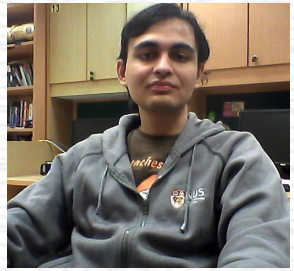
Basics of Deep Learning

Animesh Prasad
animesh@comp.nus.edu.sg

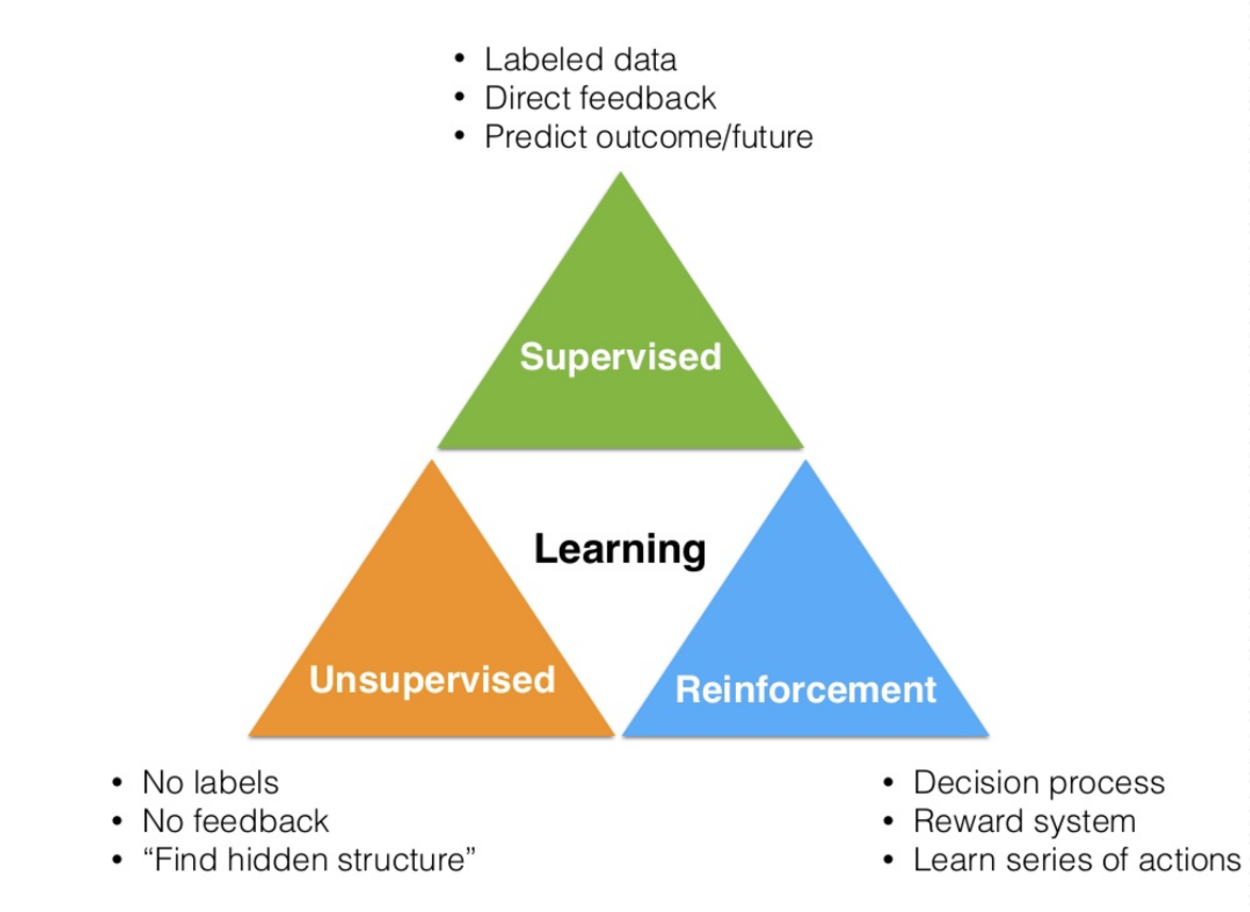
Muthu Kumar Chandrasekaran
cmkumar@comp.nus.edu.sg



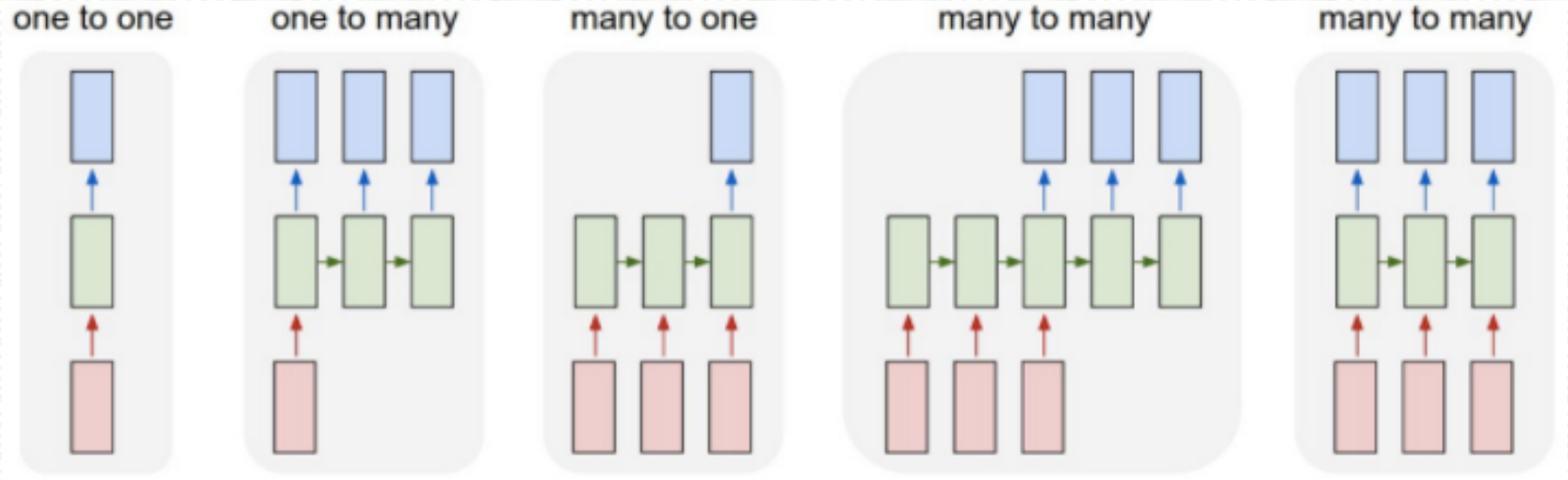
NUS National University of Singapore

WING Web Information Retrieval / Natural Language Processing Group

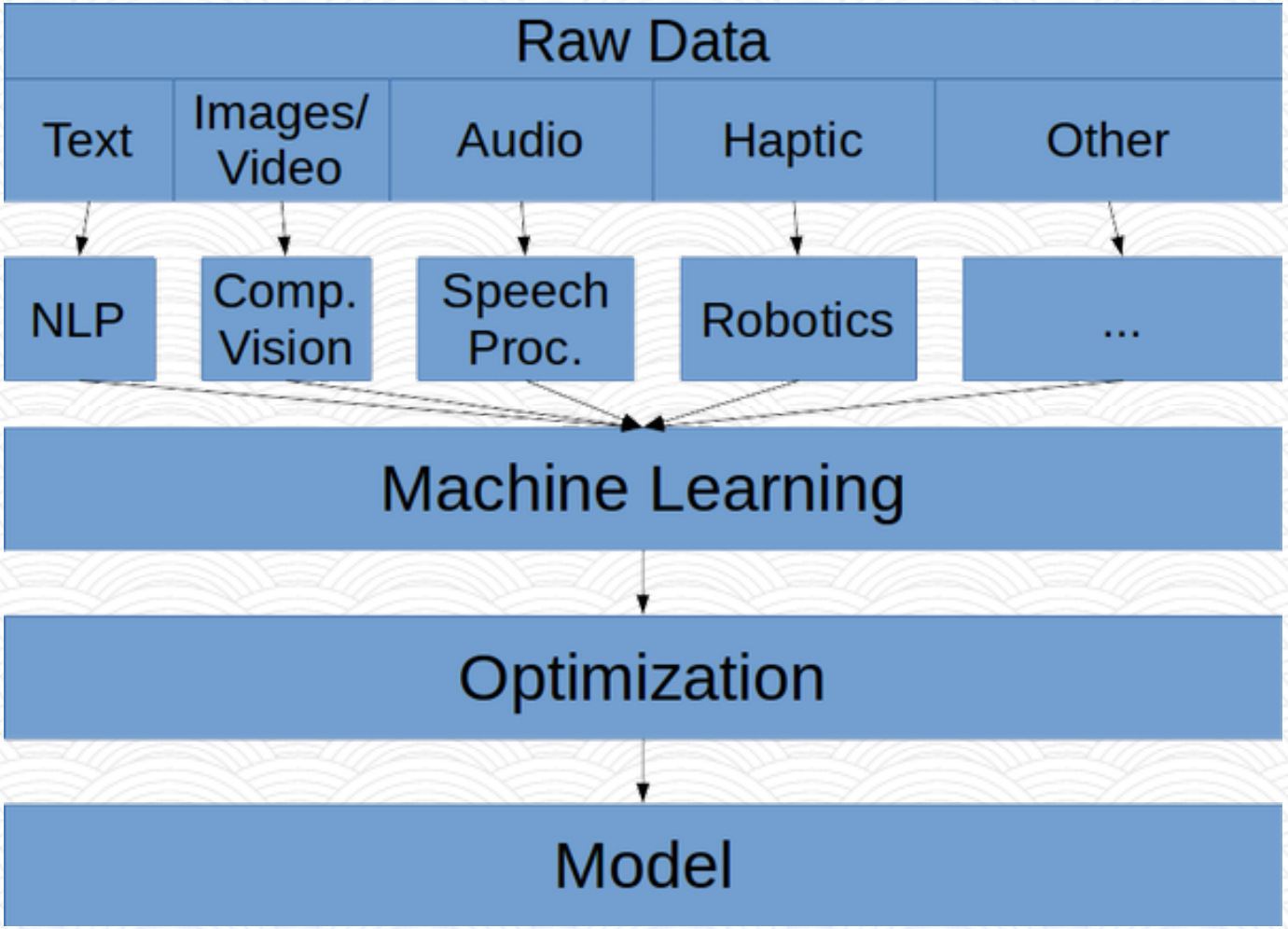
Machine Learning Primer



Machine Learning Primer



Machine Learning Primer



Machine Learning, AI

- 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

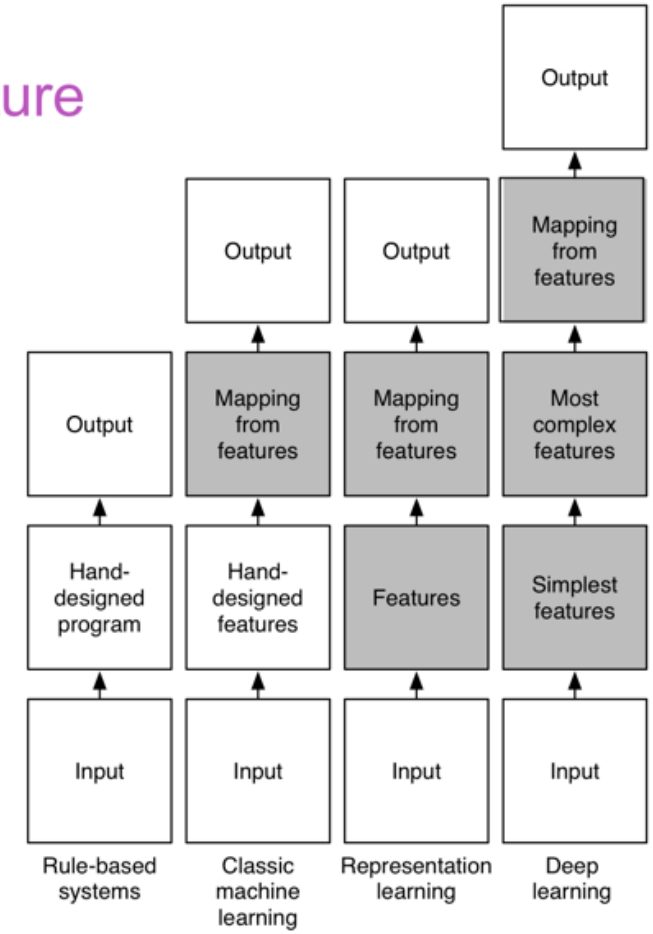
Additional prior: compositionality is useful to describe the world around us efficiently

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.

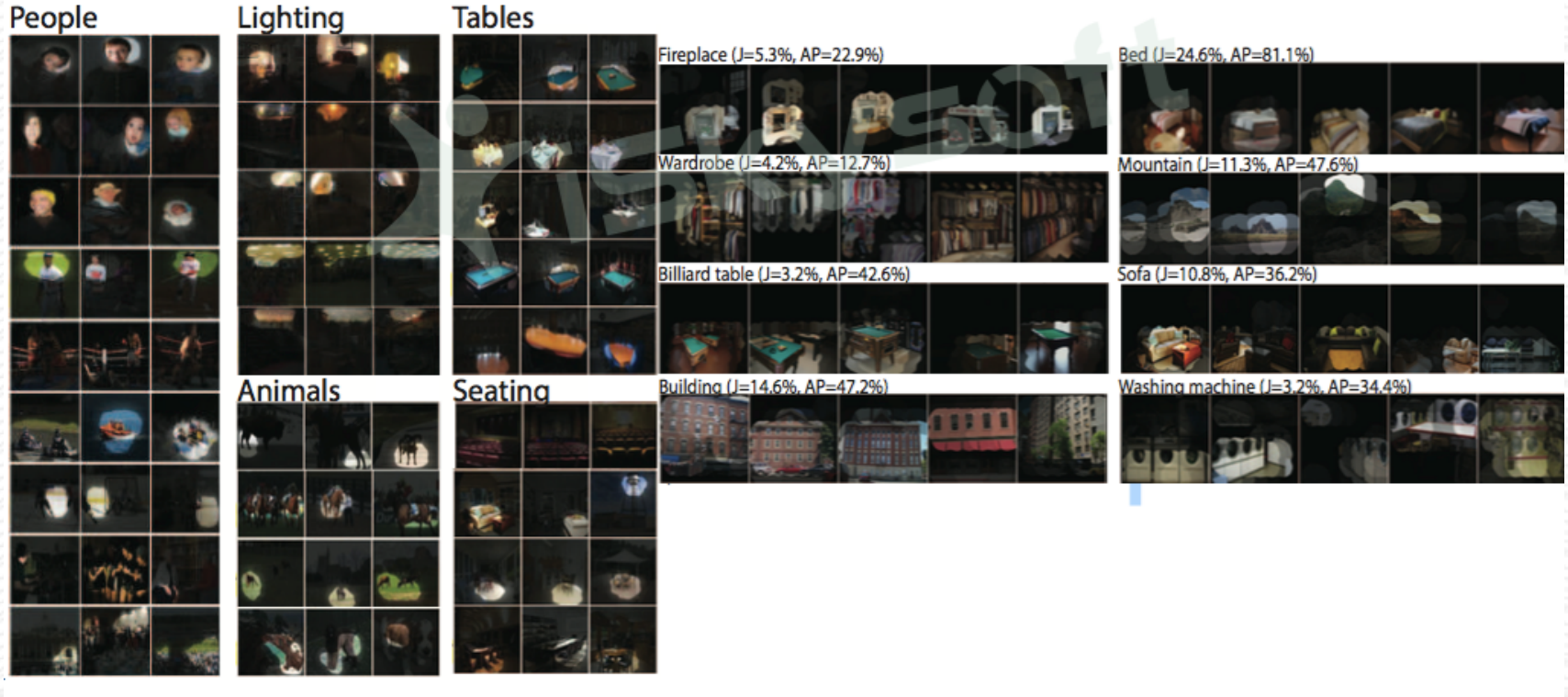
If each of n feature requires $O(k)$ parameters, need $O(nk)$ examples Non-parametric methods would require $O(n^d)$ examples

Deep Learning: Automating Feature Discovery



Hidden Units Discover Semantically Meaningful Concepts

- Network trained to recognize places, not objects



Why does it work?

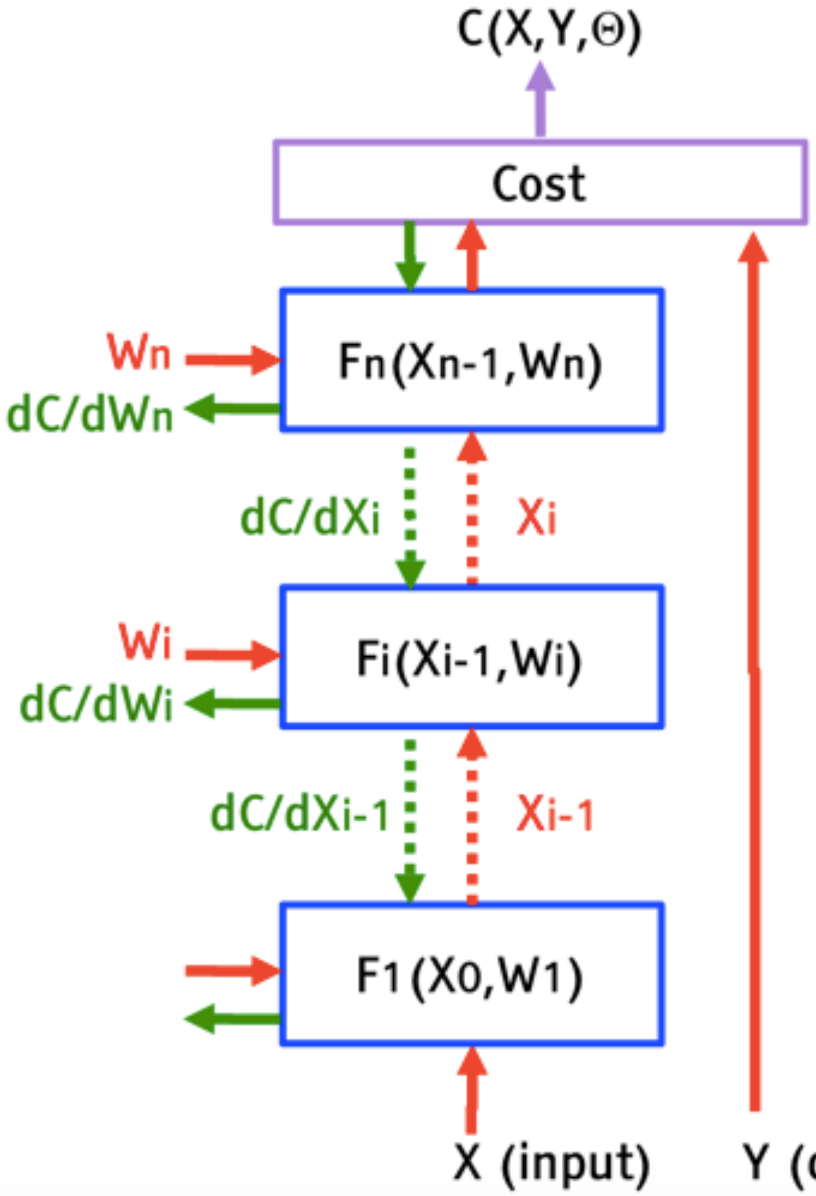
- It only works because we are making some assumptions about the data generating distribution
- Worse-case distributions still require exponential data
- But the world has structure and we can get an exponential gain by exploiting some of it

http://playground.tensorflow.org


Backprop
(modular approach)

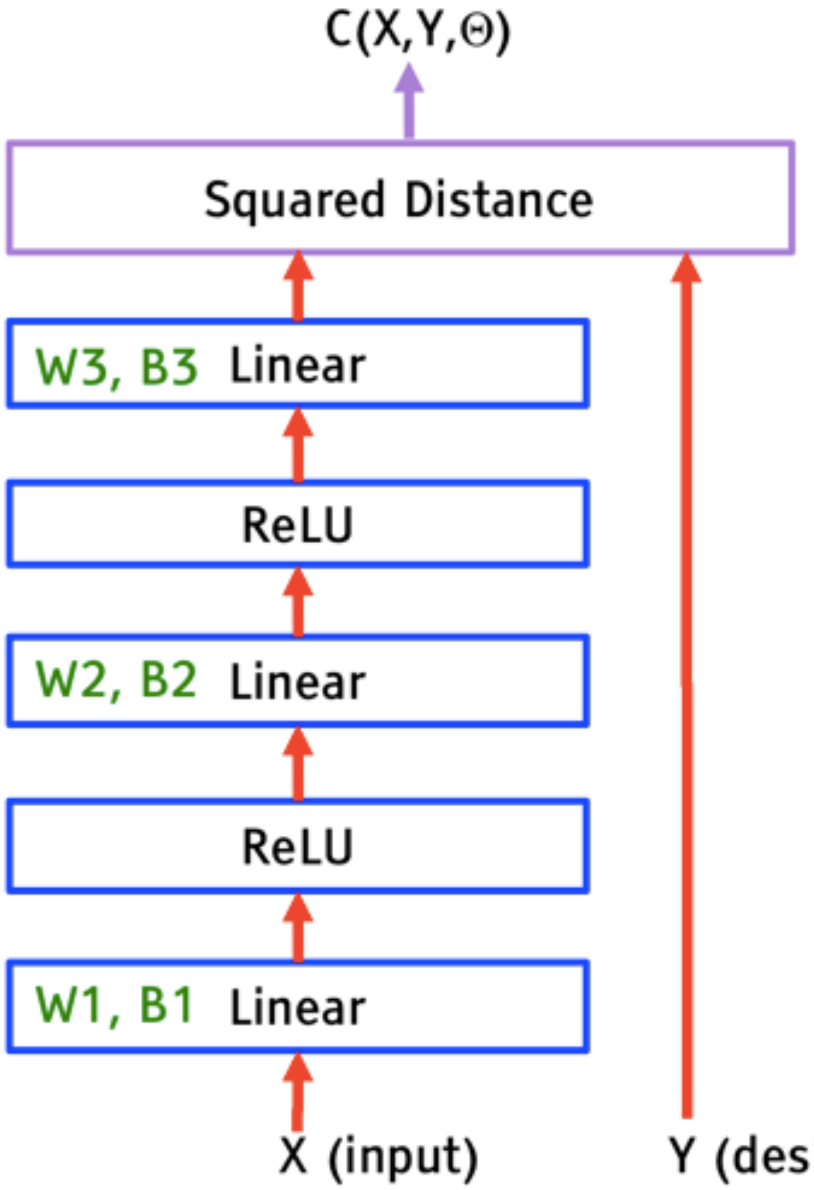
Computing Gradients by Back-Propagation

Y LeCun



- A practical Application of Chain Rule
- Backprop for the state gradients:
- dC/dXi-1 = dC/dXi . dXi/dXi-1
- dC/dXi-1 = dC/dXi . dFi(Xi-1,Wi)/dXi-1
- Backprop for the weight gradients:
- dC/dWi = dC/dXi . dXi/dWi
- dC/dWi = dC/dXi . dFi(Xi-1,Wi)/dWi

Typical Multilayer Neural Net Architecture



- 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
  - $L(\Theta)=1/p \sum_k C(X^k,Y^k,\Theta)$
  - $\Theta = (W1,B1,W2,B2,W3,B3)$

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

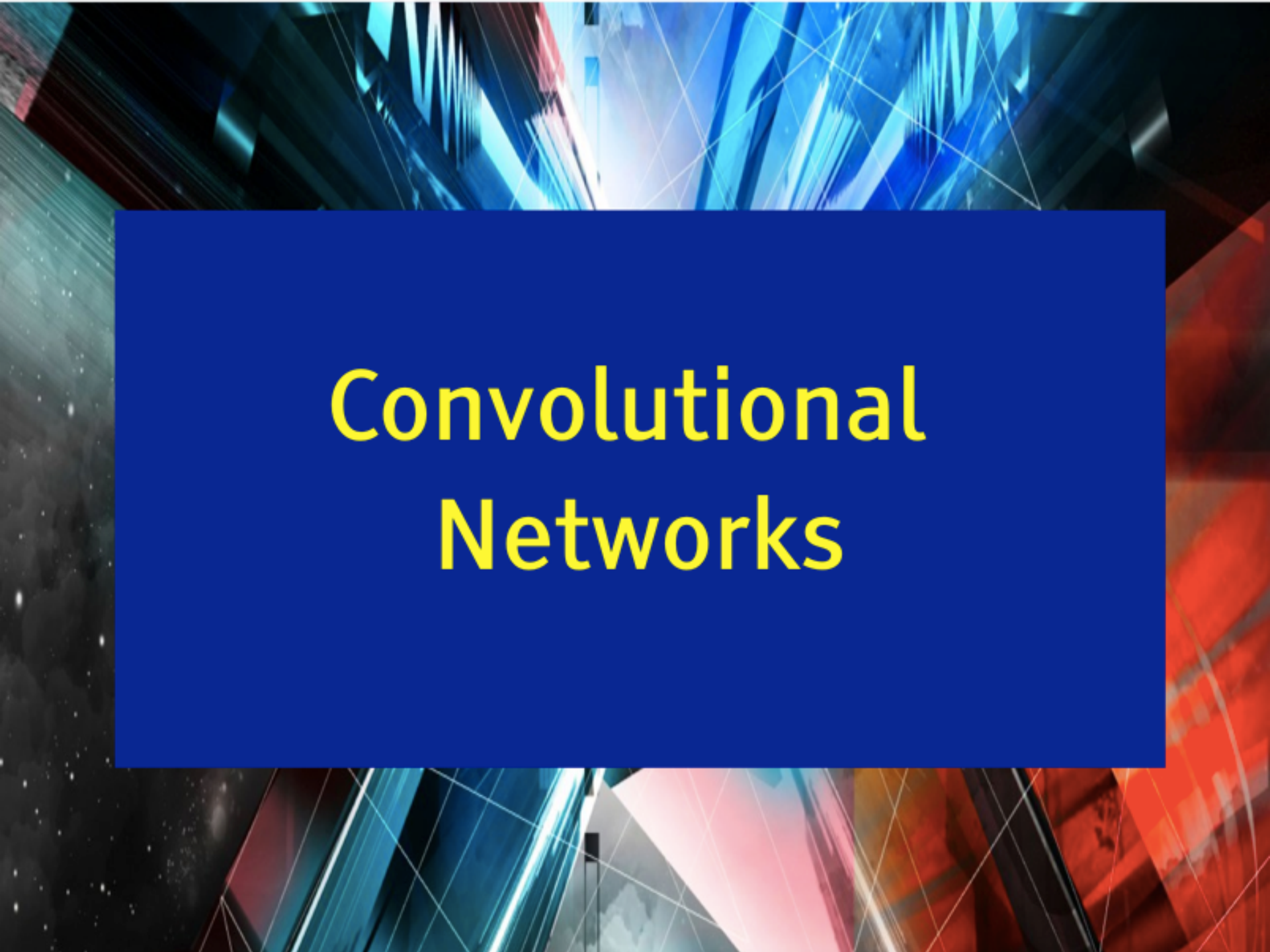
Add • $Y = X1 + X2$; $dC/dX1 = dC/dY$; $dC/dX2 = dC/dY$

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

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

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

Feature Extractor → Trainable Classifier

- **Mainstream Pattern Recognition 9until recently)**

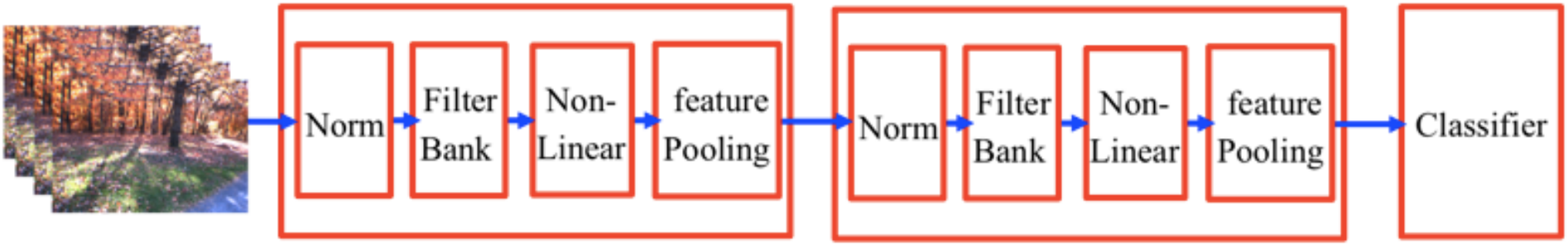
Feature Extractor → Mid-Level Features → Trainable Classifier

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

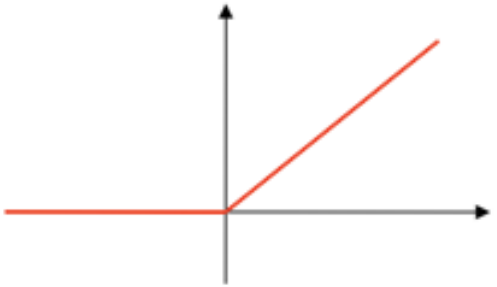
Low-Level Features → Mid-Level Features → High-Level Features → Trainable Classifier

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

Y LeCun

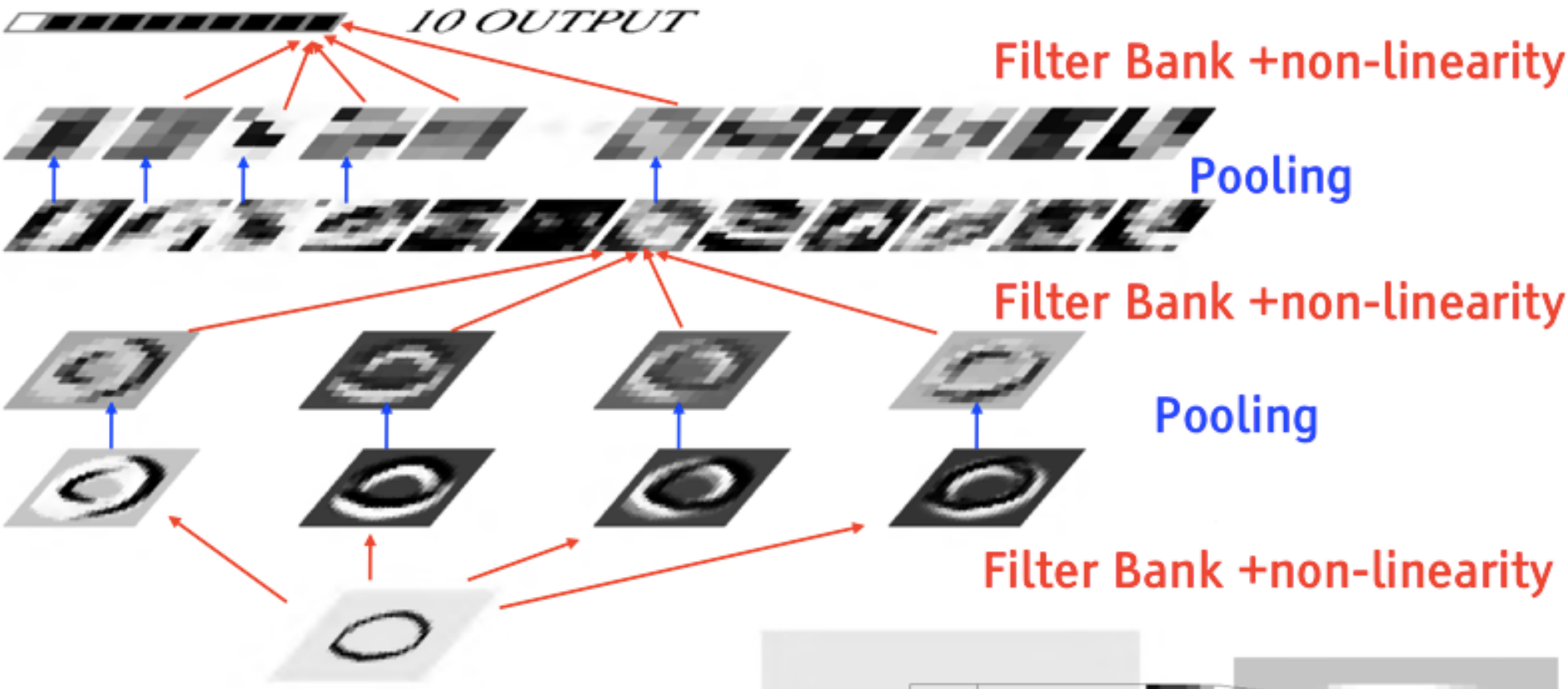


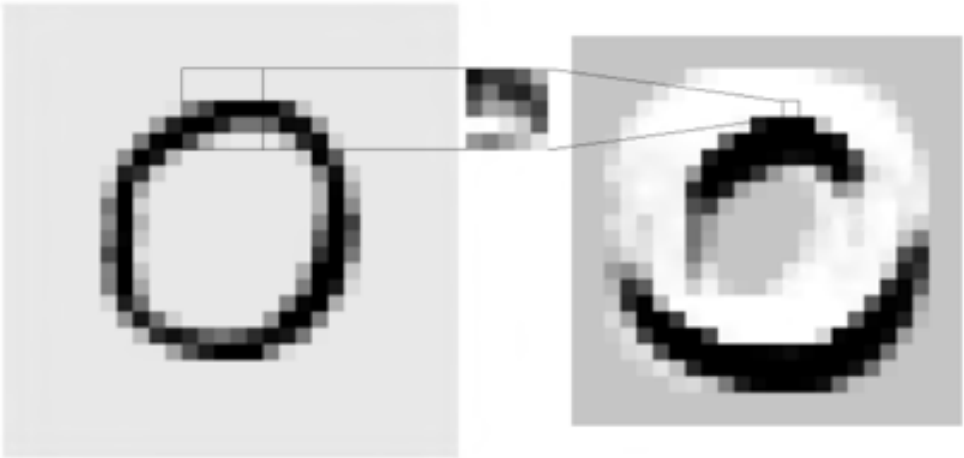
- **Normalization:** variation on whitening (optional)
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization
- **Filter Bank:** dimension expansion, projection on overcomplete basis
- **Non-Linearity:** sparsification, saturation, lateral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



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

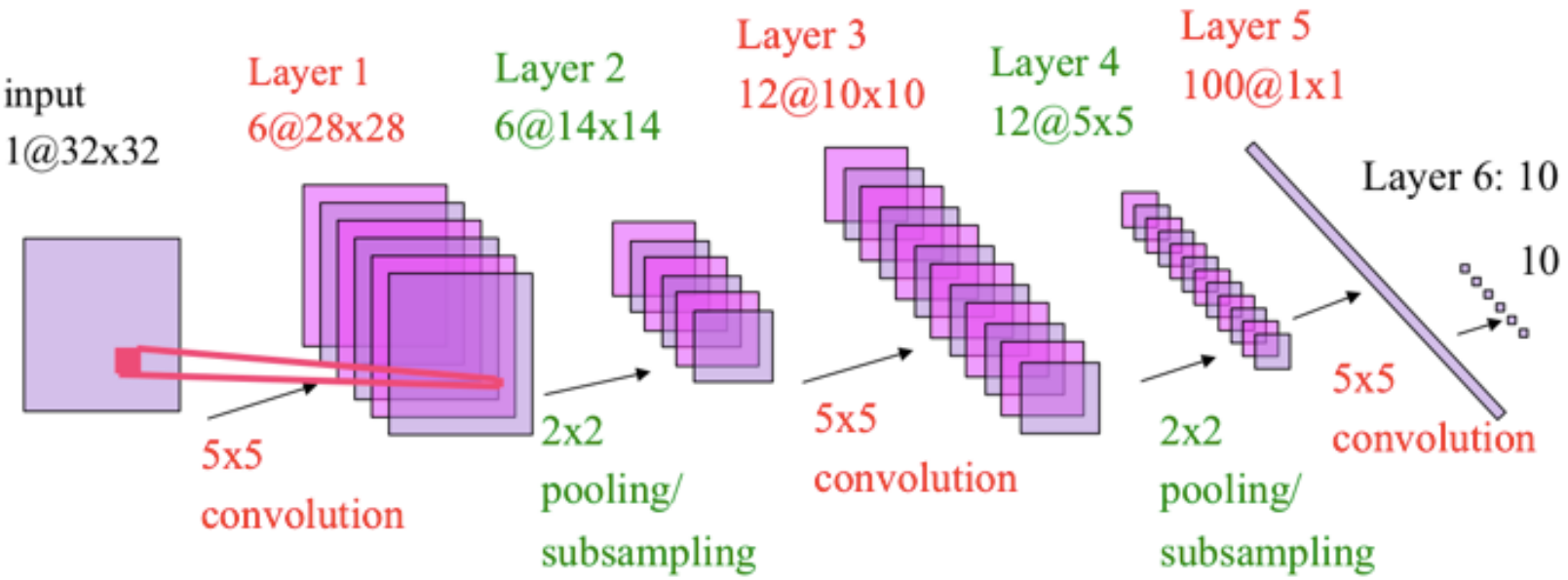
ConvNet Architecture





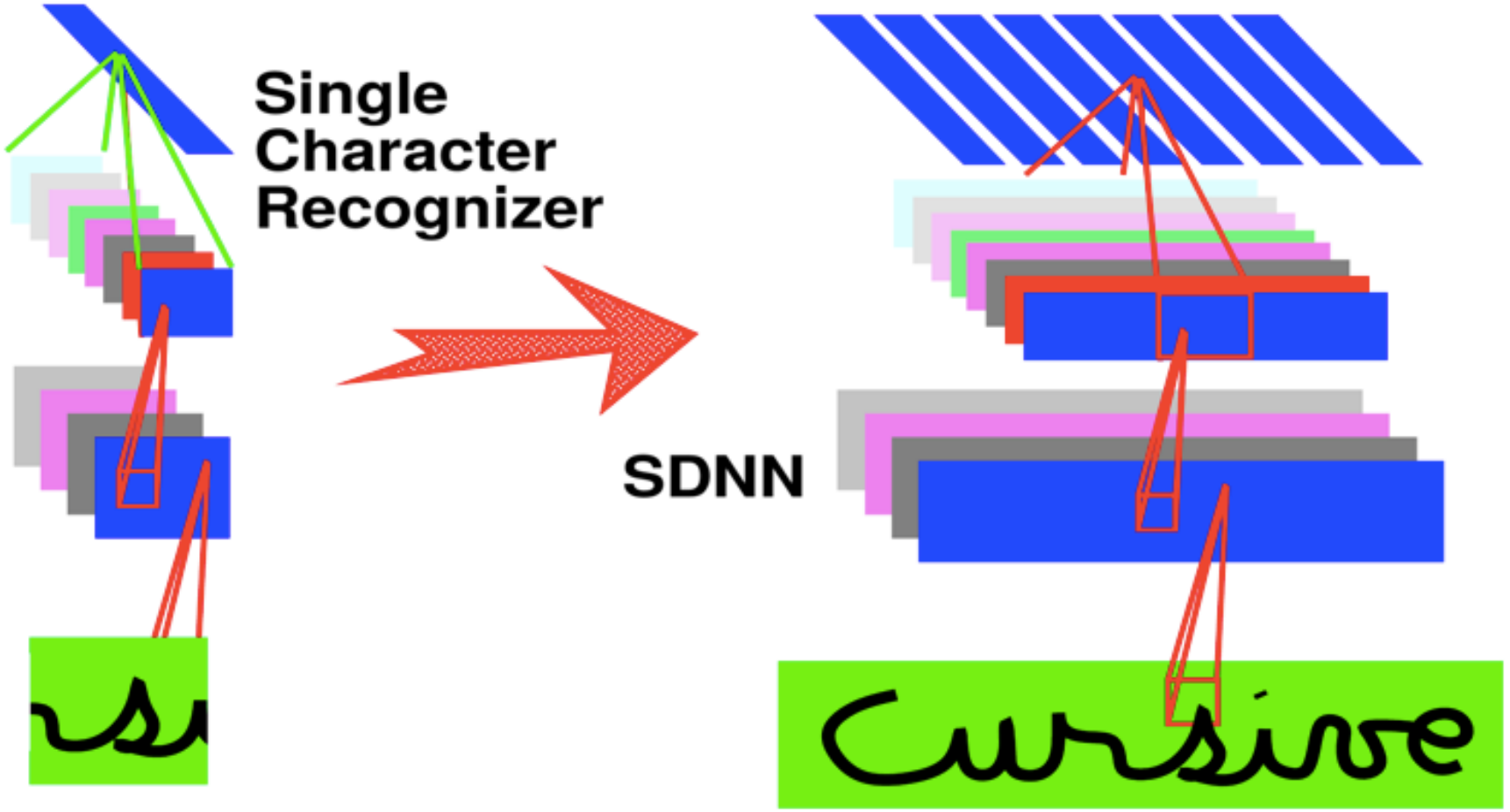
LeNet1 [LeCun et al. NIPS 1989]

- Simple ConvNet for MNIST
  [LeCun 1998]



Applying a ConvNet with a Sliding Window

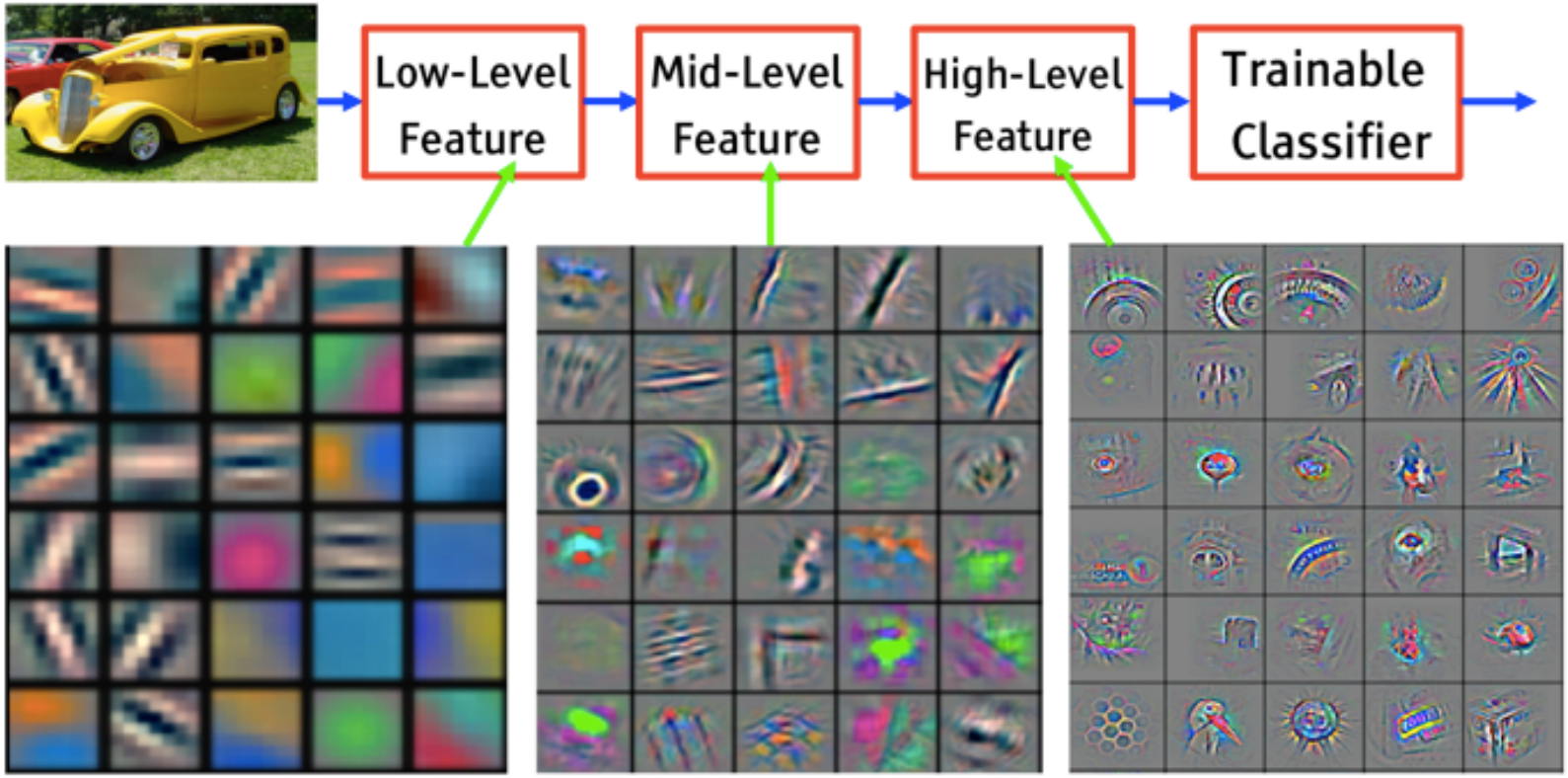
- Every layer is a convolution
- Sometimes called "fully convolutional nets"
- There is no such thing as a "fully connected layer"



Why Multiple Layers? The World is Compositional

Y LeCun

- 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



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.

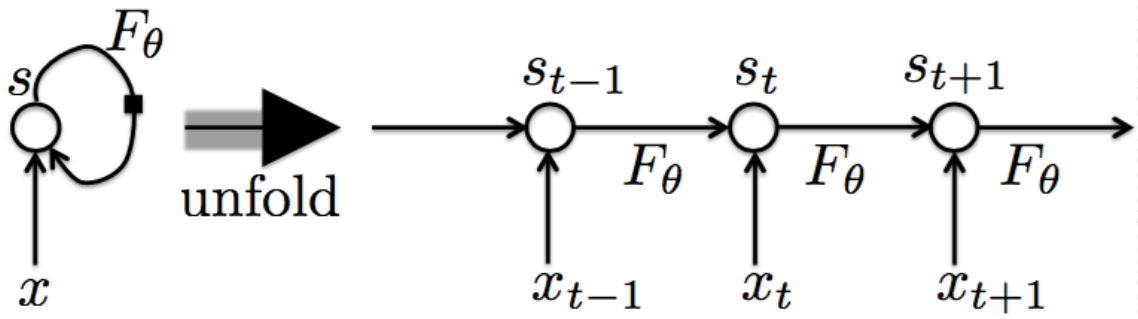
- 1D ConvNets: sequential signals, text
  - Text Classification
  - Musical Genre Recognition
  - Acoustic Modeling for Speech Recognition
  - Time-Series Prediction
- 2D ConvNets: images, time-frequency representations (speech and audio)
  - Object detection, localization, recognition
- 3D ConvNets: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

Recurrent Neural Networks (RNN)

Recurrent Neural Networks

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

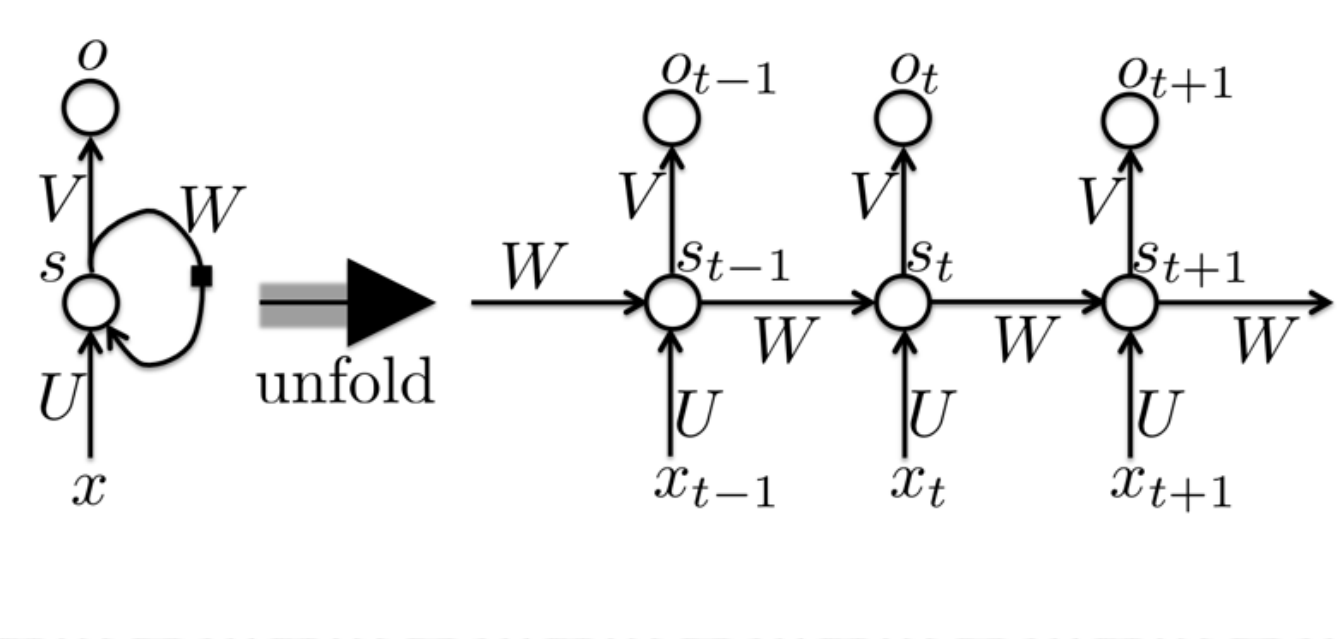
$$s_t = F_\theta(s_{t-1}, x_t)$$



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

Recurrent Neural Networks

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



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

Recurrent Neural Networks

• The RNN gradient is a product of Jacobian matrices, each associated with a step in the forward computation.

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

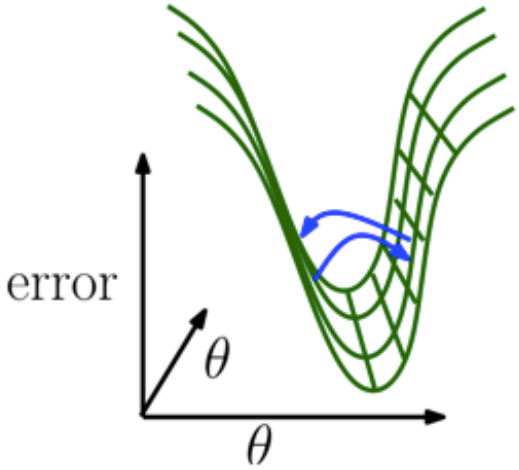
$$\frac{\partial L}{\partial s_t} = \frac{\partial L}{\partial s_T}\frac{\partial s_T}{\partial s_{T-1}} \cdots \frac{\partial s_{t+1}}{\partial s_t}$$

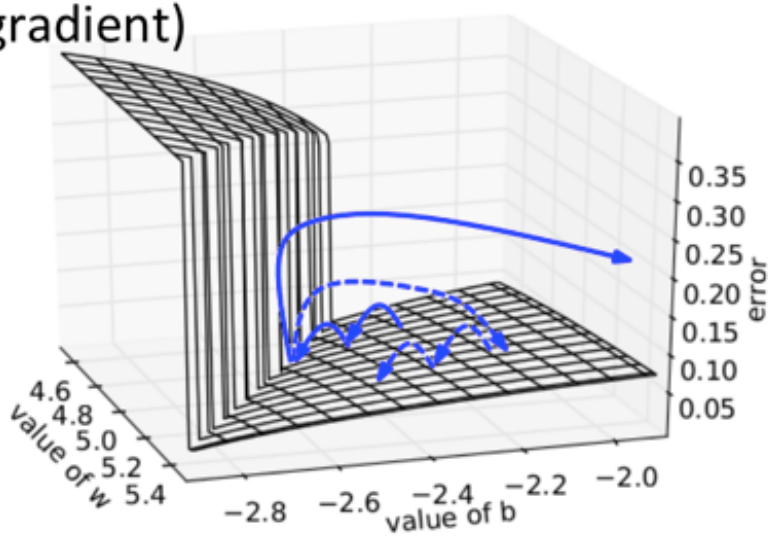
Problems:

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

RNN Tricks

- 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)

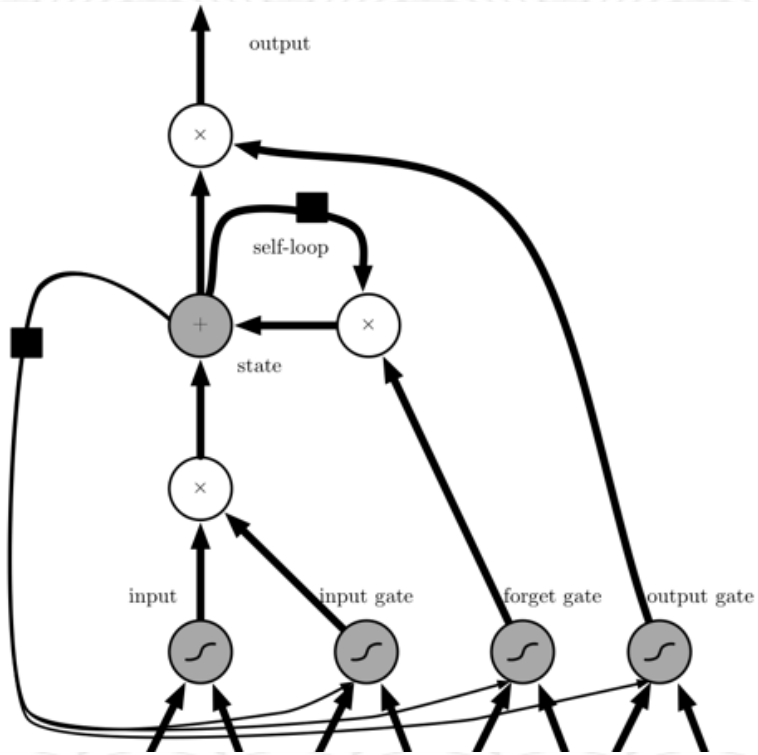




Gated Recurrent Units & LSTM

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**
- GRU light-weight version




Backprop
(modular approach)
In Practice

The Convergence of Gradient Descent

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

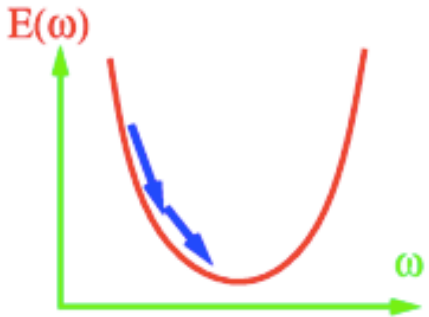
gradient of objective function

weight vector

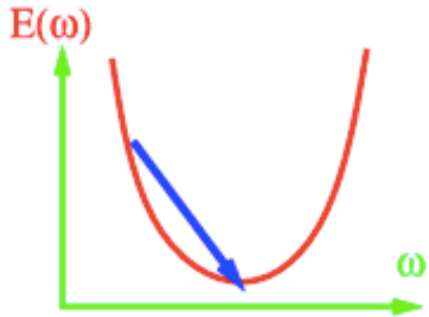
learning rate

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

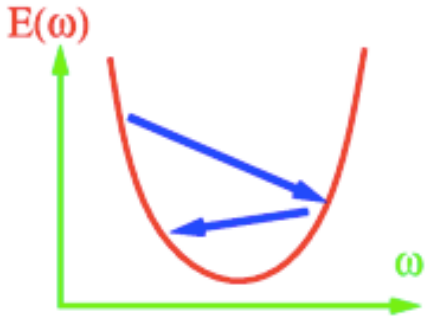
$$\eta_{opt} = \left(\frac{\partial^2 E}{\partial \omega^2} \right)^{-1}$$



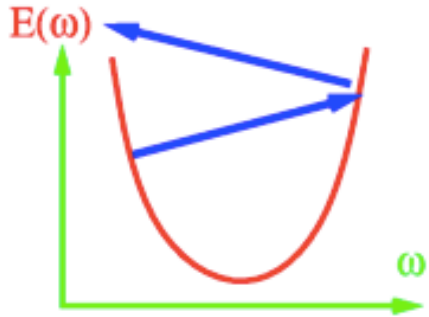
$\eta < \eta_{opt}$



$\eta = \eta_{opt}$



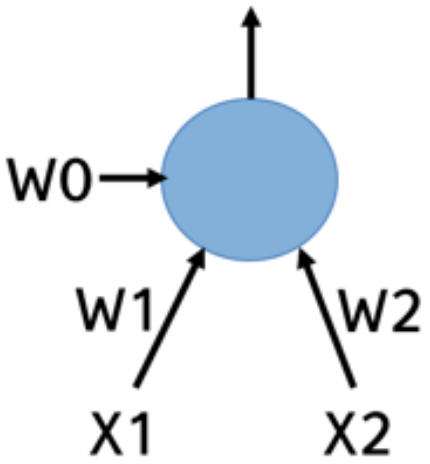
$\eta > \eta_{opt}$

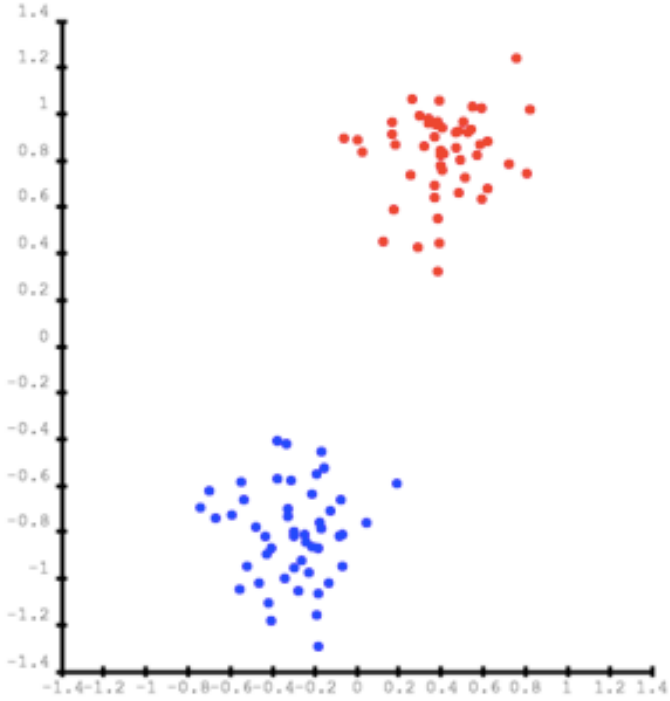


$\eta > 2\,\eta_{opt}$

Let's Look at a single linear unit

- Single unit, 2 inputs
- Quadratic loss
  - $E(W) = 1/p \sum_p (Y - W \cdot X_p)^2$
- 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:

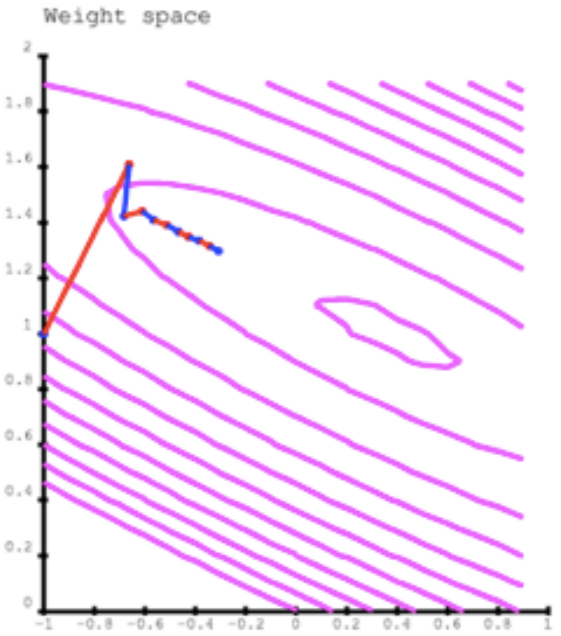
$\eta = 1.5$

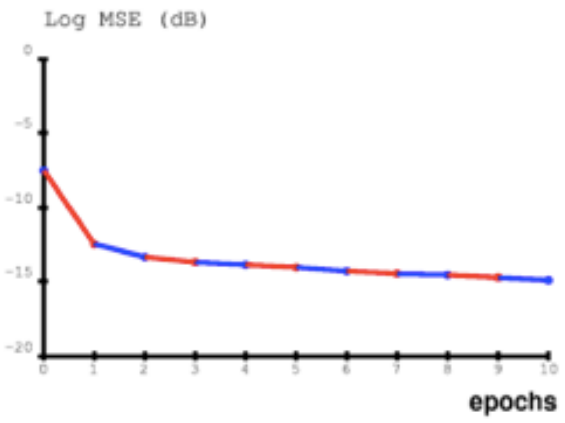
Hessian largest eigenvalue:

$\lambda_{max} = 0.84$

Maximum admissible Learning rate:

$\eta_{max} = 2.38$





Batch Gradient, large learning rate

Learning rate:

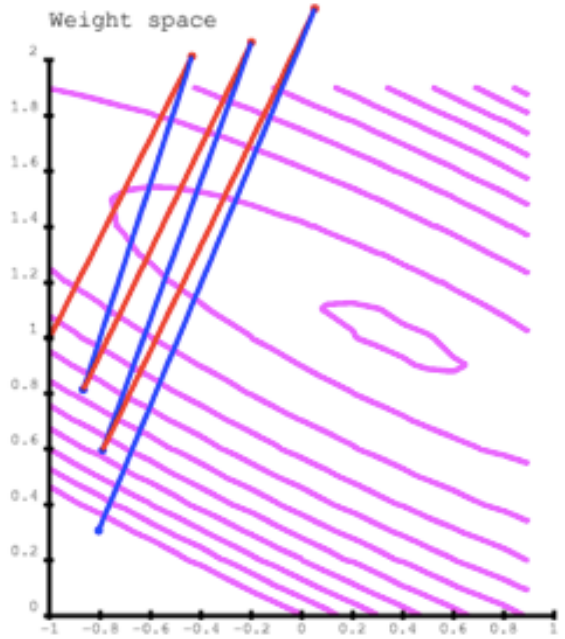
$\eta = 2.5$

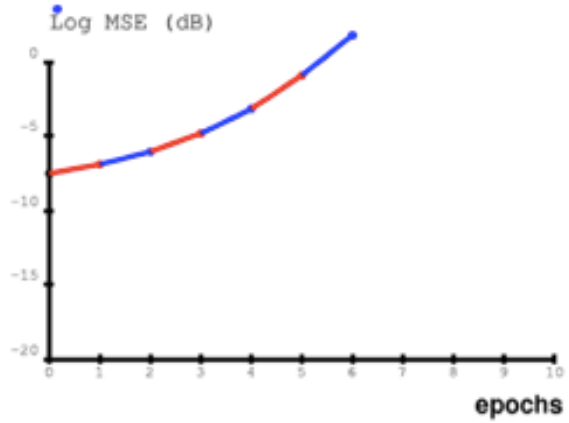
Hessian largest eigenvalue:

$\lambda_{max} = 0.84$

Maximum admissible Learning rate:

$\eta_{max} = 2.38$





Convergence is Slow When Hessian has Different Eigenvalues

- Batch Gradient, small learning rate

Learning rate:

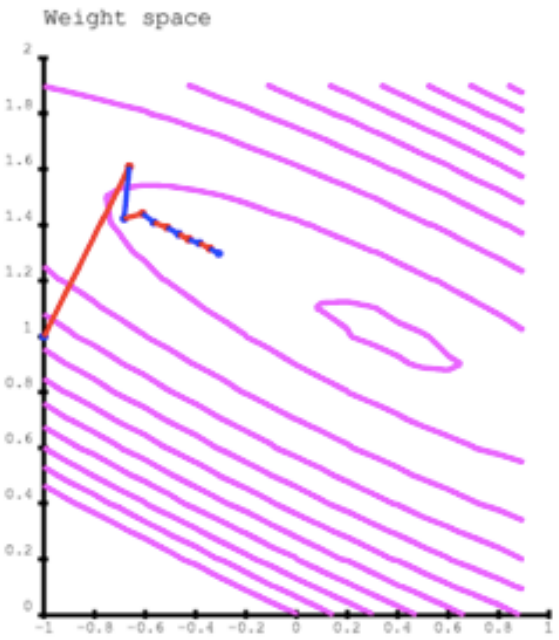
$\eta = 1.5$

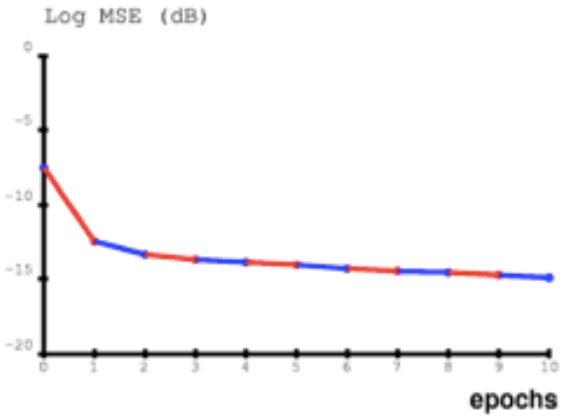
Hessian largest eigenvalue:

$\lambda_{max} = 0.84$

Maximum admissible Learning rate:

$\eta_{max} = 2.38$





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

Learning rate:

$\eta = 0.2$

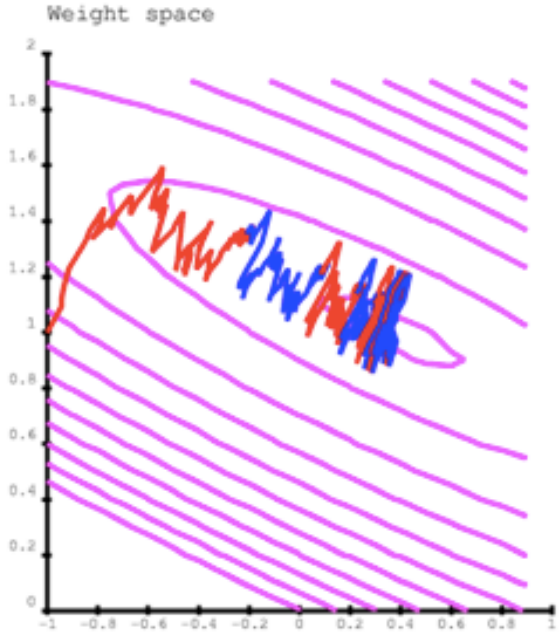
(equivalent to a batch learning rate of 20)

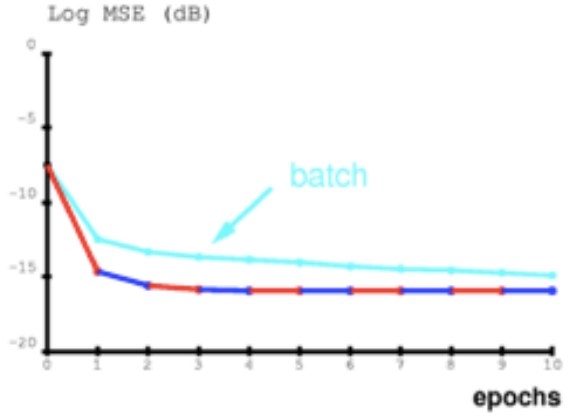
Hessian largest eigenvalue:

$\lambda_{max} = 0.84$

Maximum admissible Learning rate (for batch):

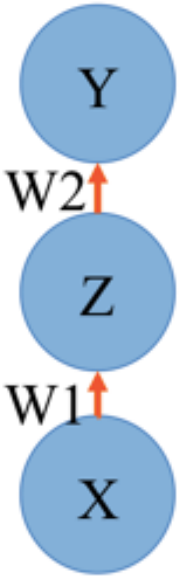
$\eta_{max} = 2.38$

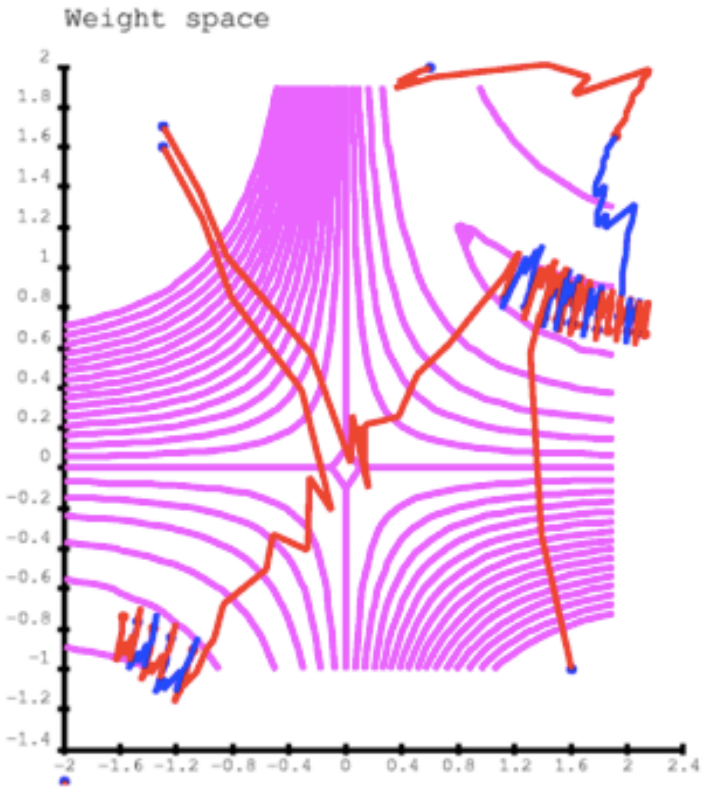


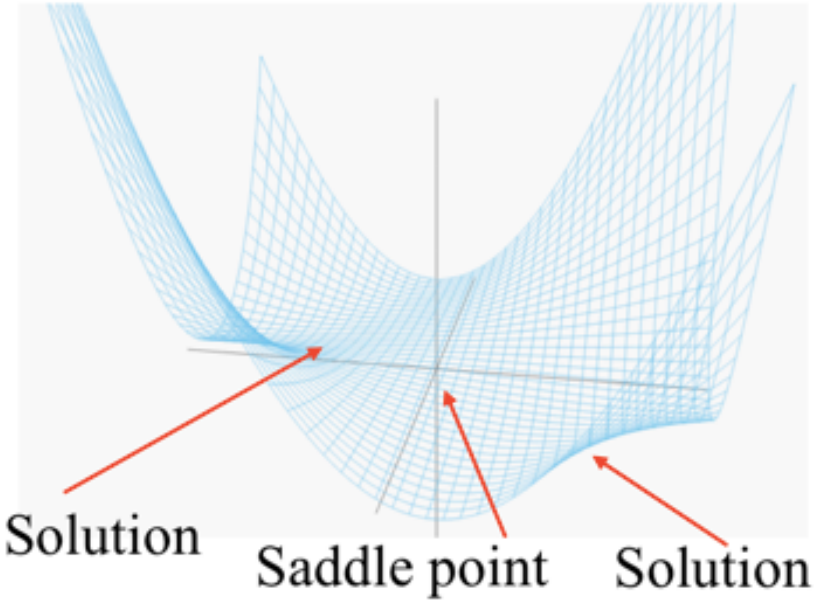


Multilayer Nets Have Non-Convex Objective Functions

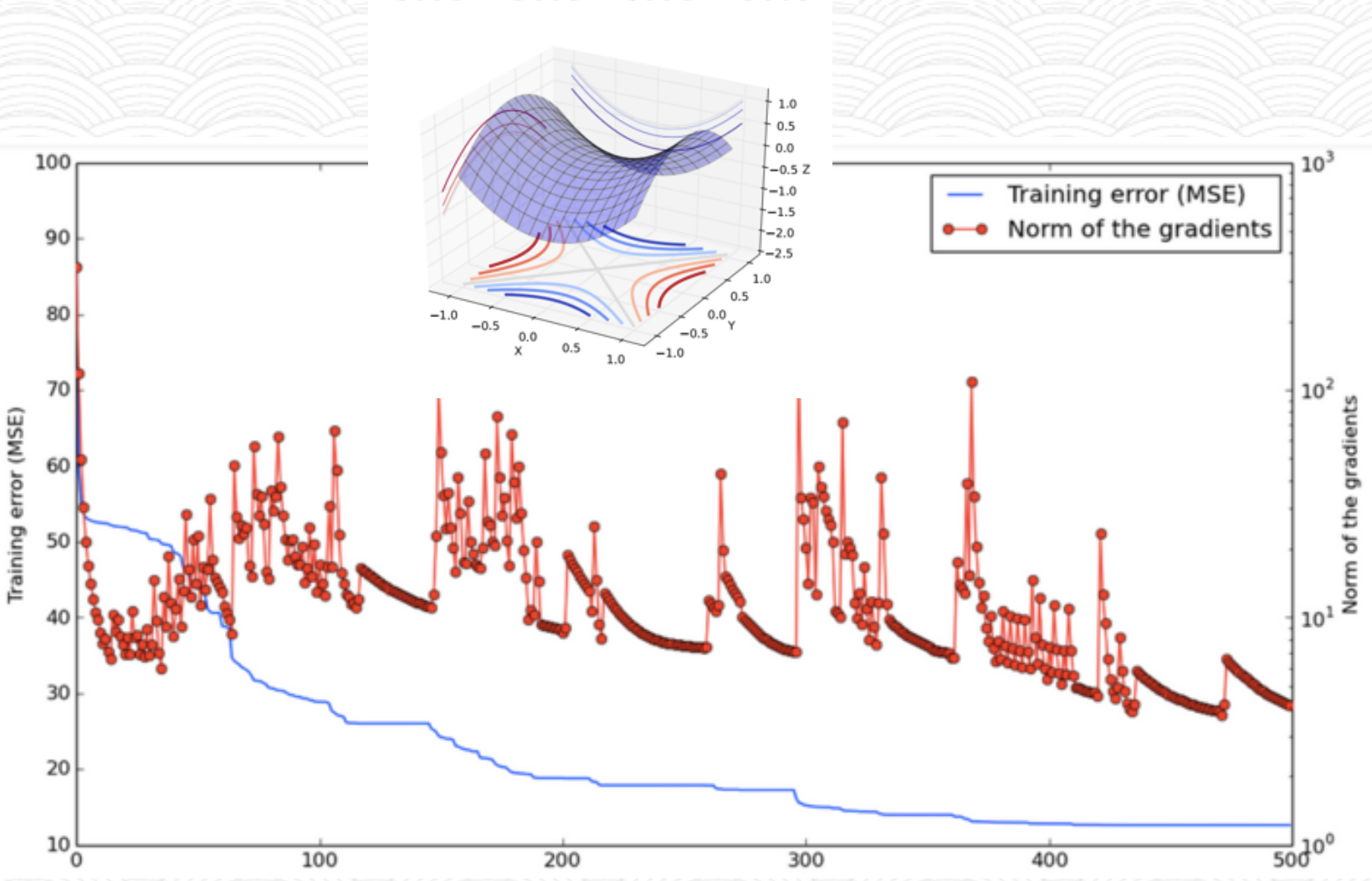
- 1-1-1 network
  - Y = W1*W2*X
- trained to compute the identity function with quadratic loss
  - Single sample X=1, Y=1 L(W) = (1-W1*W2)^2
- Solution: W2 = 1/W2 hyperbola.







Saddle Points During Training

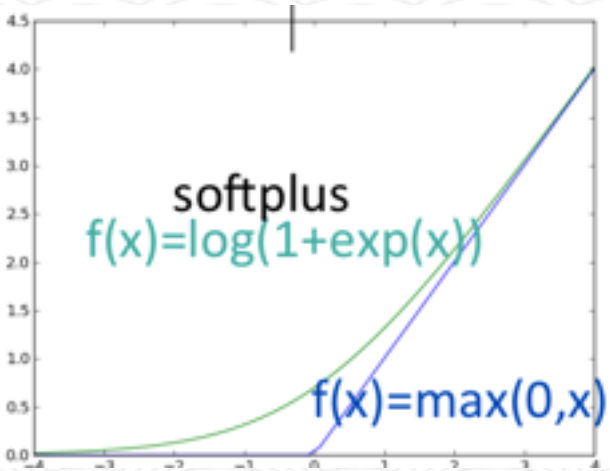


Piecewise Linear Nonlinearity

Absolute value rectification works better than tanh in lower layers of convnet.

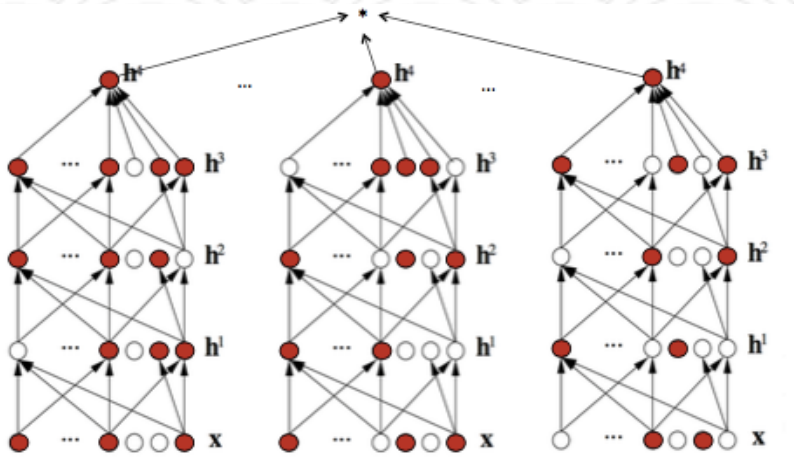
Using a rectifier non- linearity (ReLU) instead of tanh of softplus allows for the first time to train very deep supervised networks without the need for unsupervised pre-training; was biologically motivated

Rectifiers one of the crucial ingredients in ImageNet breakthrough



Dropout Regularizer: Super-Efficient Bagging

- **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)



Batch Normalization

- Regulaizes & helps to train

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

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

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

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

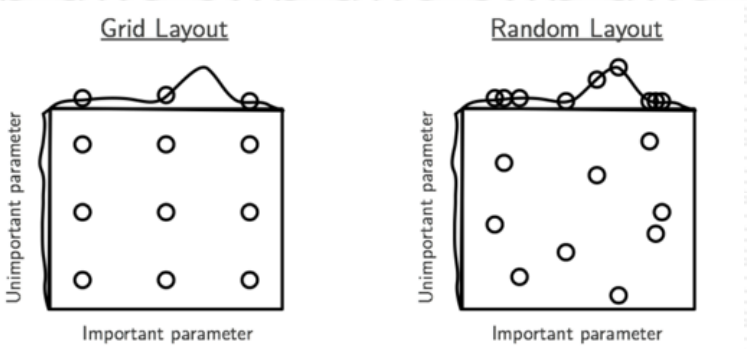
$$\mathbf{y} = \phi(BN(\mathbf{W}\mathbf{x}))$$

Early Stopping

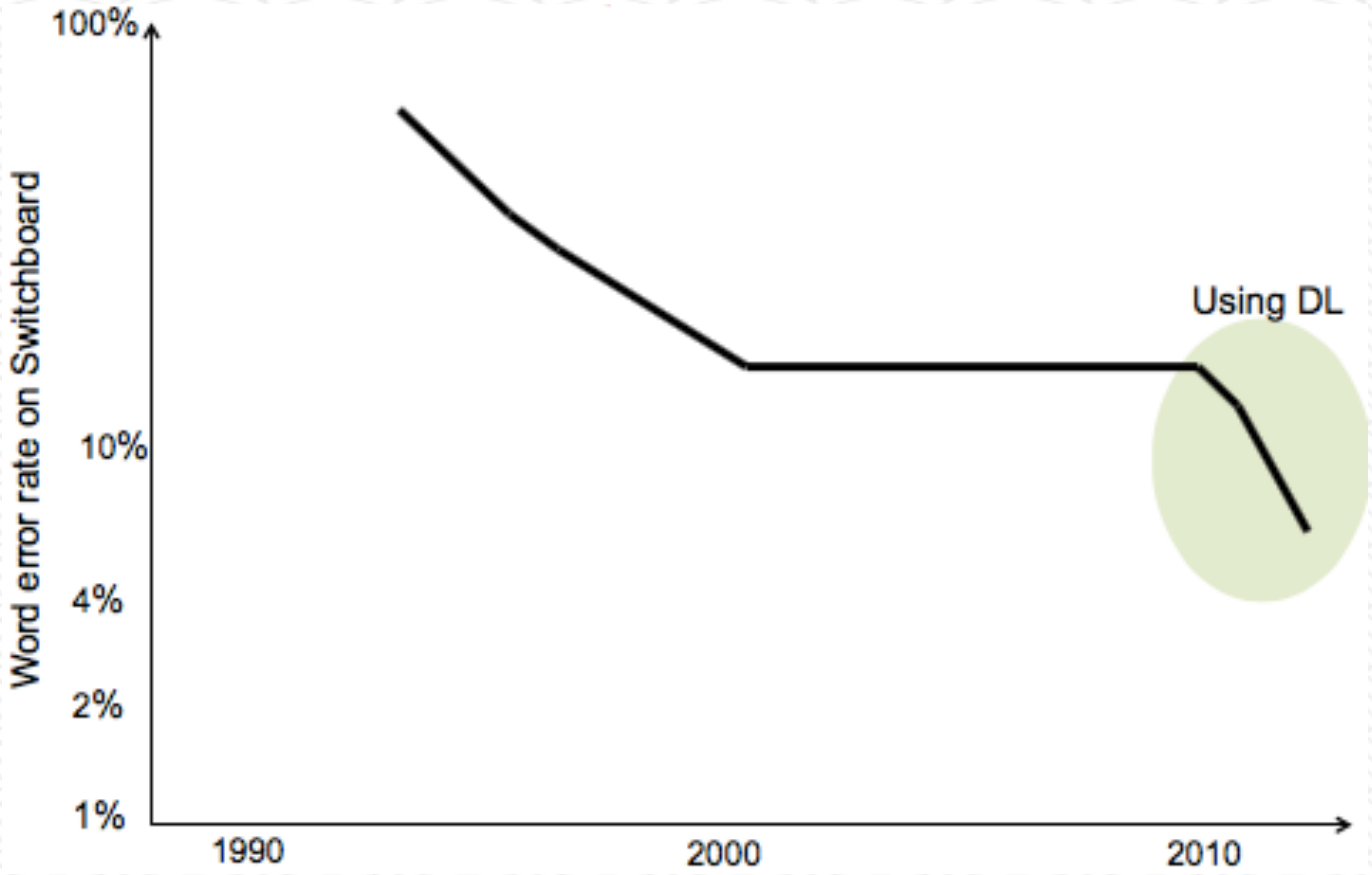
- Beautiful **FREE LUNCH** (no need to launch many different training runs for each value of hyper-parameter for #iterations)
- Monitor validation error during training (after visiting # of training examples = a multiple of validation set size)
- Keep track of parameters with best validation error and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

Sampling Hyperparameters

- Common approach: manual + grid search
- Grid search over hyperparameters: simple & wasteful
- Random search: simple & efficient
  - Independently sample each HP,
  - More convenient: ok to early-stop, continue further, etc.



Speech

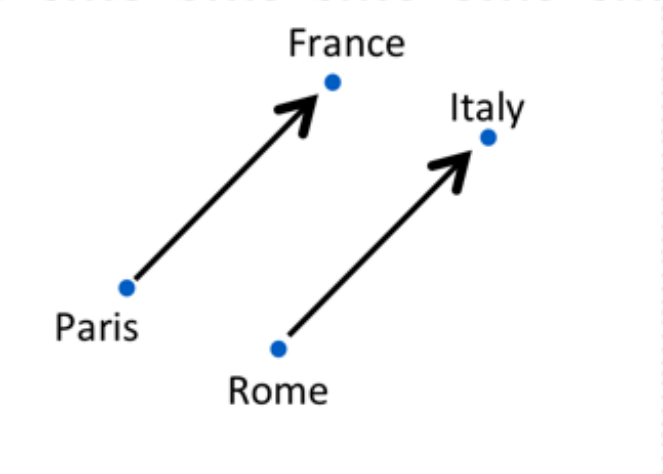

100%
10%
4%
2%
1%
Word error rate on Switchboard
1990
2000
2010
Using DL

Natural Language Representations

Neural word embeddings: visualization directions = Learned Attributes

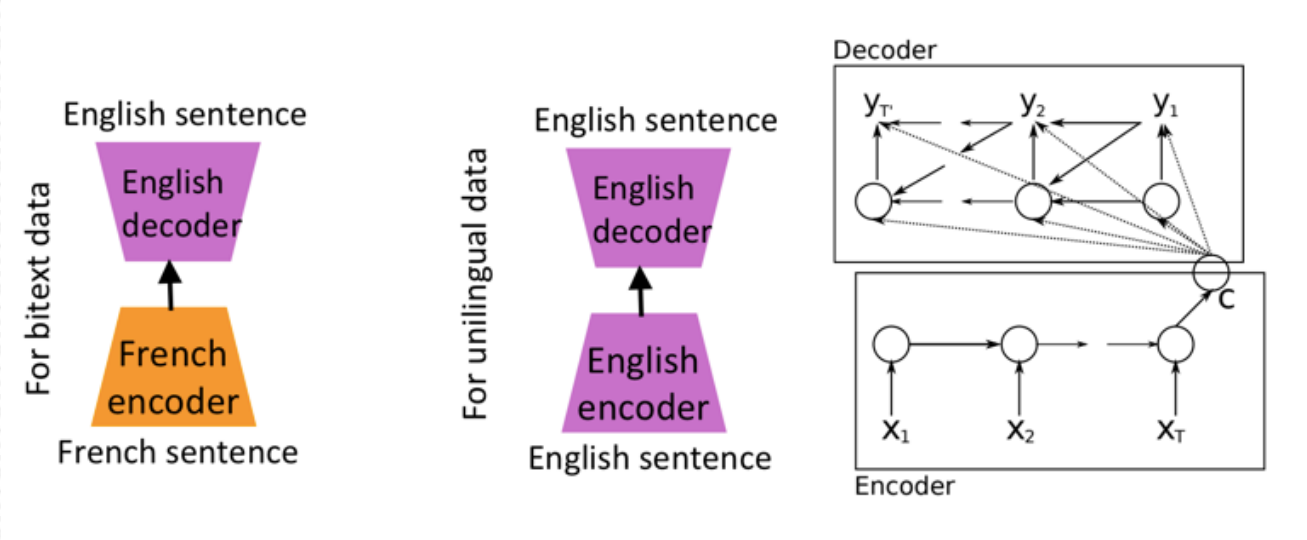
Analogical Representations for Free Text

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



Encoder-Decoder Framework

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



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

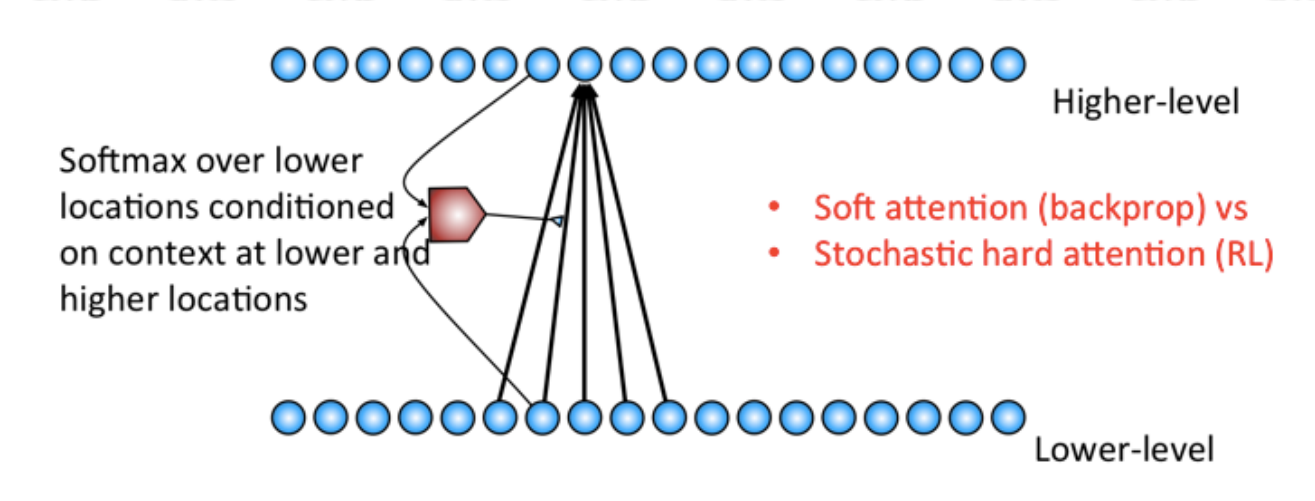


Image-to-Text: Caption Generation with Attention

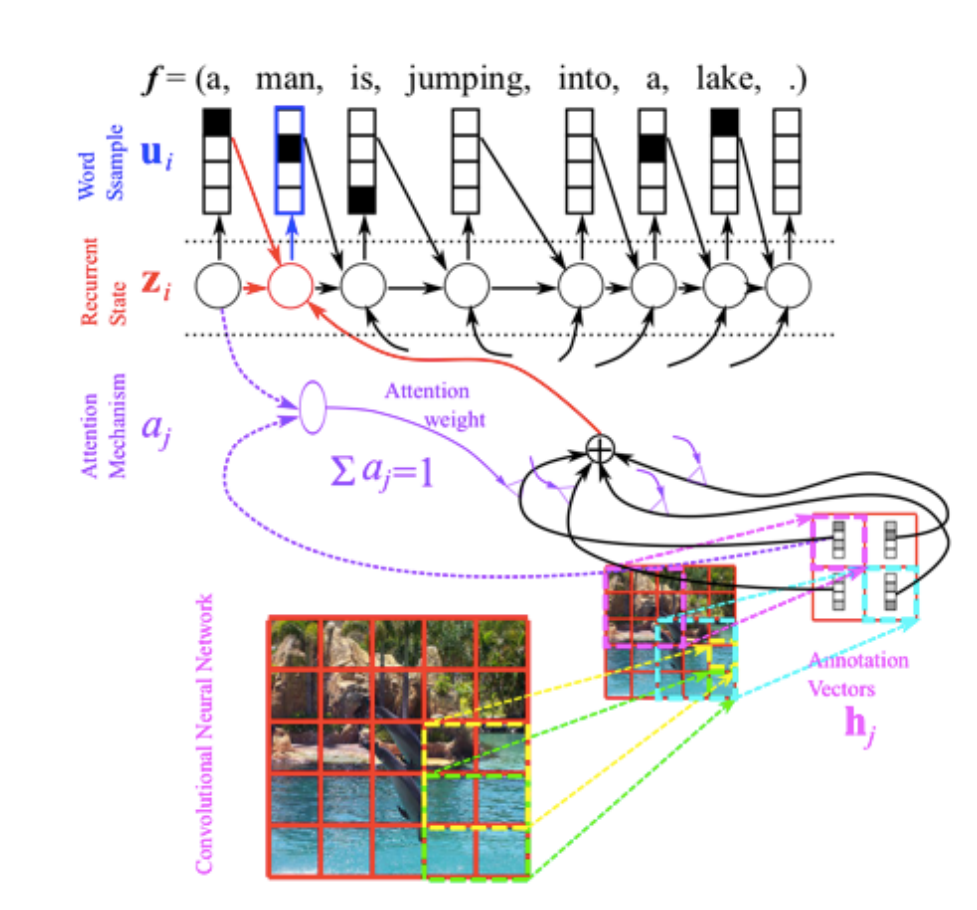
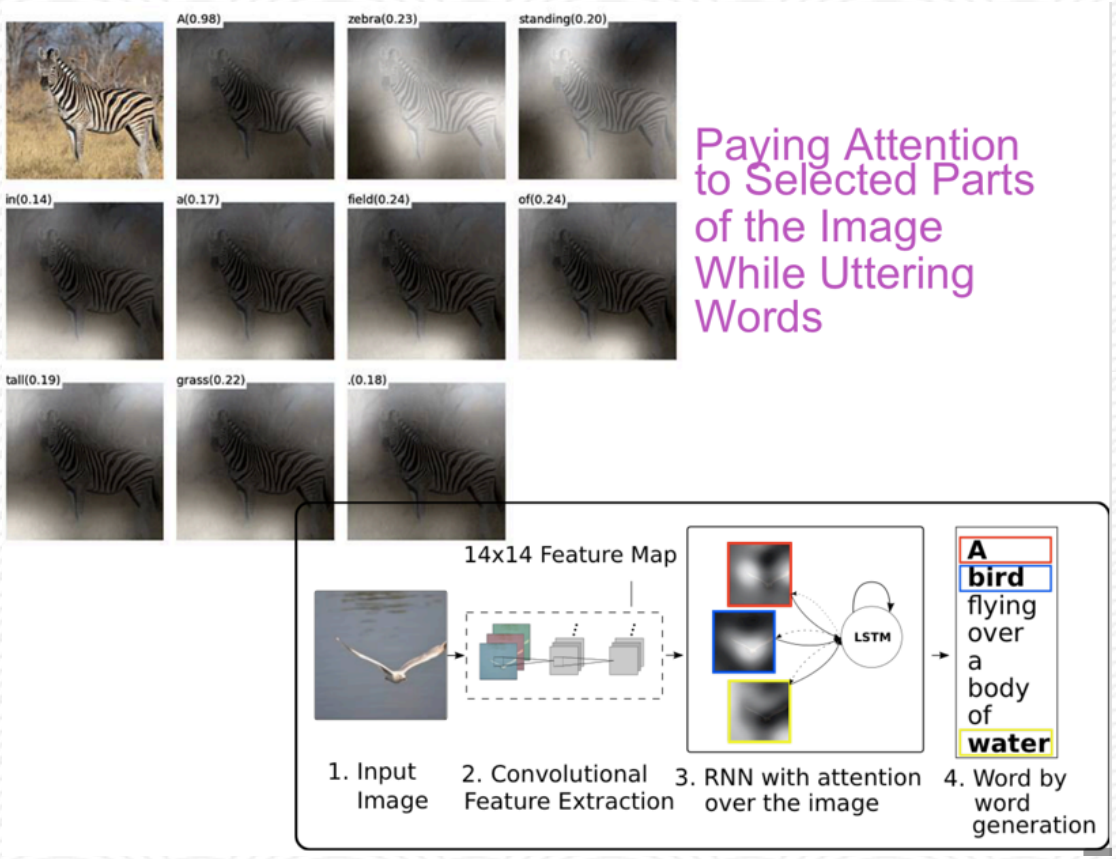


Image-to-Text: Caption Generation with Attention



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.

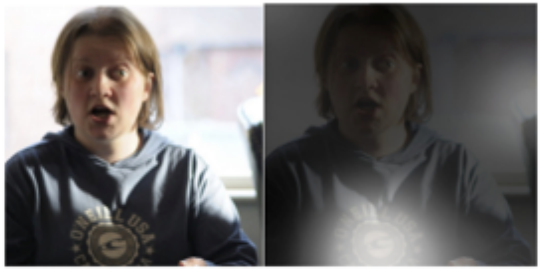
A little girl sitting on a bed with a teddy bear.

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.

Memory Networks Enable REASONING

Y LeCun

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

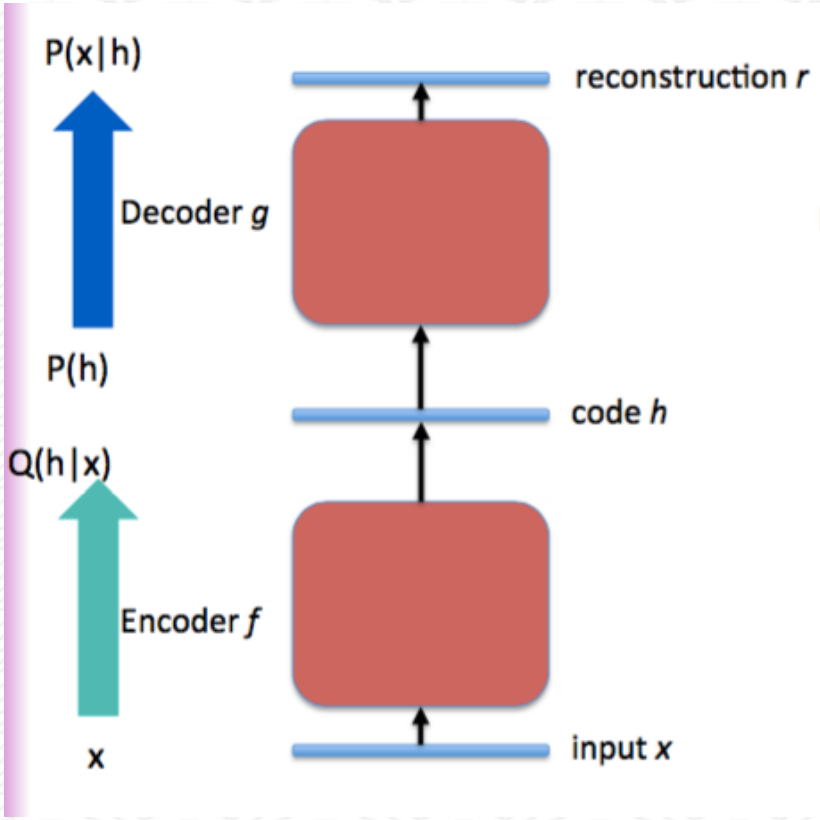
(Weston, Chopra, Bordes 2014)

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.

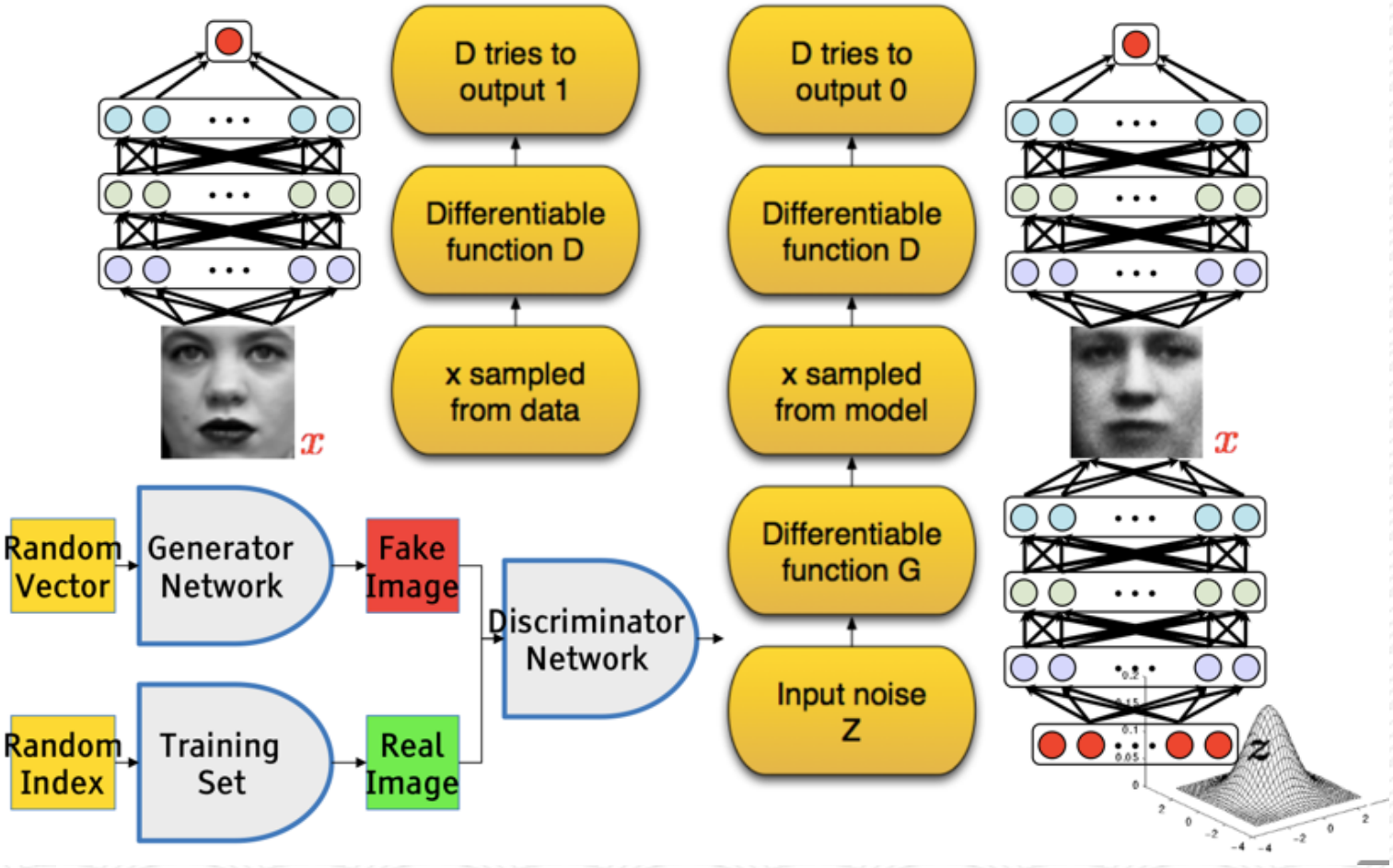
Unsupervised Representation Learning

Autoencoders

- RBM
- (Denoising) Autoencoder



GAN: Generative Adversarial Networks



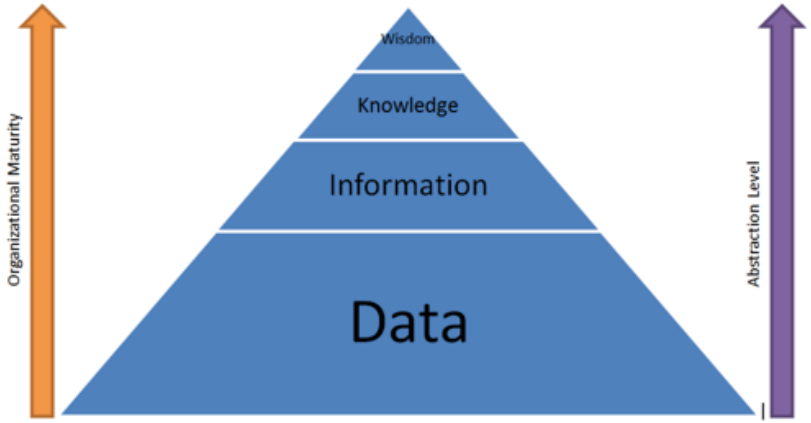
GAN: Interpolating in Latent Space

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

125

Insights

- 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



Hands On

Hands On

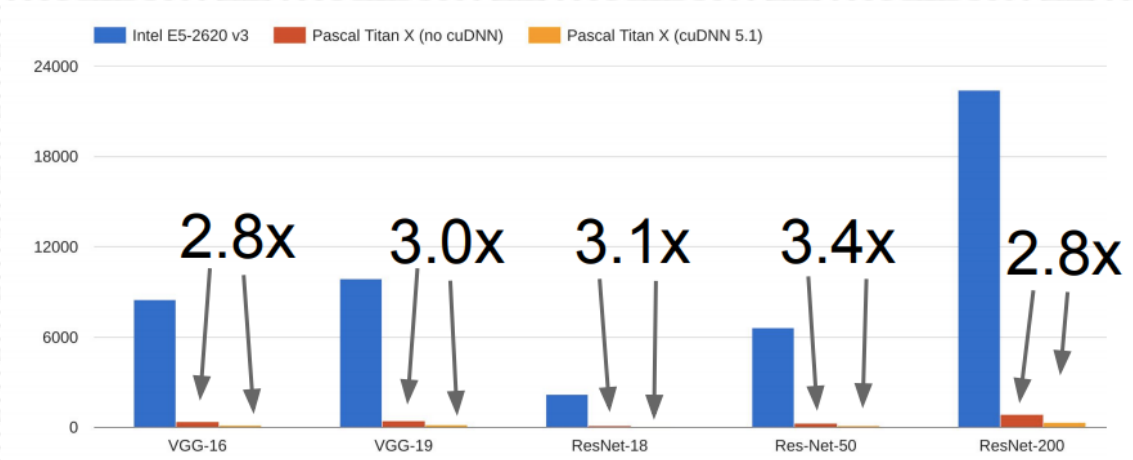
- Login to NUS SoC VPN with your NUSNET Account

- ssh wesst@kpe.d1.comp.nus.edu.sg

Password: kerasuser

- copy folder demo to a new folder rename demo_<yourname>

CPU or GPU?


Intel E5-2620 v3
Pascal Titan X (no cuDNN)
Pascal Titan X (cuDNN 5.1)
24000
18000
12000
6000
0
2.8x
3.0x
3.1x
3.4x
2.8x
VGG-16
VGG-19
ResNet-18
Res-Net-50
ResNet-200

Frameworks

This year ...

Caffe (UC Berkeley) → Caffe2 (Facebook)

Torch (NYU / Facebook) → PyTorch (Facebook)

Theano (U Montreal) → TensorFlow (Google)

Paddle (Baidu)

CNTK (Microsoft)

MXNet (Amazon)

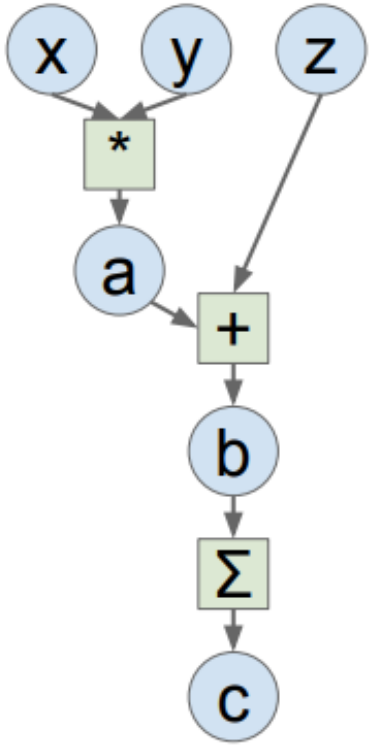
Developed by U Washington, CMU, MIT, Hong Kong U, etc but main framework of choice at AWS

And others...

Computational Graphs

Numpy

```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



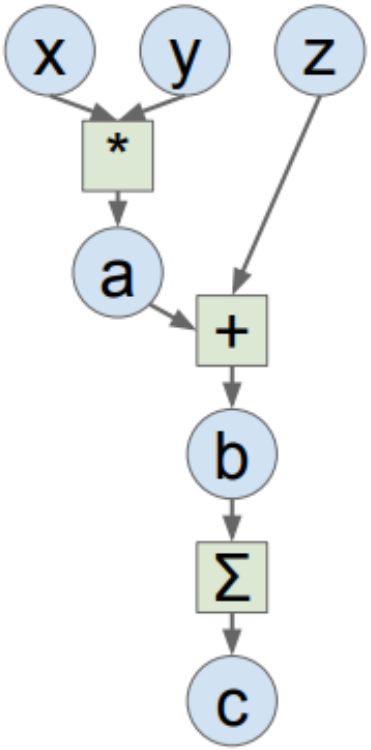
Problems:

- Can't run on GPU
- Have to compute our own gradients

Computational Graphs

Numpy

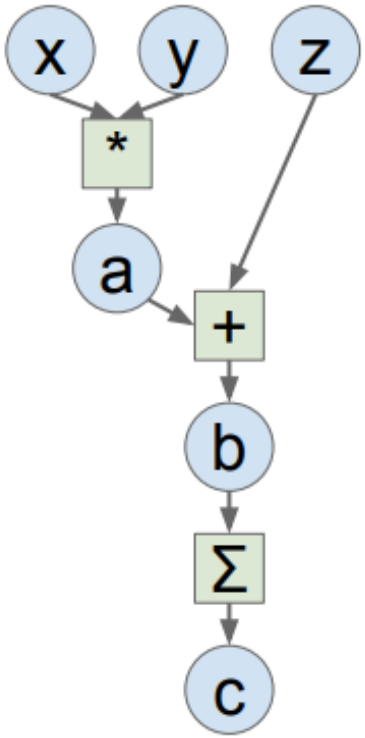
```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



TensorFlow

```
# Basic computational graph
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

a = x * y
b = a + z
c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

Computational Graphs

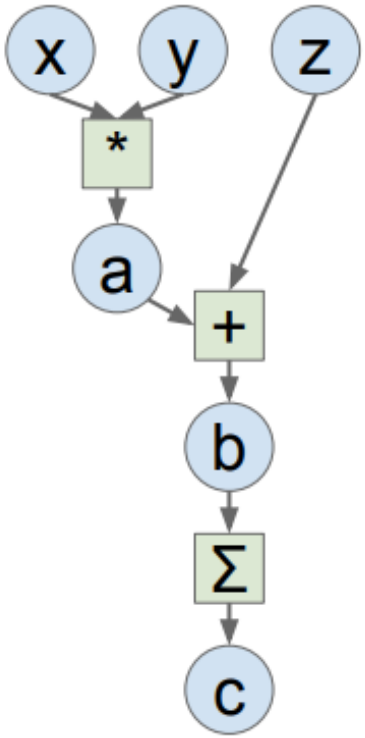


Create forward computational graph

TensorFlow

```
# Basic computational graph
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

a = x * y
b = a + z
c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

Computational Graphs



Create forward computational graph

TensorFlow

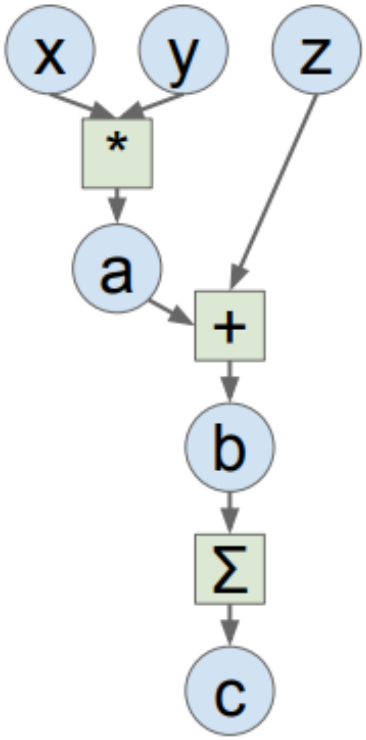
```
# Basic computational graph
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

a = x * y
b = a + z
c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

Keras: High-Level Wrapper

Define model object as a sequence of layers

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Computational Graphs



Ask TensorFlow to compute gradients

TensorFlow

```
# Basic computational graph
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

a = x * y
b = a + z
c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

Keras: High-Level Wrapper

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Train the model with a single line!

Keras: High-Level Wrapper

Define optimizer object

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Keras: High-Level Wrapper

Keras is a layer on top of TensorFlow, makes common things easy to do

(Also supports Theano backend)

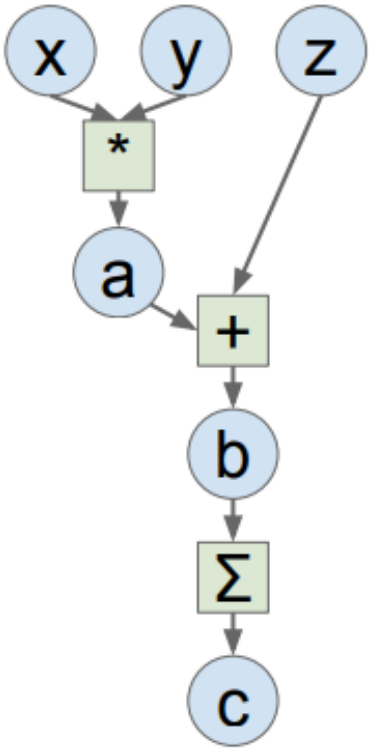
```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Keras: High-Level Wrapper

Build the model, specify loss function

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Computational Graphs

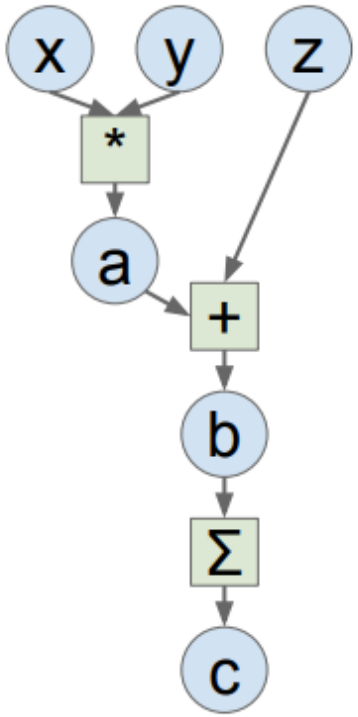


Calling c.backward() computes all gradients

PyTorch

```
import torch
from torch.autograd import Variable

N, D = 3, 4

x = Variable(torch.randn(N, D),
             requires_grad=True)
y = Variable(torch.randn(N, D),
             requires_grad=True)
z = Variable(torch.randn(N, D),
             requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()

print(x.grad.data)
print(y.grad.data)
print(z.grad.data)
```

Computational Graphs

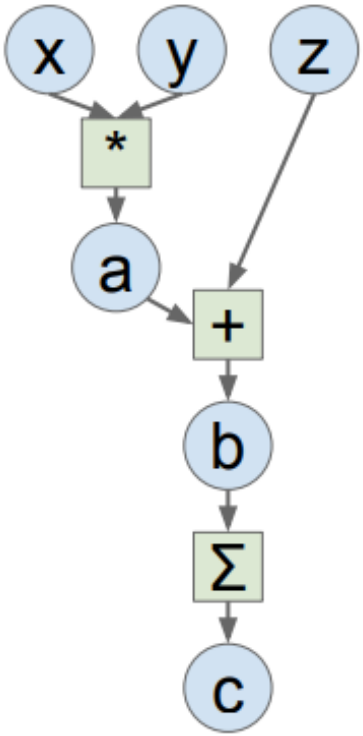


Tell TensorFlow to run on **CPU**

TensorFlow

```
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3000, 4000

with tf.device('/cpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

Computational Graphs



Run on GPU by casting to .cuda()

PyTorch

```
import torch
from torch.autograd import Variable

N, D = 3, 4

x = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
y = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
z = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()

print(x.grad.data)
print(y.grad.data)
print(z.grad.data)
```

If you are an NUS Undergrad:

Bid for CS3244 Machine Learning on CORS

Open for all (Community Course):

CS6101: Convolutional Neural Networks for Visual Recognition

References:

NIPS DL Tutorial 2015

CS231n: Convolutional Neural Networks for Visual Recognition