Deep Learning

NIPS’2015 Tutorial

Geoff Hinton, Yoshua Bengio & Yann LeCun

CIFAR

CANADIAN INSTITUTE
for ADVANCED RESEARCH

Breakthrough

Deep Learning: machine learning algorithms based on learning multiple levels of representation / abstraction.

Amazing improvements in error rate in object recognition, object detection, speech recognition, and more recently, in natural language processing / understanding

Machine Learning, AI & No Free Lunch

- Four key ingredients for ML towards AI

1. Lots & lots of data
2. Very flexible models
3. Enough computing power
4. Powerful priors that can defeat the curse of dimensionality

Bypassing the curse of dimensionality

We need to build compositionality into our ML models

Just as human languages exploit compositionality to give representations and meanings to complex ideas

Exploiting compositionality gives an exponential gain in representational power

(1) Distributed representations / embeddings: feature learning

(2) Deep architecture: multiple levels of feature learning

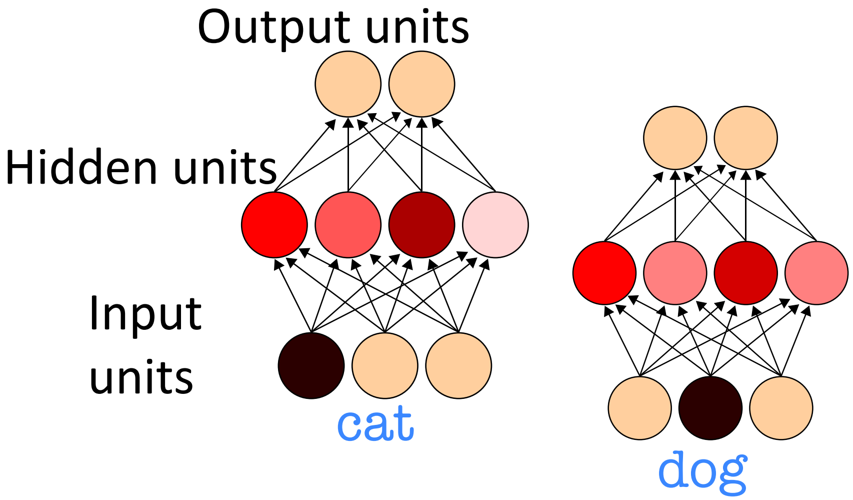
Additional prior: compositionality is useful to describe the world around us efficiently

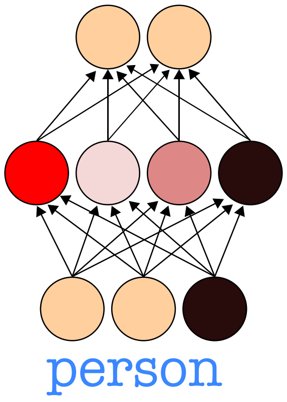
Classical Symbolic AI vs Learning Distributed Representations

- Two symbols are equally far from each other
- Concepts are not represented by symbols in our brain, but by patterns of activation

(Connectionism, 1980's)

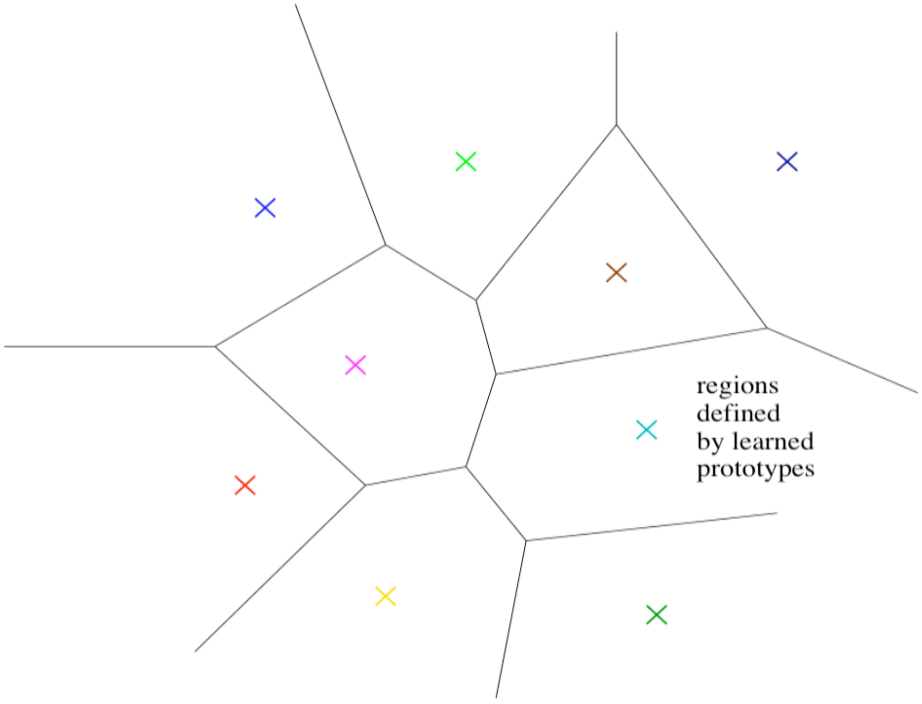
Geoffrey Hinton



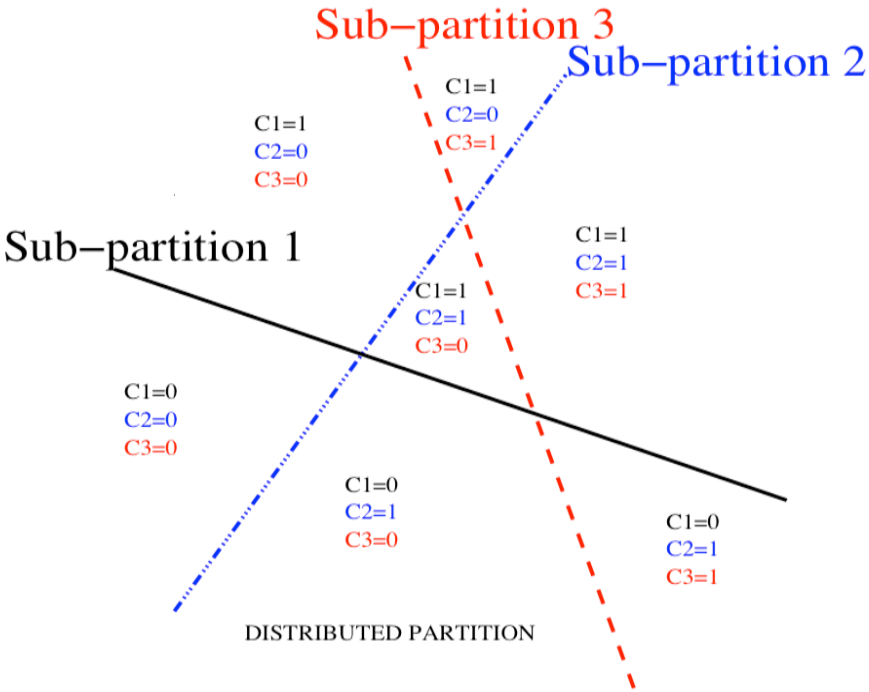


David Rumelhart

Exponential advantage of distributed representations



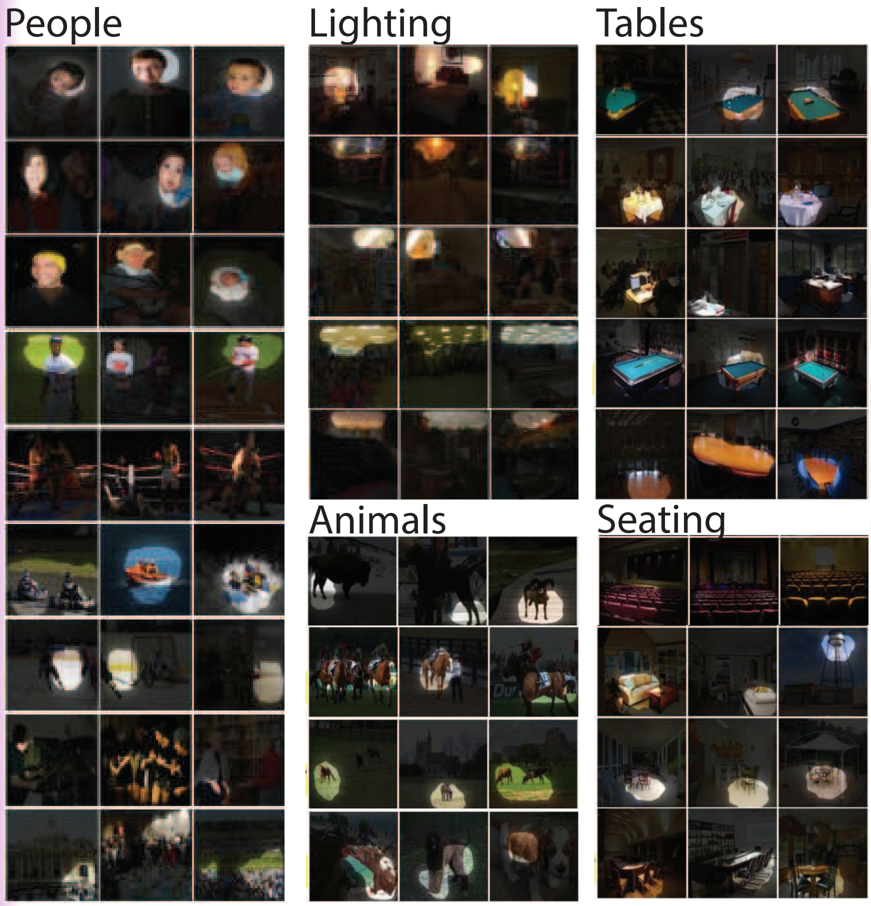
LOCAL PARTITION

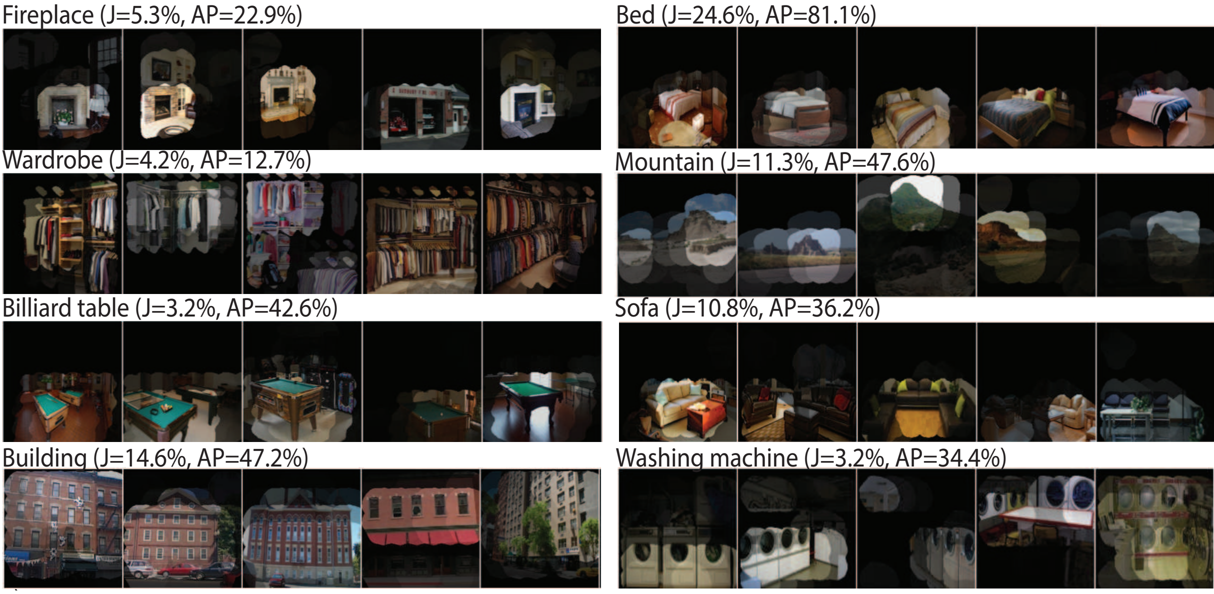


Learning a **set of parametric features** that are not mutually exclusive can be exponentially more statistically efficient than having nearest-neighbor-like or clustering-like models

Hidden Units Discover Semantically Meaningful Concepts

- *Zhou et al & Torralba, arXiv1412.6856* submitted to ICLR 2015
- Network trained to recognize places, not objects




Fireplace (J=5.3%, AP=22.9%)
Bed (J=24.6%, AP=81.1%)
Wardrobe (J=4.2%, AP=12.7%)
Mountain (J=11.3%, AP=47.6%)
Billiard table (J=3.2%, AP=42.6%)
Sofa (J=10.8%, AP=36.2%)
Building (J=14.6%, AP=47.2%)
Washing machine (J=3.2%, AP=34.4%)

Each feature can be discovered without the need for seeing the exponentially large number of configurations of the other features

- Consider a network whose hidden units discover the following features:
  - Person wears glasses
  - Person is female
  - Person is a child
  - Etc.

If each of *n* feature requires *O(k)* parameters, need *O(nk)* examples

Non-parametric methods would require $O(n^d)$ examples

Exponential advantage of distributed representations

- *Bengio 2009* (Learning Deep Architectures for AI, F & T in ML)
- *Montufar & Morton 2014* (When does a mixture of products contain a product of mixtures? SIAM J. Discr. Math)

- Longer discussion and relations to the notion of priors: ***Deep Learning***, to appear, MIT Press.

- Prop. 2 of ***Pascanu, Montufar & Bengio ICLR'2014***: number of pieces distinguished by 1-hidden-layer rectifier net with *n* units and *d* inputs (i.e. *O(nd)* parameters) is

$$\sum_{j=0}^{d} \binom{n}{j} = O(n^d)$$

Deep Learning: Automating Feature Discovery

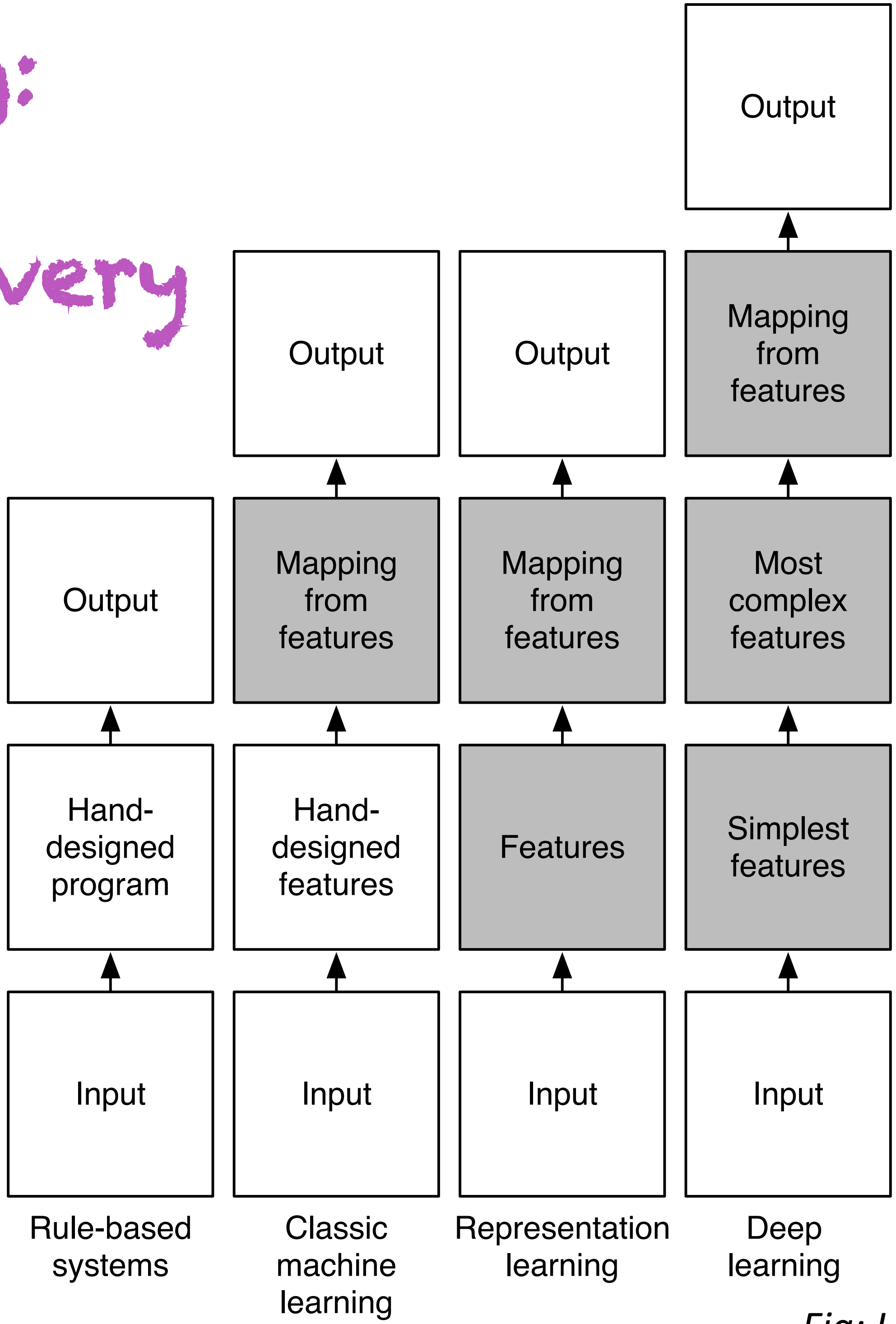


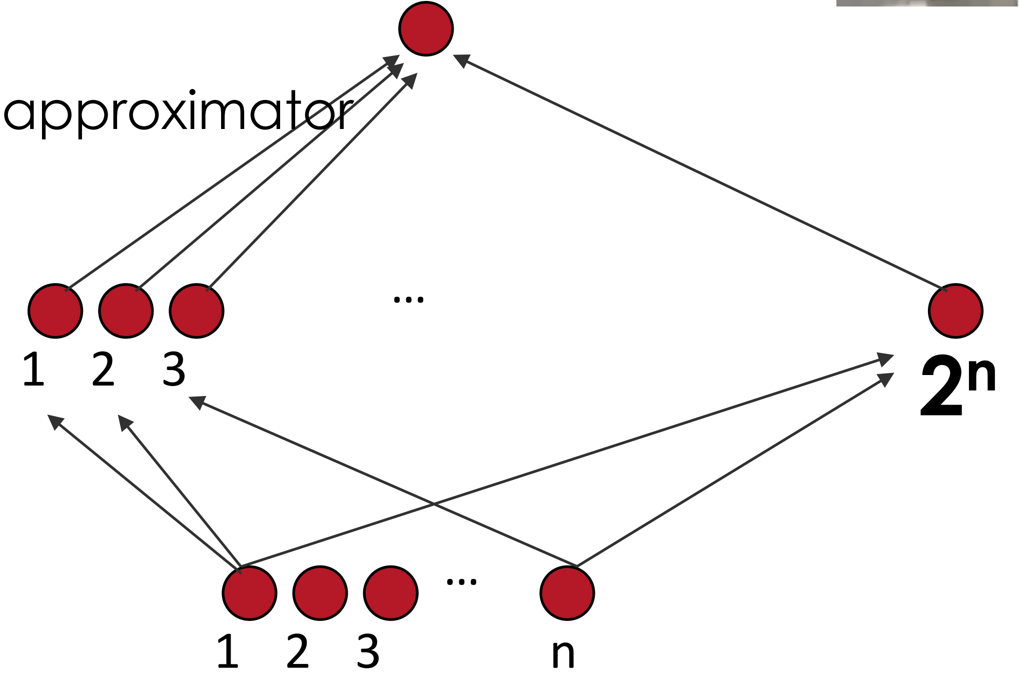
Fig: I. Goodfellow

Exponential advantage of depth

Theoretical arguments:

2 layers of { Logic gates / Formal neurons / RBF units } = universal approximator

RBMs & auto-encoders = universal approximator



Theorems on advantage of depth:

(Hastad et al 86 & 91, Bengio et al 2007, Bengio & Delalleau 2011, Martens et al 2013, Pascanu et al 2014, Montufar et al **NIPS 2014**)

Some functions compactly represented with k layers may require exponential size with 2 layers

Why does it work? No Free Lunch

- It only works because we are making some assumptions about the data generating distribution
- Worse-case distributions still require exponential data
- But the world has structure and we can get an exponential gain by exploiting some of it

Exponential advantage of depth

- Expressiveness of deep networks with piecewise linear activation functions: exponential advantage for depth *(Montufar et al, NIPS 2014)*
- Number of pieces distinguished for a network with depth *L* and n_i units per layer is at least

$$\left(\prod_{i=1}^{L-1} \left\lfloor \frac{n_i}{n_0} \right\rfloor^{n_0}\right) \sum_{j=0}^{n_0} \binom{n_L}{j}$$

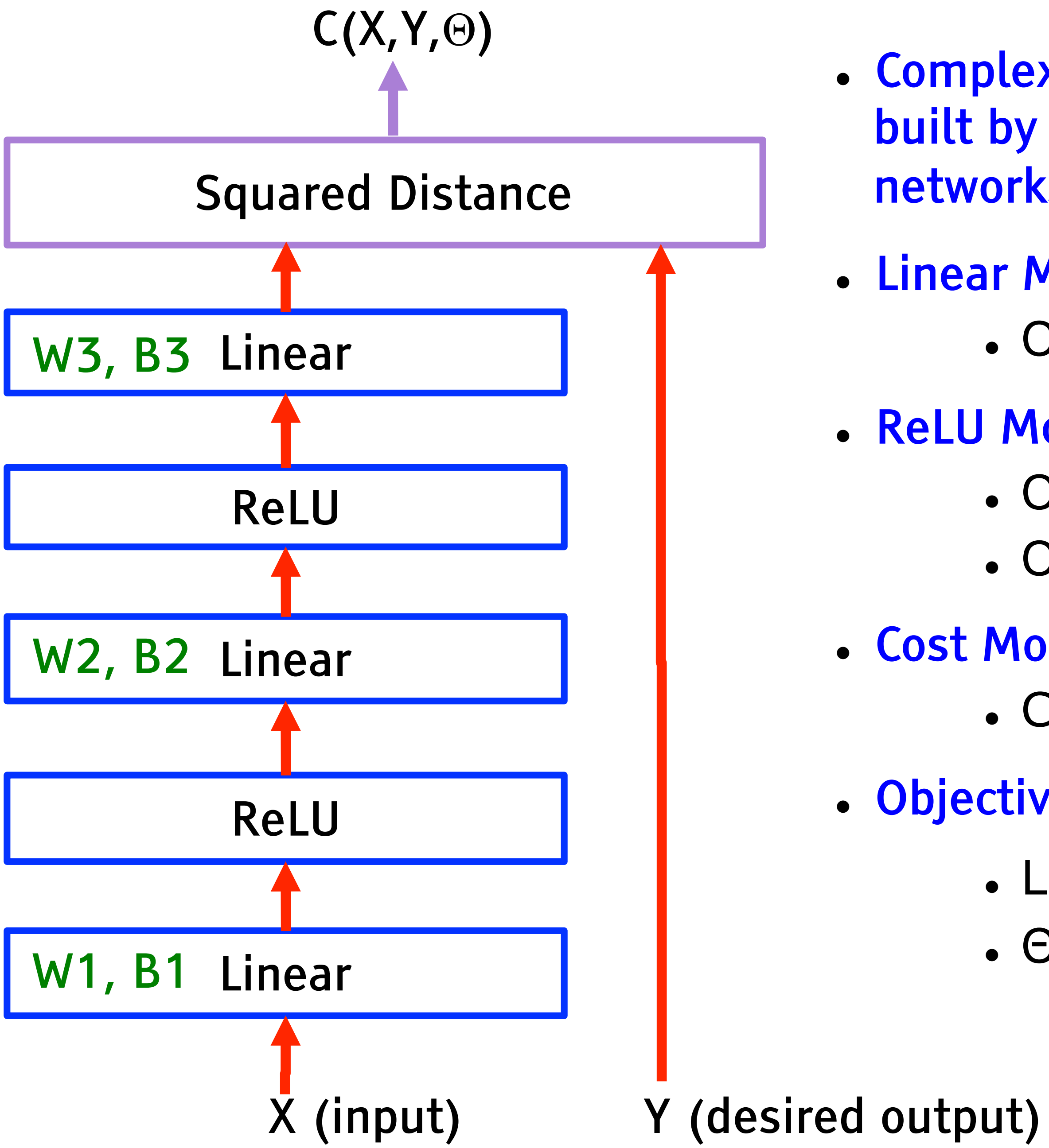
or, if hidden layers have width *n* and input has size n_0

$$\Omega\left(\left({}^{n}\!/_{n_0}\right)^{(L-1)n_0} n^{n_0}\right)$$

Backprop
(modular approach)

Typical Multilayer Neural Net Architecture

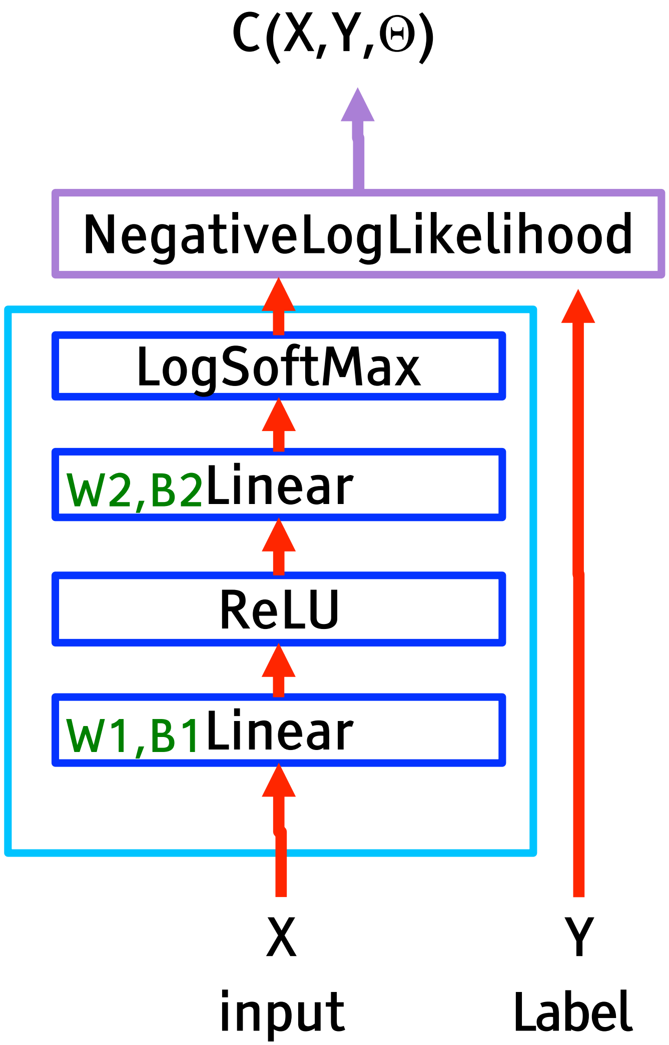
Y LeCun



- Complex learning machines can be built by assembling modules into networks
- Linear Module
    - Out = W.In+B
- ReLU Module (Rectified Linear Unit)
    - $Out_i = 0$ if $In_i<0$
    - $Out_i = In_i$ otherwise
- Cost Module: Squared Distance
    - $C = ||In1 - In2||^2$
- Objective Function
    - $L(\Theta)=1/p \sum_k C(X^k,Y^k,\Theta)$
    - $\Theta = (W1,B1,W2,B2,W3,B3)$

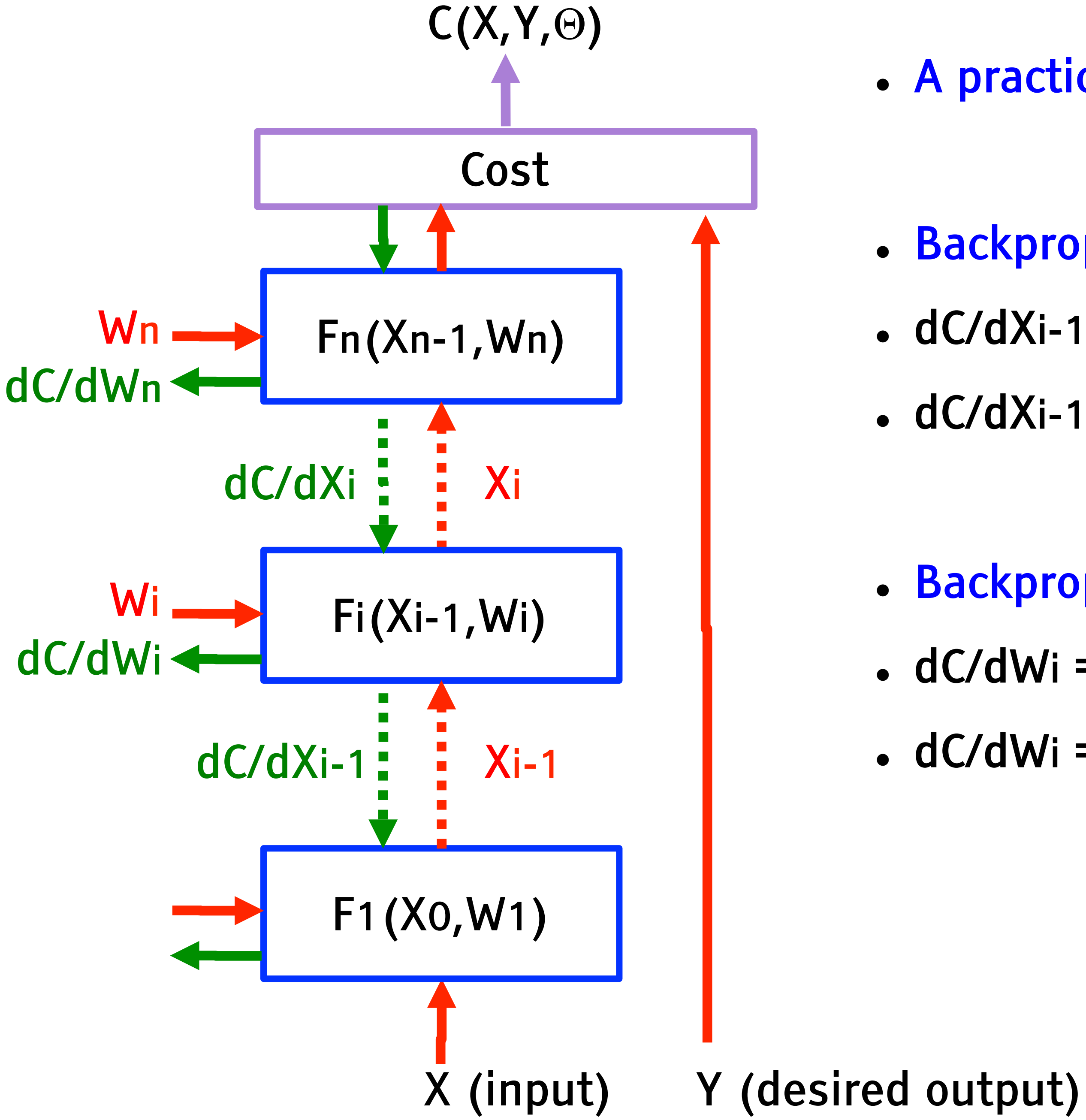
Building a Network by Assembling Modules

- All major deep learning frameworks use modules (inspired by SN/Lush, 1991)
  - Torch7, Theano, TensorFlow....



```
-- sizes
ninput = 28*28  -- e.g. for MNIST
nhidden1 = 1000
noutput = 10

-- network module
net = nn.Sequential()
net:add(nn.Linear(ninput, nhidden))
net:add(nn.Threshold())
net:add(nn.Linear(nhidden, noutput))
net:add(nn.LogSoftMax()))

-- cost module
cost = nn.ClassNLLCriterion()

-- get a training sample
input = trainingset.data[k]
target = trainingset.labels[k]

-- run through the model
output = net:forward(input)
c = cost:forward(output, target)
```

Computing Gradients by Back-Propagation



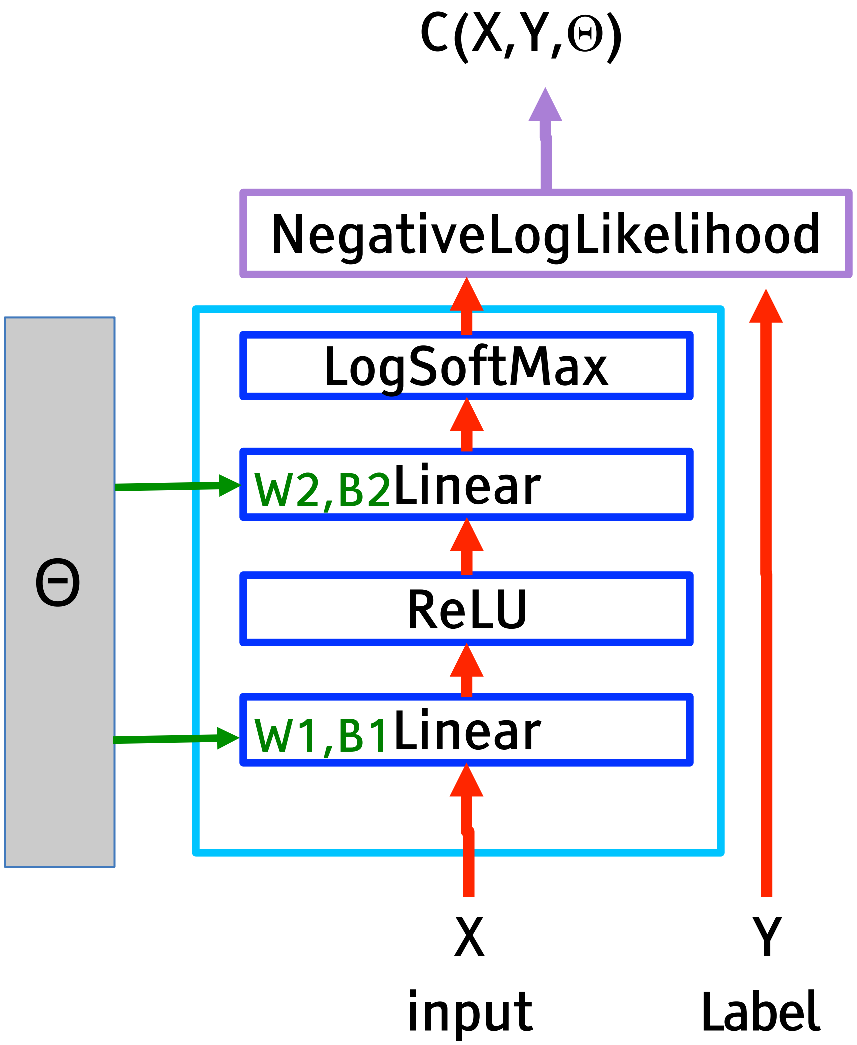
- A practical Application of Chain Rule

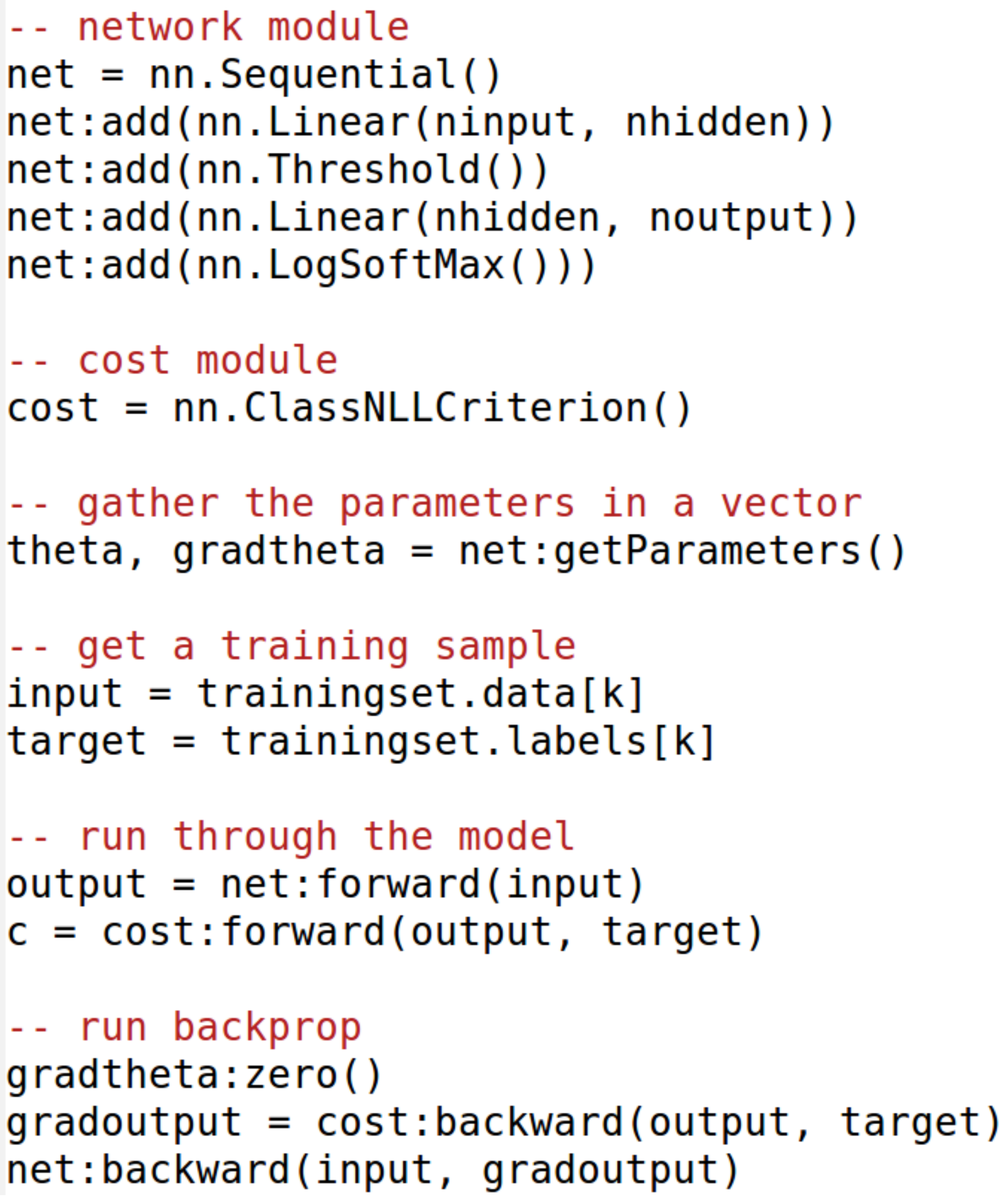
- Backprop for the state gradients:
- $dC/dX_{i-1} = dC/dX_i \;.\; dX_i/dX_{i-1}$
- $dC/dX_{i-1} = dC/dX_i \;.\; dF_i(X_{i-1},W_i)/dX_{i-1}$

- Backprop for the weight gradients:
- $dC/dW_i = dC/dX_i \;.\; dX_i/dW_i$
- $dC/dW_i = dC/dX_i \;.\; dF_i(X_{i-1},W_i)/dW_i$

Running Backprop

- Torch7 example
- Gradtheta contains the gradient





```
-- network module
net = nn.Sequential()
net:add(nn.Linear(ninput, nhidden))
net:add(nn.Threshold())
net:add(nn.Linear(nhidden, noutput))
net:add(nn.LogSoftMax()))

-- cost module
cost = nn.ClassNLLCriterion()

-- gather the parameters in a vector
theta, gradtheta = net:getParameters()

-- get a training sample
input = trainingset.data[k]
target = trainingset.labels[k]

-- run through the model
output = net:forward(input)
c = cost:forward(output, target)

-- run backprop
gradtheta:zero()
gradoutput = cost:backward(output, target)
net:backward(input, gradoutput)
```

Module Classes

Linear

- $Y = W.X$; $dC/dX = W^T . dC/dY$; $dC/dW = dC/dY . (dC/dX)^T$

ReLU

- $y = ReLU(x)$; if $(x<0)$ $dC/dx = 0$ else $dC/dx = dC/dy$

Duplicate

- $Y1 = X, Y2 = X$; $dC/dX = dC/dY1 + dC/dY2$

Add

- $Y = X1 + X2$; $dC/dX1 = dC/dY$; $dC/dX2 = dC/dY$

Max

- $y = max(x1,x2)$; if $(x1>x2)$ $dC/dx1 = dC/dy$ else $dC/dx1=0$

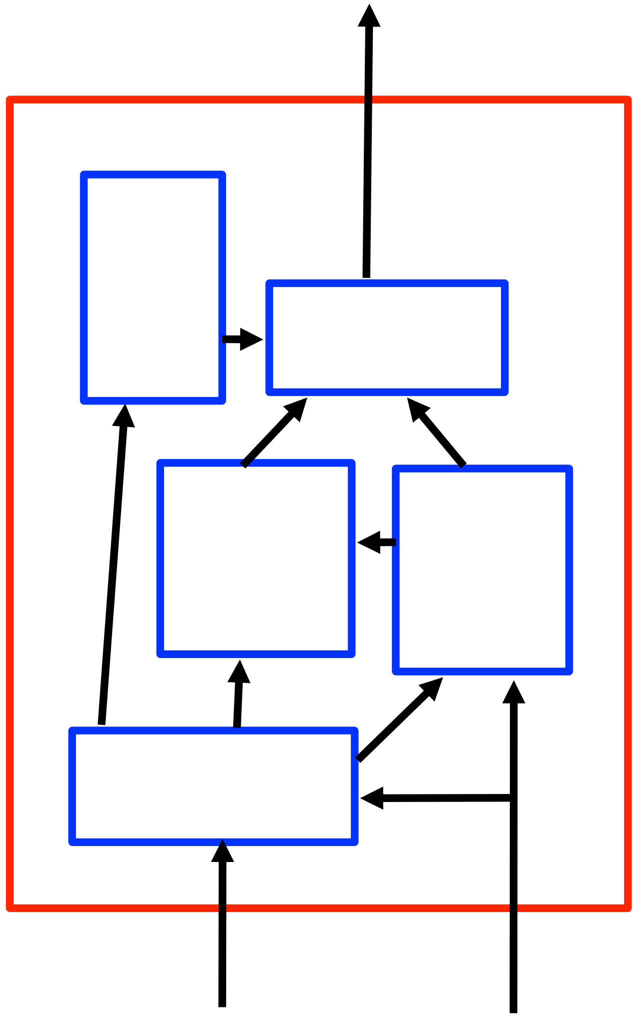
LogSoftMax

- $Yi = Xi - \log [\sum_j \exp(Xj)]$; .....

Module Classes

- Many more basic module classes
- Cost functions:
  - Squared error
  - Hinge loss
  - Ranking loss
- Non-linearities and operators
  - ReLU, “leaky” ReLU, abs,….
  - Tanh, logistic
  - Just about any simple function (log, exp, add, mul,….)
- Specialized modules
  - Multiple convolutions (1D, 2D, 3D)
  - Pooling/subsampling: max, average, Lp, log(sum(exp())), maxout
  - Long Short-Term Memory, attention, 3-way multiplicative interactions.
  - Switches
  - Normalizations: batch norm, contrast norm, feature norm...
  - inception

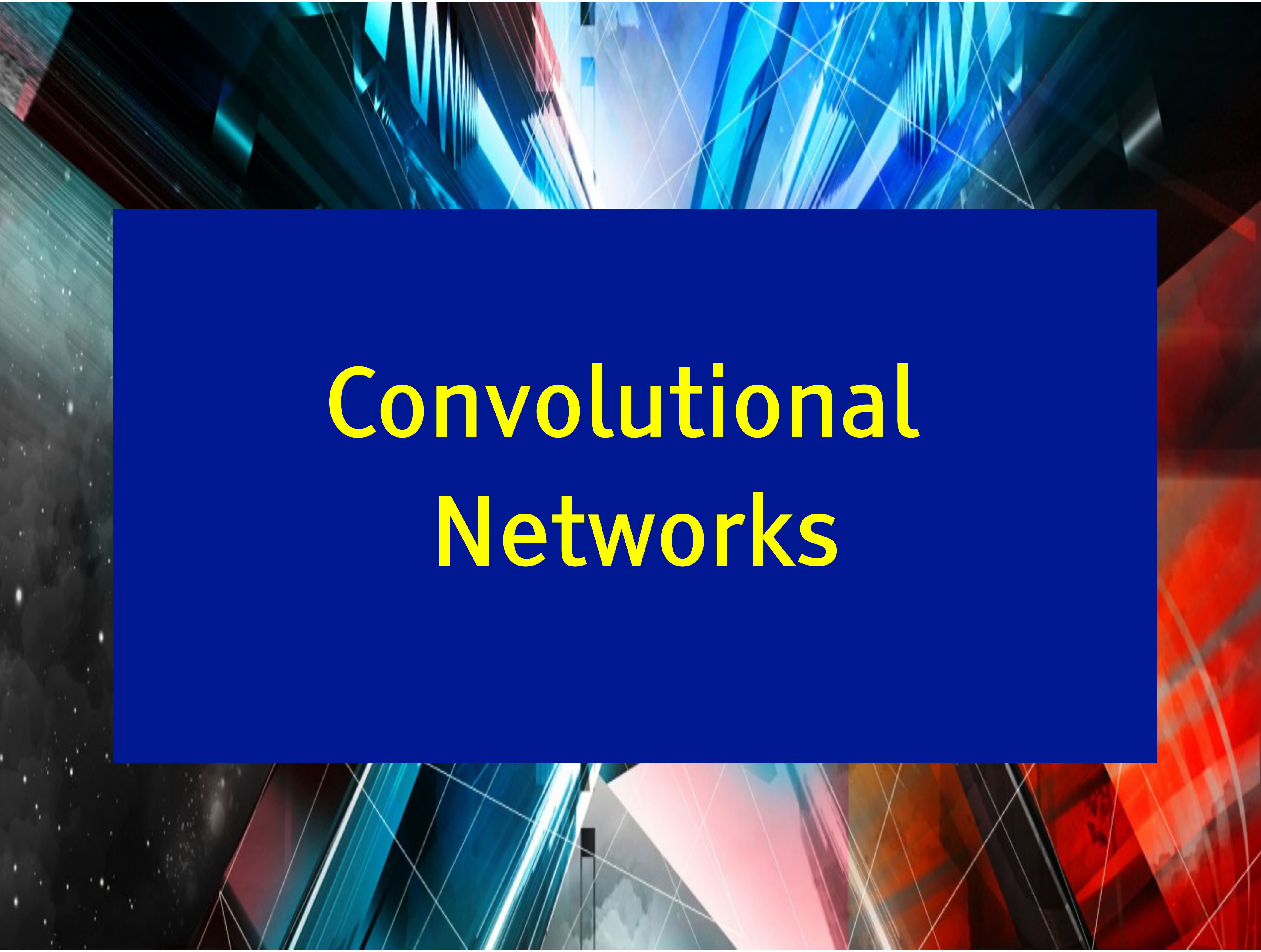
Any Architecture works



- **Any connection graph is permissible**
  - Directed acyclic graphs (DAG)
  - Networks with loops must be "unfolded in time".
- **Any module is permissible**
  - As long as it is continuous and differentiable almost everywhere with respect to the parameters, and with respect to non-terminal inputs.
- **Most frameworks provide automatic differentiation**
  - Theano, Torch7+autograd,…
  - Programs are turned into computation DAGs and automatically differentiated.

Backprop in Practice

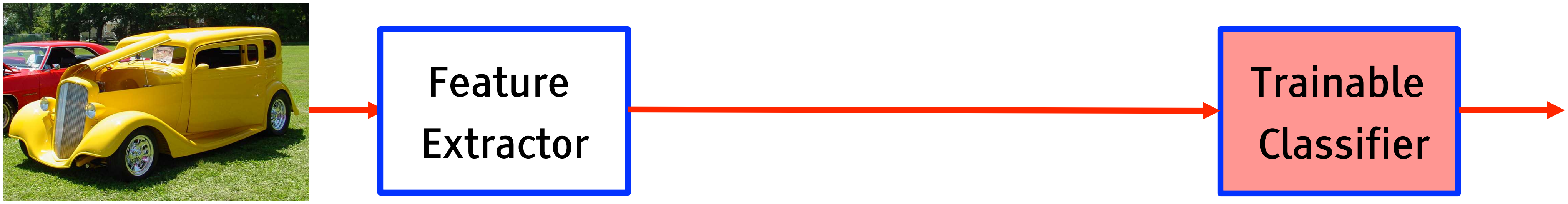
- Use ReLU non-linearities
- Use cross-entropy loss for classification
- Use Stochastic Gradient Descent on minibatches
- Shuffle the training samples (← very important)
- Normalize the input variables (zero mean, unit variance)
- Schedule to decrease the learning rate
- Use a bit of L1 or L2 regularization on the weights (or a combination)
    - But it's best to turn it on after a couple of epochs
- Use “dropout” for regularization
- Lots more in [LeCun et al. “Efficient Backprop” 1998]
- Lots, lots more in “Neural Networks, Tricks of the Trade” (2012 edition) edited by G. Montavon, G. B. Orr, and K-R Müller (Springer)
- More recent: Deep Learning (MIT Press book in preparation)



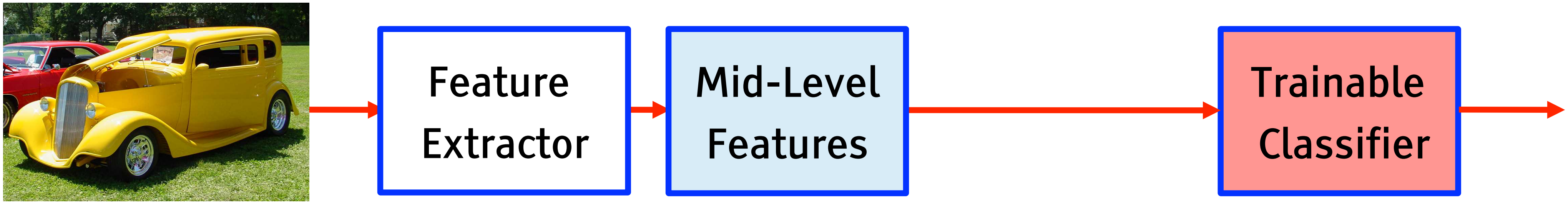
Convolutional Networks

Deep Learning = Training Multistage Machines

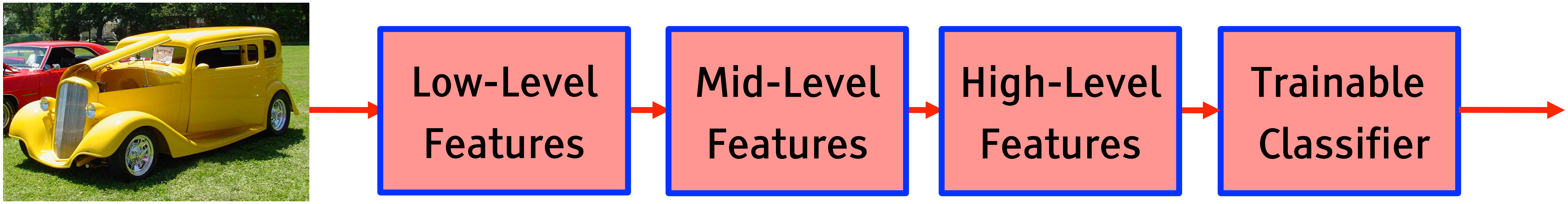
- Traditional Pattern Recognition: Fixed/Handcrafted Feature Extractor



- Mainstream Pattern Recognition 9until recently)

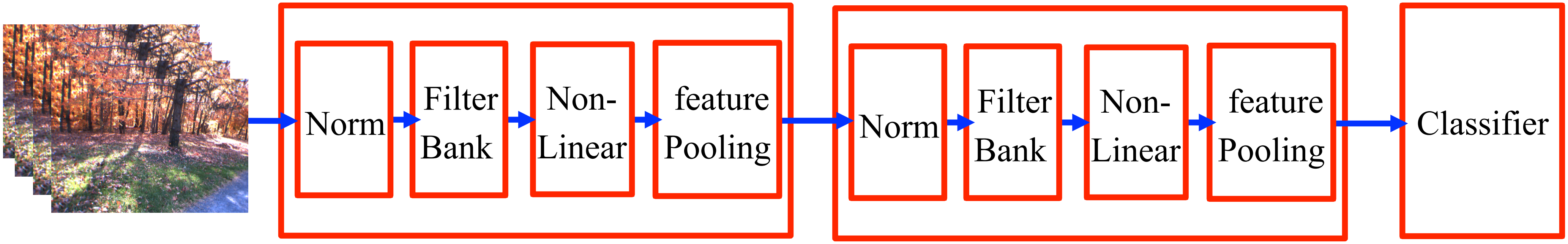


- Deep Learning: Multiple stages/layers trained end to end

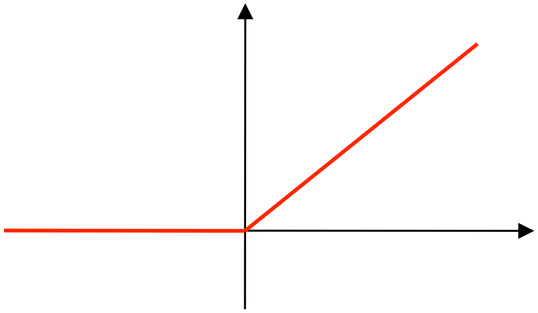


Overall Architecture: multiple stages of Normalization → Filter Bank → Non-Linearity → Pooling

Y LeCun

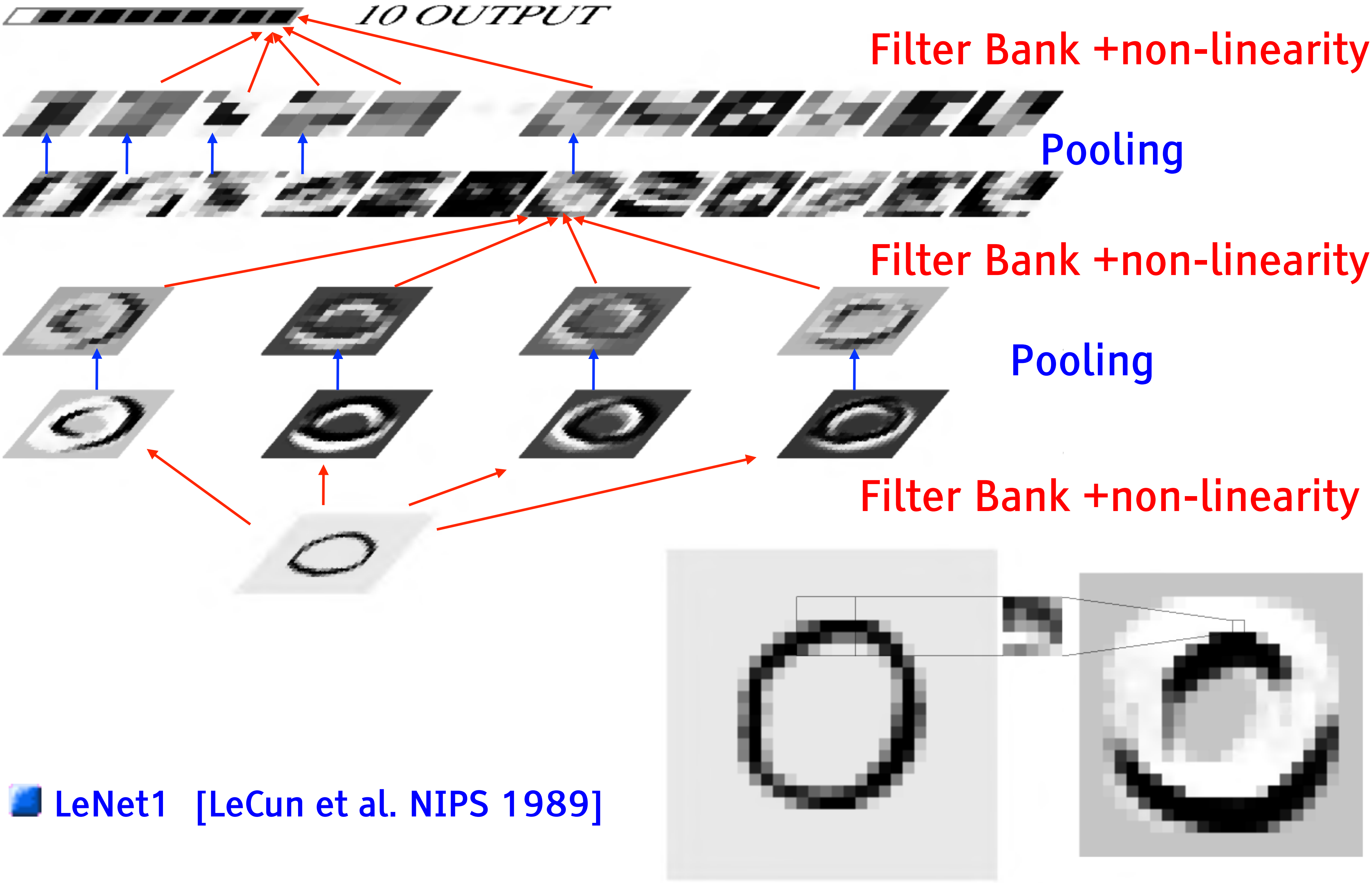


- **Normalization: variation on whitening (optional)**
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization
- **Filter Bank: dimension expansion, projection on overcomplete basis**
- **Non-Linearity: sparsification, saturation, lateral inhibition....**
  - Rectification (ReLU), Component-wise shrinkage, tanh,..



- **Pooling: aggregation over space or feature type**
  - Max, Lp norm, log prob.

ConvNet Architecture



Filter Bank +non-linearity

Pooling

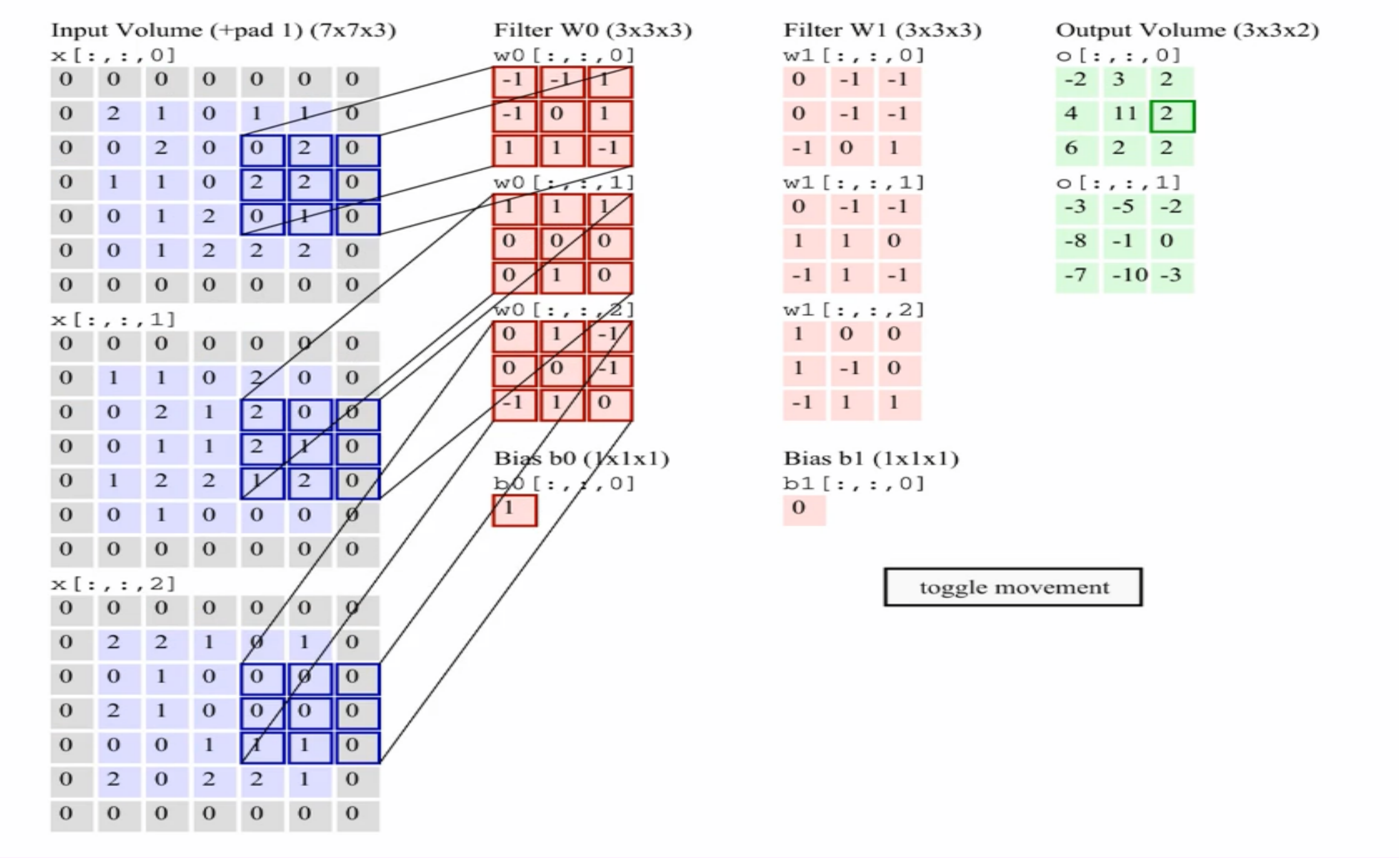
Filter Bank +non-linearity

Pooling

Filter Bank +non-linearity

- LeNet1 [LeCun et al. NIPS 1989]

Multiple Convolutions



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	1	0	1	1	0
0	0	2	0	0	2	0
0	1	1	0	2	2	0
0	0	1	2	0	1	0
0	0	1	2	2	2	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	1	1	0	2	0	0
0	0	2	1	2	0	0
0	0	1	1	2	1	0
0	1	2	2	1	2	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	2	1	0	1	0
0	0	1	0	0	0	0
0	2	1	0	0	0	0
0	0	0	1	1	1	0
0	2	0	2	2	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	-1	1
-1	0	1
1	1	-1

w0[:,:,1]

1	1	1
0	0	0
0	1	0

w0[:,:,2]

0	1	-1
0	0	-1
-1	1	0

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	-1	-1
0	-1	-1
-1	0	1

w1[:,:,1]

0	-1	-1
1	1	0
-1	1	-1

w1[:,:,2]

1	0	0
1	-1	0
-1	1	1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

-2	3	2
4	11	2
6	2	2

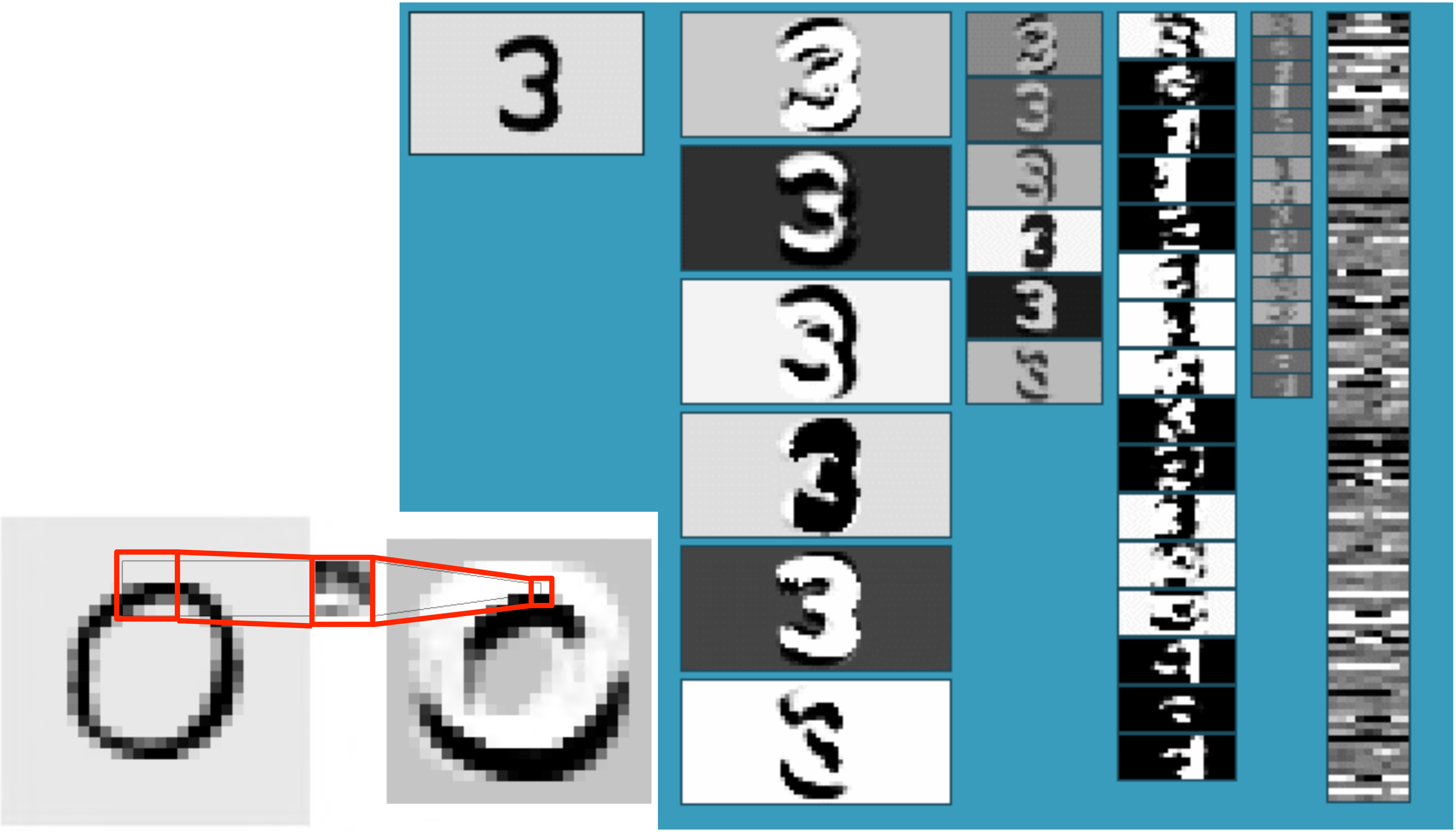
o[:,:,1]

-3	-5	-2
-8	-1	0
-7	-10	-3

Animation: Andrej Karpathy http://cs231n.github.io/convolutional-networks/

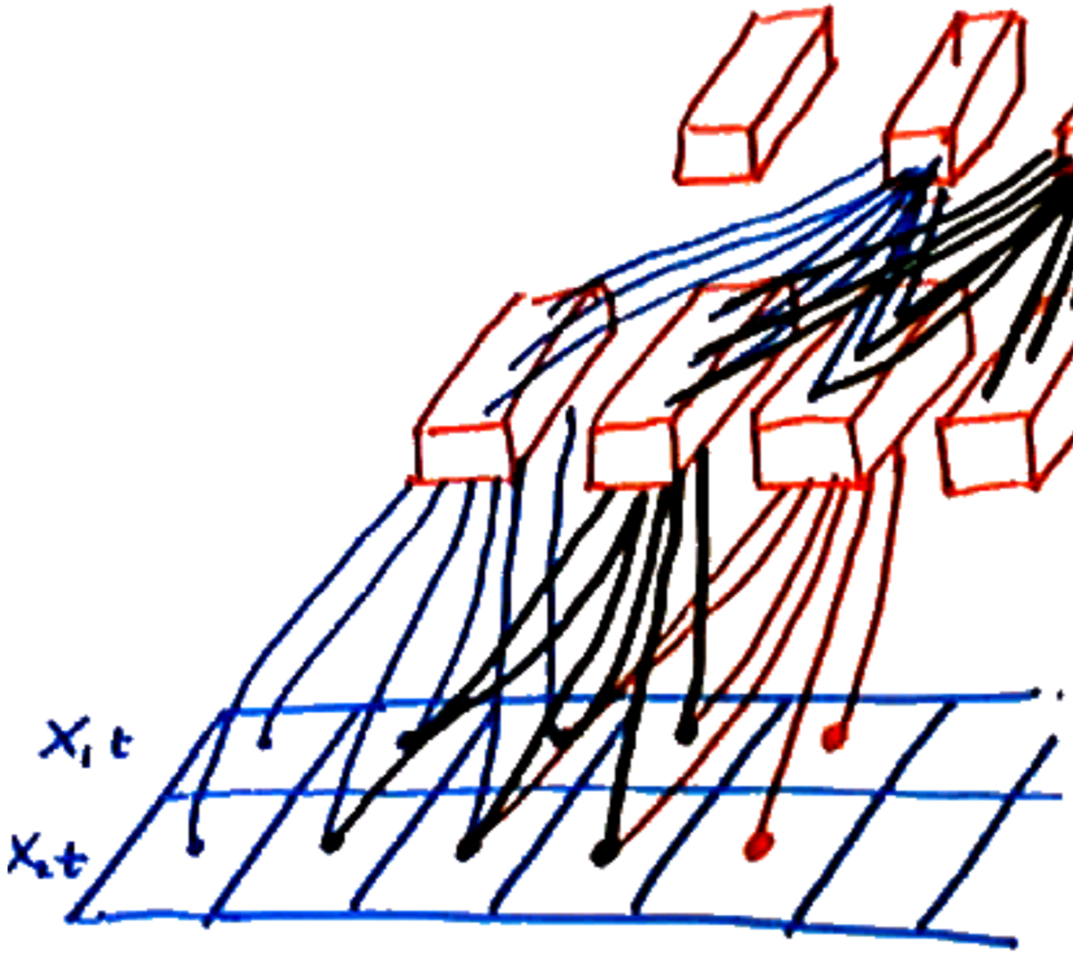
Convolutional Networks (vintage 1990)

- filters → tanh → average-tanh → filters → tanh → average-tanh → filters → tanh



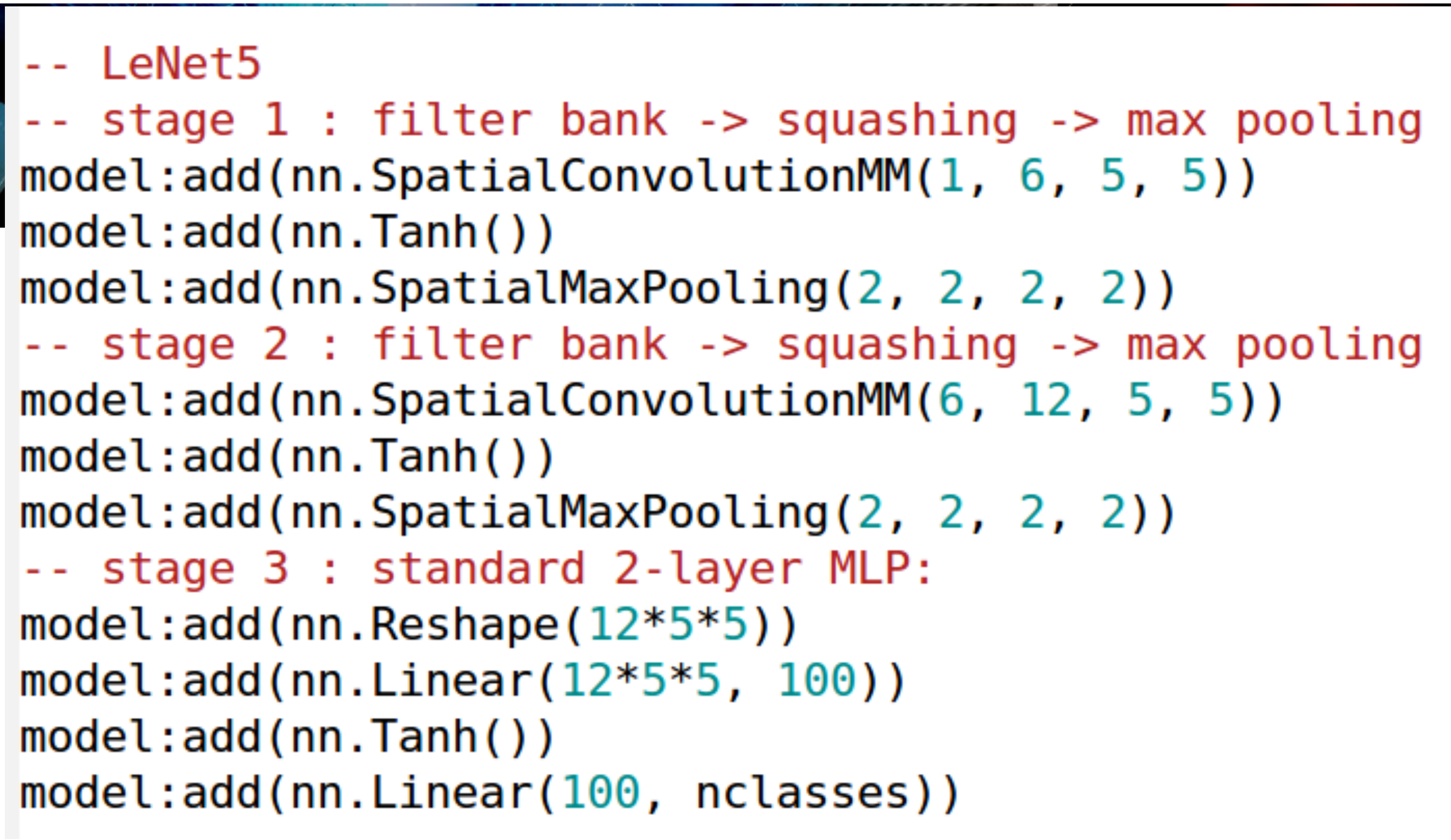
Example: 1D (Temporal) convolutional net

- 1D (Temporal) ConvNet, aka Timed-Delay Neural Nets
- Groups of units are replicated at each time step.
- Replicas have identical (shared) weights.

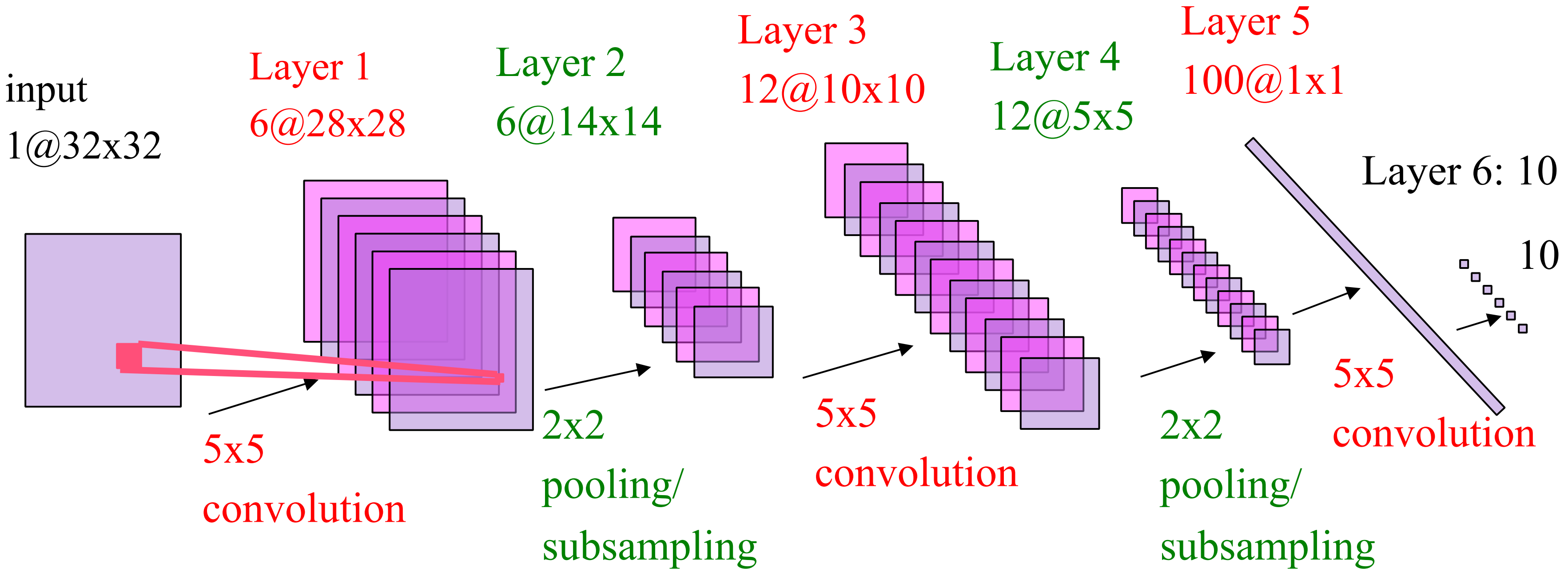


LeNet5

- Simple ConvNet
- for MNIST
- [LeCun 1998]

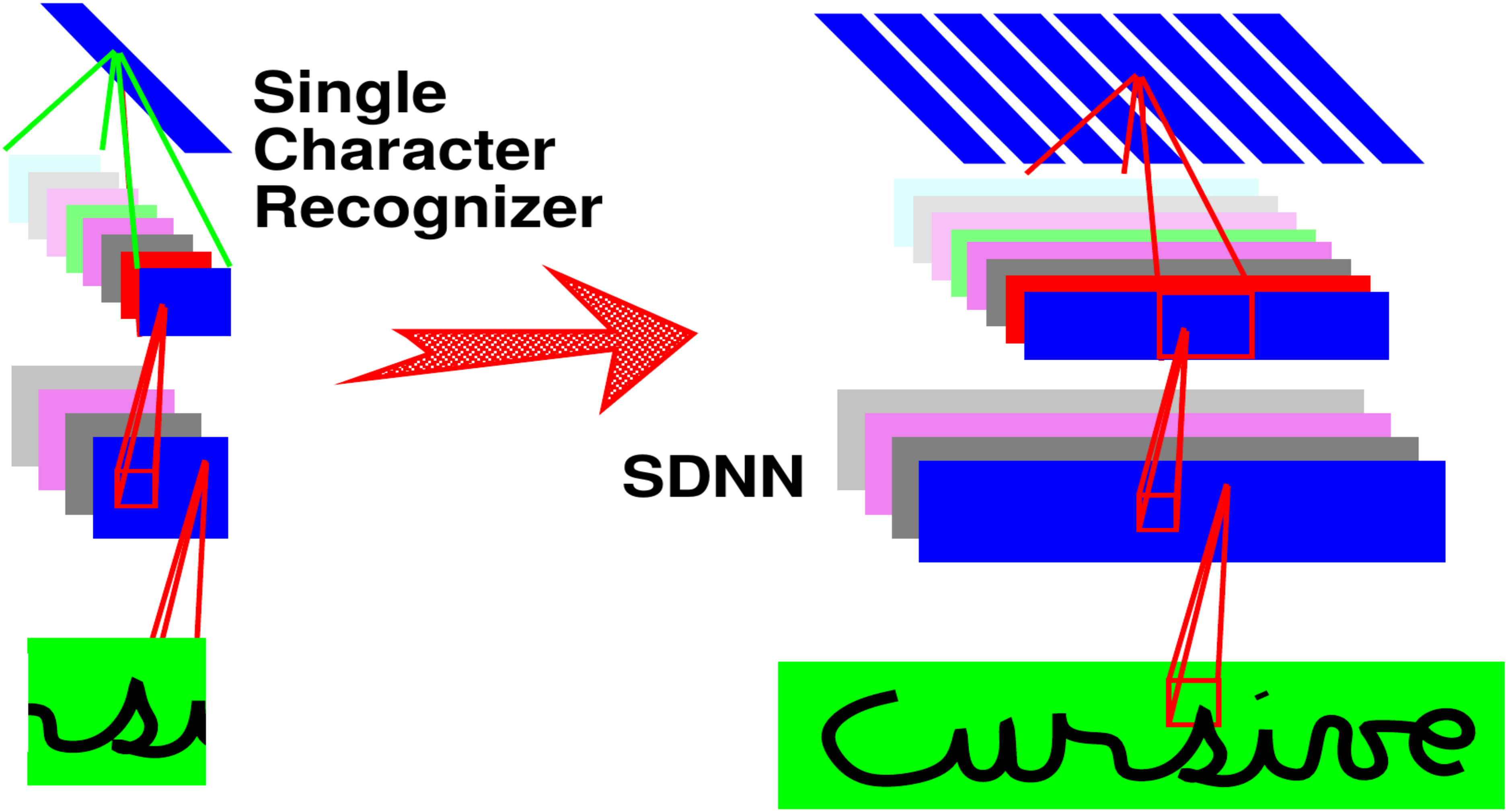


```
-- LeNet5
-- stage 1 : filter bank -> squashing -> max pooling
model:add(nn.SpatialConvolutionMM(1, 6, 5, 5))
model:add(nn.Tanh())
model:add(nn.SpatialMaxPooling(2, 2, 2, 2))
-- stage 2 : filter bank -> squashing -> max pooling
model:add(nn.SpatialConvolutionMM(6, 12, 5, 5))
model:add(nn.Tanh())
model:add(nn.SpatialMaxPooling(2, 2, 2, 2))
-- stage 3 : standard 2-layer MLP:
model:add(nn.Reshape(12*5*5))
model:add(nn.Linear(12*5*5, 100))
model:add(nn.Tanh())
model:add(nn.Linear(100, nclasses))
```



Applying a ConvNet with a Sliding Window

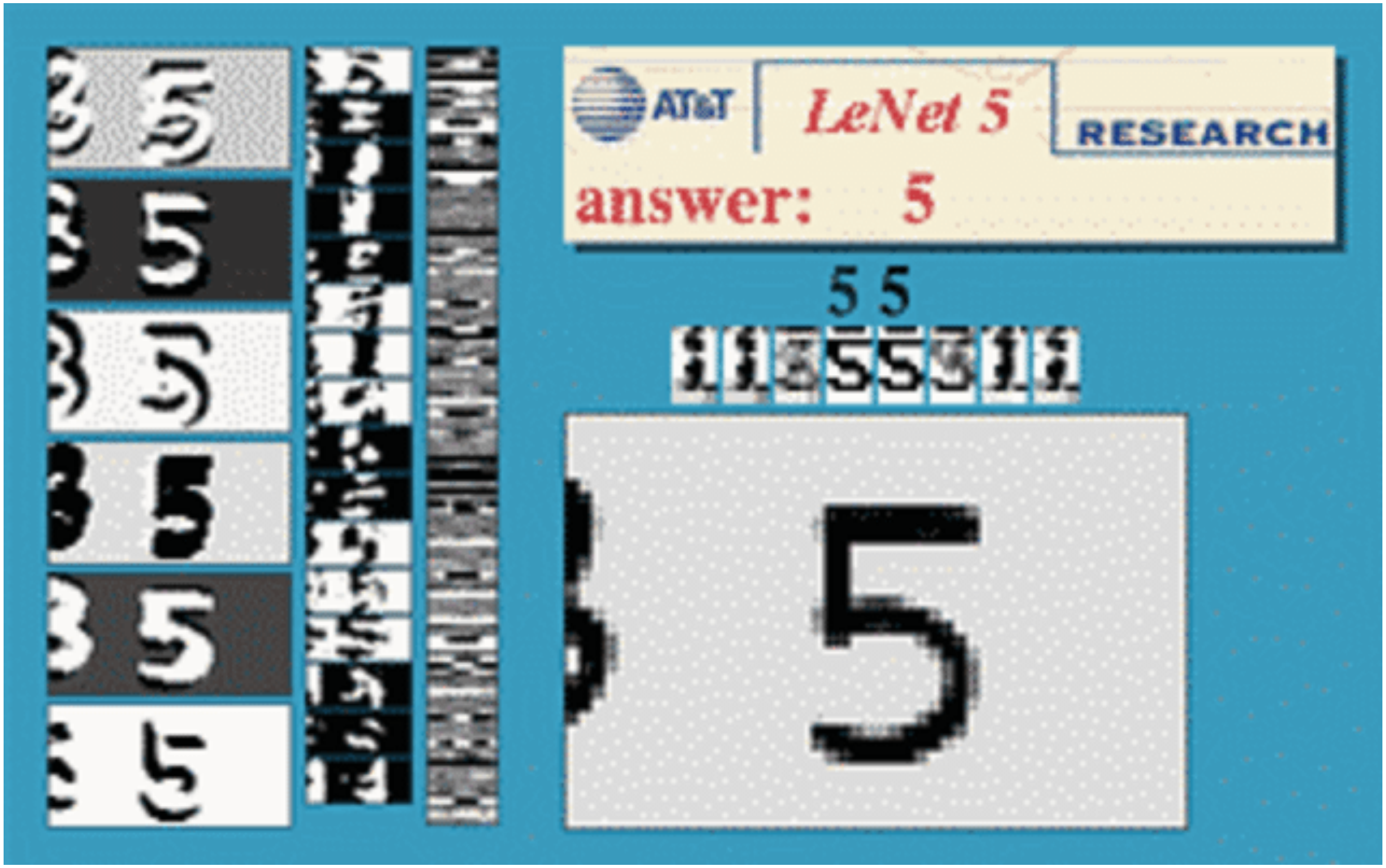
- Every layer is a convolution
- Sometimes called “fully convolutional nets”
- There is no such thing as a “fully connected layer”



Sliding Window ConvNet + Weighted FSM (Fixed Post-Proc)

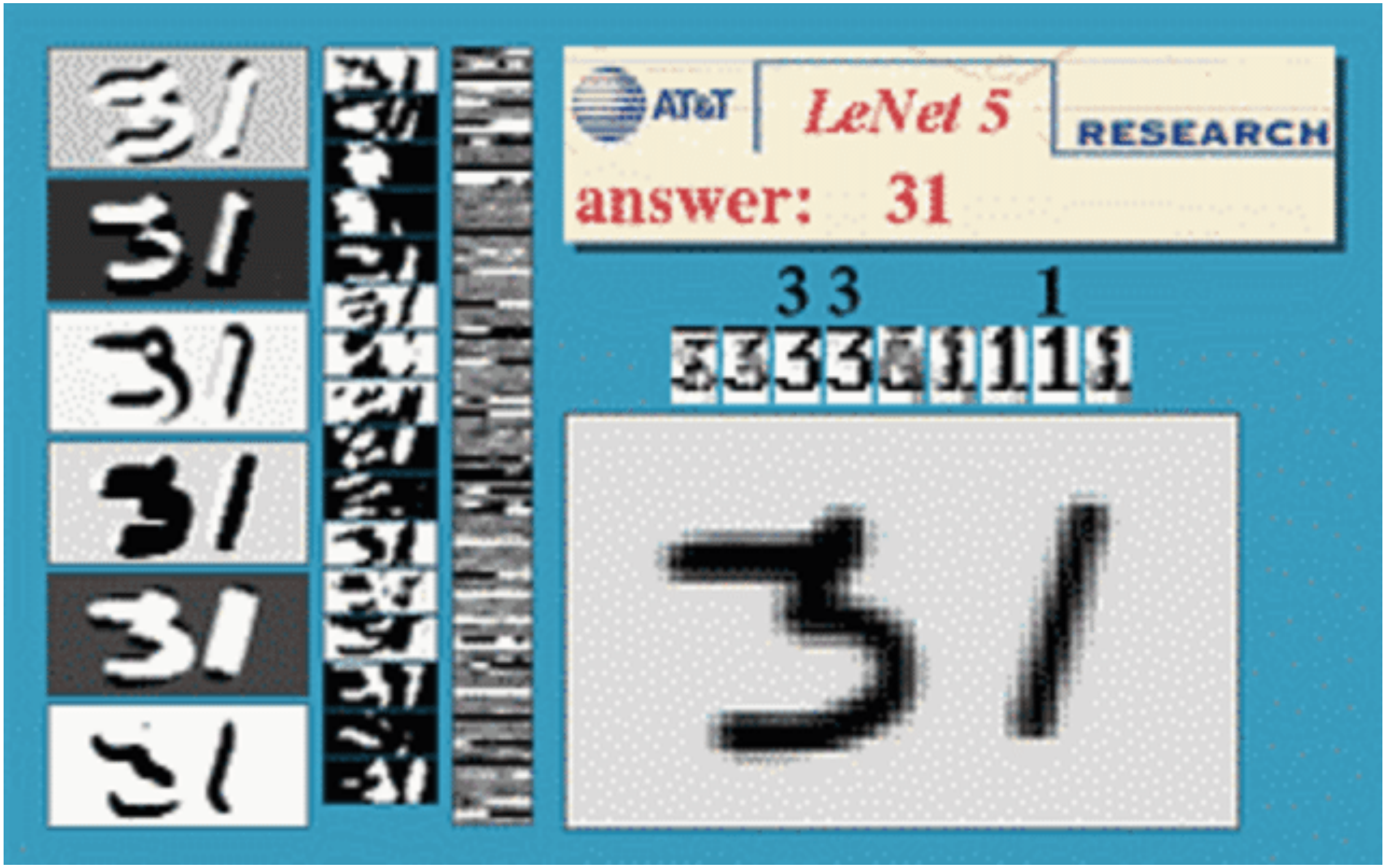
Y LeCun

[Matan, Burges, LeCun, Denker NIPS 1991] [LeCun, Bottou, Bengio, Haffner, Proc IEEE 1998]



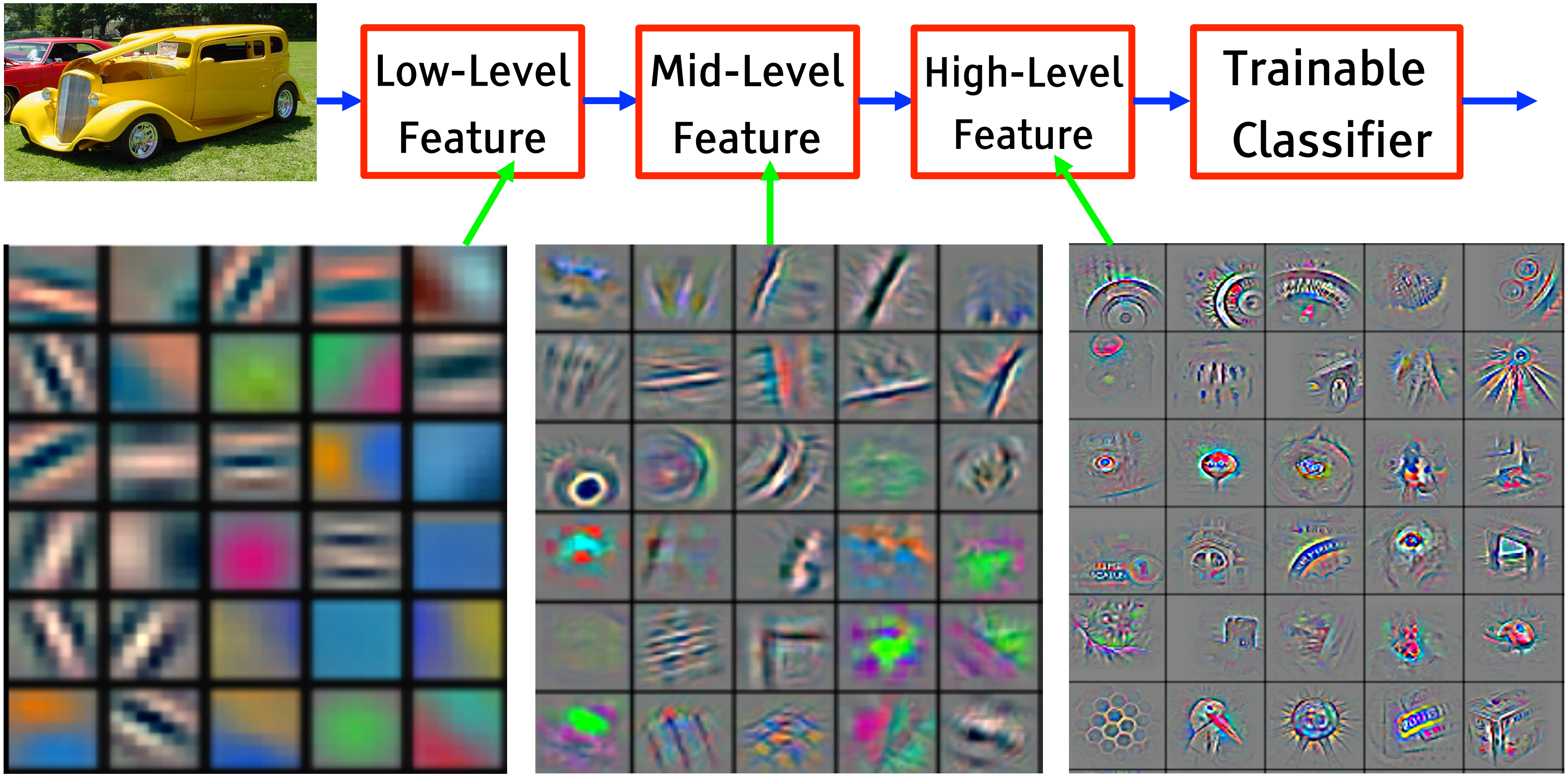
Sliding Window ConvNet + Weighted FSM

Y LeCun



Why Multiple Layers? The World is Compositional

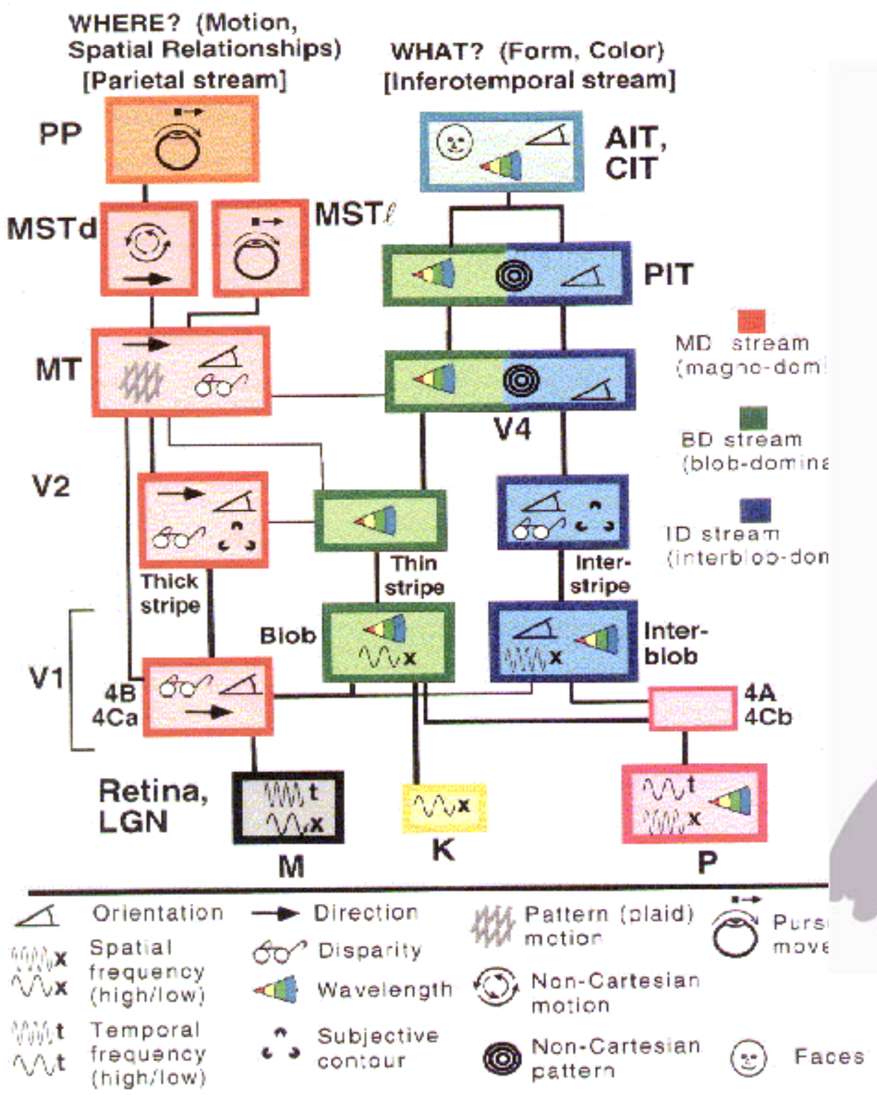
- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition: Pixel → edge → texton → motif → part → object
- Text: Character → word → word group → clause → sentence → story
- Speech: Sample → spectral band → sound → … → phone → phoneme → word

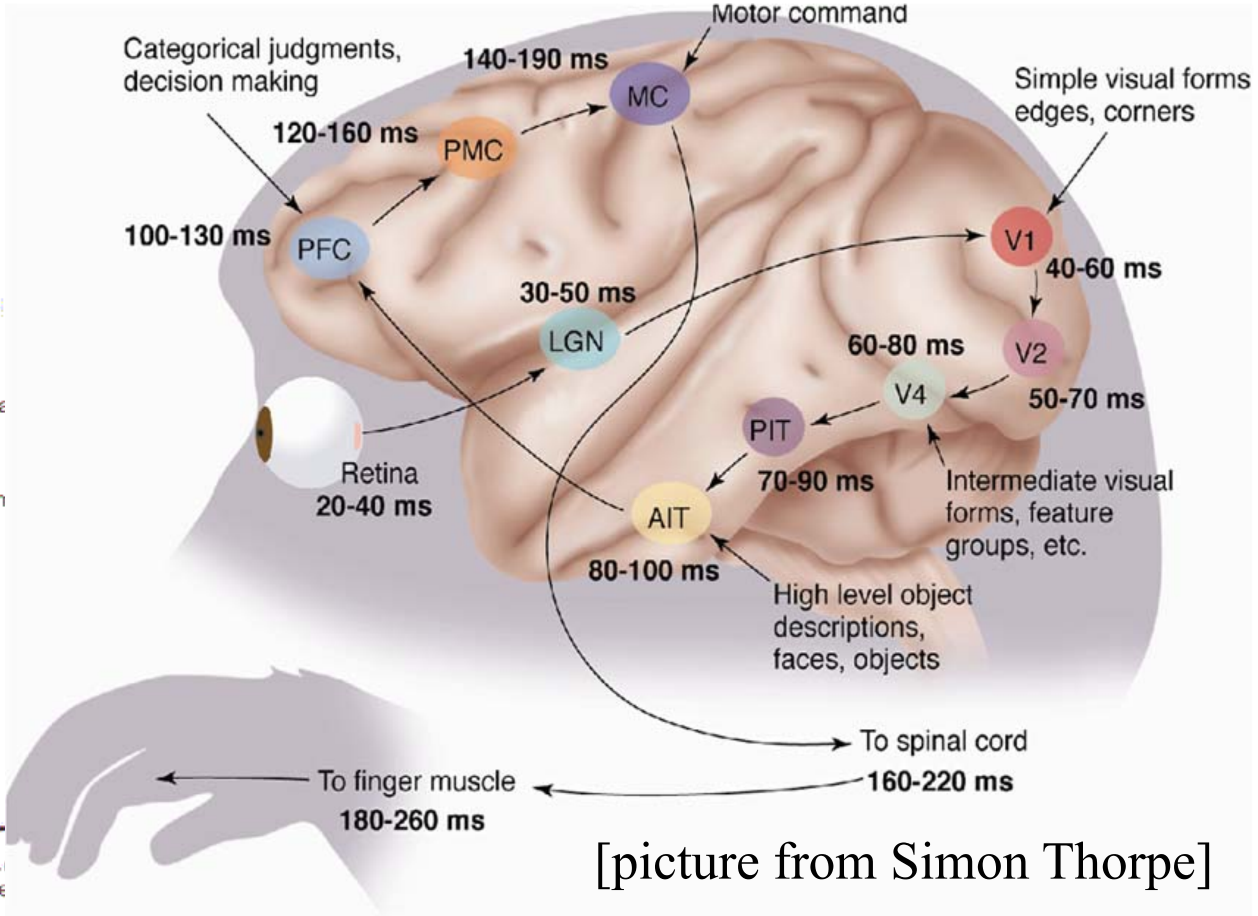


Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier →

Yes, ConvNets are somewhat inspired by the Visual Cortex

- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....





[picture from Simon Thorpe]

[Gallant & Van Essen]

What are ConvNets Good For

- Signals that comes to you in the form of (multidimensional) arrays.
- Signals that have strong local correlations
- Signals where features can appear anywhere
- Signals in which objects are invariant to translations and distortions.

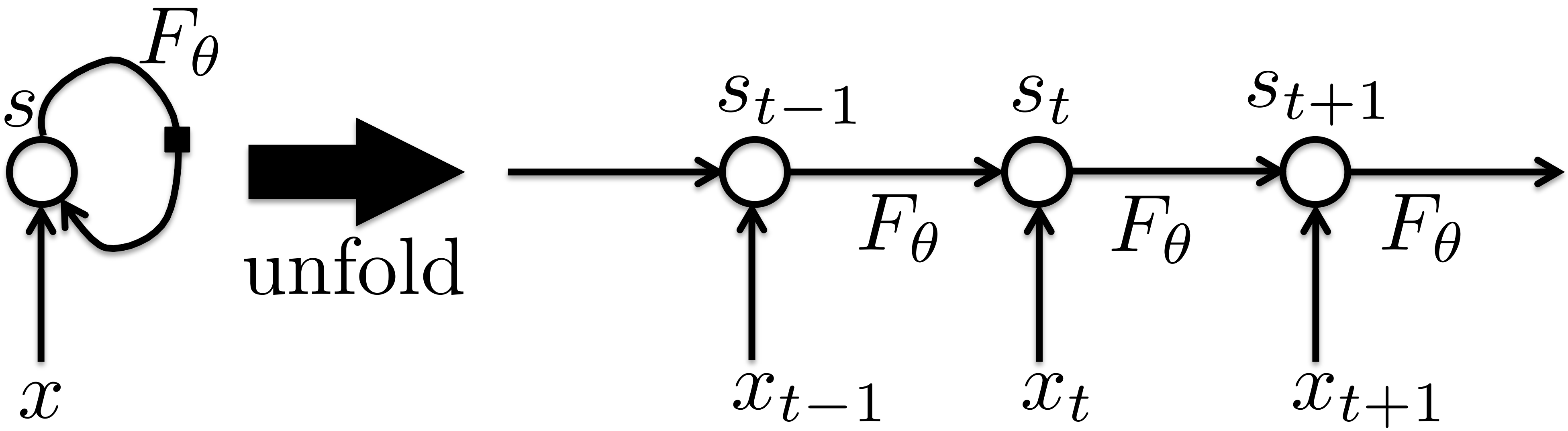
- 1D ConvNets: sequential signals, text
  - Text Classification
  - Musical Genre Recognition
  - Acoustic Modeling for Speech Recognition
  - Time-Series Prediction
- 2D ConvNets: images, time-frequency representations (speech and audio)
  - Object detection, localization, recognition
- 3D ConvNets: video, volumetric images, tomography images
  - Video recognition / understanding
  - Biomedical image analysis
  - Hyperspectral image analysis

Recurrent Neural Networks

Recurrent Neural Networks

- Selectively summarize an input sequence in a fixed-size state vector via a recursive update

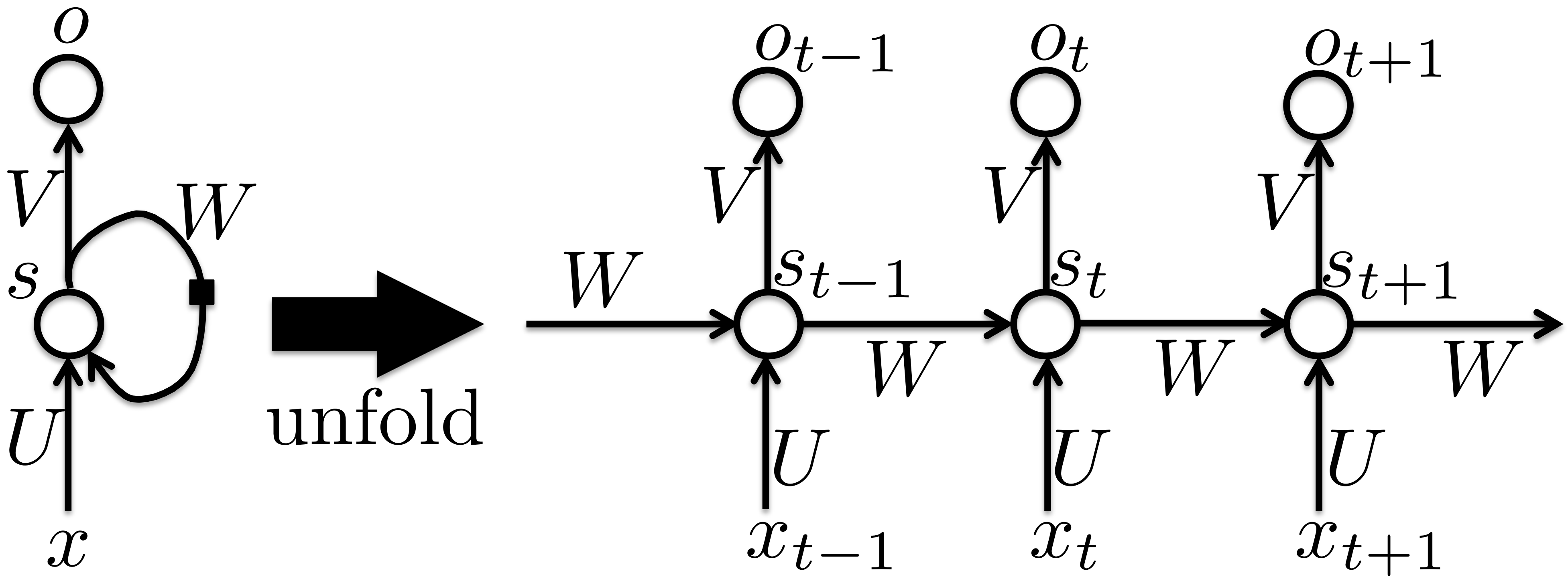
$$s_t = F_\theta(s_{t-1}, x_t)$$



$$s_t = G_t(x_t, x_{t-1}, x_{t-2}, \ldots, x_2, x_1)$$

Recurrent Neural Networks

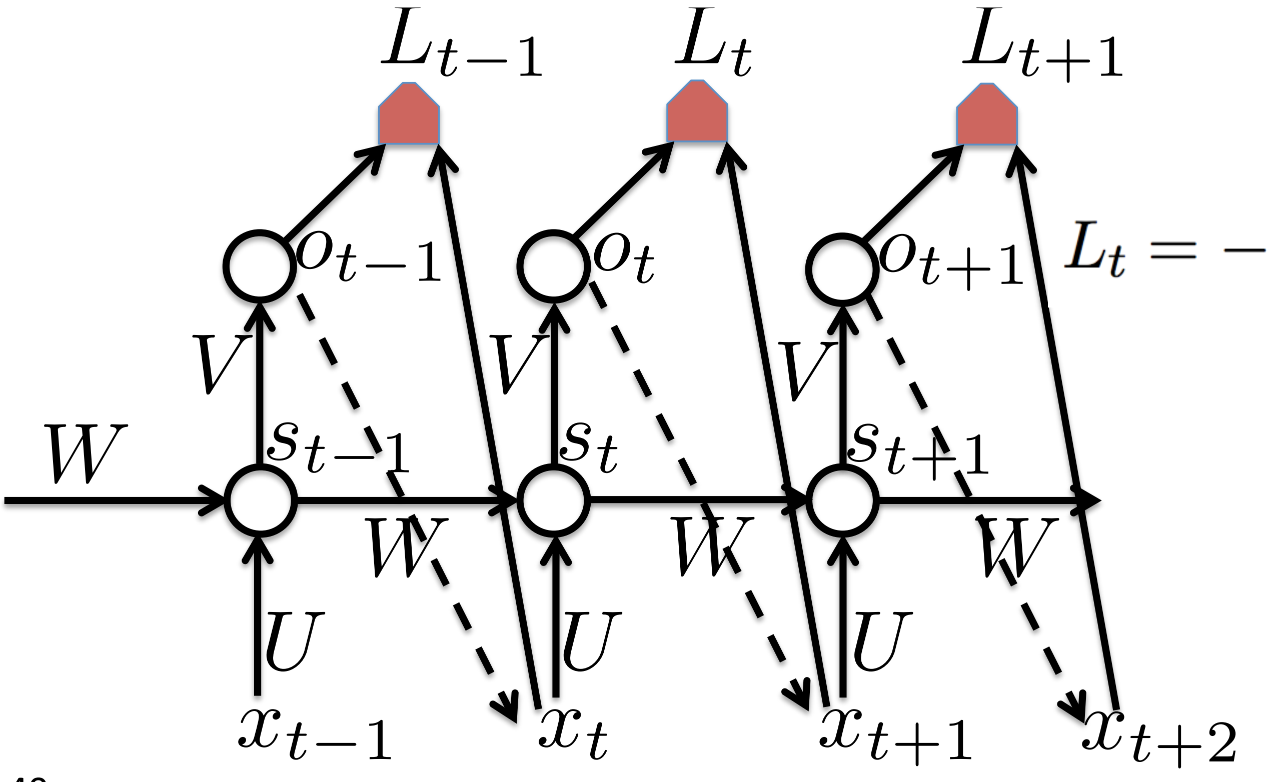
- Can produce an output at each time step: unfolding the graph tells us how to back-prop through time.



Generative RNNs

- An RNN can represent a fully-connected **directed generative model**: every variable predicted from all previous ones.

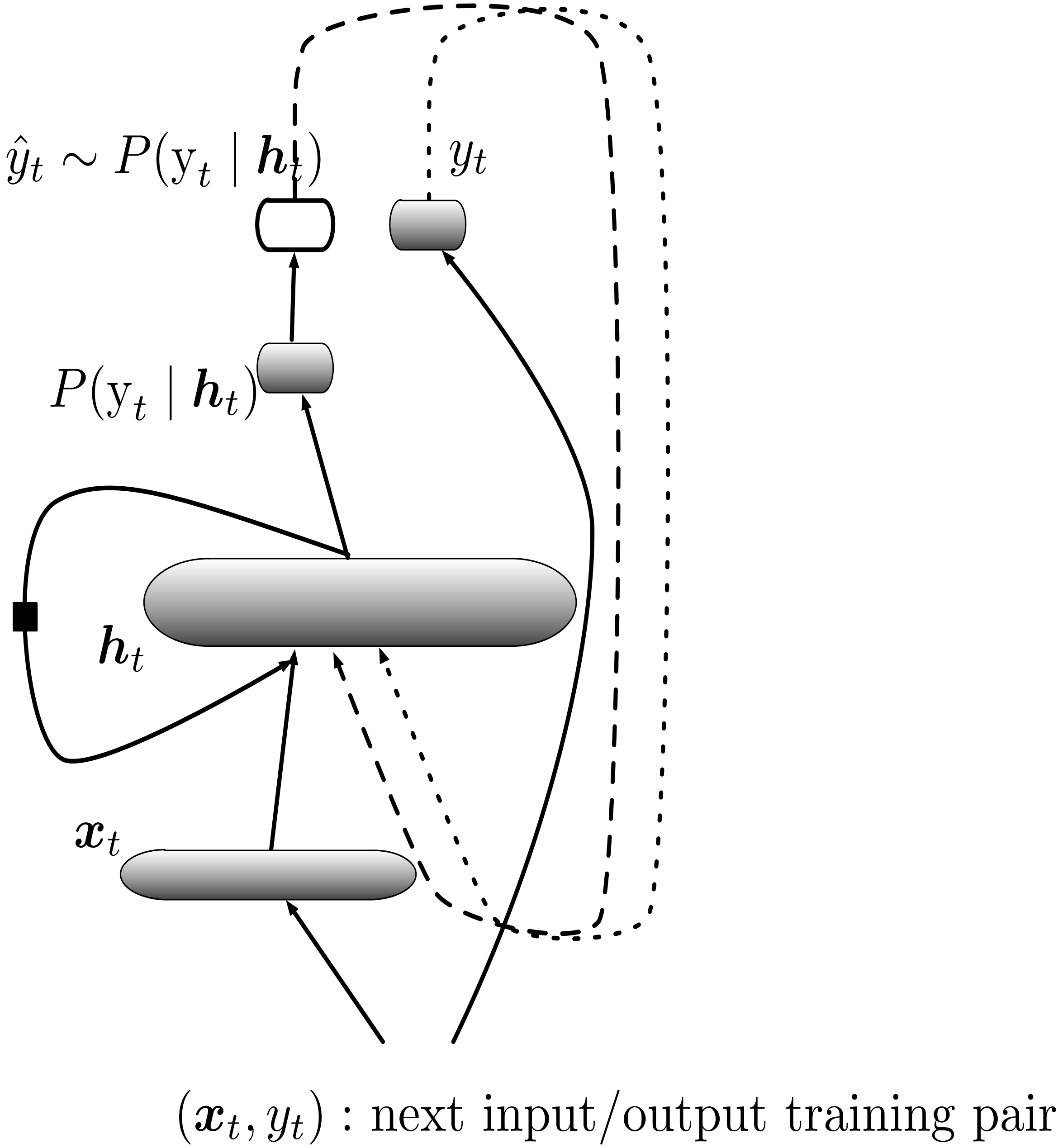
$$P(\mathbf{x}) = P(x_1, \ldots x_T) = \prod_{t=1}^{T} P(x_t | x_{t-1}, x_{t-2}, \ldots x_1)$$



$$L_t = -\log P(x_t | x_{t-1}, x_{t-2}, \ldots x_1)$$

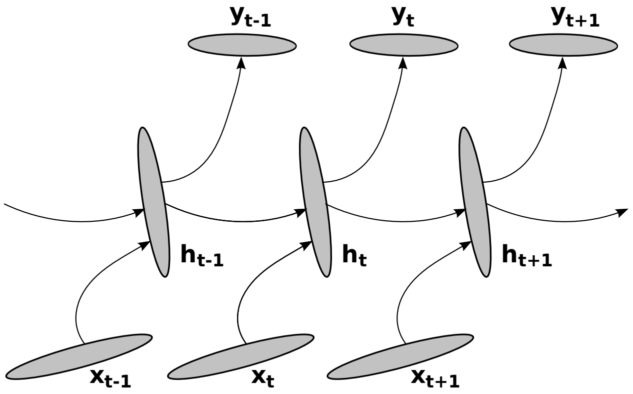
Maximum Likelihood = Teacher Forcing

- During training, past *y* in input is from training data
- At generation time, past *y* in input is generated
- Mismatch can cause ”compounding error”


$\hat{y}_t \sim P(\mathrm{y}_t \mid \boldsymbol{h}_t)$
y_t
$P(\mathrm{y}_t \mid \boldsymbol{h}_t)$
$\boldsymbol{h}_t$
$\boldsymbol{x}_t$
$(\boldsymbol{x}_t, y_t)$: next input/output training pair

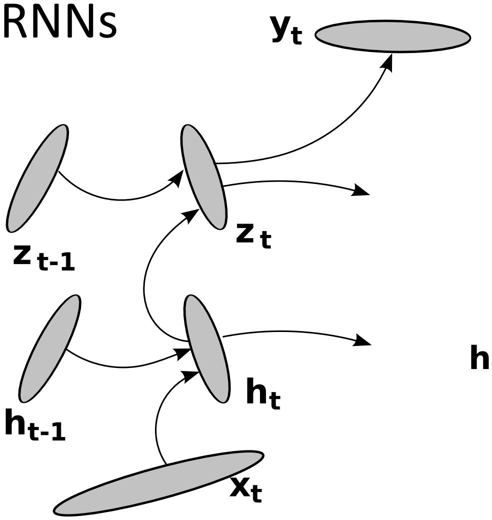
Increasing the Expressive Power of RNNs with more Depth

- ICLR 2014, *How to construct deep recurrent neural networks*

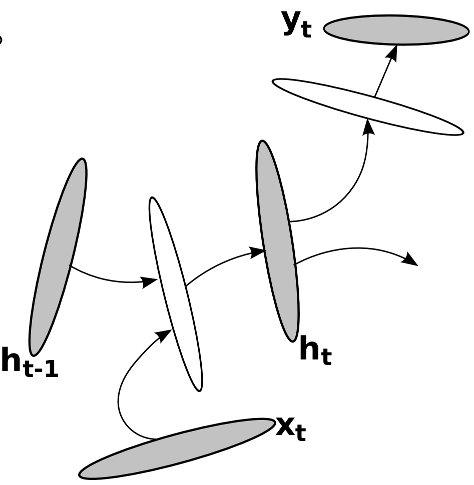


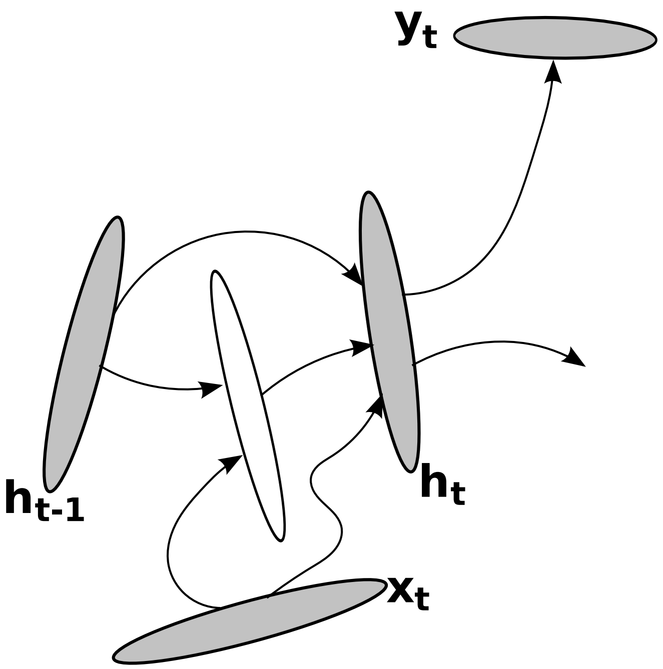
Ordinary RNNs

+ stacking



+ deep hid-to-out
+ deep hid-to-hid
+deep in-to-hid





+ skip connections for creating shorter paths

Long-Term Dependencies

- The RNN gradient is a product of Jacobian matrices, each associated with a step in the forward computation. To store information robustly in a finite-dimensional state, the dynamics must be contractive [Bengio et al 1994].

$$L = L(s_T(s_{T-1}(\ldots s_{t+1}(s_t, \ldots))))$$

$$\frac{\partial L}{\partial s_t} = \frac{\partial L}{\partial s_T} \frac{\partial s_T}{\partial s_{T-1}} \cdots \frac{\partial s_{t+1}}{\partial s_t}$$

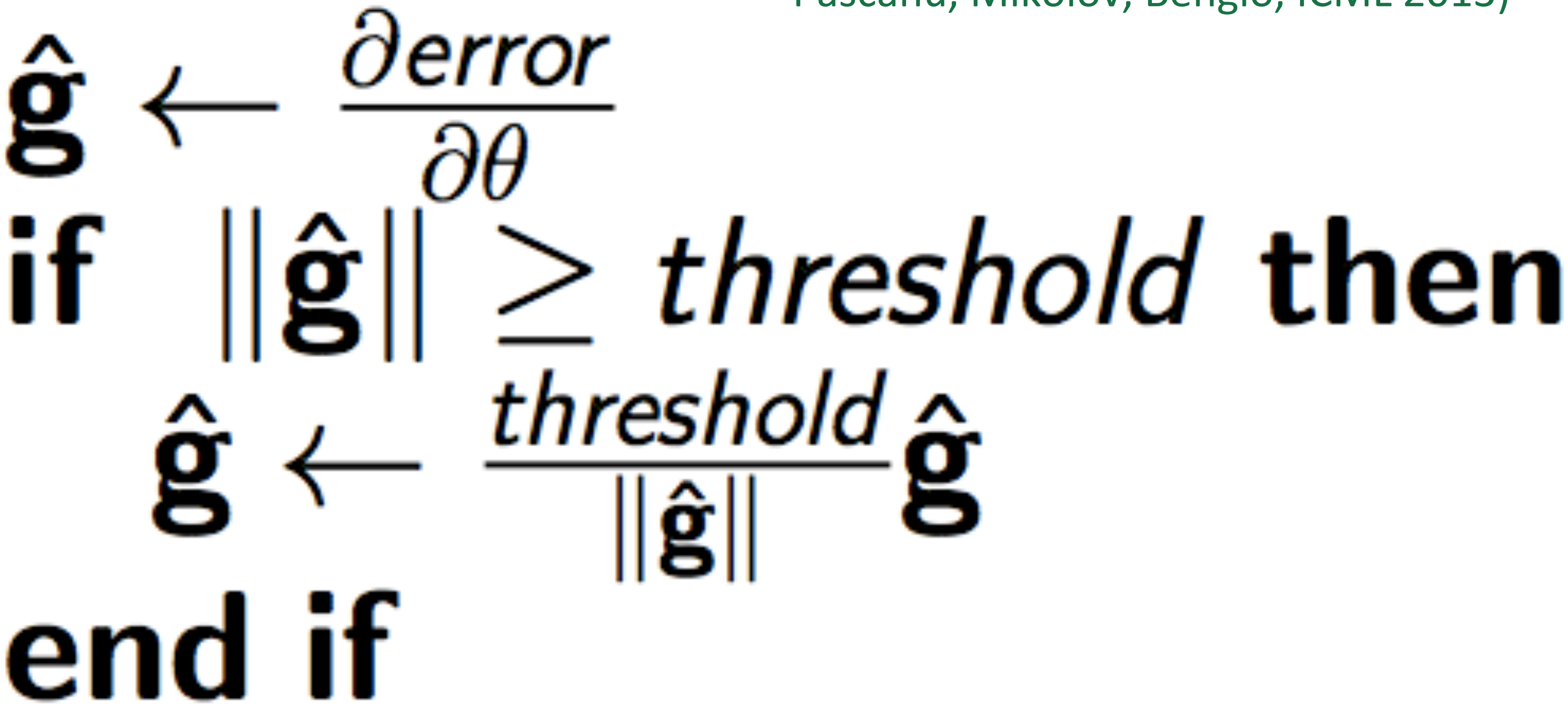
Storing bits robustly requires sing. values<1

- Problems:
  - sing. values of Jacobians > 1 → *gradients explode* → **Gradient clipping**
  - or sing. values < 1 → *gradients shrink & vanish*
  - or random → variance grows exponentially

(Hochreiter 1991)

Gradient Norm Clipping

(Mikolov thesis 2012;
Pascanu, Mikolov, Bengio, ICML 2013)

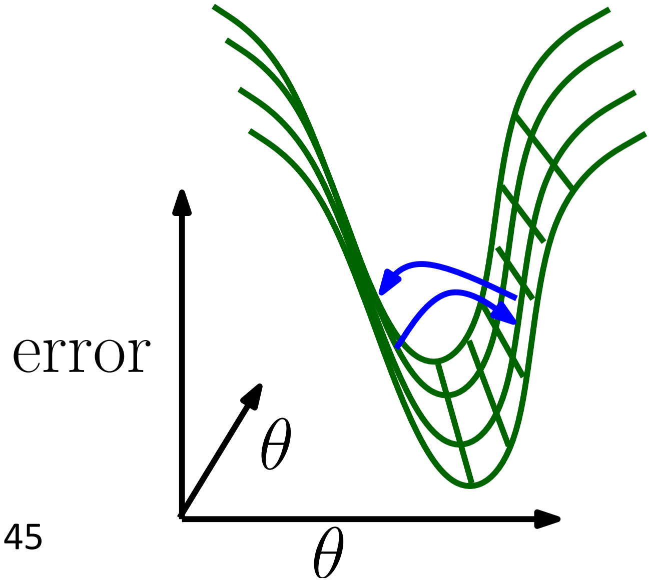


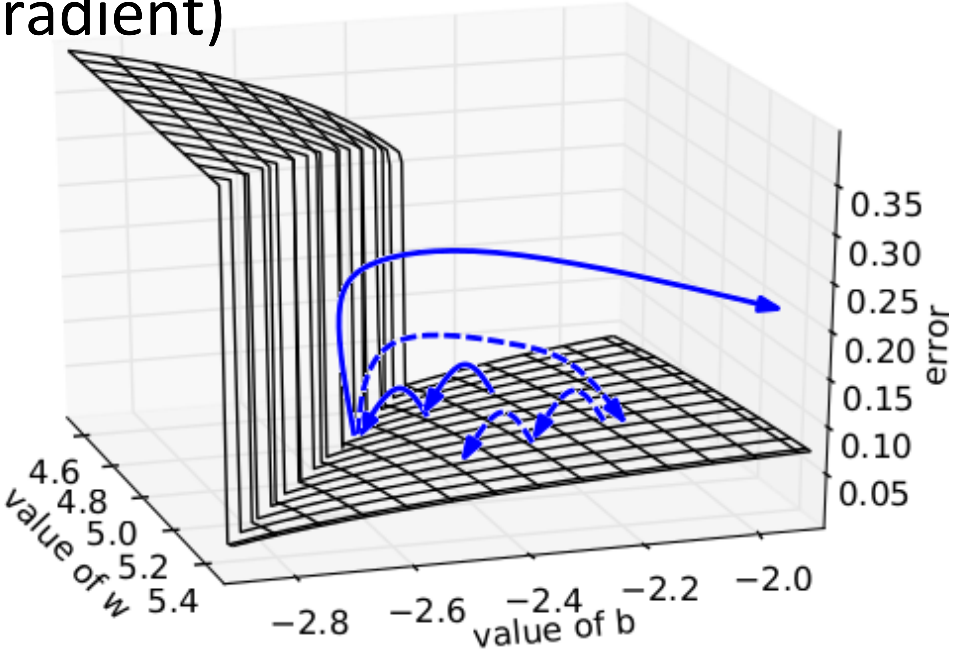
```
$\hat{\mathbf{g}} \leftarrow \frac{\partial error}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ then
    $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if
```

RNN Tricks

(Pascanu, Mikolov, Bengio, ICML 2013; Bengio, Boulanger & Pascanu, ICASSP 2013)

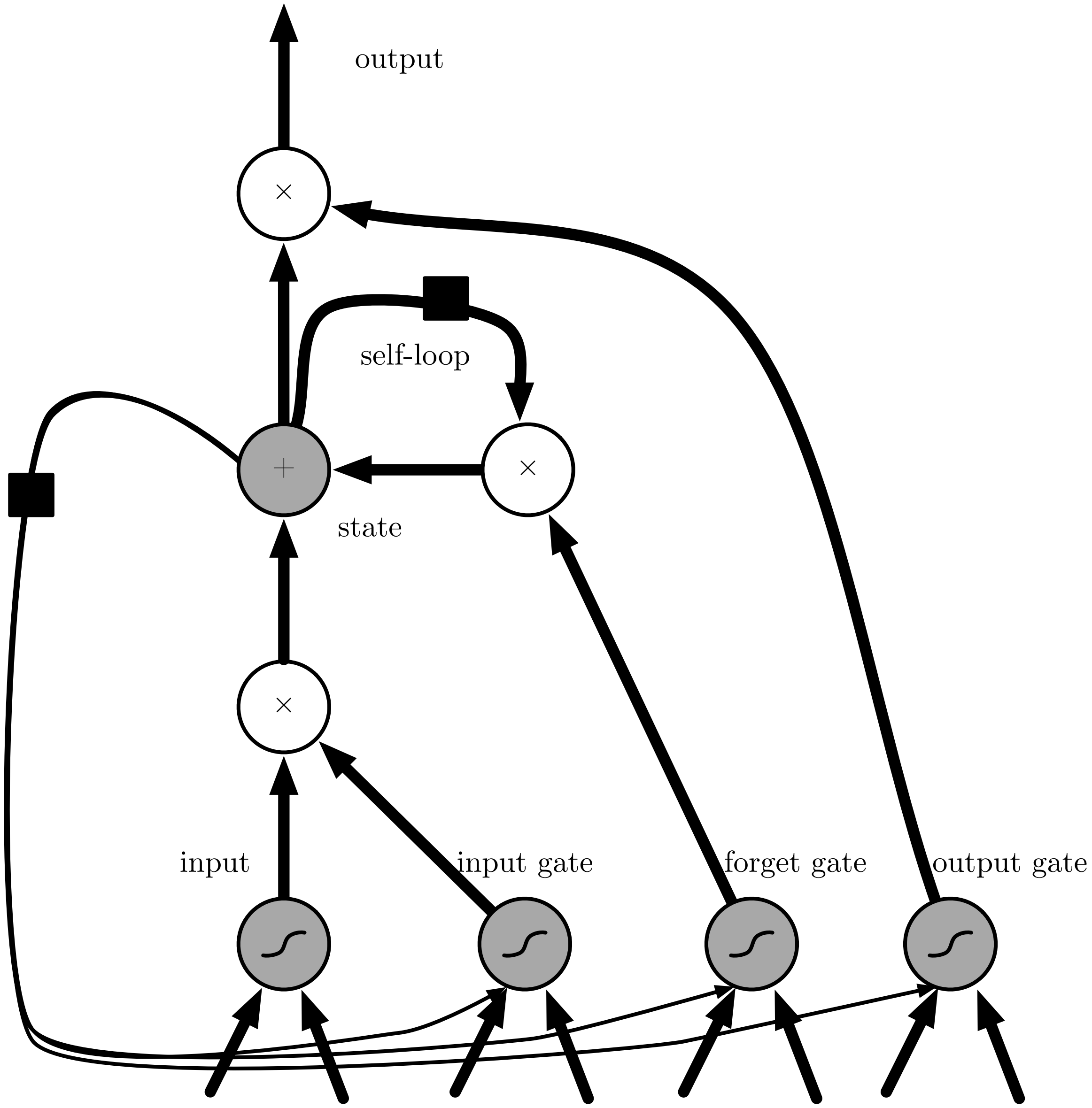
- Clipping gradients (avoid exploding gradients)
- Leaky integration (propagate long-term dependencies)
- Momentum (cheap 2nd order)
- Initialization (start in right ballpark avoids exploding/vanishing)
- Sparse Gradients (symmetry breaking)
- Gradient propagation regularizer (avoid vanishing gradient)
- LSTM self-loops (avoid vanishing gradient)





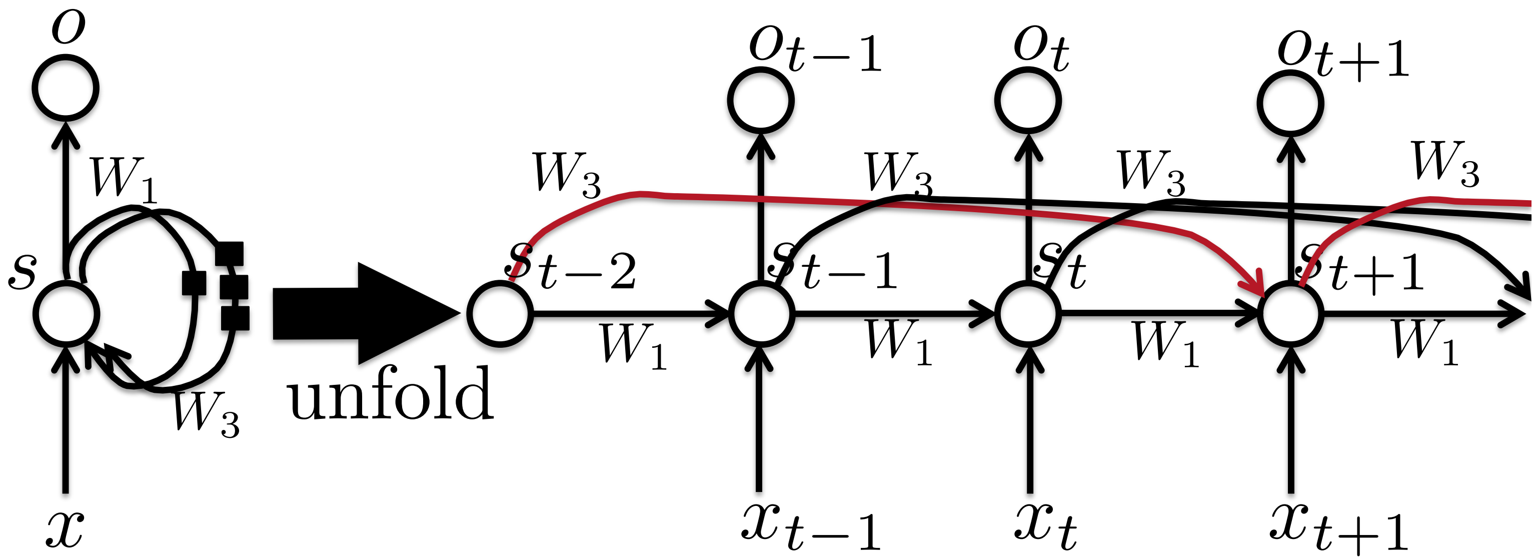
Gated Recurrent Units & LSTM

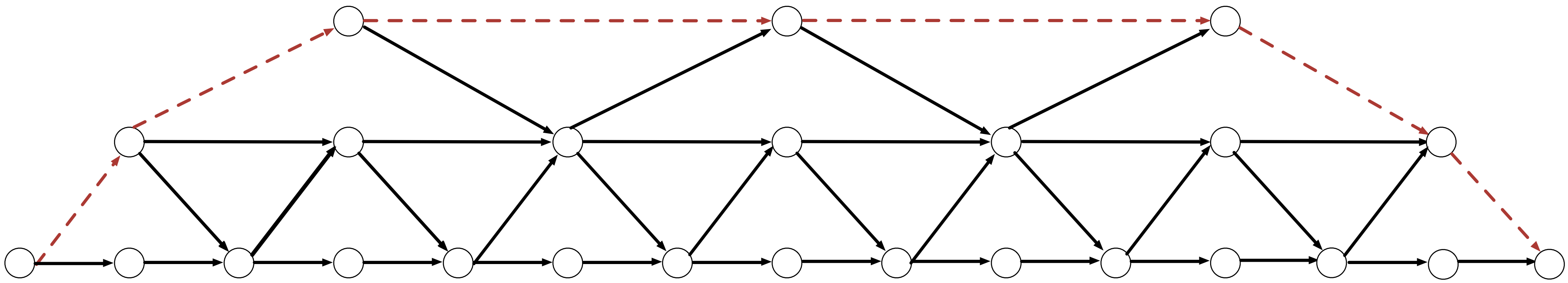
- Create a path where gradients can flow for longer with self-loop
- Corresponds to an eigenvalue of Jacobian slightly less than 1
- LSTM is **heavily used** *(Hochreiter & Schmidhuber 1997)*
- GRU light-weight version *(Cho et al 2014)*



RNN Tricks

- Delays and multiple time scales, Elhihi & Bengio NIPS 1996





Backprop in Practice

Other tricks: see **Deep Learning** book (in preparation, online)

The Convergence of Gradient Descent

$$\omega \leftarrow \omega - \eta \frac{\partial E}{\partial \omega}$$

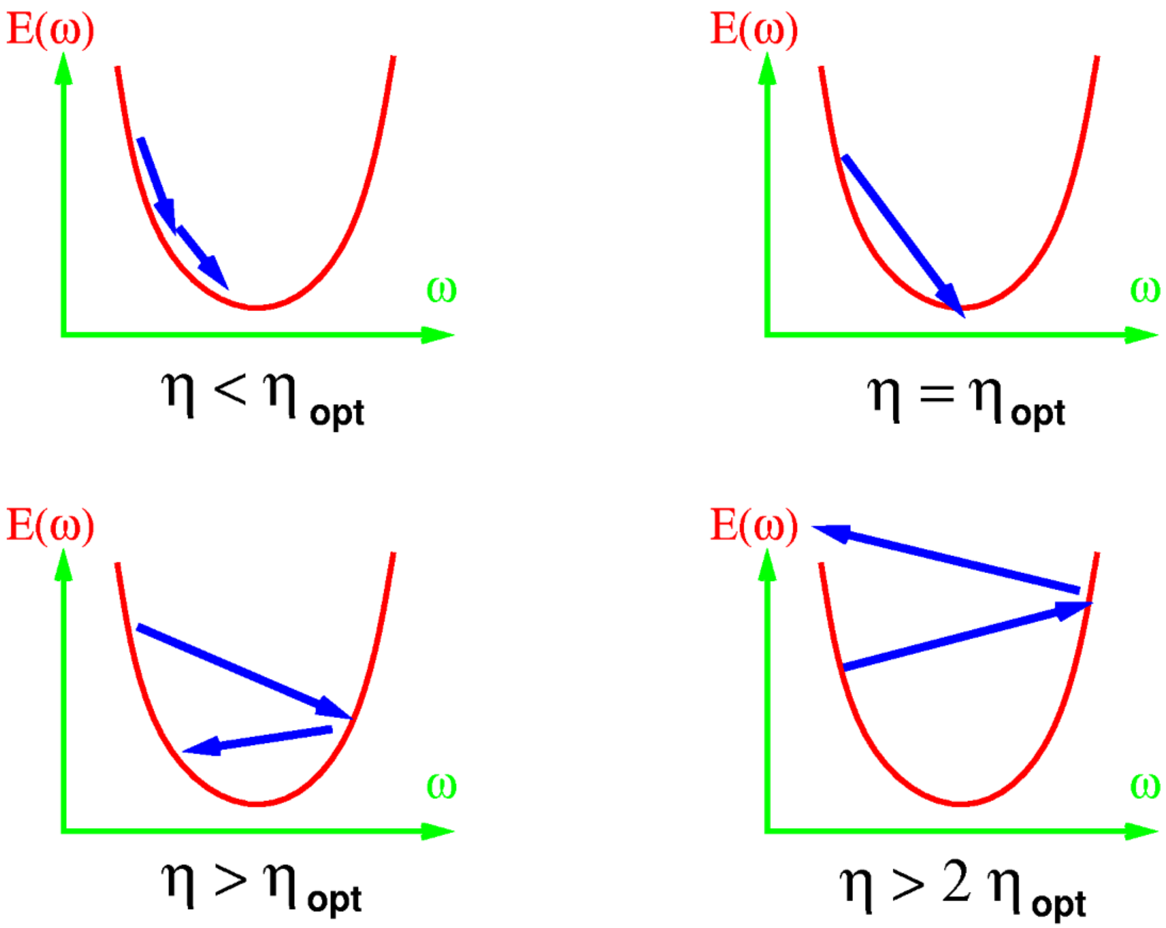
gradient of objective function

weight vector

learning rate

- Batch Gradient
- There is an optimal learning rate
- Equal to inverse 2nd derivative

$$\boxed{\eta_{opt} = \left(\frac{\partial^2 E}{\partial \omega^2}\right)^{-1}}$$



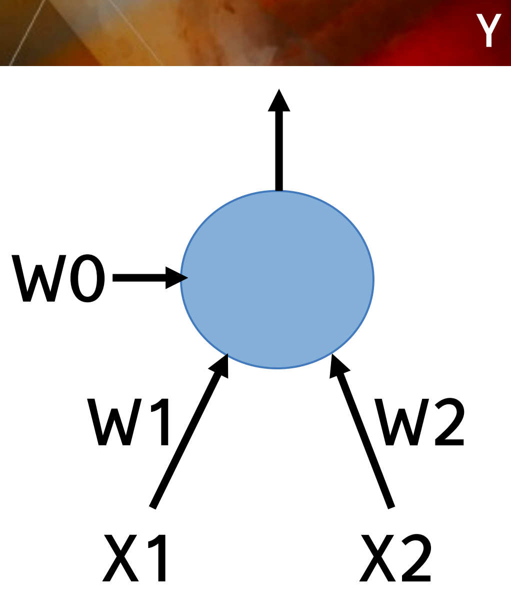
$\eta < \eta_{opt}$

$\eta = \eta_{opt}$

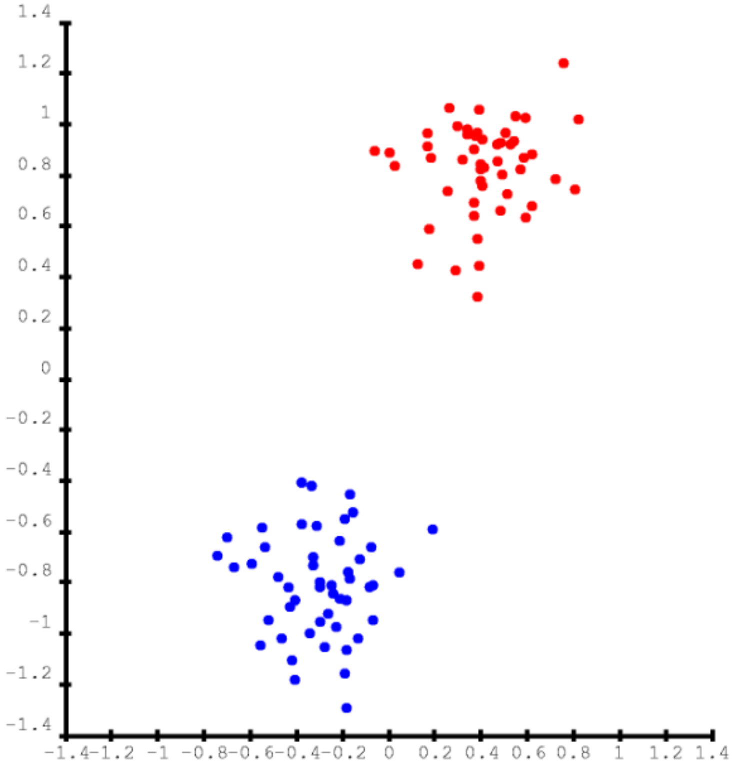
$\eta > \eta_{opt}$

$\eta > 2\,\eta_{opt}$

Let's Look at a single linear unit



- Single unit, 2 inputs
- Quadratic loss
  - $E(W) = 1/p \sum_p (Y - W \bullet X_p)^2$
- Dataset: classification: Y=-1 for blue, +1 for red.
- Hessian is covariance matrix of input vectors
  - $H = 1/p \sum X_p X_p^T$
- To avoid ill conditioning: normalize the inputs
  - Zero mean
  - Unit variance for all variable

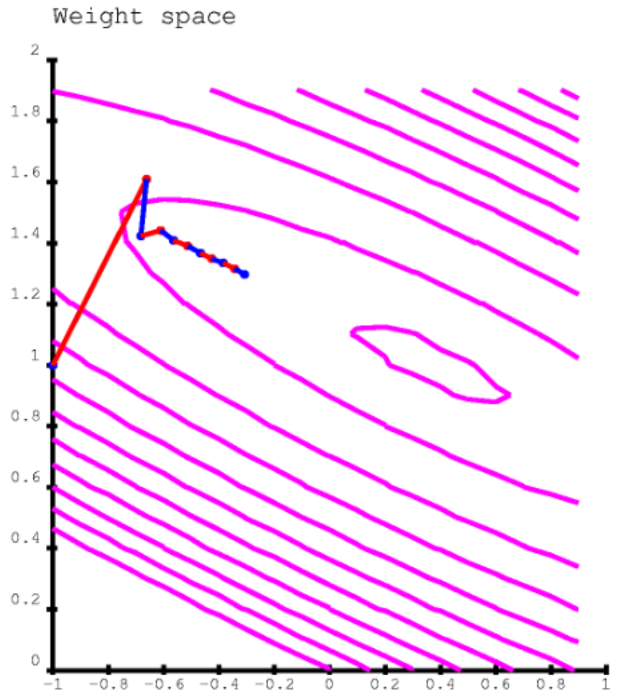


Convergence is Slow When Hessian has Different Eigenvalues

Batch Gradient, small learning rate

Learning rate:

$\eta = 1.5$

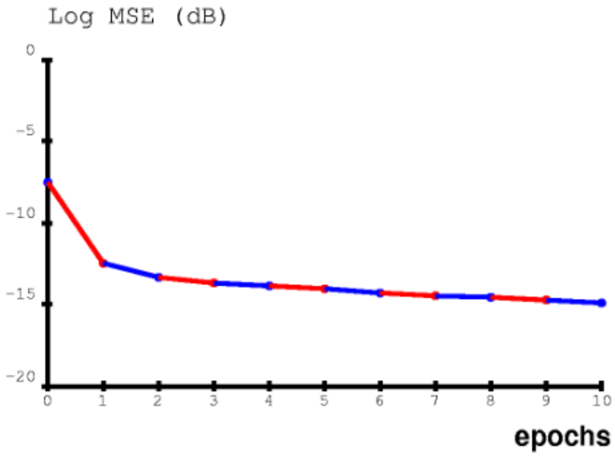


Hessian largest eigenvalue:

$\lambda_{max} = 0.84$

Maximum admissible Learning rate:

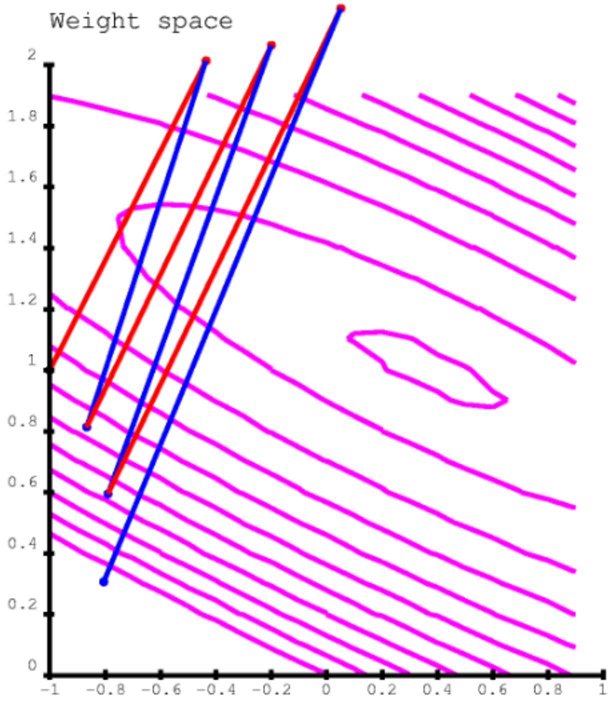
$\eta_{max} = 2.38$



Batch Gradient, large learning rate

Learning rate:

$\eta = 2.5$

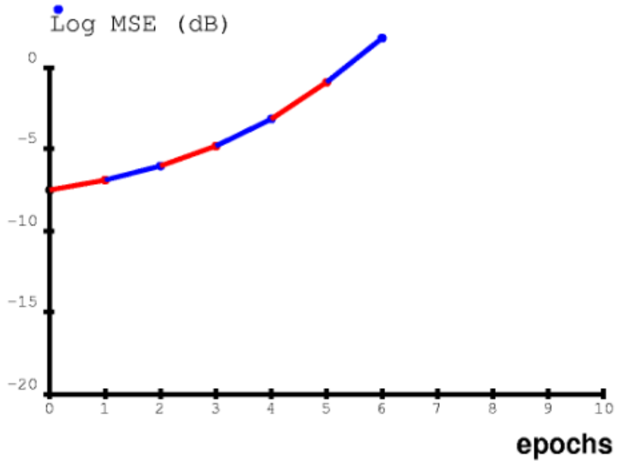


Hessian largest eigenvalue:

$\lambda_{max} = 0.84$

Maximum admissible Learning rate:

$\eta_{max} = 2.38$



Convergence is Slow When Hessian has Different Eigenvalues

- Batch Gradient, small learning rate

Learning rate:

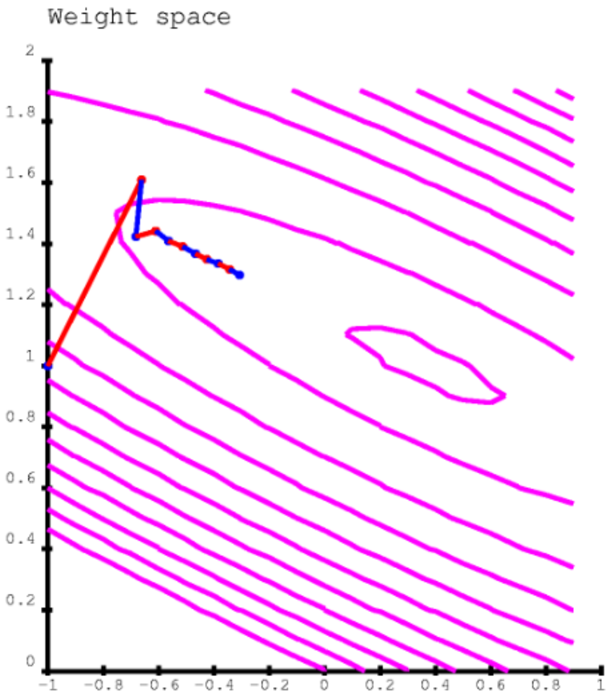
$\eta = 1.5$

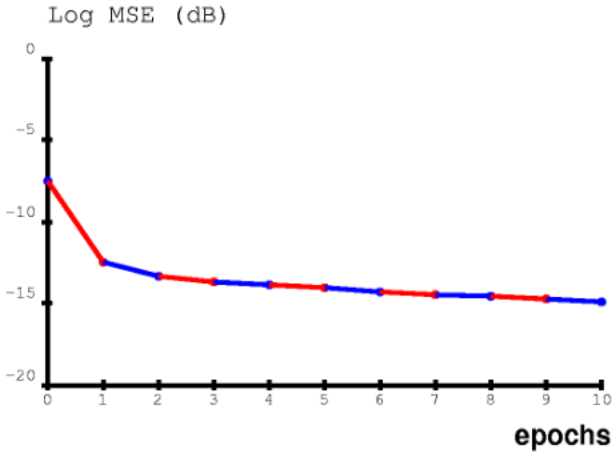
Hessian largest eigenvalue:

$\lambda_{max} = 0.84$

Maximum admissible Learning rate:

$\eta_{max} = 2.38$





- Stochastic Gradient: Much Faster
- But fluctuates near the minimum

Learning rate:

$\eta = 0.2$

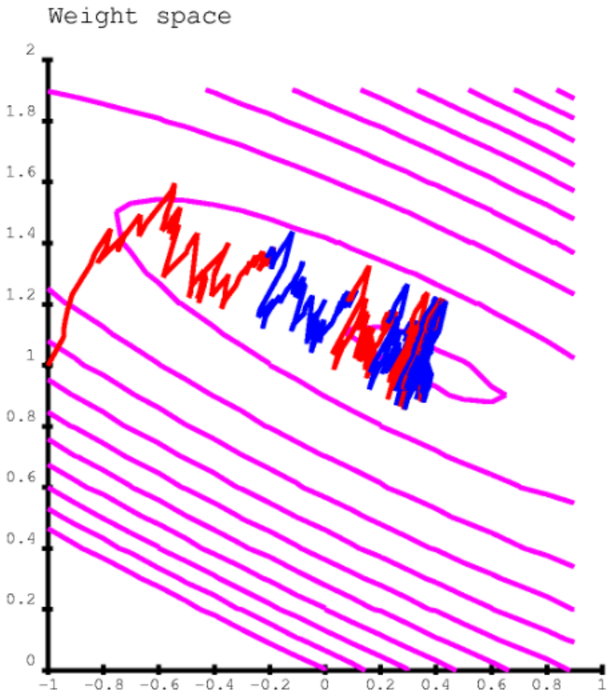
(equivalent to a batch learning rate of 20)

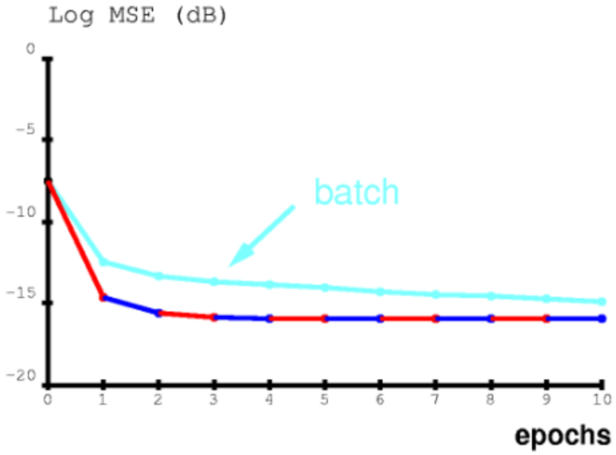
Hessian largest eigenvalue:

$\lambda_{max} = 0.84$

Maximum admissible Learning rate (for batch):

$\eta_{max} = 2.38$

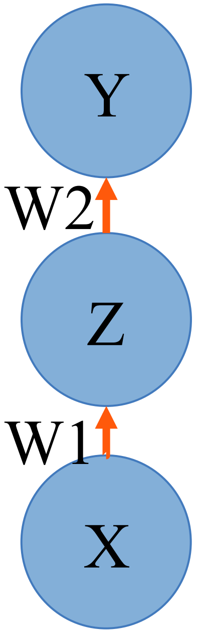


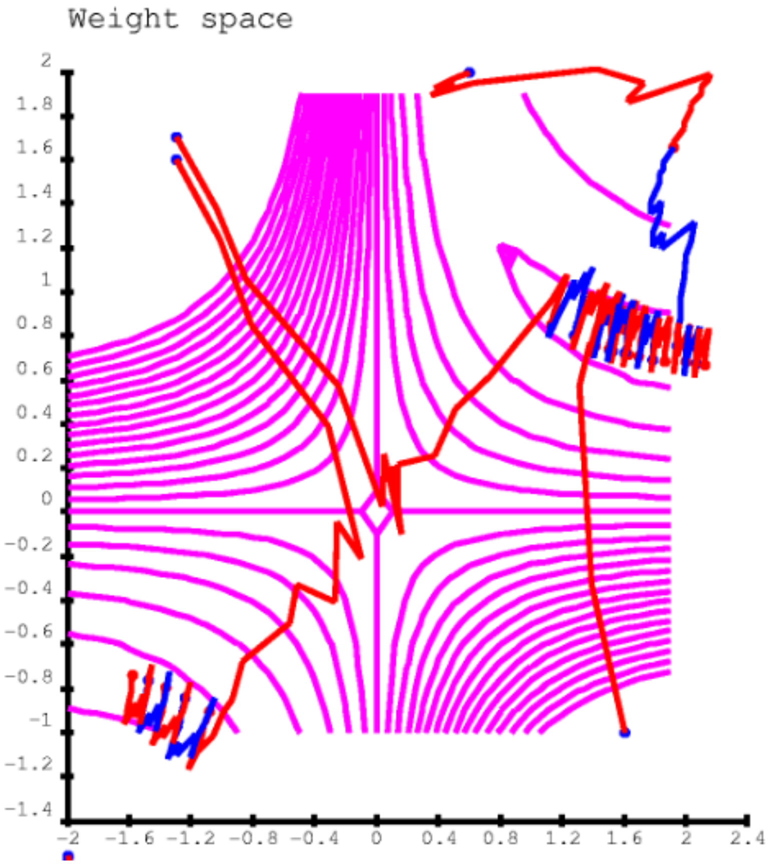


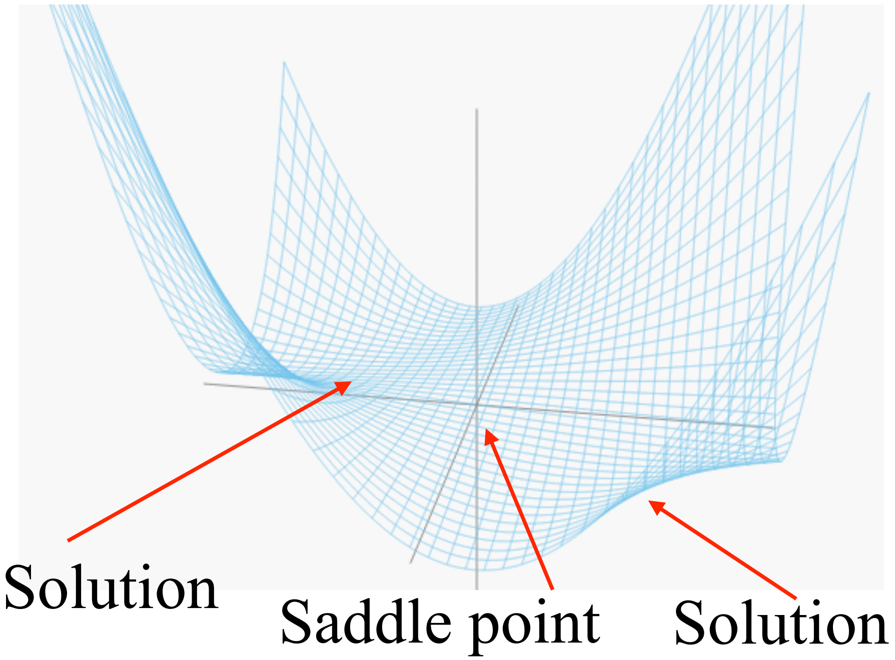
Multilayer Nets Have Non-Convex Objective Functions

Y

- 1-1-1 network
  - Y = W1*W2*X
- trained to compute the identity function with quadratic loss
  - Single sample X=1, Y=1 L(W) = (1-W1*W2)^2
- Solution: W2 = 1/W2 hyperbola.

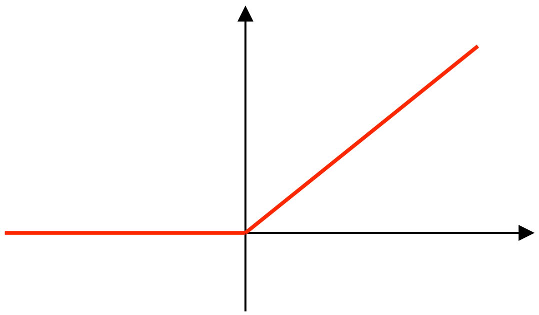




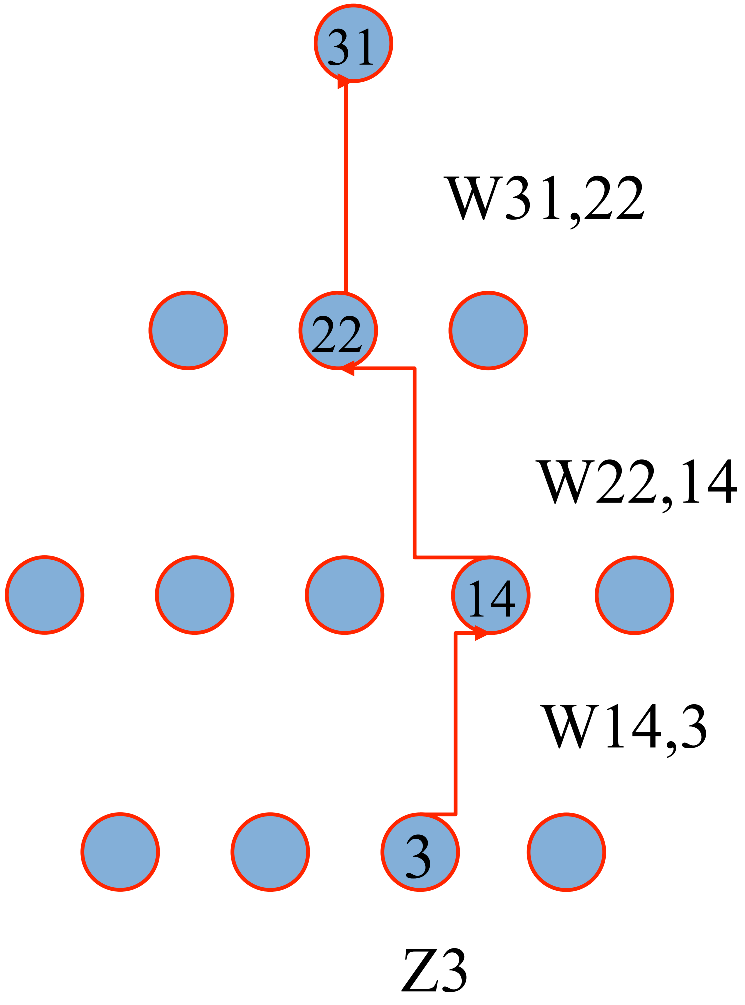


Deep Nets with ReLUs and Max Pooling

- Stack of linear transforms interspersed with Max operators
- Point-wise ReLUs:



- Max Pooling
  - "switches" from one layer to the next
- Input-output function
  - Sum over active paths
  - Product of all weights along the path
  - Solutions are hyperbolas
- Objective function is full of saddle points

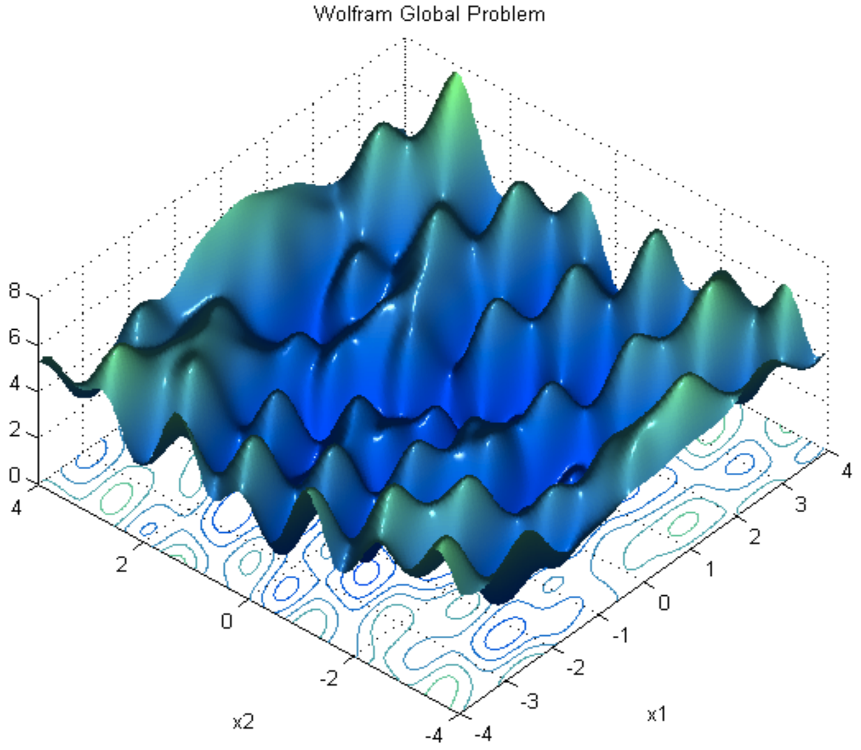


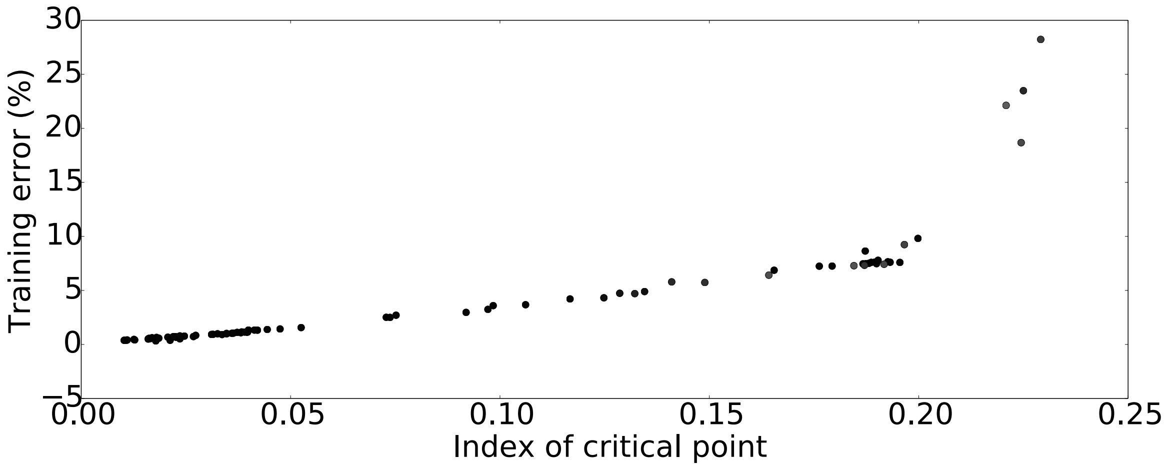
A Myth Has Been Debunked: Local Minima in Neural Nets → Convexity is not needed

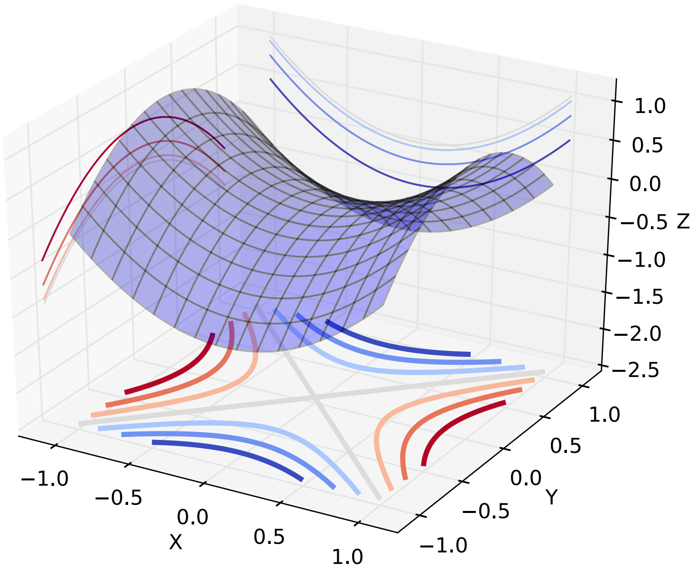
- (Pascanu, Dauphin, Ganguli, Bengio, arXiv May 2014): *On the saddle point problem for non-convex optimization*
- (Dauphin, Pascanu, Gulcehre, Cho, Ganguli, Bengio, NIPS' 2014): *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*
- (Choromanska, Henaff, Mathieu, Ben Arous & LeCun, AISTATS'2015): *The Loss Surface of Multilayer Nets*

Saddle Points

- Local minima dominate in low-D, but saddle points dominate in high-D
- Most local minima are close to the bottom (global minimum error)

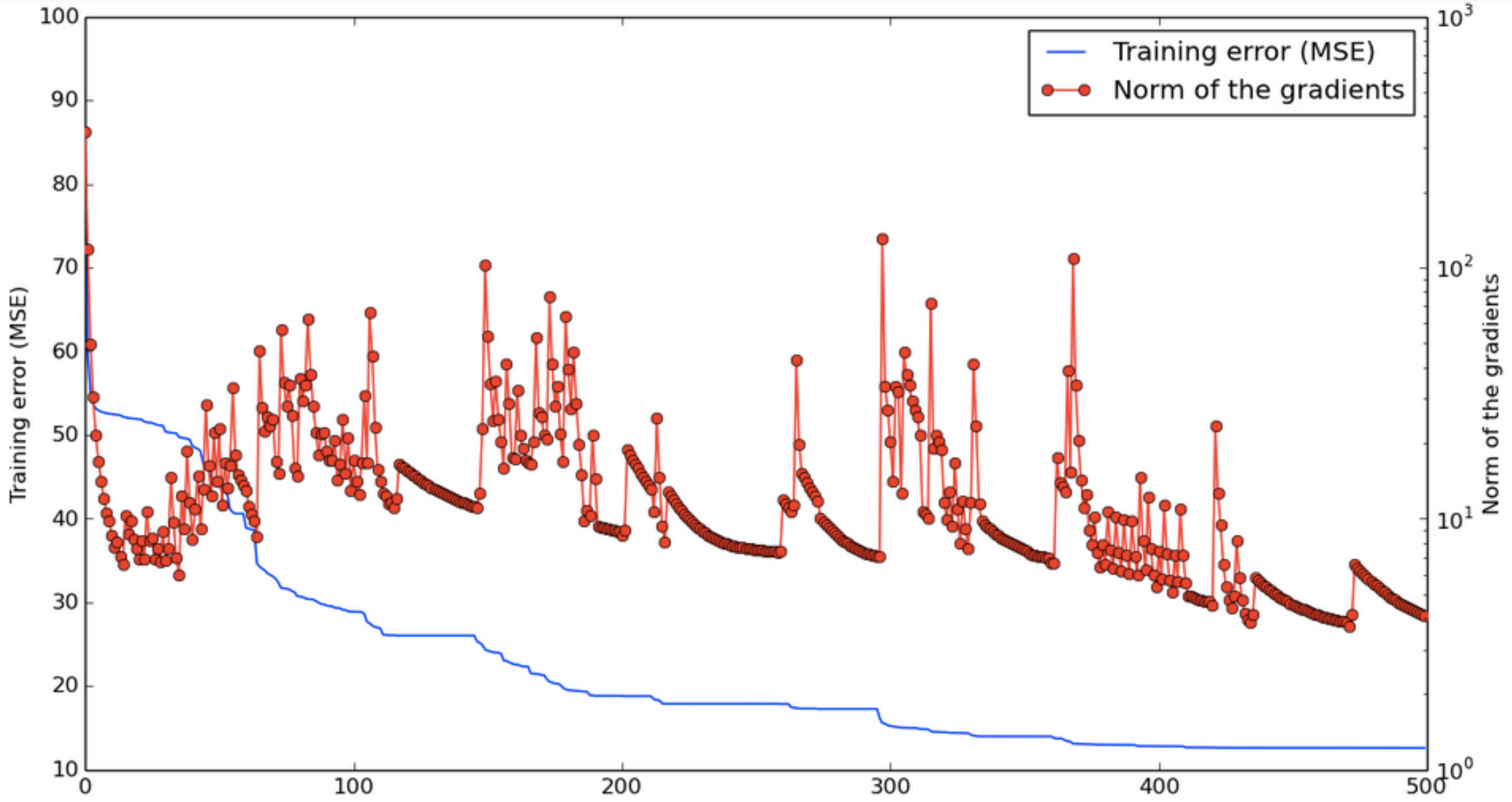






Saddle Points During Training

- Oscillating between two behaviors:
  - Slowly approaching a saddle point
  - Escaping it

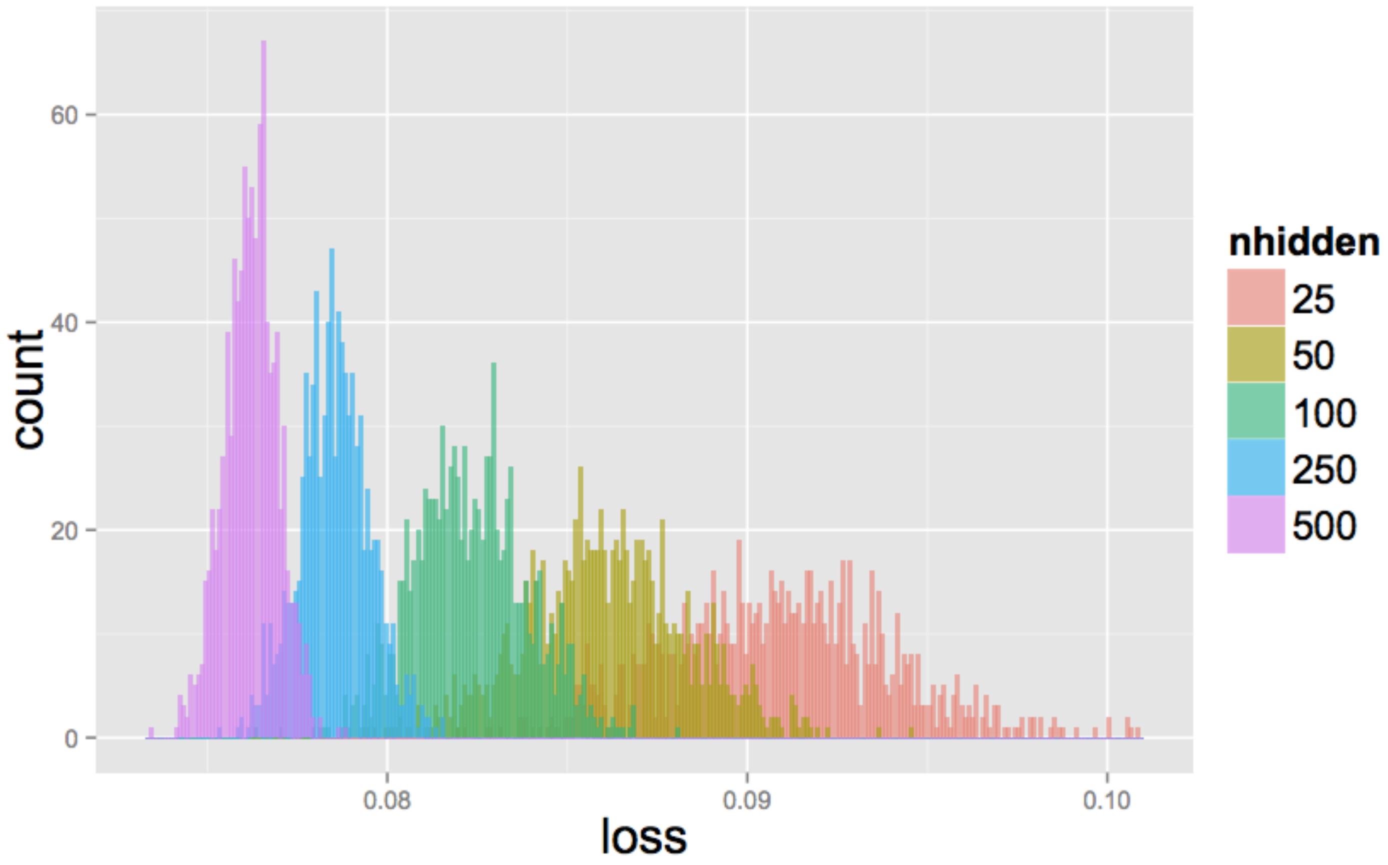


Low Index Critical Points

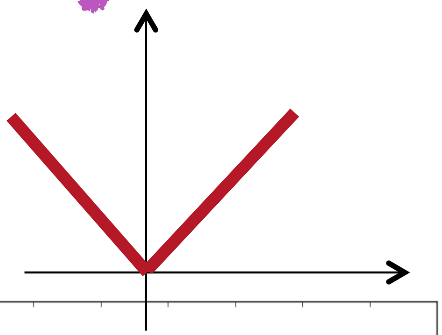
Choromanska et al & LeCun 2014, 'The Loss Surface of Multilayer Nets'

Shows that deep rectifier nets are analogous to spherical spin-glass models

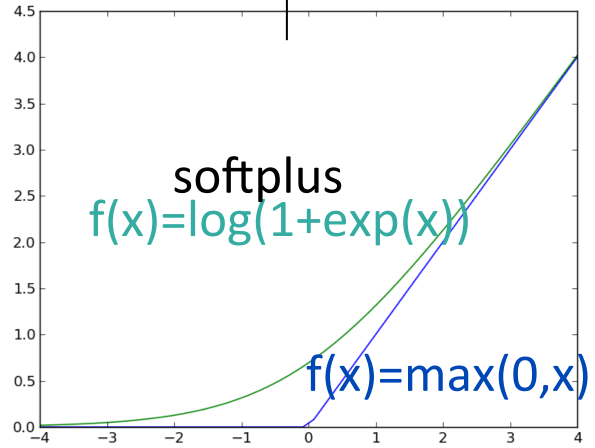
The low-index critical points of large models concentrate in a band just above the global minimum



Piecewise Linear Nonlinearity



- *Jarreth, Kavukcuoglu, Ranzato & LeCun ICCV 2009*: absolute value rectification works better than tanh in lower layers of convnet

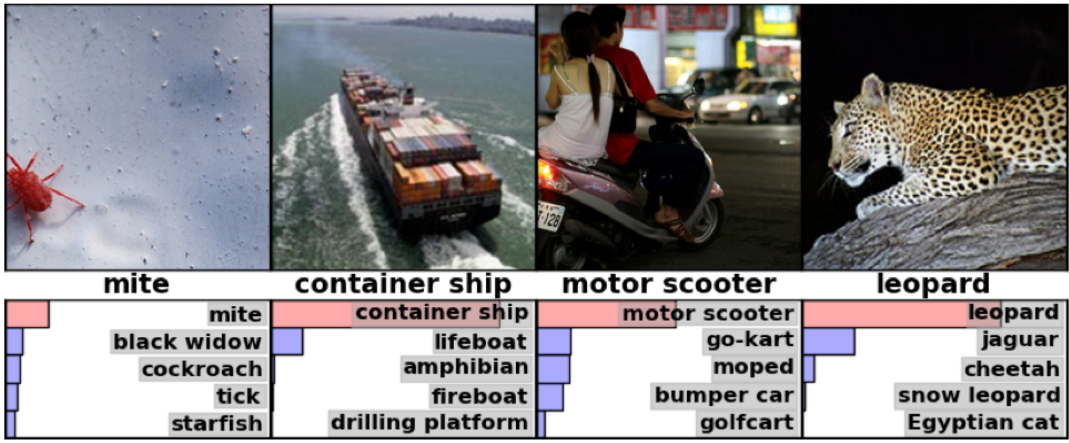


- *Nair & Hinton ICML 2010*: Duplicating sigmoid units with same weights but different bias in an RBM approximates a rectified linear unit (ReLU)
- *Glorot, Bordes and Bengio AISTATS 2011*: Using a rectifier non-linearity (ReLU) instead of tanh of softplus allows for the first time to **train very deep supervised networks** without the need for unsupervised pre-training; was **biologically motivated**

Neuroscience motivations

Leaky integrate-and-fire model

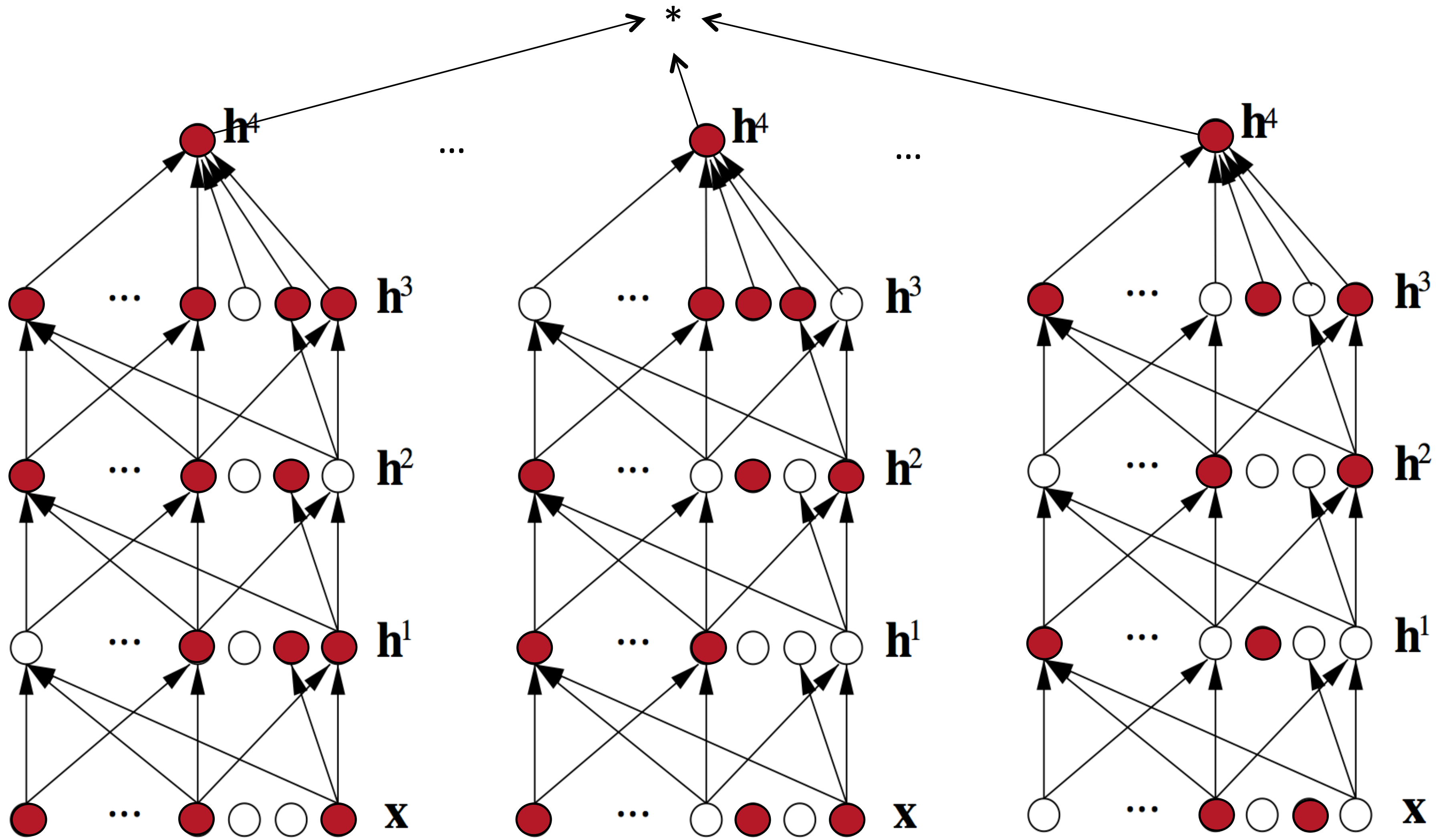
- *Krizhevsky, Sutskever & Hinton NIPS 2012*: rectifiers one of the crucial ingredients in ImageNet breakthrough



Stochastic Neurons as Regularizer: Improving neural networks by preventing co-adaptation of feature detectors (Hinton et al 2012, arXiv)

- **Dropouts** trick: during training multiply neuron output by random bit (p=0.5), during test by 0.5
- Used in deep supervised networks
- Similar to denoising auto-encoder, but corrupting every layer
- Works better with some non-linearities (rectifiers, maxout) (Goodfellow et al. ICML 2013)
- Equivalent to averaging over exponentially many architectures
  - Used by Krizhevsky et al to break through ImageNet SOTA
  - Also improves SOTA on CIFAR-10 (18→16% err)
  - Knowledge-free MNIST with DBMs (.95→.79% err)
  - TIMIT phoneme classification (22.7→19.7% err)

Dropout Regularizer: Super-Efficient Bagging



Batch Normalization

(Ioffe & Szegedy ICML 2015)

- Standardize activations (before nonlinearity) across **minibatch**
- **Backprop through this operation**
- Regularizes & helps to train

$$\bar{\mathbf{x}}_k = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_{i,k},$$

$$\sigma_k^2 = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_{i,k} - \bar{\mathbf{x}}_k)^2$$

$$\hat{\mathbf{x}}_k = \frac{\mathbf{x}_k - \bar{\mathbf{x}}_k}{\sqrt{\sigma_k^2 + \epsilon}}$$

$$BN(\mathbf{x}_k) = \gamma_k \hat{\mathbf{x}}_k + \beta_k$$

$$\mathbf{y} = \phi(BN(\mathbf{Wx}))$$

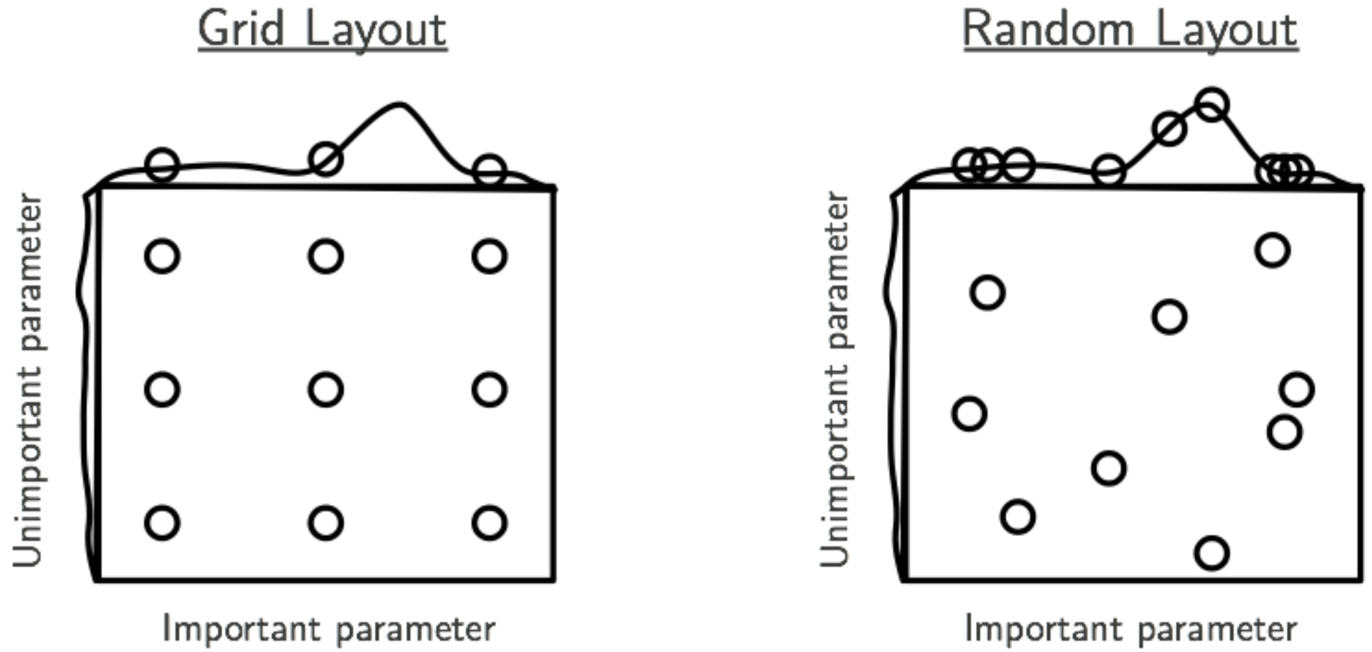
Early Stopping

- Beautiful **FREE LUNCH** (no need to launch many different training runs for each value of hyper-parameter for #iterations)
- Monitor validation error during training (after visiting # of training examples = a multiple of validation set size)
- Keep track of parameters with best validation error and report them at the end
- If error does not improve enough (with some patience), stop.

Random Sampling of Hyperparameters

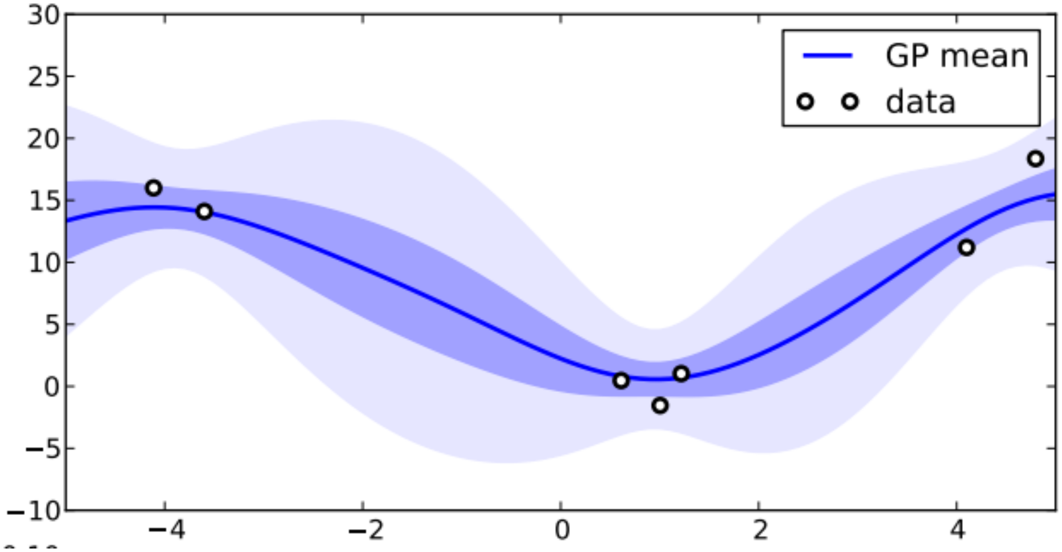
(Bergstra & Bengio 2012)

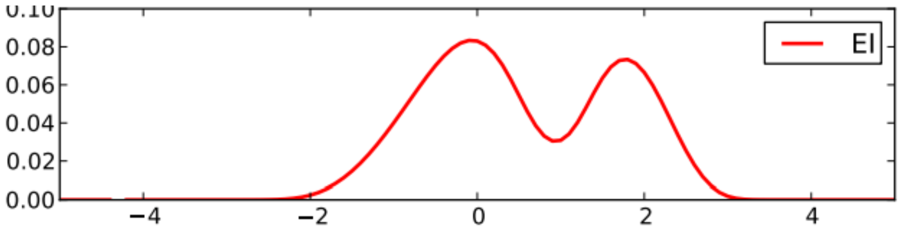
- Common approach: manual + grid search
- Grid search over hyperparameters: simple & wasteful
- Random search: simple & efficient
  - Independently sample each HP, e.g. l.rate~exp(U[log(.1),log(.0001)])
  - Each training trial is iid
  - If a HP is irrelevant grid search is wasteful
  - More convenient: ok to early-stop, continue further, etc.



Sequential Model-Based Optimization of Hyper-Parameters

- (Hutter et al JAIR 2009; Bergstra et al NIPS 2011; Thornton et al arXiv 2012; Snoek et al NIPS 2012)
- Iterate
- Estimate P(valid. err | hyper-params config x, D)
- choose optimistic x, e.g. $\max_x$ P(valid. err < current min. err | x)
- train with config x, observe valid. err. v, D ← D U {(x,v)}





Distributed Training

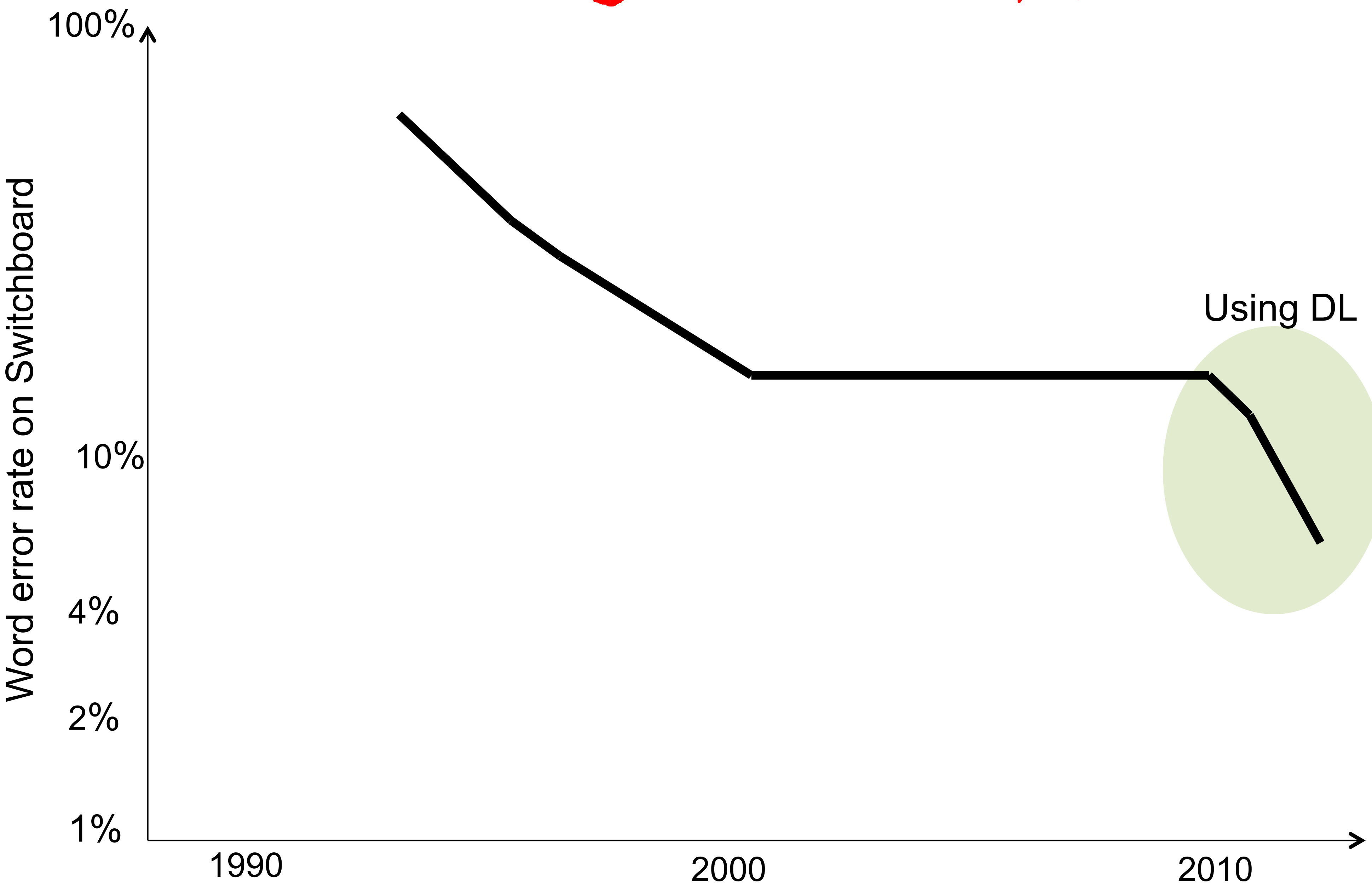
- Minibatches
- Large minibatches + 2^{nd} order & natural gradient methods
- Asynchronous SGD (Bengio et al 2003, Le et al ICML 2012, Dean et al NIPS 2012)
  - Data parallelism vs model parallelism
  - Bottleneck: sharing weights/updates among nodes, to avoid node-models to move too far from each other
- EASGD *(Zhang et al NIPS 2015)* works well in practice
- Efficiently exploiting more than a few GPUs remains a challenge

Vision

(switch laptops)

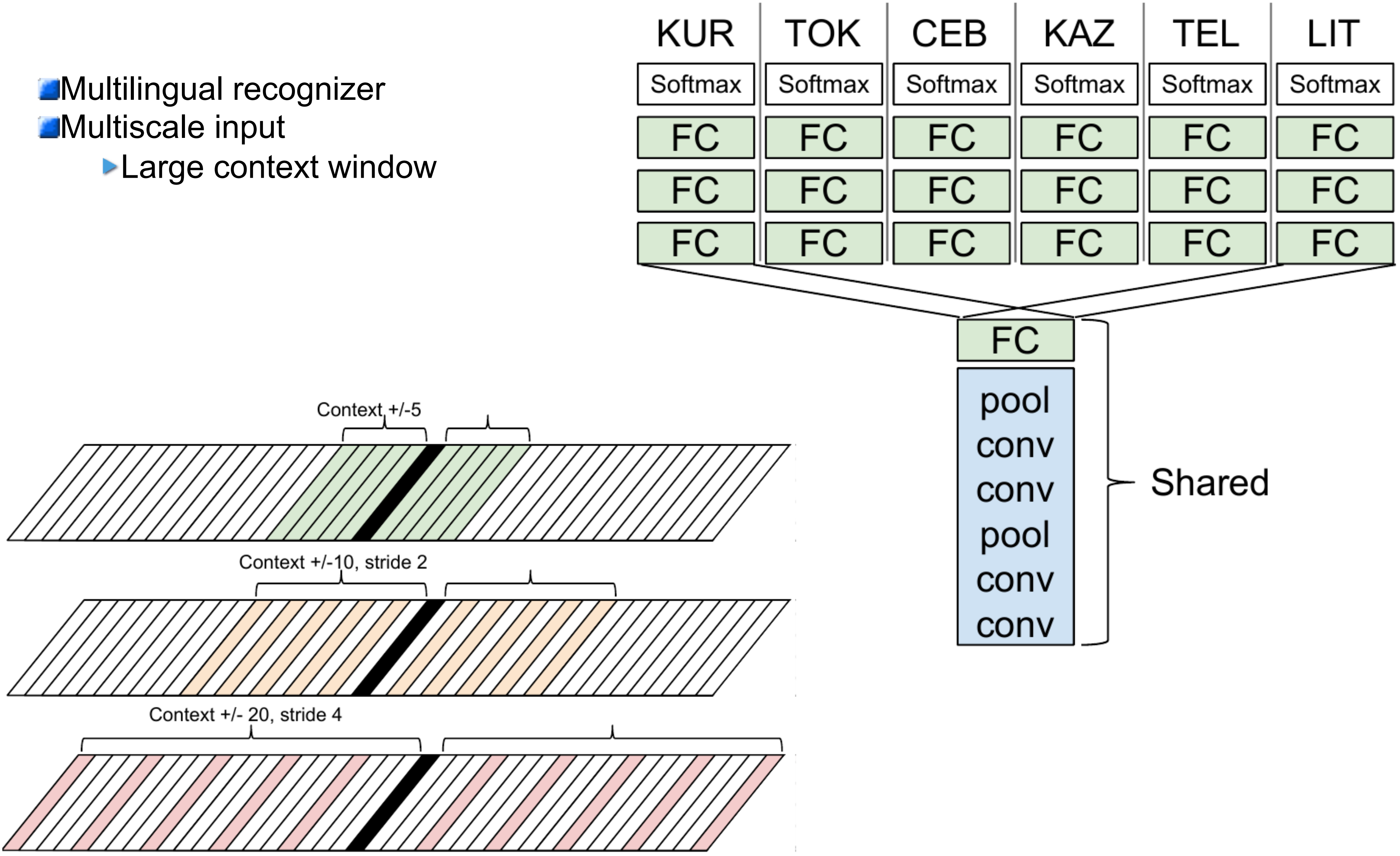
Speech Recognition

The dramatic impact of Deep Learning on Speech Recognition (according to Microsoft)

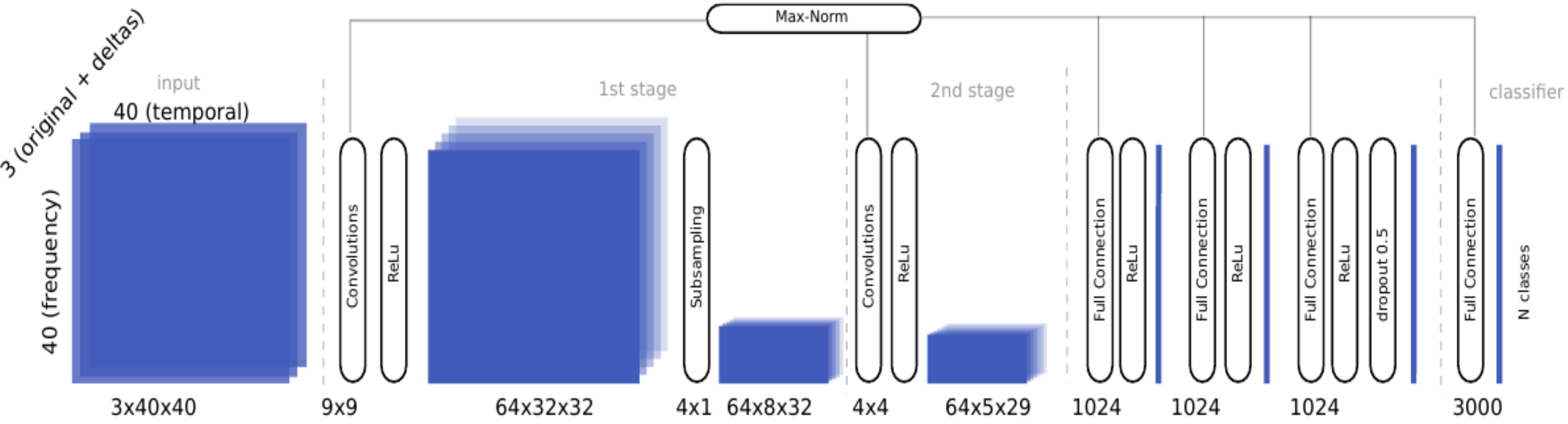


Speech Recognition with Convolutional Nets (NYU/IBM)

- Multilingual recognizer
- Multiscale input
  - Large context window



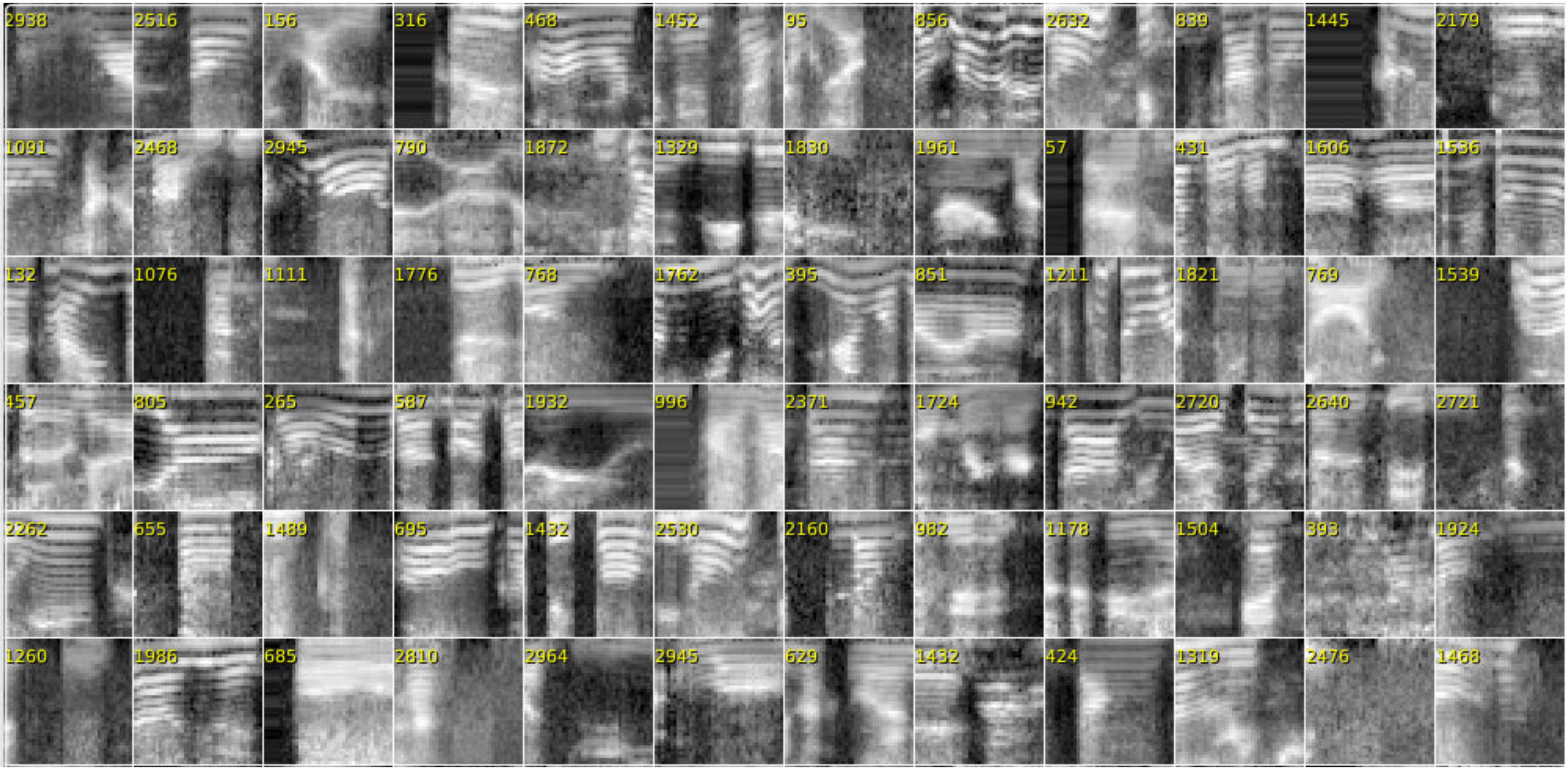
Speech Recognition with Convolutional Nets (NYU/IBM)



- Acoustic Model: ConvNet with 7 layers. 54.4 million parameters.
- Classifies acoustic signal into 3000 context-dependent subphones categories
- ReLU units + dropout for last layers
- Trained on GPU. 4 days of training

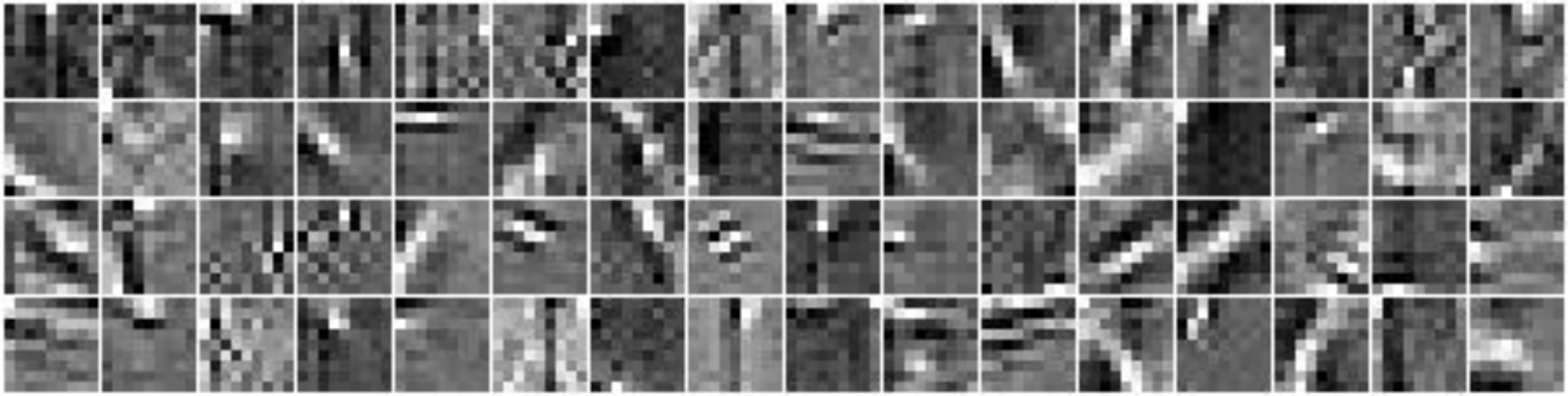
Speech Recognition with Convolutional Nets (NYU/IBM)

- Training samples.
  - 40 MEL-frequency Cepstral Coefficients
  - Window: 40 frames, 10ms each



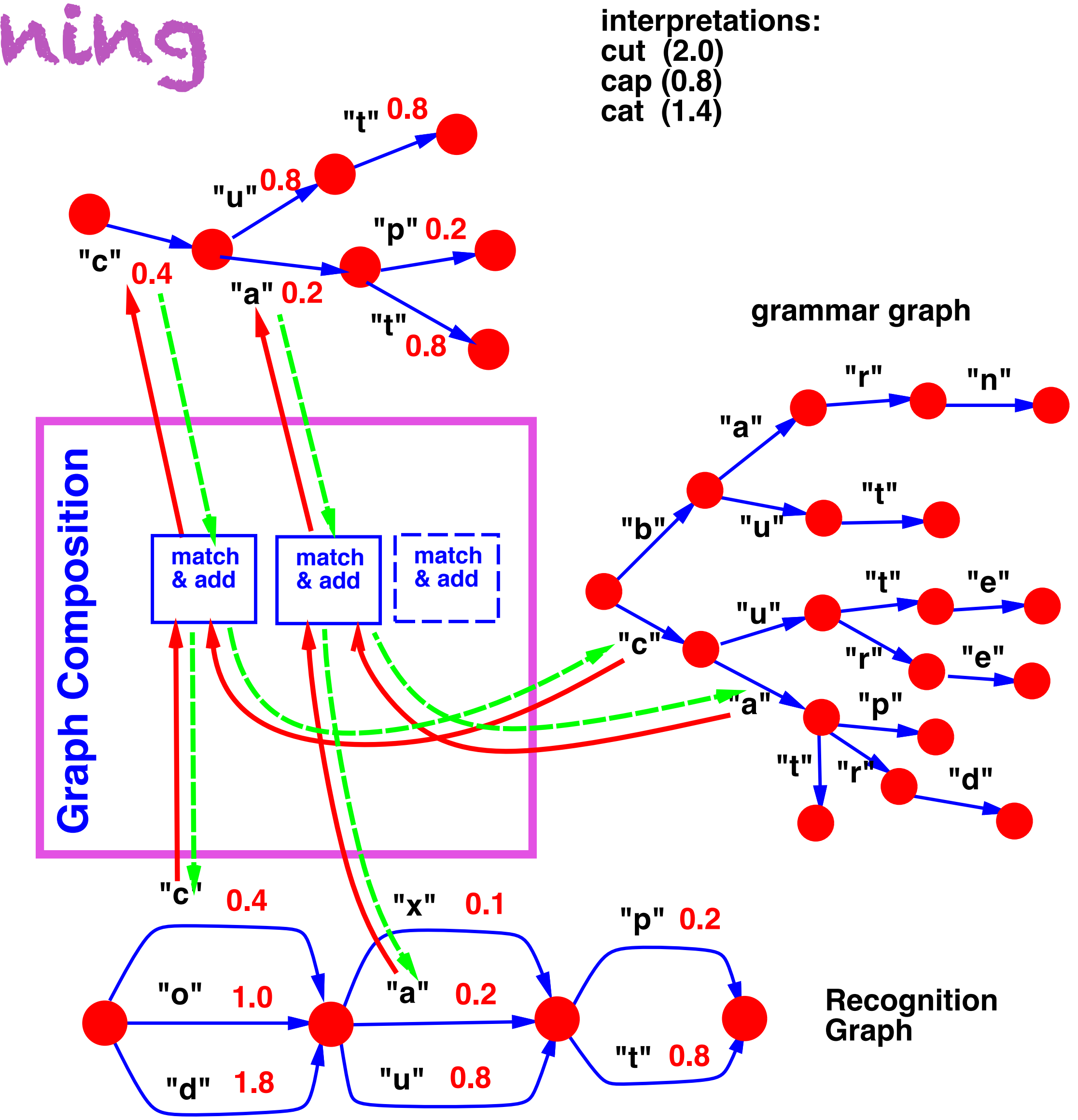
Speech Recognition with Convolutional Nets (NYU/IBM)

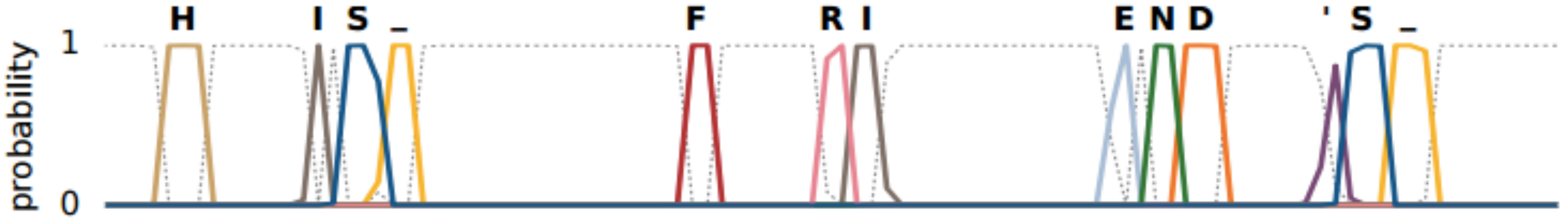
- Convolution Kernels at Layer 1:
  - 64 kernels of size 9x9



End-to-End Training with Search

- Hybrid systems, neural nets + HMMs *(Bengio 1991, Bottou 1991)*
- Neural net outputs scores for each arc, recognized output = labels along best path; trained discriminatively *(LeCun et al 1998)*
- Connectionist Temporal Classification (Graves 2006)
- DeepSpeech and attention-based end-to-end RNNs (Hannun et al 2014; Graves & Jaitly 2014; Chorowski et al NIPS 2015)

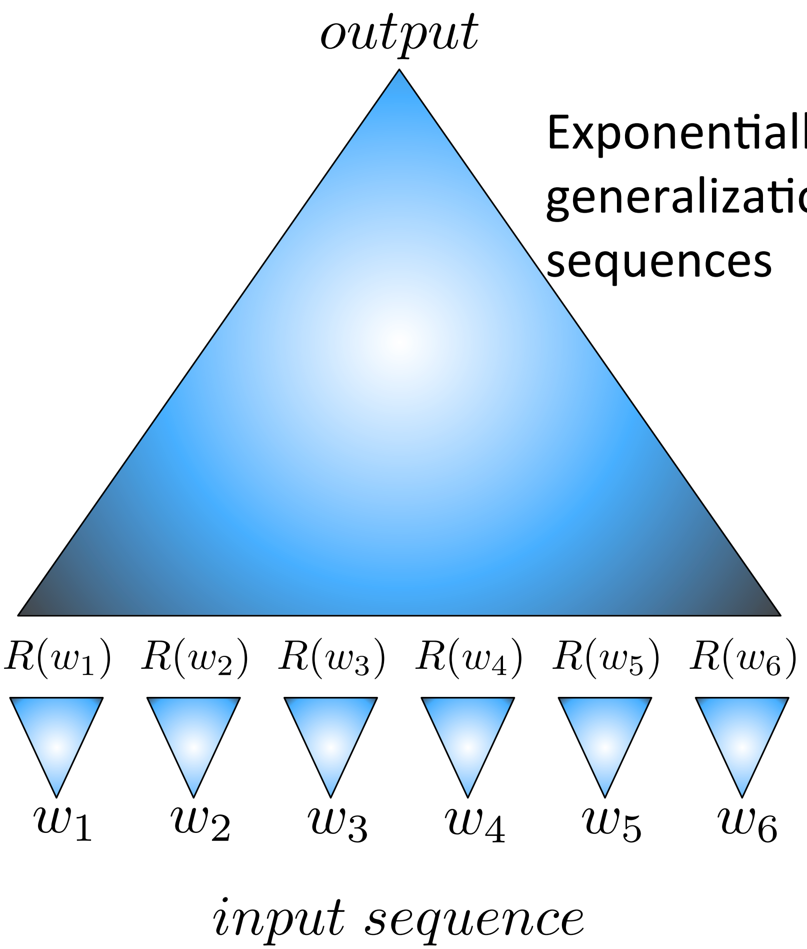




Natural Language Representations

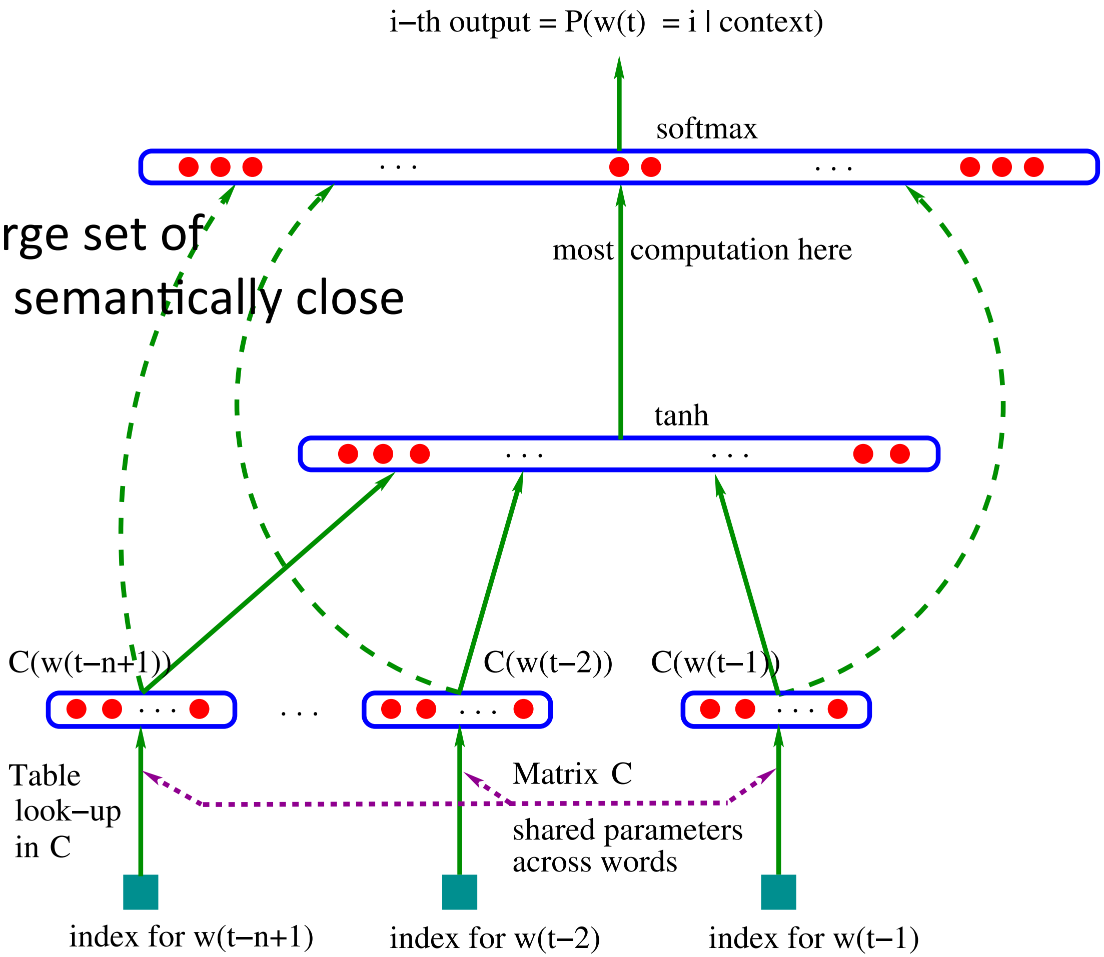
Neural Language Models: fighting one exponential by another one!

- (Bengio et al NIPS'2000)



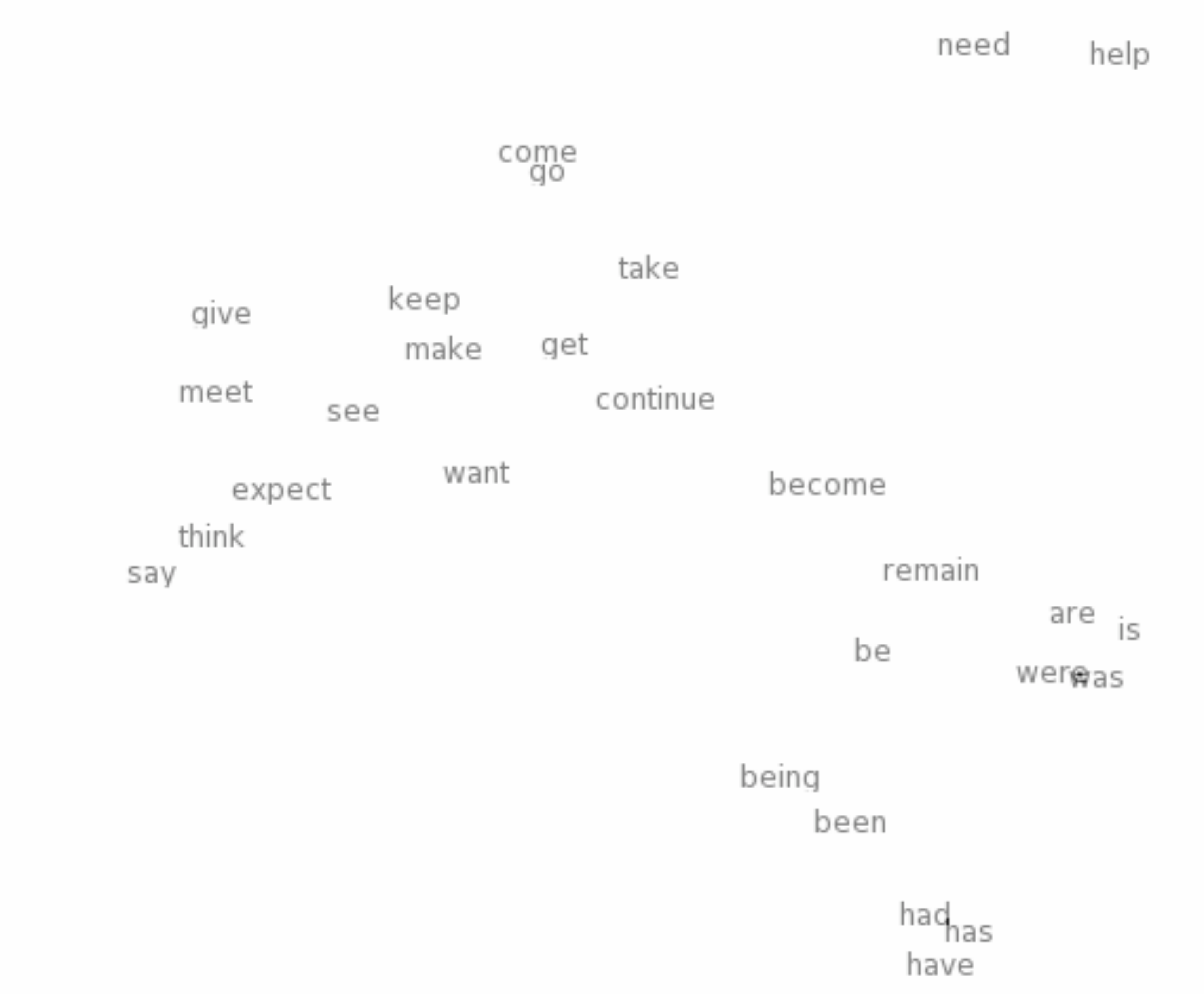
output
$R(w_1)$ $R(w_2)$ $R(w_3)$ $R(w_4)$ $R(w_5)$ $R(w_6)$
w_1 w_2 w_3 w_4 w_5 w_6
input sequence

Exponentially large set of generalizations: semantically close sequences



Exponentially large set of possible contexts

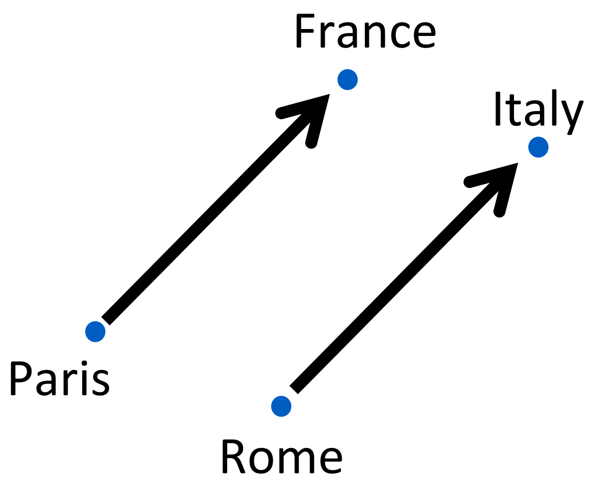
Neural word embeddings: visualization directions = Learned Attributes



Analogical Representations for Free

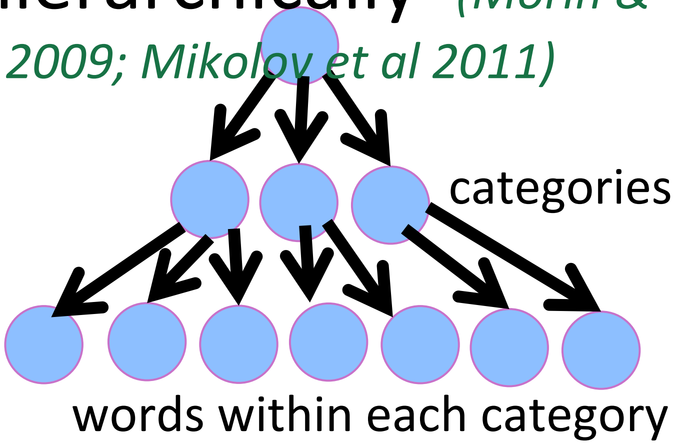
(Mikolov et al, ICLR 2013)

- Semantic relations appear as linear relationships in the space of learned representations
- King – Queen ≈ Man – Woman
- Paris – France + Italy ≈ Rome



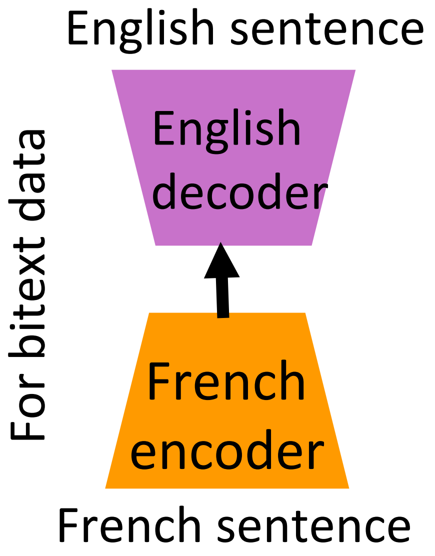
Handling Large Output Spaces

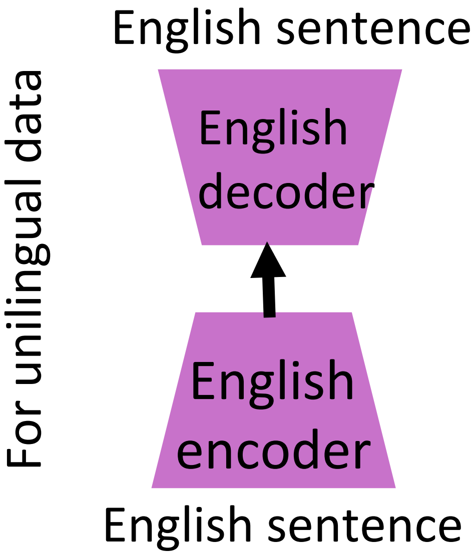
- Sampling "negative" examples: increase score of correct word and stochastically decrease all the others
  - Uniform sampling *(Collobert & Weston, ICML 2008)*
  - Importance sampling, *(Bengio & Senecal AISTATS 2003; Dauphin et al ICML 2011) ;* GPU friendly implementation *(Jean et al ACL 2015)*
- Decompose output probabilities hierarchically *(Morin & Bengio 2005; Blitzer et al 2005; Mnih & Hinton 2007,2009; Mikolov et al 2011)*

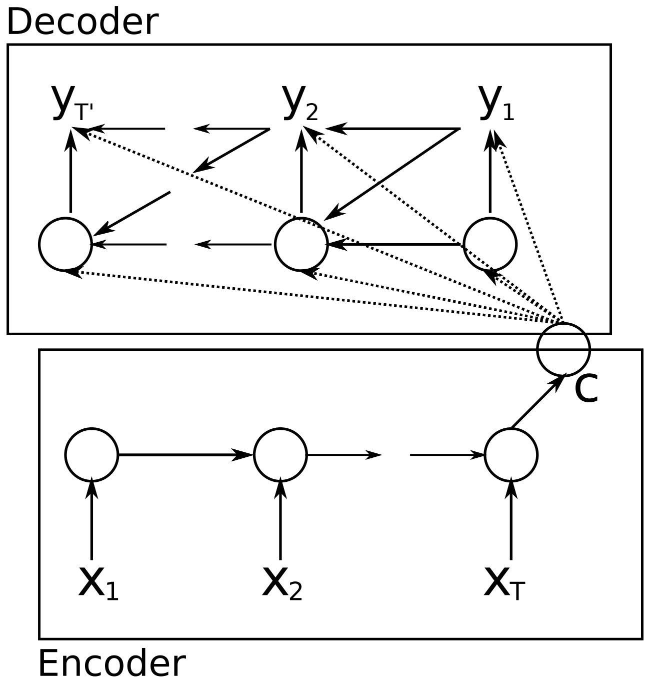


Encoder-Decoder Framework

- Intermediate representation of meaning
  = ‘universal representation’
- Encoder: from word sequence to sentence representation
- Decoder: from representation to word sequence distribution



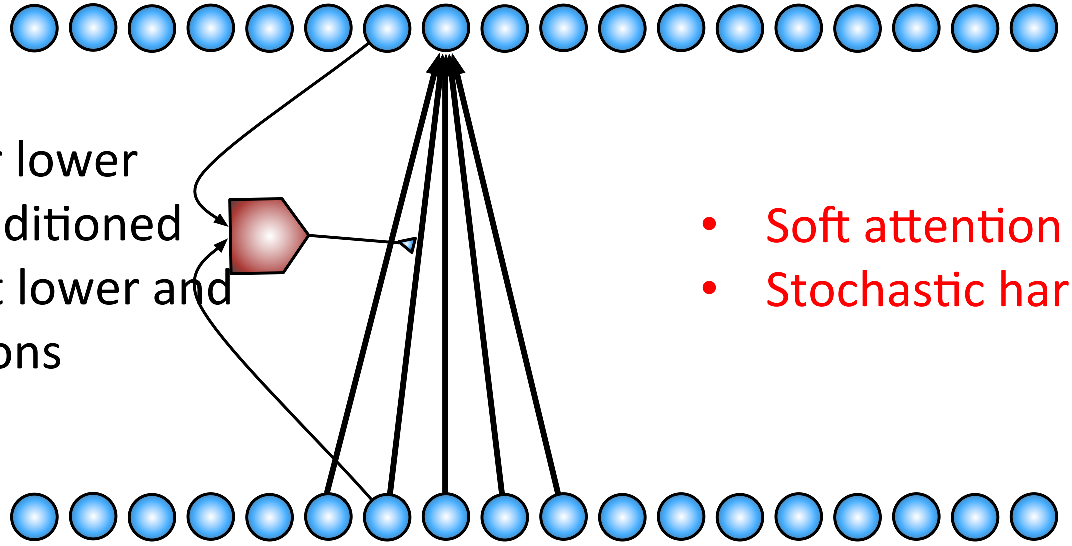




(Cho et al EMNLP 2014; Sutskever et al NIPS 2014)

Attention Mechanism for Deep Learning

- Consider an input (or intermediate) sequence or image
- Consider an upper level representation, which can choose « where to look », by assigning a weight or probability to each input position, as produced by an MLP, applied at each position



Higher-level

Softmax over lower locations conditioned on context at lower and higher locations

Lower-level

- Soft attention (backprop) vs
- Stochastic hard attention (RL)

(Bahdanau, Cho & Bengio, arXiv sept. 2014) following up on *(Graves 2013)* and *(Larochelle & Hinton NIPS 2010)*

End-to-End Machine Translation with Recurrent Nets and Attention Mechanism

(Bahdanau et al 2014, Jean et al 2014, Gulcehre et al 2015, Jean et al 2015)

- Reached the state-of-the-art in one year, from scratch

(a) **English→French (WMT-14)**

	NMT(A)	Google	P-SMT
NMT	32.68	$30.6^{\star}$	**$37.03^{\bullet}$**
+Cand	33.28	–	
+UNK	33.99	32.7°	
+Ens	**36.71**	**36.9°**	

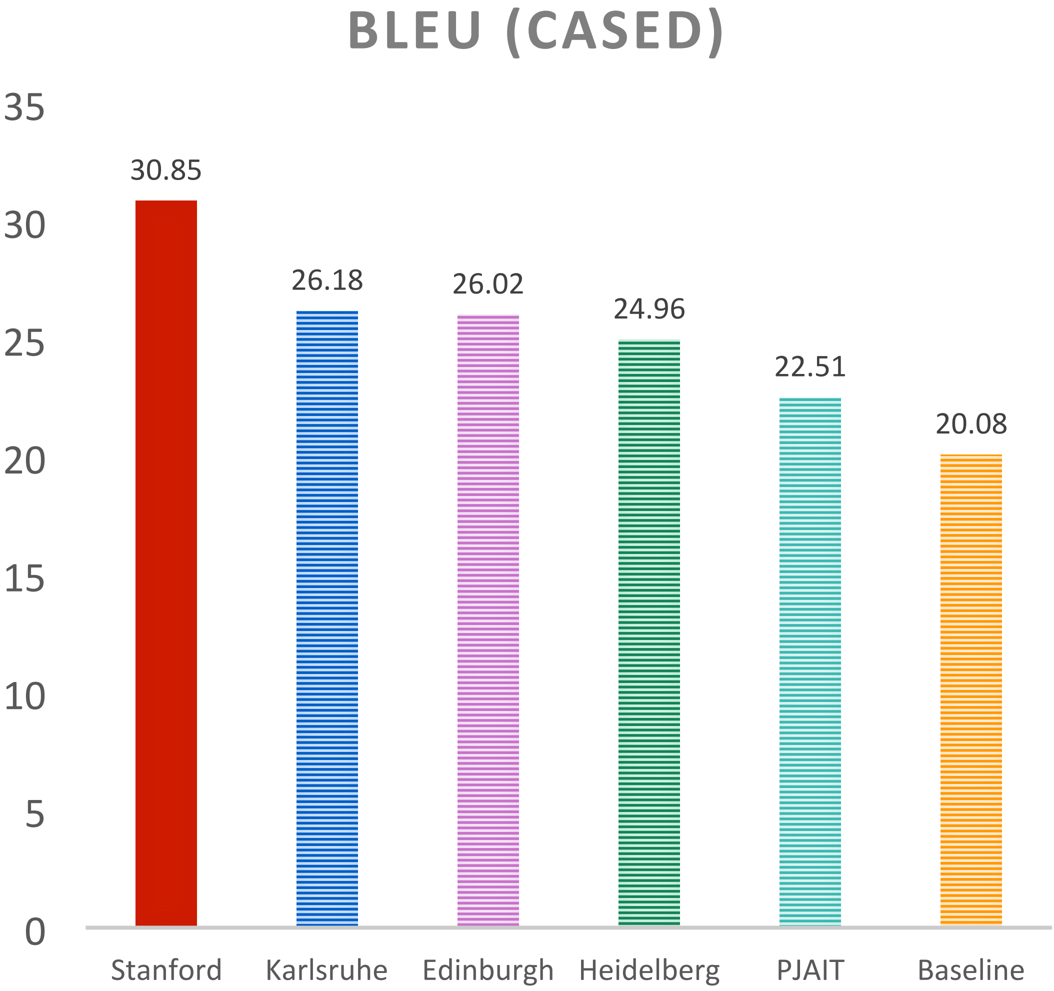
(b) **English→German (WMT-15)**

Model	Note
24.8	Neural MT
24.0	U.Edinburgh, Syntactic SMT
23.6	LIMSI/KIT
22.8	U.Edinburgh, Phrase SMT
22.7	KIT, Phrase SMT

(c) **English→Czech (WMT-15)**

Model	Note
18.3	Neural MT
18.2	JHU, SMT+LM+OSM+Sparse
17.6	CU, Phrase SMT
17.4	U.Edinburgh, Phrase SMT
16.1	U.Edinburgh, Syntactic SMT

IWSLT 2015 – Luong & Manning (2015)
TED talk MT, English-German



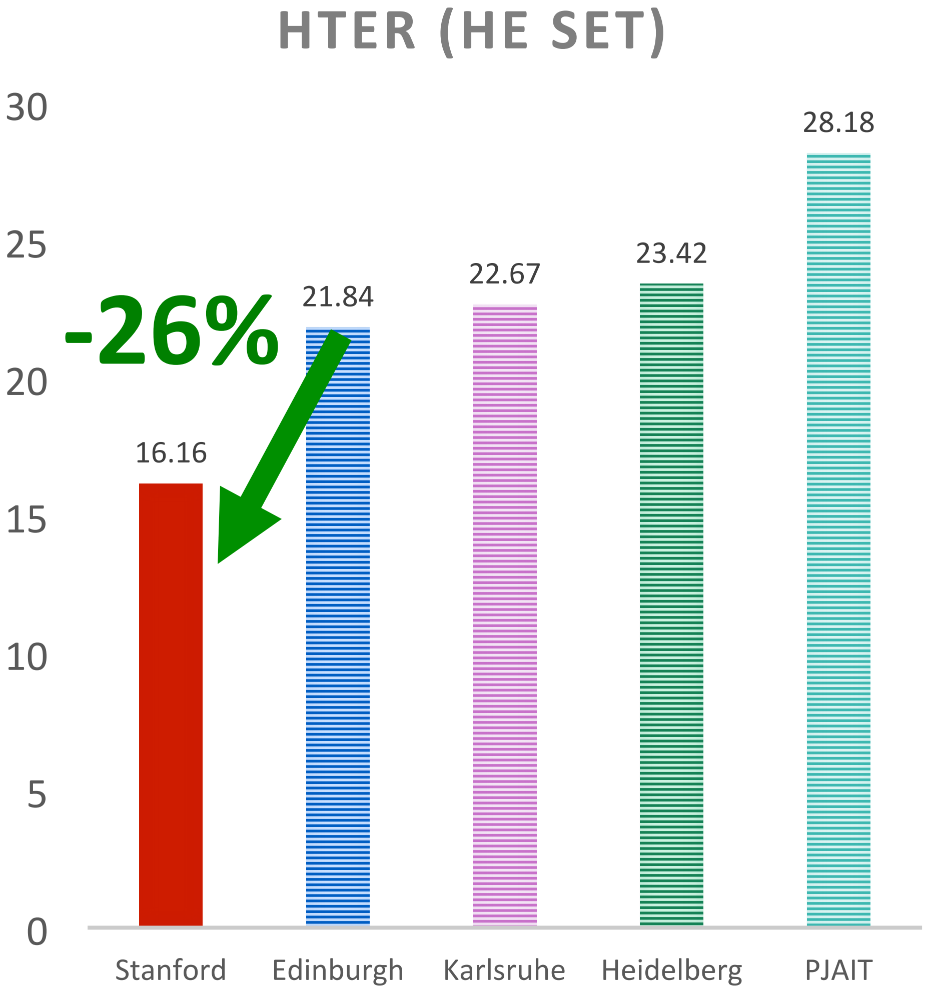
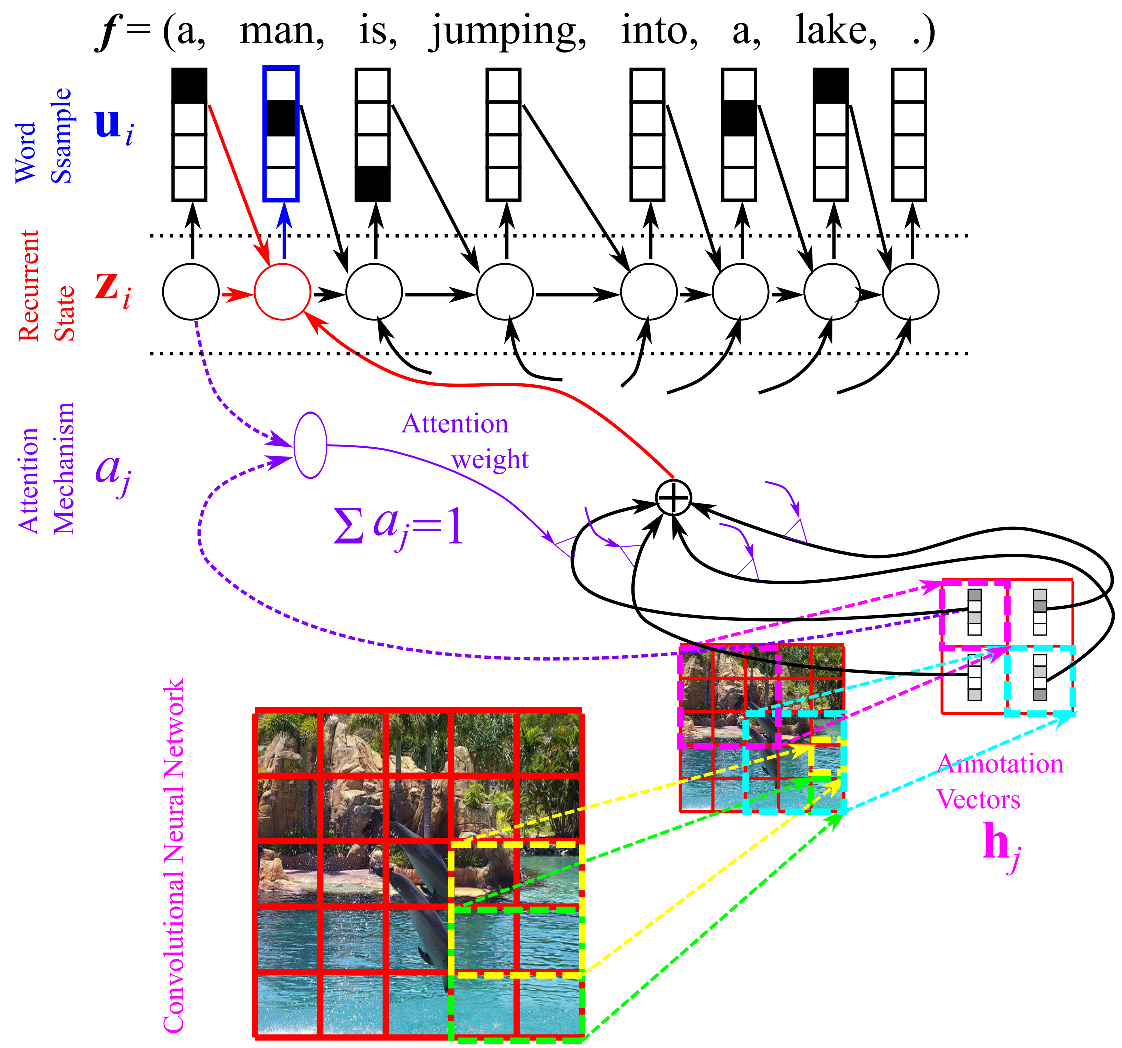


Image-to-Text: Caption Generation with Attention

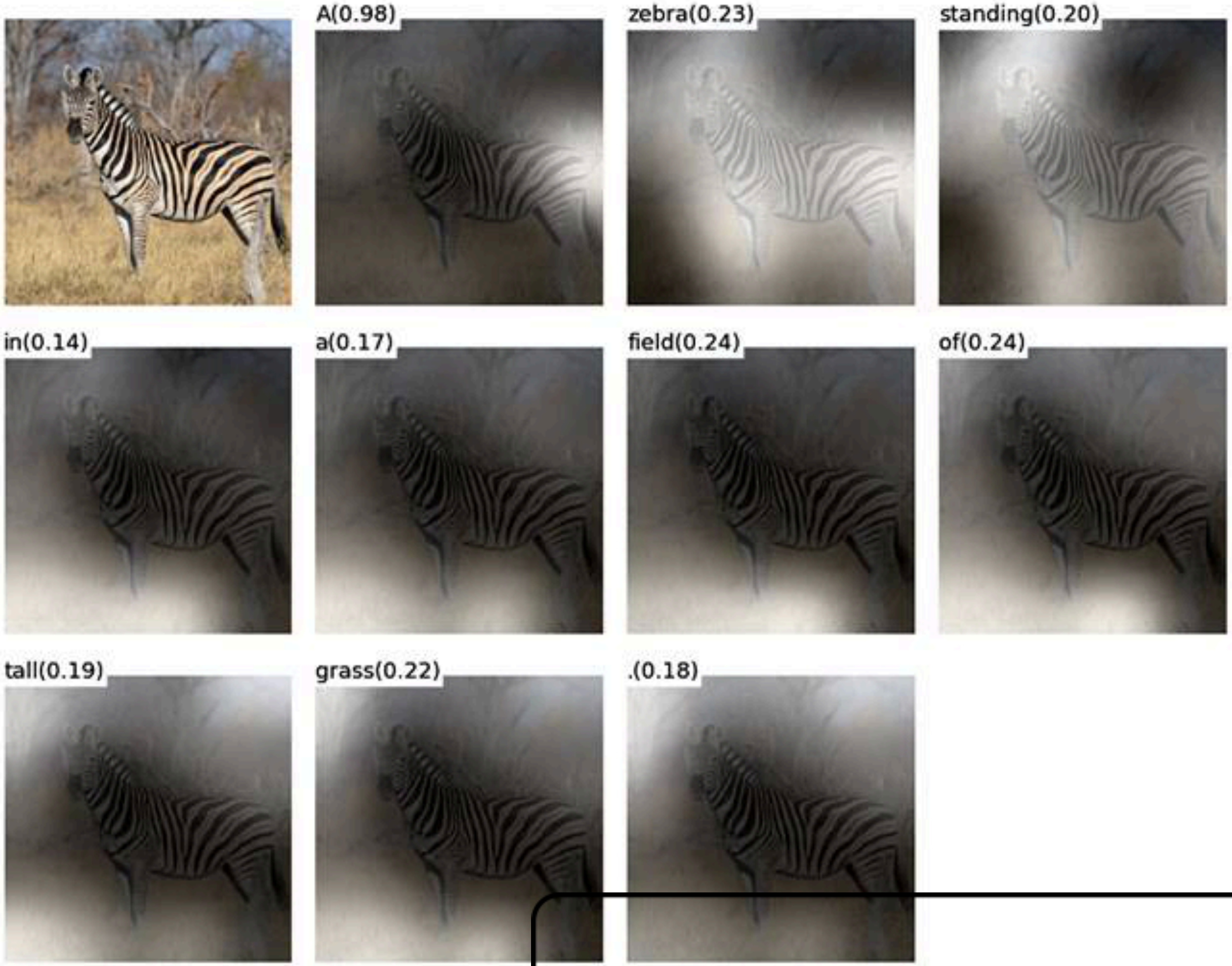
(Xu et al, ICML 2015)

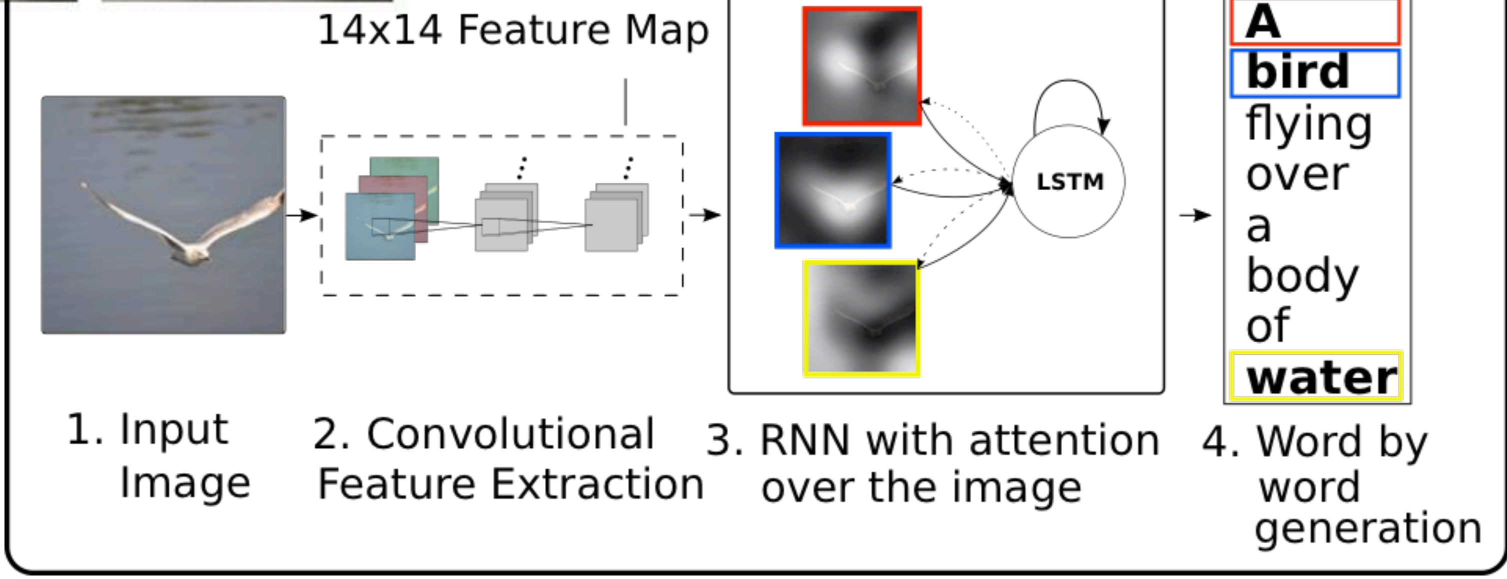


Following many papers on caption generation, including *(Kiros et al 2014; Mao et al 2014; Vinyals et al 2014; Donahue et al 2014; Karpathy & Li 2014; Fang et al 2014)*

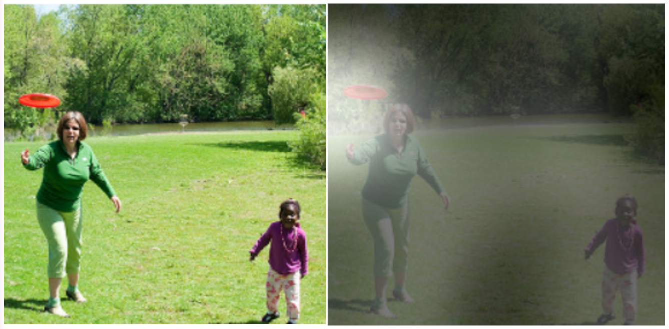
(Xu et al., 2015), (Yao et al., 2015)

Paying Attention to Selected Parts of the Image While Uttering Words

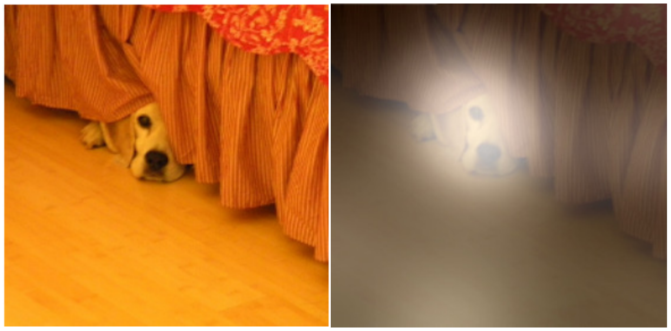



14x14 Feature Map
LSTM
A
bird
flying
over
a
body
of
water
1. Input Image
2. Convolutional Feature Extraction
3. RNN with attention over the image
4. Word by word generation

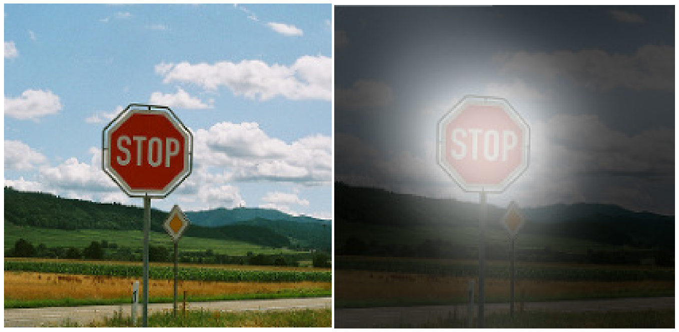
The Good



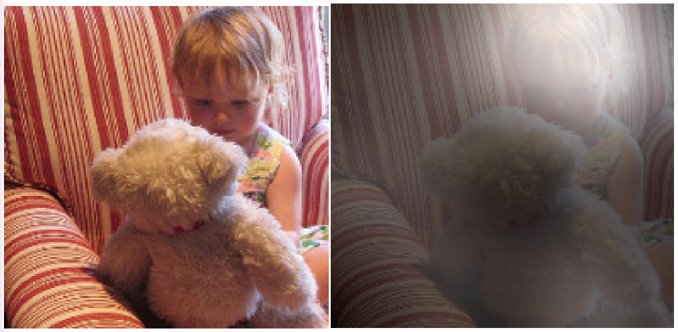
A woman is throwing a frisbee in a park.



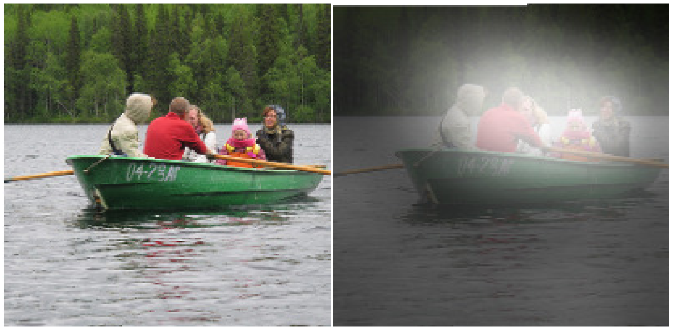
A dog is standing on a hardwood floor.



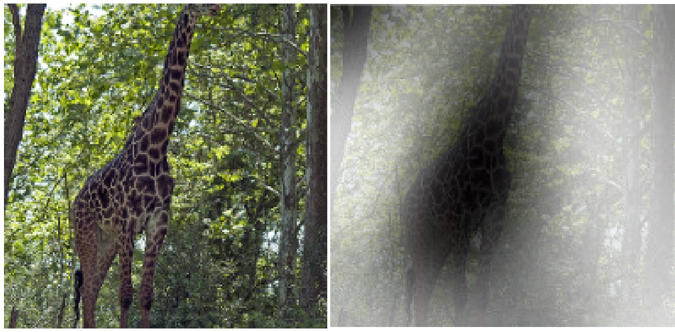
A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.

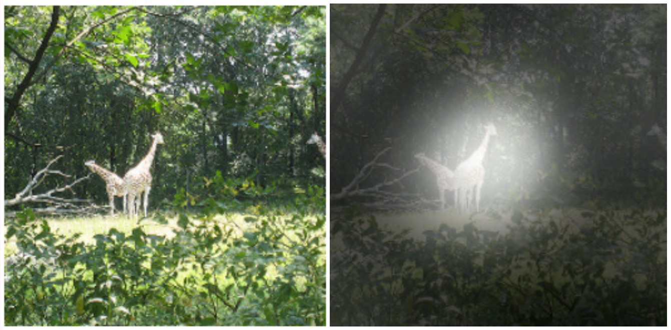


A group of people sitting on a boat in the water.

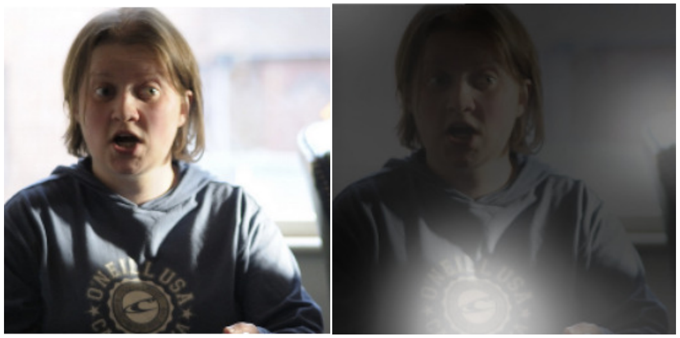


A giraffe standing in a forest with trees in the background.

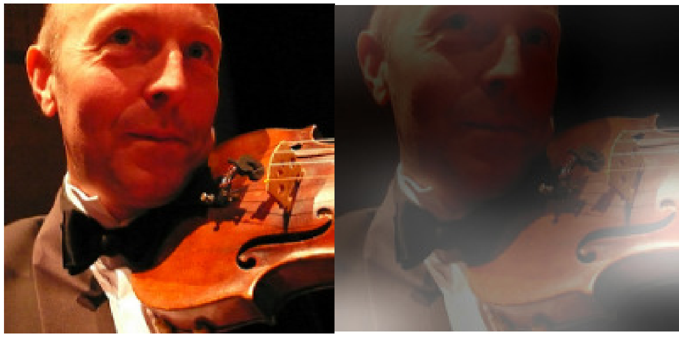
And the Bad



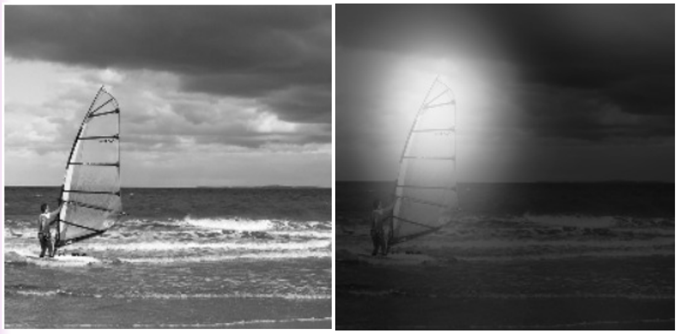
A large white bird standing in a forest.



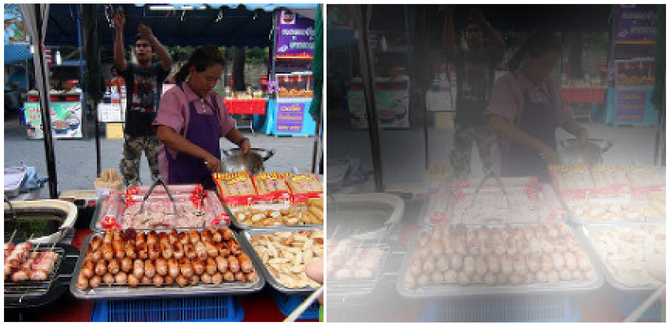
A woman holding a clock in her hand.



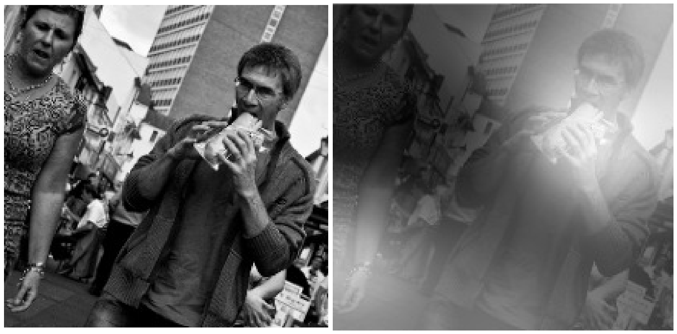
A man wearing a hat and
a hat on a skateboard.



A person is standing on a beach
with a surfboard.



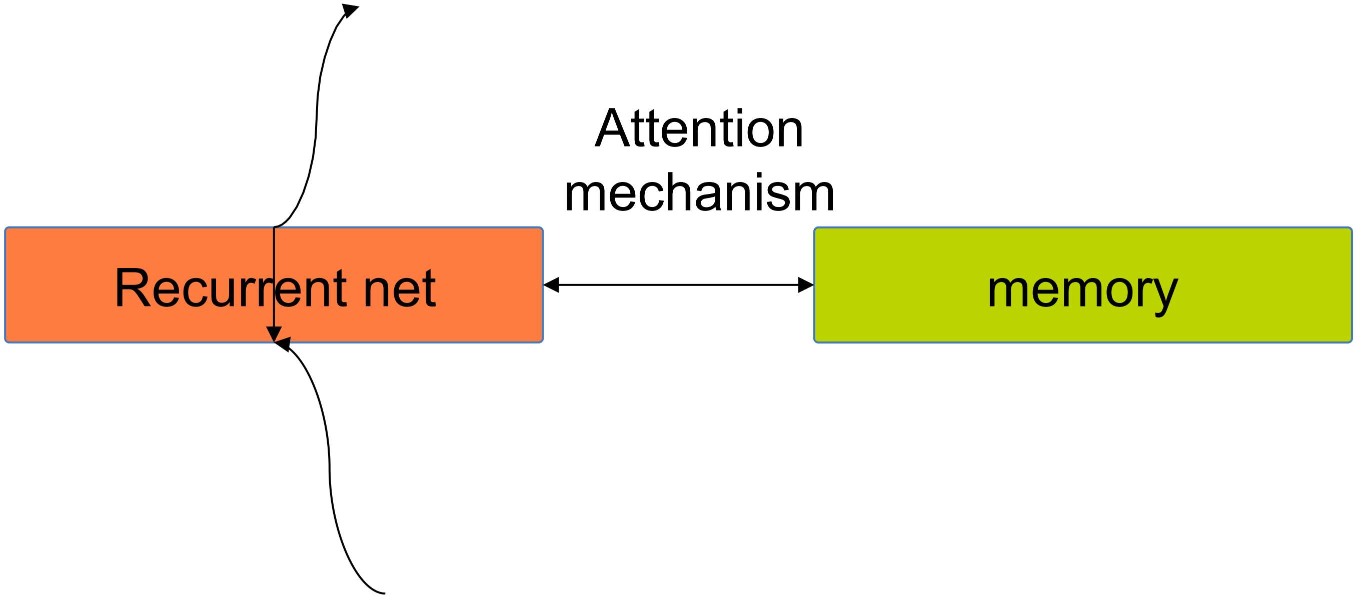
A woman is sitting at a table
with a large pizza.



A man is talking on his cell phone
while another man watches.

But How can Neural Nets Remember Things?

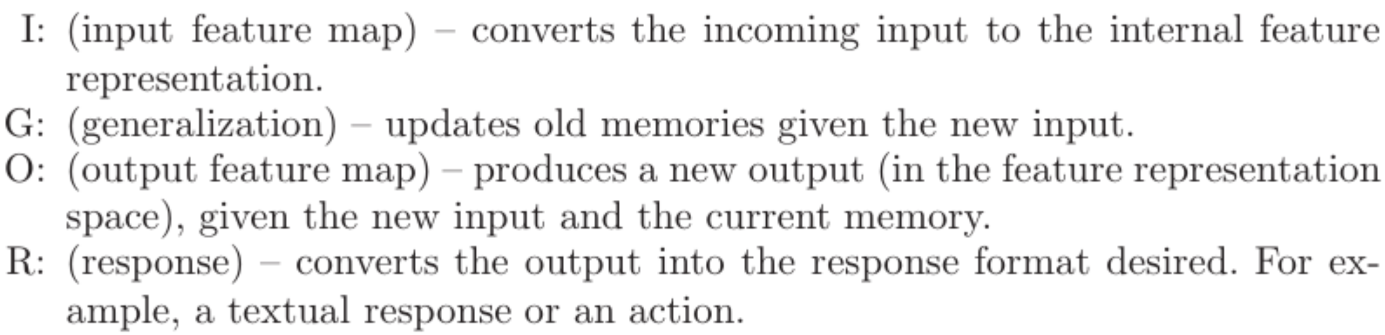
- Recurrent networks cannot remember things for very long
  - The cortex only remember things for 20 seconds
- We need a “hippocampus” (a separate memory module)
  - LSTM [Hochreiter 1997], registers
  - **Memory networks** [Weston et 2014] (FAIR), associative memory
  - NTM [Graves et al. 2014], “tape”.



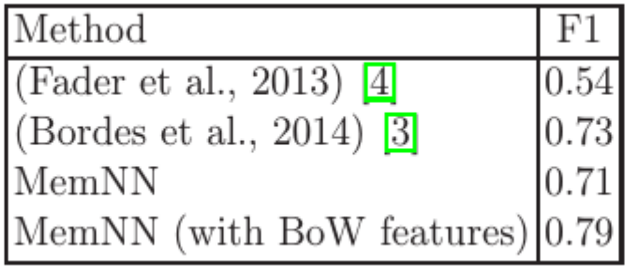
Memory Networks Enable REASONING

- **Add a short-term memory to a network**

http://arxiv.org/abs/1410.3916

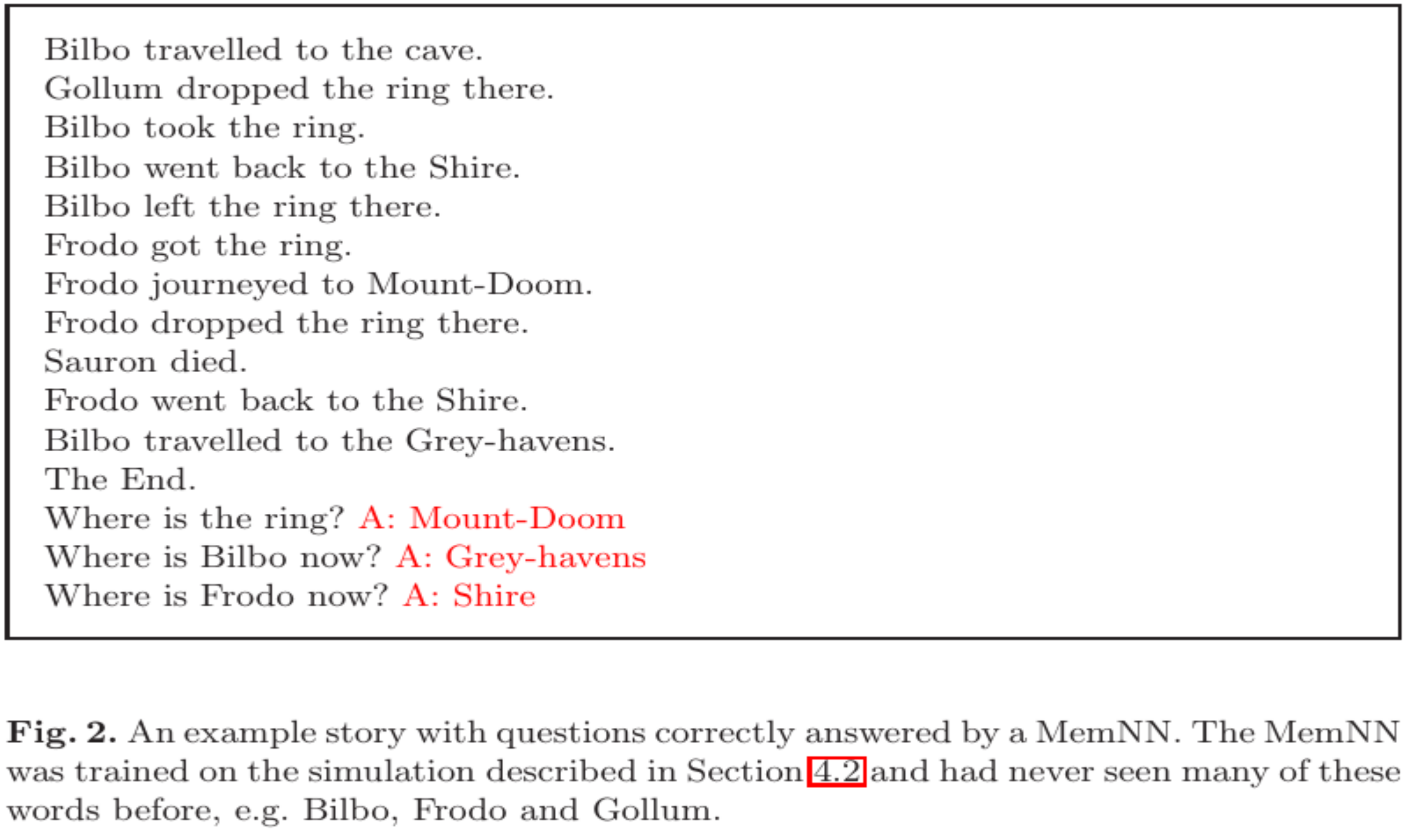


I: (input feature map) – converts the incoming input to the internal feature representation.
G: (generalization) – updates old memories given the new input.
O: (output feature map) – produces a new output (in the feature representation space), given the new input and the current memory.
R: (response) – converts the output into the response format desired. For example, a textual response or an action.



Method	F1
(Fader et al., 2013) [4]	0.54
(Bordes et al., 2014) [3]	0.73
MemNN	0.71
MemNN (with BoW features)	0.79

Results on Question Answering Task



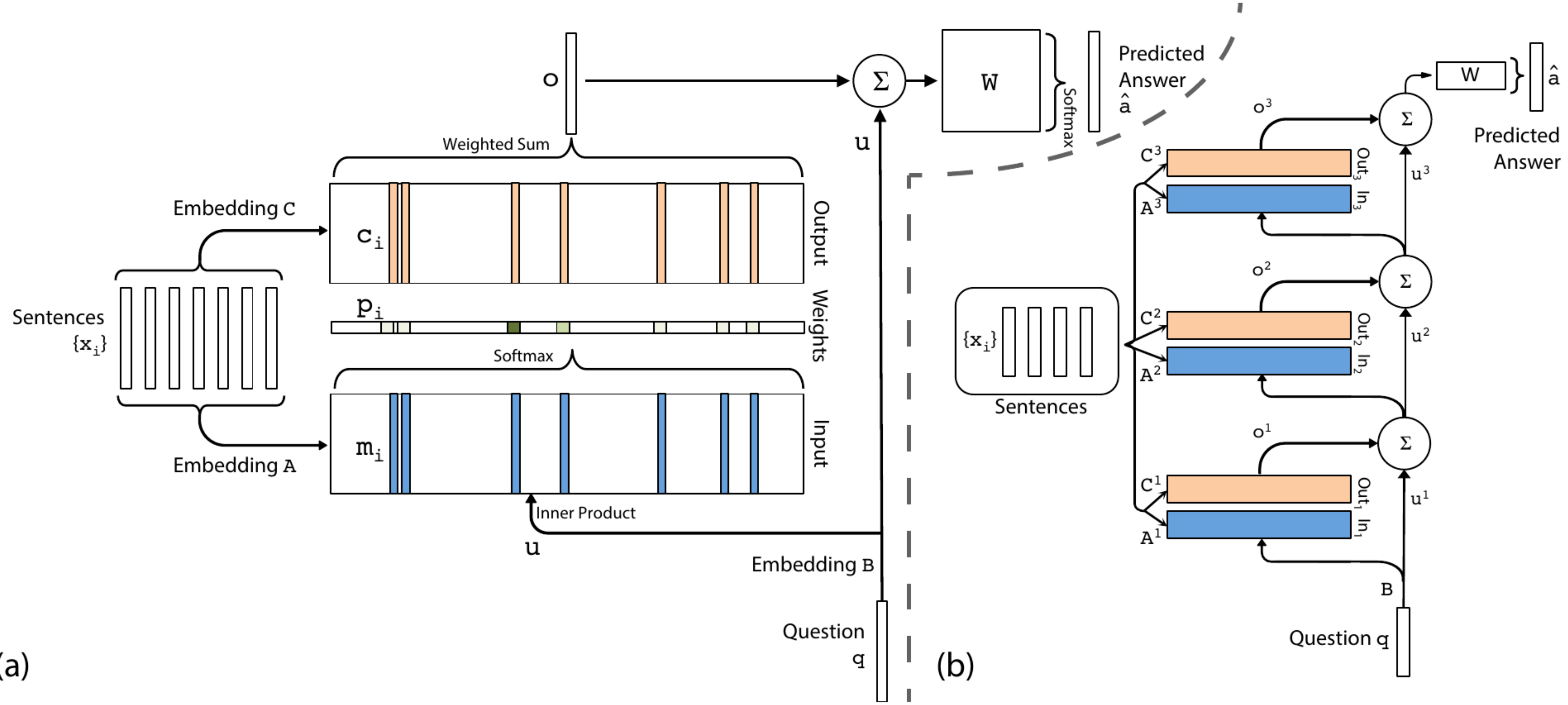
Bilbo travelled to the cave.
Gollum dropped the ring there.
Bilbo took the ring.
Bilbo went back to the Shire.
Bilbo left the ring there.
Frodo got the ring.
Frodo journeyed to Mount-Doom.
Frodo dropped the ring there.
Sauron died.
Frodo went back to the Shire.
Bilbo travelled to the Grey-havens.
The End.
Where is the ring? A: Mount-Doom
Where is Bilbo now? A: Grey-havens
Where is Frodo now? A: Shire

Fig. 2. An example story with questions correctly answered by a MemNN. The MemNN was trained on the simulation described in Section 4.2 and had never seen many of these words before, e.g. Bilbo, Frodo and Gollum.

(Weston, Chopra, Bordes 2014)

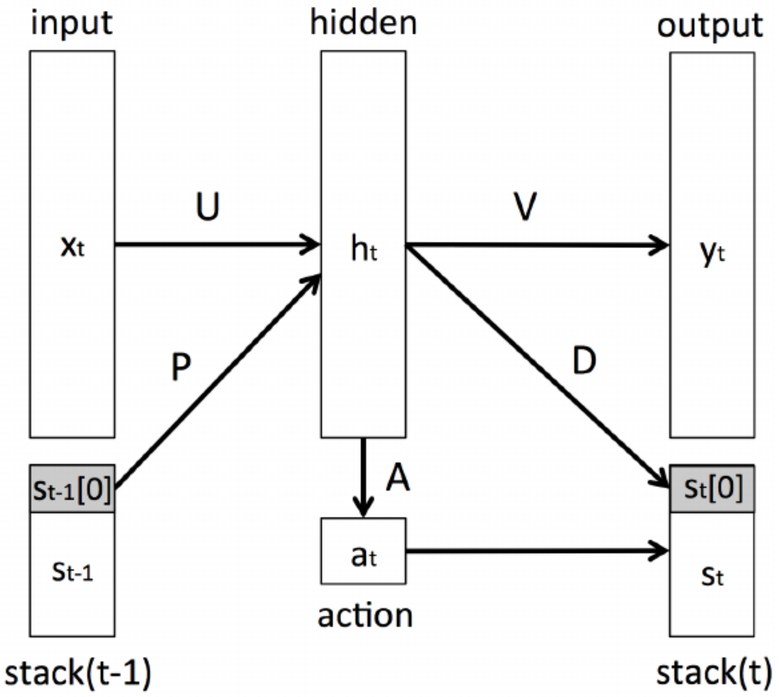
End-to-End Memory Network

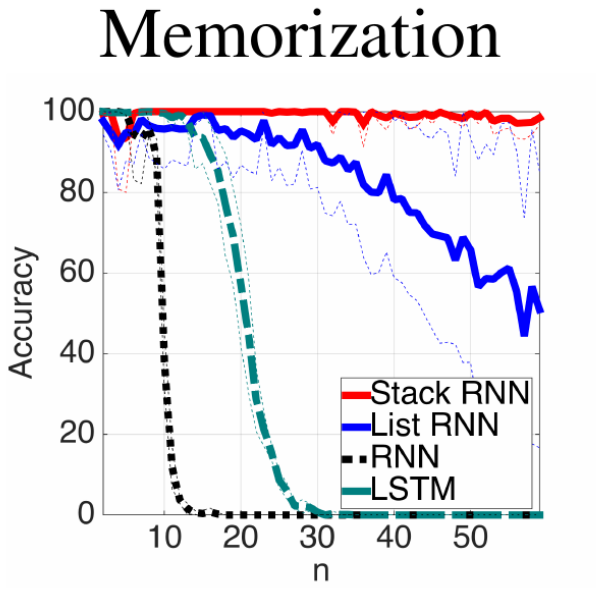
- [Sukhbataar, Szlam, Weston, Fergus NIPS 2015, ArXiv:1503.08895]
- Weakly-supervised MemNN: no need to tell which memory location to use.

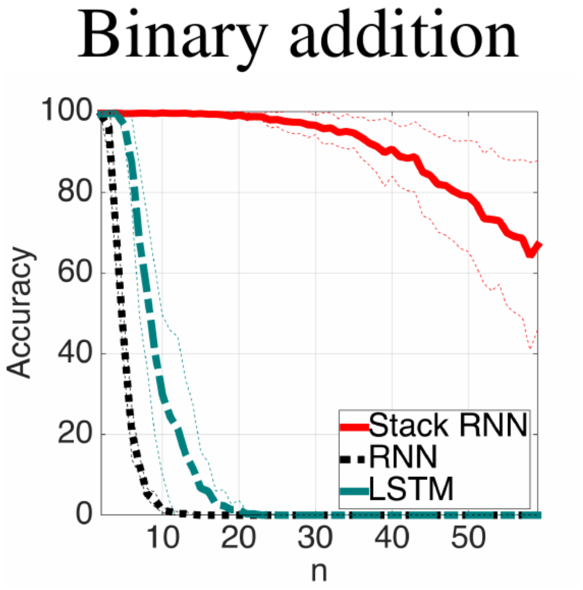


Stack-Augmented RNN: learning “algorithmic” sequences

- [Joulin & Mikolov, ArXiv:1503.01007]



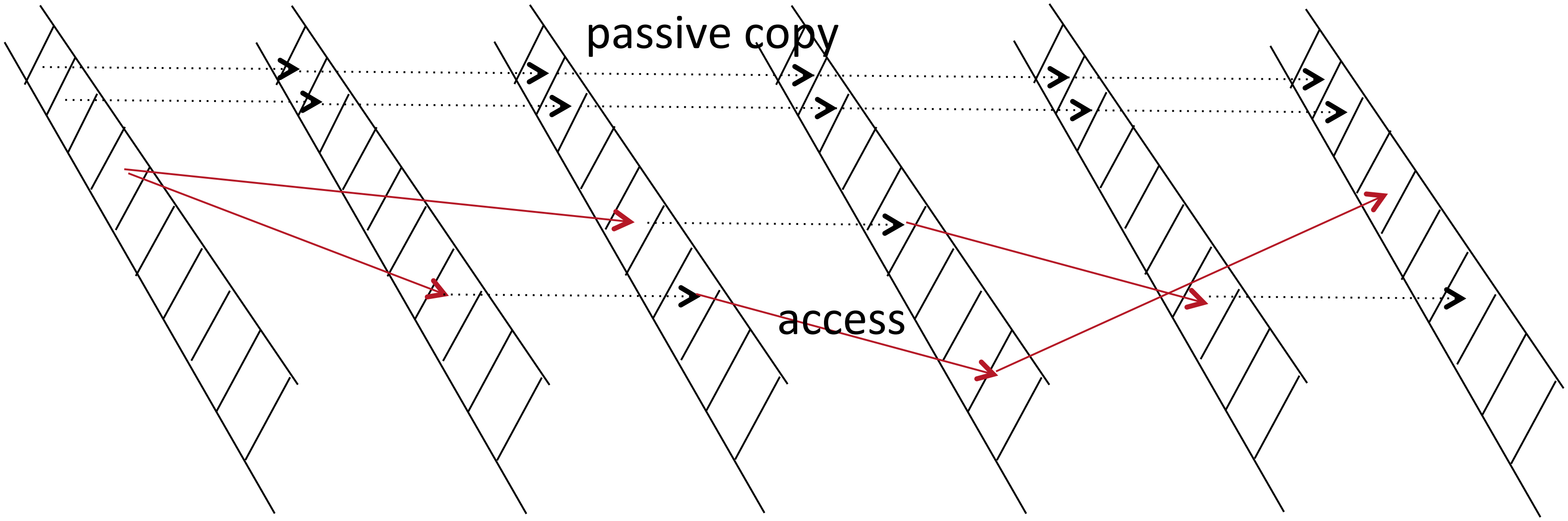




method	a^nb^n	$a^nb^nc^n$	$a^nb^nc^nd^n$	a^nb^{2n}	$a^nb^mc^{n+m}$
RNN	25%	23.3%	13.3%	23.3%	33.3%
LSTM	100%	100%	68.3%	75%	100%
List RNN 40+5	100%	33.3%	100%	100%	100%
Stack RNN 40+10	100%	100%	100%	100%	43.3%
Stack RNN 40+10 + rounding	100%	100%	100%	100%	100%

Sparse Access Memory for Long-Term Dependencies

- A mental state stored in an external memory can stay for arbitrarily long durations, until evoked for read or write
- Forgetting = vanishing gradient.
- Memory = larger state, reducing the need for forgetting/vanishing



How do humans generalize from very few examples?

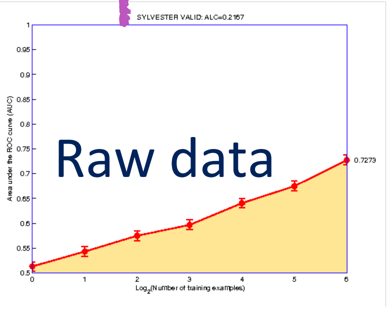
- They **transfer** knowledge from previous learning:
  - Representations
  - Explanatory factors

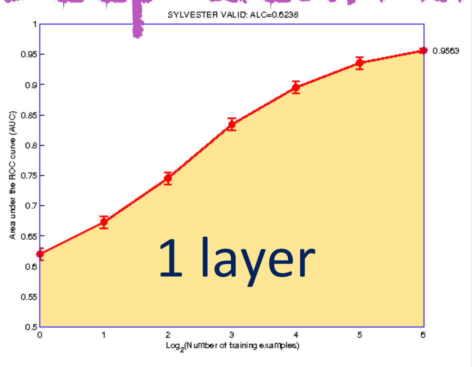
- Previous learning from: unlabeled data

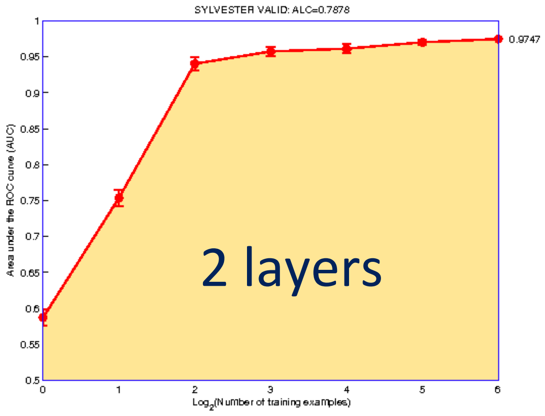
  + labels for other tasks

- **Prior: shared underlying explanatory factors, in particular between P(x) and P(Y|x)**

Unsupervised and Transfer Learning Challenge + Transfer Learning Challenge: Won by Unsupervised Deep Learning

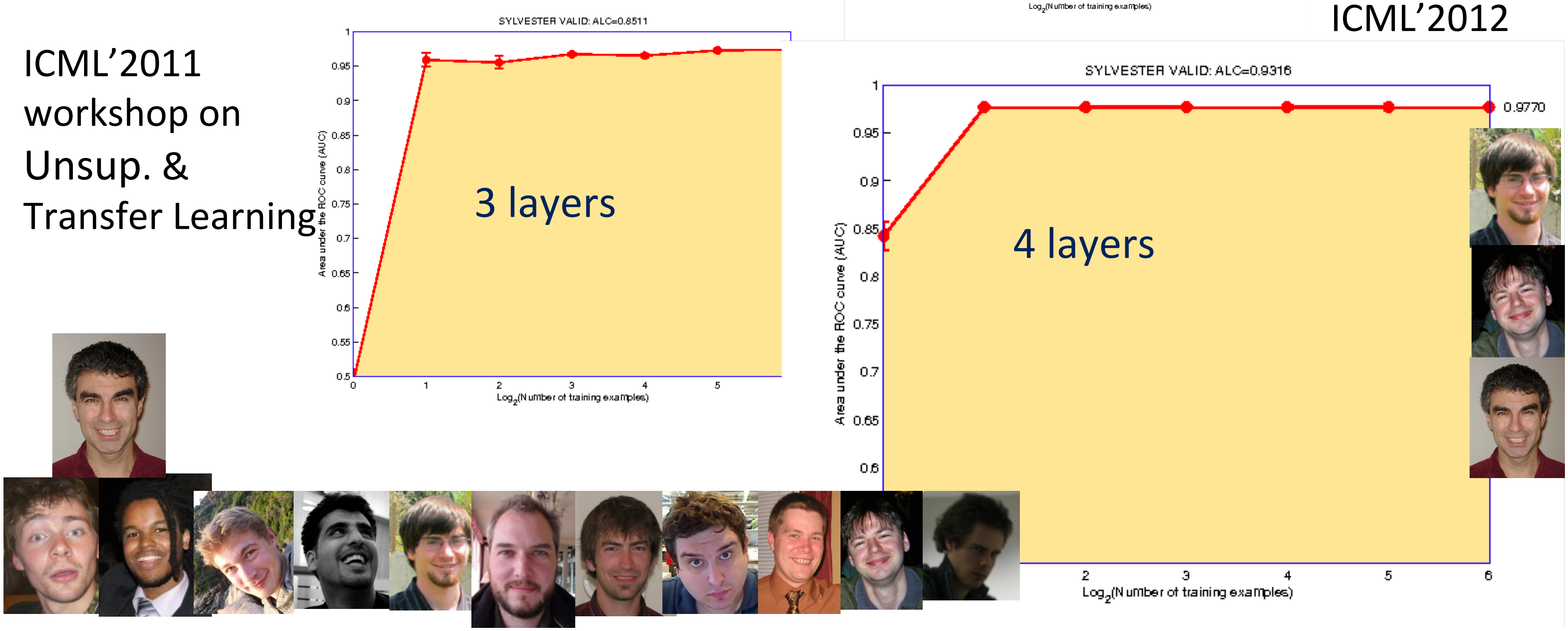






NIPS'2011 Transfer Learning Challenge
Paper: ICML'2012

ICML'2011 workshop on Unsup. & Transfer Learning



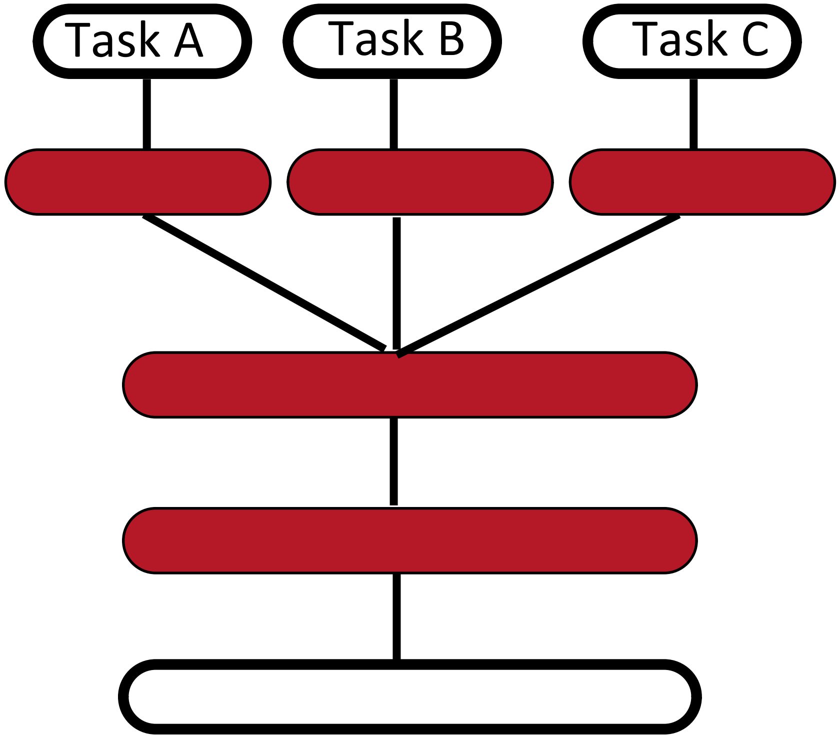
Multi-Task Learning

- Generalizing better to new tasks (tens of thousands!) is crucial to approach AI
- Example: speech recognition, sharing across multiple languages

- Deep architectures learn good intermediate representations that can be shared across tasks

(Collobert & Weston ICML 2008, Bengio et al AISTATS 2011)

- Good representations that disentangle underlying factors of variation make sense for many tasks because **each task concerns a subset of the factors**



E.g. dictionary, with intermediate concepts re-used across many definitions

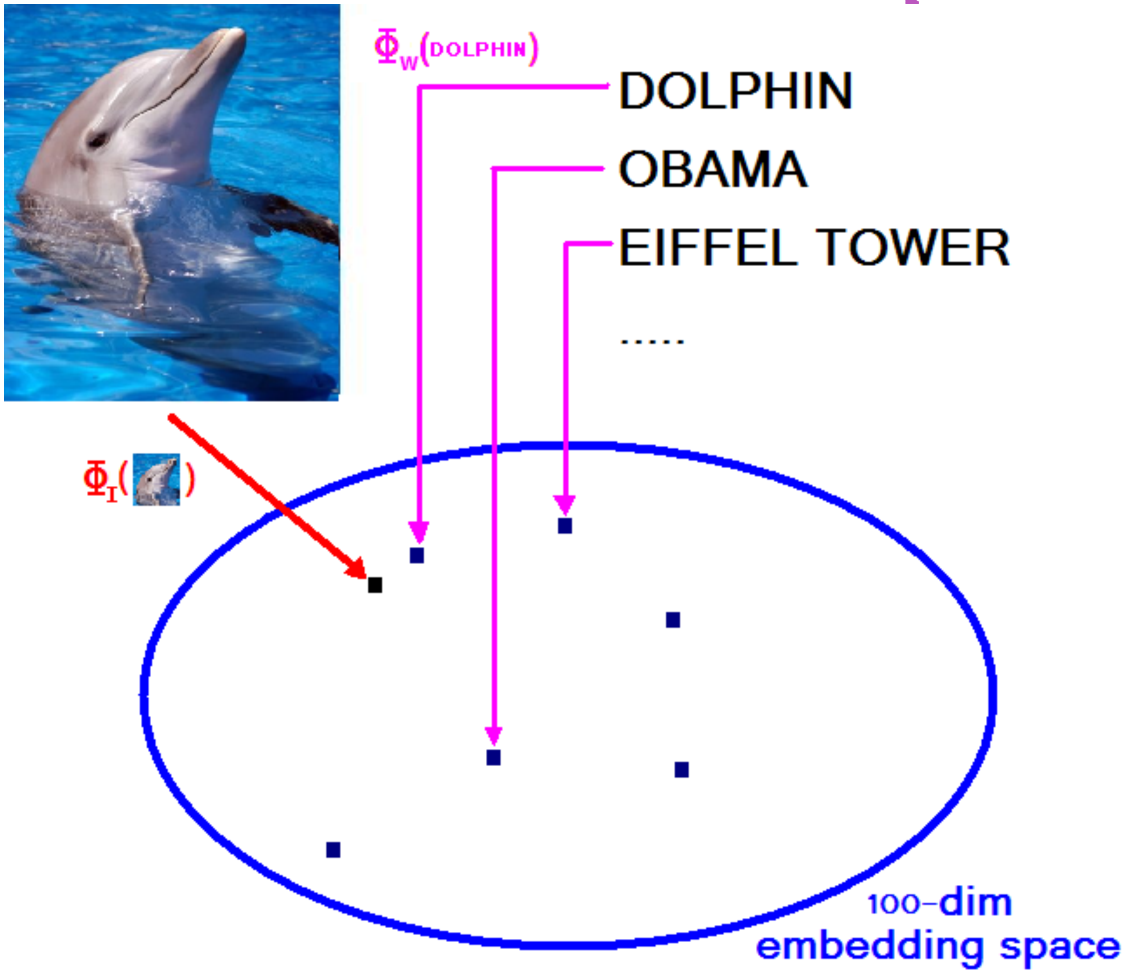
Prior: shared underlying explanatory factors between tasks

Google Image Search Joint Embedding: different object types represented in same space

Google:

S. Bengio, J. Weston & N. Usunier

(IJCAI 2011, NIPS’2010, JMLR 2010, ML J 2010)

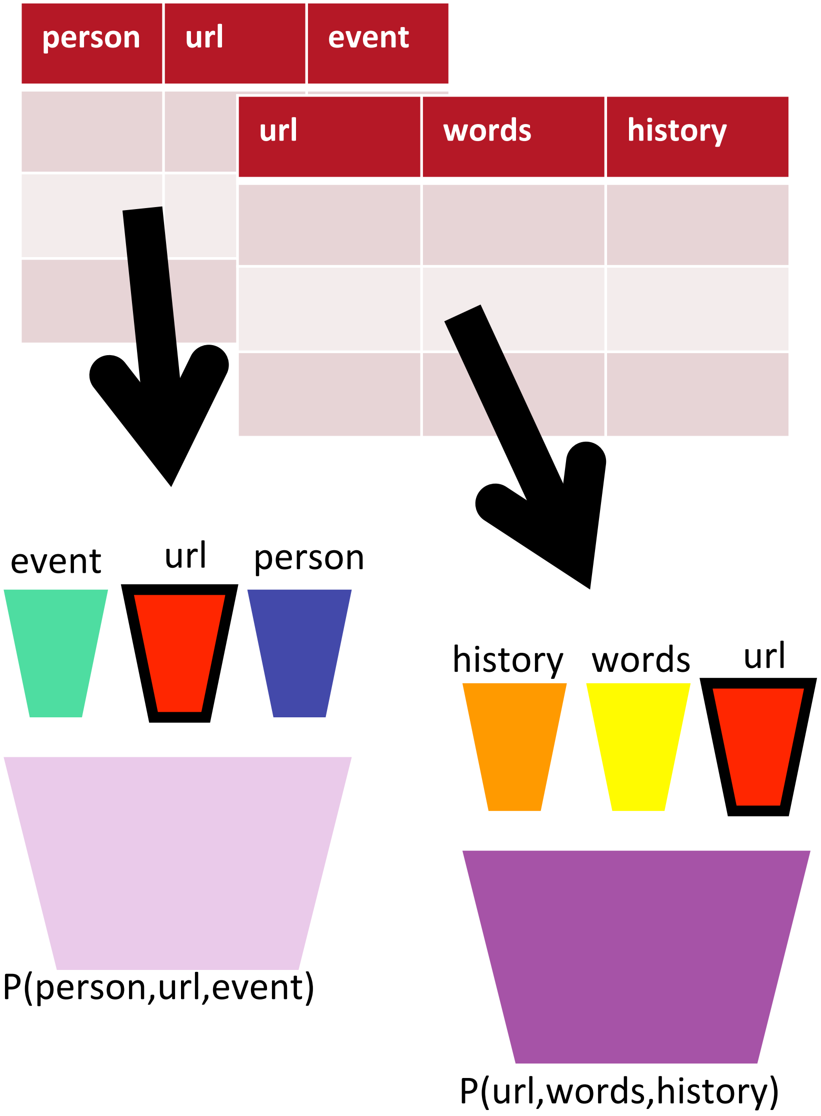


WSABIE objective function:

Learn $\Phi_I(\cdot)$ ***and*** $\Phi_W(\cdot)$ ***to optimize precision@k.***

Combining Multiple Sources of Evidence with Shared Representations

- Traditional ML: data = matrix
- Relational learning: multiple sources, different tuples of variables
- Share representations of same types across data sources
- Shared learned representations help propagate information among data sources: e.g., WordNet, XWN, Wikipedia, **FreeBase**, ImageNet… (Bordes et al AISTATS 2012, ML J. 2013)
- **FACTS = DATA**
- **Deduction = Generalization**



person url event

url words history

event url person

P(person,url,event)

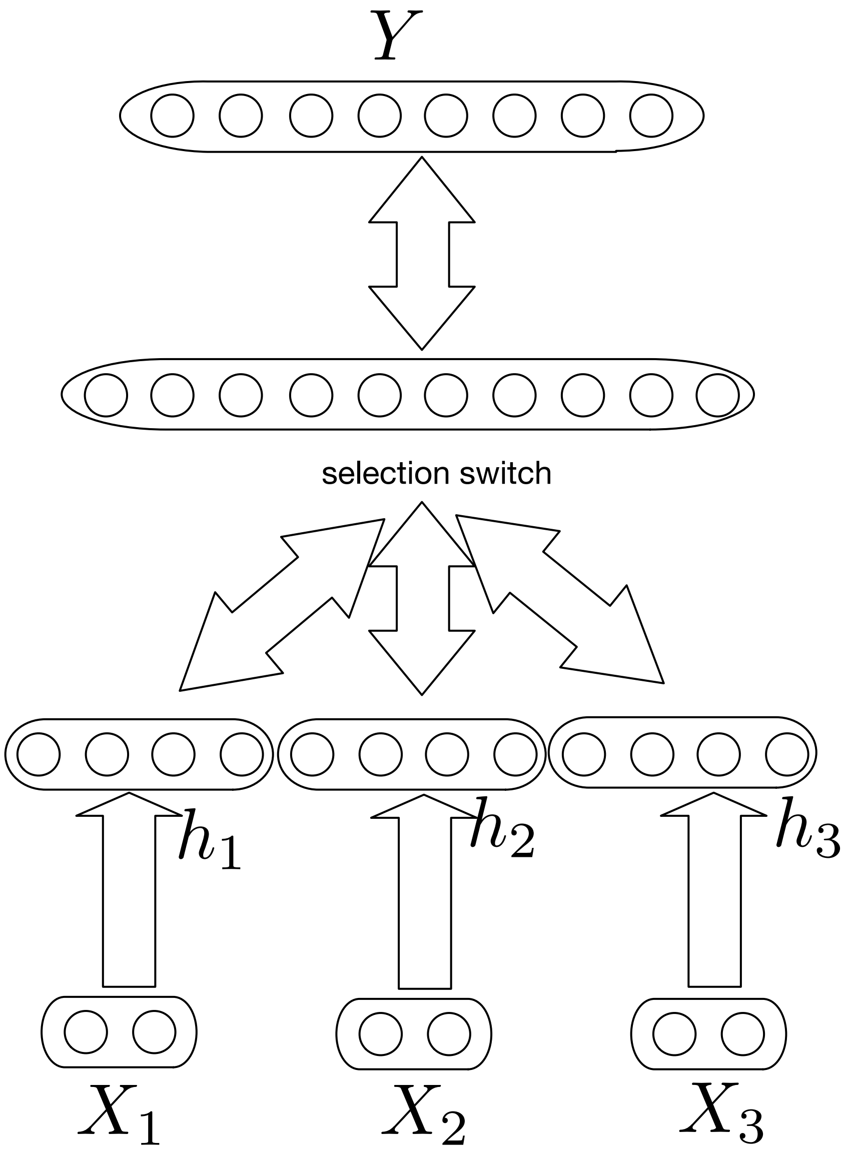
history words url

P(url,words,history)

Multi-Task / Multimodal Learning with Different Inputs for Different Tasks

E.g. speaker adaptation, multimodal input…

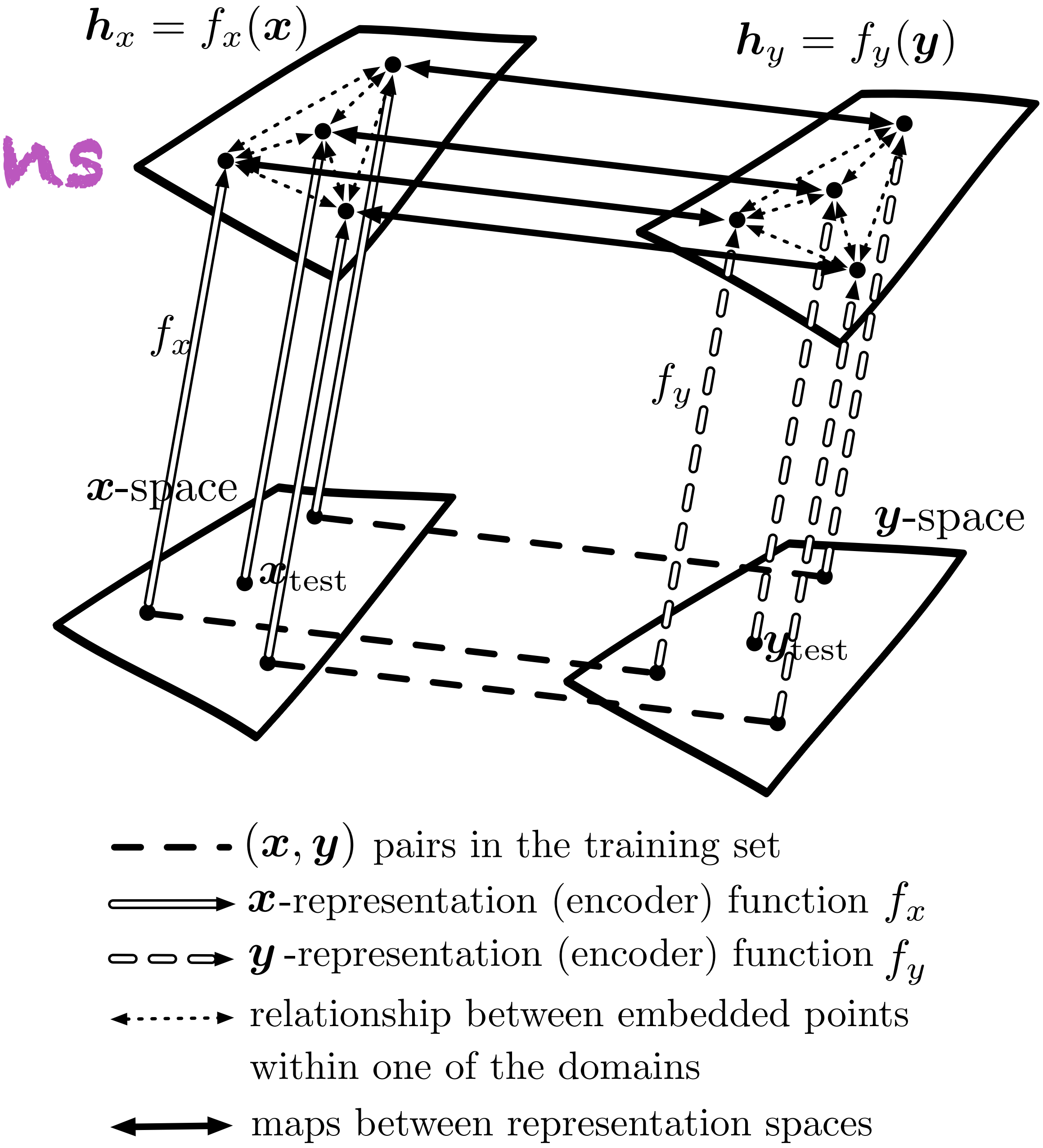
Unsupervised multimodal case:
(Srivastava & Salakhutdinov NIPS 2012)



Maps Between Representations

x and *y* represent different modalities, e.g., image, text, sound…

Can provide 0-shot generalization to new categories (values of *y*)


$\boldsymbol{h}_x = f_x(\boldsymbol{x})$
$\boldsymbol{h}_y = f_y(\boldsymbol{y})$
f_x
f_y
$\boldsymbol{x}$-space
$\boldsymbol{y}$-space
$\boldsymbol{x}_{\text{test}}$
$\boldsymbol{y}_{\text{test}}$
$(\boldsymbol{x}, \boldsymbol{y})$ pairs in the training set
$\boldsymbol{x}$-representation (encoder) function f_x
$\boldsymbol{y}$-representation (encoder) function f_y
relationship between embedded points within one of the domains
maps between representation spaces

Unsupervised Representation Learning

Why Unsupervised Learning?

- Recent progress mostly in supervised DL
- Real challenges for unsupervised DL
- Potential benefits:
  - **Exploit tons of unlabeled data**
  - Answer new questions about the variables observed
  - Regularizer – transfer learning – domain adaptation
  - Easier optimization (divide and conquer)
  - Joint (structured) outputs

Why Latent Factors & Unsupervised Representation Learning? Because of Causality.

On causal and anticausal learning, (Janzing et al ICML 2012)

- If Ys of interest are among the causal factors of X, then

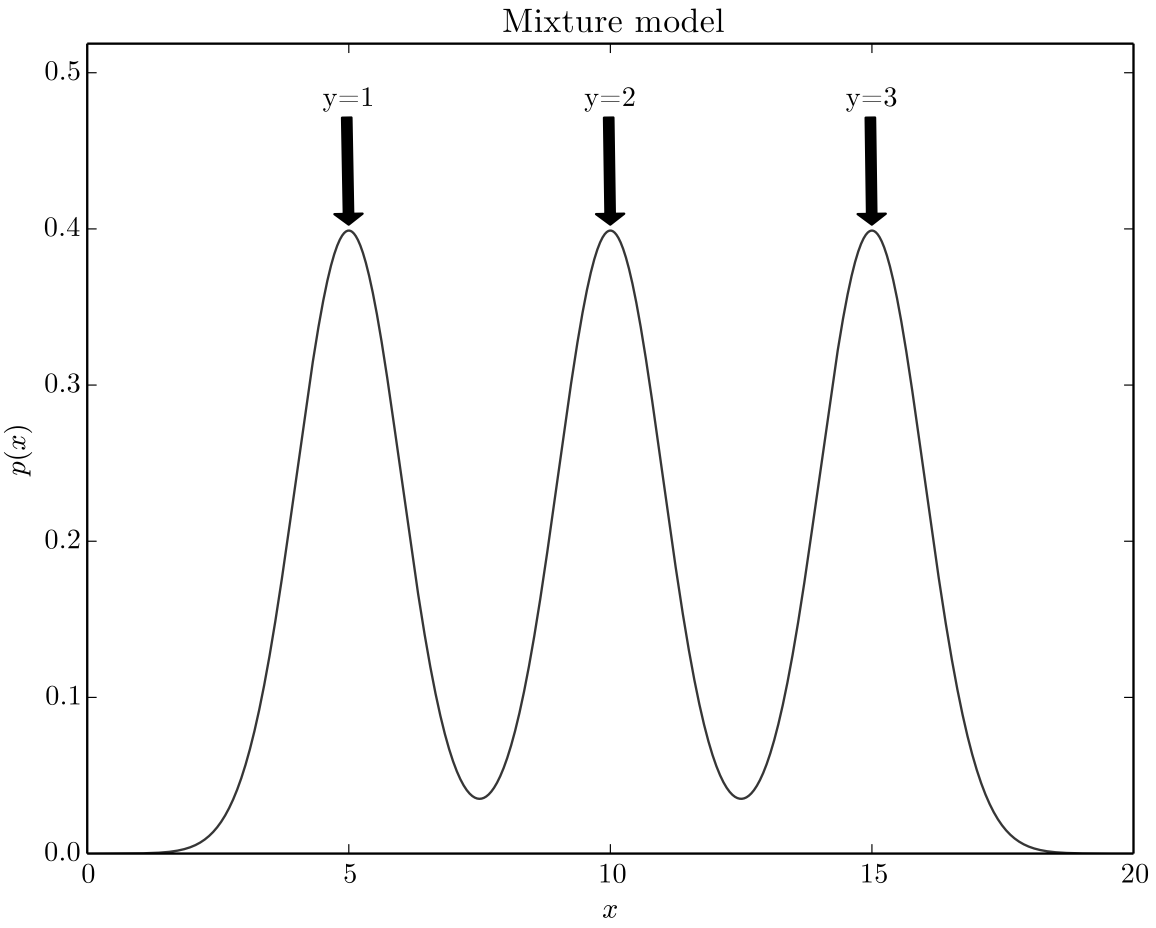
$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

is tied to P(X) and P(X|Y), and P(X) is defined in terms of P(X|Y), i.e.

- The best possible model of X (unsupervised learning) MUST involve Y as a latent factor, implicitly or explicitly.
- Representation learning SEEKS the latent variables H that explain the variations of X, making it likely to also uncover Y.

If Y is a Cause of X, Semi-Supervised Learning Works

- Just observing the *x*-density reveals the causes *y* (cluster ID)
- After learning *p(x)* as a mixture, a single labeled example per class suffices to learn *p(y|x)*



Invariance & Disentangling Underlying Factors

- Invariant features
- Which invariances?
- Alternative: learning to disentangle factors, i.e. keep all the explanatory factors in the representation
- Good disentangling →
  avoid the curse of dimensionality
- Emerges from representation learning
  (Goodfellow et al. 2009, Glorot et al. 2011)

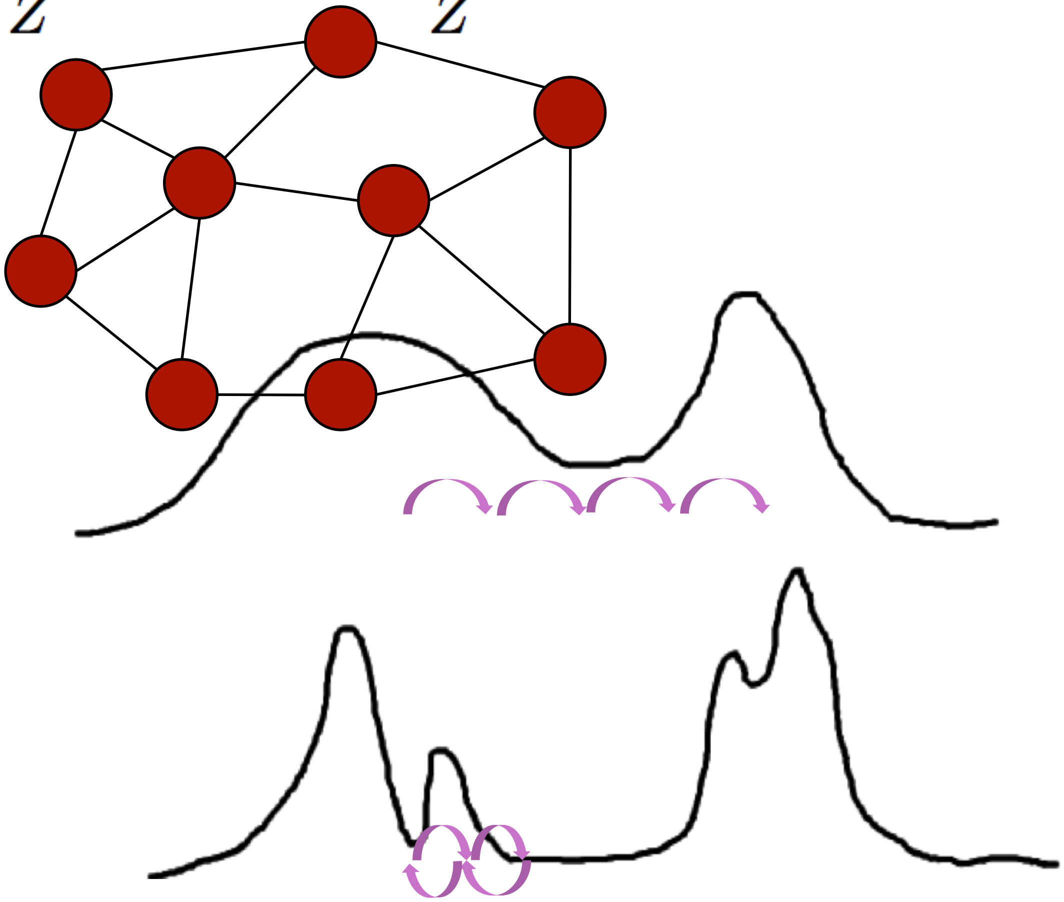
Boltzmann Machines / Undirected Graphical Models

- Boltzmann machines:

(Hinton 84)
$$P(x) = \frac{1}{Z} e^{-\text{Energy}(x)} = \frac{1}{Z} e^{c^T x + x^T W x} = \frac{1}{Z} e^{\sum_i c_i x_i + \sum_{i,j} x_i W_{ij} x_j}$$

- Iterative sampling scheme = stochastic relaxation, Monte-Carlo Markov chain

- Training requires sampling: might take a lot of time to converge if there are well-separated modes

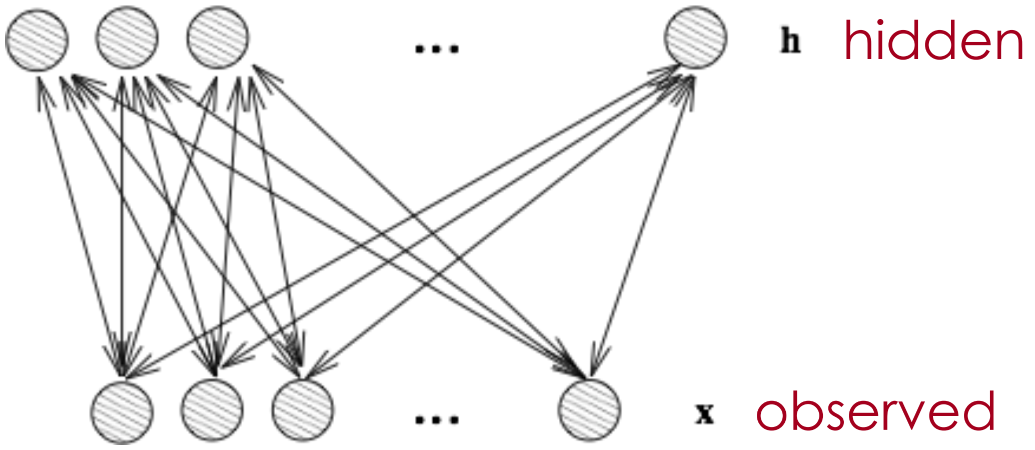


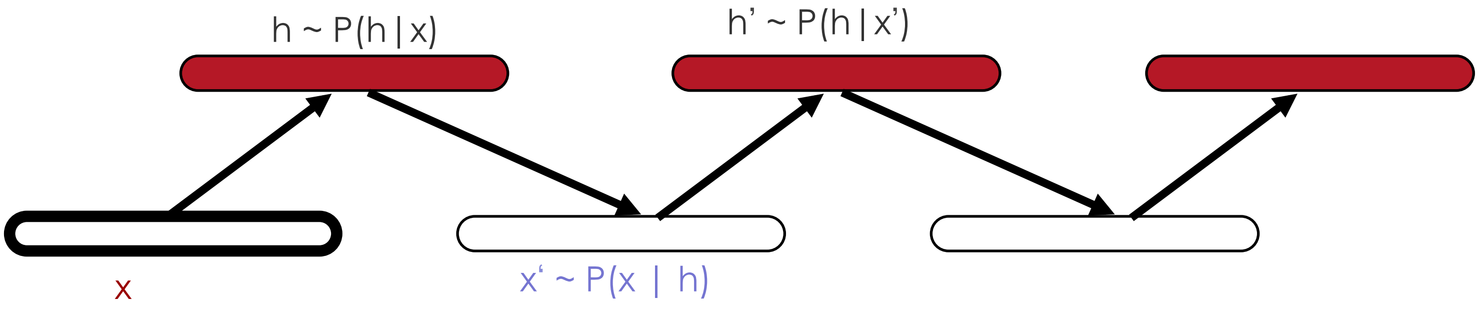
Restricted Boltzmann Machine (RBM)

(Smolensky 1986, Hinton et al 2006)

$$P(x,h) = \frac{1}{Z} e^{b^T h + c^T x + h^T W x} = \frac{1}{Z} e^{\sum_i b_i h_i + \sum_j c_j x_j + \sum_{i,j} h_i W_{ij} x_j}$$

- A building block (single-layer) for deep architectures
- **Bipartite** undirected graphical model





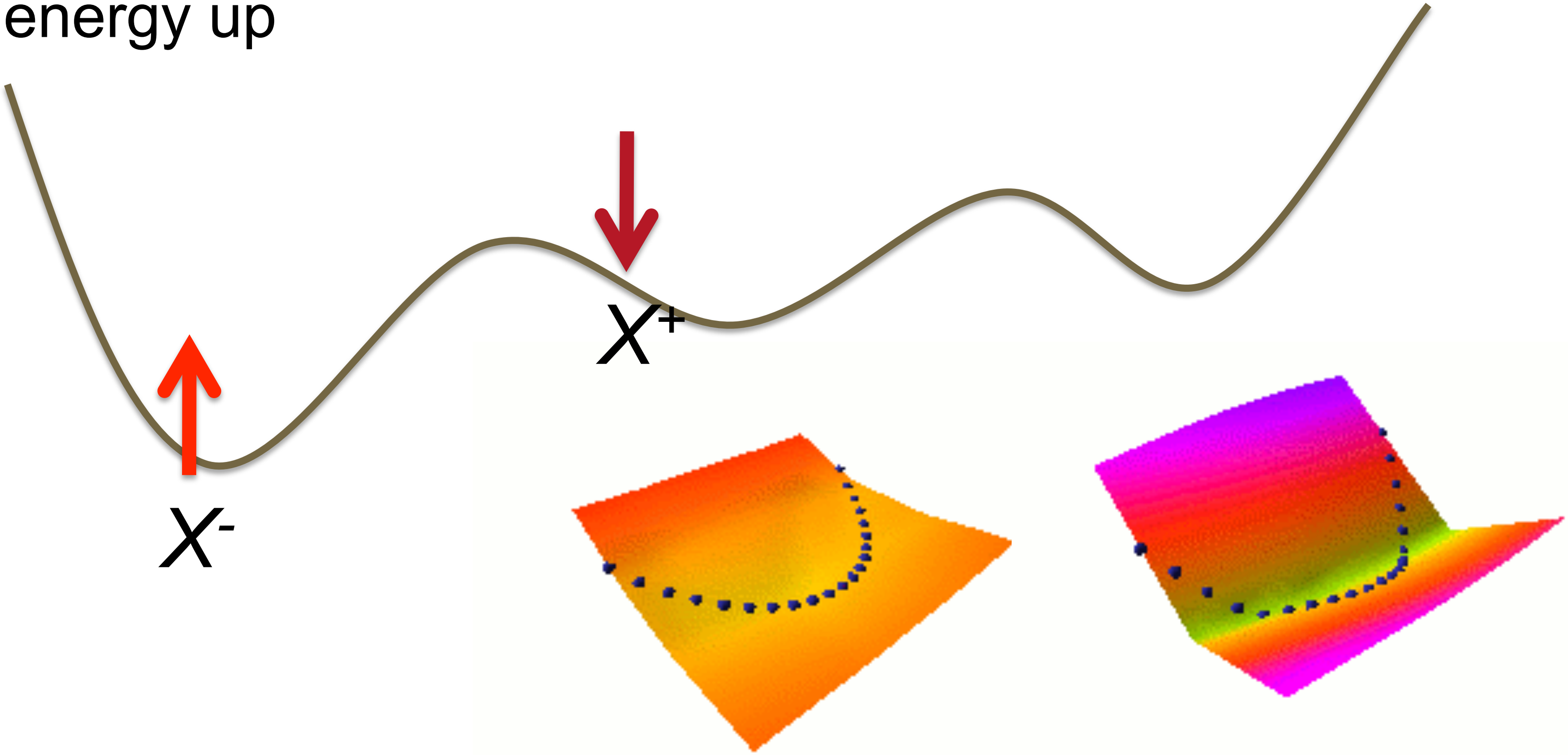
Block Gibbs sampling

Capturing the Shape of the Distribution: Positive & Negative Samples

Boltzmann machines, undirected graphical models, RBMs, energy-based models

$$Pr(x) = \frac{e^{-Energy(x)}}{Z}$$

- Observed (+) examples push the energy down
- Generated / dream / fantasy (-) samples / particles push the energy up



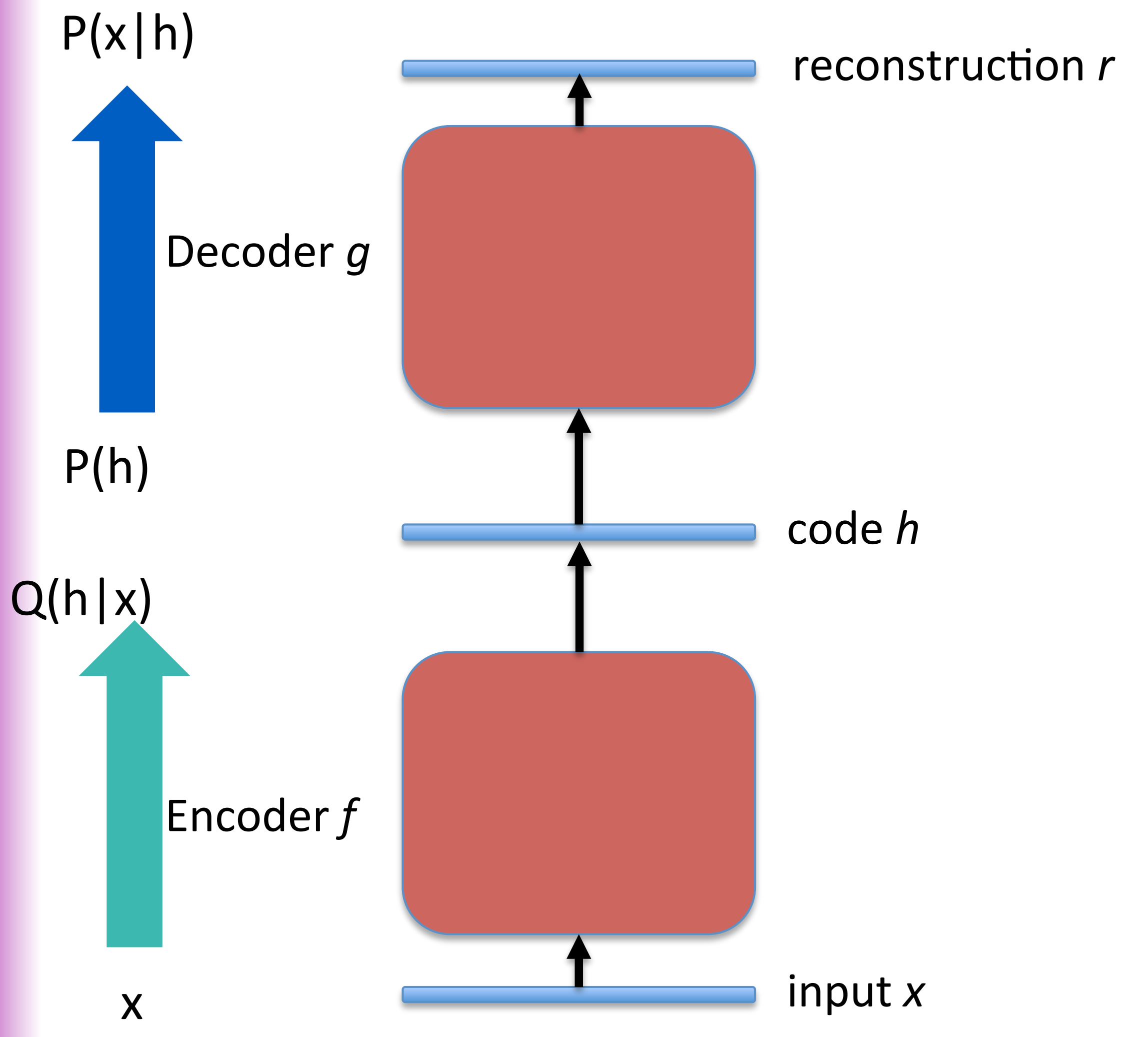
Eight Strategies to Shape the Energy Function

- 1. build the machine so that the volume of low energy stuff is constant
  - PCA, K-means, GMM, square ICA
- 2. push down of the energy of data points, push up everywhere else
  - Max likelihood (needs tractable partition function)
- 3. push down of the energy of data points, push up on chosen locations
  - contrastive divergence, Ratio Matching, Noise Contrastive Estimation, Minimum Probability Flow
- 4. minimize the gradient and maximize the curvature around data points
  - score matching
- 5. train a dynamical system so that the dynamics goes to the manifold
  - denoising auto-encoder, diffusion inversion (nonequilibrium dynamics)
- 6. use a regularizer that limits the volume of space that has low energy
  - Sparse coding, sparse auto-encoder, PSD
- 7. if E(Y) = ||Y - G(Y)||^2, make G(Y) as "constant" as possible.
  - Contracting auto-encoder, saturating auto-encoder
- 8. Adversarial training: generator tries to fool real/synthetic classifier.

Auto-Encoders

- Iterative sampling / undirected models:
  RBM, denoising auto-encoder

- Ancestral sampling / directed models
  Helmholtz machine, VAE, etc.
  (Hinton et al 1995)



Probabilistic reconstruction criterion:
Reconstruction log-likelihood =
- log P(x | h)

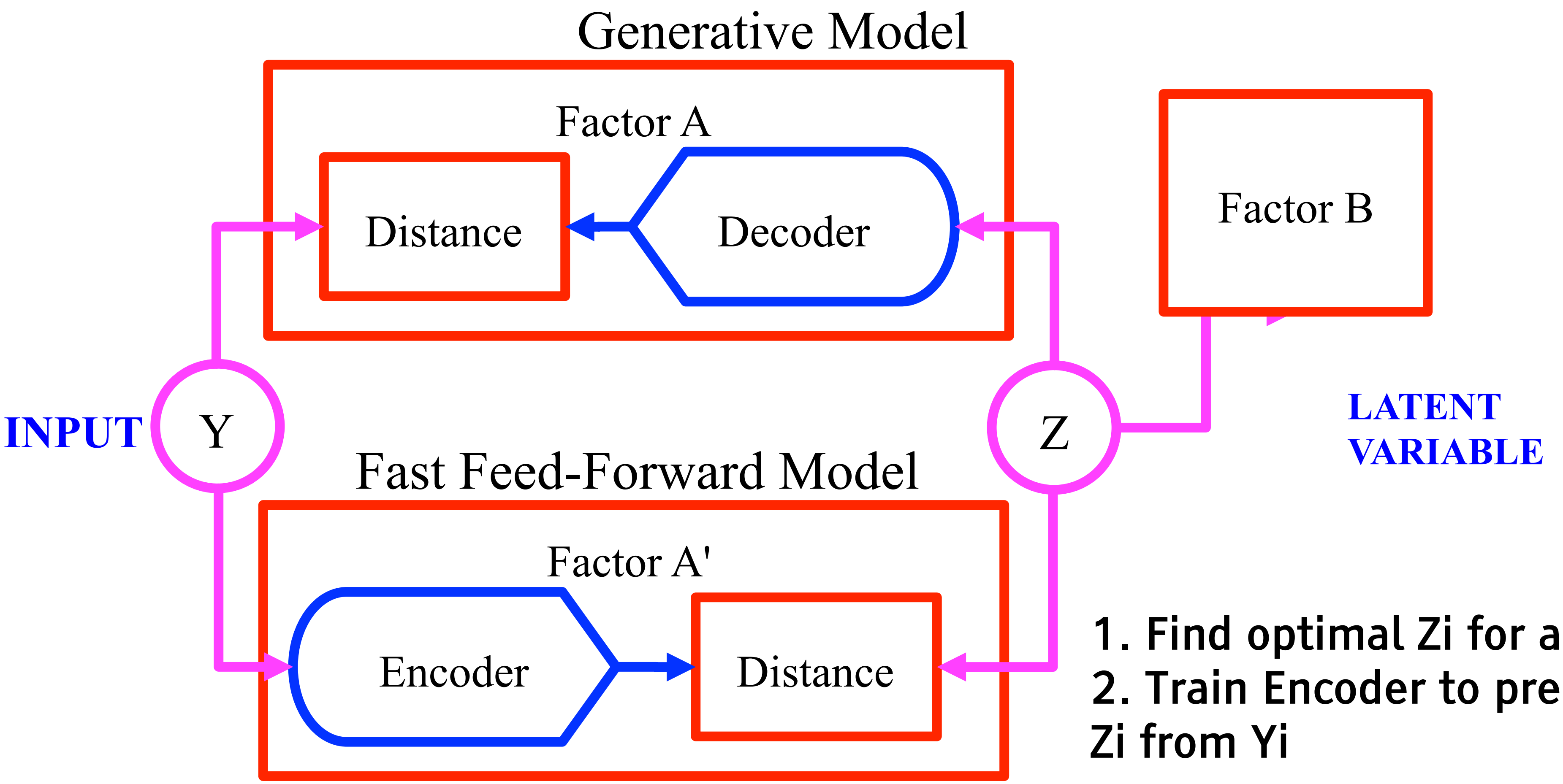
Denoising auto-encoder:
During training, input is corrupted stochastically, and auto-encoder must learn to guess the distribution of the missing information.

Predictive Sparse Decomposition (PSD)

Yann LeCun

[Kavukcuoglu, Ranzato, LeCun, rejected by every conference, 2008-2009]

- Train a “simple” feed-forward function to predict the result of a complex optimization on the data points of interest



1. Find optimal Zi for all Yi;
2. Train Encoder to predict Zi from Yi

Energy = reconstruction_error + code_prediction_error + code_sparsity

Probabilistic interpretation of auto-encoders

- Manifold & probabilistic interpretations of auto-encoders
  - Denoising Score Matching as inductive principle

    (Vincent 2011)
  - Estimating the gradient of the energy function

    (Alain & Bengio ICLR 2013)
  - Sampling via Markov chain

    (Bengio et al NIPS 2013; Sohl-Dickstein et al ICML 2015)
  - Variational auto-encoders

    (Kingma & Welling ICLR 2014)

    (Gregor et al arXiv 2015)

Denoising Auto-Encoder

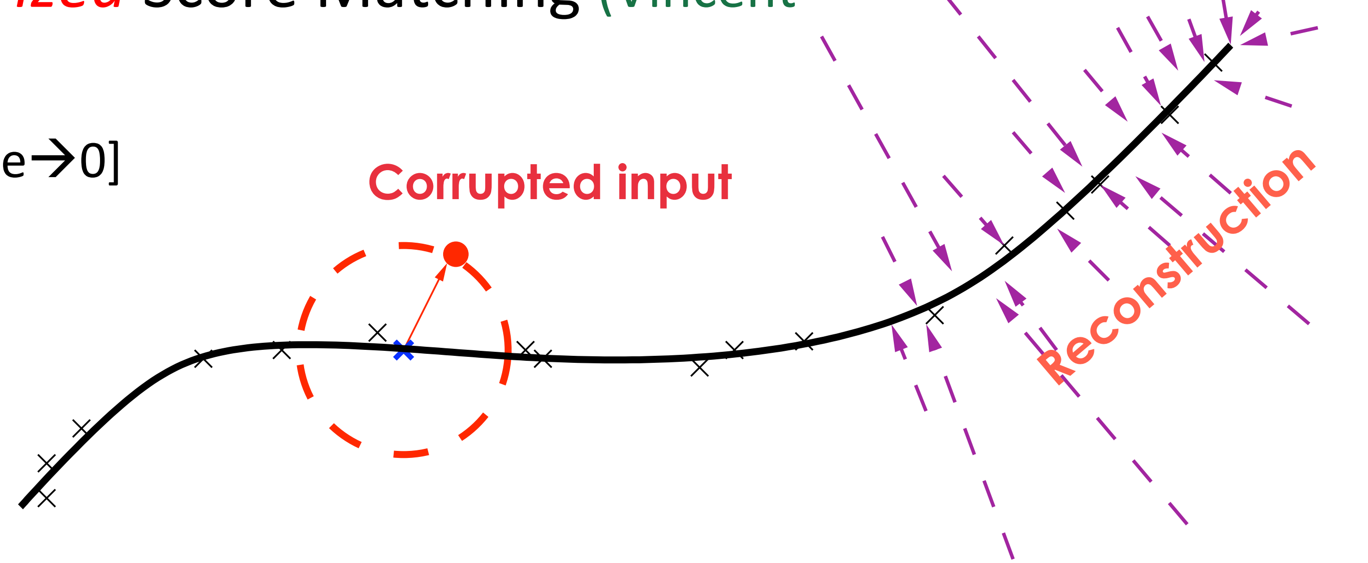
- Learns a vector field pointing towards high probability direction (Alain & Bengio 2013)

$$\text{reconstruction}(x) - x \quad \rightarrow \quad \sigma^2 \frac{\partial \log p(x)}{\partial x}$$

- Some DAEs correspond to a kind of Gaussian RBM with *regularized* Score Matching (Vincent 2011)

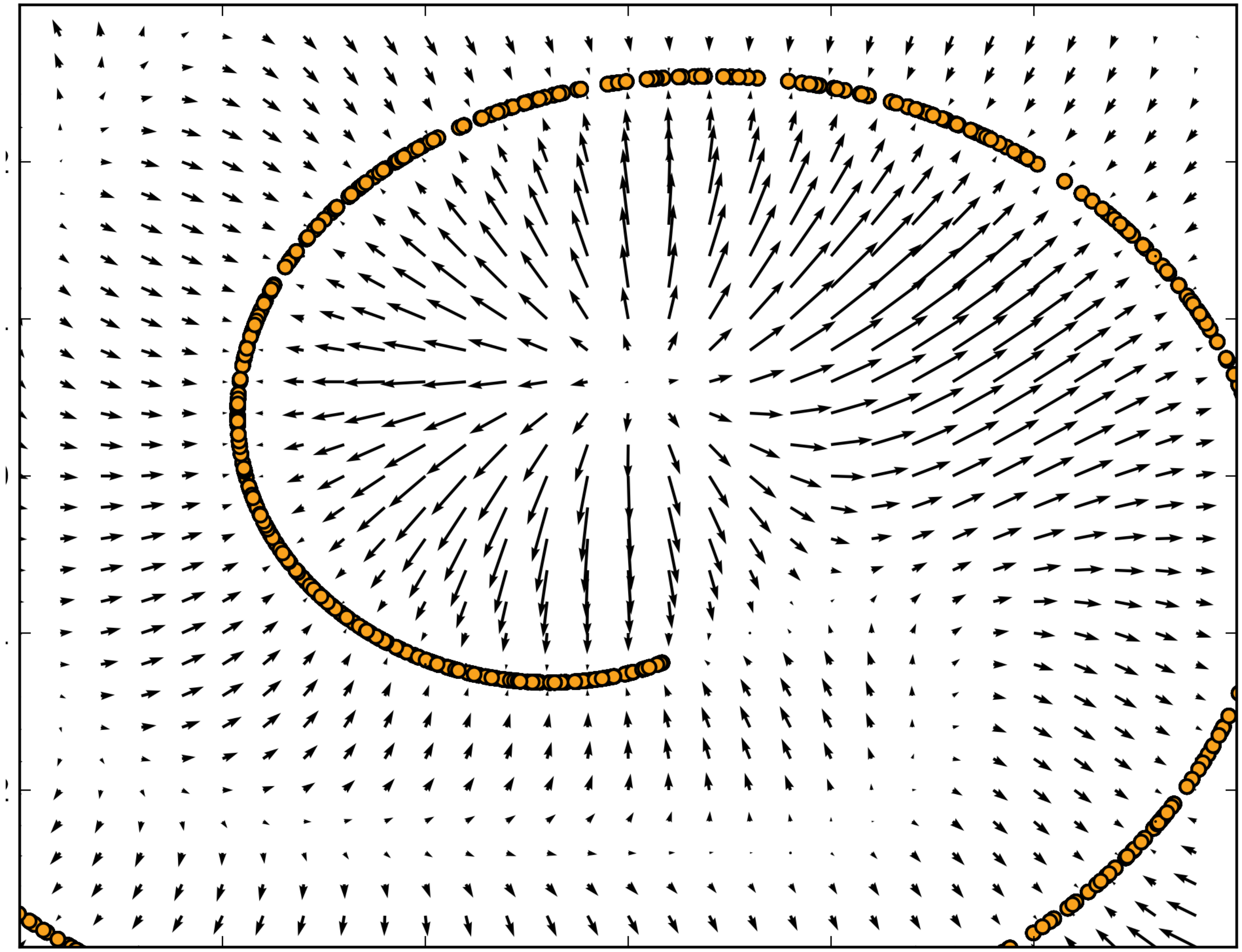
[equivalent when noise→0]

prior: examples concentrate near a lower dimensional "manifold"

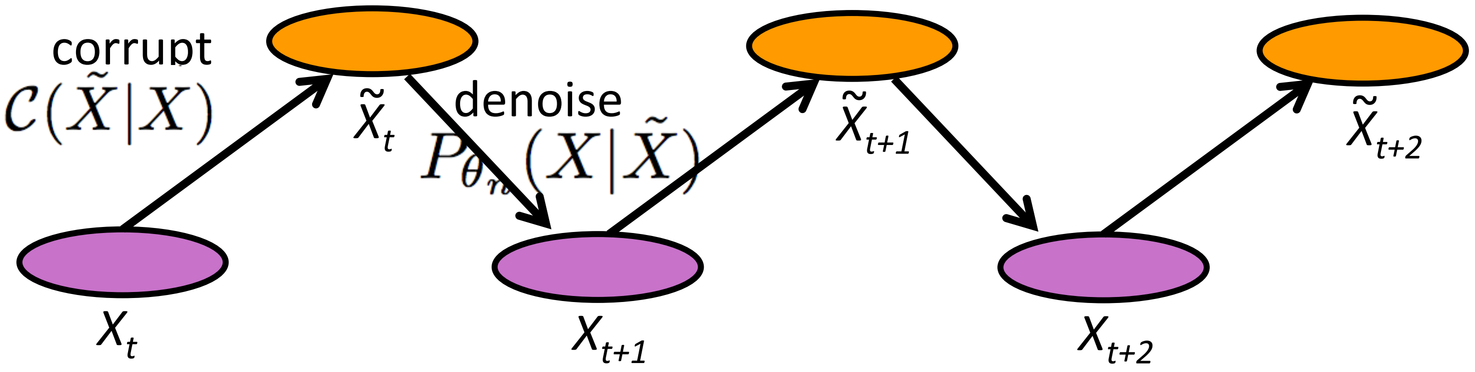


Regularized Auto-Encoders Learn a Vector Field that Estimates a Gradient Field

(Alain & Bengio ICLR 2013)



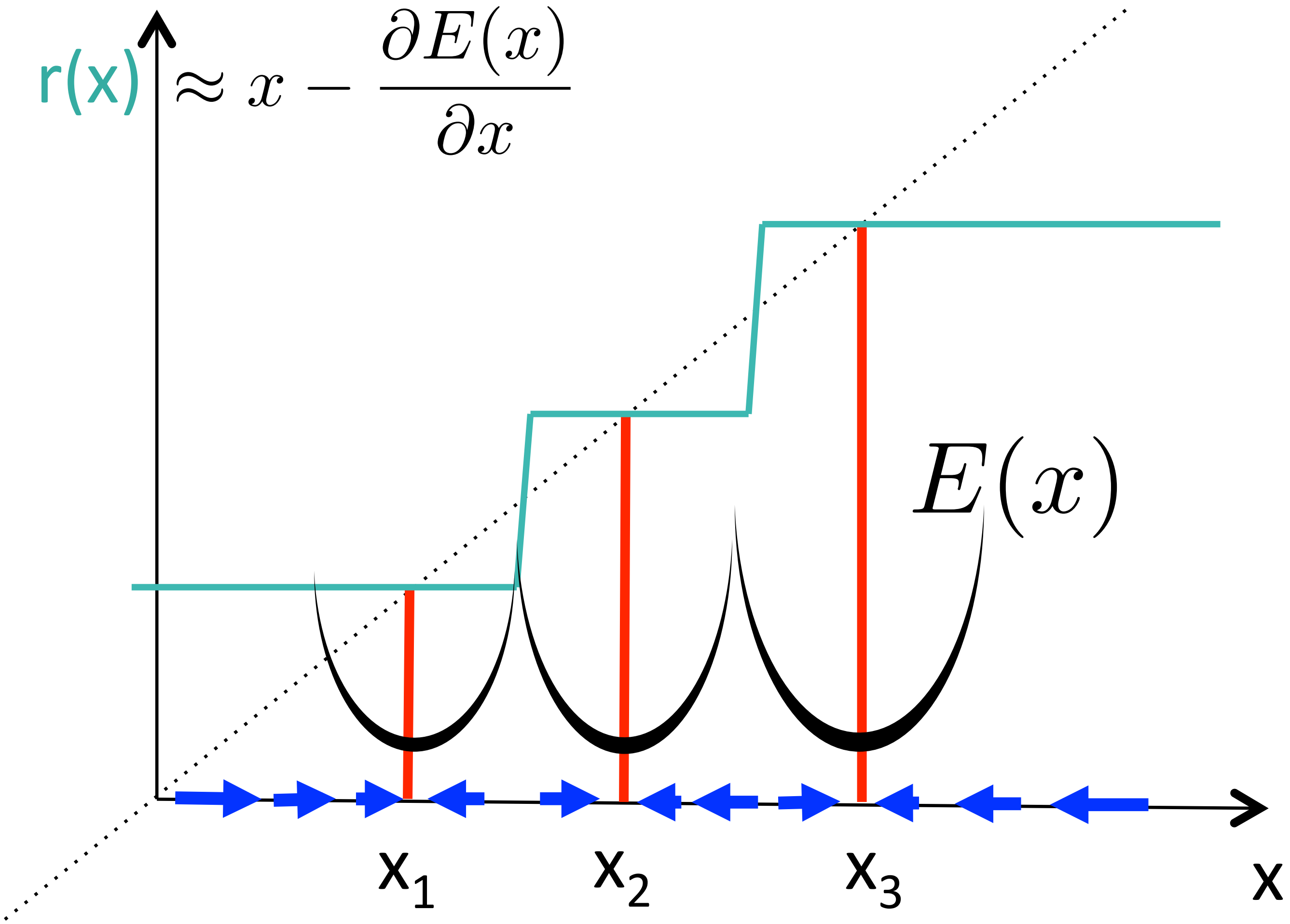
Denoising Auto-Encoder Markov Chain



The corrupt-encode-decode-sample Markov chain associated with a DAE samples from a consistent estimator of the data generating distribution

Preference for Locally Constant Features

- Denoising or contractive auto-encoder on 1-D input:



$$E[||r(x+\sigma z)-x||^2] \approx E[||r(x)-x||^2] + \sigma^2 ||\frac{\partial r(x)}{\partial x}||_F^2$$

Helmholtz Machines *(Hinton et al 1995)* and Variational Auto-Encoders (VAEs)

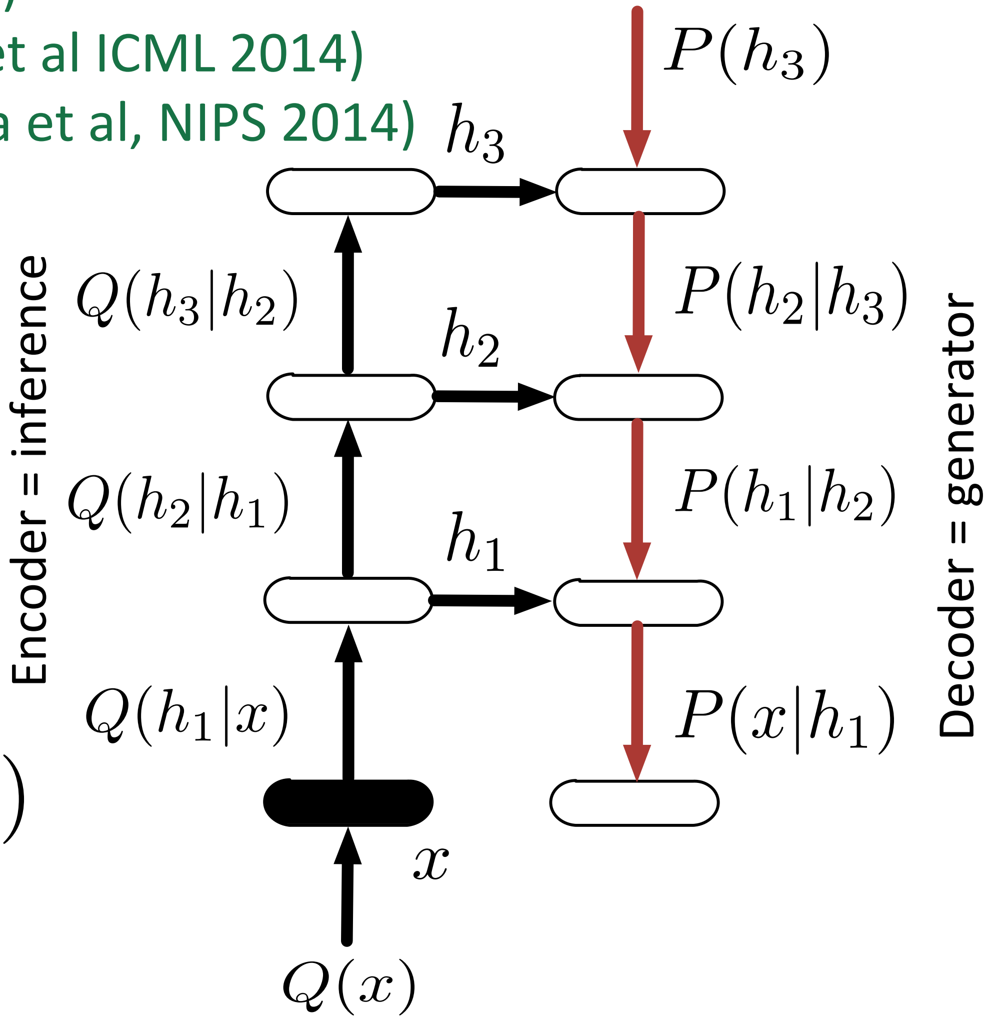
(Kingma & Welling 2013, ICLR 2014)
(Gregor et al ICML 2014; Rezende et al ICML 2014)
(Mnih & Gregor ICML 2014; Kingma et al, NIPS 2014)

- Parametric approximate inference
- Successors of Helmholtz machine *(Hinton et al '95)*
- Maximize variational lower bound on log-likelihood:

$$\min KL(Q(x,h)||P(x,h))$$

where $Q(x)$ = data distr.

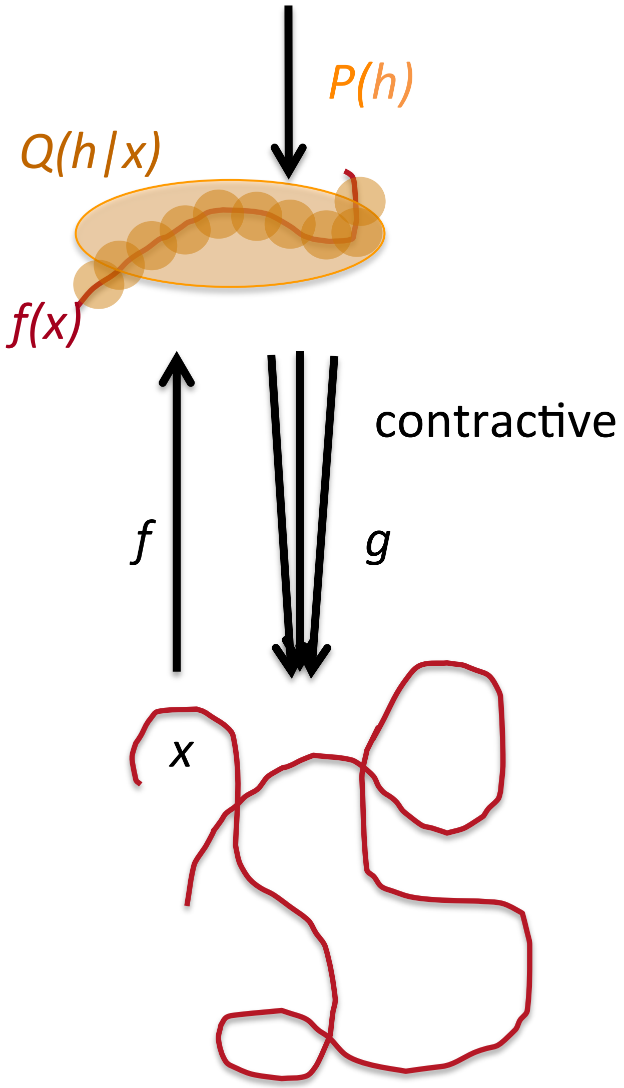
or equivalently



$$\max \sum_x Q(h|x) \log \frac{P(x,h)}{Q(h|x)} = \max \sum_x Q(h|x) \log P(x|h) + KL(Q(h|x)||P(h))$$

Geometric Interpretation

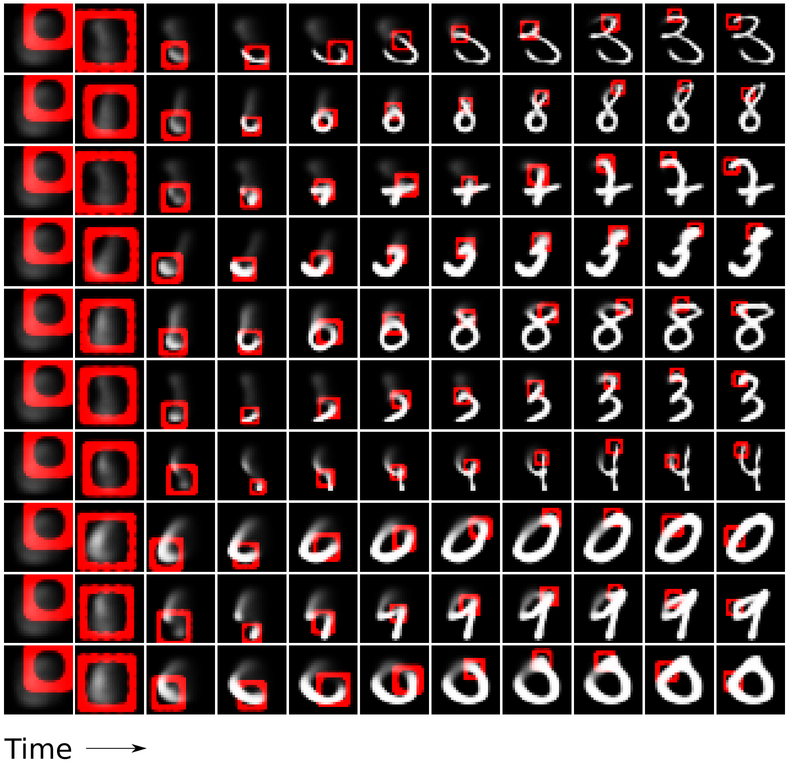
- Encoder: map input to a new space where the data has a simpler distribution
- Add noise between encoder output and decoder input: train the decoder to be robust to mismatch between encoder output and prior output.

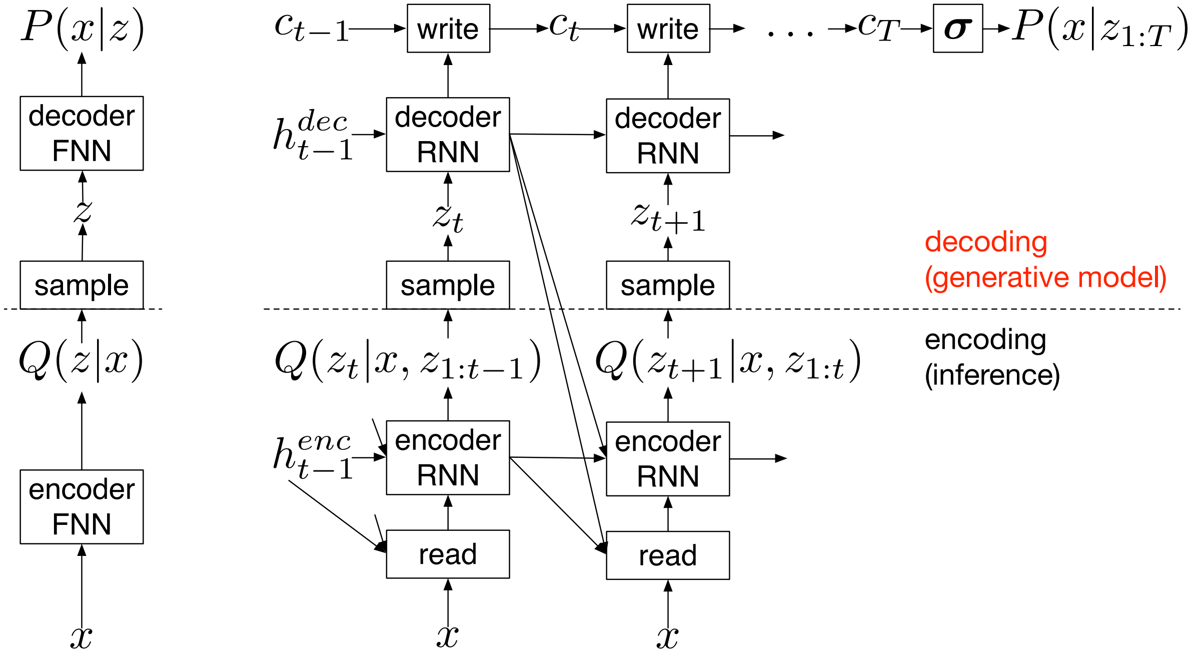


DRAW: Sequential Variational Auto-Encoder with Attention

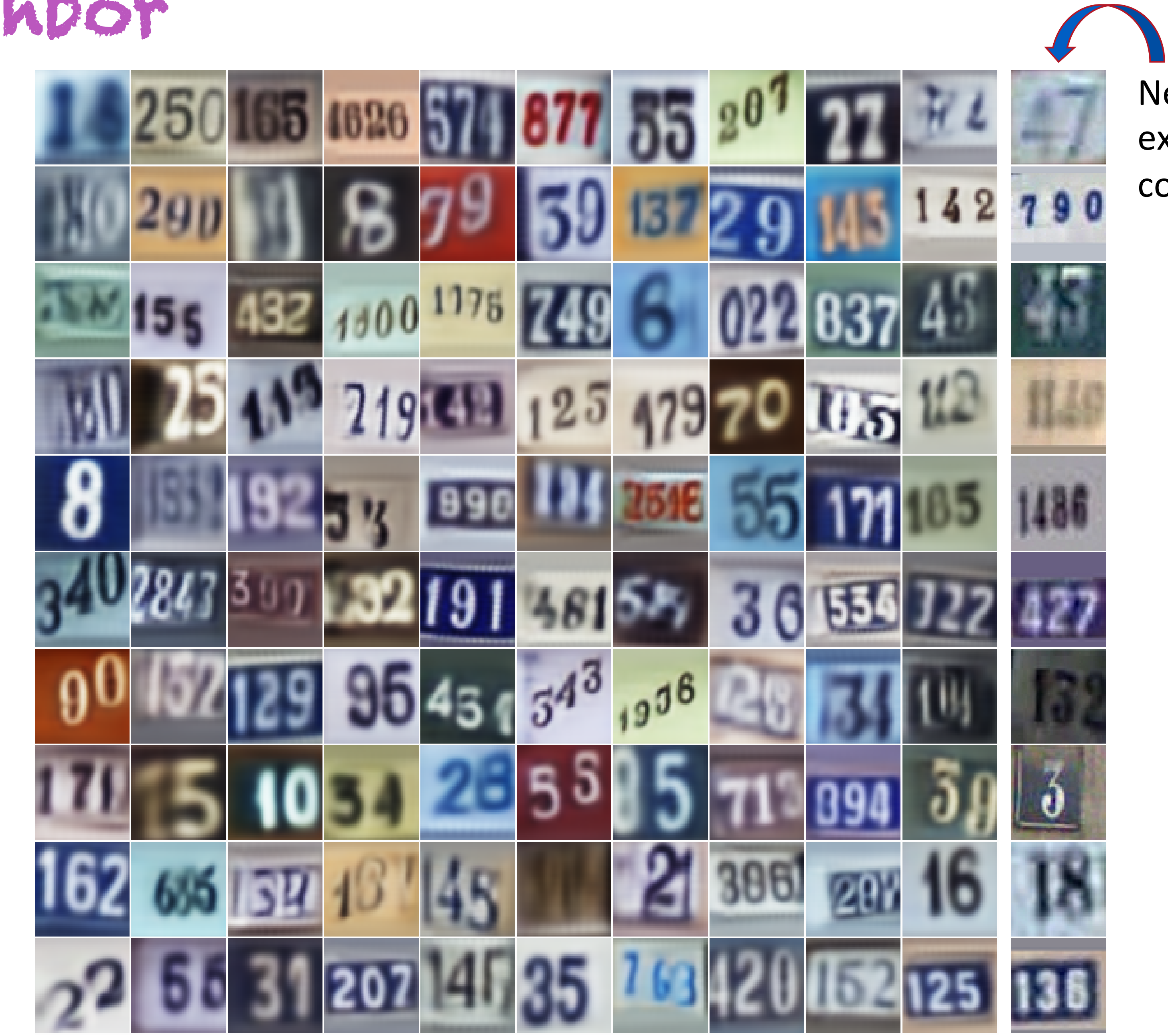
(Gregor et al of Google DeepMind, arXiv 1502.04623, 2015)

- Even for a static input, the encoder and decoder are now **recurrent nets**, which gradually add elements to the answer, and use an attention mechanism to choose where to do so.





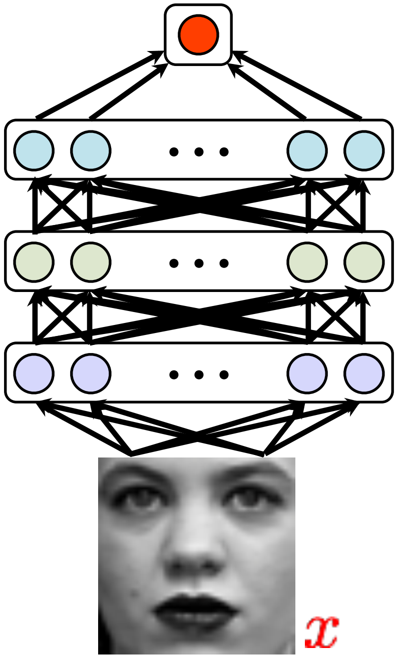
DRAW Samples of SVHN Images: generated samples vs training nearest neighbor

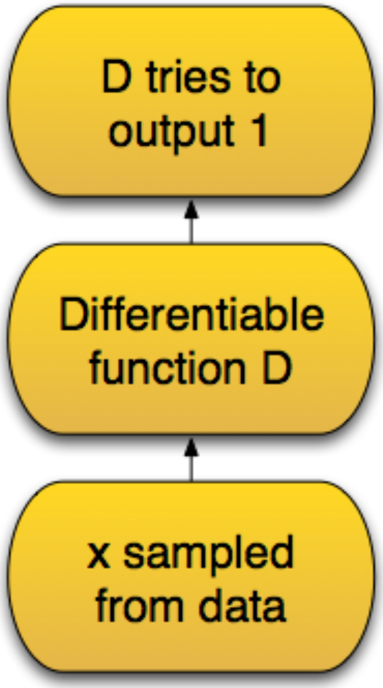


Nearest training example for last column of samples

GAN: Generative Adversarial Networks

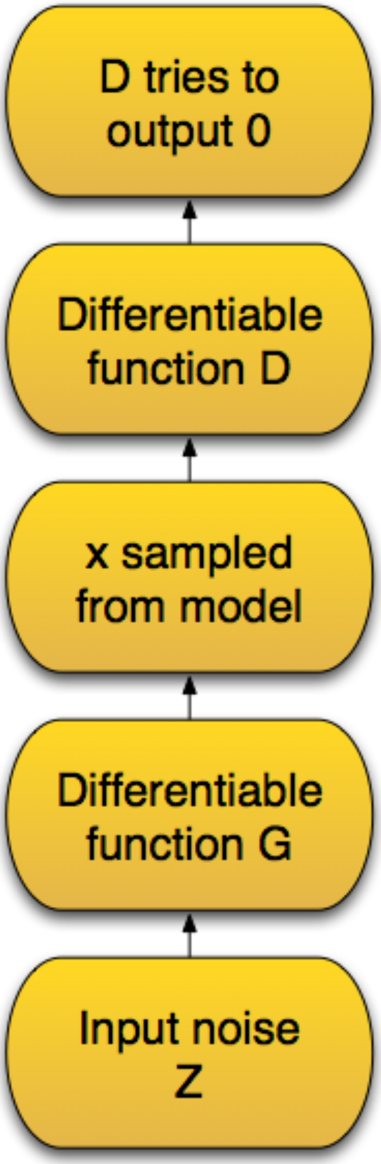
Goodfellow et al NIPS 2014




D tries to output 1

Differentiable function D

x sampled from data

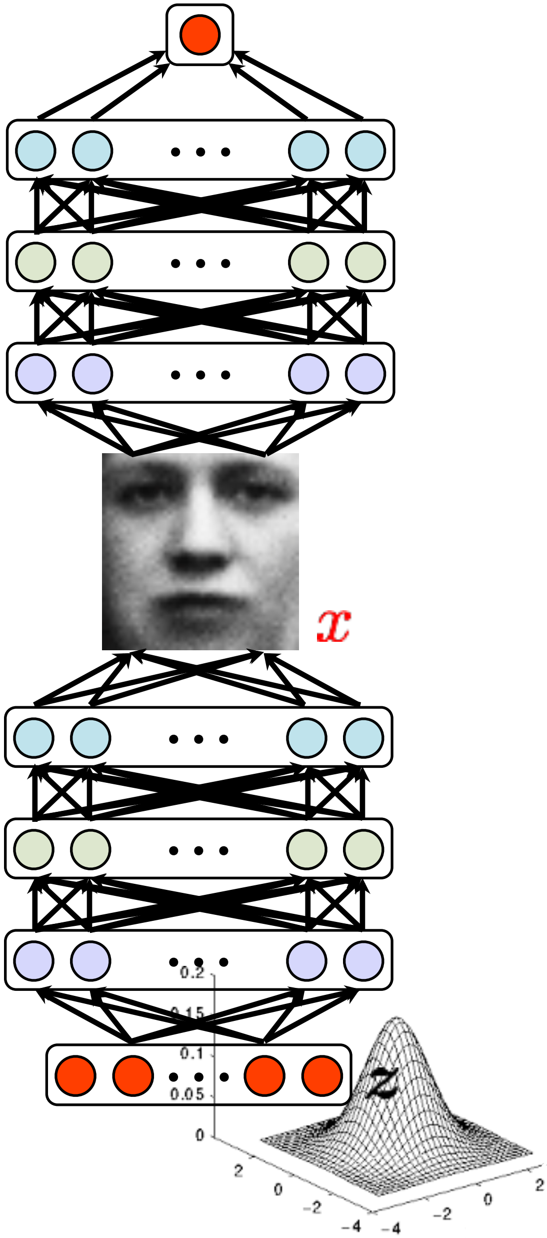

D tries to output 0

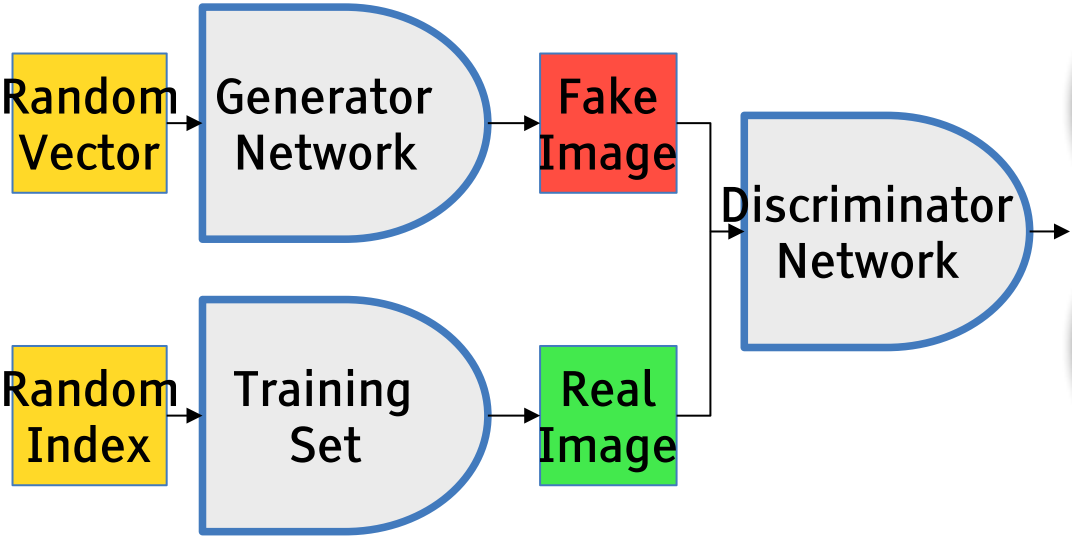
Differentiable function D

x sampled from model

Differentiable function G

Input noise Z




Random Vector

Generator Network

Fake Image

Random Index

Training Set

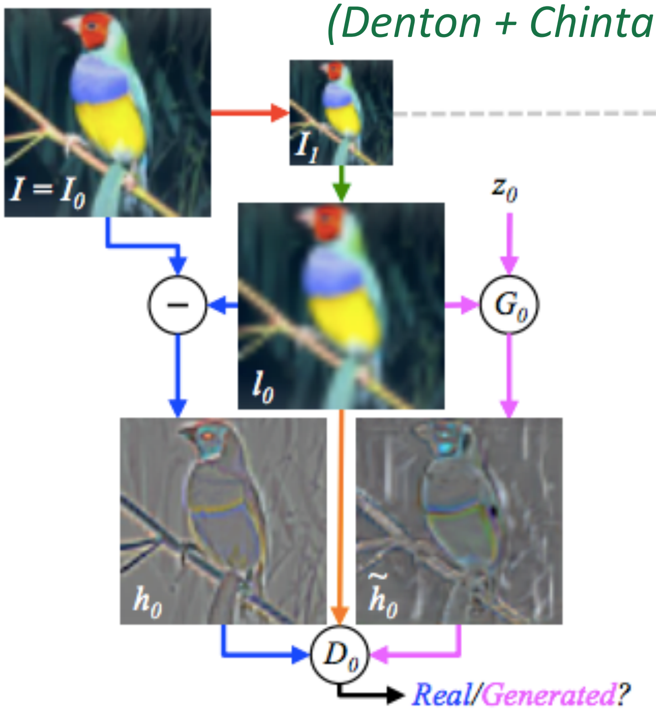
Real Image

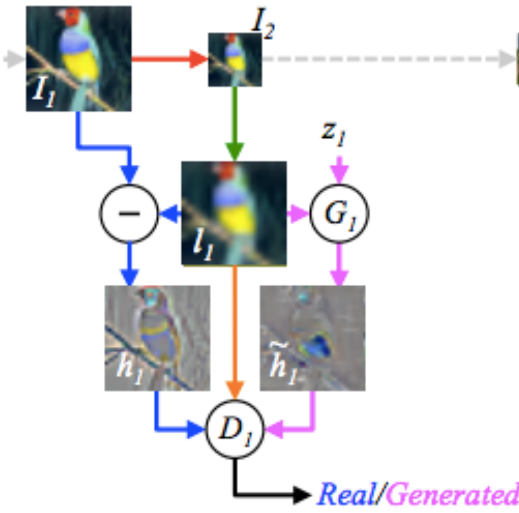
Discriminator Network

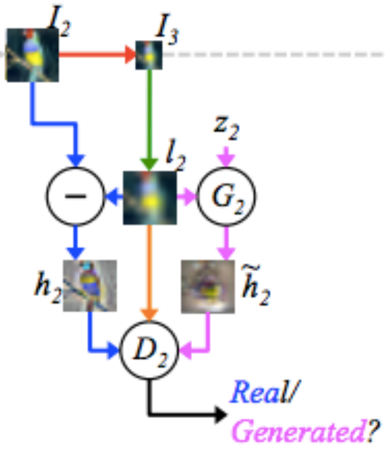
LAPGAN: Laplacian Pyramid of Generative Adversarial Networks

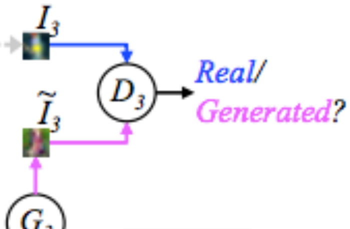
http://soumith.ch/eyescream/

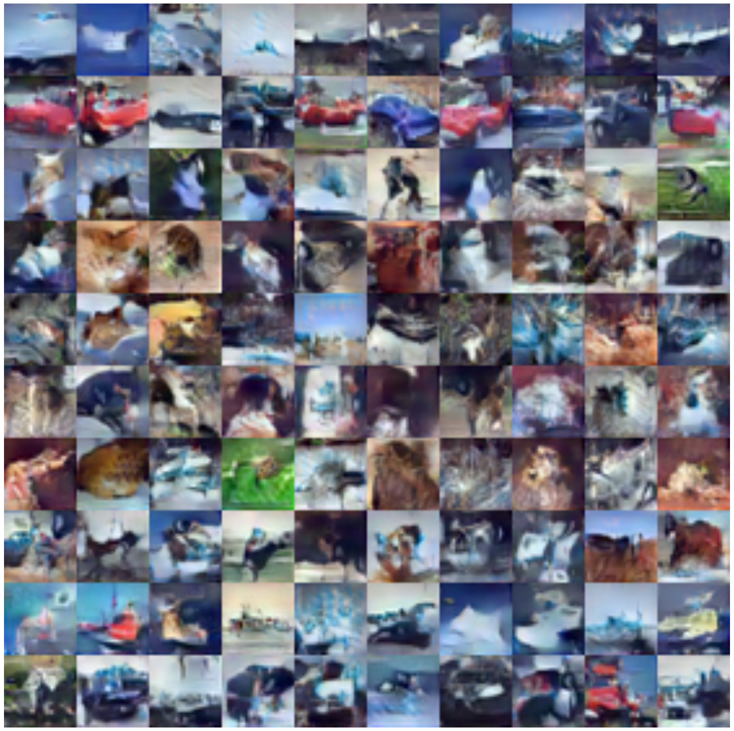
(Denton + Chintala, et al 2015)

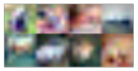


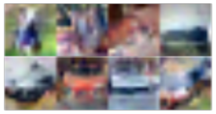


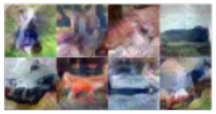






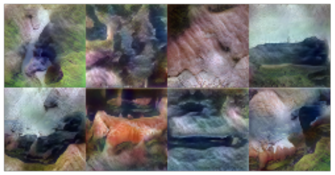

CIFAR-8


CIFAR-16

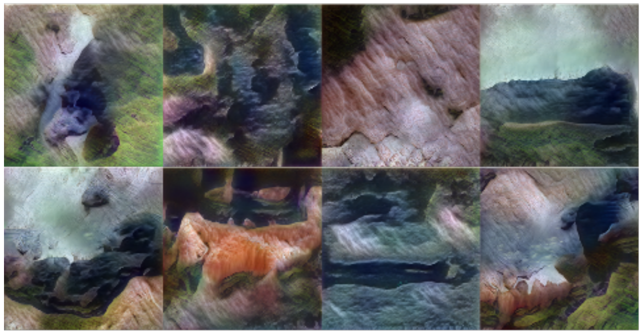

Imagenet-32

Imagenet-32

(recursive)


Imagenet-32

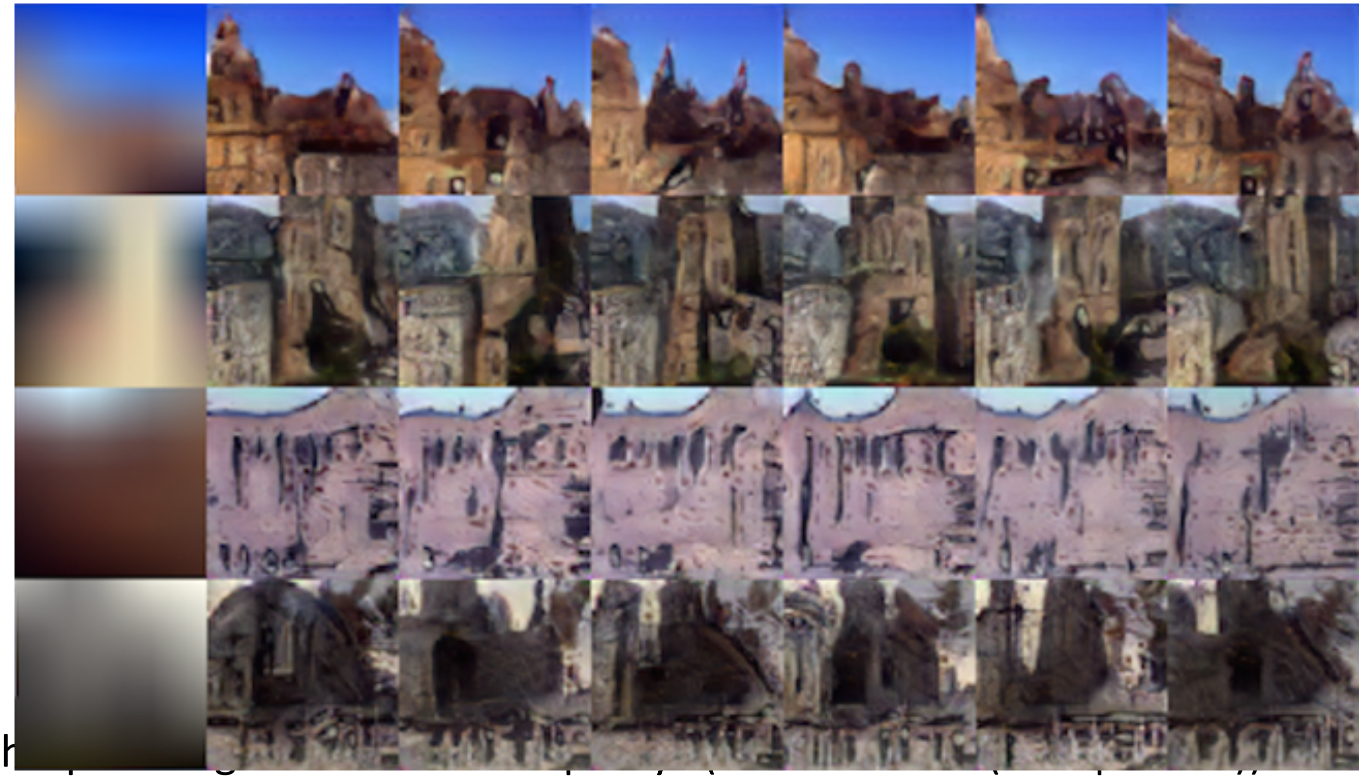
(recursive)



LAPGAN: Visual Turing Test

(Denton + Chintala, et al 2015)

- 40% of samples mistaken *by humans* for real photos

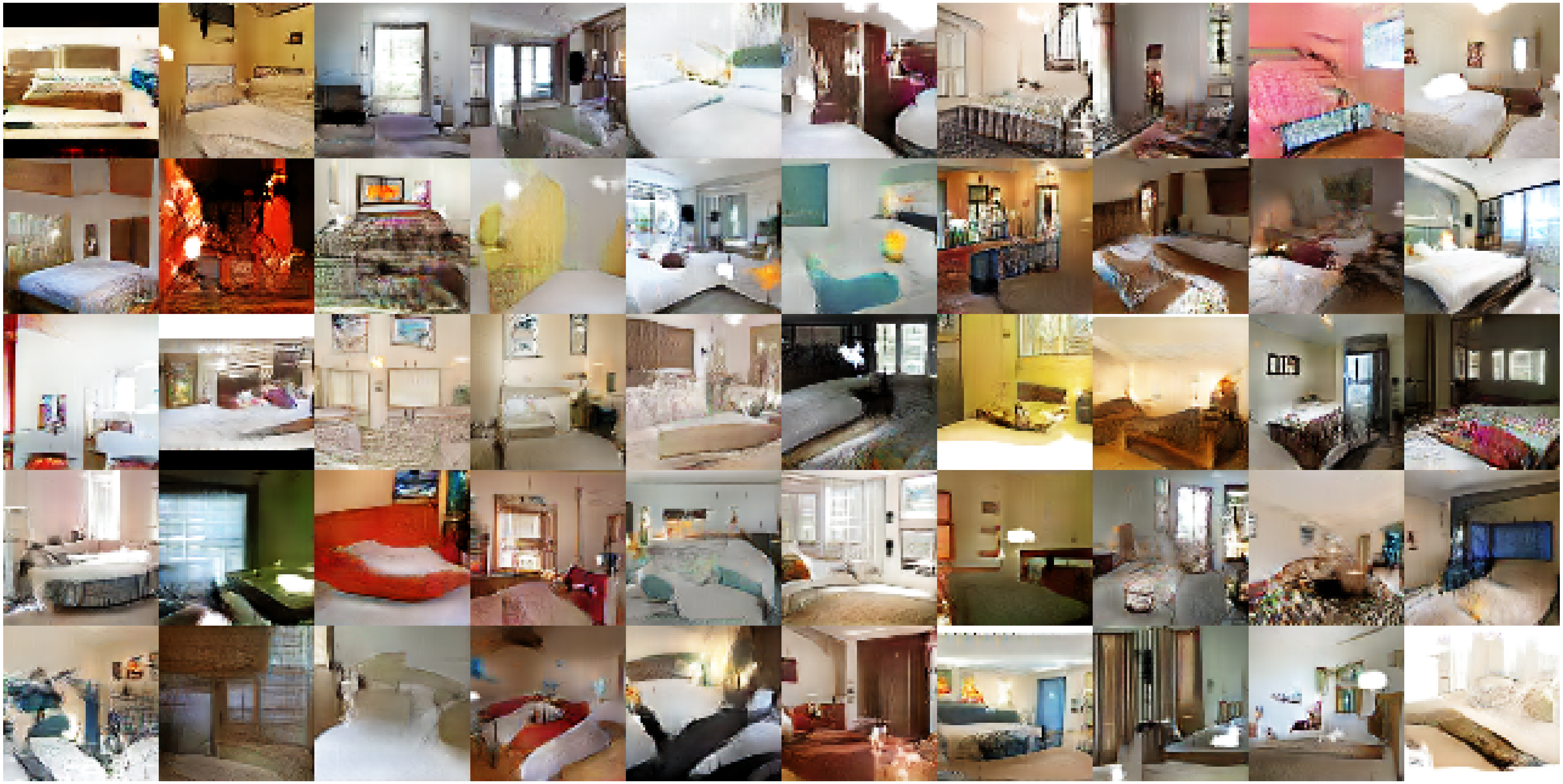


- Sh
- GAN objective = compromise between KL(data|model) and KL(model|data)

Convolutional GANs

(Radford et al, arXiv 1511.06343)

Strided convolutions, batch normalization, only convolutional layers, ReLU and leaky ReLU

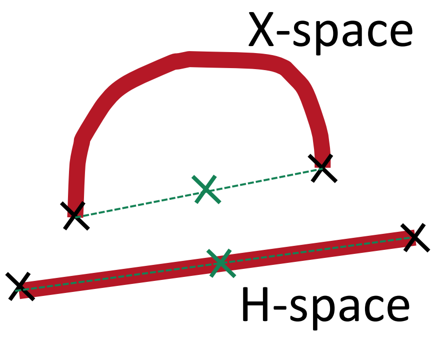


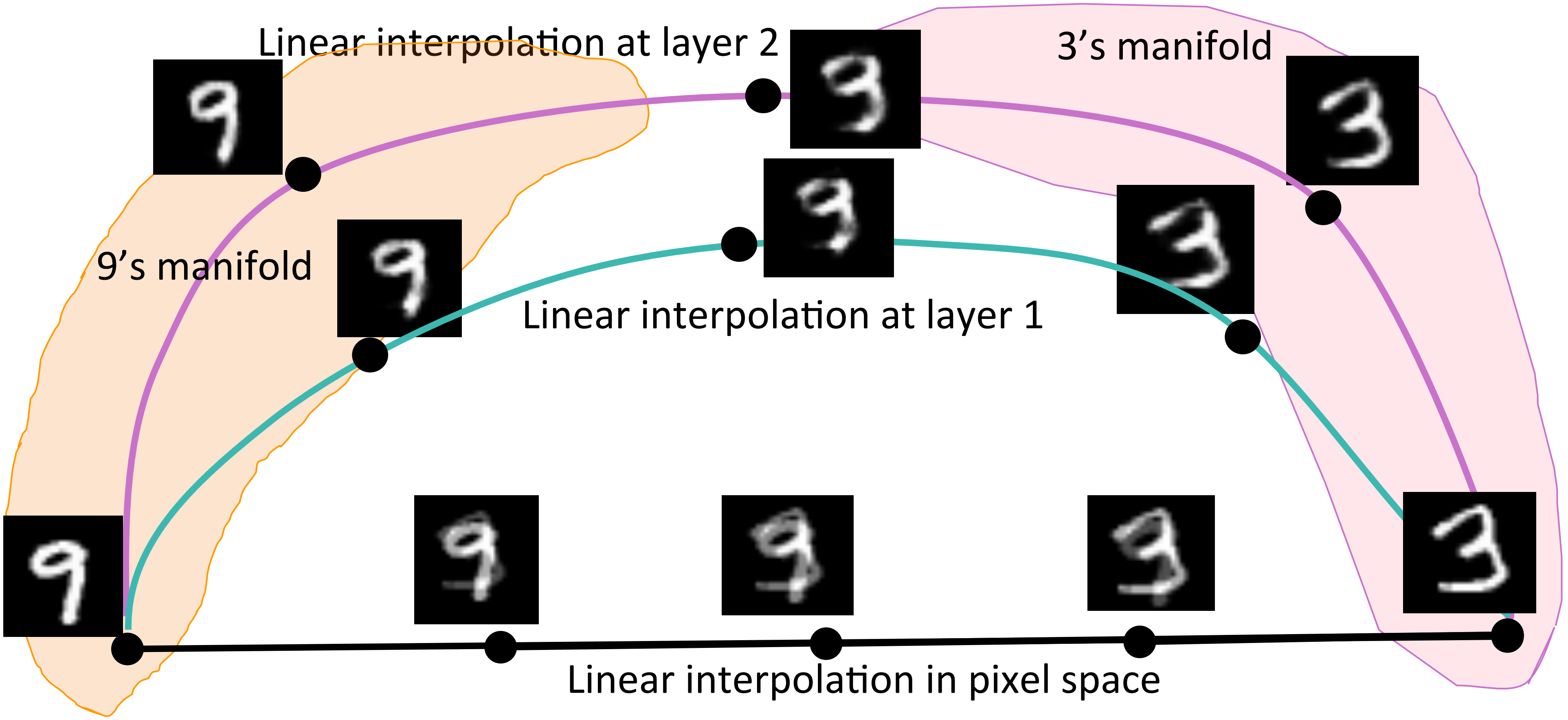
Space-Filling in Representation-Space

(Bengio et al ICML 2013)

Deeper representations ➔ abstractions ➔ disentangling

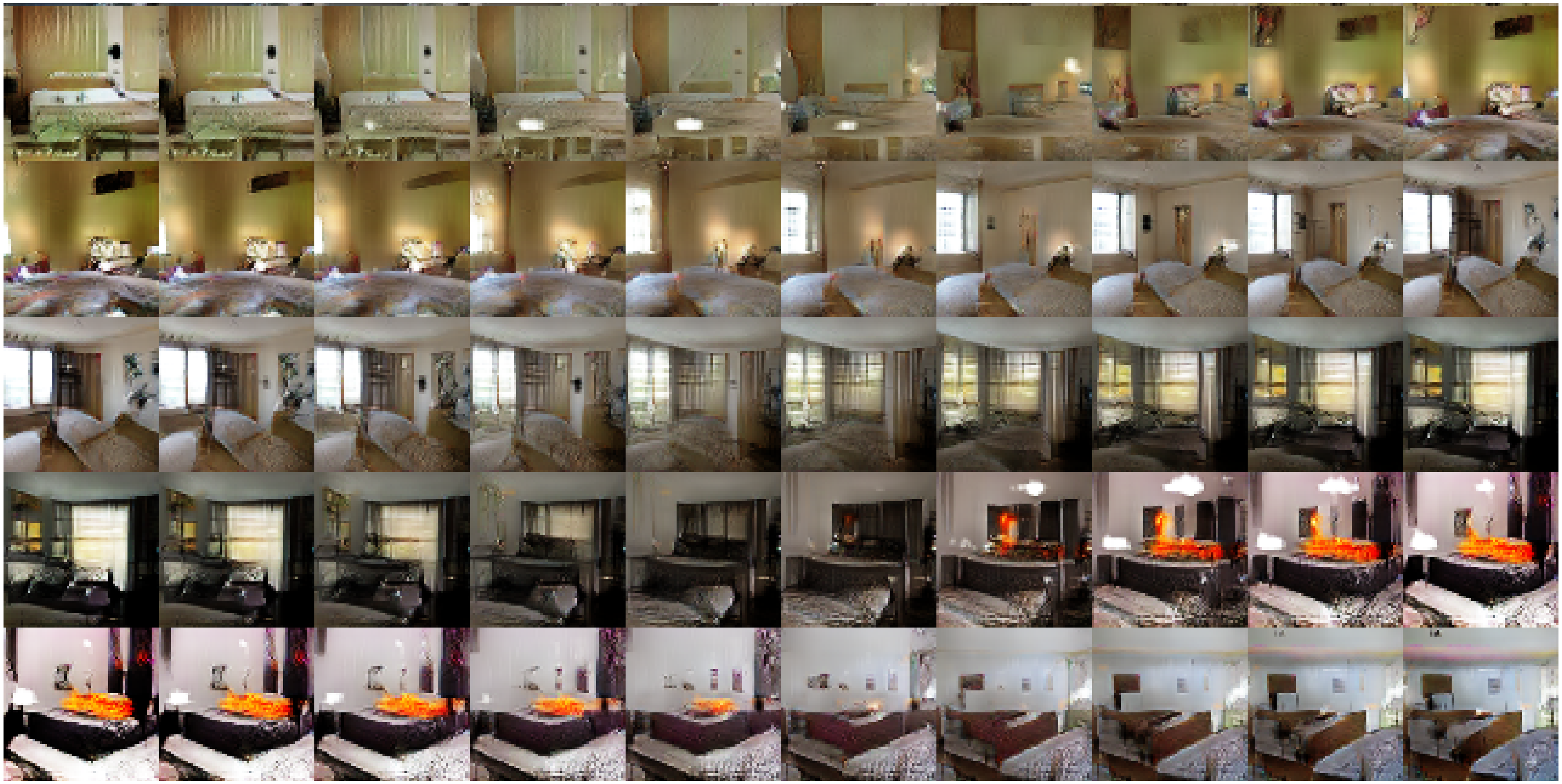
Manifolds are expanded and flattened

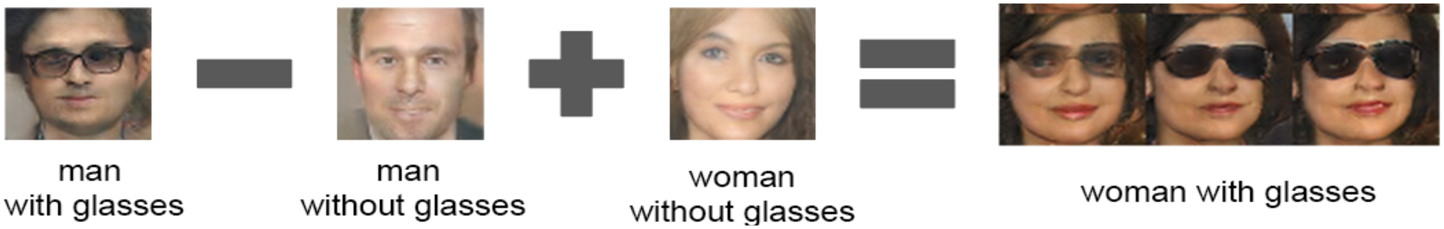



Linear interpolation at layer 2
3's manifold
9's manifold
Linear interpolation at layer 1
Linear interpolation in pixel space

GAN: Interpolating in Latent Space

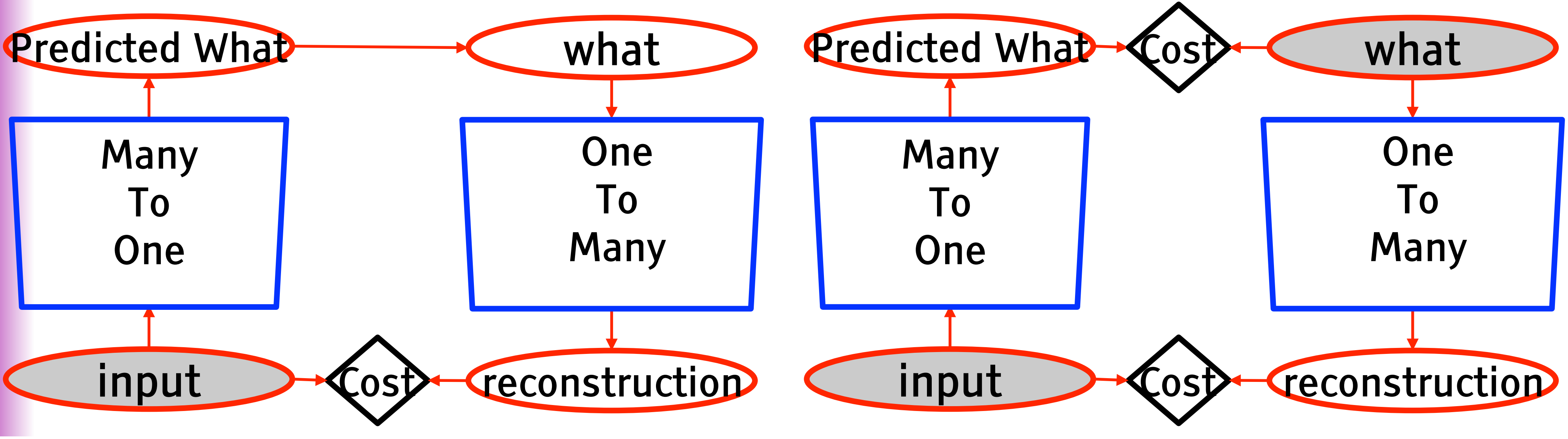
If the model is good (unfolds the manifold), interpolating between latent values yields plausible images.





Supervised and Unsupervised in One Learning Rule?

- **Boltzmann Machines have all the right properties [Hinton 1831] [OK, OK 1983 ;-]**
  - Sup & unsup, generative & discriminative in one simple/local learning rule
  - Feedback circuit reconstructs and propagates virtual hidden targets
  - But they don't really work (or at least they don't scale).
- **Problem: the feedforward path eliminates information**
- **If the feedforward path is invariant, then**
- **the reconstruction path is a one-to-many mapping**
  - Usual solution: sampling. But I'm allergic.



Deep Semi-Supervised Learning

- Unlike unsupervised pre-training, modern approaches optimize jointly the supervised and unsupervised objective

- Discriminative RBMs *(Larochelle & Bengio, ICML 2008)*

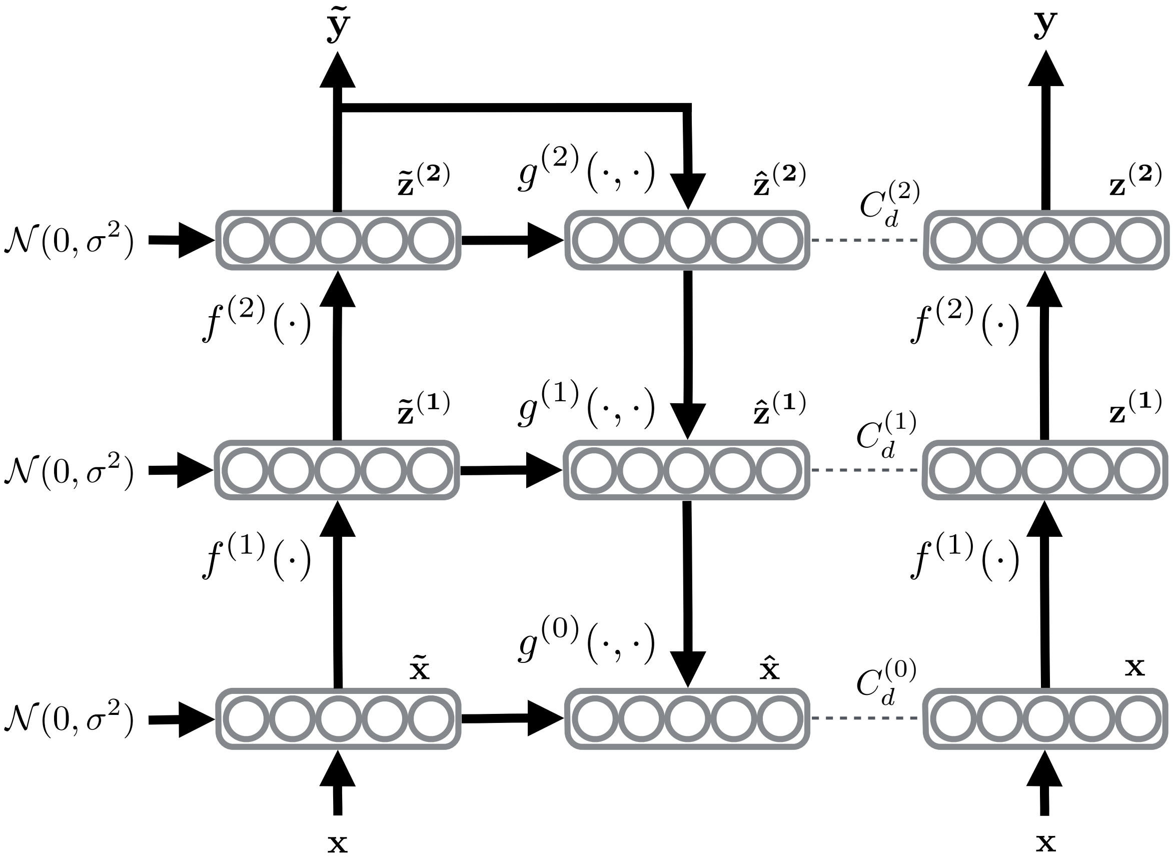
- Semi-Supervised VAE *(Kingma et al, NIPS 2014)*

- Ladder Network *(Rasmus et al, NIPS 2015)*

Semisupervised Learning with Ladder Network

(Rasmus et al, NIPS 2015)

- Jointly trained stack of denoising auto-encoders with gated lateral connections and semi-supervised objective



Semi-supervised objective:

$$-\log P(\tilde{\mathbf{y}} = t(n) \mid \mathbf{x})$$
$$+\sum_{l=1}^{L} \lambda_l \left\| \mathbf{z}^{(l)} - \hat{\mathbf{z}}_{\mathrm{BN}}^{(l)} \right\|^2$$

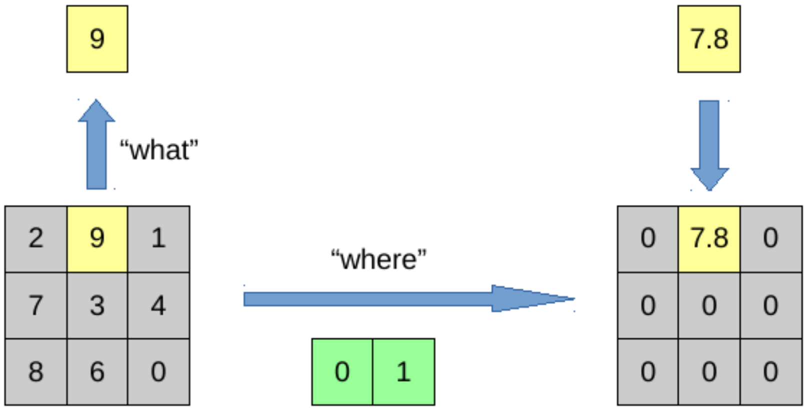
They also use Batch Normalization

1% error on PI-MNIST with 100 labeled examples *(Pezeshki et al arXiv 1511.06430)*

Stacked What-Where Auto-Encoder (SWWAE)

[Zhao, Mathieu, LeCun arXiv:1506.023

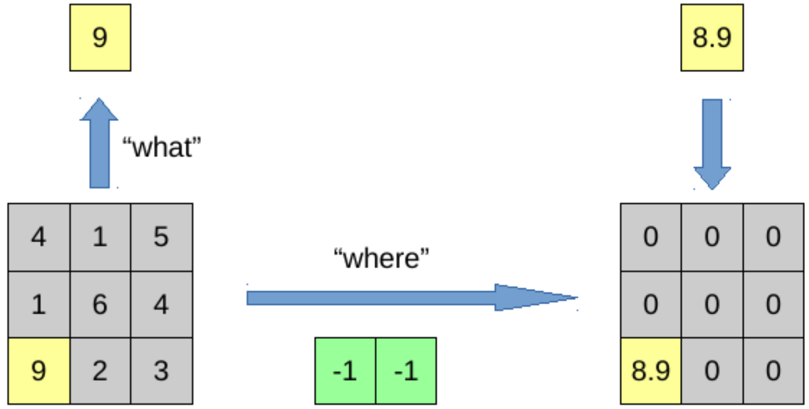
Stacked What-Where Auto-Encoder



2	9	1
7	3	4
8	6	0

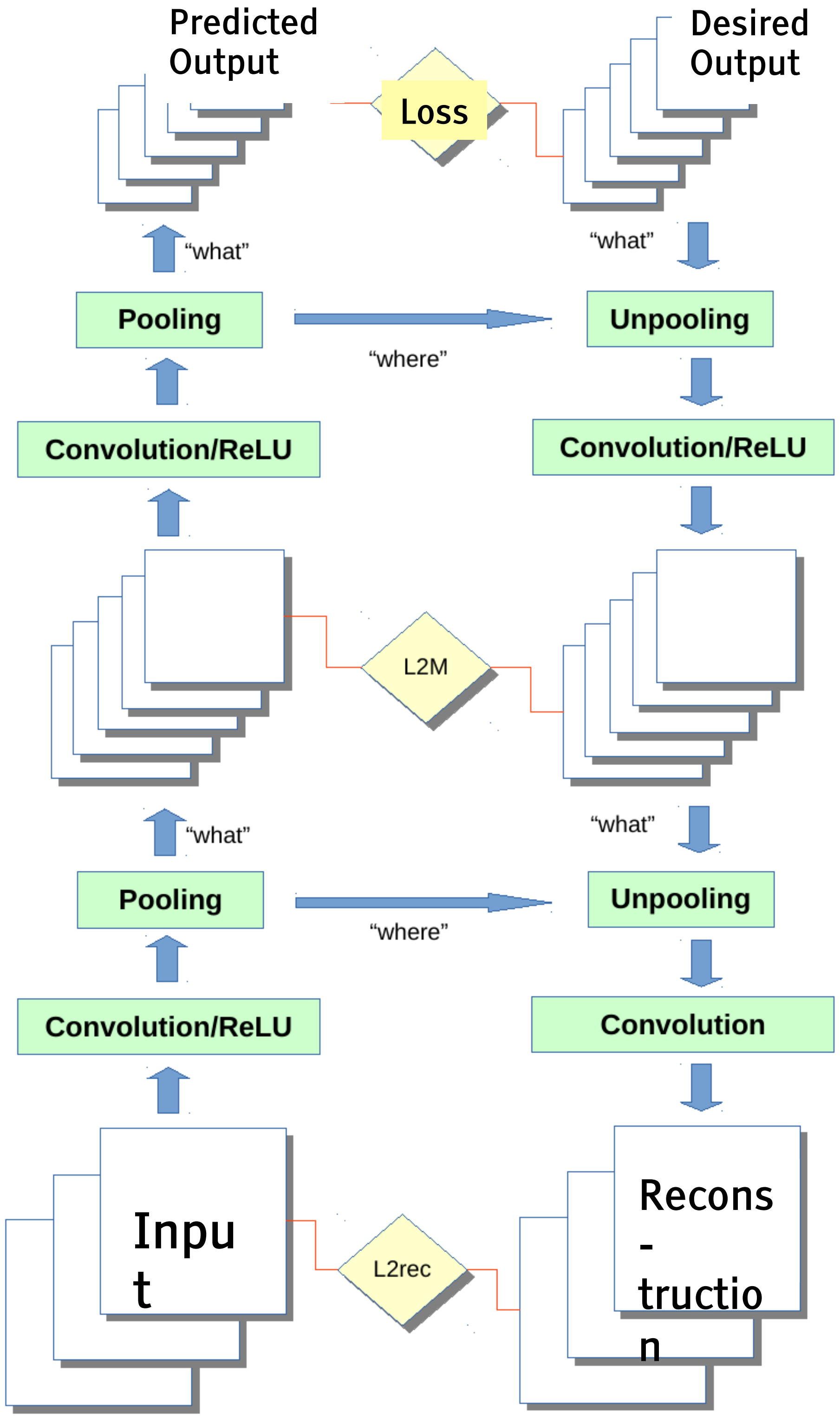
0	7.8	0
0	0	0
0	0	0

A bit like a ConvNet paired with a DeConvNet



4	1	5
1	6	4
9	2	3

0	0	0
0	0	0
8.9	0	0



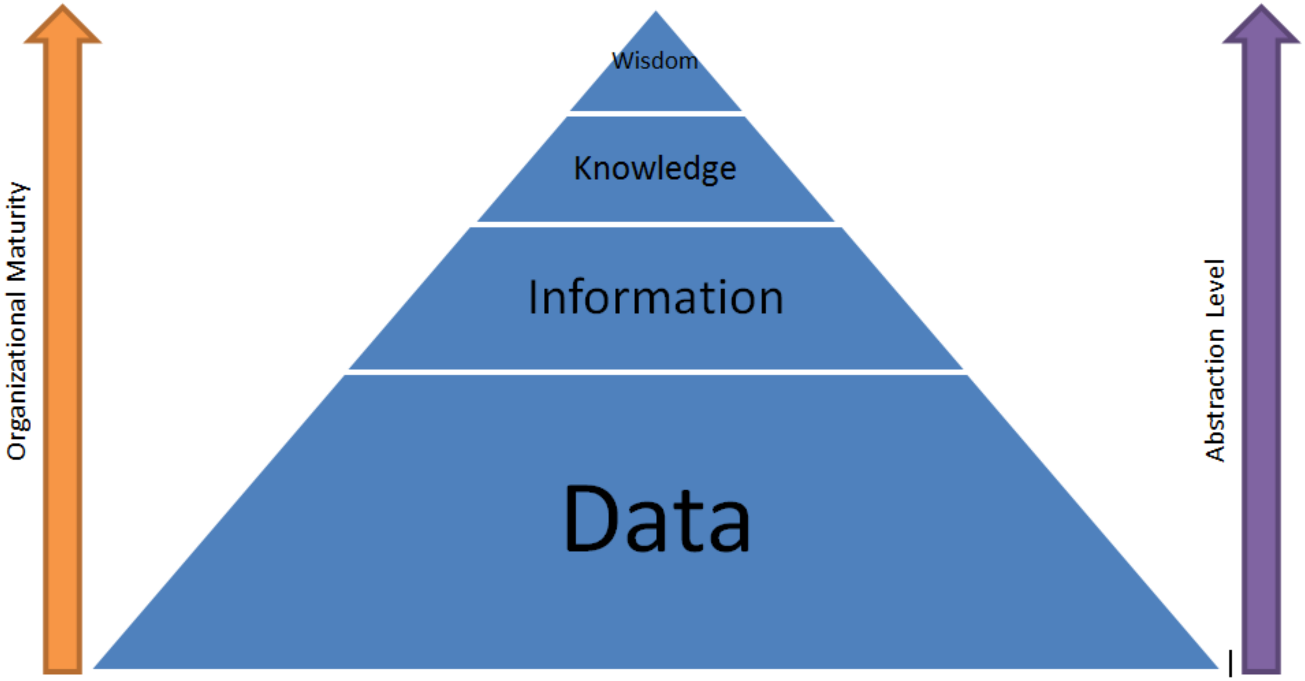
Conclusions & Challenges

Learning « How the world ticks »

- So long as our machine learning models « cheat » by relying only on surface statistical regularities, they remain vulnerable to out-of-distribution examples
- Humans generalize better than other animals by implicitly having a more accurate internal model of the underlying causal relationships
- This allows one to predict future situations (e.g., the effect of planned actions) that are far from anything seen before, an essential component of reasoning, intelligence and science

Learning Multiple Levels of Abstraction

- The big payoff of deep learning is to allow learning higher levels of abstraction
- Higher-level abstractions disentangle the factors of variation, which allows much easier generalization and transfer



Challenges & Open Problems

A More Scientific Approach is Needed, not Just Building Better Systems

- Unsupervised learning
  - How to evaluate?
- Long-term dependencies
- Natural language understanding & reasoning
- More robust optimization (or easier to train architectures)
- Distributed training (that scales) & specialized hardware
- Bridging the gap to biology
- Deep reinforcement learning