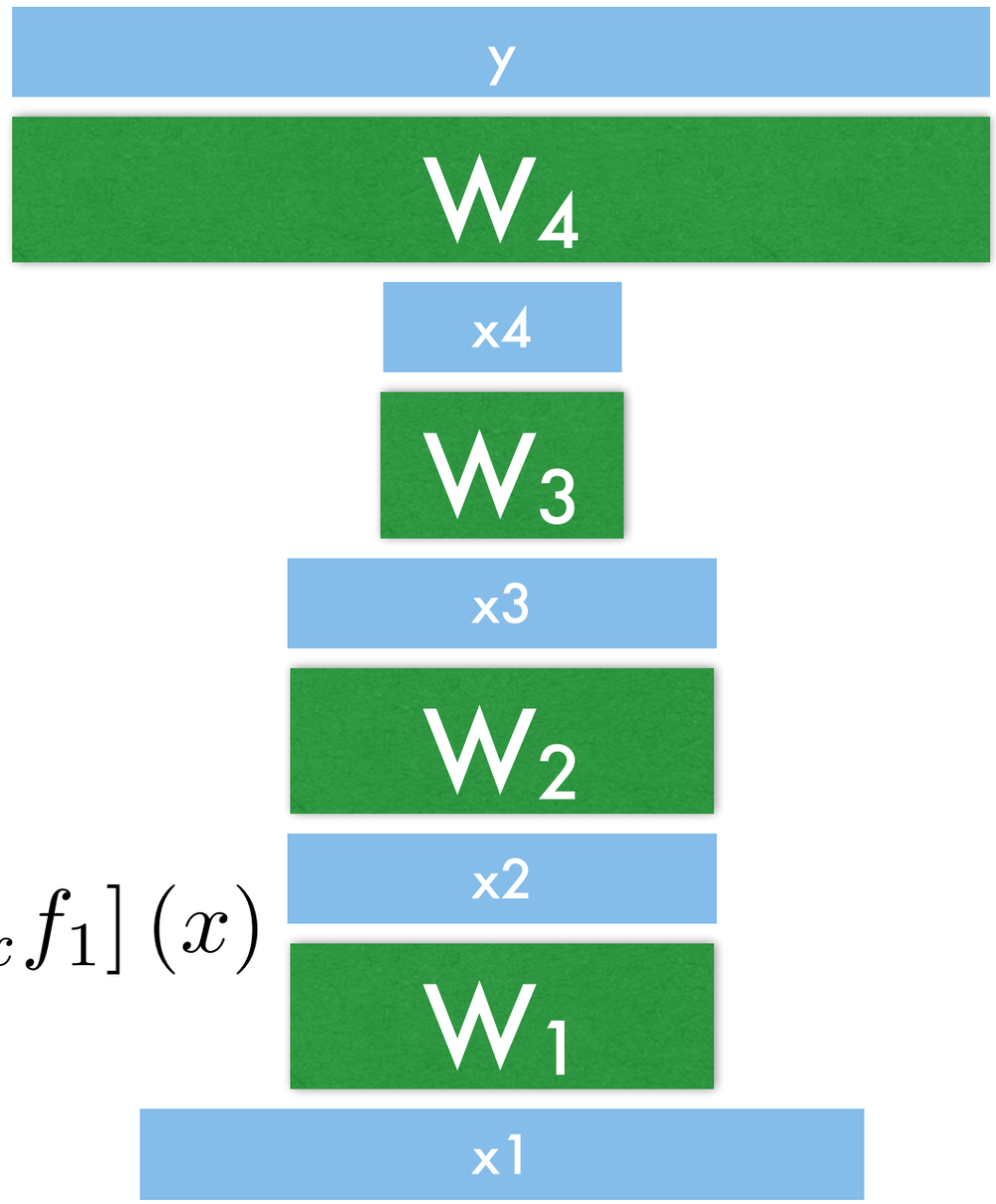


Multilayer Perceptron

- **Layer Representation**

$$y_i = W_i x_i$$

$$x_{i+1} = \sigma(y_i)$$



- **Change in Objective**

$$g_j = \partial_{W_j} l(y, y_i)$$

- **Chain Rule**

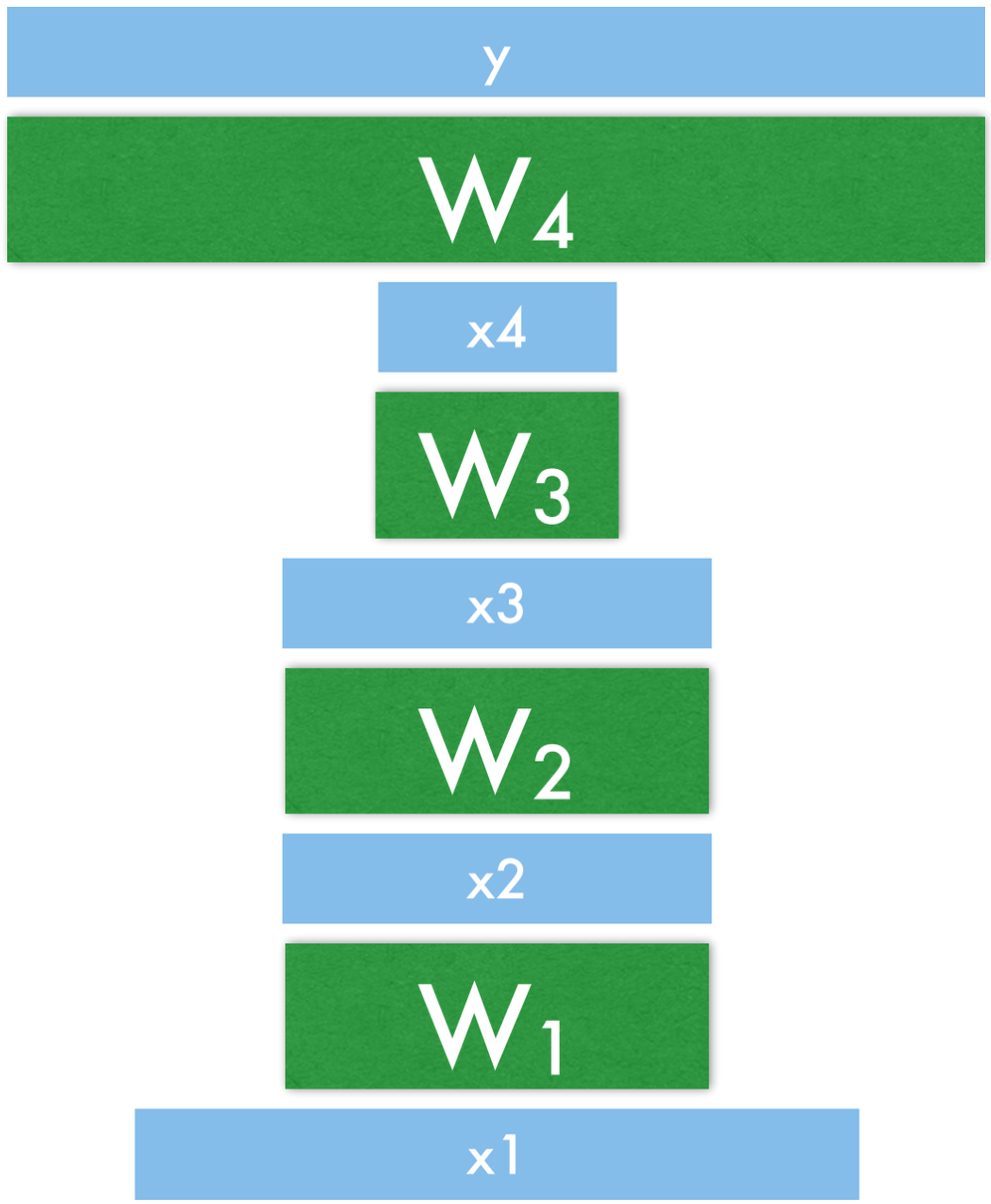
$$\partial_x \left[f_2 \circ f_1\right](x) = \left[\partial_{f_1} f_2 \circ f_1\right](x) \left[\partial_x f_1\right](x)$$

Multilayer Perceptron

- **Layer Representation**

$$y_i = W_i x_i$$

$$x_{i+1} = \sigma(y_i)$$



- **Gradients**

$$\partial_{x_i} y_i = W_i$$

$$\partial_{W_i} y_i = x_i$$

$$\partial_{y_i} x_{i+1} = \sigma'(y_i)$$

$$\Longrightarrow \partial_{x_i} x_{i+1} = \sigma'(y_i) W_i^\top$$

Backpropagation

- **Layers & Gradients**

$$y_i = W_i x_i$$

$$x_{i+1} = \sigma(y_i)$$

$$\partial_{x_i} y_i = W_i$$

$$\partial_{W_i} y_i = x_i$$

$$\partial_{y_i} x_{i+1} = \sigma'(y_i)$$

$$\Longrightarrow \partial_{x_i} x_{i+1} = \sigma'(y_i) W_i^\top$$

- **Backpropagation**

$$\begin{aligned}
\boxed{\partial_{x_i} l(y, y_n)} &= \boxed{\partial_{x_{i+1}} l(y, y_n)} \partial_{x_i} x_{i+1} \\
&= \boxed{\partial_{x_{i+1}} l(y, y_n)} \boxed{\sigma'(y_i)} W_i^\top \\
\partial_{W_i} l(y, y_n) &= \partial_{y_i} l(y, y_n) \partial_{W_i} y_i \\
&= \boxed{\partial_{x_{i+1}} l(y, y_n)} \boxed{\sigma'(y_i)} x_i^\top
\end{aligned}$$

Backpropagation

- **Layers & Gradients**

$$y_i = W_i x_i$$

$$x_{i+1} = \sigma(y_i)$$

$$\partial_{x_i} y_i = W_i$$

$$\partial_{W_i} y_i = x_i$$

$$\partial_{y_i} x_{i+1} = \sigma'(y_i)$$

$$\implies \partial_{x_i} x_{i+1} = \sigma'(y_i) W_i^\top$$

- **Backpropagation**

$$g_n = l'(y, y_n) W_n$$

$$g_i = g_{i+1} \sigma'(y_i) W_i^\top$$

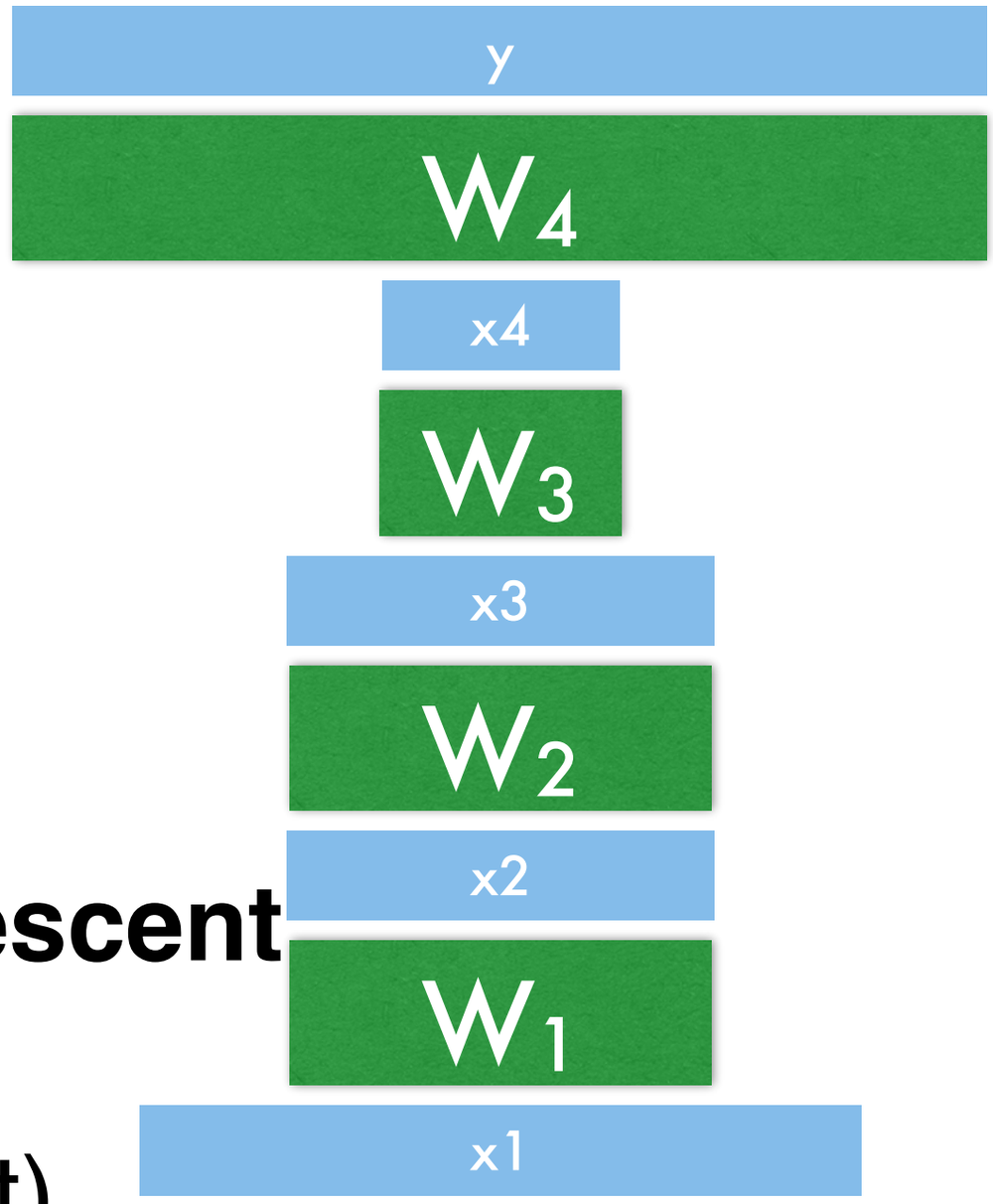
$$\partial_{W_i} l(y, n_n) = g_{i+1} \sigma'(y_i) x_i^\top$$

Optimization

- **Layer Representation**

$$y_i = W_i x_i$$

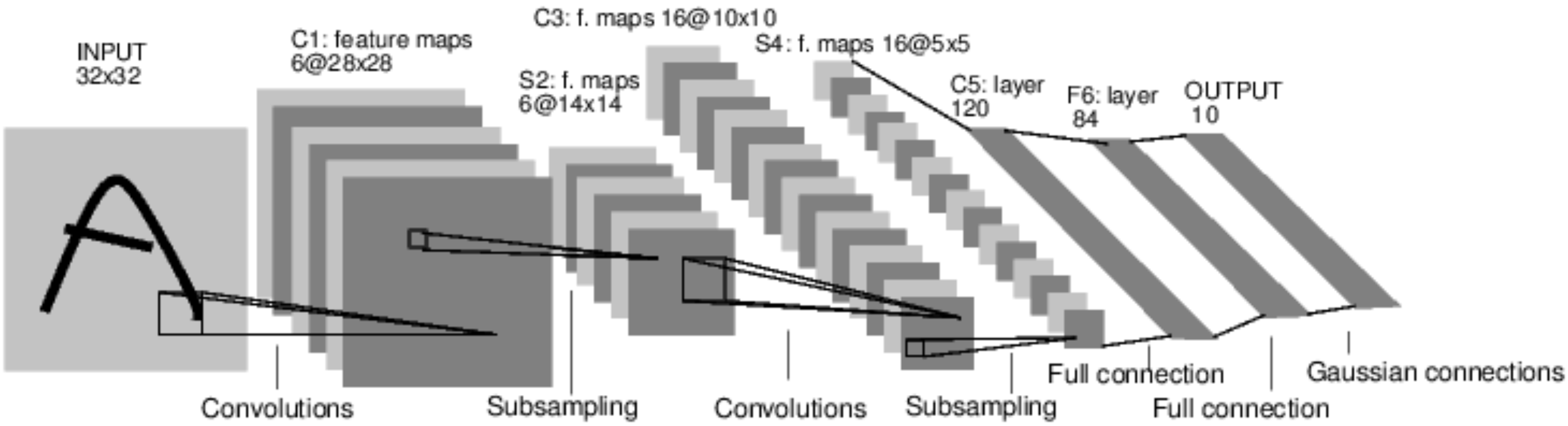
$$x_{i+1} = \sigma(y_i)$$



- **Gradient descent**

$$W_i \leftarrow W_i - \eta \partial_{W_i} l(y, y_n)$$

- Second order method (use higher derivatives)
- **Stochastic gradient descent** (use only one sample)
- **Minibatch** (small subset)



4. Layers & Symmetries

Marianas Labs

Carnegie Mellon University

Fully Connected

- Forward mapping

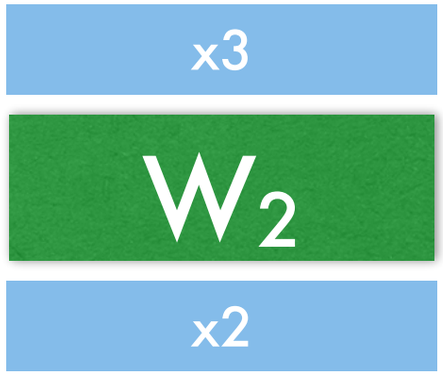
$$y_i = W_i x_i$$
$$x_{i+1} = \sigma(y_i)$$

  with subsequent nonlinearity

- Backprop gradients

$$\partial_{x_i} x_{i+1} = \sigma'(y_i) W_i^\top$$
$$\partial_{W_i} x_{i+1} = \sigma'(y_i) x_i^\top$$

- General purpose layer



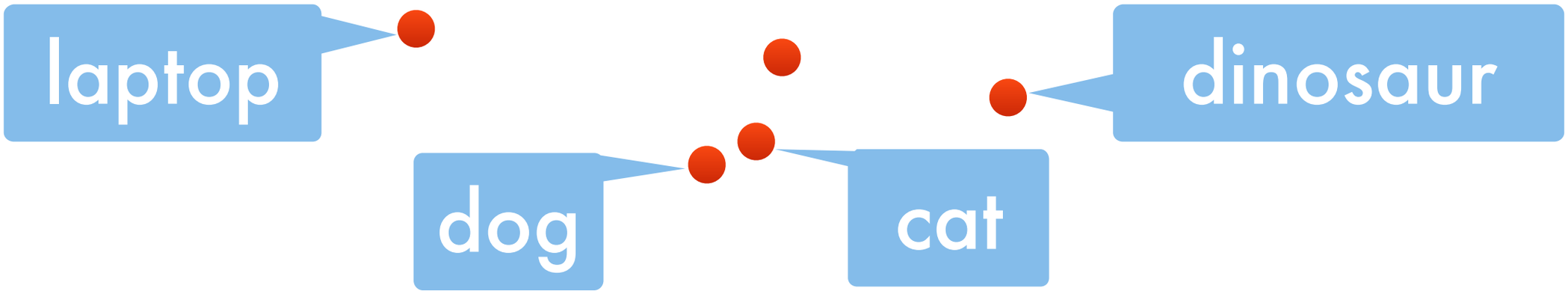
Example - Word Vectors

- Full vocabulary (say 30k words)
- One-hot encoding for words
  (0,0,0,0,1,0,0,0) (0,0,0,0,0,0,1,0)

cat

dog

- Map words into vector space via y = Wx



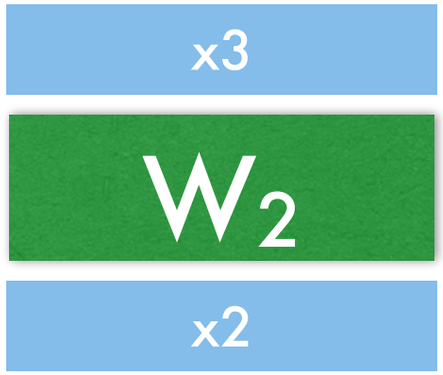
- Postprocess as normal.

Rectified Linear Unit (ReLu)

- Forward mapping

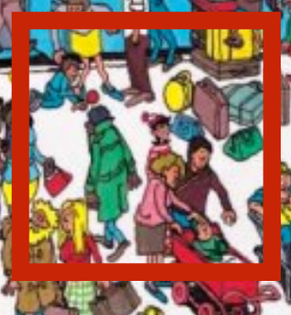
$$y_i = W_i x_i$$
$$x_{i+1} = \sigma(y_i)$$

with subsequent nonlinearity

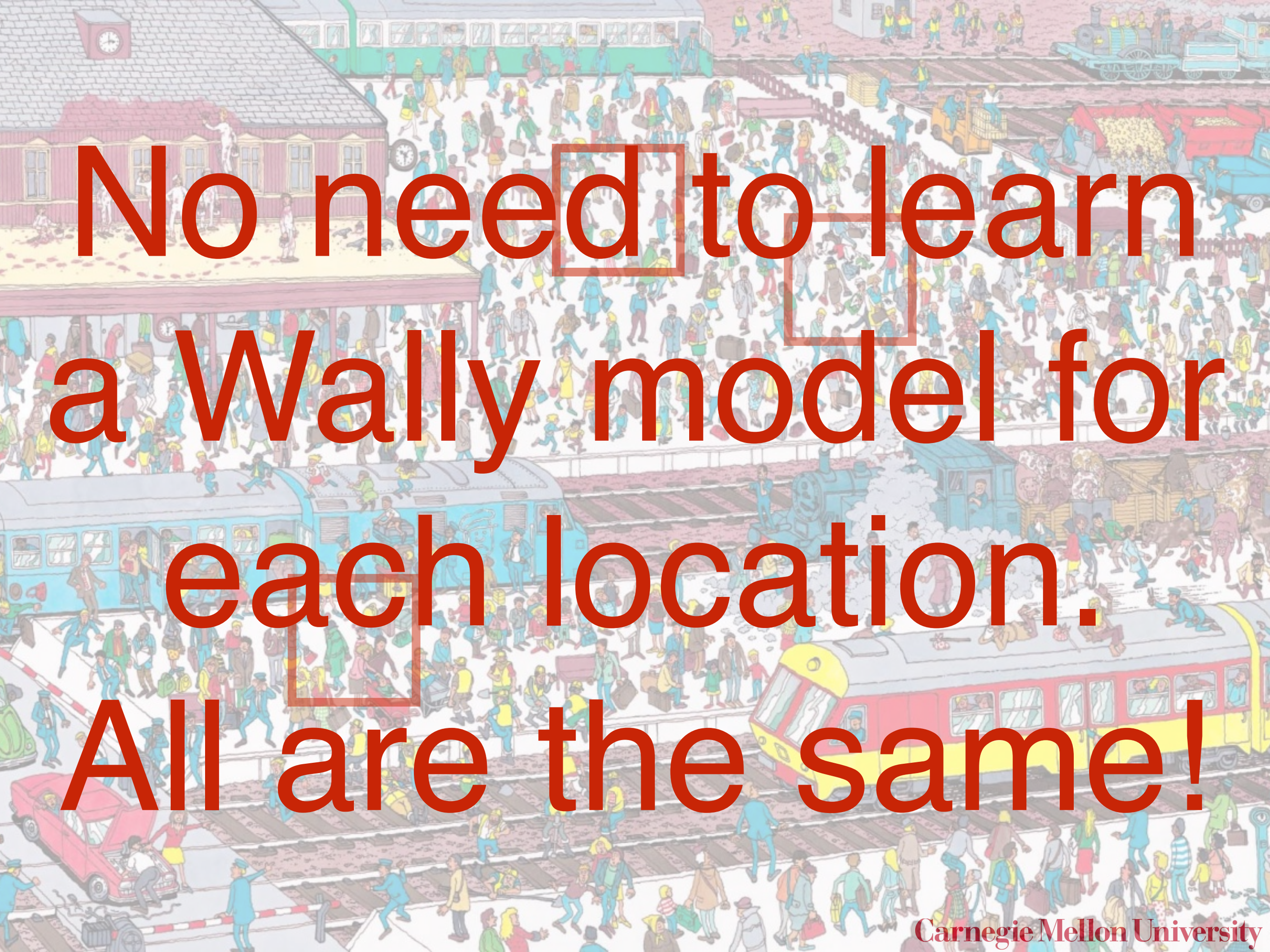


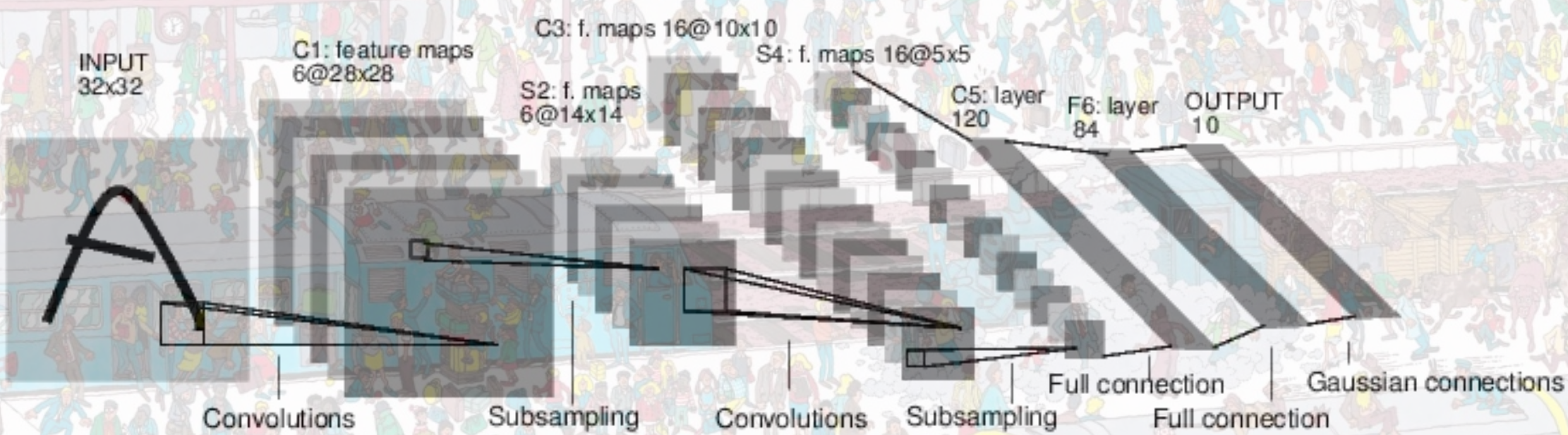
- Gradients vanish at tails
- Solution - replace by max(0,x)
  - Derivative is in {0; 1}
  - Sparsity of signal
    (Nair & Hinton, machinelearning.wustl.edu/mlpapers/paper_files/icml2010_NairH10.pdf)

Where is Wally?



3.2.1

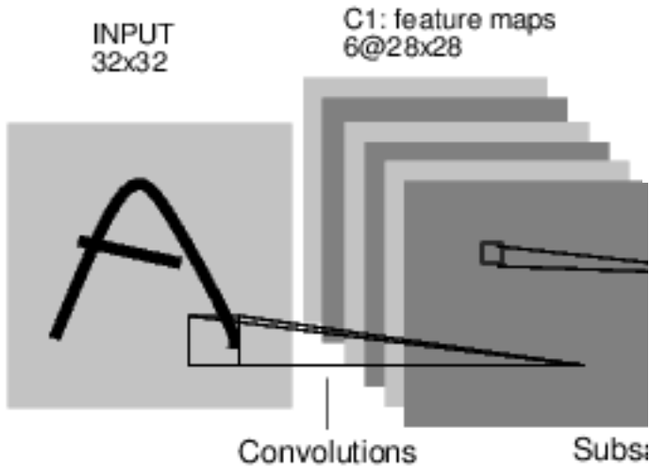

No need to learn
a Wally model for
each location.
All are the same!
Carnegie Mellon University

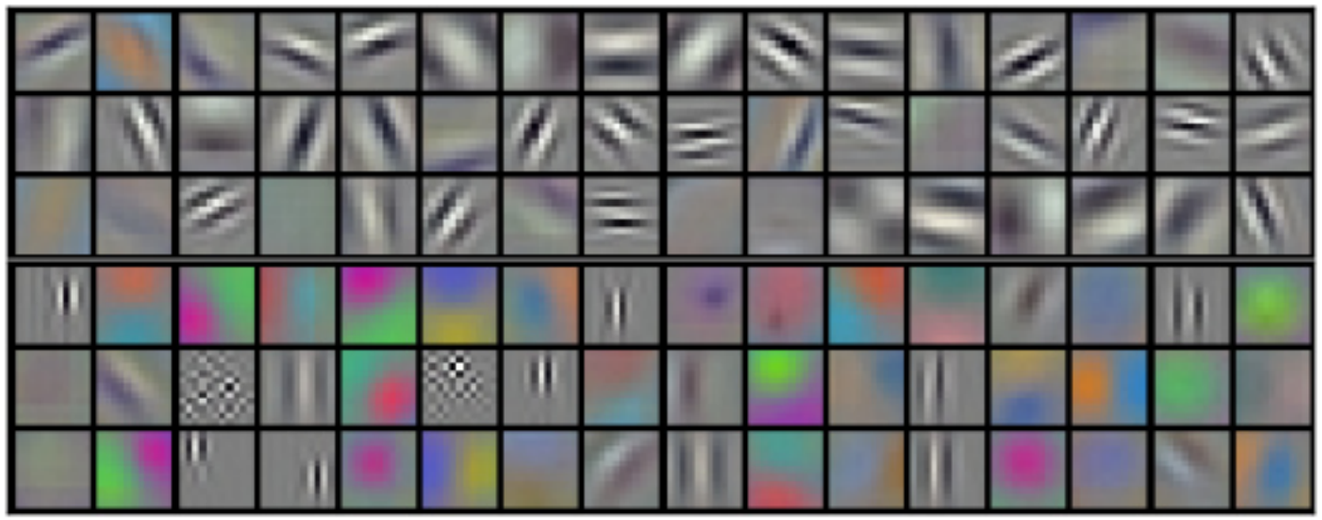


LeNet for OCR (1990s)

Convolutional Layers

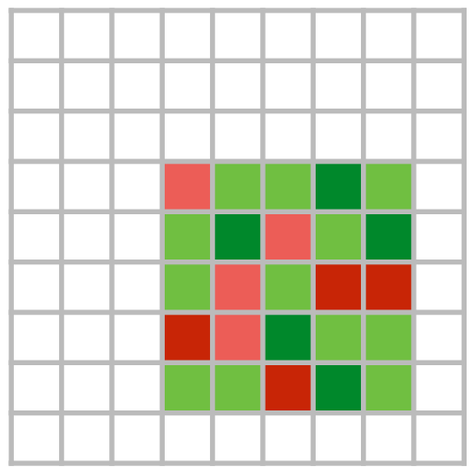
- Images have translation invariance (to some extent)
- Low level is mostly edge and feature detectors
- Usually via convolution (plus nonlinearity)





Convolutional Layers

- **Feature Locality**
  Relevant information only in neighborhood of pixel



$$y_{ij} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} W_{ij,ab} x_{i+a,j+b}$$

- **Translation Invariance**
  Weights invariant relative to shift in point of view

$$y_{ij} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} W_{ab} x_{i+a,j+b}$$

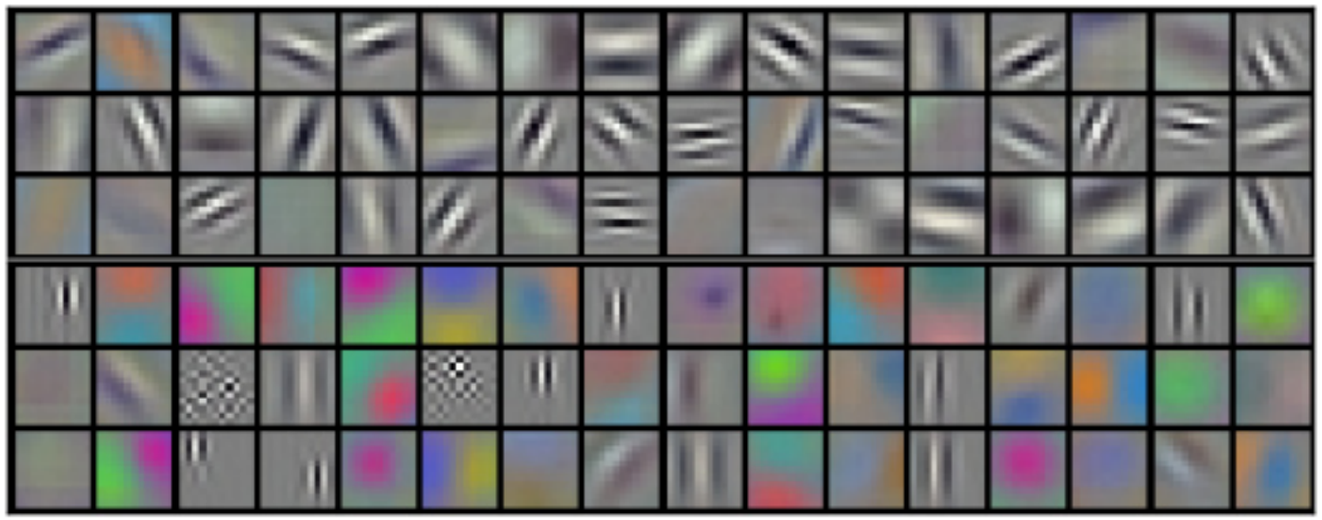
Convolutional Layers

- Images have translation invariance
- Forward (brute force BLAS beats convolution)

$$y_i = x_i \circ W_i$$

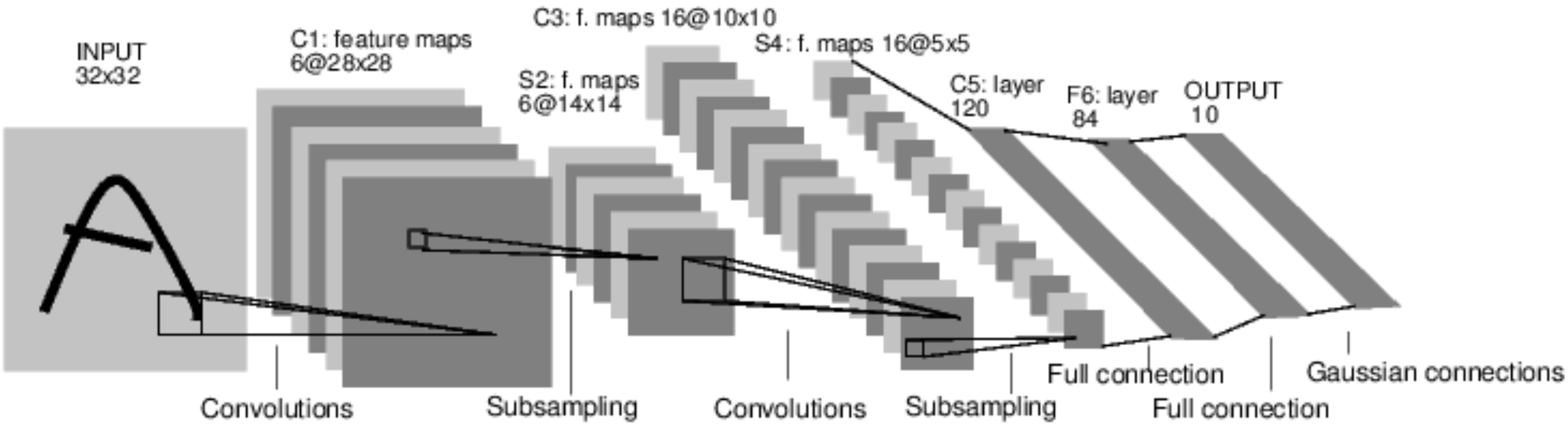
$$x_{i+1} = \sigma(y_i)$$

- Backward gradients: convolve appropriately (see homework)



Subsampling & MaxPooling

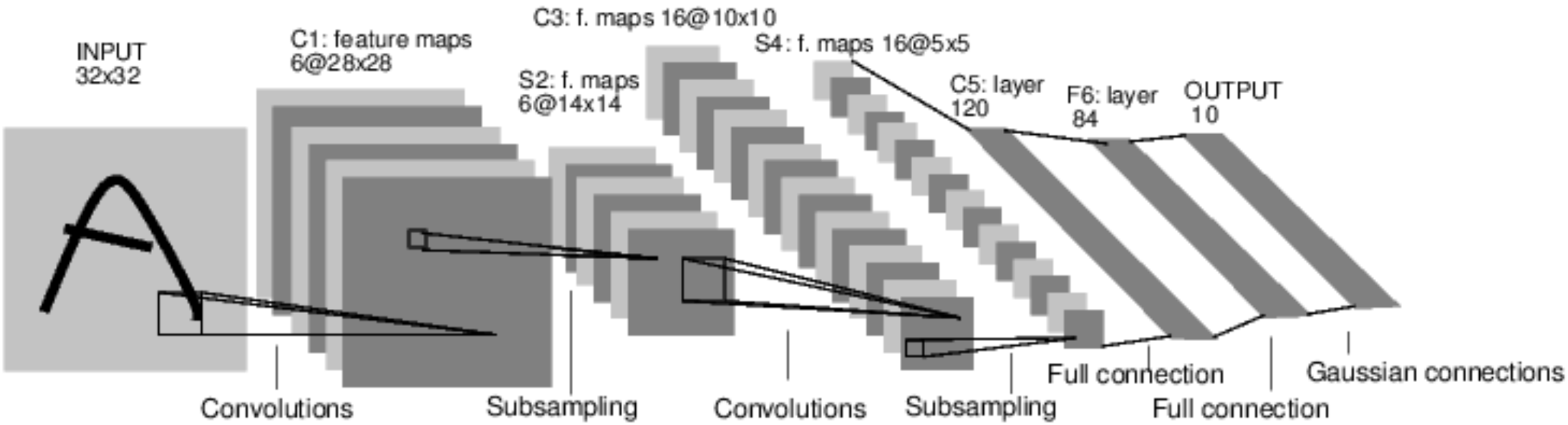
- Multiple convolutions blow up dimensionality



- Subsampling - average over patches (this works decently)
- MaxPooling - pick the maximum over patches (often non overlapping ones)

Subsampling & MaxPooling

- Multiple convolutions blow up dimensionality



- Subsampling

$$y_{ij} = \frac{1}{4}\left[x_{2i,2j} + x_{2i+1,2j} + x_{2i,2j+1} + x_{2i+1,2j+1}\right]$$

- MaxPooling

$$y_{ij} = \max\left[x_{2i,2j}, x_{2i+1,2j}, x_{2i,2j+1}, x_{2i+1,2j+1}\right]$$

Depth vs. Width

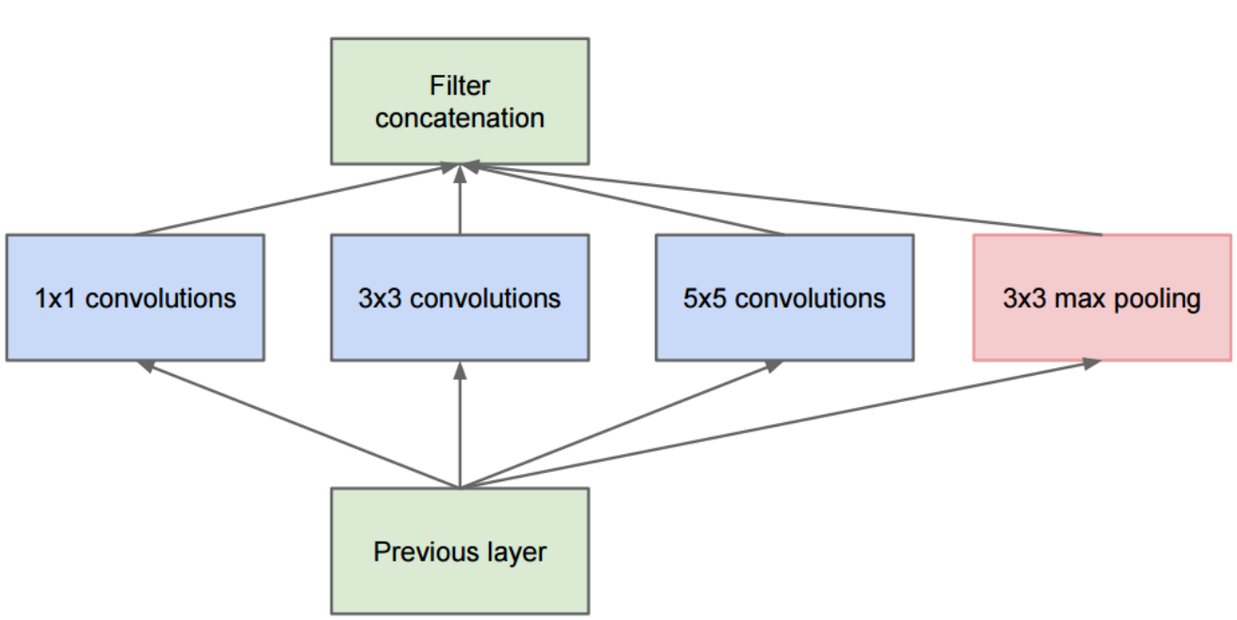
- Longer range effects
  - many narrow convolutions
  - few wide convolutions
- More nonlinearities work better (same number of parameters)

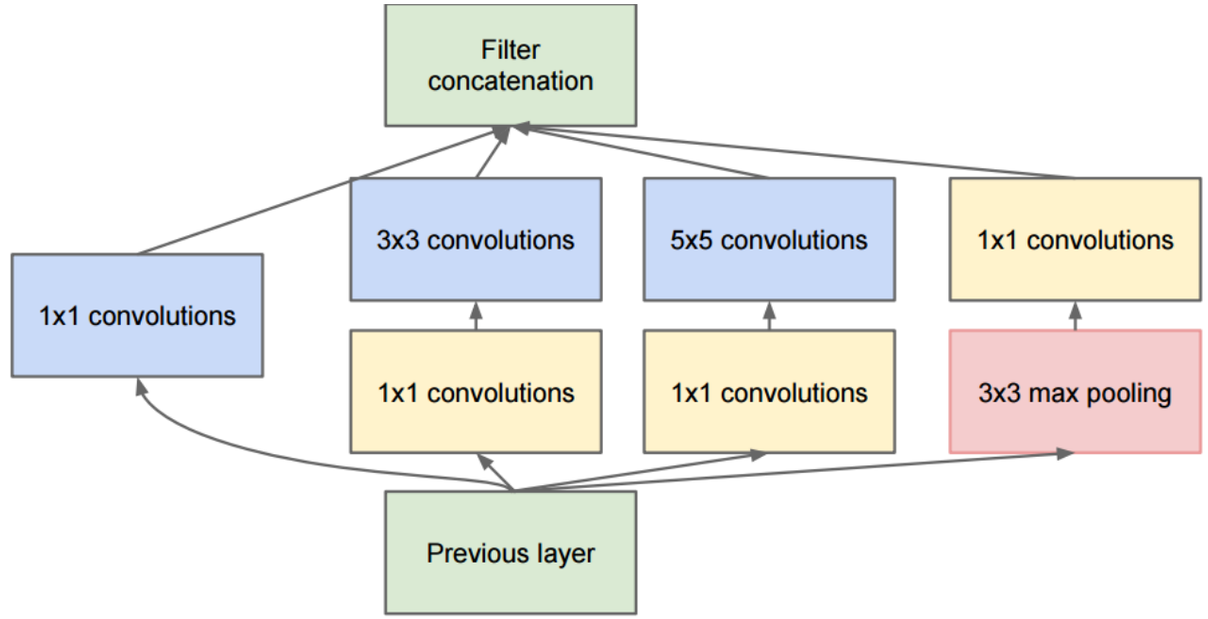
Simonyan and Zisserman
arxiv.org/pdf/1409.1556v6.pdf

Marianas Labs

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Fancy structures

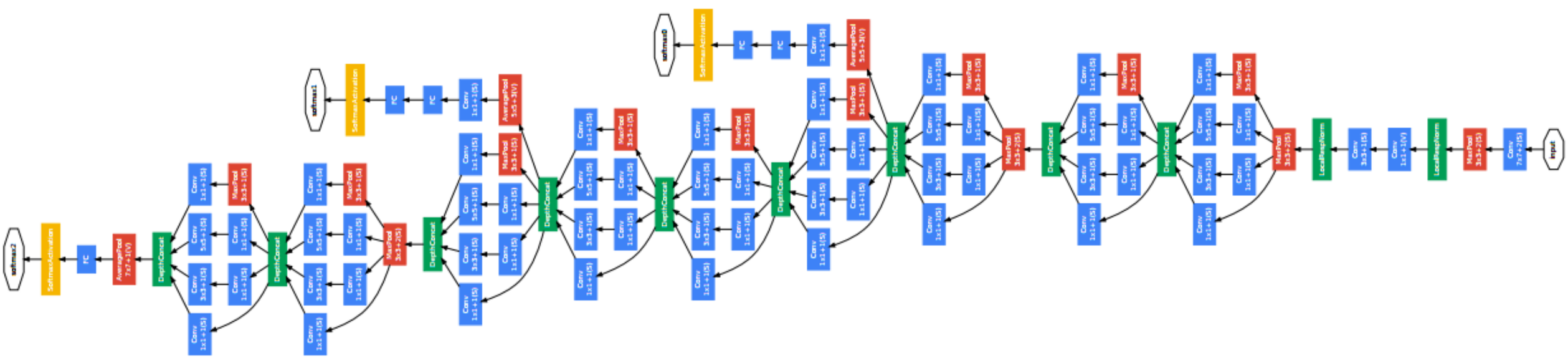




- Compute different filters
- Compose one big vector from all of them
- Layer this iteratively

Szegedy et al. arxiv.org/pdf/1409.4842v1.pdf

Fancy structures



- Compute different filters
- Compose one big vector from all of them
- Layer this iteratively

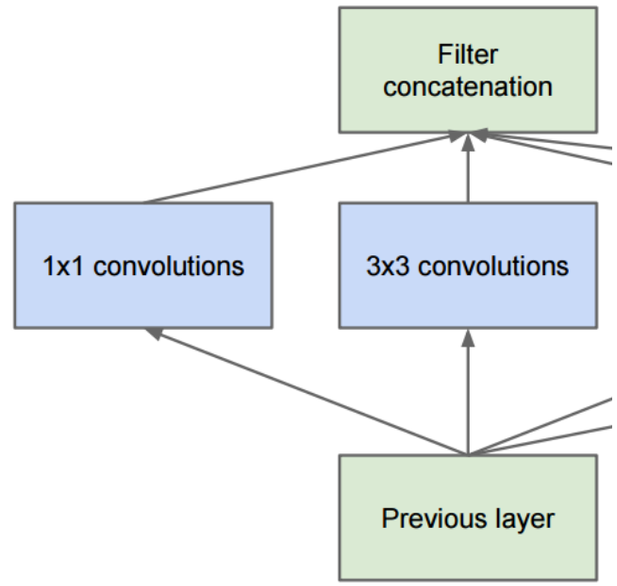
Szegedy et al. arxiv.org/pdf/1409.4842v1.pdf

Fancy structures

```
def SimpleFactory(data, ch_1x1, ch_3x3):
    global concat_cnt
    param = {}
    # 1x1
    param["kernel"] = (1, 1)
    param["num_filter"] = ch_1x1
    param["pad"] = (0, 0)
    param["stride"] = (1, 1)
    param["act_type"] = "relu"
    param["data"] = data
    conv1x1 = ConvFactory(**param)

    # 3x3
    param["kernel"] = (3, 3)
    param["num_filter"] = ch_3x3
    param["pad"] = (1, 1)
    conv3x3 = ConvFactory(**param)

    #concat
    concat = mx.symbol.Concat(*[conv1x1, conv3x3], name="concat%d" % concat_cnt)
    concat_cnt += 1
    return concat
```



https://github.com/dmlc/mxnet/blob/master/example/cifar10/cifar10.py

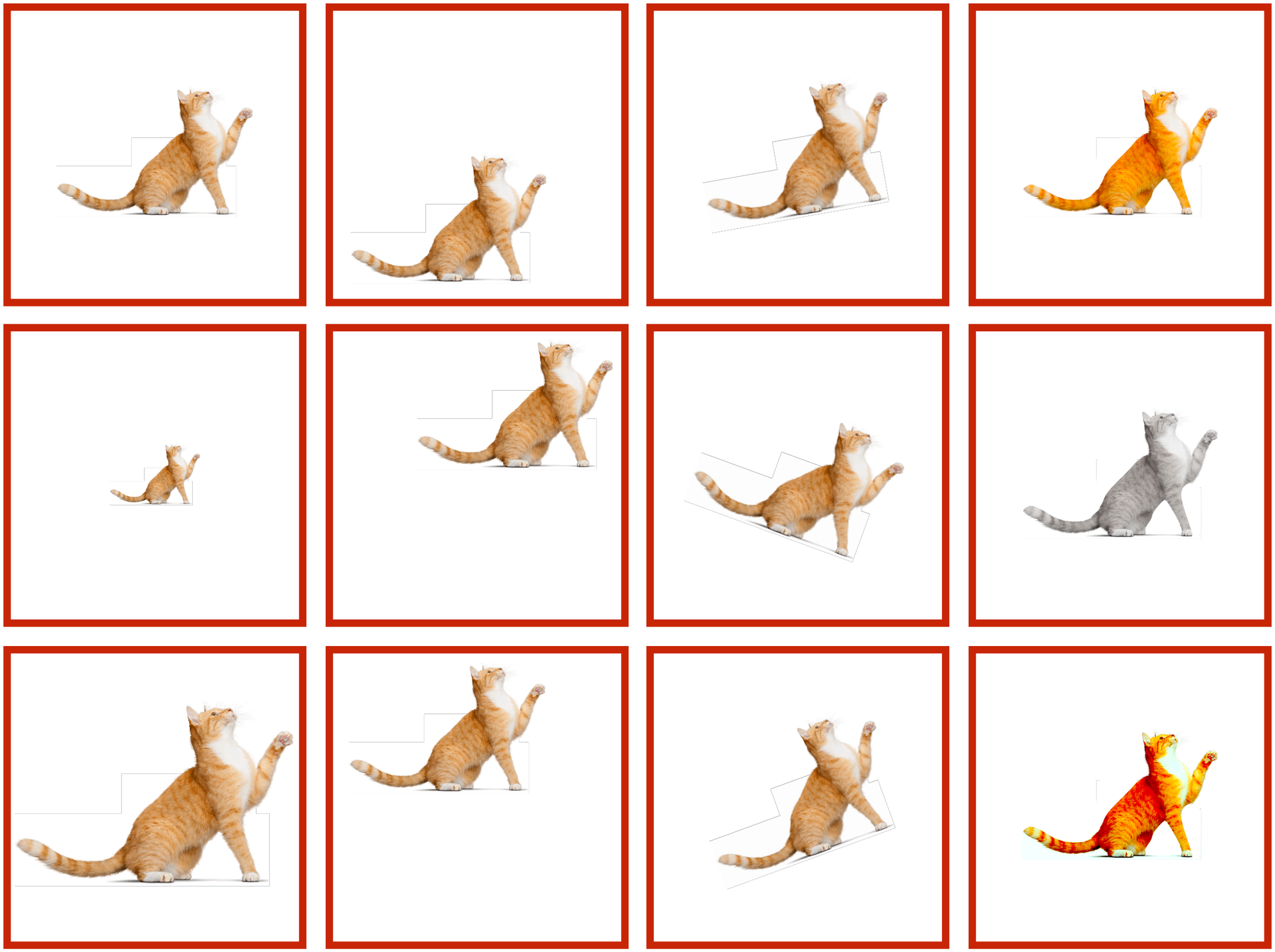
Convolutions alone cannot handle this!

scale

shift

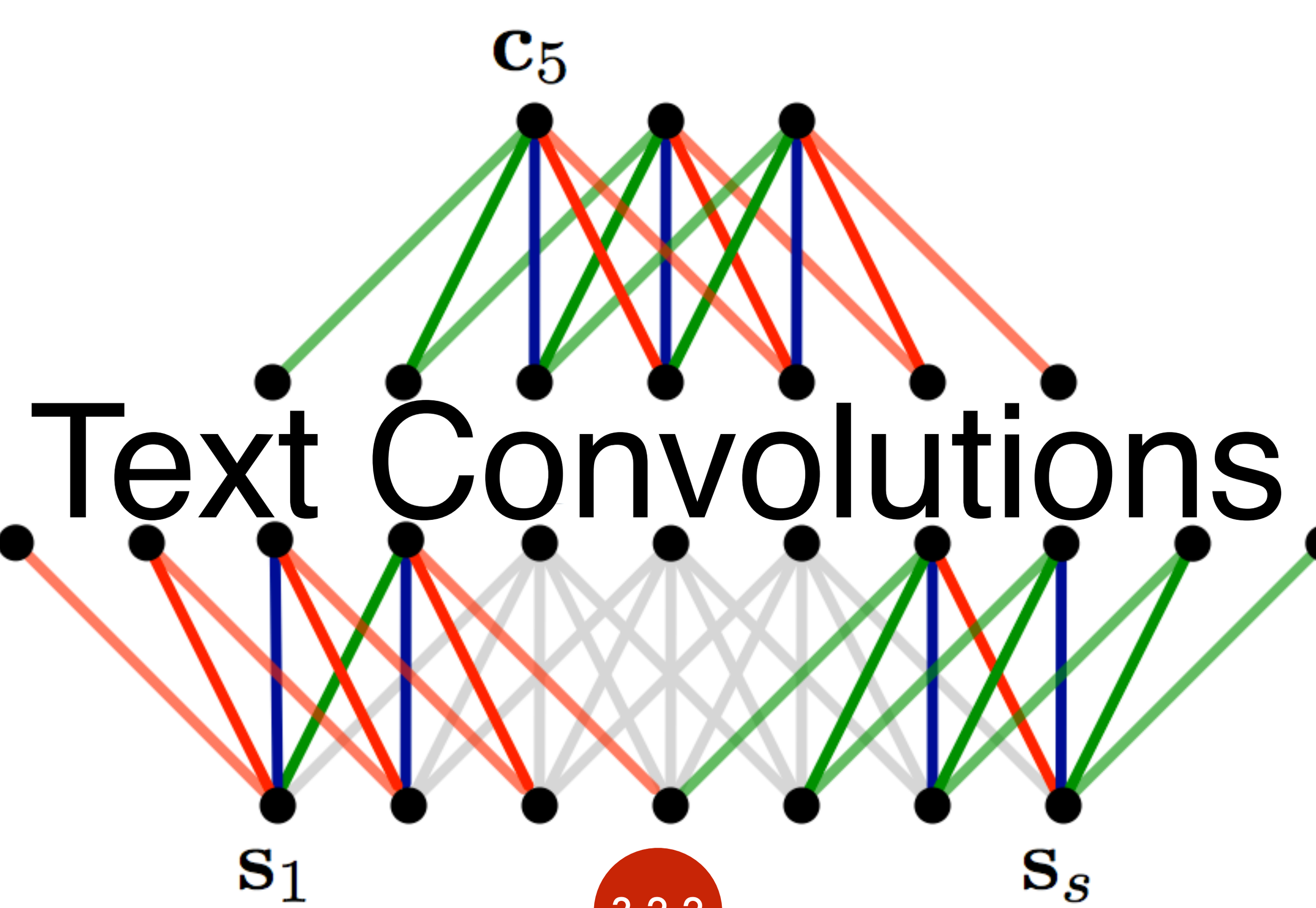
rotation

color space



Marianas Labs

Carnegie Mellon University

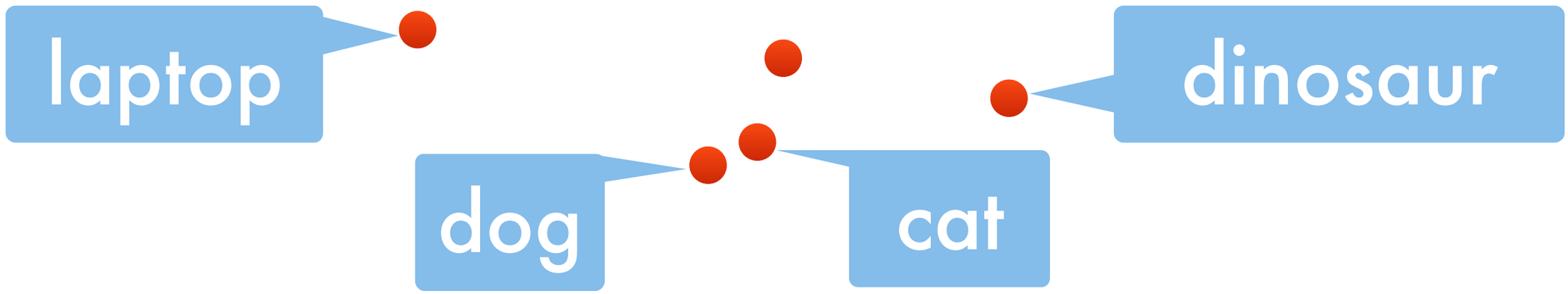


Text Convolutions

3.2.2

Convolutions on Text

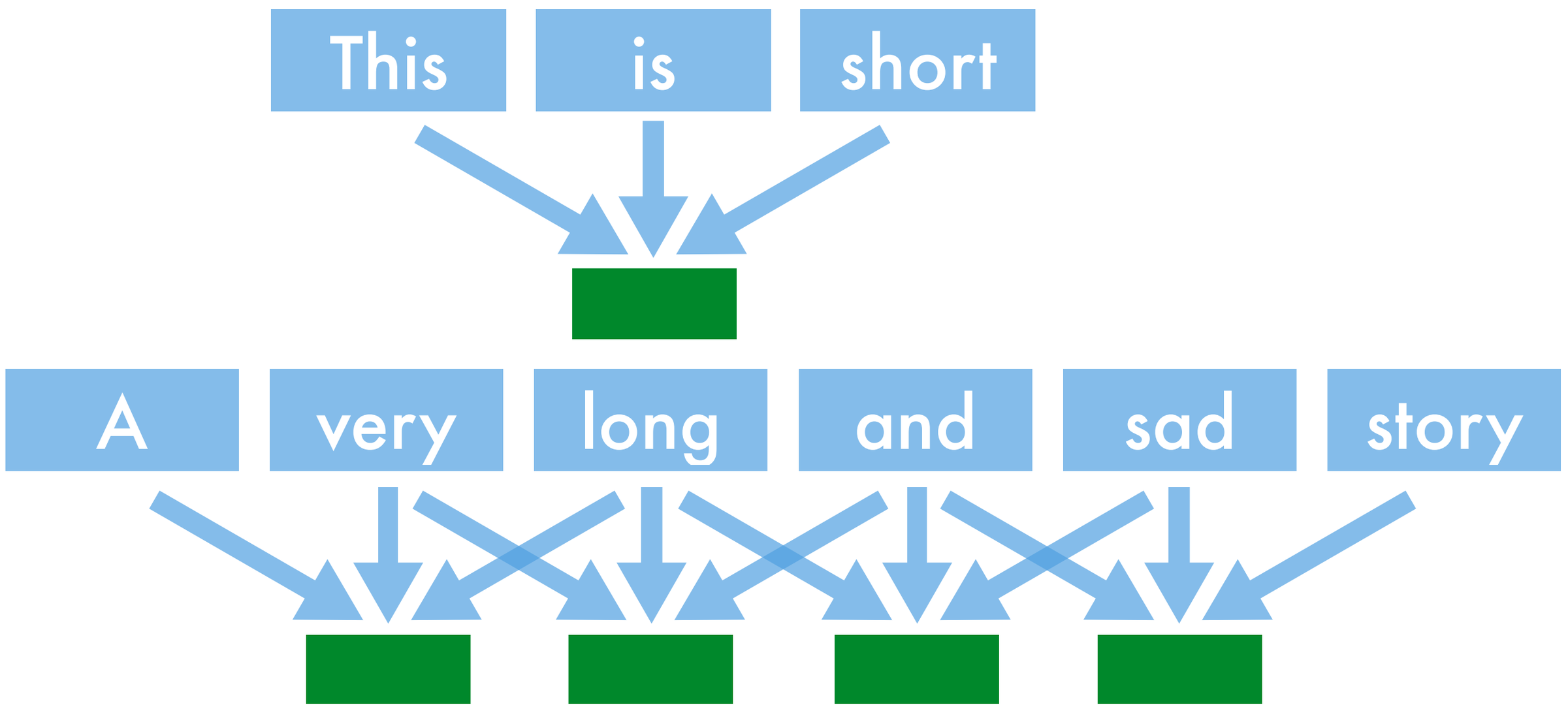
- Embed words into vector space



- Nonlinear function based on inputs
- What to do about sequential structure of text?
- What about typpos, omisions?

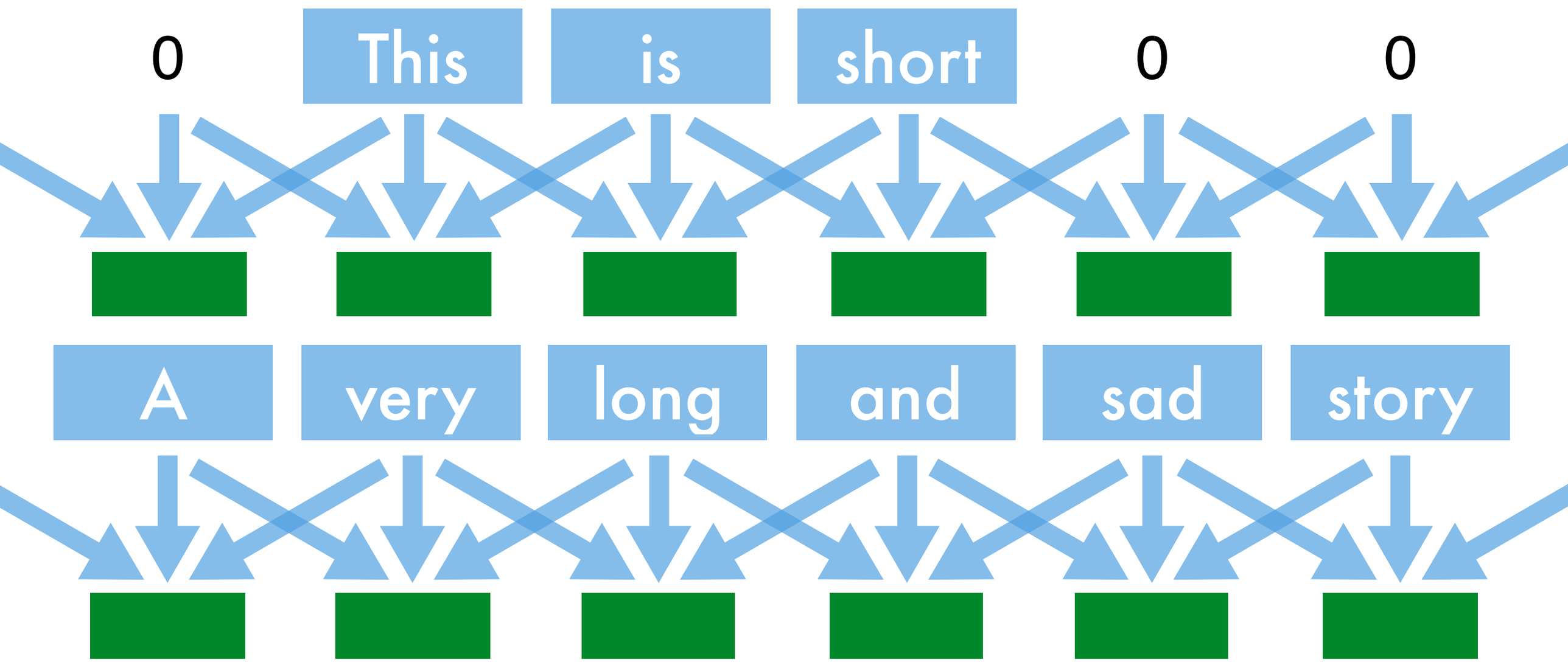
… this is a long sentence and it just keeps on going …

Outer Convolutions



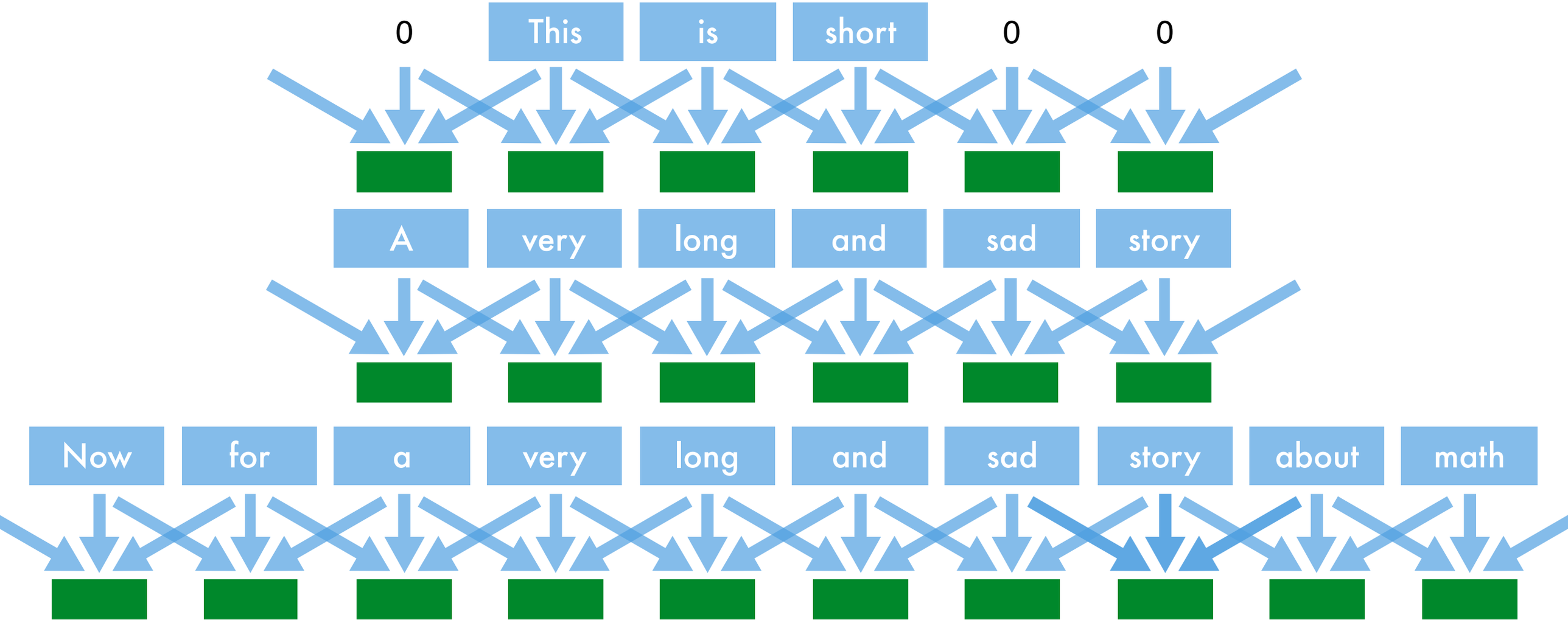
- Different length sequences lead to different size of convolutions.

Outer Convolutions



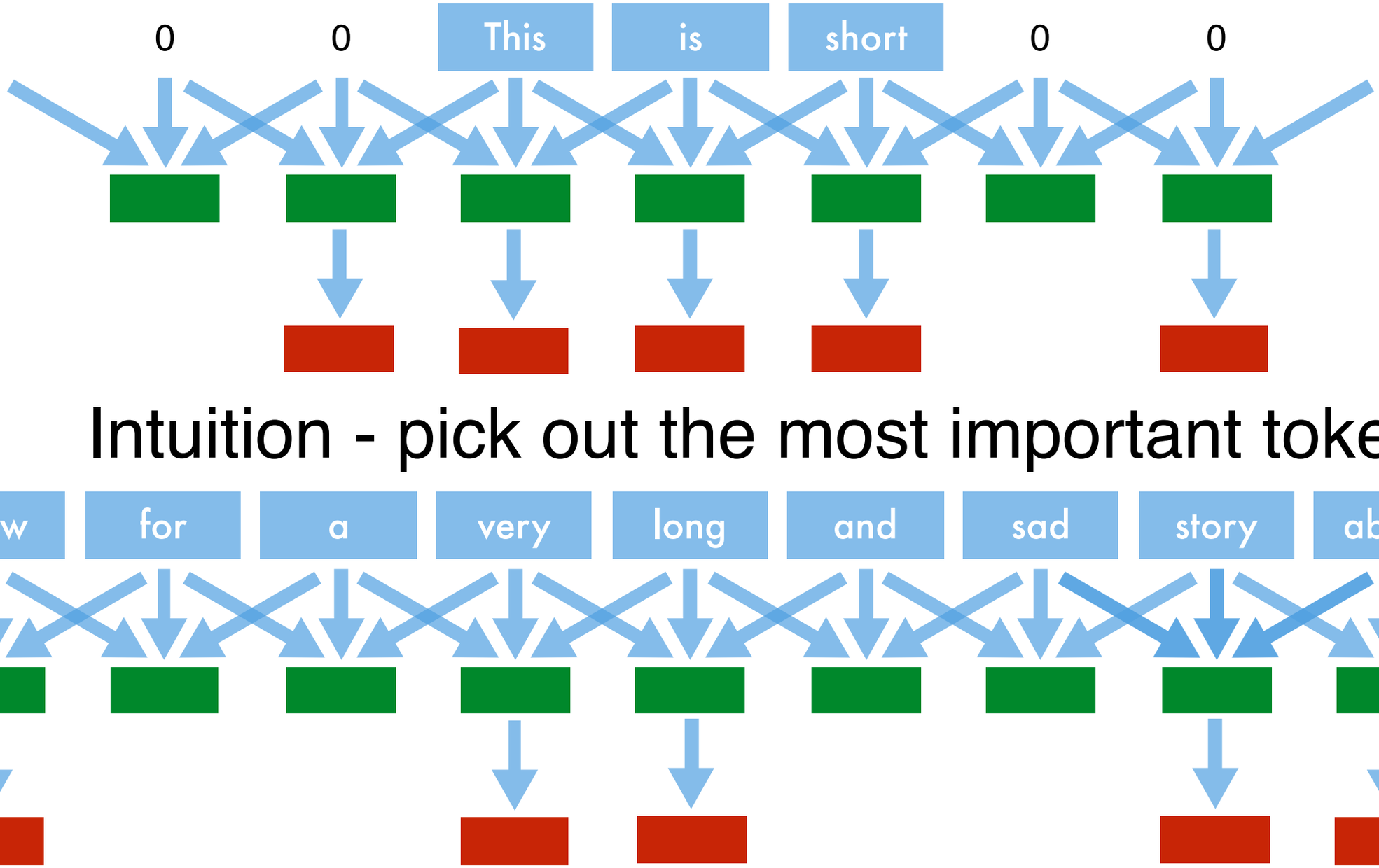
- Pad with zeros

Outer Convolutions

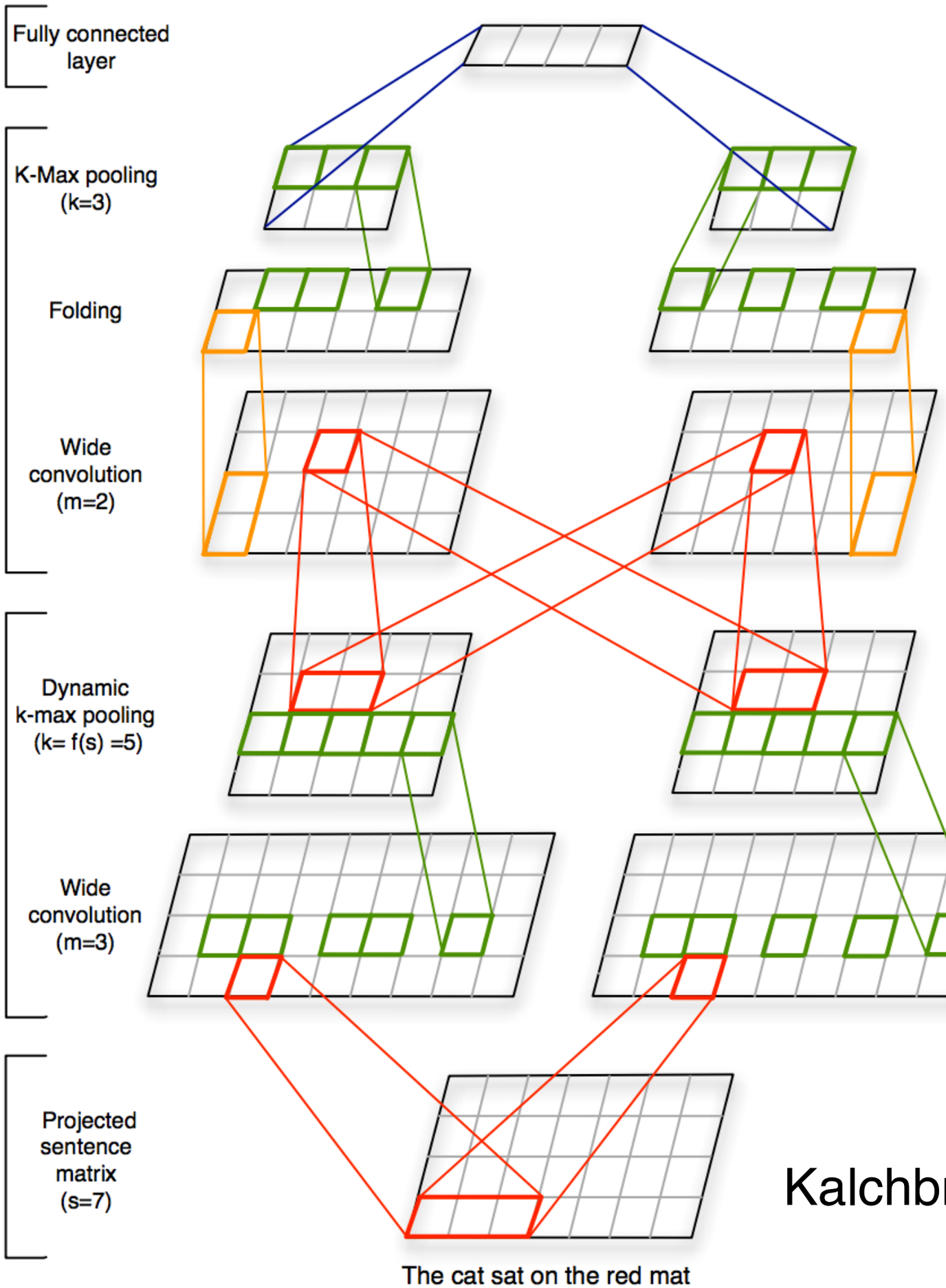


- Need to shorten sequence, e.g. max pooling. But insufficient if only pick the largest from sequence.

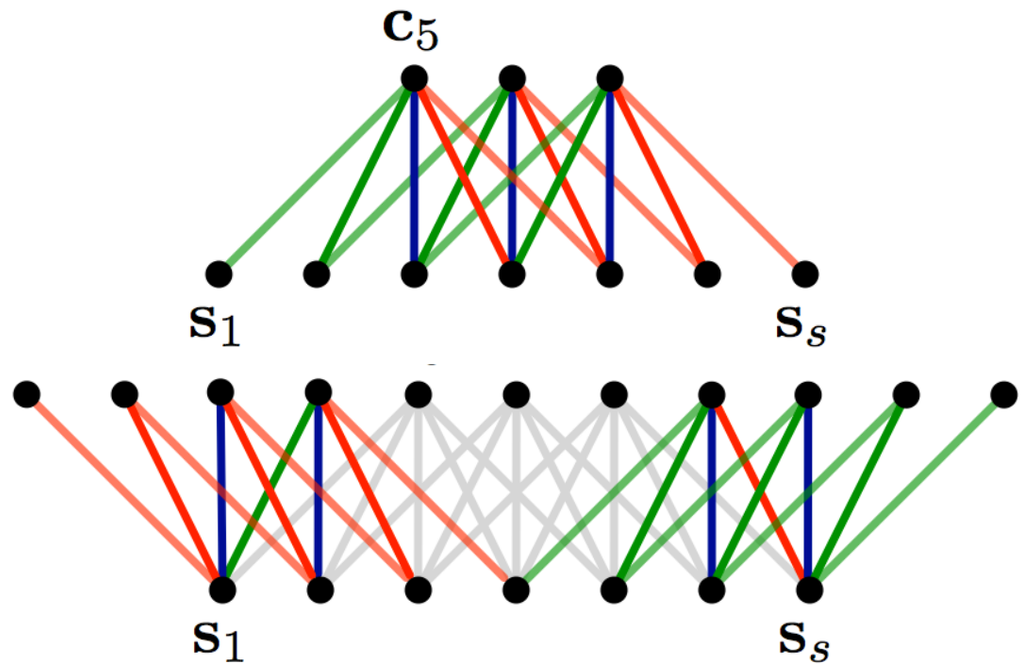
k-max Pooling



Convolutions on Text



narrow



wide

Kalchbrenner, Grefenstette, Blunsom ‘14

6 6 6 6 6 9 9 9

Approximate symmetries

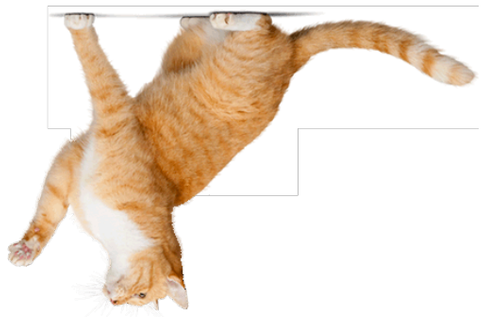
3.2.3

Marianas Labs

Carnegie Mellon University

Approximate Symmetries

- Image rotation



vs

- Text

  I like the flowers. Thanks. I like the flowers.

  I like the **red** flowers.

- Audio
  - Pitch and duration
  - Background noise

Features from group theory

- **Averaging**

$$f(x) \longrightarrow \bar{f}(x) := \sum_{g \in \mathcal{G}} f(g \circ x)$$

Sums over all transformations due to group

$$\bar{f}(g' \circ x) = \sum_{g \in \mathcal{G}} f(g \circ g' \circ x) = \sum_{g'' \in \mathcal{G}} f(g'' \circ x) = \bar{f}(x)$$

- **Maximum (puzzle - approximate by averaging)**

$$f(x) \longrightarrow \bar{f}(x) := \max_{g \in \mathcal{G}} f(g \circ x)$$

Again, invariance since $g \circ g' \in \mathcal{G}$

Features from group theory

- **Averaging**

$$f(x) \longrightarrow \bar{f}(x) := \sum_{g \in \mathcal{G}} f(g \circ x)$$

For loss functions

$$\bar{l}(x, y, f) := \sum_{g \in \mathcal{G}} l(g \circ x, y, f) - \Delta(g)$$

temper loss

- **Maximum**

$$f(x) \longrightarrow \bar{f}(x) := \max_{g \in \mathcal{G}} f(g \circ x)$$

$$\bar{l}(x, y, f) := \max_{g \in \mathcal{G}} l(g \circ x, y, f) - \Delta(g)$$

temper loss

Virtual Observations

- Summing over entire group is dangerous (6 vs 9)

$$\bar{l}(x, y, f) := \sum_{g \in \mathcal{G}} l(g \circ x, y, f) - \Delta(g)$$

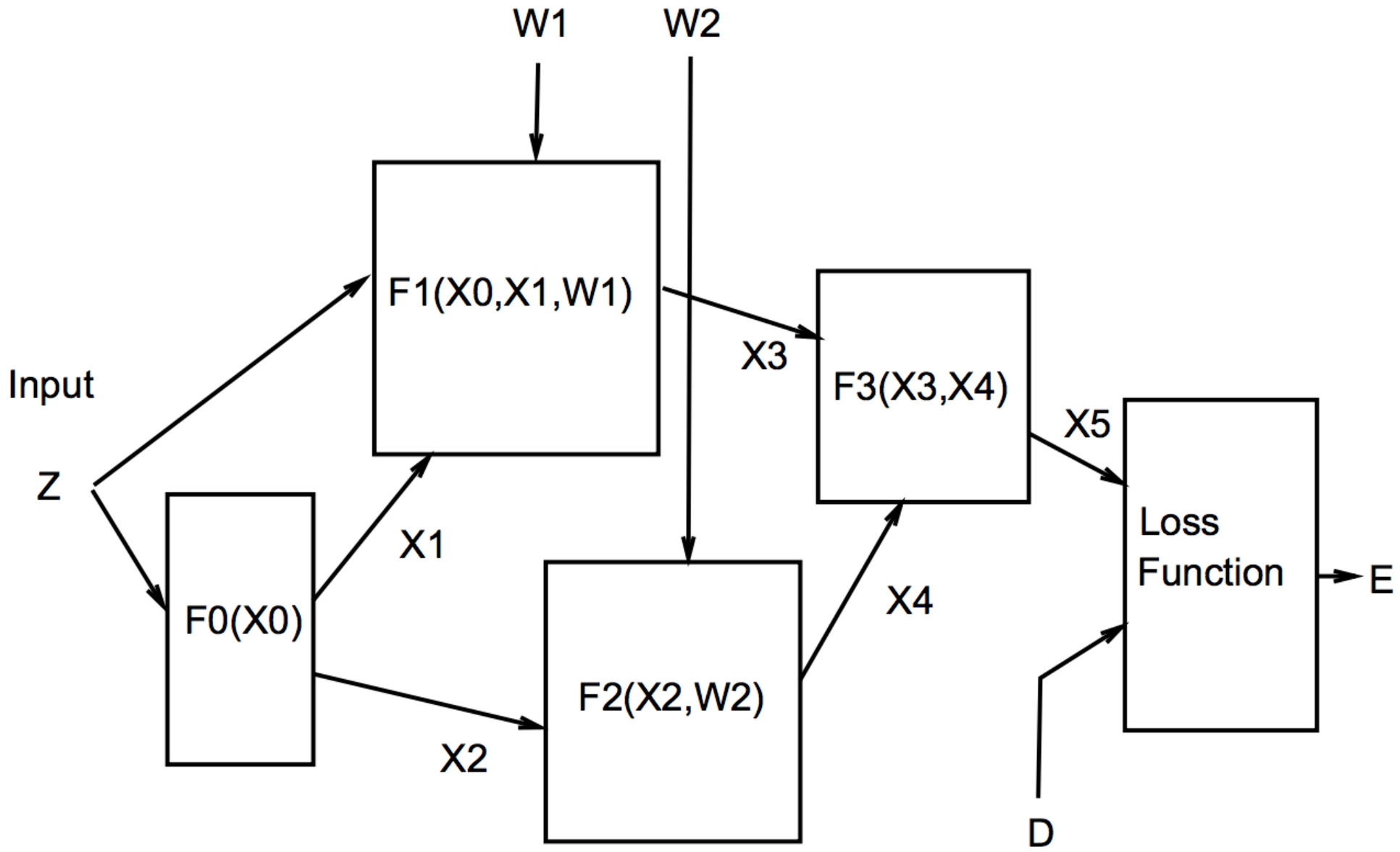
- Generate transformed instances (virtual SVs) only within ‘reasonable’ range

$$\bar{l}(x, y, f) := \sum_{g \in \mathcal{G}} p(g) l(g \circ x, y, f) - \Delta(g)$$

weighting

- E.g. Baidu speech recognizer (background noise), early Le Cun object classifier (clutter)

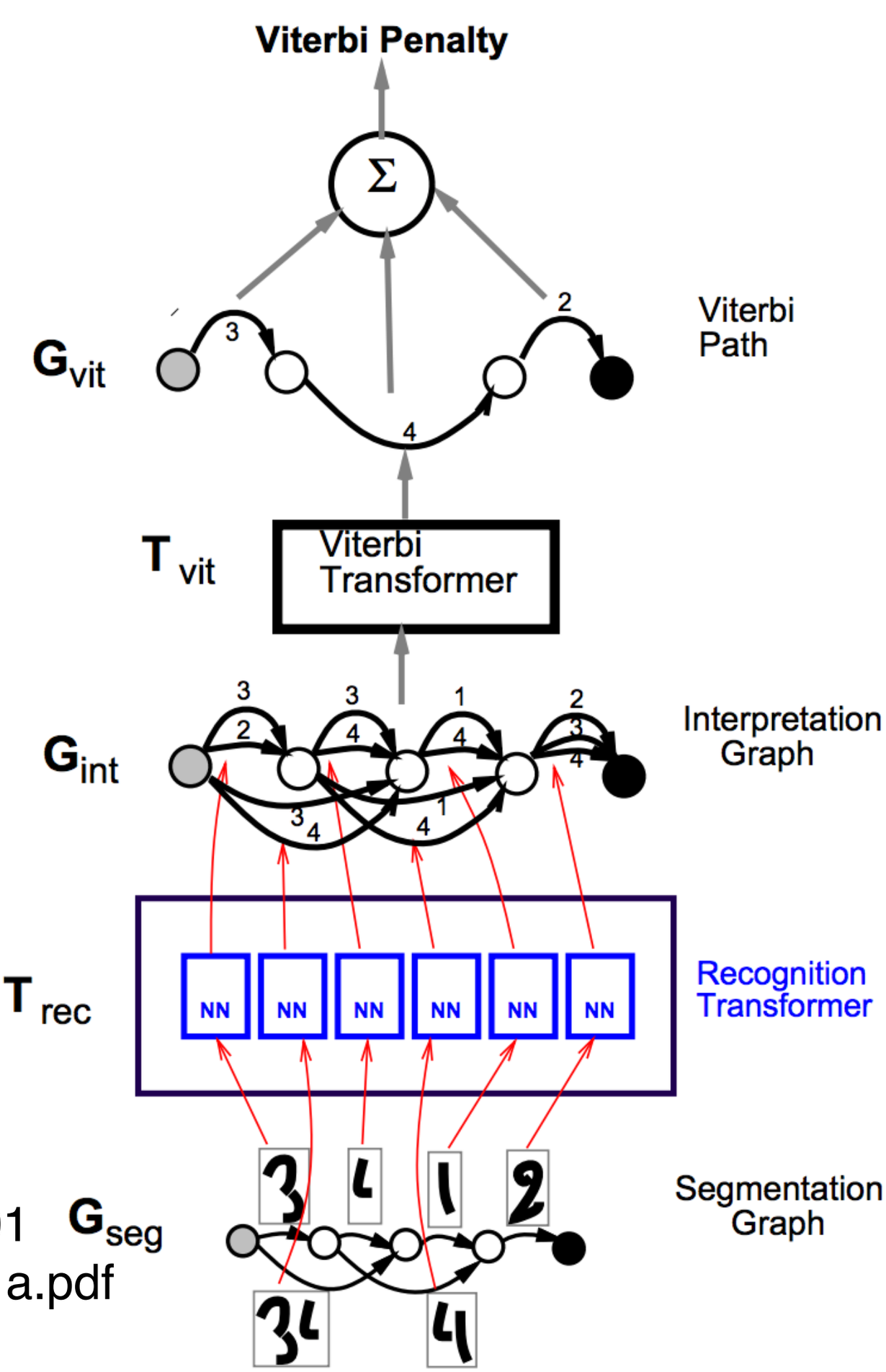
Backdrop for whole system training



Le Cun, Bottou, Bengio, Haffner, 2001
yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

Whole system training

- Layers need not be ‘neural networks’
  - Rankers
  - Segmenters
  - Finite state automata
- Jointly train a full OCR system



Le Cun, Bottou, Bengio, Haffner, 2001
yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

5. Advanced Objectives

Marianas Labs

Carnegie Mellon University

Recall - Regression

- Real valued y, quality measured by y - f(x)

$$l(y, f(x)) = |y - f(x)|$$ absolute value

$$l(y, f(x)) = \frac{1}{2}(y - f(x))^2$$ least mean squares

$$l(y, f(x)) = \max(0, |y - f(x)|)$$ ϵ-insensitive

$$l(y, f(x)) = \begin{cases} |y - f(x)| - 0.5 & \text{if } |y - f(x)| > 1 \\ 0.5(y - f(x))^2 & \text{otherwise} \end{cases}$$ Huber

- Vectorial data

$$l(y, f(x)) = \frac{1}{2}\|y - f(x)\|_2^2$$

Recall - Classification

- Binary y, e.g. {apples, oranges}
- Multiple categories, e.g. {red, green, blue}
- Ordinal relationship, e.g. {A, B, C, D, Fail}

$$l(y, f(x)) = \log\left(1 + e^{-yf(x)}\right) \quad \text{logistic}$$

$$l(y, f(x)) = \max(0, 1 - yf(x)) \quad \text{soft-margin}$$

$$l(y, f(x, \cdot)) = \log \sum_{y'} e^{f(x,y')} - f(x, y) \quad \text{multiclass logistic}$$

$$l(y, f(x, \cdot)) = \max_{y'} \left[f(x, y') + \Delta(y, y')\right] - f(x, y)$$

Multiclass Classification

- Multiclass classification (softmax)
  Multinomial exponential model

$$p(y|x) = \frac{e^{f(x,y)}}{\sum_{y'} e^{f(x,y')}}$$

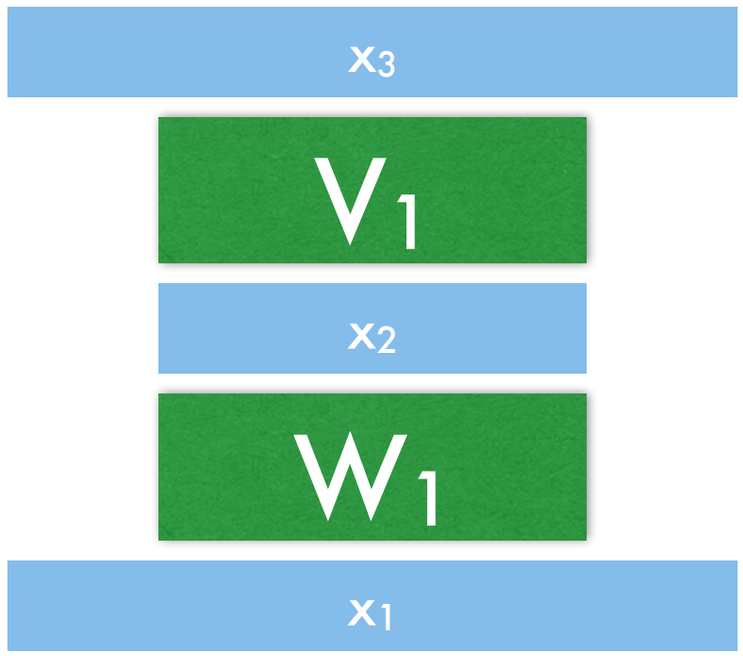
$$-\log p(y|x) = \log \sum_{y'} e^{f(x,y')} - f(x,y)$$

- Example - words, e.g. to predict the next word.

  Oh wow, I found my …
  {cat, dog, car, laptop, dinosaur, theorem …}

Principal Component Analysis

- Regress from observation to itself
- Lower-dimensional layer is bottleneck
- Linear transfer function



$$\underset{W,V}{\text{minimize}} \sum_i \|x_i - WVx_i\|_2^2$$

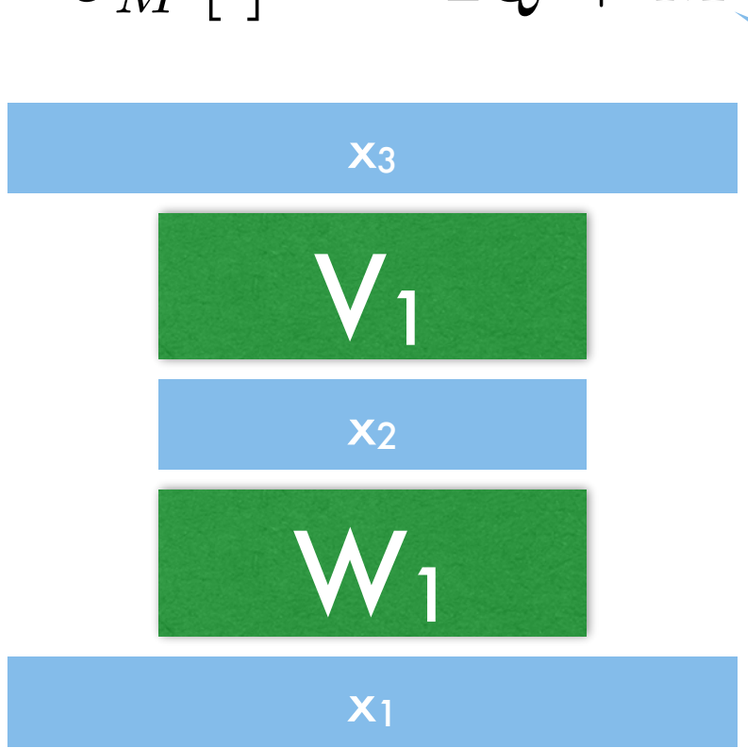
$$\sum_i \|x_i - WVx_i\|_2^2 = \operatorname{tr} XX^\top - 2 \operatorname{tr} XMX^\top + \operatorname{tr} XMM^\top X^\top$$

Principal Component Analysis

$$\sum_i \|x_i - WVx_i\|_2^2 = \operatorname{tr} XX^\top - 2\operatorname{tr} XWVX^\top + \operatorname{tr} XV^\top W^\top WVX^\top$$

$$= \operatorname{tr} Q - 2\operatorname{tr} M^\top Q + \operatorname{tr} M^\top QM$$

$$\partial_M [\cdot] = -2Q + M^\top Q + QM$$

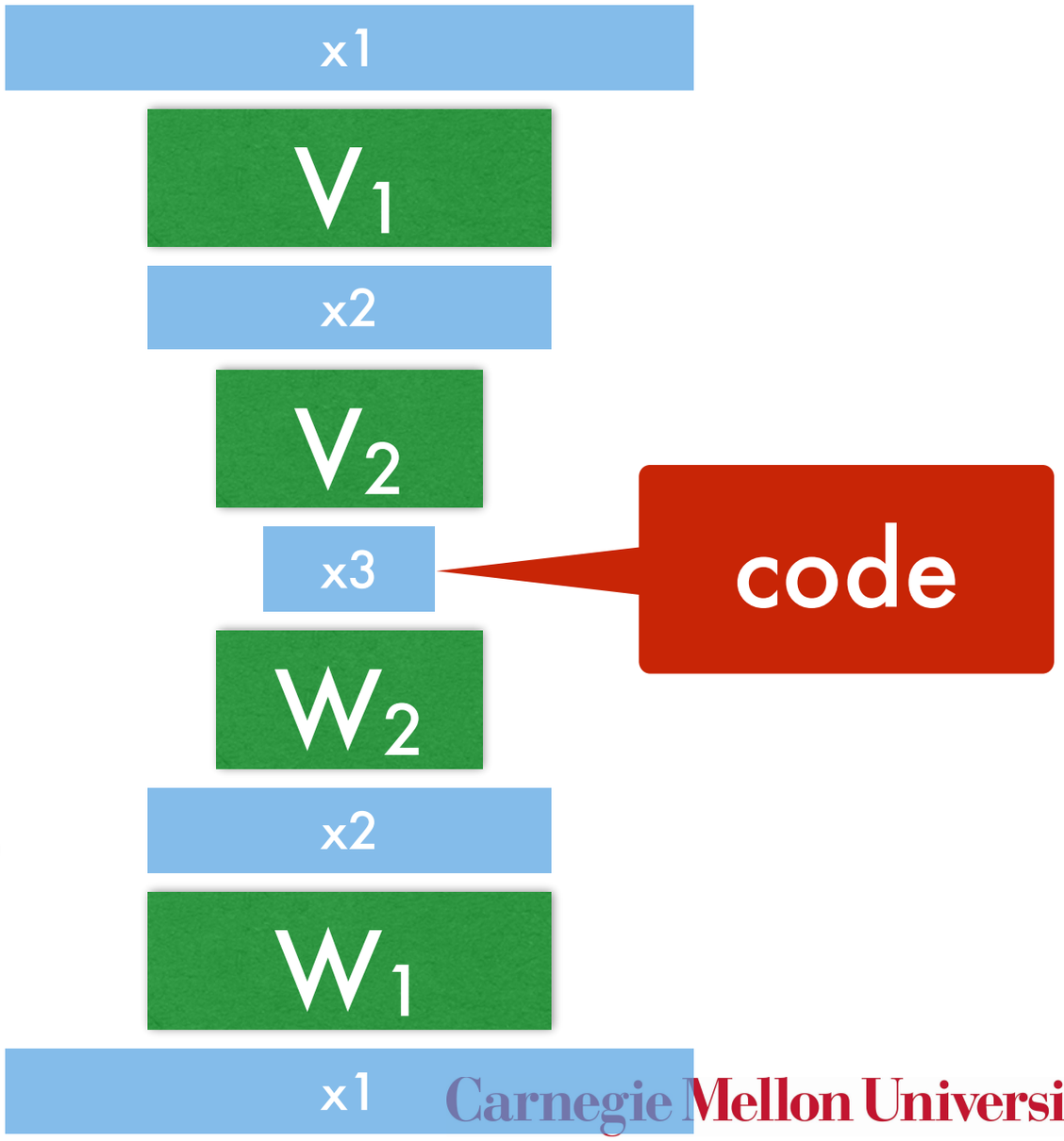


M symmetric
low rank constraint

Optimal M is projection. PCA!

Autoencoder

- Regress from observation to itself ($y_n = x_1$)
- Lower-dimensional layer is bottleneck
- **Nonlinearity per layer**
- Often trained iteratively (one layer at a time)
- Extracts approximate sufficient statistic of data (reconstruct x from itself)



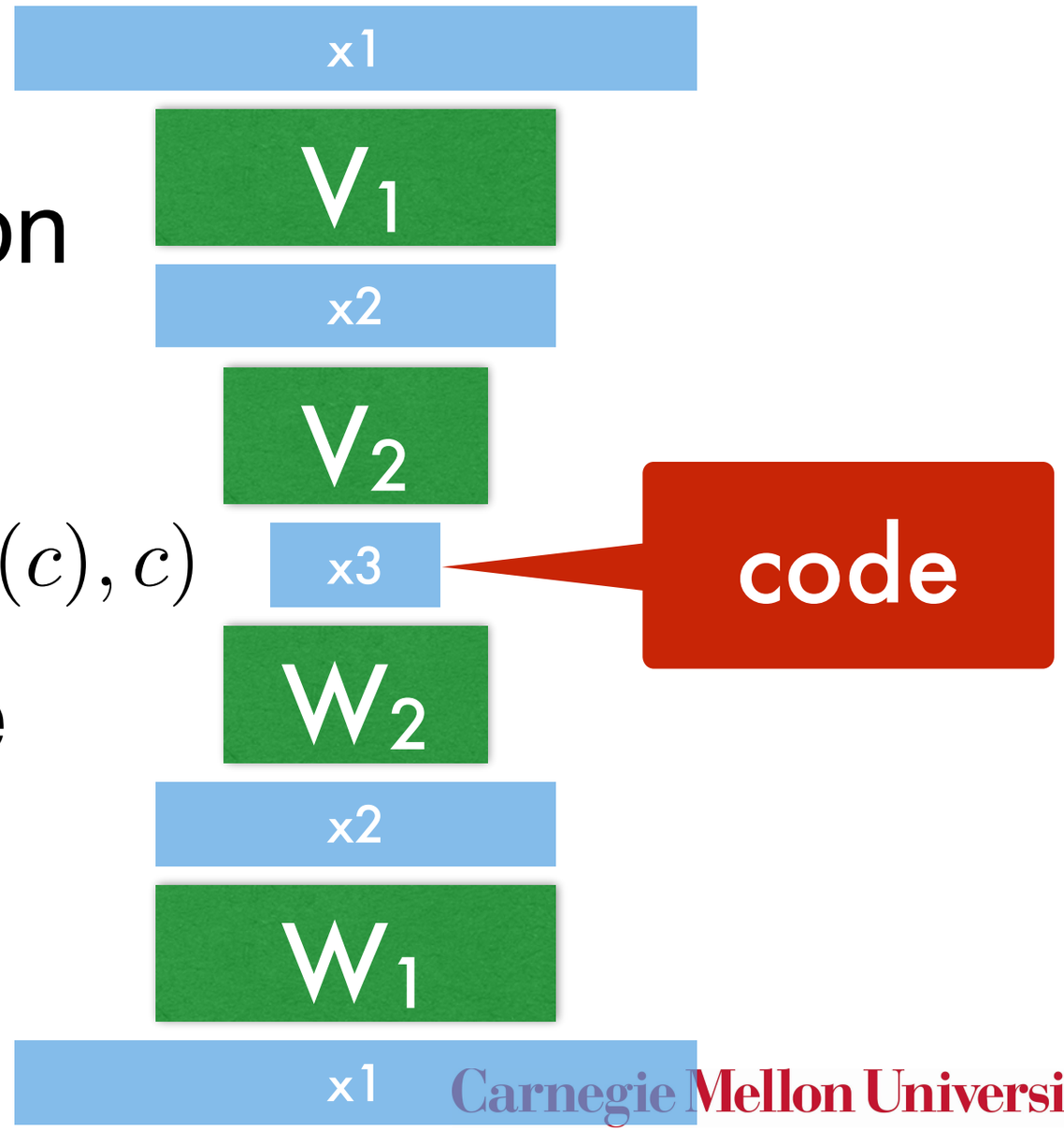
Autoencoder

- Encode x to code c
- Decode c to data x

- Code c has all information needed about x

$$p(y|x) = p(y|x, c(x)) \approx p(y|x(c), c)$$

- We can synthesize more data by finding new c' (density model for code)



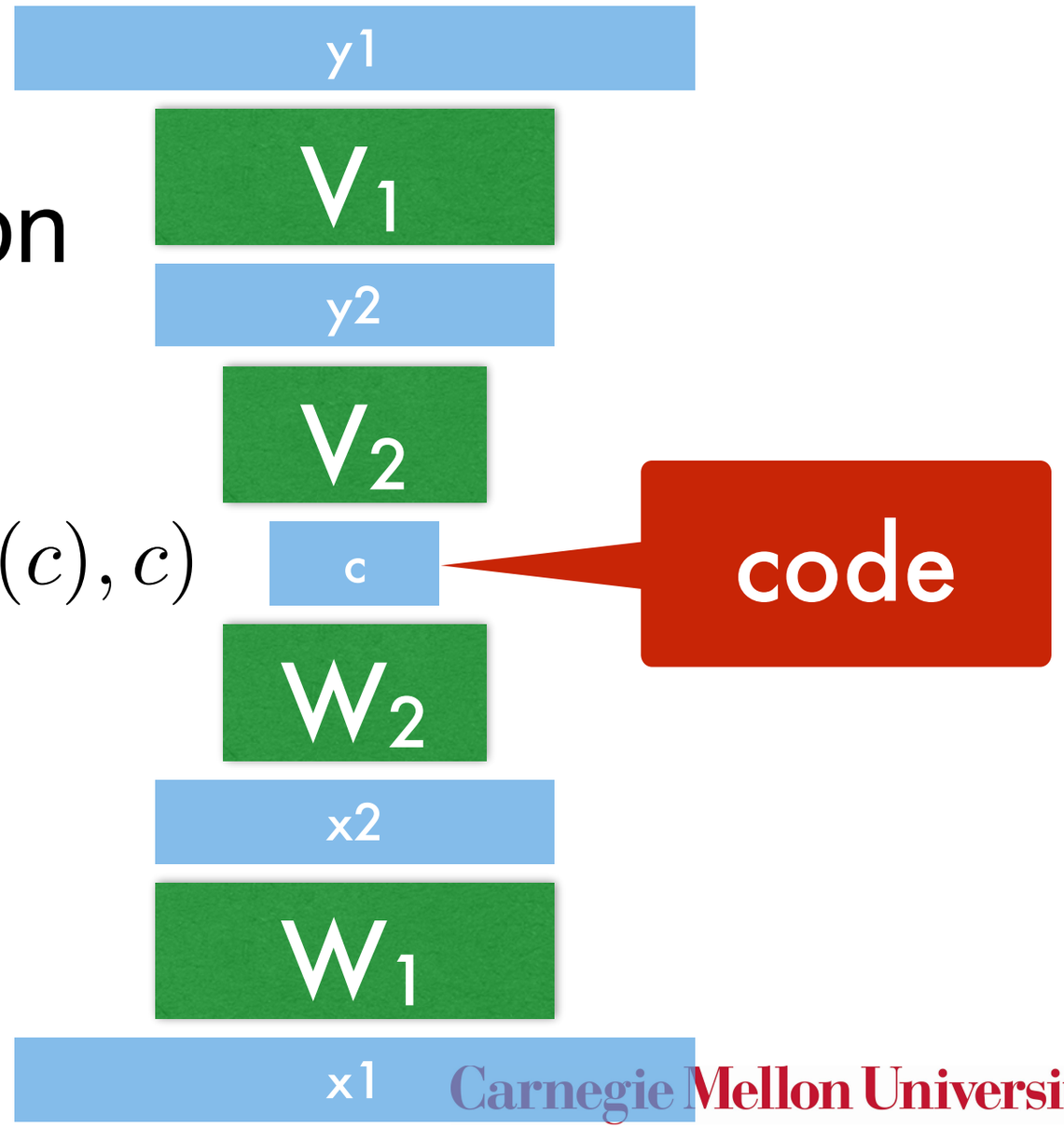
Translator

- Encode x to code c
- Decode c to data y

- Code c has all information needed about x

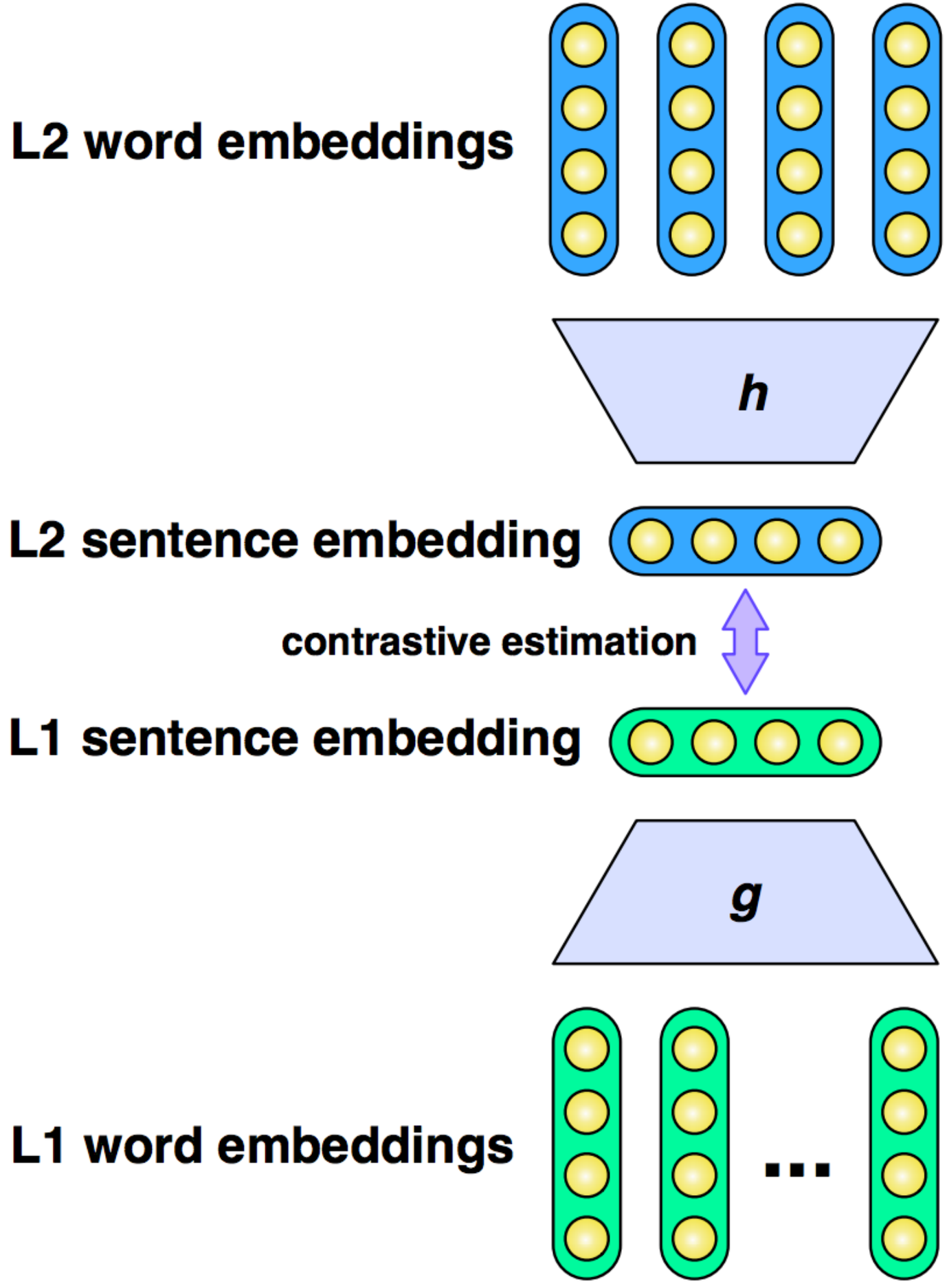
$$p(y|x) = p(y|x, c(x)) \approx p(y|x(c), c)$$

- We can synthesize new data y' by encoding new data x' via c(x')



‘Synesthesia’

- Different data sources
  - Images and captions
  - Natural language queries and SQL queries
  - Movies and actions
- Generative embedding for both entities
- Minimize distance between pairs
- Need to prevent clumping all together

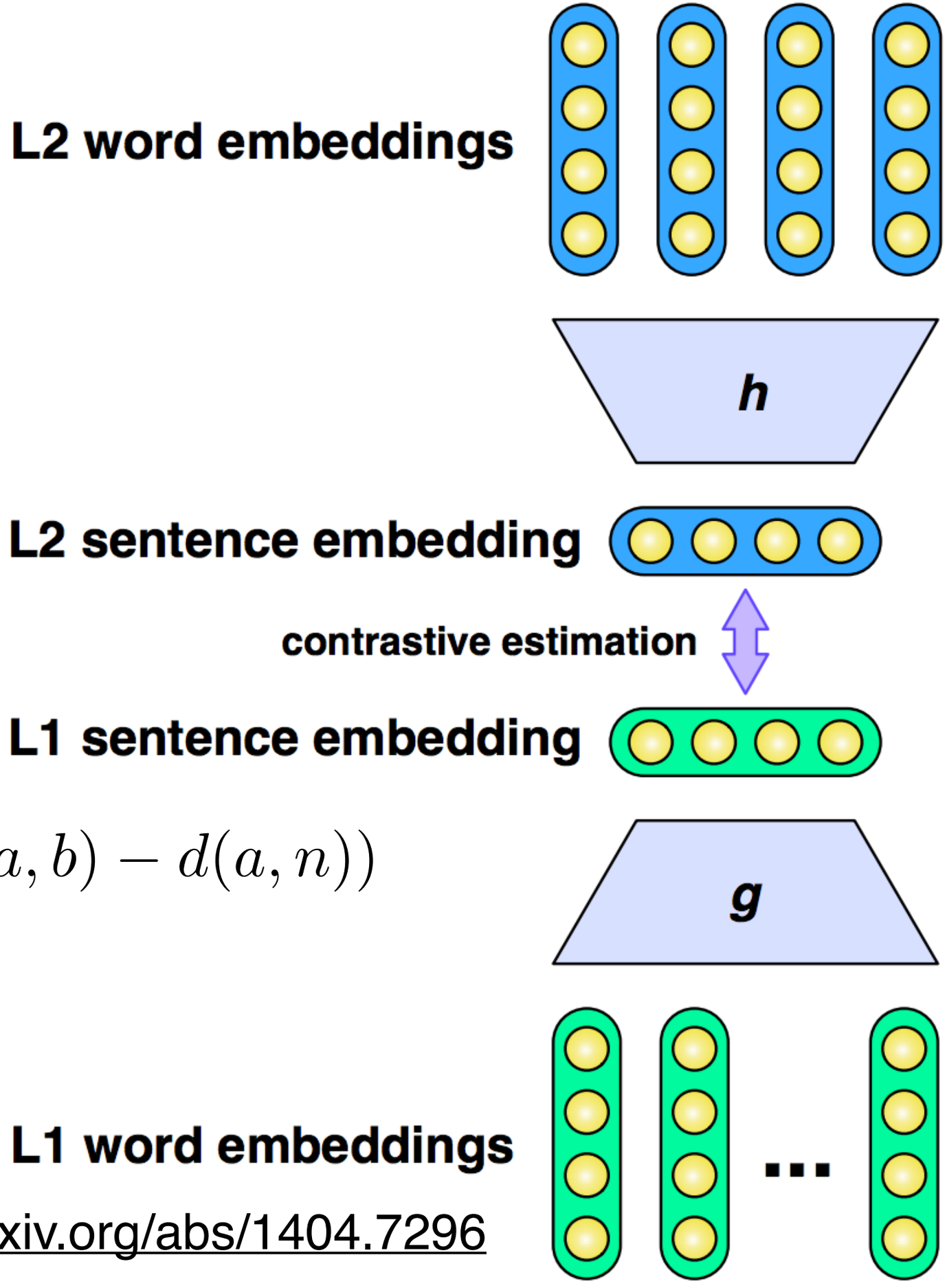


‘Synesthesia’

- Different data sources
  - Images and captions
  - Natural language queries and SQL queries
  - Movies and actions

$$\max(0, \text{margin} + d(a, b) - d(a, n))$$

large margin of similarity



Grefenstette et al, 2014, arxiv.org/abs/1404.7296

In a Nutshell

Marianas Labs

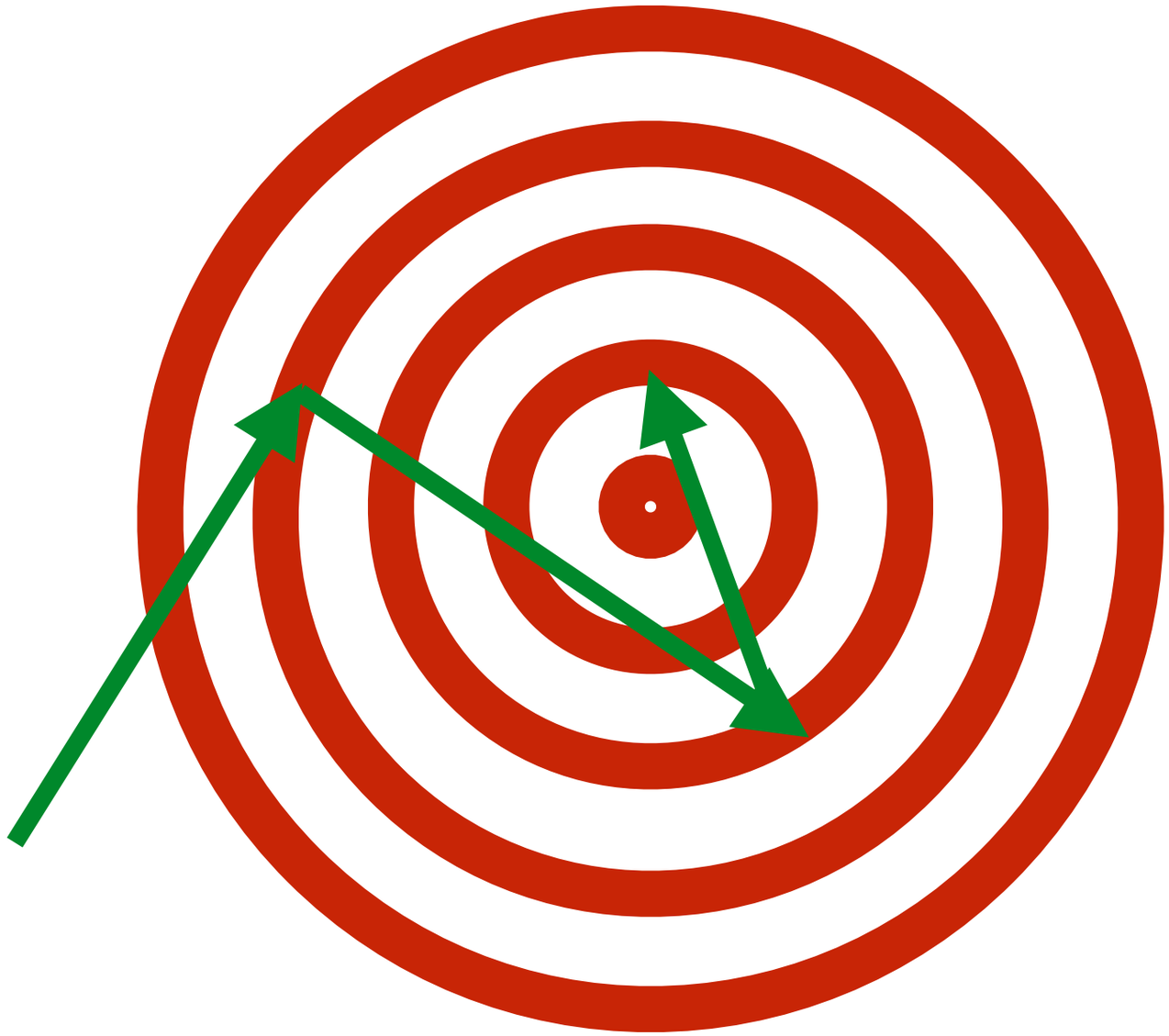
Carnegie Mellon University

Key Components

- **Backprop**
  Optimize whole system by propagating gradients
- **Layers**
  Adapted to data type and invariance (+ macros)
- **Nonlinearities (max pooling, k-max)**
  Select subset of attributes (algebraic IF)
- **Loss Function**
  - Multiclass, large margin, regression
  - Autoencoder, translator

Outline

- **Basics**
  - Perceptron (and convergence rule)
  - Stochastic gradient descent
  - Loss functions and objectives
- **Deep Networks**
  - Layers and Invariances
  - Advanced objectives
- **State and Structure**
  - Optimization
  - Autoregressive Models & Hidden State
  - Toolkits



6. Optimization

Marianas Labs

Carnegie Mellon University

Recall - Stochastic Gradient Descent

- Update parameters according to

$$w_{t+1} \leftarrow w_t - \eta_t \partial_{f(x_i)} l(y_i, f(x_i)) \partial_w f(x_i)$$

  - Rate of decay for learning rate
    - Adjust rate for each layer
    - Adjust each parameter individually
    - Minibatch size, momentum terms
- Lots of things that can (should) be adjusted (Bayesian optimization, e.g. Spearmint, MOE)

Senior, Heigold, Ranzato and Yang, 2013
http://static.googleusercontent.com/media/research.google.com/en/us/pubs/archive/40808.pdf

Minibatch

- Aggregate gradients before applying them
  Small number of instances (x_i, y_i) … (x_{i+b}, y_{i+b})
  - Reduces variance in gradients
  - Better for vectorization (GPUs)
    vector, vector < vector, matrix < matrix, matrix

$$\langle x, x' \rangle \;<\; Mx \;<\; MX$$

  - Large minibatch may need large memory (and slow updates).
- Magic numbers are 64 to 256 on GPUs

Senior, Heigold, Ranzato and Yang, 2013
http://static.googleusercontent.com/media/research.google.com/en/us/pubs/archive/40808.pdf

Learning rate decay

- **Constant**
  (requires schedule for piecewise constant, tricky)
- **Useful hack**
  Constant until no more improvement on validation set
- **Polynomial decay**

$$\eta(t) = \frac{\alpha}{(\beta + t)^\gamma}$$

  Recall exponent of 0.5 for conventional SGD, 1 for strong convexity. Bottou picks 0.75
- **Exponential decay**

$$\eta(t) = \alpha e^{-\beta t}$$

  not recommended since quite aggressive

AdaGrad

- **Adaptive learning rate** (preconditioner)

$$\eta_{ij}(t) = \frac{\eta_0}{\sqrt{K + \sum_t g_{ij}^2(t)}}$$

- For directions with large gradient, decrease learning rate aggressively to avoid instability
- If gradients start vanishing, learning rate decrease reduces, too
- Local variant

$$\eta_{ij}(t) = \frac{\eta_t}{\sqrt{K + \sum_{t'=t-\tau}^{t} g_{ij}^2(t')}}$$

time window

Duchi, Hazan, Singer, 2010
http://www.magicbroom.info/Papers/DuchiHaSi10.pdf

Momentum

- Average over recent gradients
  - Helps with local minima
  - Flat (noisy) gradients

momentum

$$m_t = (1 - \lambda)m_{t-1} + \lambda g_t$$

$$w_t \leftarrow w_t - \eta_t g_t - \tilde{\eta}_t m_t$$

- Can lead to oscillations for large momentum
- Nesterov's accelerated gradient

$$m_{t+1} = \mu m_t + \epsilon g(w_t - \mu m_t)$$

$$w_{t+1} = w_t - m_{t+1}$$

Capacity control

- Minimizing loss can lead to overfitting
- **Weight decay**

$$w_t \leftarrow w_t - \eta_t g_t$$

$$w_t \leftarrow (1 - \lambda) w_t - \eta_t g_t$$

prevents parameters from diverging

- **Parameter clipping**
  - Overheated GPU
  - Numerical instabilities
  - Hacky but necessary (e.g. deep architectures)

Dropout

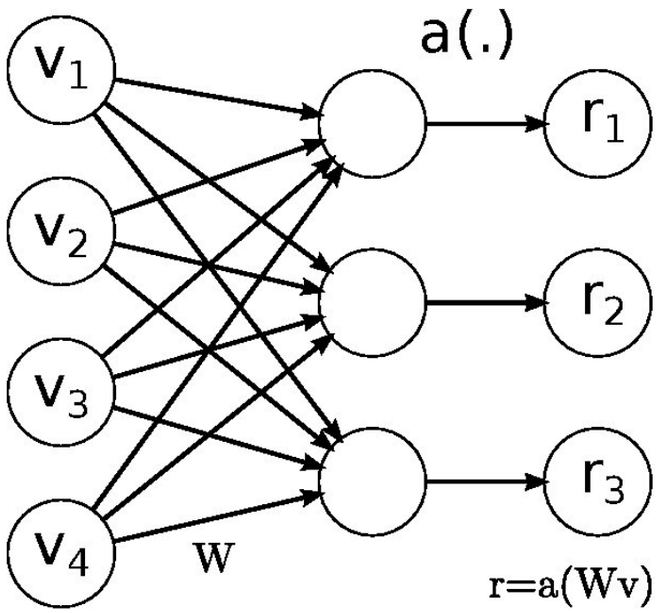
- **Avoid parameter sensitivity**
  (small changes in value shouldn't change result)
- **Distributed representation**
  (information carried by more than 1 dimension)
- **Randomized sparsification**

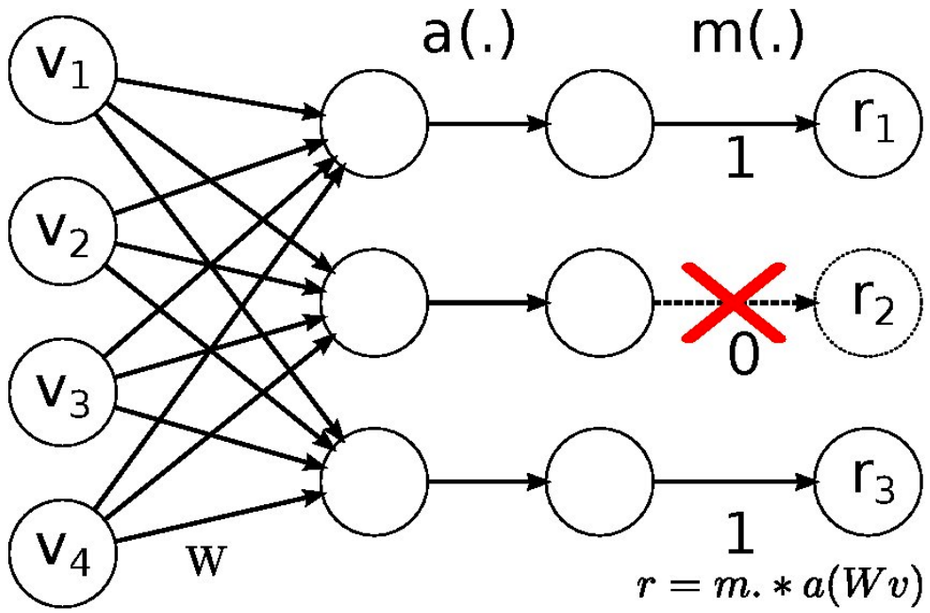
$$y_{ti} = \xi_{ti} y_{ti} \text{ where } \begin{cases} \Pr(\xi_{ti} = \pi^{-1}) & = \pi \\ \Pr(\xi_{ti} = 0) & = 1 - \pi \end{cases}$$

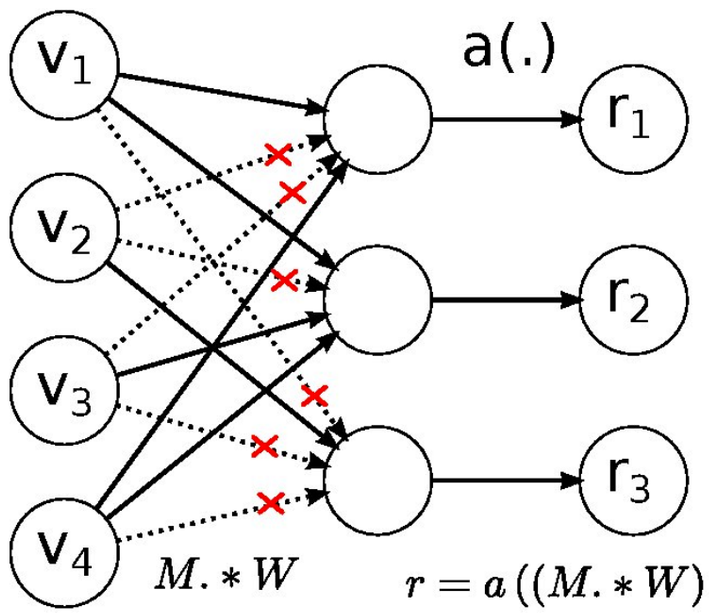
- Same trick works for matrix W, too.
  DropConnect gives slightly better performance.
  http://cs.nyu.edu/~wanli/dropc/

Srivastava, Hinton, Krizhevski, Sutskever, Salakhutdinov
http://jmlr.org/papers/v15/srivastava14a.html

Dropout & DropConnect







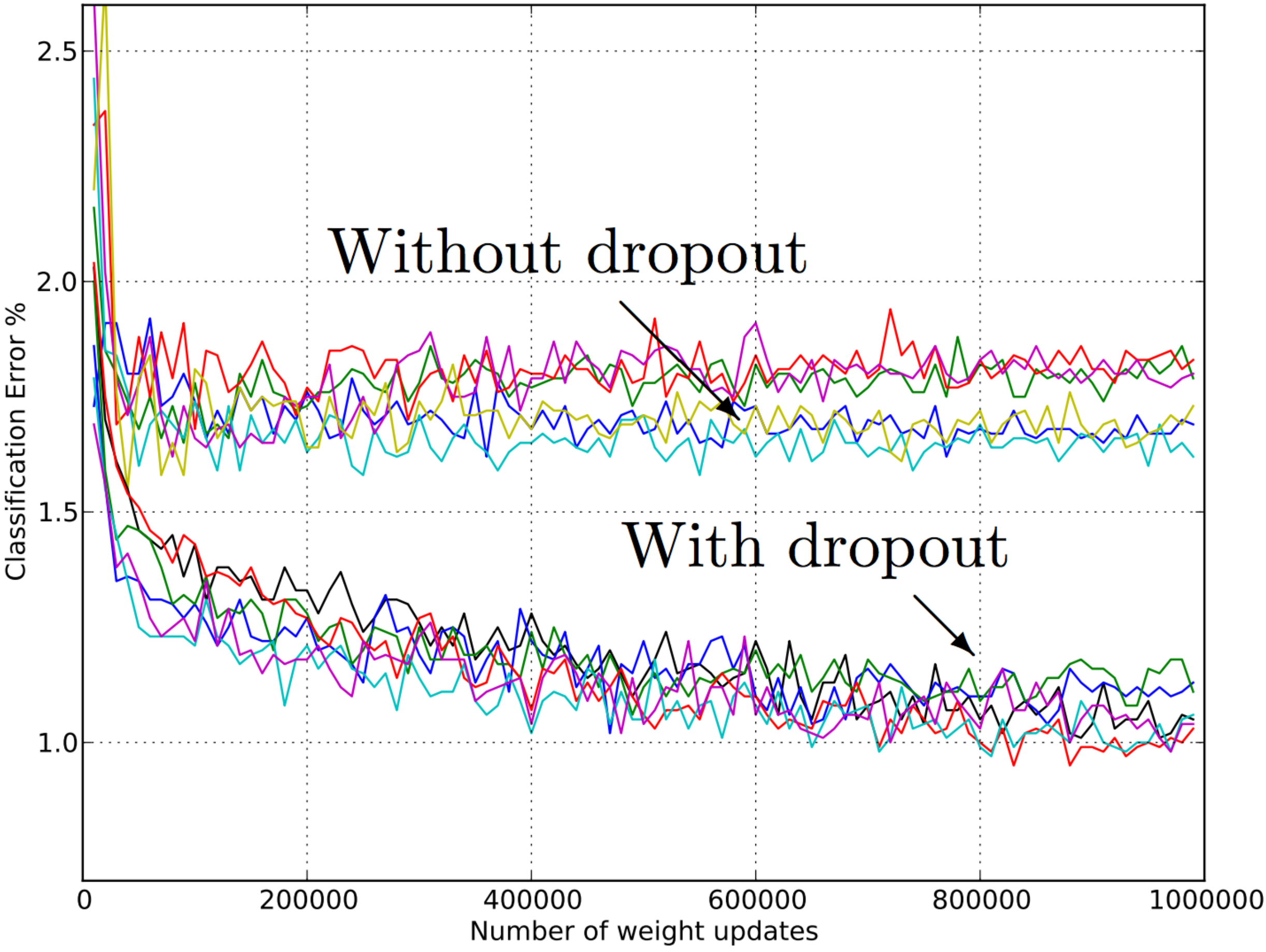
Regular

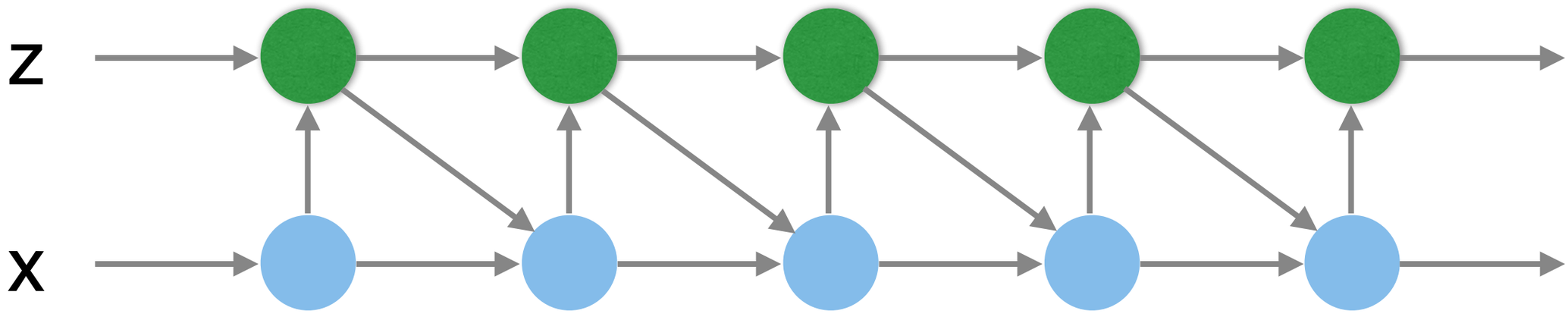
Dropout

DropConnect

Marianas Labs

Carnegie Mellon University

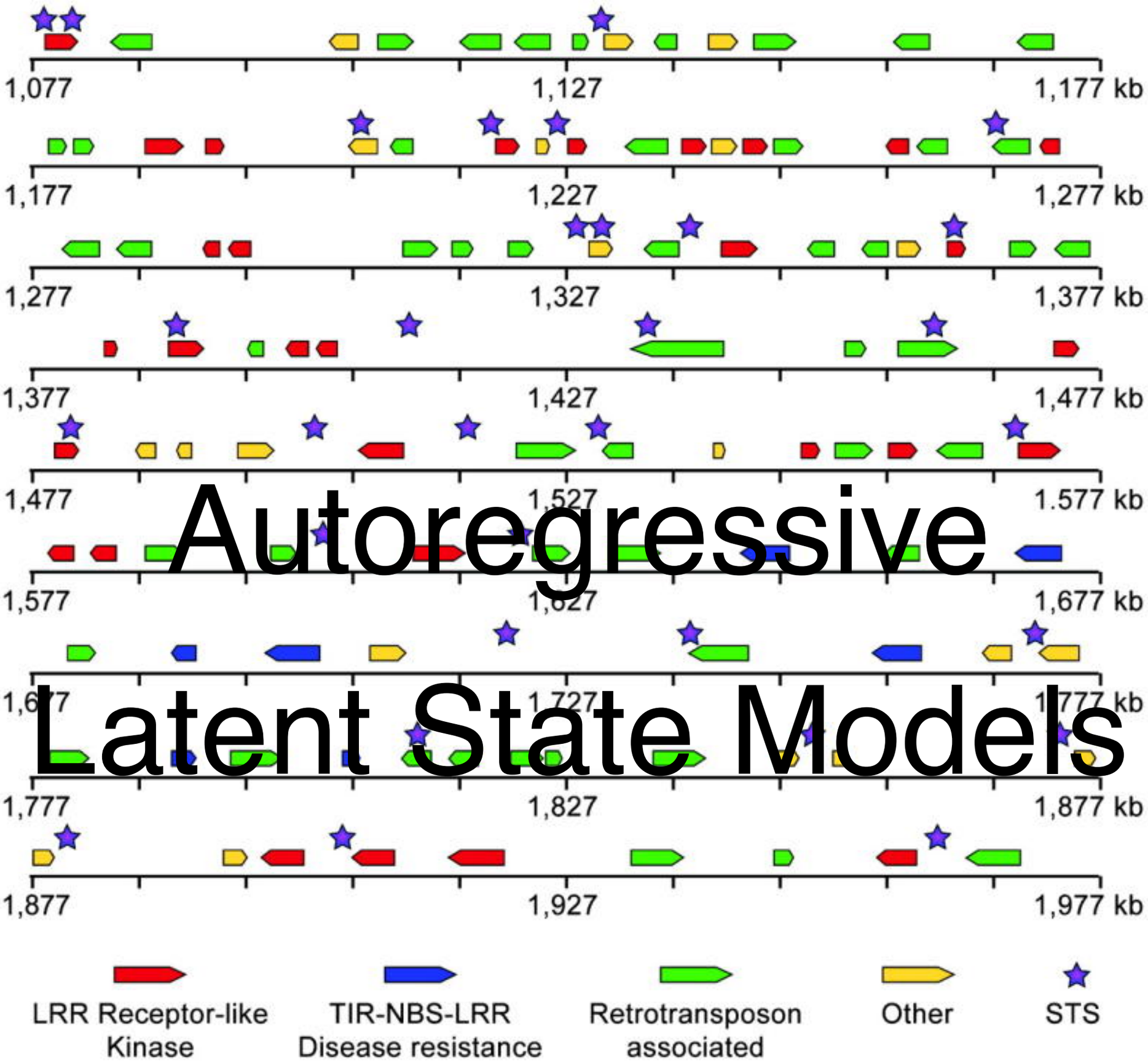

Without dropout
With dropout
Classification Error %
2.5
2.0
1.5
1.0
0
200000
400000
600000
800000
1000000
Number of weight updates



7. Memory & State

Marianas Labs

Carnegie Mellon University



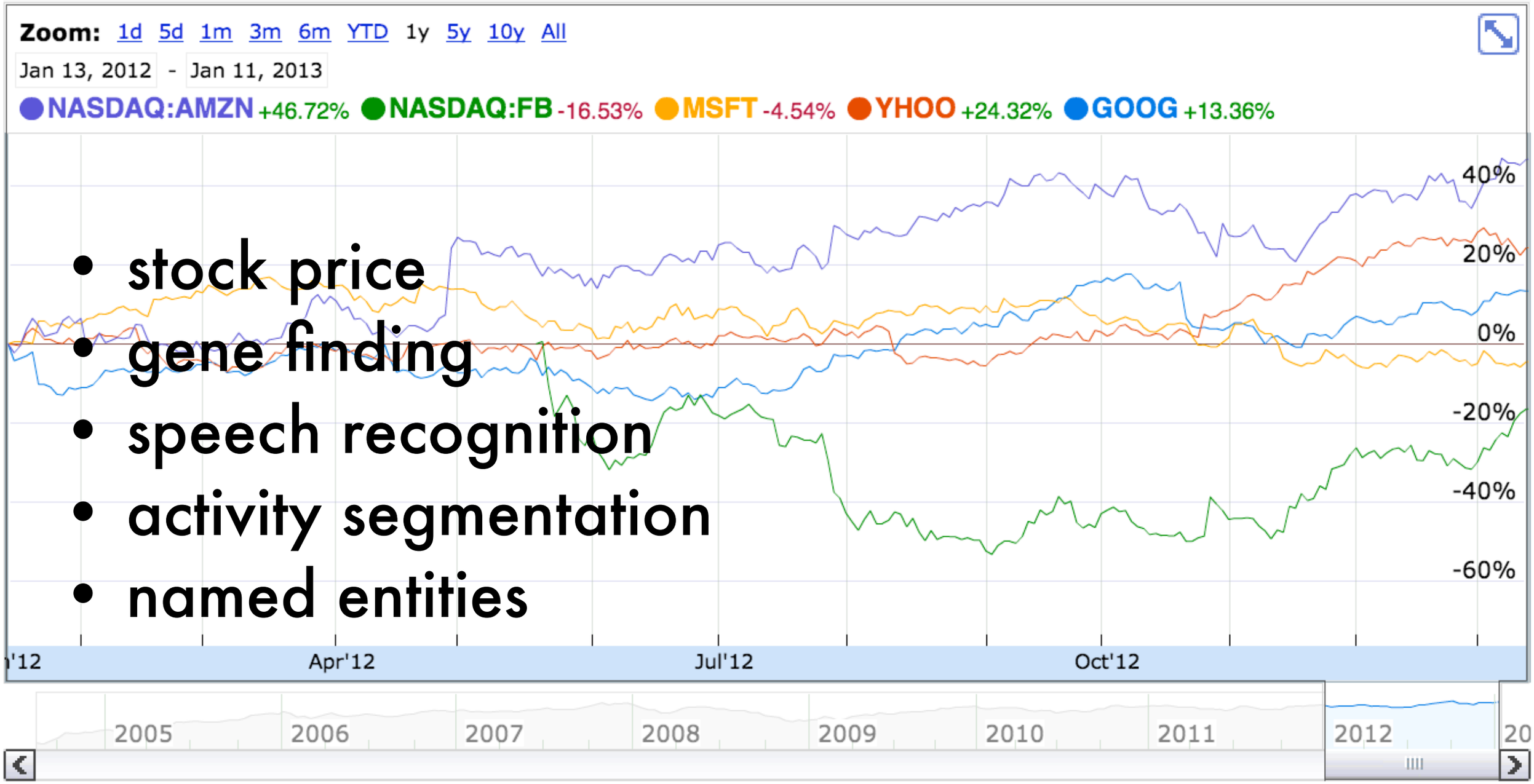
Autoregressive Latent State Models

Marianas Labs

Carnegie Mellon University

Prediction

- stock price
- gene finding
- speech recognition
- activity segmentation
- named entities



State and models

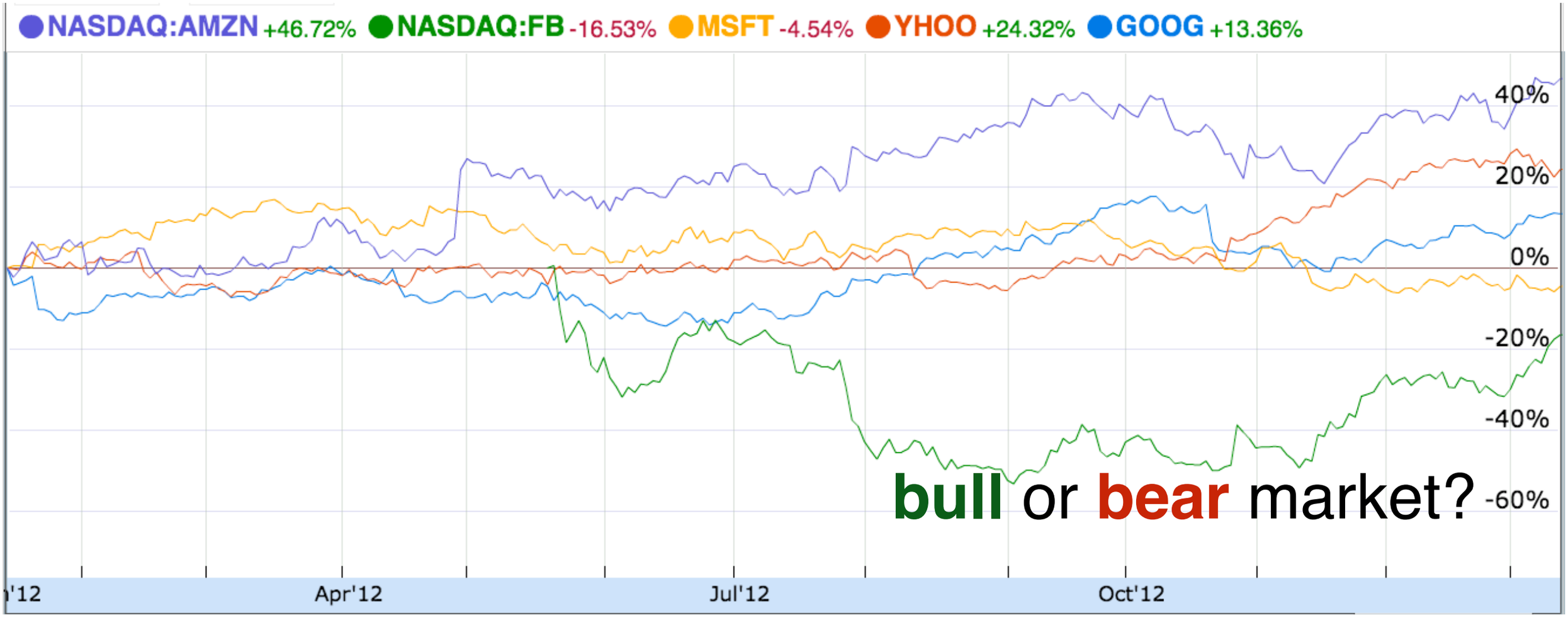
- **IID data**
  - Classification
  - Regression
  - Feature representation …
- **Most of the data isn't IID**
  - Sequence annotation (tagging, parsing)
  - Sequence generation (translation)
  - Summarization
  - Image annotation (content extraction)
- **Alternatives**: dynamic programs / stepwise prediction

Autoregressive Models / RNN

- Time series of observations

  … x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2} …

- Estimate $x_{t+1} = f(x_t, \ldots, x_{t-\tau})$ e.g. via deep net



Autoregressive Models

- Time series of observations

  … x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2} …

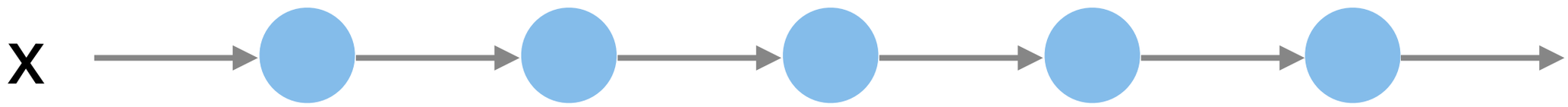
- Estimate $x_{t+1} = f(x_t, \ldots, x_{t-\tau})$ e.g. via deep net
- **Problem**
  - Hard to encode latent state (e.g. parity)
  - Hard to encode long range context/knowledge
- **Solution** - latent state models

$$x_{t+1} = f(x_t, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})$$

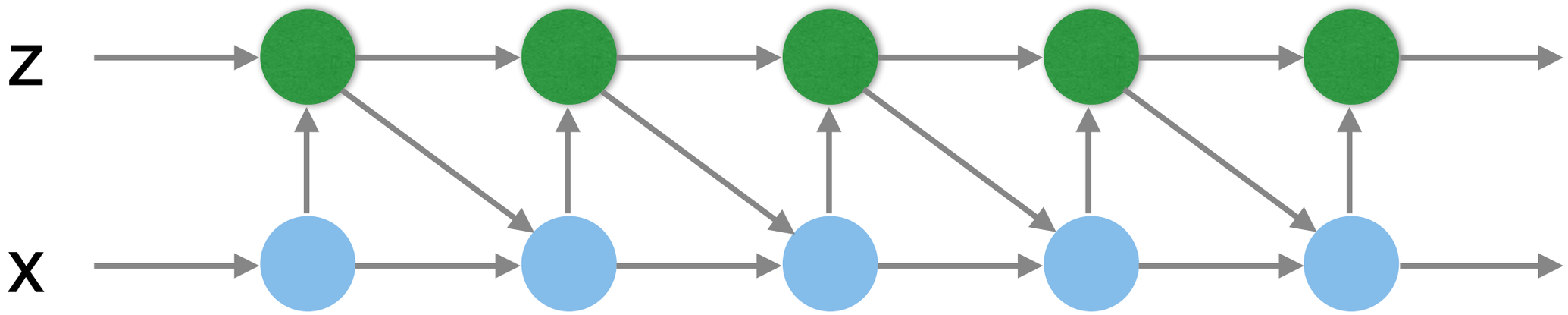
$$z_{t+1} = g(x_{t+1}, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})$$

  also called Recurrent Neural Networks

Autoregressive Models / RNN



$$x_{t+1} = f(x_t, \ldots, x_{t-\tau})$$

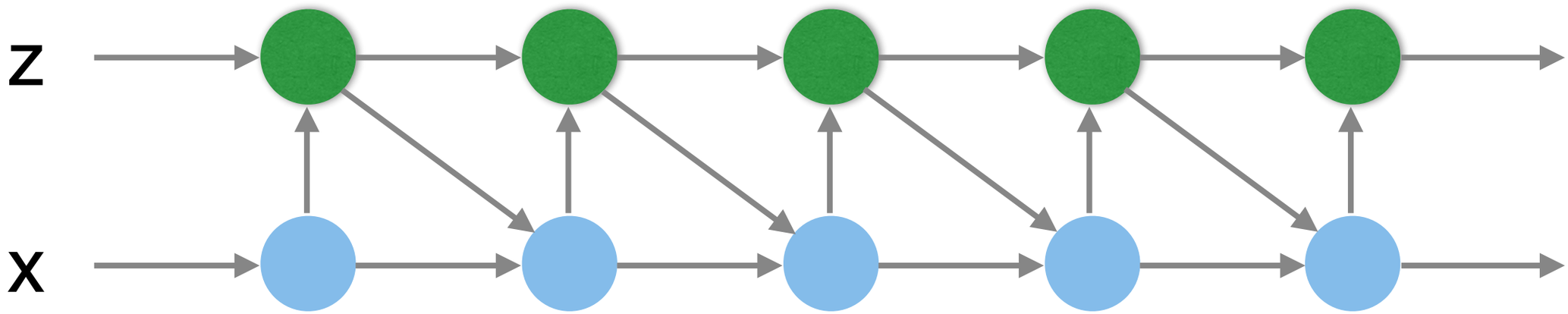


$$x_{t+1} = f(x_t, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})$$

$$z_{t+1} = g(x_{t+1}, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})$$

Autoregressive Models / RNN

- Sequence of observations
  - Gradients need to propagate back through t
  - Gradient may vanish. Makes model difficult to train. Due to stability condition on gradients



$$x_{t+1} = f(x_t, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})$$

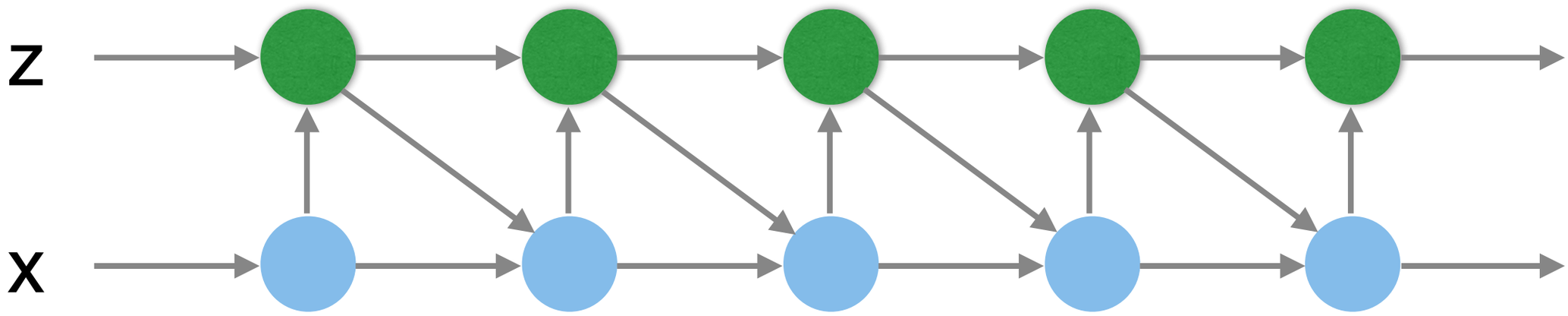
$$z_{t+1} = g(x_{t+1}, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})$$

Autoregressive Models / RNN

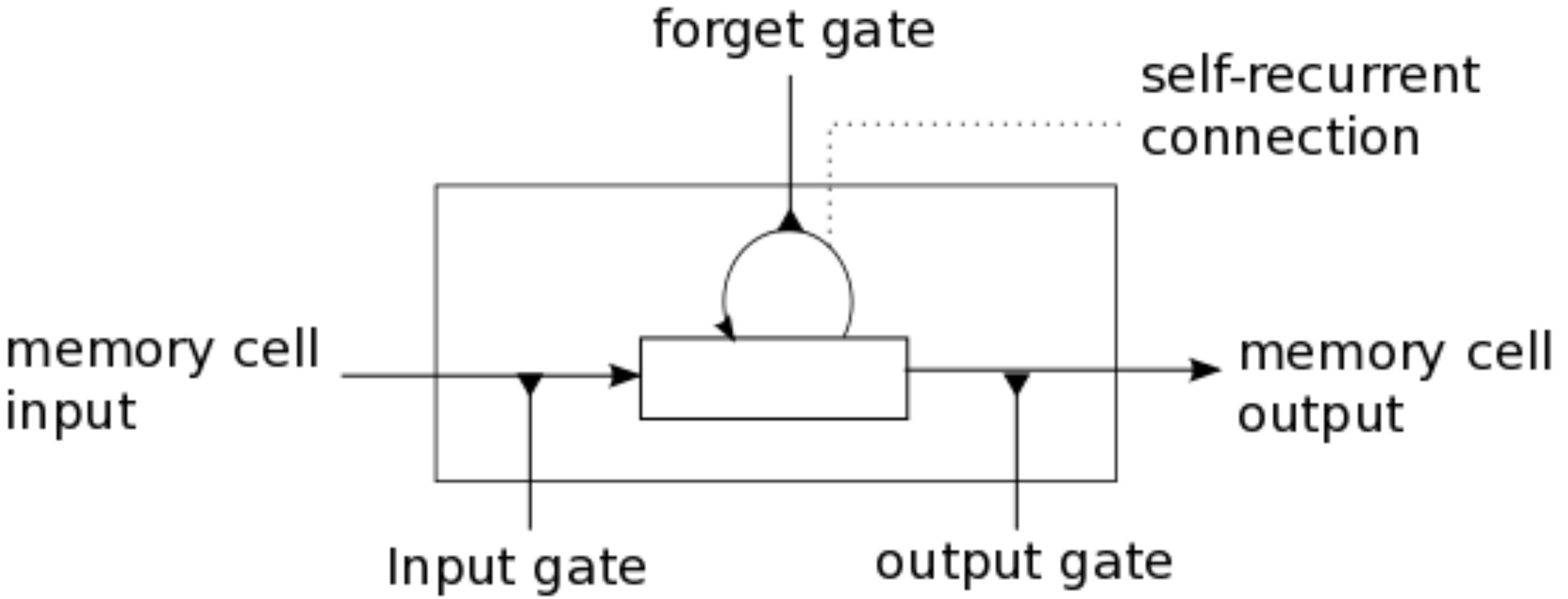
- Gradients need to propagate back through time

$$x_{t+1} = f(x_t, \ldots, x_{t-\tau})$$
$$= f(f(x_{t-1}, \ldots, x_{t-\tau-1}), x_{t-1}, \ldots, x_{t-\tau}) = \ldots$$
$$\partial_{x_{t-a}} x_t = f'_{t-1} \cdot f'_{t-2} \cdot \ldots$$

gradients may vanish / explode



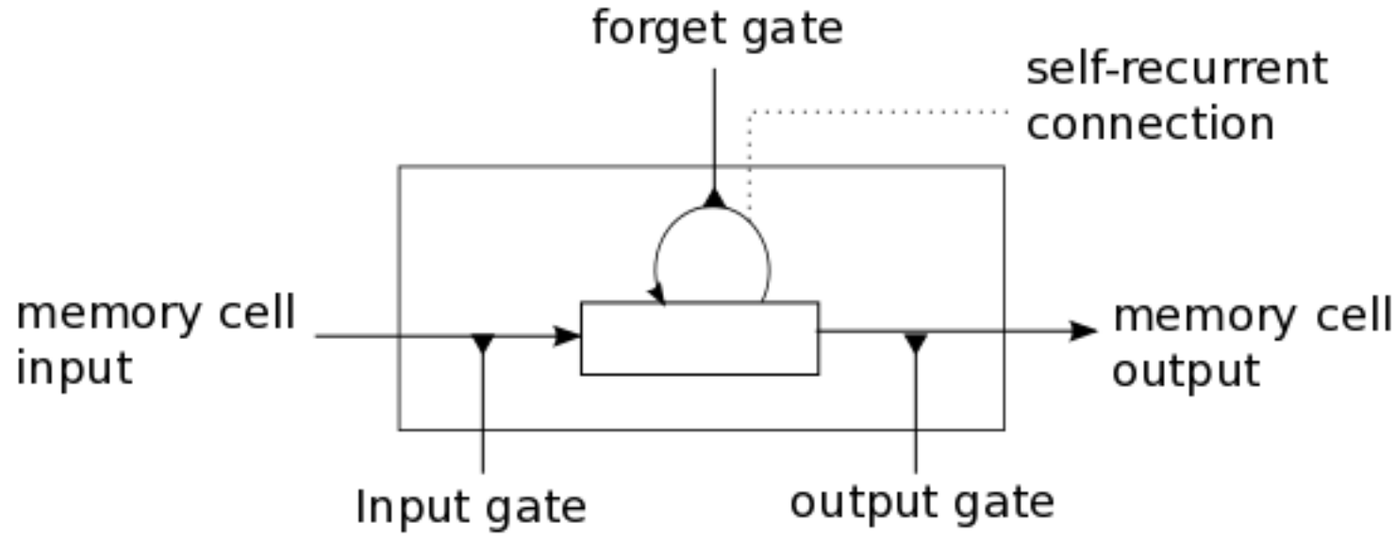
$$x_{t+1} = f(x_t, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})$$
$$z_{t+1} = g(x_{t+1}, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})$$

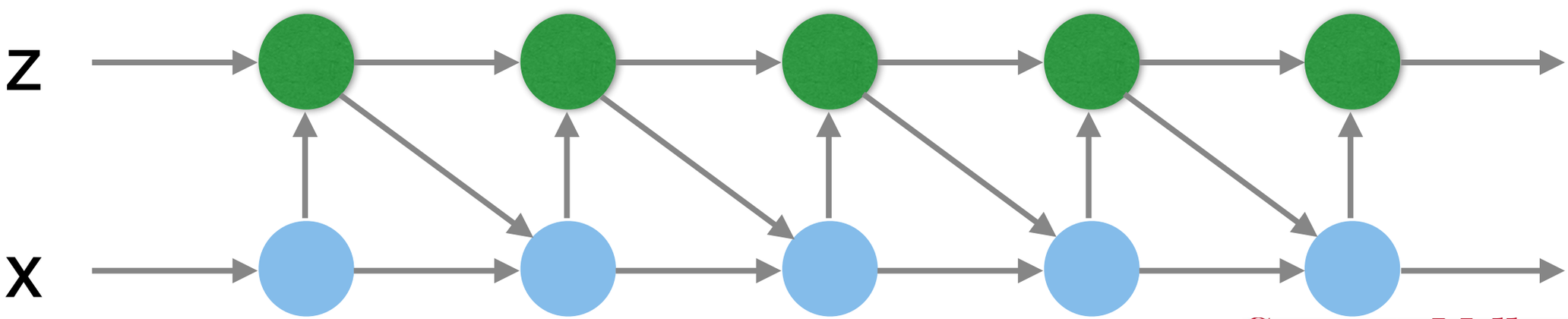


Long Short Term Memory

LSTM (Long Short Term Memory)

- Sequence of observations
  Latent state has custom update (like FlipFlop)
  Hochreiter & Schmidhuber, 1998

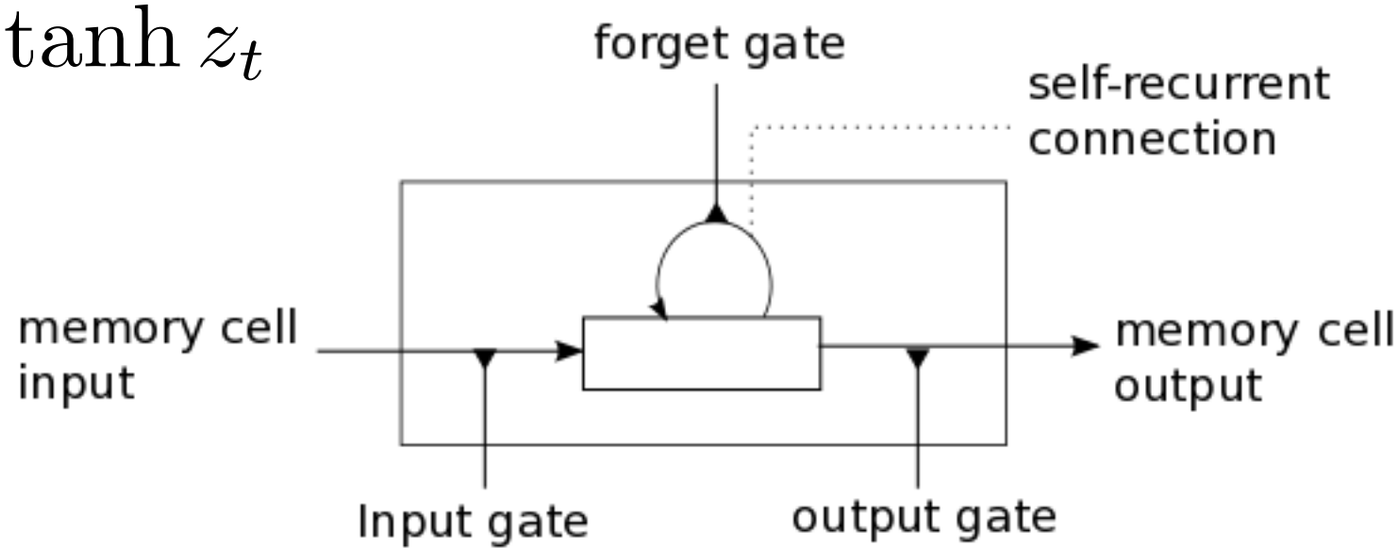




LSTM (Long Short Term Memory)

- Sequence of observations
  Latent state has custom update semantics

$$i_t = \sigma(W_i(x_t, h_{t-1}) + b_i)$$
$$f_t = \sigma(W_f(x_t, h_{t-1}) + b_f)$$
$$z_t = f_t * z_{t-1} + i_t * \tanh(W_z(x_t, h_{t-1}) + b_z)$$
$$o_t = \sigma(W_o(x_t, h_{t-1}, z_t) + b_f)$$
$$h_t = o_t * \tanh z_t$$



LSTM (Long Short Term Memory)

- Sequence of observations
  Latent state has custom update semantics

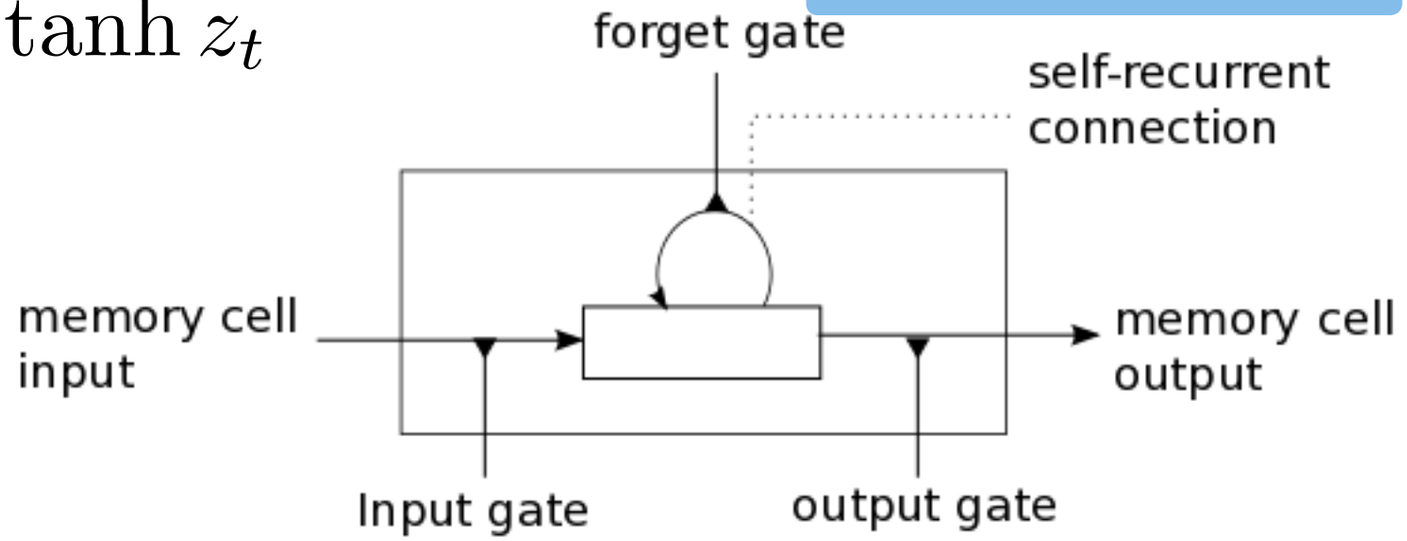
$$i_t = \sigma(W_i(x_t, h_{t-1}) + b_i)$$ input

$$f_t = \sigma(W_f(x_t, h_{t-1}) + b_f)$$ forgetting

$$z_t = f_t * z_{t-1} + i_t * \tanh(W_z(x_t, h_{t-1}) + b_z)$$ state

$$o_t = \sigma(W_o(x_t, h_{t-1}, z_t) + b_f)$$ output gate

$$h_t = o_t * \tanh z_t$$ output



LSTM (Long Short Term Memory)

- Sequence of observations
  Latent state has custom update semantics

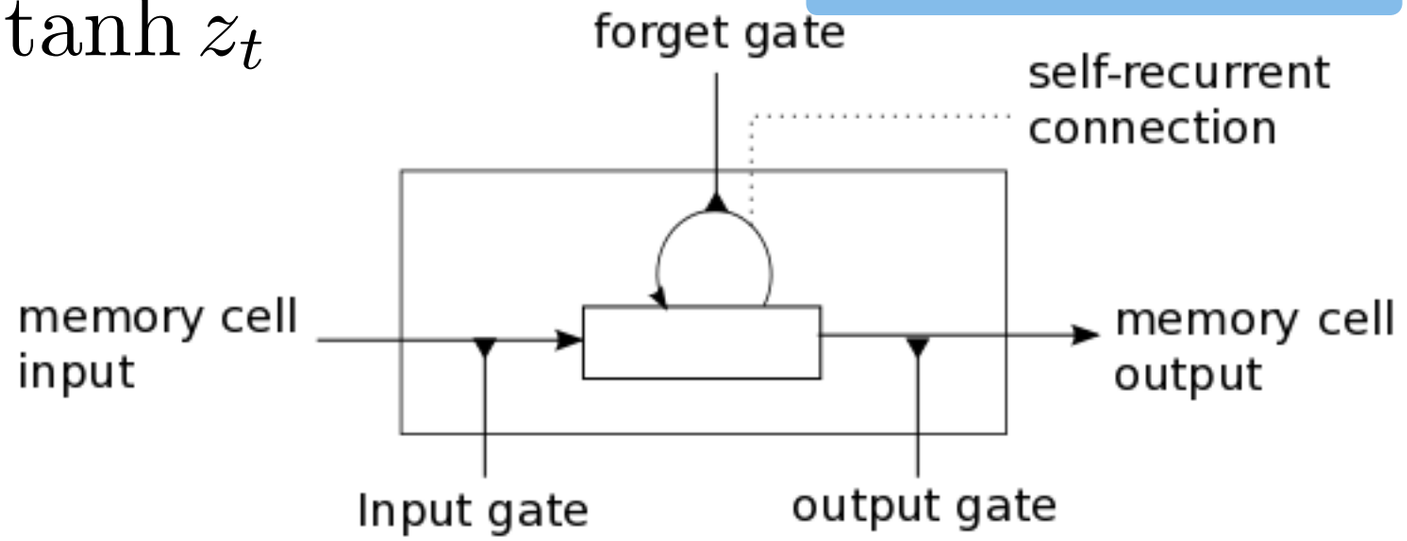
$$i_t = \sigma(W_i(x_t, h_{t-1}) + b_i)$$ input

$$f_t = \sigma(W_f(x_t, h_{t-1}) + b_f)$$ forgetting

$$z_t = f_t * z_{t-1} + i_t * \tanh(W_z(x_t, h_{t-1}) + b_z)$$ state

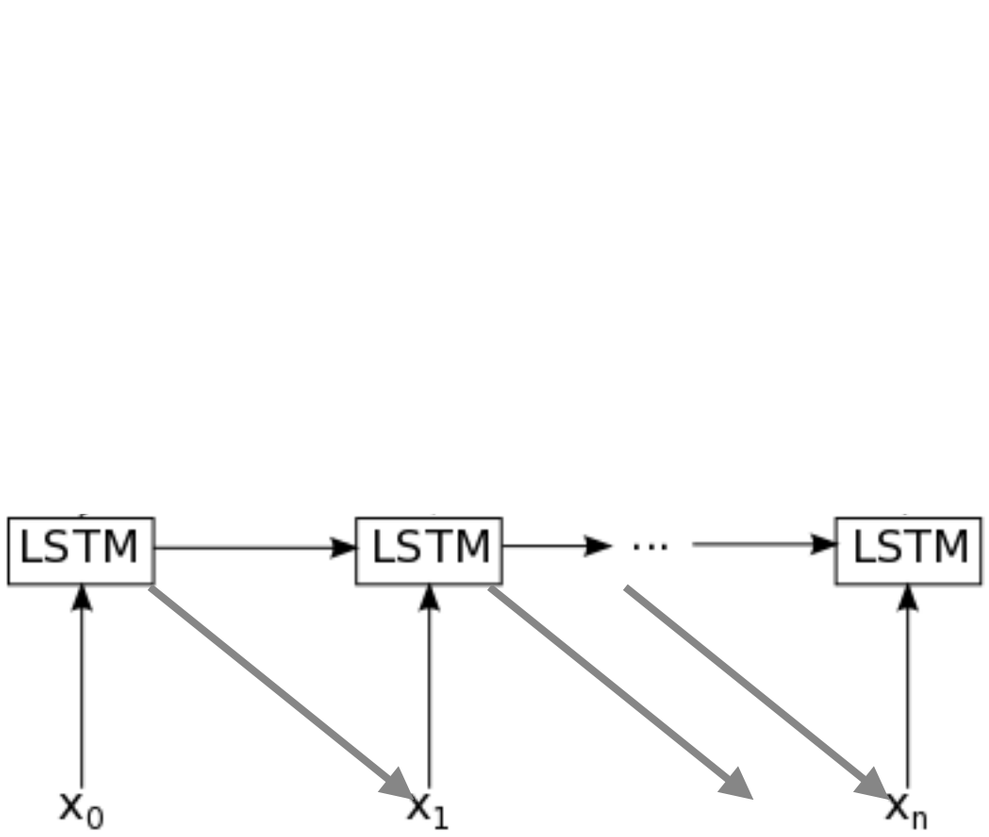
$$o_t = \sigma(W_o(x_t, h_{t-1}, z_t) + b_f)$$ output gate

$$h_t = o_t * \tanh z_t$$ output

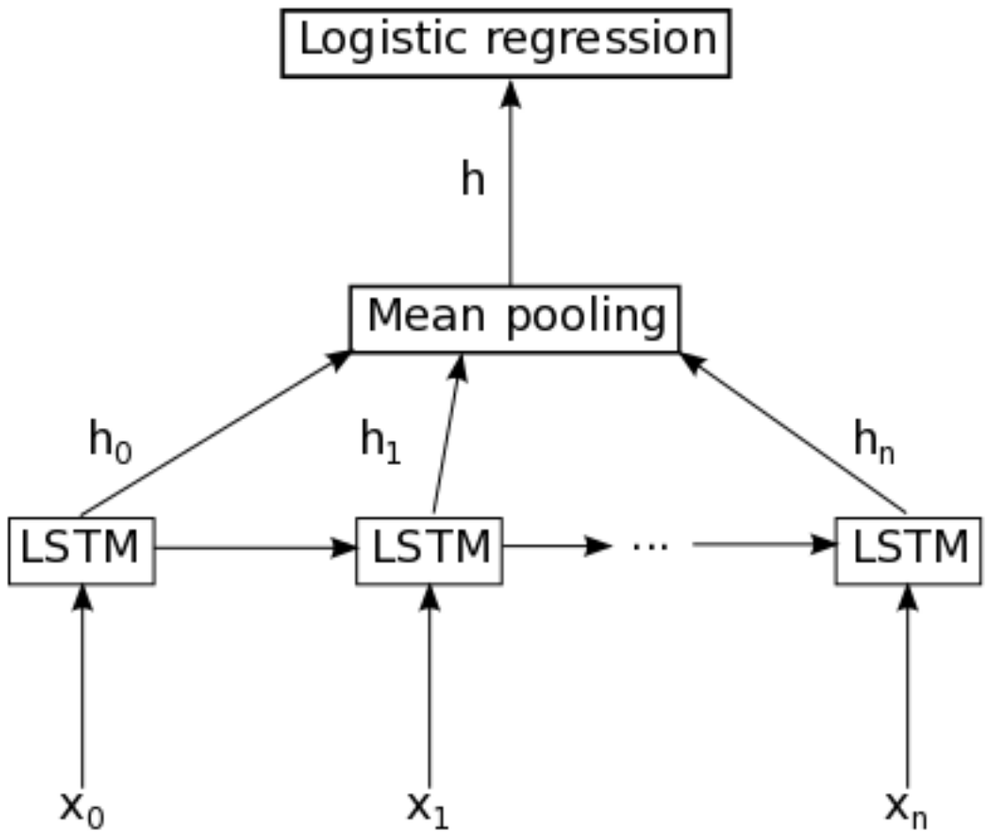


LSTM (Long Short Term Memory)

- Sequence of observations
  Latent state has custom update semantics



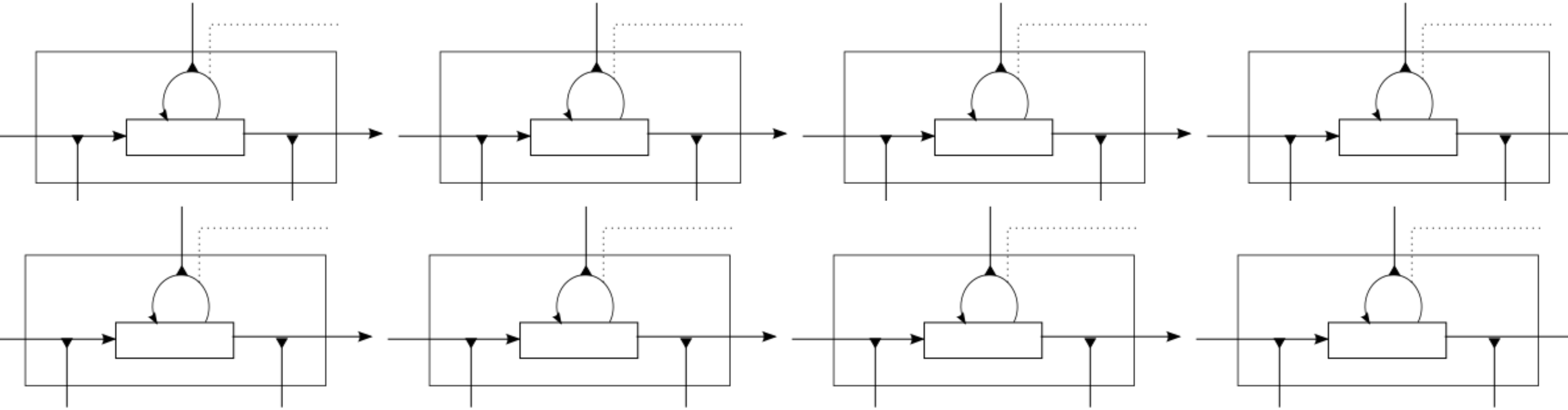
sequence generation



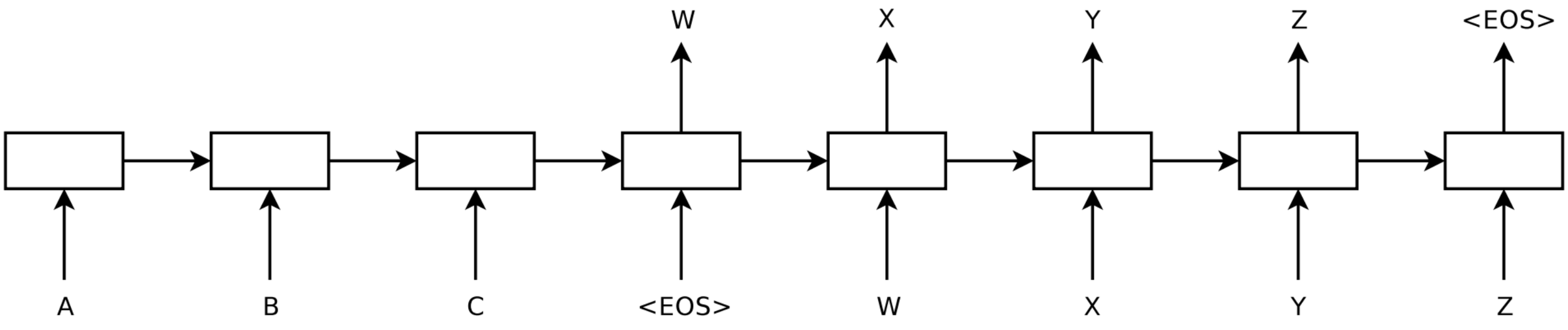
sequence classification

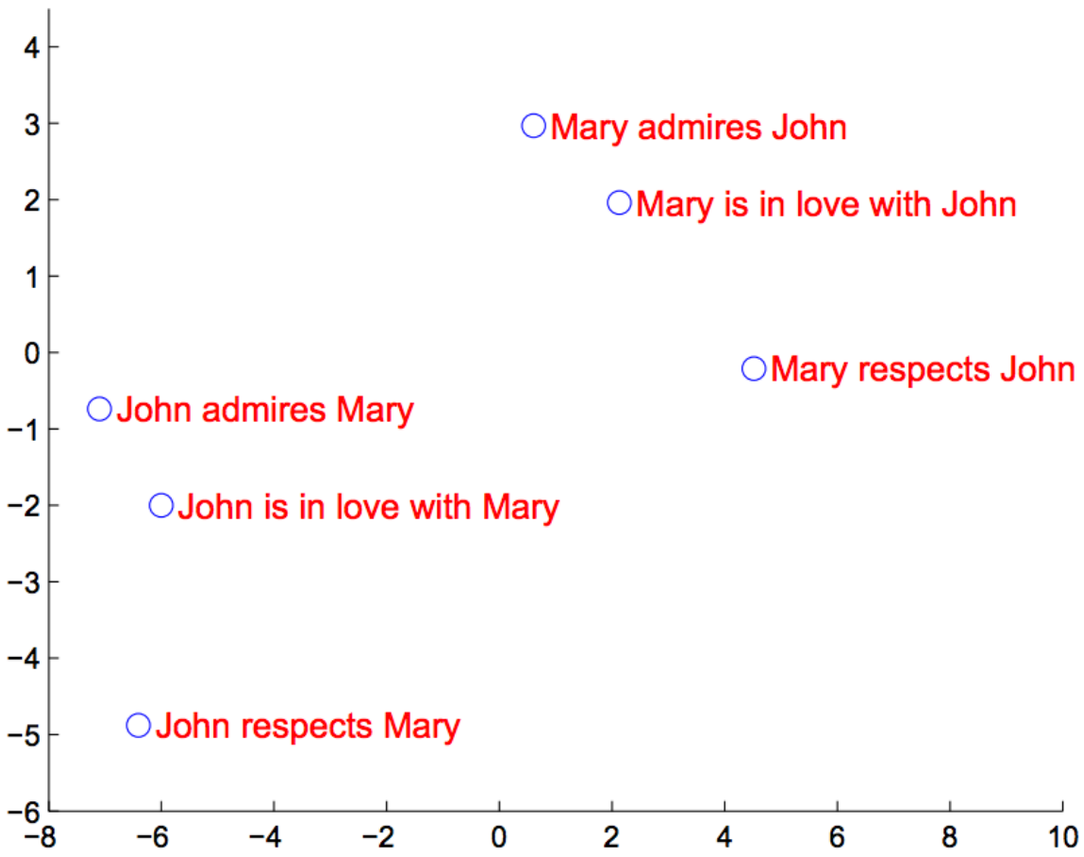
LSTM (Long Short Term Memory)

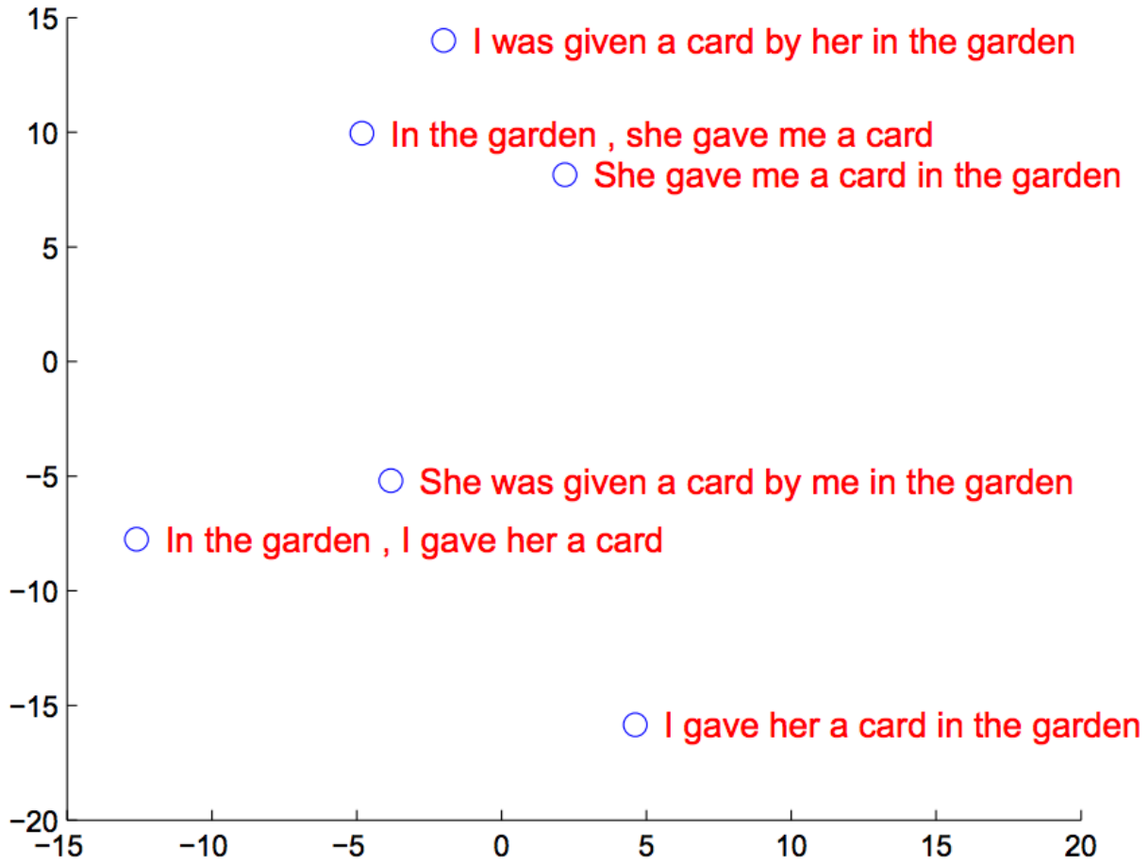
- Group LSTM cells into layer
- Multiple layers
- Can model different scales of dynamics



Example (Le, Sutskever, Vinyals, NIPS 2014) Natural Language Translation

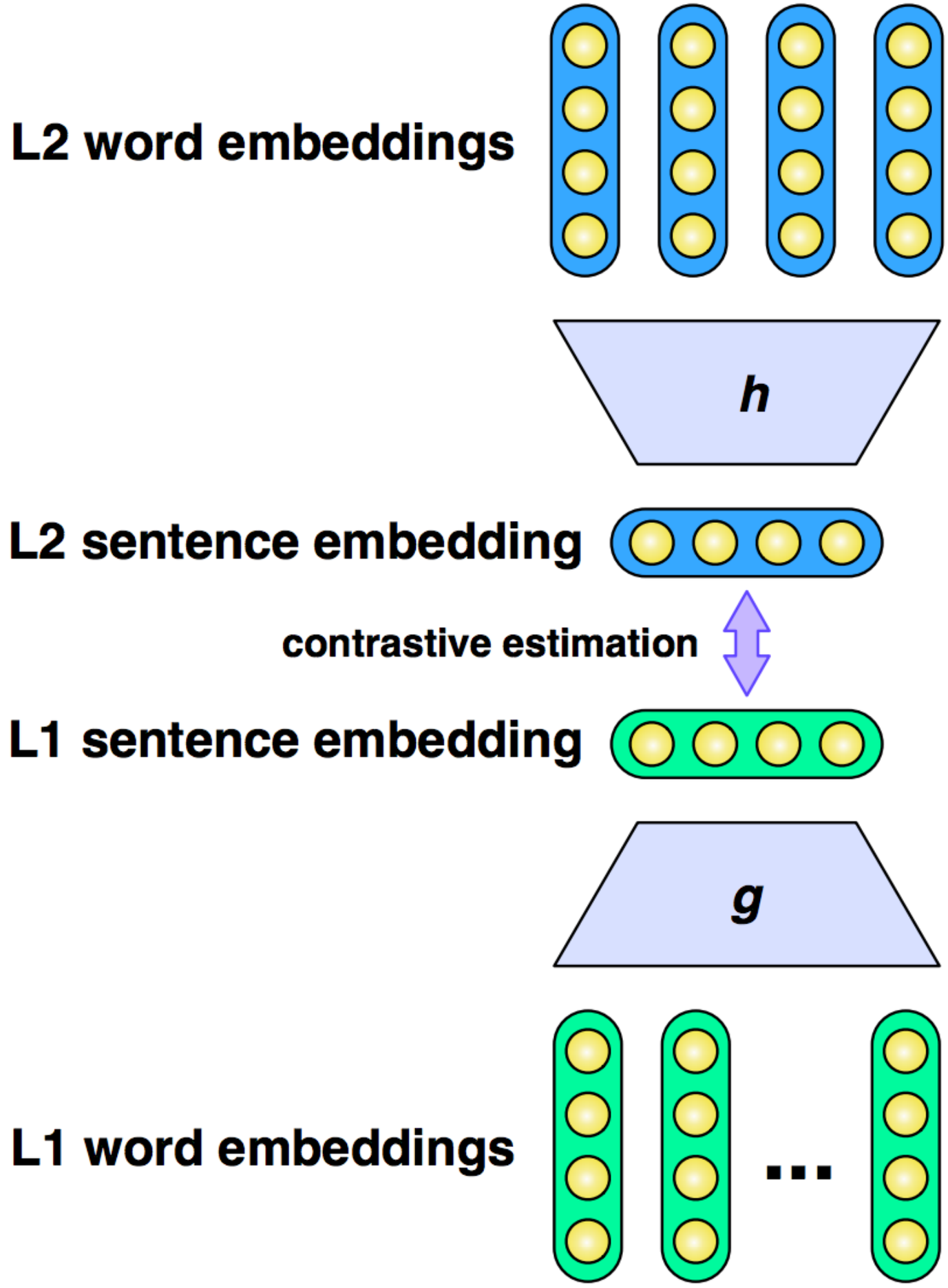






‘Synesthesia’

- Sequence embedding via LSTM
- Enforce closeness between LSTM state to obtain similarity between sequences



Much more

- Memory is area of active research
  - Neural Turing Machine
    http://arxiv.org/abs/1410.5401
  - Memory Networks
    http://arxiv.org/abs/1410.3916
- Attention models (Kyunghyun Cho)

8. Tools


2.5 3 3.5 4 5 + -
T3 T4 T5 T6 T7 T8
H3.0 H3.5 H4.0 H4.5 H5.0 H6.0
0.8 1.0 1.5 2.0 2.5 3.0
3.5 4.0 J000 J00 J0 J1
PH000 PH00 PH0 PH1 PH2 3.0
T9 T10 T15 T20 Y0 Y1

Quick overview

- Caffe http://caffe.berkeleyvision.org/
  Efficient for convolutional models / images
- Torch http://torch.ch/
  Very efficient. But you must LIKE Lua …
  Google and Facebook love it
- Theano http://deeplearning.net/software/theano/
  Compiled from Python. Not as efficient as Torch
- Minerva https://github.com/dmlc/minerva
  Compiler layout of execution on machines
- CXXNet https://github.com/dmlc/cxxnet
  Simpler than Caffe. More efficient
- Parameter Server bindings to https://github.com/dmlc/
  Minerva, Caffe, CXXNet, …