MACHINE LEARNING
DEPARTMENT

9.1 Overview

9 Deep Learning

Alexander Smola

Introduction to Machine Learning 10-701

http://alex.smola.org/teaching/10-701-15

Carnegie Mellon University

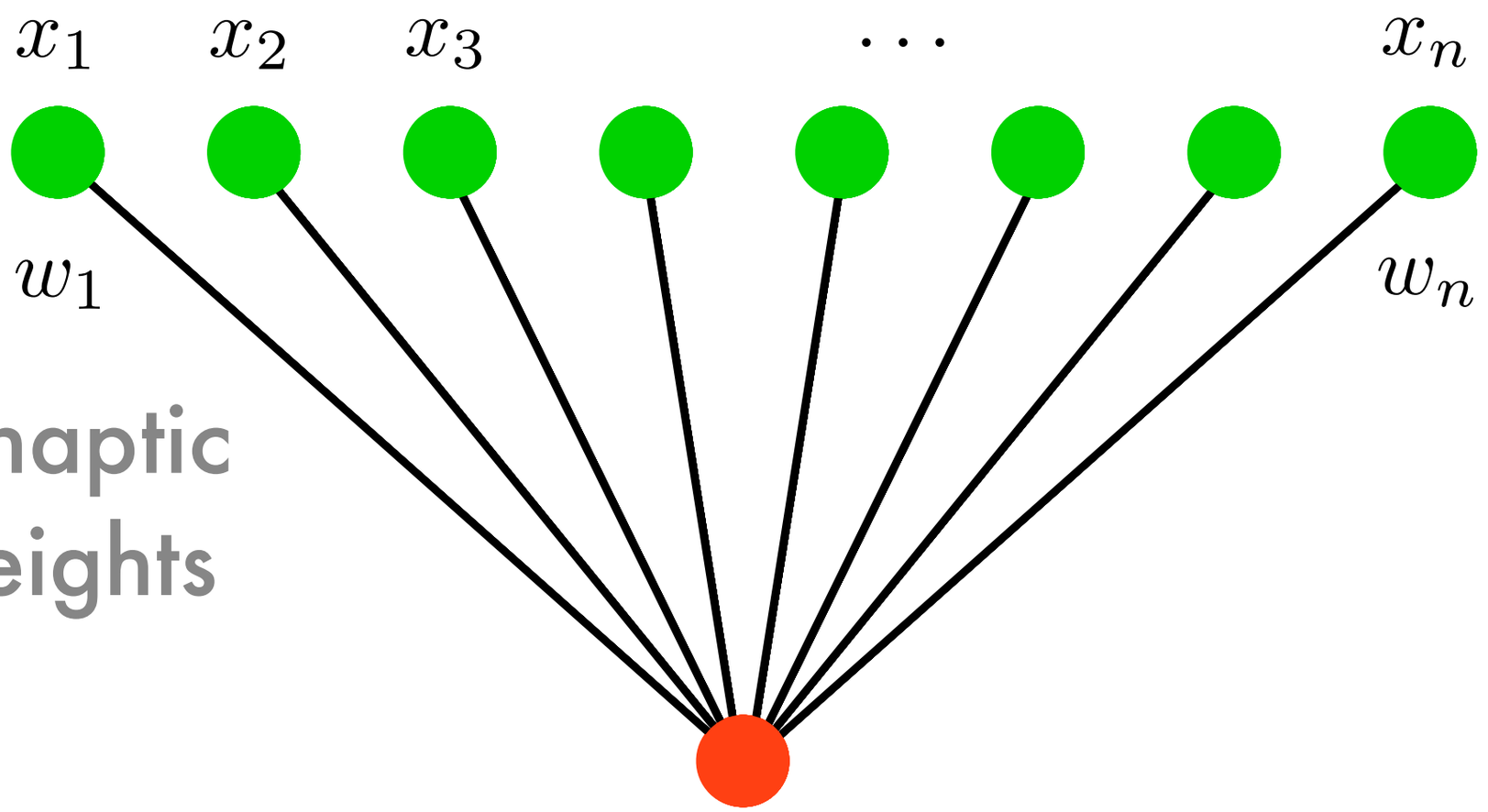
A brief history of computers

	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

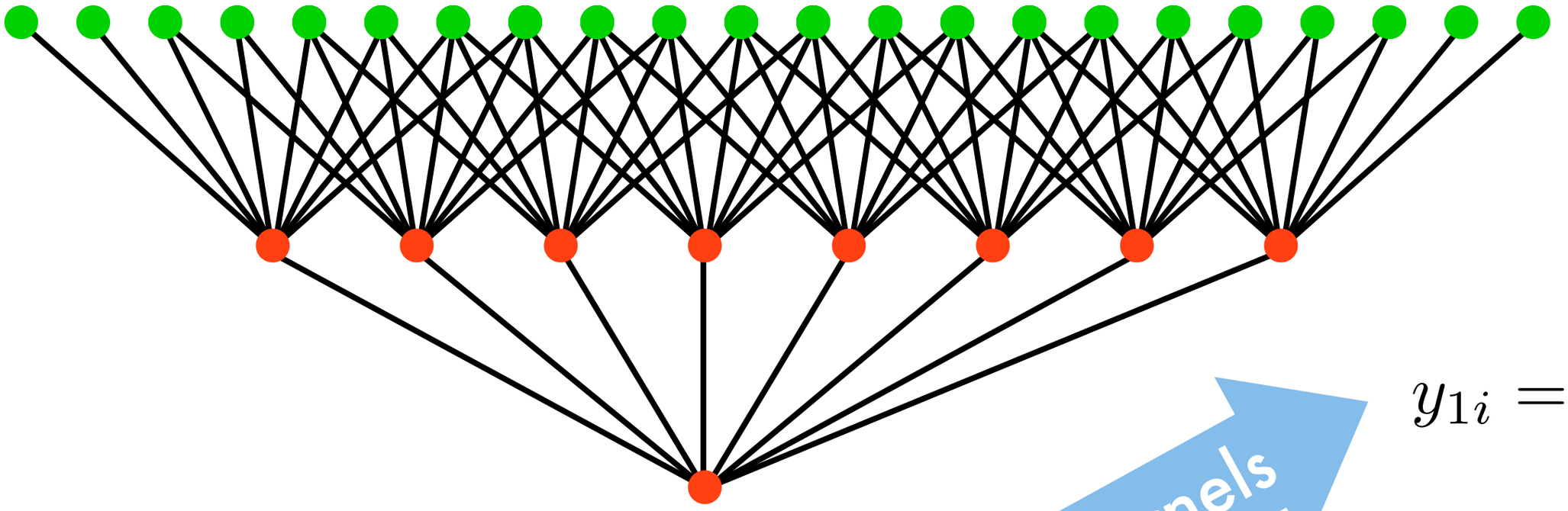
- Data grows at higher exponent
- Moore's law (silicon) vs. Kryder's law (disks)
- Early algorithms data bound, now CPU/RAM bound

Perceptron



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

Nonlinearities via Layers



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

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

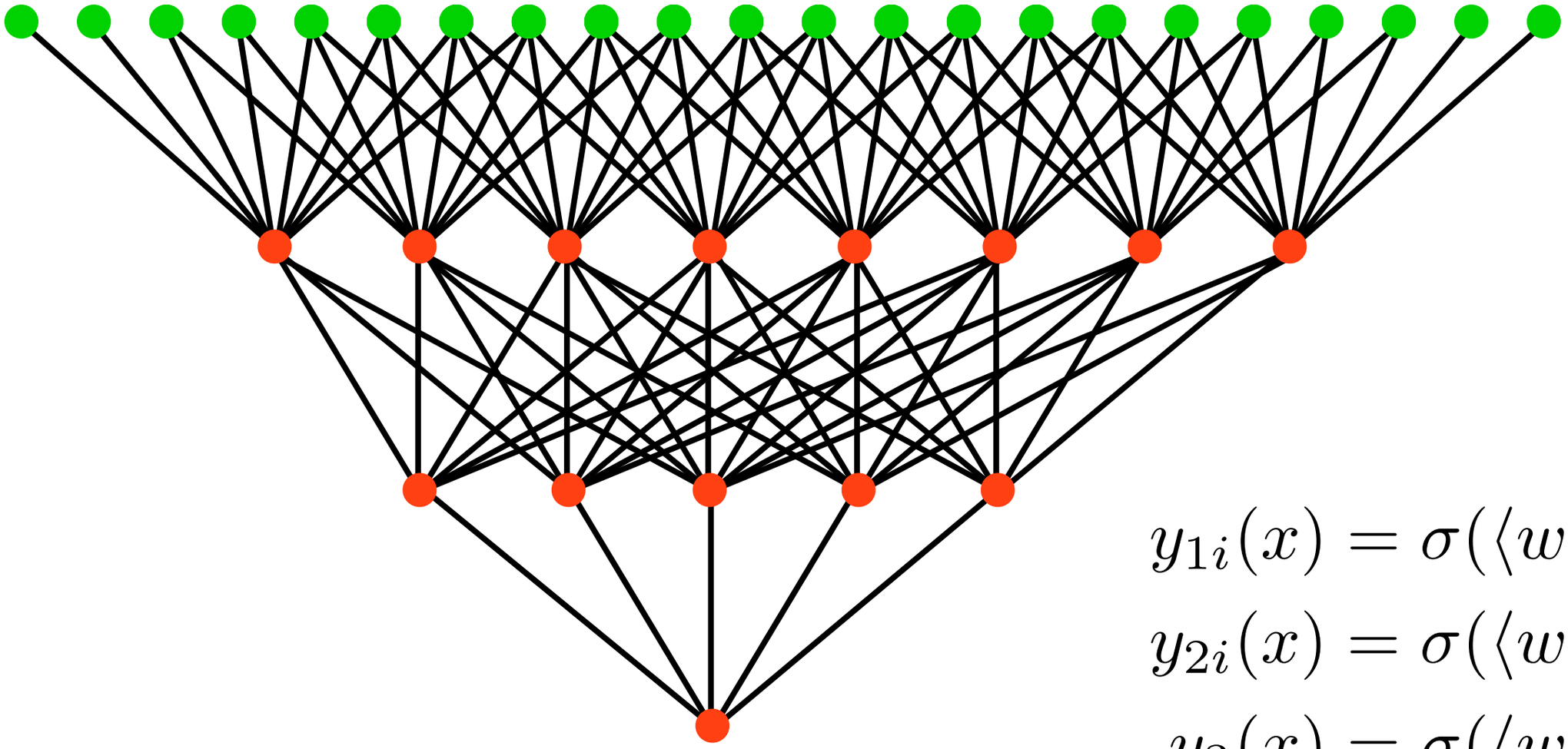
Kernels

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

Deep Nets

optimize
all weights

Nonlinearities via Layers



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

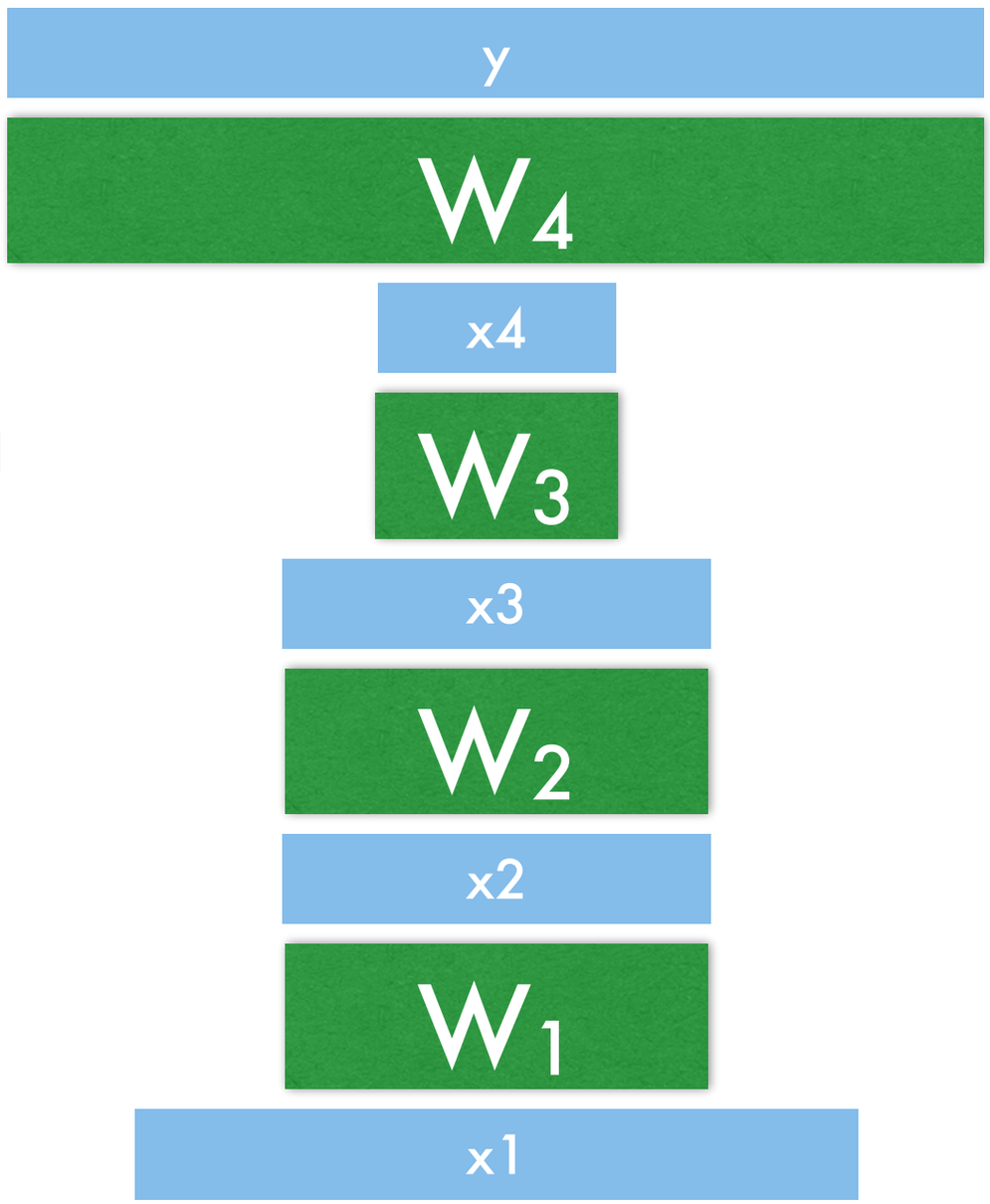
Multilayer Perceptron

- 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

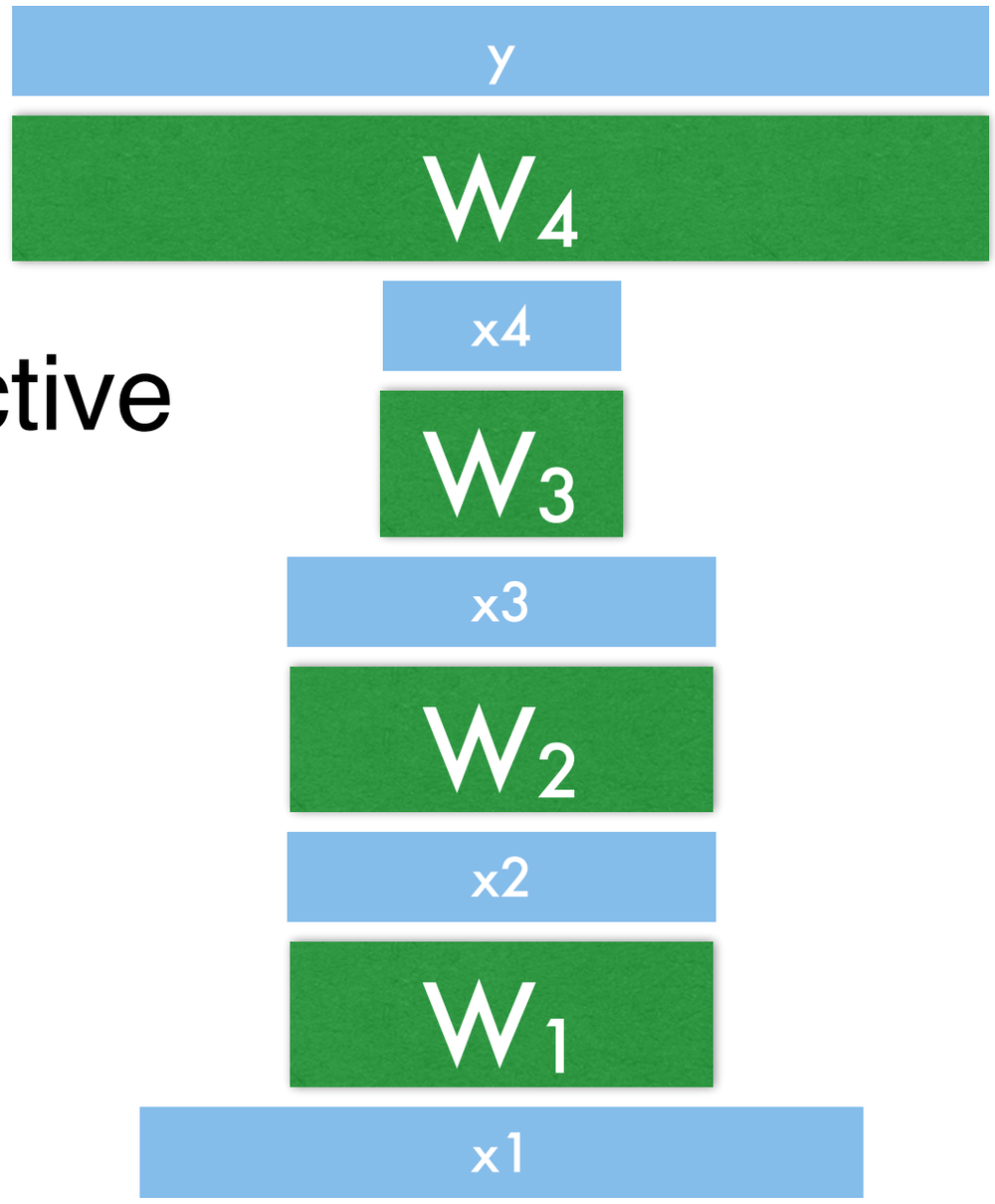


Backpropagation

- Layer Representation

$$y_i = W_i x_i$$

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



- Compute change in objective

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

- Chain rule

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

Backpropagation

- Layer Representation

$$y_i = W_i x_i$$

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

- Gradients

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

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

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

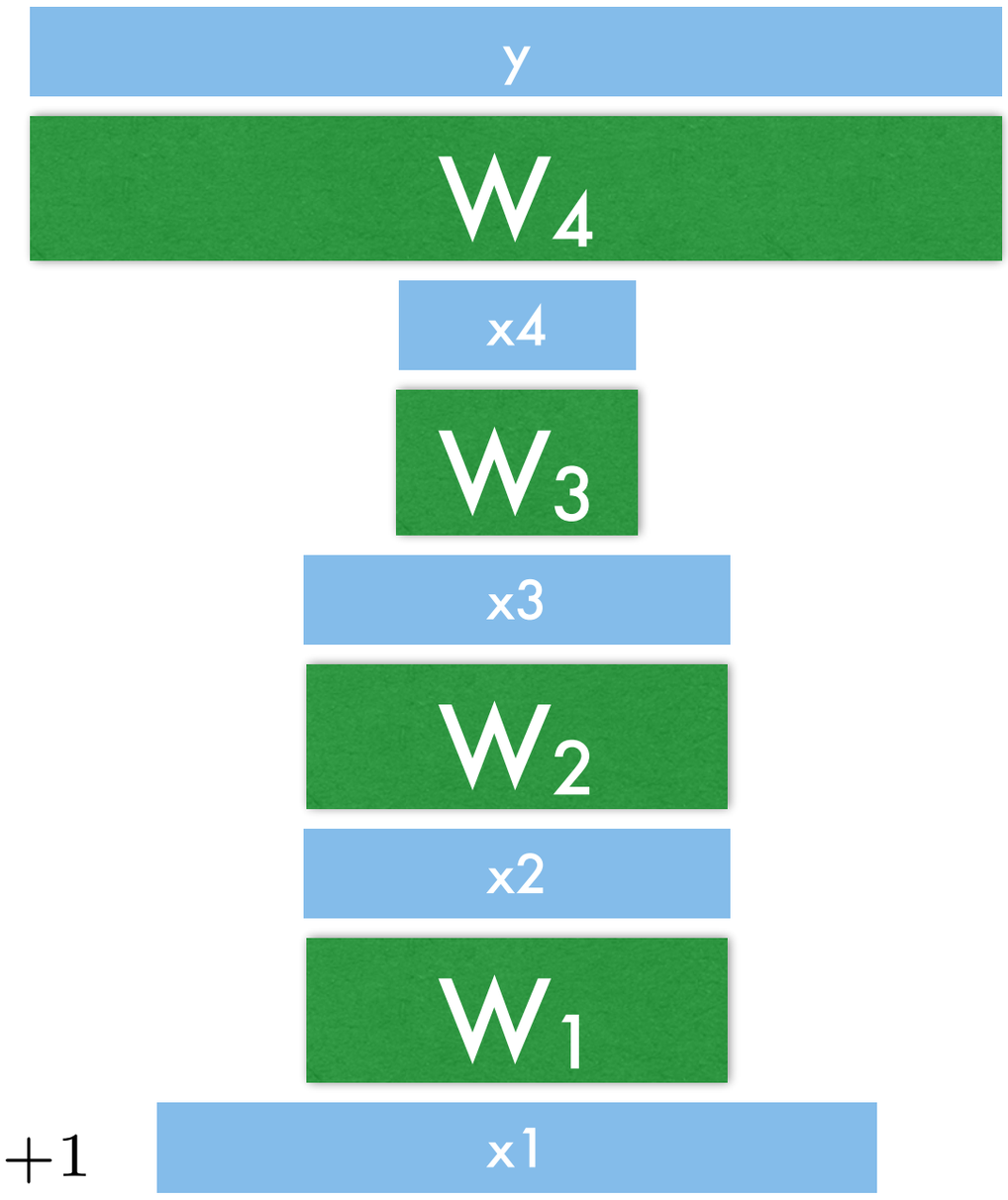
$$\implies \partial_{x_i} x_{i+1} = \sigma'(y_i) W_i^\top$$

- Backprop

$$g_n = \partial_{x_n} l(y, y_n)$$

$$g_i = \partial_{x_i} l(y, y_n) = g_{i+1} \partial_{x_i} x_{i+1}$$

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



Optimization

- Layer Representation

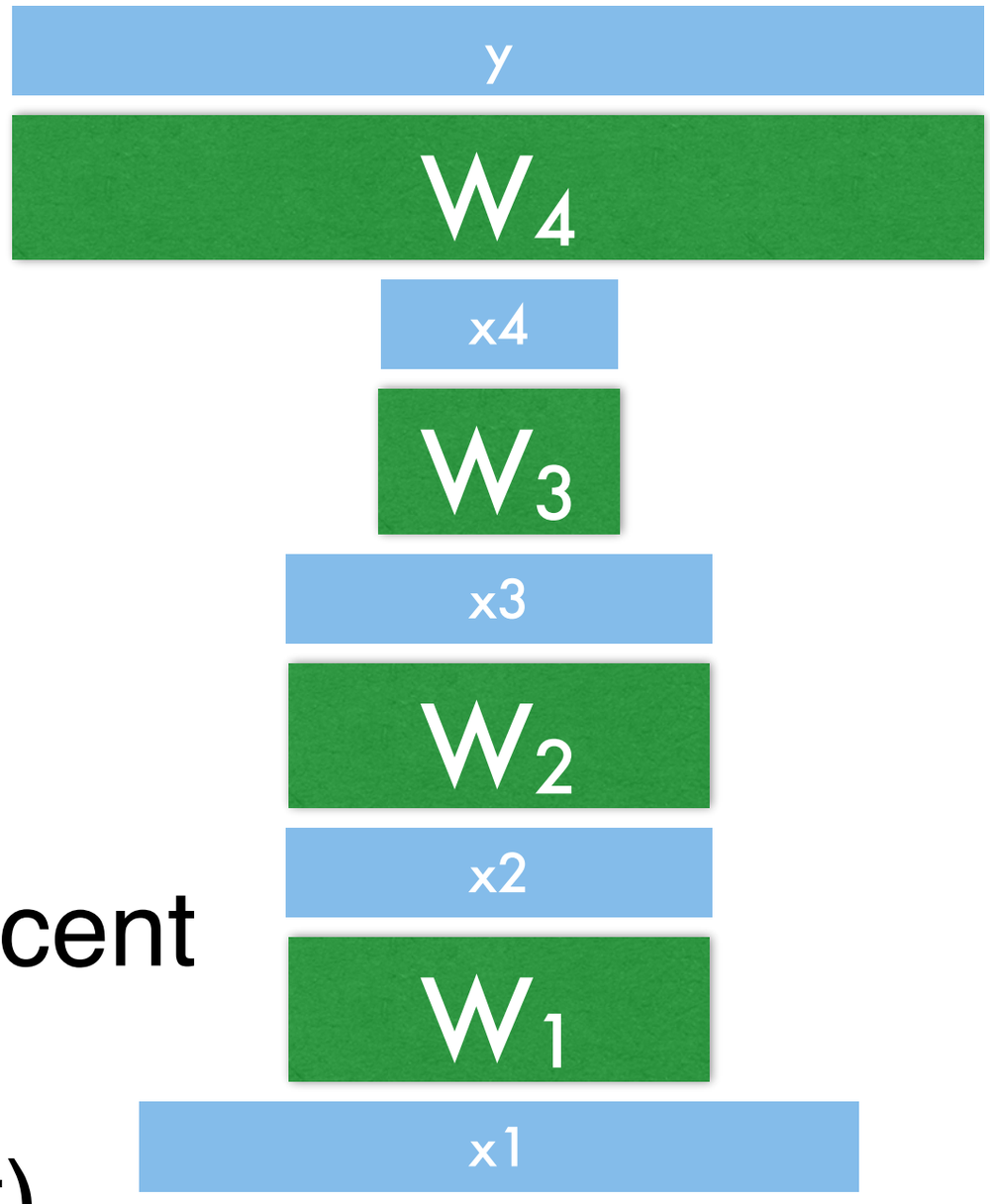
$$y_i = W_i x_i$$

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

- 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)



Things we could learn

- Binary classification

$$\log(1 + \exp(-yy_n))$$

- Multiclass classification (softmax)

$$\log \sum_{y'} \exp(y_n[y']) - y_n[y]$$

- Regression

$$\frac{1}{2} \|y - y_n\|^2$$

- Ranking (top-k)
- Preferences
- Sequences (see CRFs)

MACHINE LEARNING
DEPARTMENT

9.2 Layers

9 Deep Learning

Alexander Smola

Introduction to Machine Learning 10-701

http://alex.smola.org/teaching/10-701-15

Carnegie Mellon University

Fully Connected

- Forward mapping

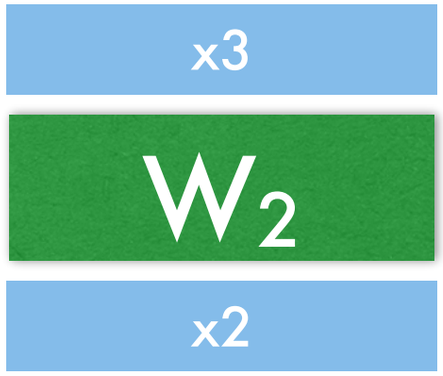
$$y_i = W_i x_i$$
$$x_{i+1} = \sigma(y_i)$$

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

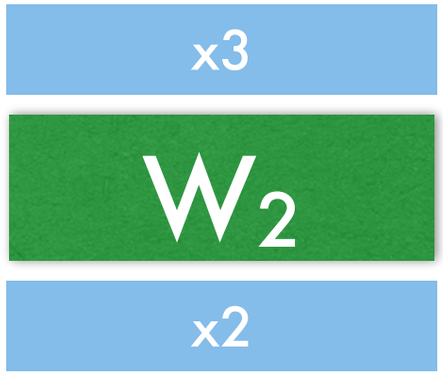


Rectified Linear Unit (ReLu)

- Forward mapping

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

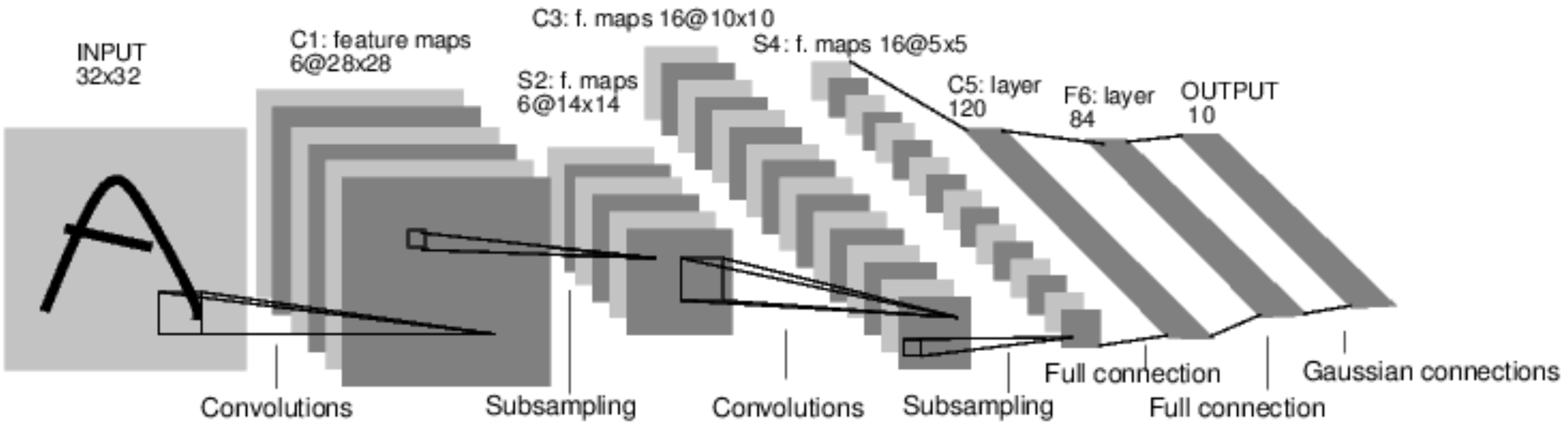
with subsequent nonlinearity



- 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)

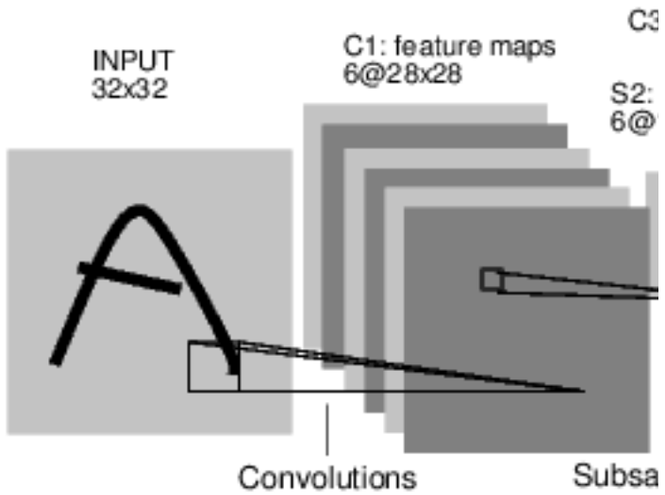
Where is Wally

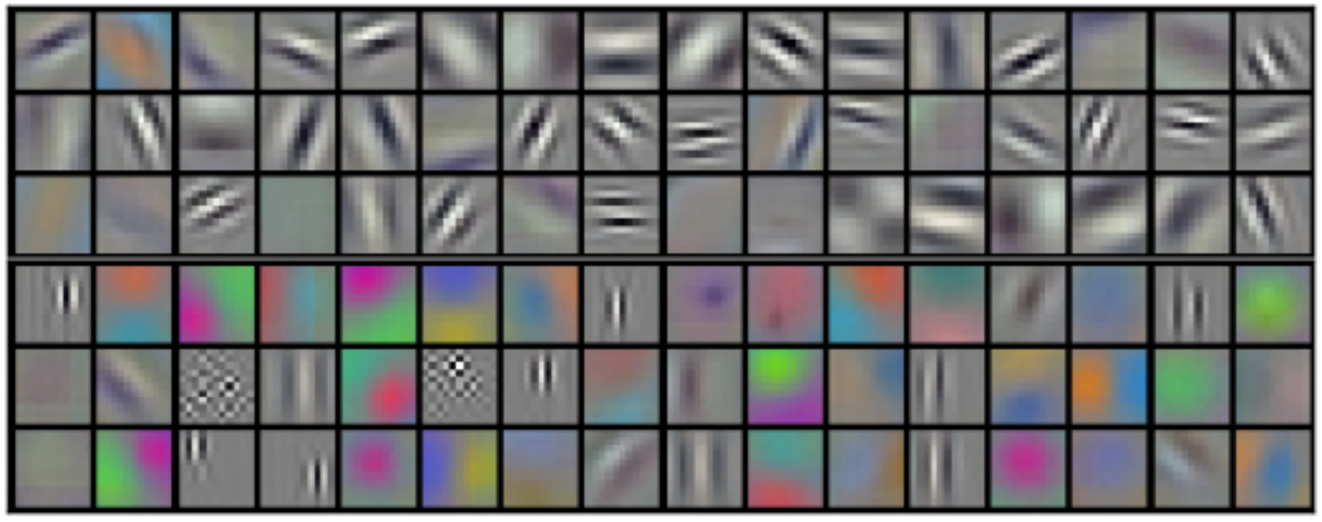


LeNet for OCR (1990s)

Convolutional Layers

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





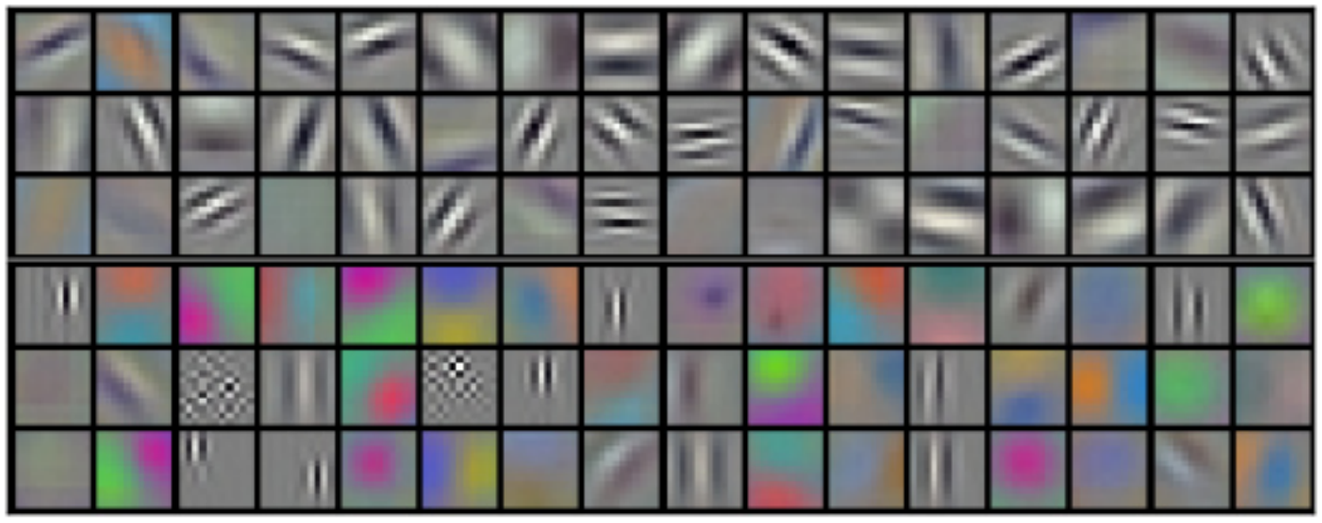
Convolutional Layers

- Images have translation invariance
- Forward (usually implemented brute force)

$$y_i = x_i \circ W_i$$

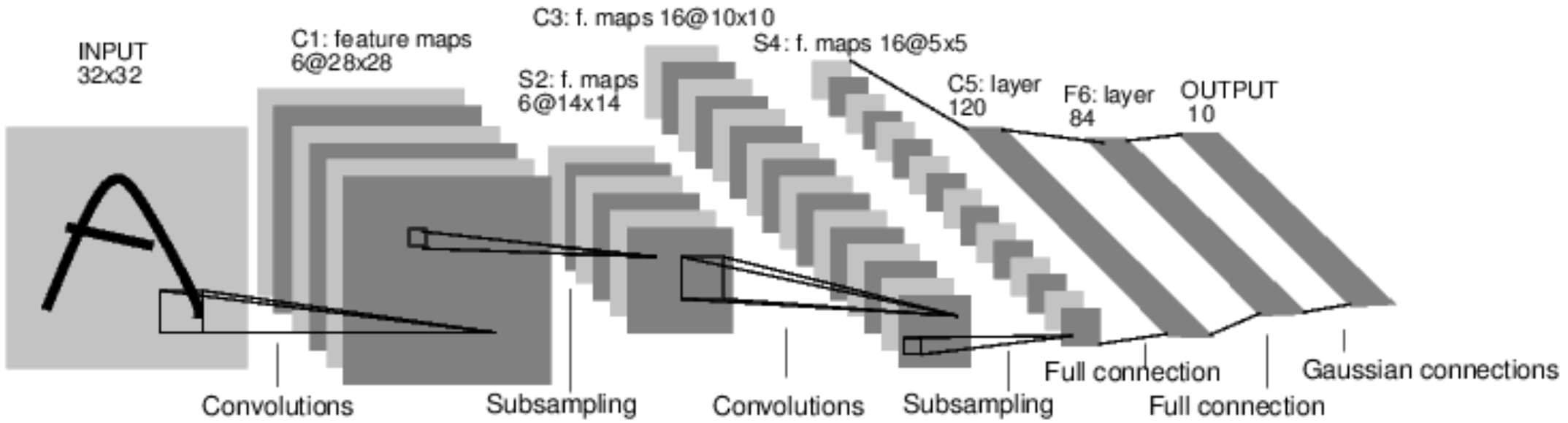
$$x_{i+1} = \sigma(y_i)$$

- Backward gradients
  (need to convolve appropriately)



Subsampling & MaxPooling

- Multiple convolutions blow up dimensionality



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

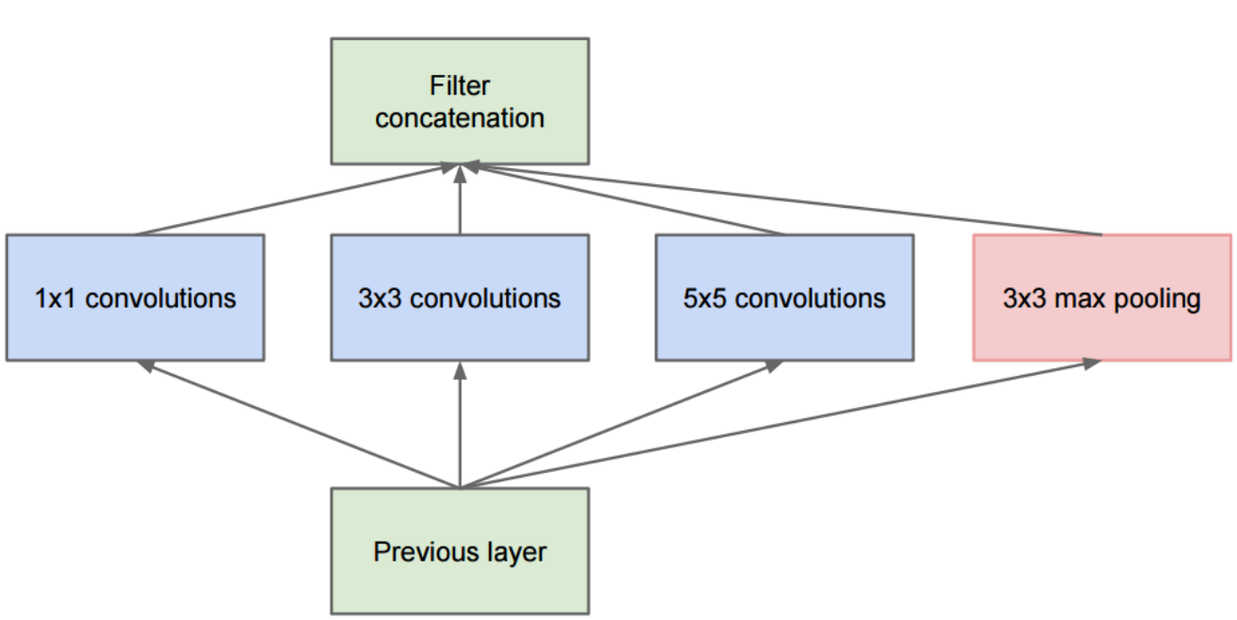
Depth vs. Width

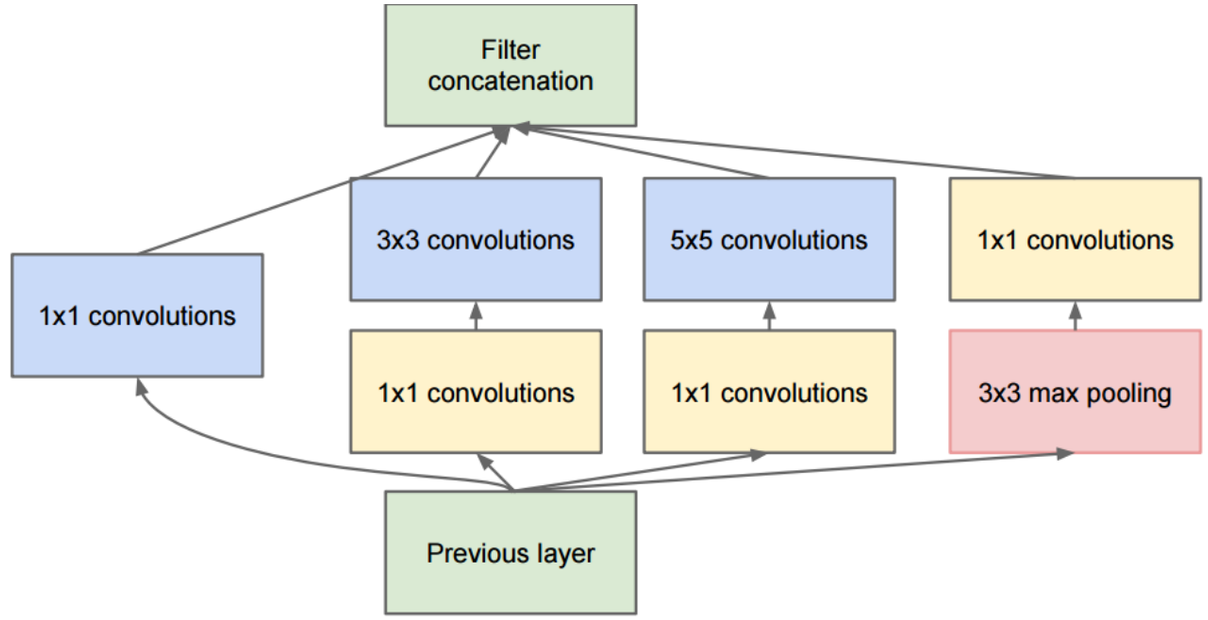
- Longer range effects
  - many narrow convolutions
  - few wide convolutions
- More nonlinearities work better (same number of parameters)

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

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

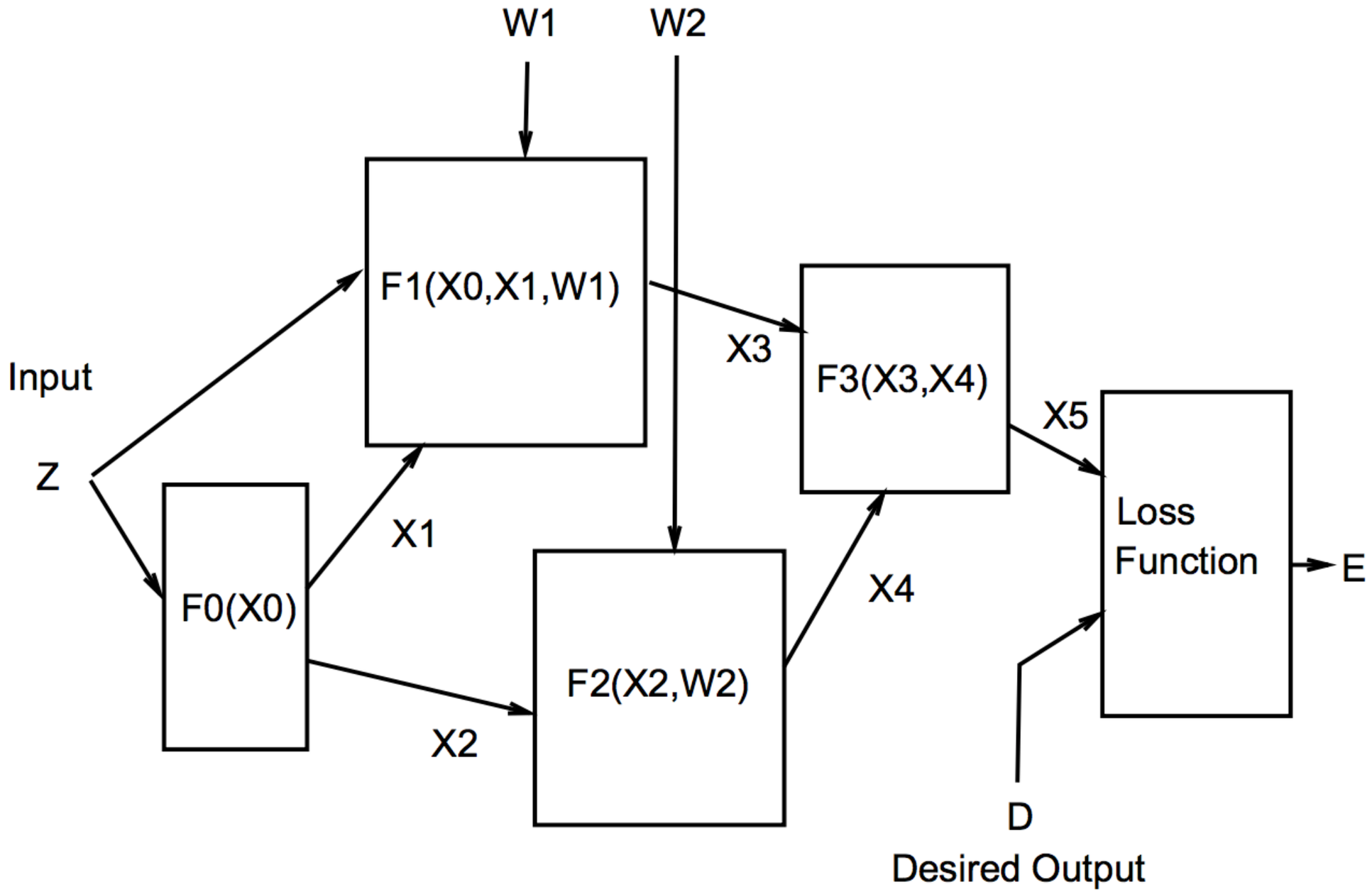




- Compute different filters
- Compose one big vector from all of them
- Layer this iteratively

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

Whole system training

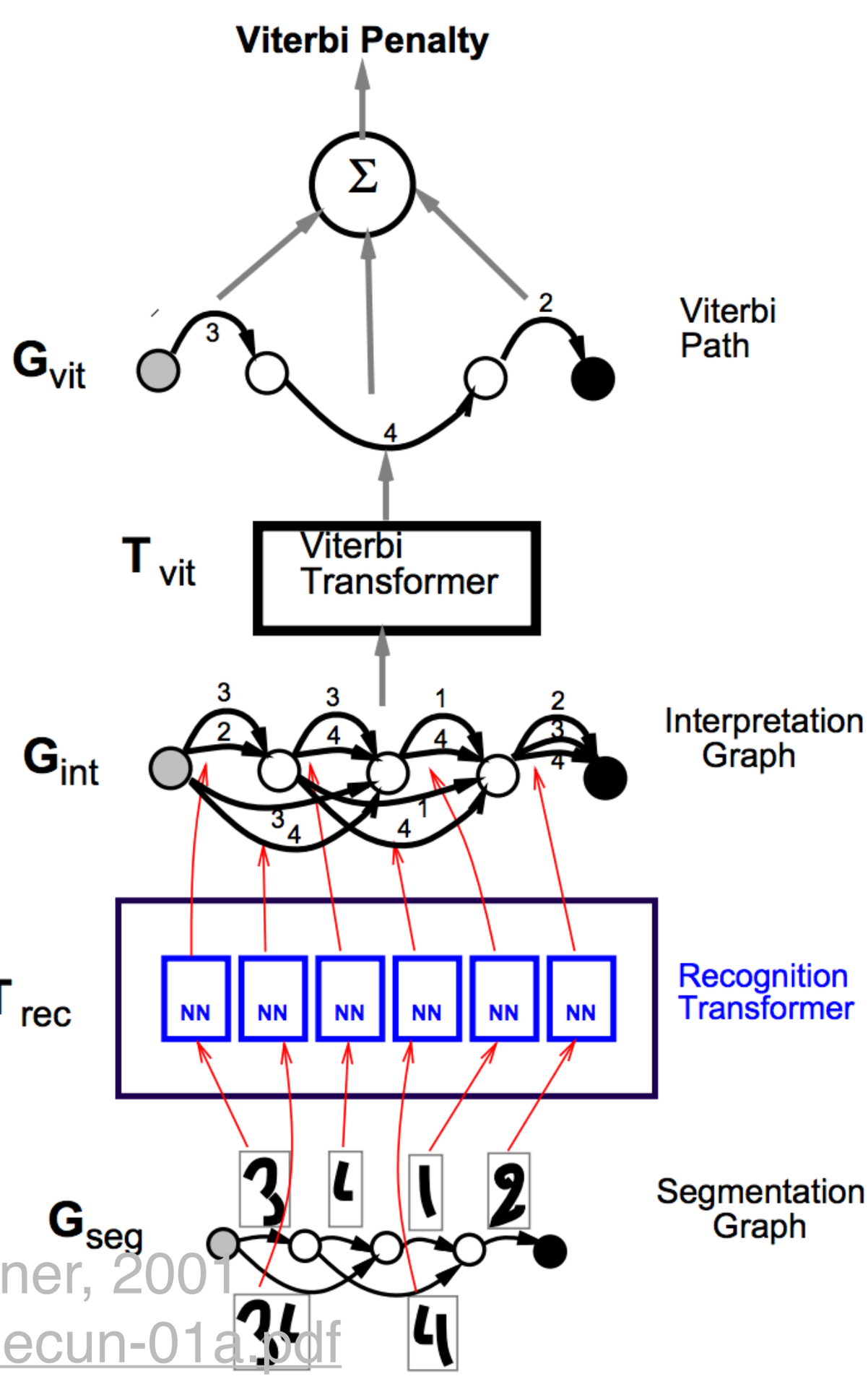


Le Cun, Bottou, Bengio, Haffner, 2001

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

MACHINE LEARNING
DEPARTMENT

9.3 Objectives

9 Deep Learning

Alexander Smola

Introduction to Machine Learning 10-701

http://alex.smola.org/teaching/10-701-15

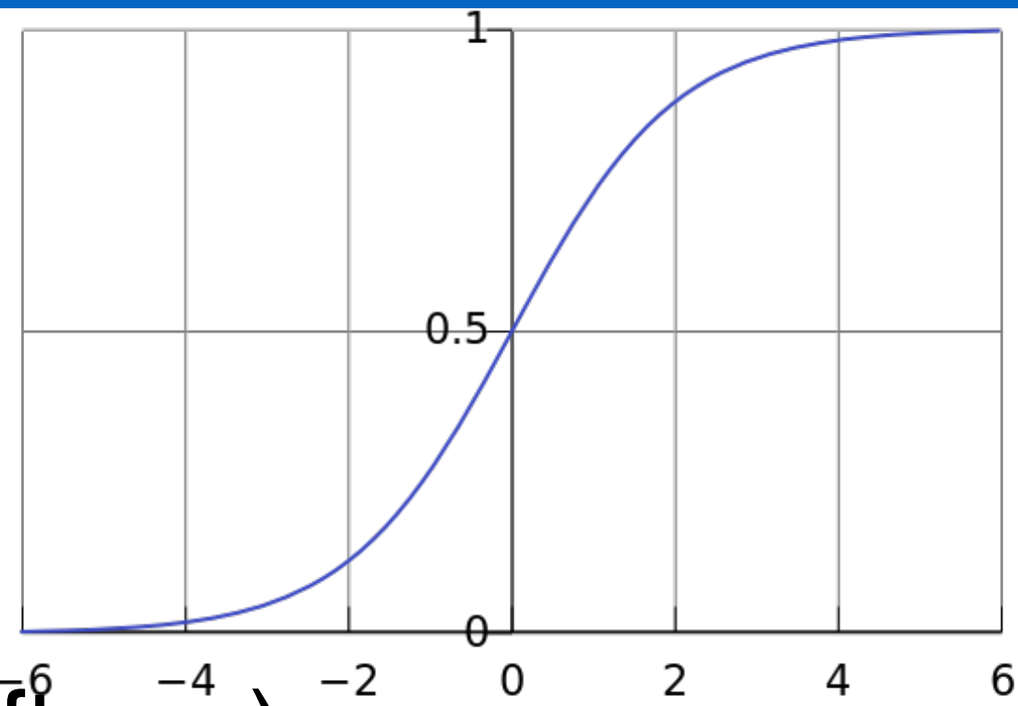
Carnegie Mellon University

Classification

- Binary classification

$$\log(1 + \exp(-yy_n))$$

Binary exponential model



- Multiclass classification (softmax)
  Multinomial exponential model

$$-\log p(y|y_n) = -\log \frac{e^{y_n[y]}}{\sum_{y'} e^{y_n[y']}} = \log \sum_{y'} e^{y_n[y']} - y_n[y]$$

- Pretty much anything else we did so far in 10-701

Regression

- Least mean squares

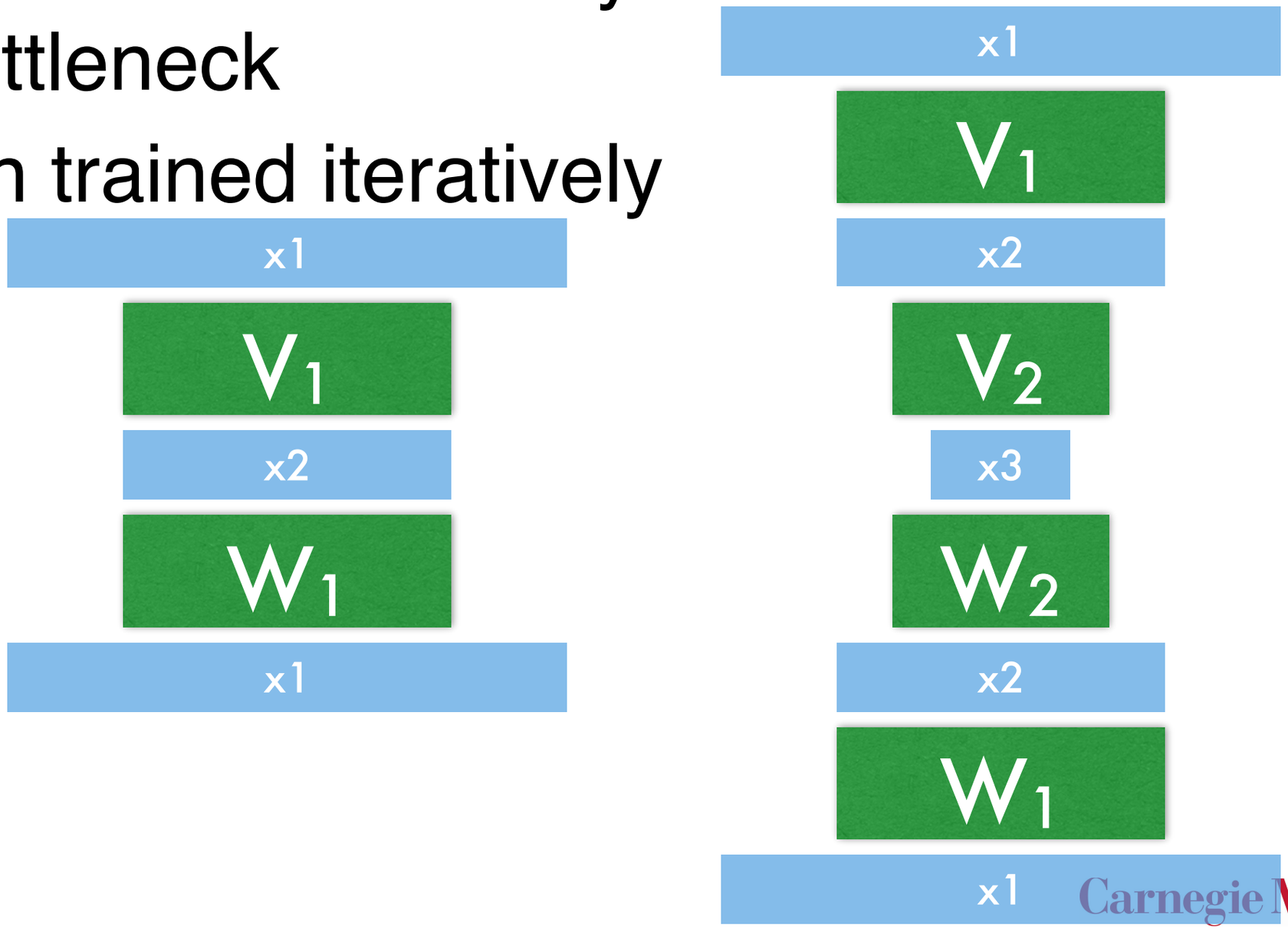
$$\frac{1}{2} \|y - y_n\|_2^2$$

  this works for vectors, too

- Applications
  - Stock market prediction (more on this later)
  - Image superresolution
    (regress from lower dimensional to higher dimensional image)
  - Recommendation and rating (Netflix)

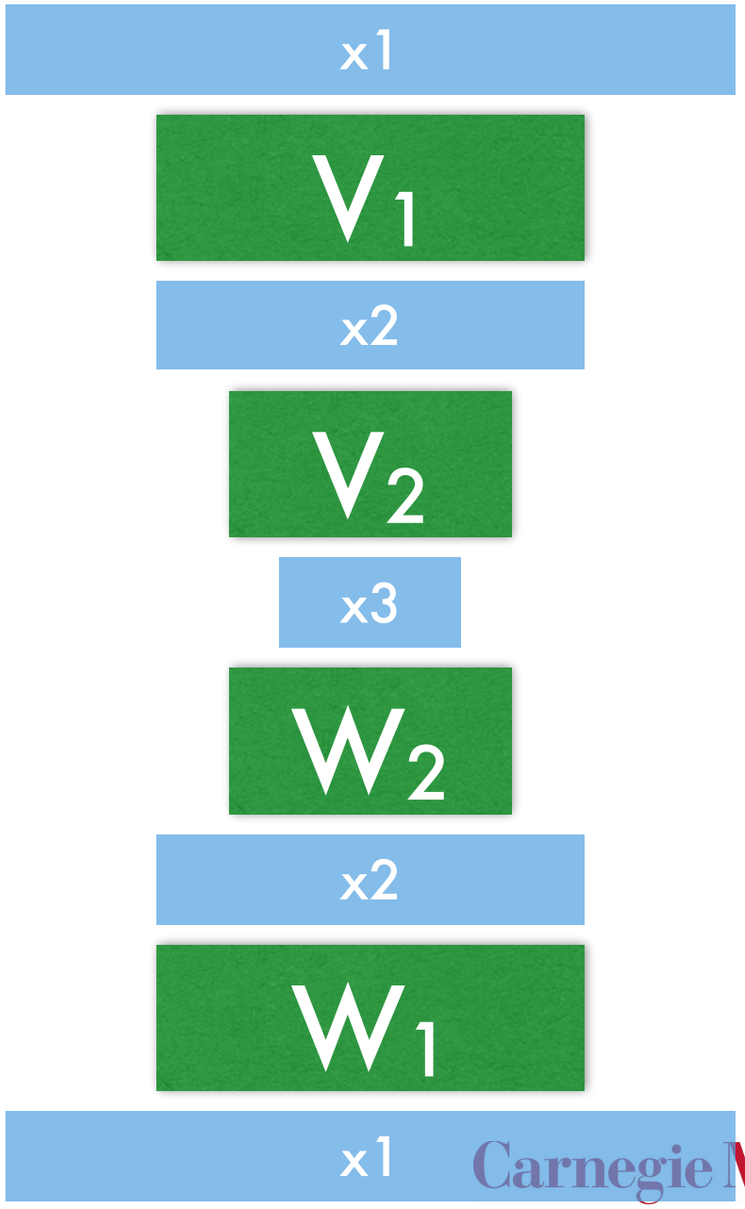
Autoencoder

- Regress from observation to itself ($y_n = x_1$)
- Lower-dimensional layer is bottleneck
- Often trained iteratively



Autoencoder

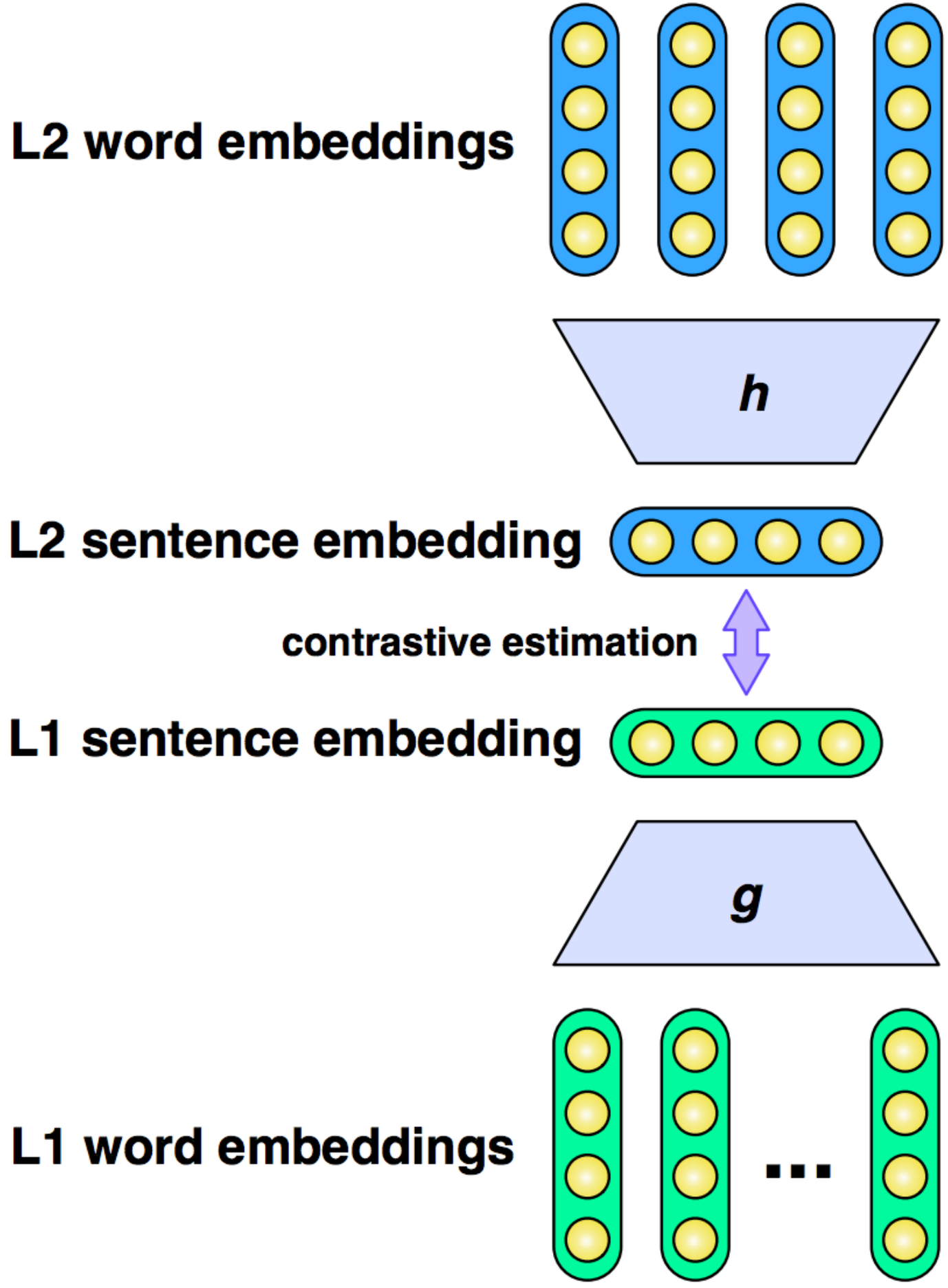
- Regress from observation to itself ($y_n = x_1$)
- Lower-dimensional layer is bottleneck
- Often trained iteratively
- Extracts approximate sufficient statistic of data



- Special case - PCA
  - linear mapping
  - only single layer

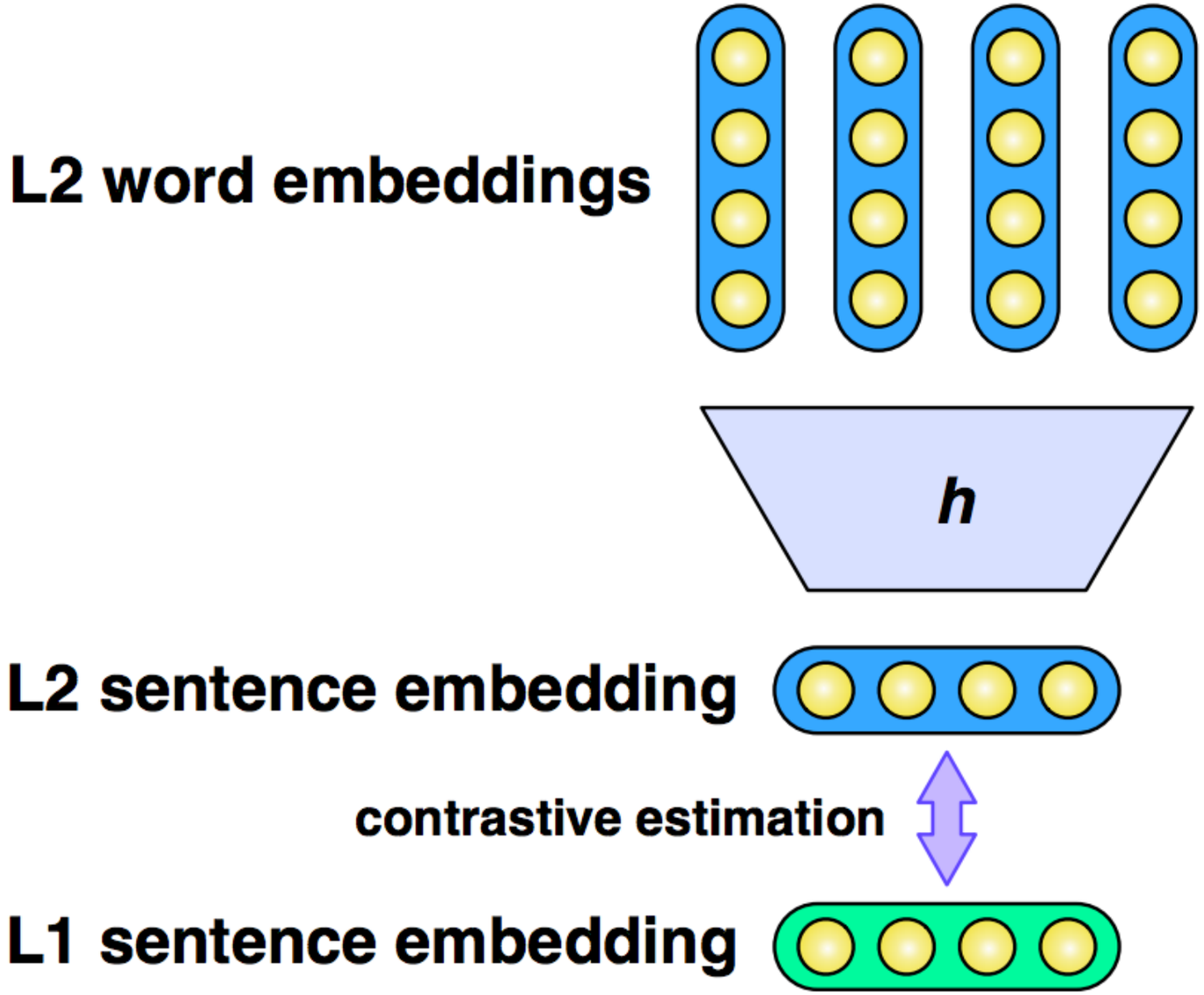
‘Synesthesia’

- 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



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

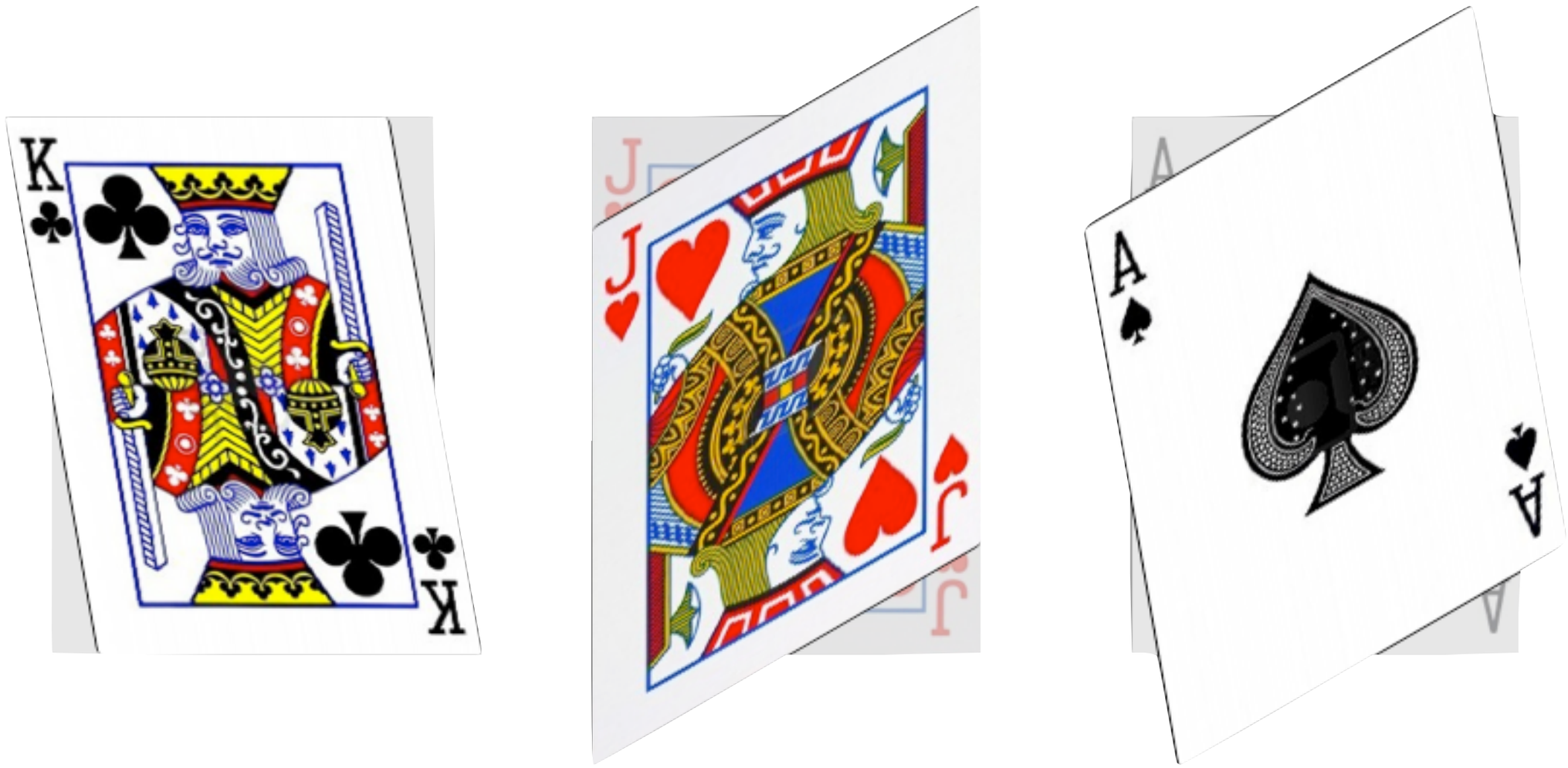
large margin of similarity

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

Synthetic Data Generation

- Dataset often has useful invariance
  - Images can be shifted, scaled, RGB transformed, blurred, sharpened, etc.
  - Speech can have echo, background noise, environmental noise
  - Text can have typos, omissions, etc.
- Generate data and train on extended noisy set
  - Record breaking speech recognition (Baidu)
  - Record breaking image recognition (Baidu, LeCun)
  - Can be very computationally expensive

Synthetic Data Generation



- Sample according to relevance of transform
- Similar to Virtual Support Vectors (Schölkopf, 1998)
- Training with input noise & regularization (Bishop, 1995)

MACHINE LEARNING
DEPARTMENT

9.4 Optimization

9 Deep Learning

Alexander Smola

Introduction to Machine Learning 10-701

http://alex.smola.org/teaching/10-701-15

Carnegie Mellon University

Stochastic Gradient Descent

- Update parameters according to

$$W_{ij} \leftarrow W_{ij} - \eta_{ij}(t) g_{ij}$$

  - Rate of decay
  - Adjust each layer
    - Adjust each parameter individually
    - Minibatch size
    - Momentum terms
- Lots of things that can (should) be adjusted (via 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

- Update parameters according to

$$W_{ij} \leftarrow W_{ij} - \eta_{ij}(t) g_{ij}$$

- Aggregate gradients before applying
  - Reduces variance in gradients
  - Better for vectorization (GPUs)
    vector, vector < vector, matrix < matrix, matrix
  - 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)
- **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**

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

  risky since decay could be to 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

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

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

Dropout

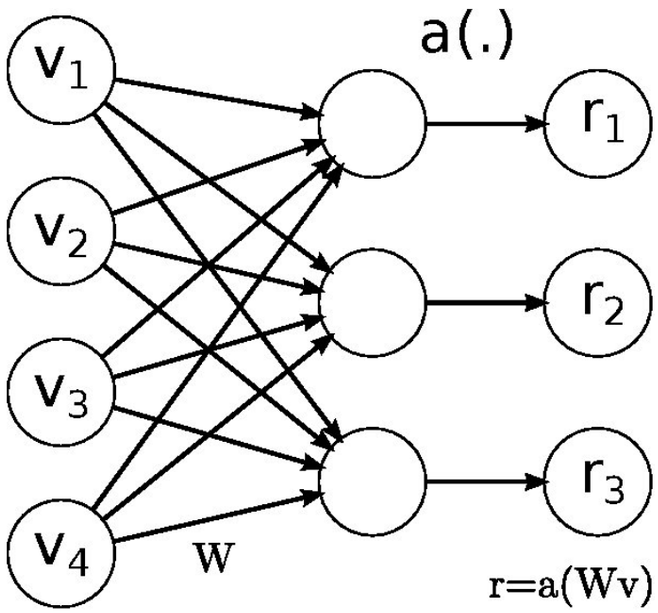
- Avoid parameter sensitivity
  (small changes in value shouldn't change result)
- Distributed representation
  (information carried by more than 1 dimension)
- Randomized sparsification

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

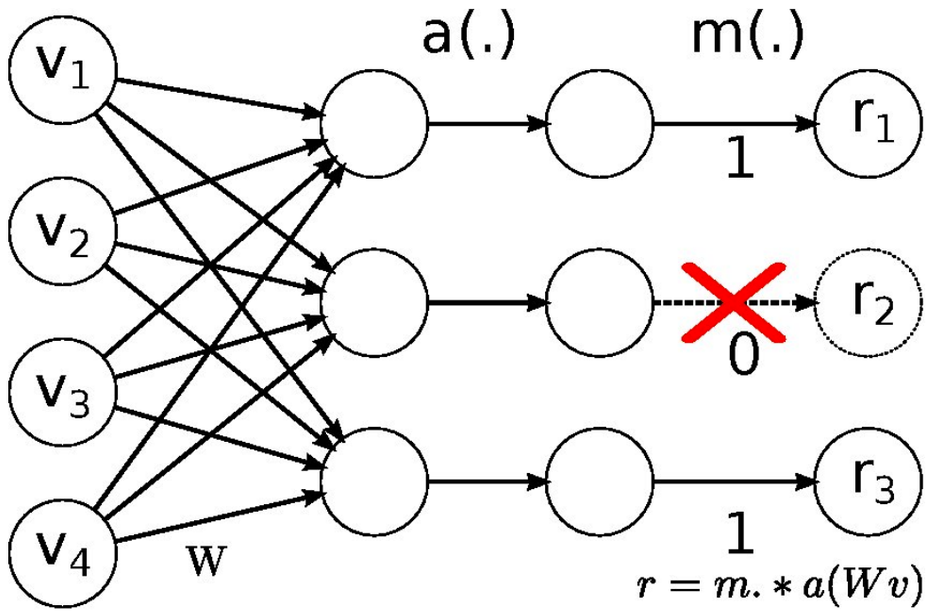
- Same trick works for matrix W, too: DropConnect slightly better performance … http://cs.nyu.edu/~wanli/dropc/

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

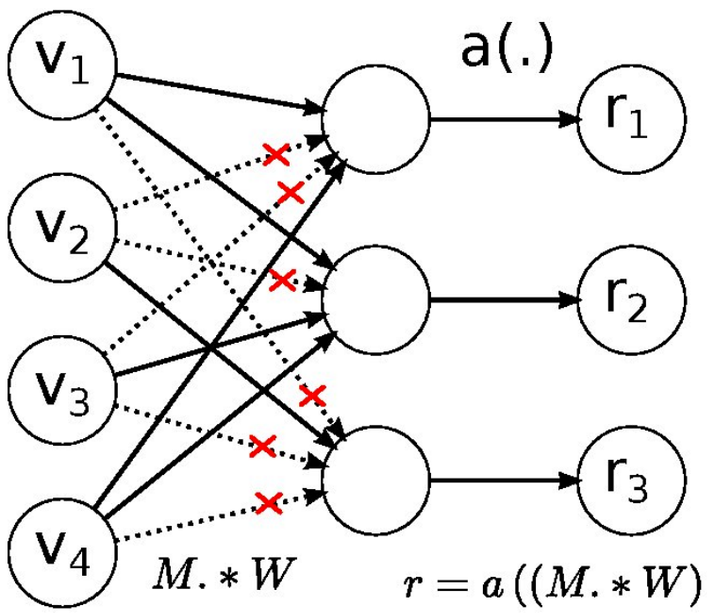
Dropout & DropConnect



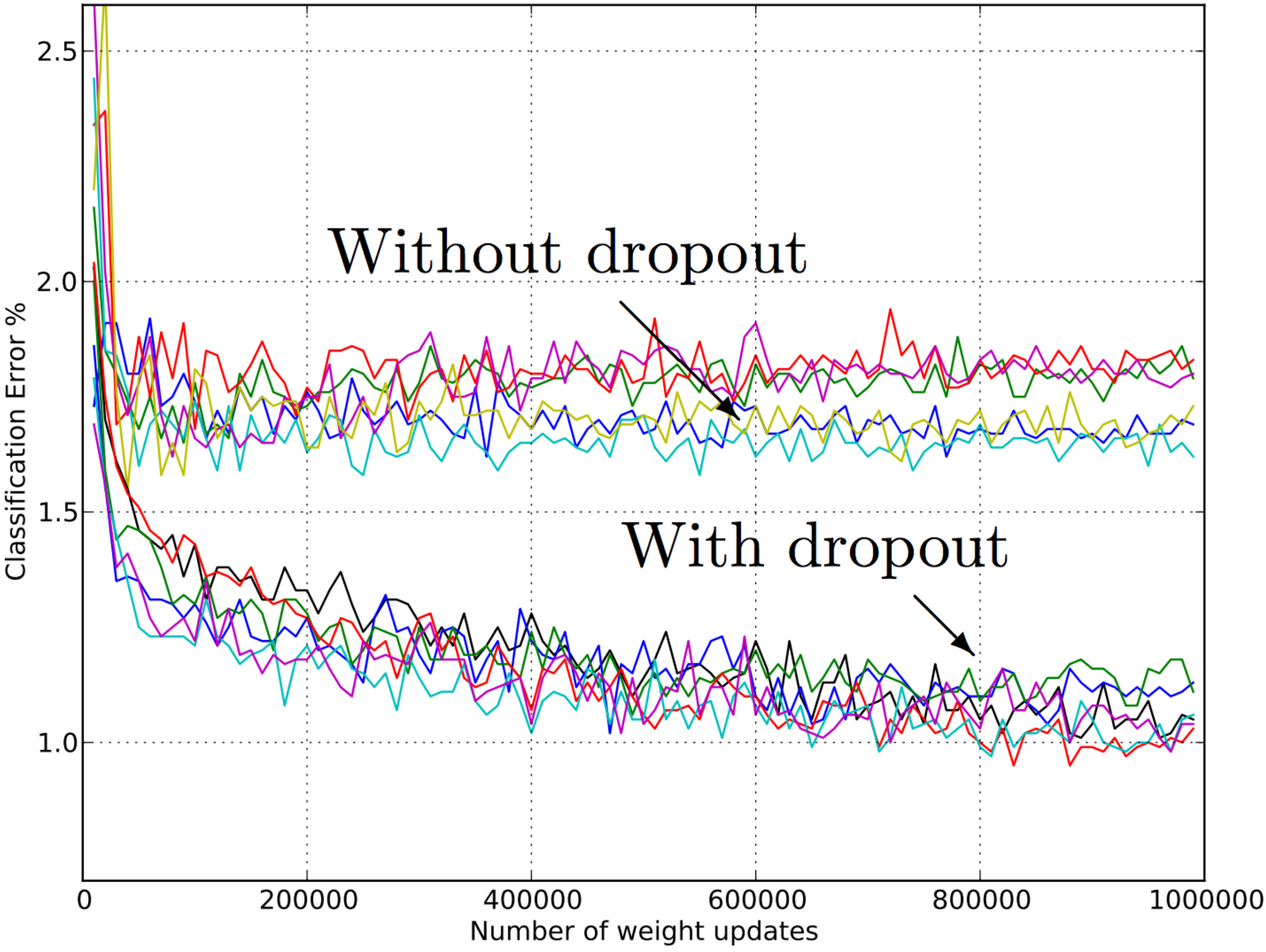
Regular



Dropout



DropConnect


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

MACHINE LEARNING
DEPARTMENT

9.5 Memory

9 Deep Learning

Alexander Smola

Introduction to Machine Learning 10-701

http://alex.smola.org/teaching/10-701-15

Carnegie Mellon University

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 programming/stepwise prediction

Autoregressive Models / RNN

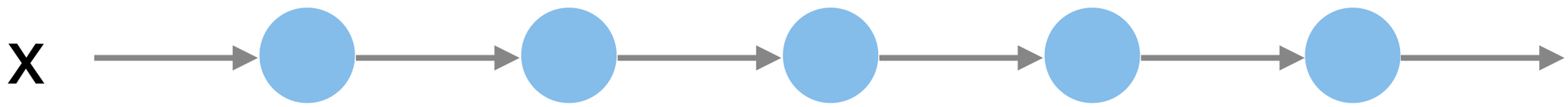
- Time series of observations

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

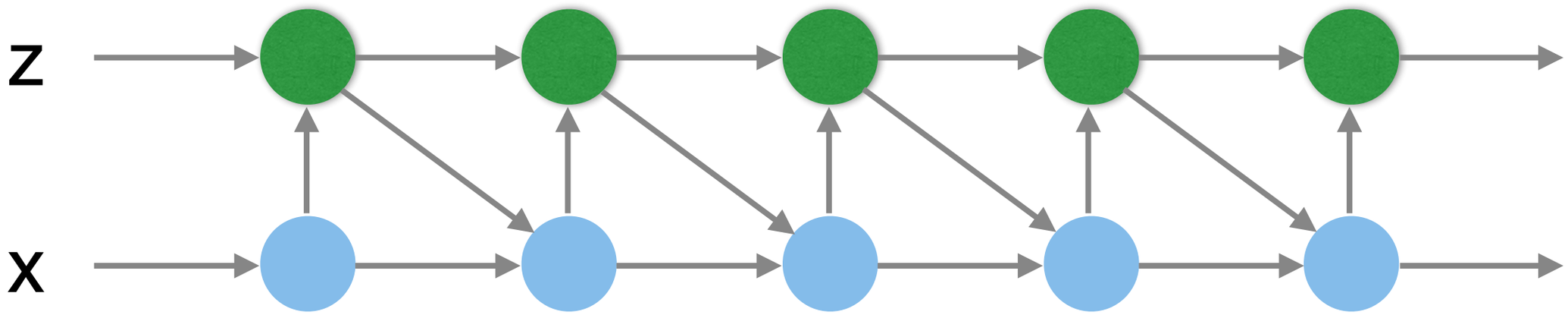
- Estimate $x_{t+1} = f(x_t, \ldots, x_{t-\tau})$ e.g. via deep net
- Problem
  - Hard to encode latent state (e.g. parity)
  - Hard to encode long range context/knowledge
- Solution - latent state

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

Autoregressive Models / RNN



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

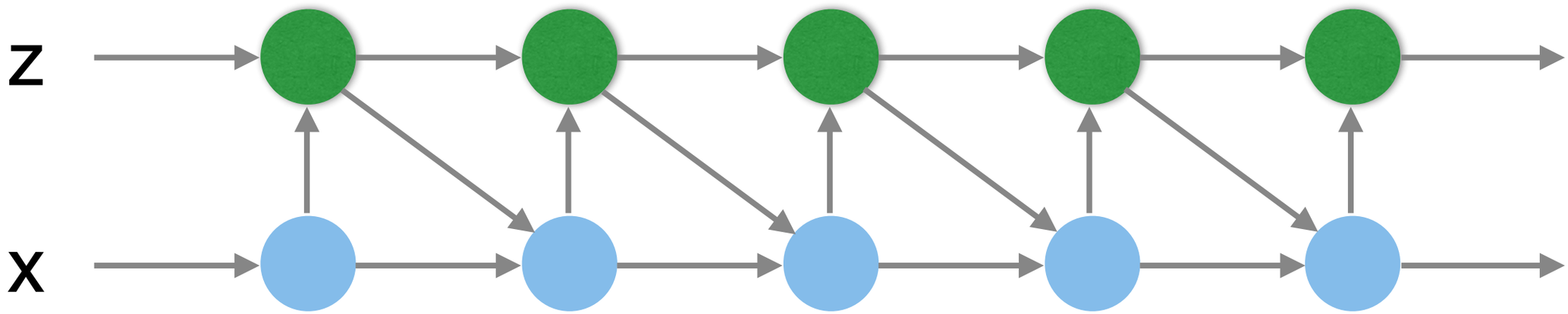


$$x_{t+1} = f(x_t, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})$$

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

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

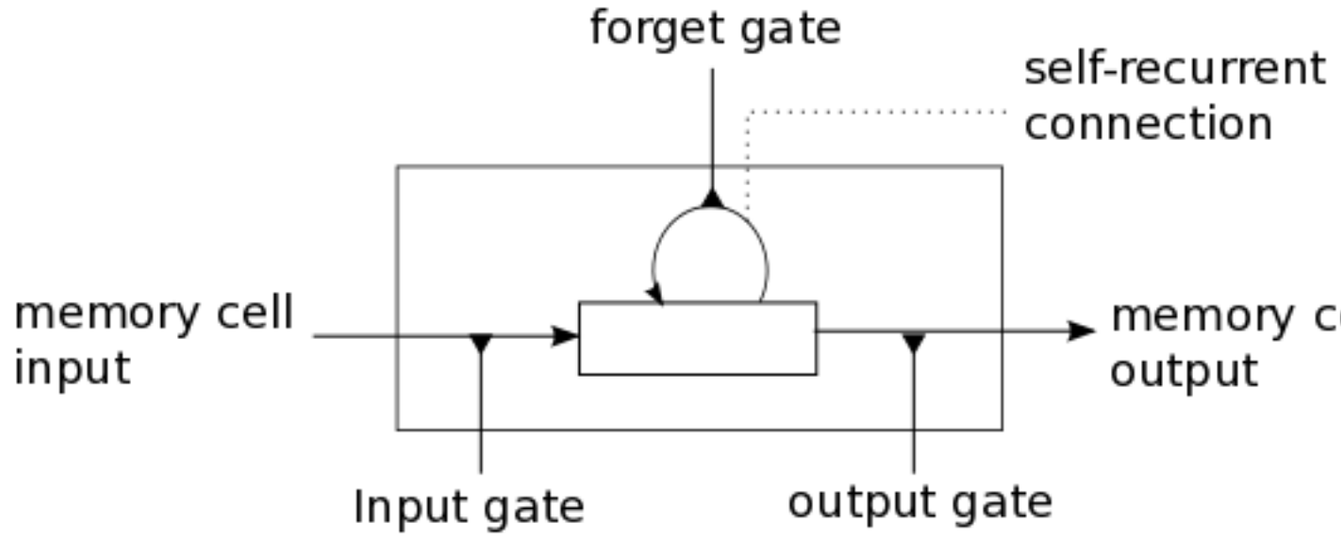


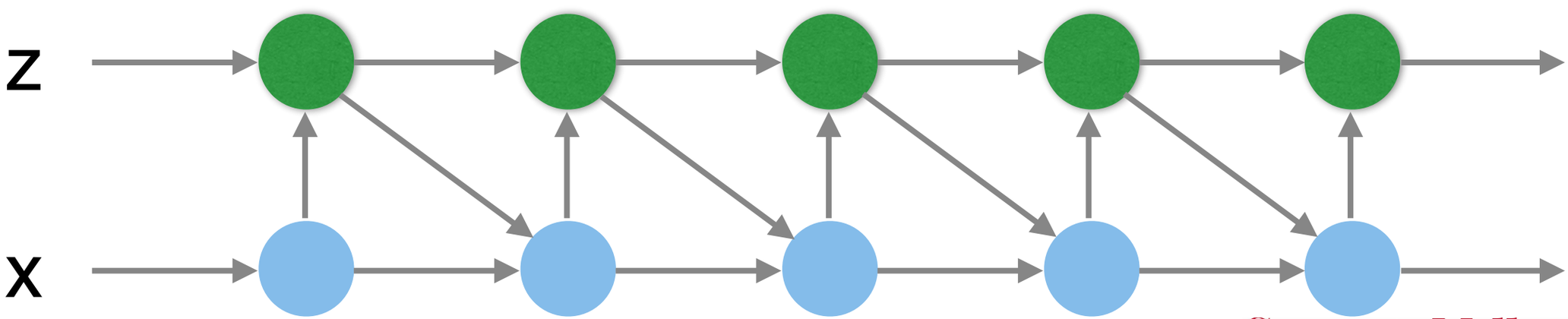
$$x_{t+1} = f(x_t, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})$$

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

LSTM (Long Short Term Memory)

- Sequence of observations
  Latent state has custom update semantics
  (like a memory cell), Hochreiter & Schmidhuber

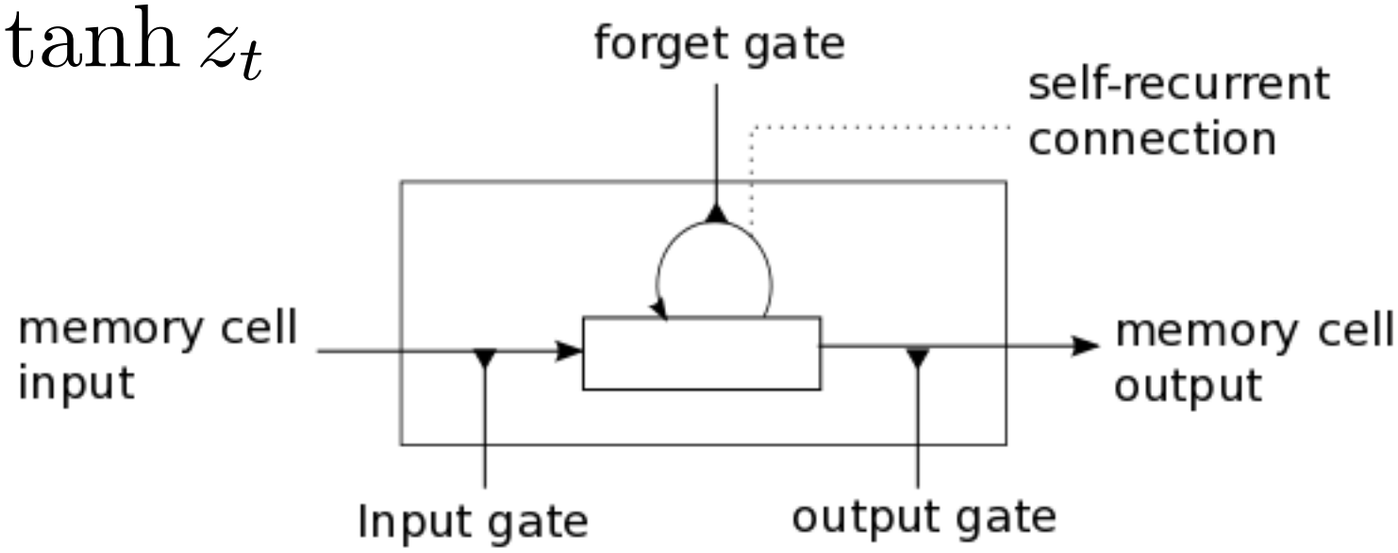




LSTM (Long Short Term Memory)

- 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

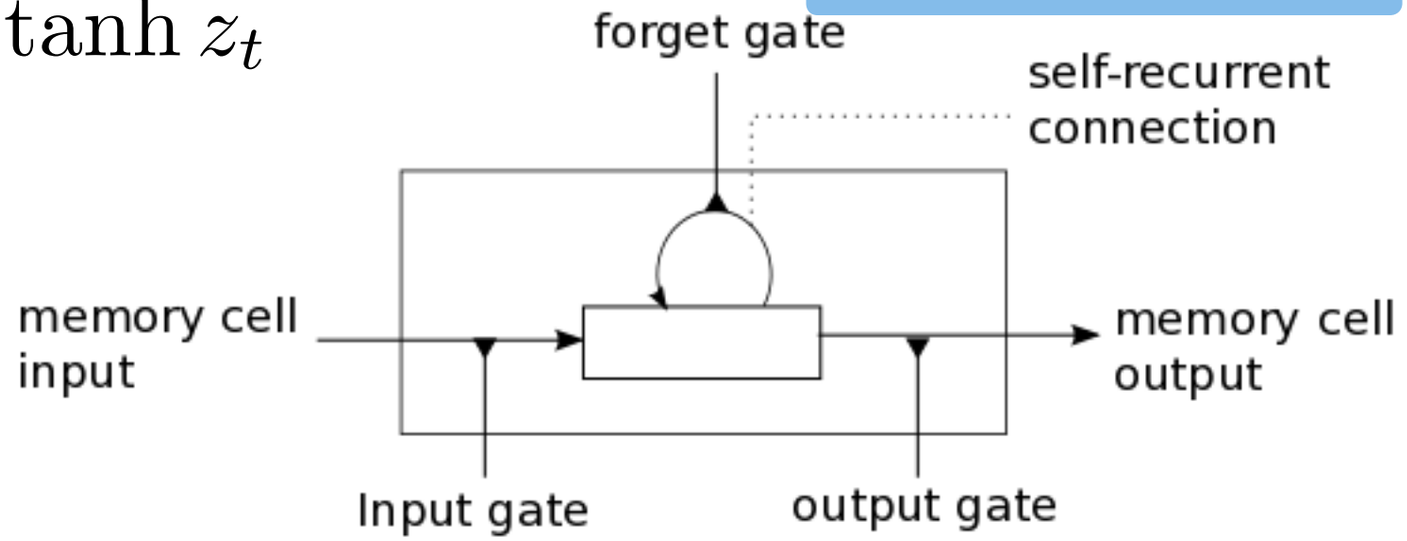
$$i_t = \sigma(W_i(x_t, h_{t-1}) + b_i)$$ input

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

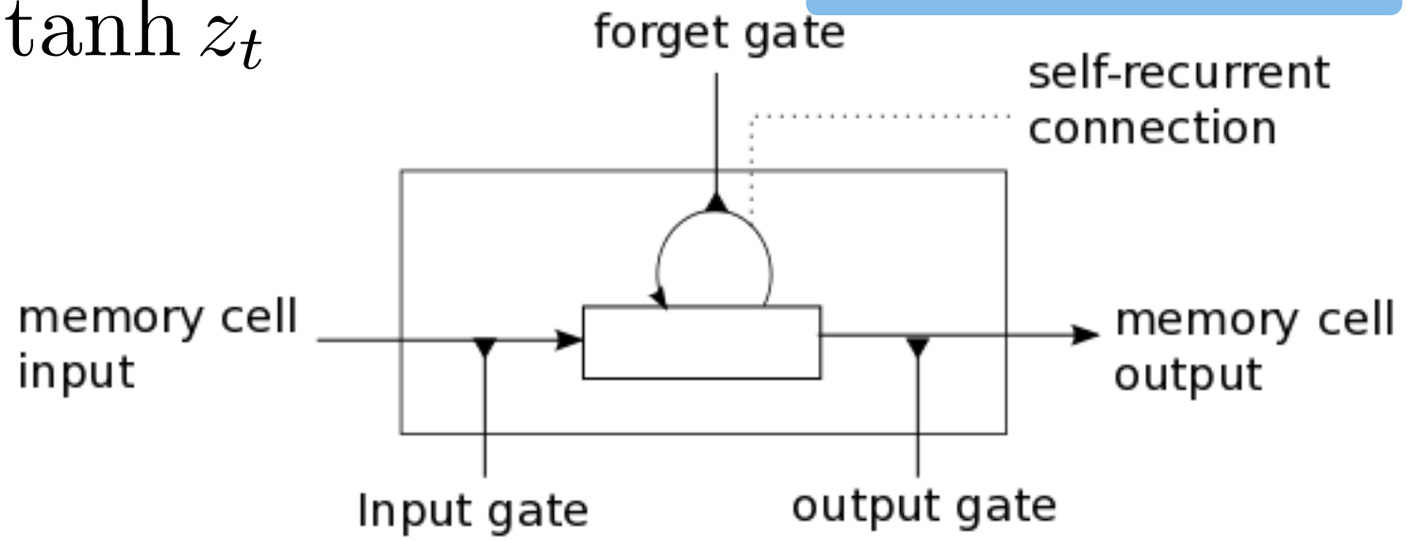
$$i_t = \sigma(W_i(x_t, h_{t-1}) + b_i)$$ input

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

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

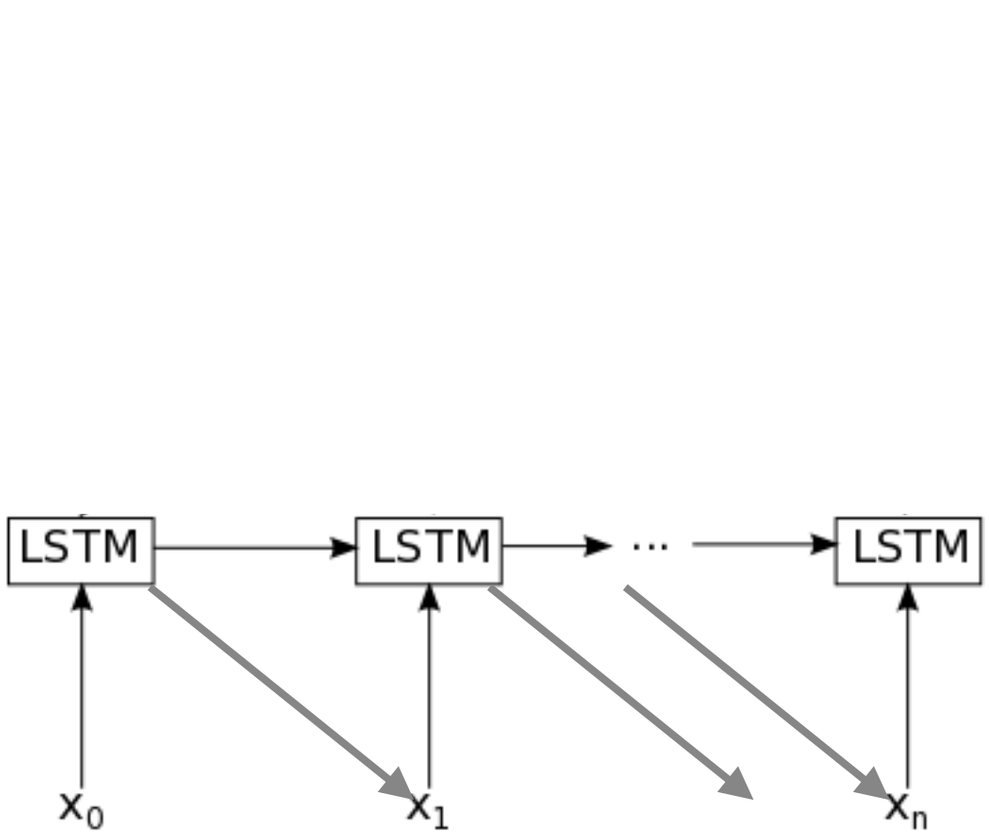
$$o_t = \sigma(W_o(x_t, h_{t-1}, z_t) + b_f)$$ output gate

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

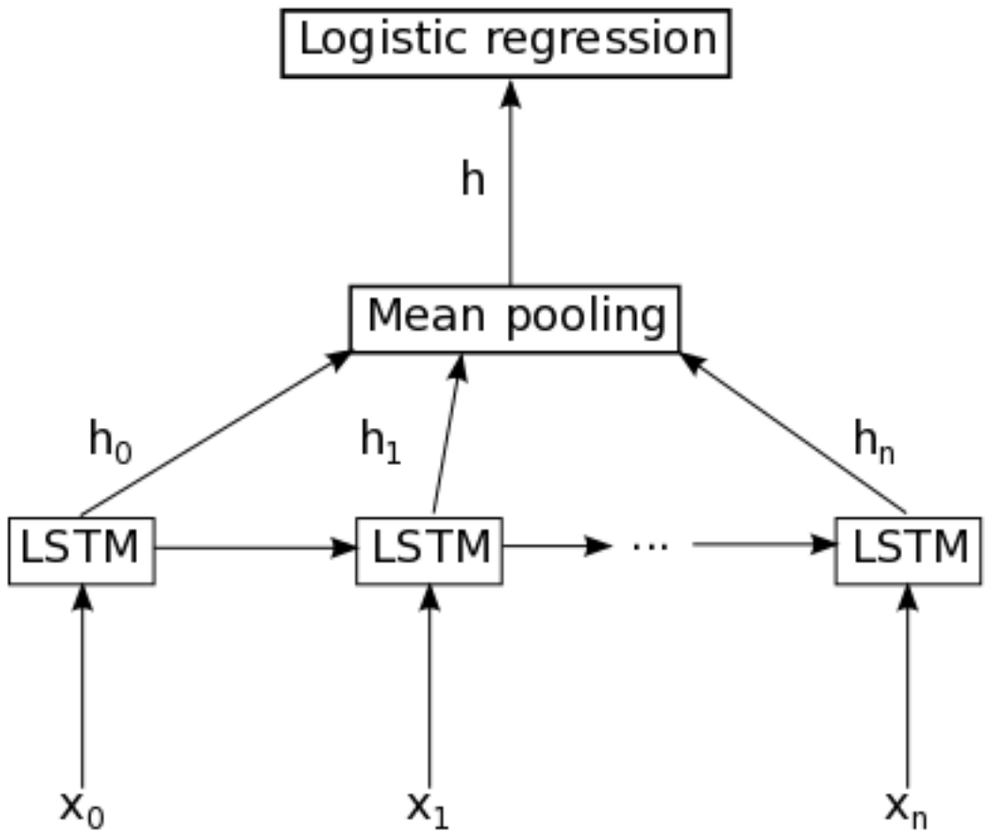


LSTM (Long Short Term Memory)

- Sequence of observations
  Latent state has custom update semantics



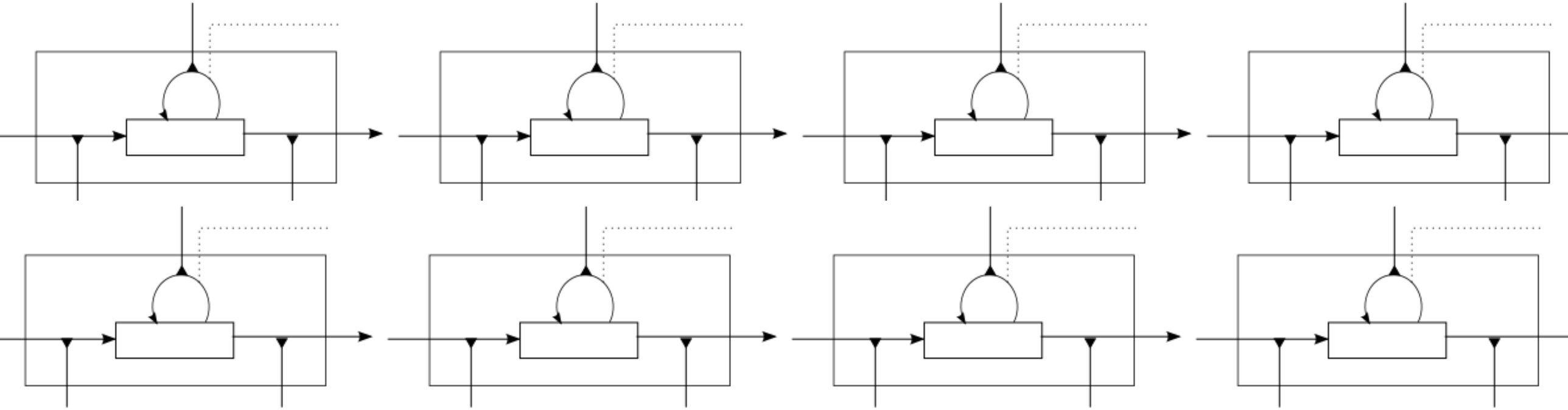
sequence generation



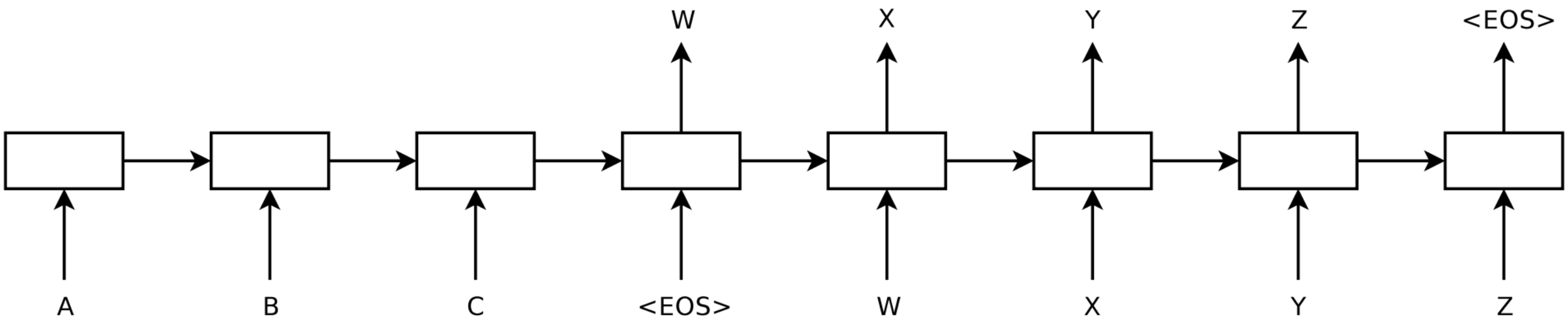
sequence classification

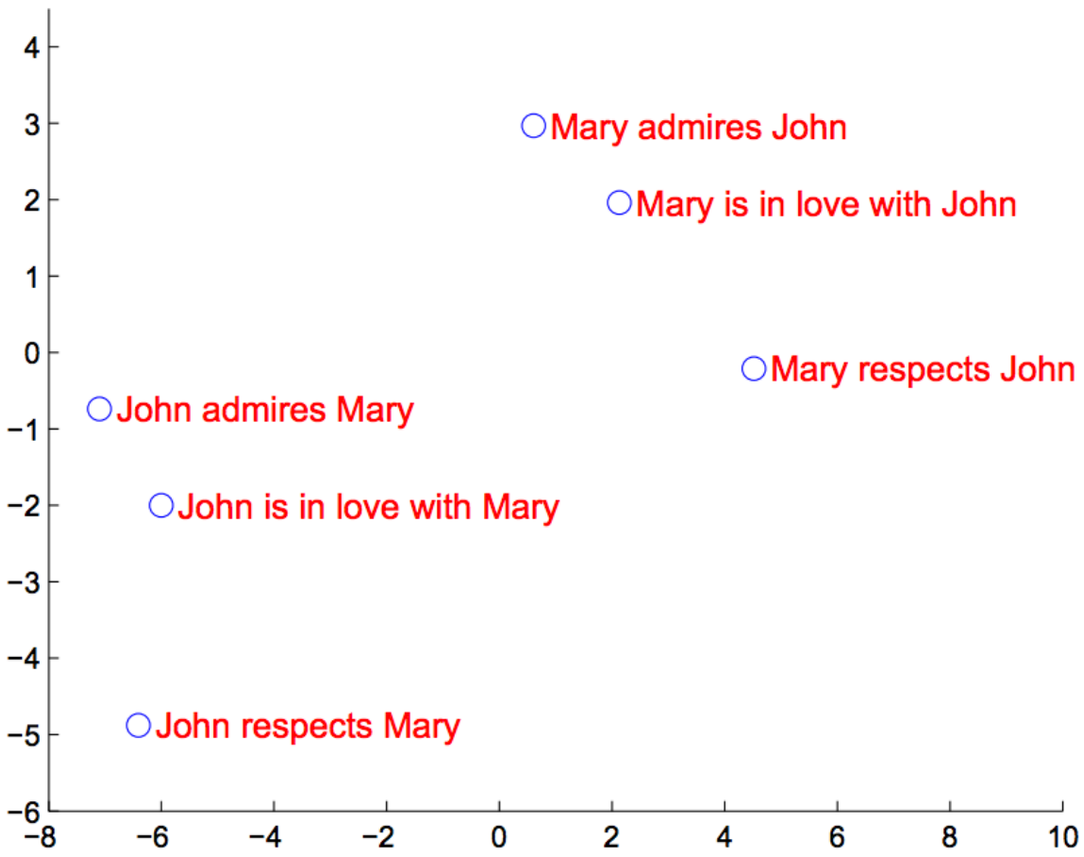
LSTM (Long Short Term Memory)

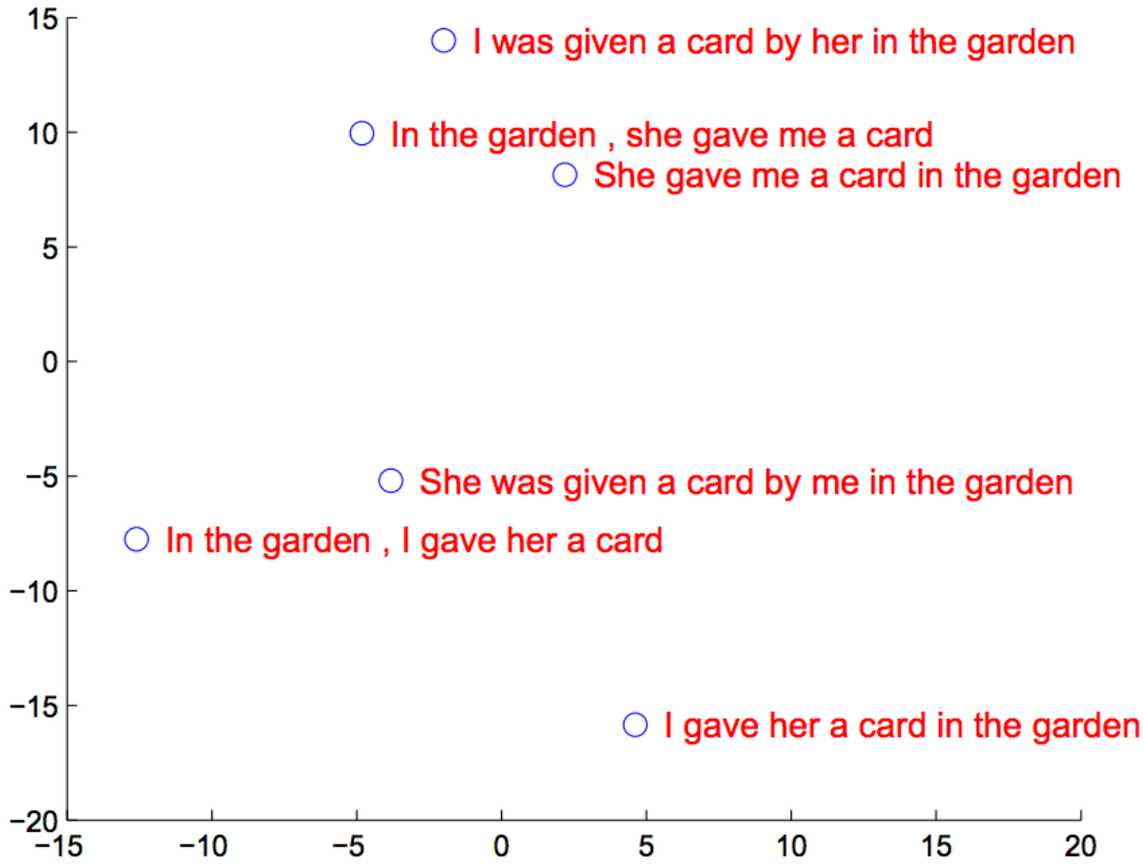
- Group LSTM cells into layer
- Multiple layers
- Can model different scales of dynamics



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

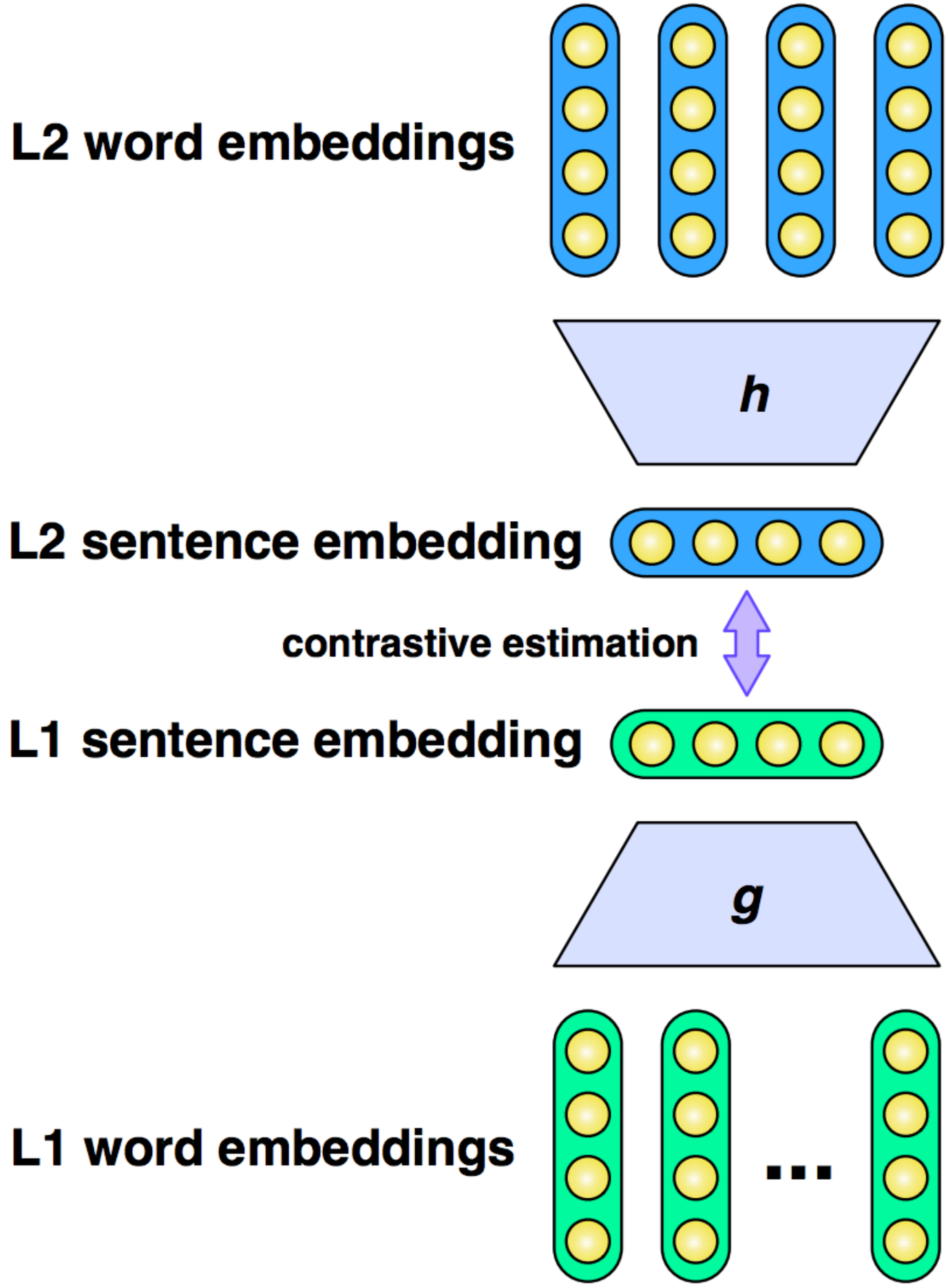






‘Synesthesia’

- 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)

MACHINE LEARNING
DEPARTMENT

9.6 Toolkits

9 Deep Learning

Alexander Smola

Introduction to Machine Learning 10-701

http://alex.smola.org/teaching/10-701-15

Carnegie Mellon University

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, …