Neural Networks and Deep Learning II

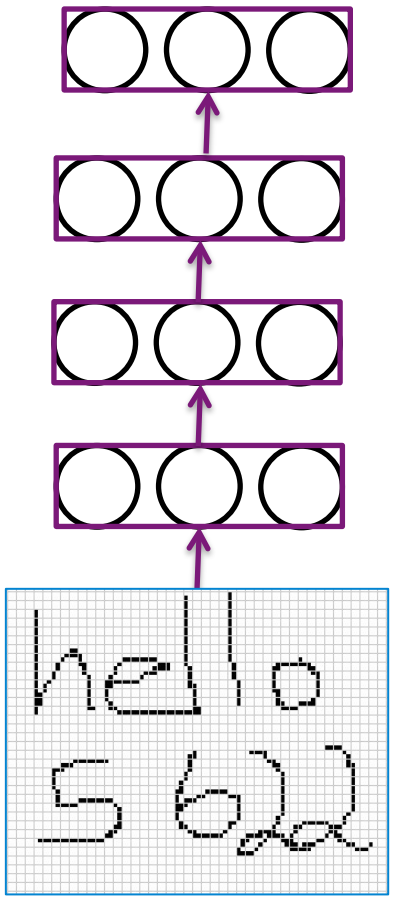
CSCI 5622

Fall 2015

Why Stop At One Hidden Layer?

E.g., vision hierarchy for recognizing handprinted text

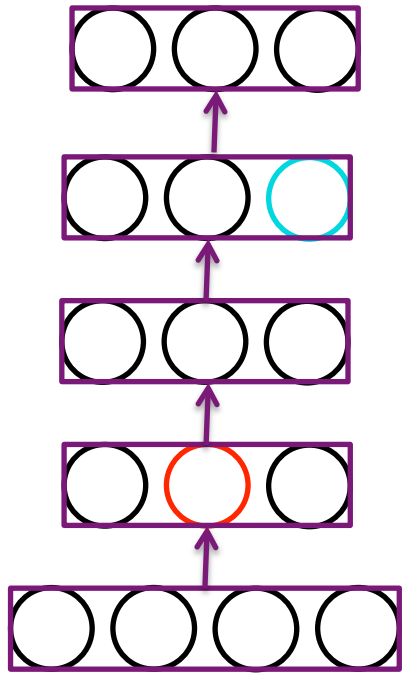
Word	output layer
Character	hidden layer 3
Stroke	hidden layer 2
Edge	hidden layer 1
Pixel	input layer



Why Deep Nets Fail 1: Credit Assignment Problem

How is a neuron in layer 2 supposed to know what it should output until all the neurons above it do something sensible?

How is a neuron in layer 4 supposed to know what it should output until all the neurons below it do something sensible?



Why Deep Nets Fail 2: Vanishing Error Gradients

With logistic activation function,

$$\Delta w_{ji} = \varepsilon \delta_j x_i \qquad \delta_j = \begin{cases} \dfrac{\partial E}{\partial y_j} y_j(1-y_j) & \text{for output unit} \\ \left(\sum_k w_{kj} \delta_k \right) y_j(1-y_j) & \text{for hidden unit} \end{cases}$$

Error gradients get squashed as they are passed back through a deep network

- $y_j(1-y_j) \leq 0.25$

Approach To Solving Both Problems (2000-2012)

Traditional method of training

Random initial weights

Alternative

Do unsupervised learning layer by layer to get weights in a sensible configuration for the statistics of the input.

If these initial weights are sensible, then

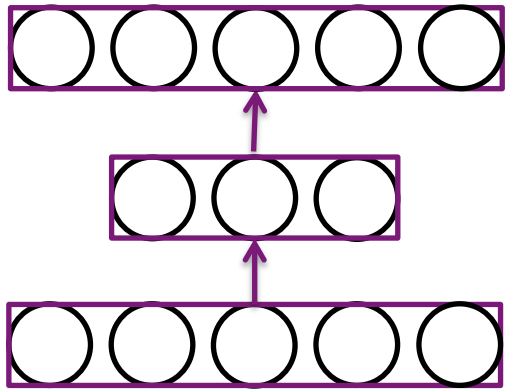
- credit assignment problem will be mitigated
- gradients should convey information all the way down

Autoencoder Networks

Self-supervised training procedure

Given a set of input vectors (no target outputs)

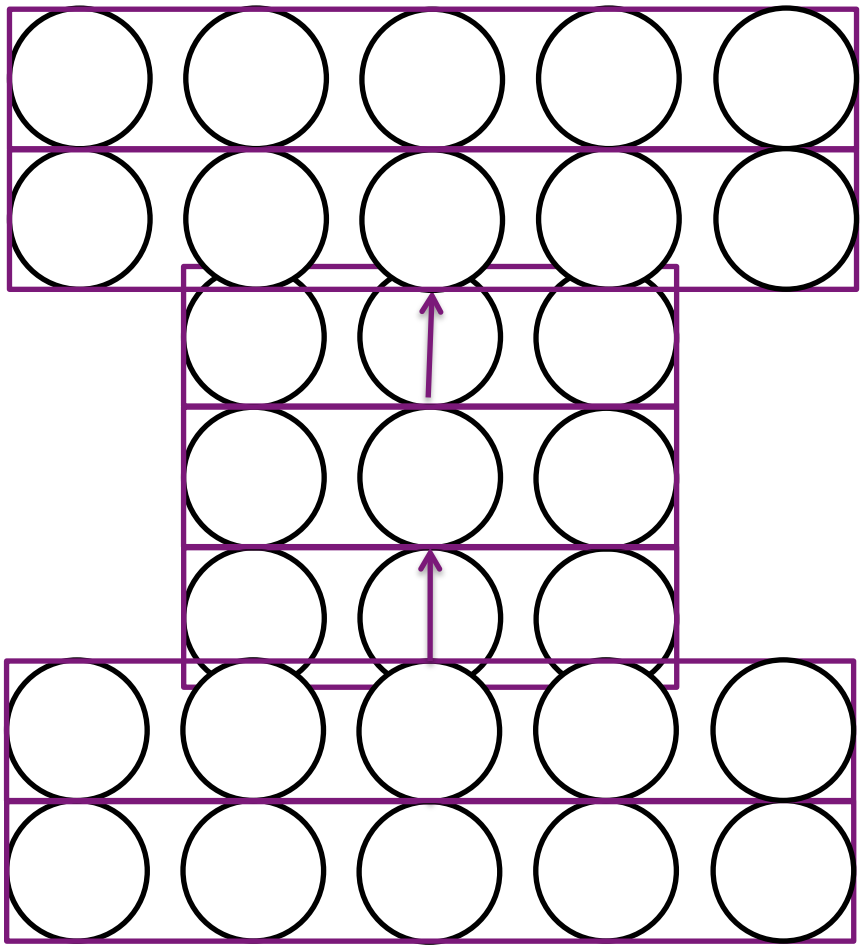
Map input back to itself via a hidden layer bottleneck



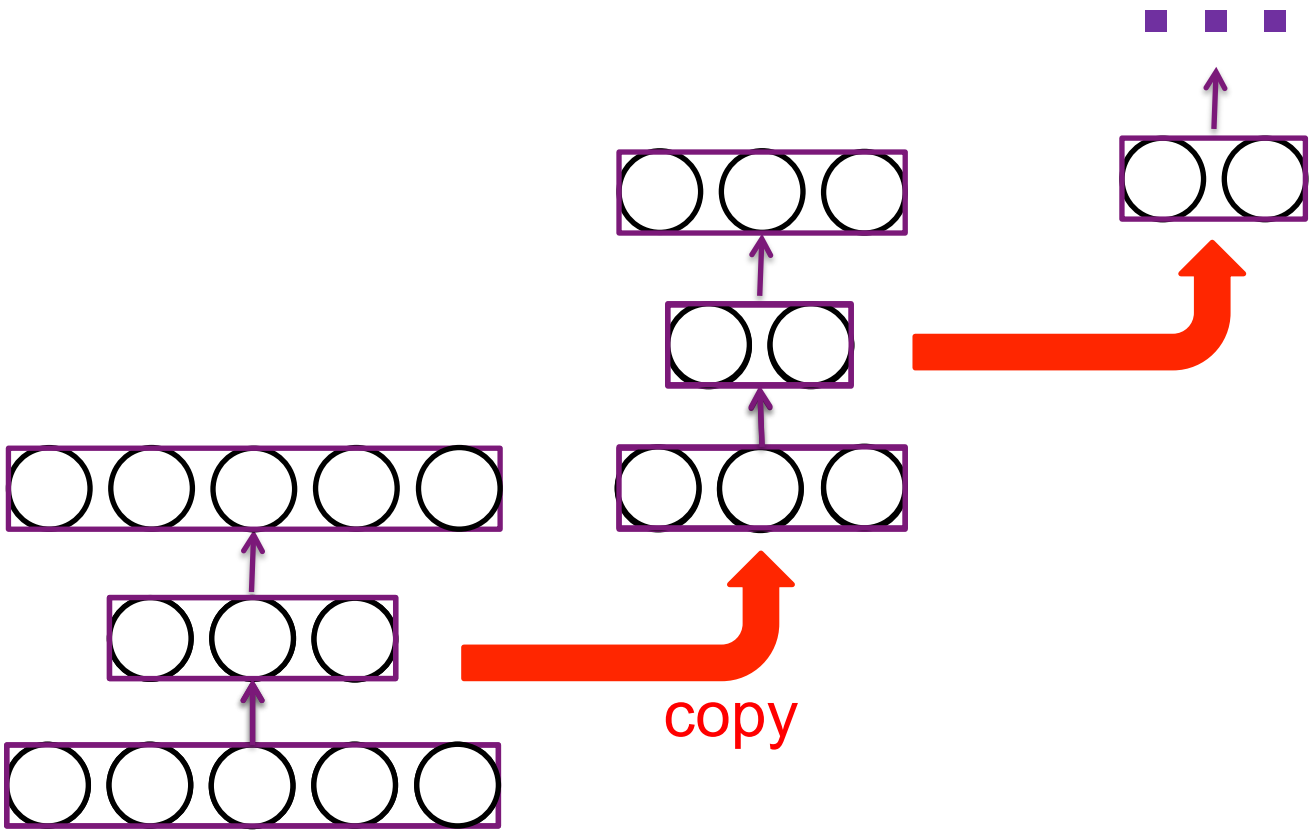
How to achieve bottleneck?

- Fewer neurons
- Sparsity constraint
- Information transmission constraint (e.g., add noise to unit)

Autoencoder Combines An Encoder And A Decoder



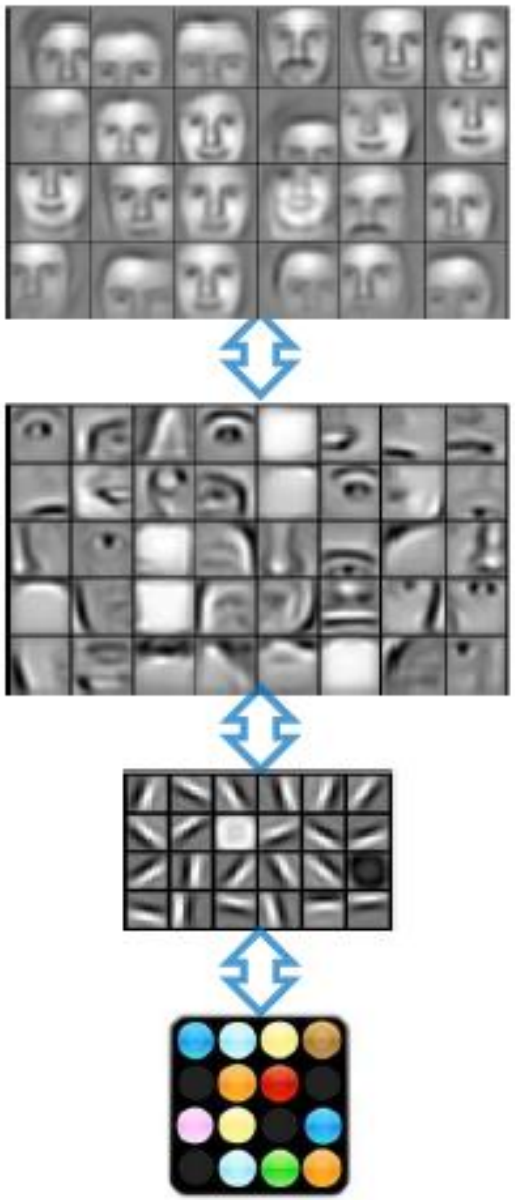
Stacked Autoencoders



deep network

Note that decoders can be stacked to produce a generative model of the domain

From Ng's group



3rd layer
"Objects"

2nd layer
"Object parts"

1st layer
"Edges"

Pixels

Suskever, Martens, Hinton (2011)
Generating Text From A Deep Belief Net

Wikipedia

The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. In the show's agreement unanimously resurfaced. The wild pasteured with consistent street forests were incorporated by the 15th century BE. In 1996 the primary rapford undergoes an effort that the reserve conditioning, written into Jewish cities, sleepers to incorporate the .St Eurasia that activates the population. Mar??a Nationale, Kelli, Zedlat-Dukastoe, Florendon, Ptu's thought is. To adapt in most parts of North America, the dynamic fairy Dan please believes, the free speech are much related to the

NYT

while he was giving attention to the second advantage of school building a 2-for-2 stool killed by the Cultures saddled with a half- suit defending the Bharatiya Fernall 's office . Ms . Claire Parters will also have a history temple for him to raise jobs until naked Prodiena to paint baseball partners , provided people to ride both of Manhattan in 1978 , but what was largely directed to China in 1946 , focusing on the trademark period is the sailboat yesterday and comments on whom they obtain overheard within the 120th anniversary , where many civil rights defined , officials said early that forms , " said Bernard J. Marco Jr. of Pennsylvania , was monitoring New York

Neural Net History

Until ~2012, most deep nets were built with unsupervised pretraining

- **Some researchers used back propagation autoencoders**
- **Other researchers used stochastic neural nets (restricted Boltzmann machines or RBMs)**

Eventually, it was discovered that no need for this approach is you combine a bunch of tricks…

Hinton Group Bag O' Tricks For Deep Learning

Initialize weights to sensible values

Rectified linear units

Hard weight regularization

Drop out

Convolutional architectures

Weight Initialization

How to ensures gradients don't vanish (or explode)?

- **Pretrain if you have a lot of unlabeled data and not a lot of labeled data (semisupervised learning)**
- **Echo State Network weight initialization procedure**
- **Various heuristics, e.g.,**

  Draw all weights feeding into neuron j (including bias) via

  $$w_{ji} \sim \text{Normal}(0,1)$$

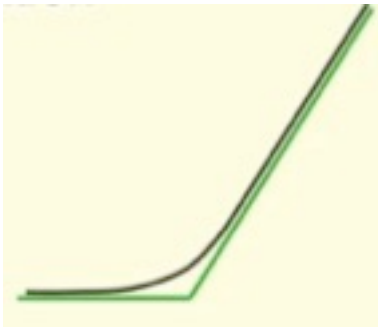
  If input activities lie in [-1, +1], then variance of input to unit j grows with fan-in to j, f_j

  Normalize such that variance of input is equal to C^2, i.e.,

  $$w_{ji} \leftarrow \frac{C}{\sqrt{f_j}} w_{ji}$$

  If input activities lie in [-1, +1], most net inputs will be in [-2C, +2C]

Rectified Linear Units



- **Version 1**

$$y = \log(1 + e^z)$$

$$\frac{\partial y}{\partial z} = \frac{e^z}{1 + e^z}$$

- **Version 2**

$$y = \max(0, z)$$

$$\frac{\partial y}{\partial z} = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

- **Raf, do we need to worry about z=0?**
- **Produces sparse activity patterns**

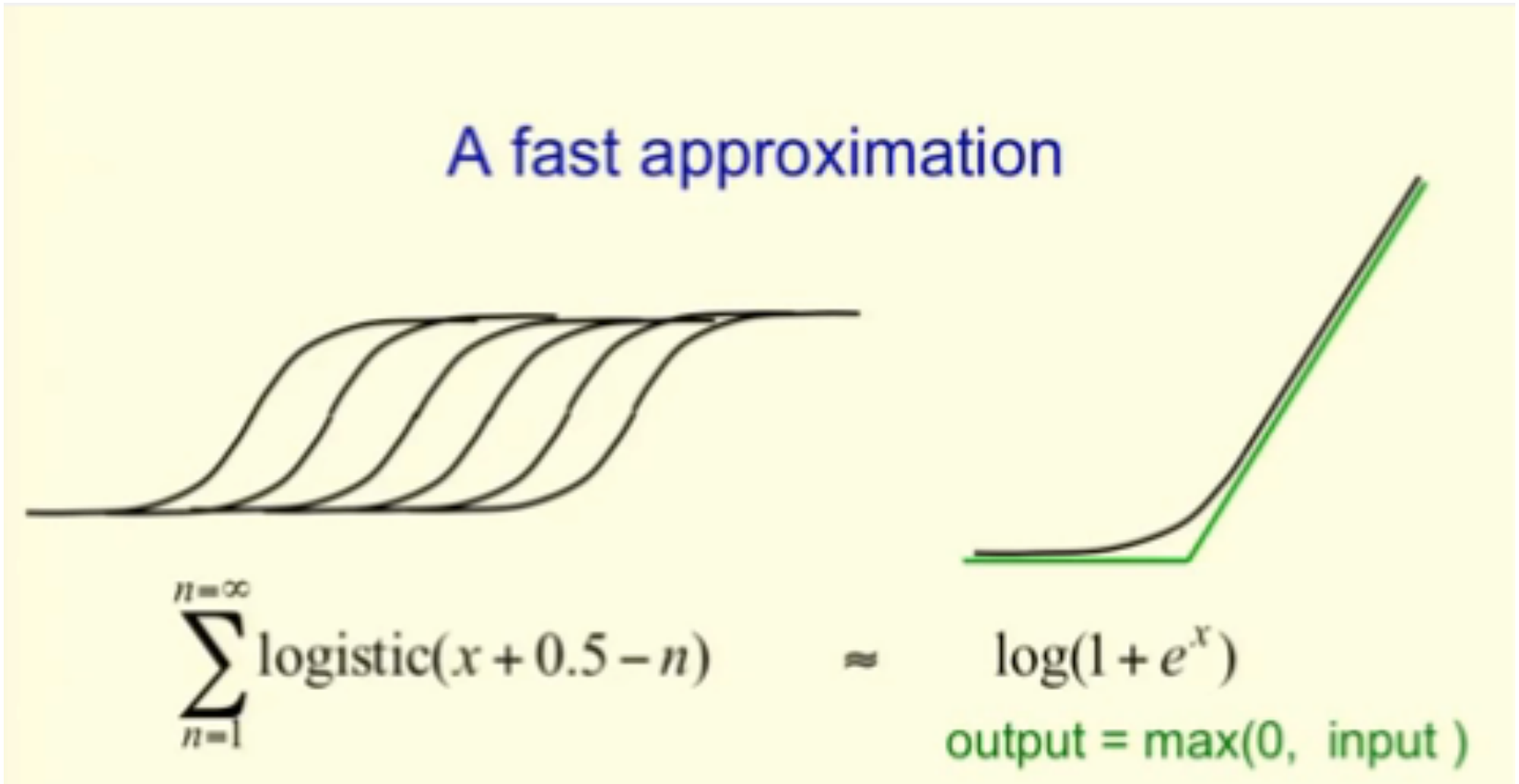
  Any unit that is turned off (y=0) does not learn
- **Avoids vanishing gradient problem**

  Any unit that is on (y>0) has a gradient of 1

  Possibility of exploding gradients!
- **In terms of implementation, V2 is much faster than logistic or V1**

Rectified Linear Units

Hinton argues that this is a form of model averaging



Source: Hinton Coursera slides

Regularization Techniques

Soft weight constraints

L2 weight decay

$$E = \frac{1}{2}\sum_j \left(t_j - y_j\right)^2 + \frac{\lambda}{2}\sum_{i,j} w_{ji}^2$$

$$\Delta w_{ji} = \varepsilon \delta_j x_i - \varepsilon \lambda w_{ji}$$

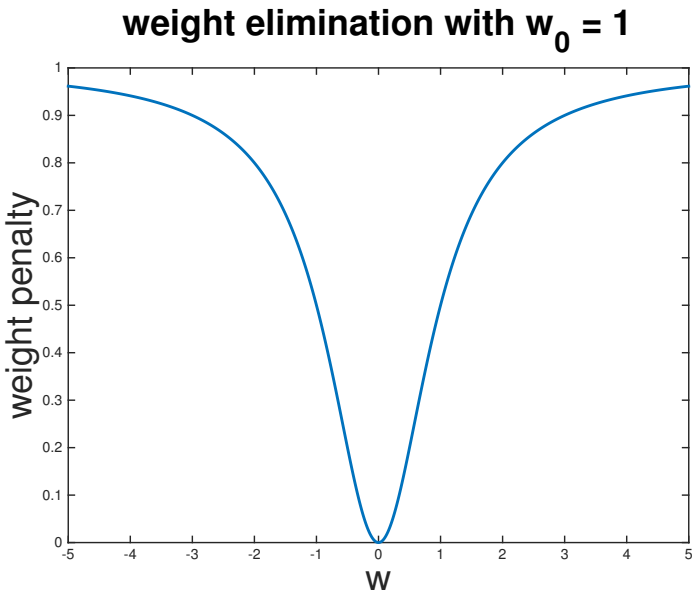
L1 weight decay

$$E = \frac{1}{2}\sum_j \left(t_j - y_j\right)^2 + \frac{\lambda}{2}\sum_{i,j} \left|w_{ji}\right|$$

$$\Delta w_{ji} = \varepsilon \delta_j x_i - \varepsilon \lambda \text{sign}(w_{ji})$$

weight elimination

$$E = \frac{1}{2}\sum_j \left(t_j - y_j\right)^2 + \frac{\lambda}{2}\sum_{i,j} \frac{w_{ji}^2 / w_0^2}{1 + w_{ji}^2 / w_0^2}$$



Regularization Techniques

Hard weight constraint

Ensure that $\sum_i w_{ji}^2 < \phi$ for every unit

If constraint is violated, rescale all weights: $w_{ji} \leftarrow w_{ji} \frac{\phi}{\sum_i w_{ji}^2}$

My argument for why this works

- With rectified linear units, rescaling weights simply rescales outputs (no loss of information), but units are more pliable
- With logistic units, it does limit nonlinearity but it also keeps the units more pliable

Dropout (Hinton, 2012)

During training

- As each training example is presented, randomly remove a fraction κ of all hidden units.
- Can also drop out input units, depending on the domain.

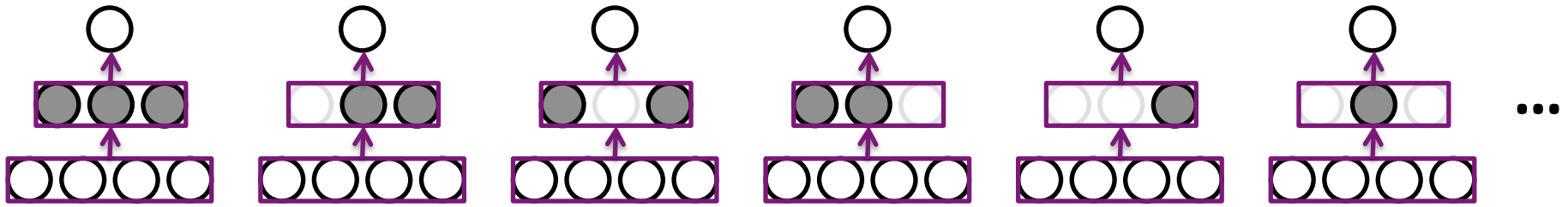
During testing

- Include all units in the network.
- Multiply output of each hidden (or input) unit by κ

Note: this represents the expected value of the unit during the training phase

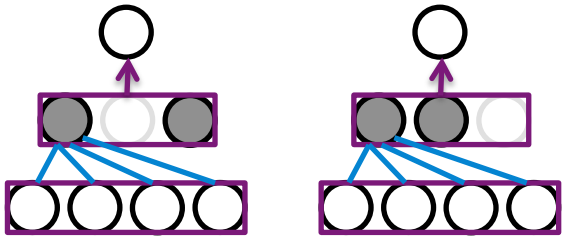
More On Dropout

With H hidden units, each of which can be dropped, we have 2^H possible models



- **each of these models needs to be able to perform the task**
- **no unit can depend on the presence of another unit**

Each of the 2^{H-1} models that include hidden unit h must share the same weights for the units



- **serves as a form of regularization**
- **makes the models cooperate**

More On Dropout

With one hidden layer and a logistic output,

- **Including all hidden units at test with a scaling of κ is exactly equivalent to computing the geometric mean of all 2^H models**

With multiple hidden layers,

- **"pretty good approximation" according to Geoff**

Convolutional Architectures

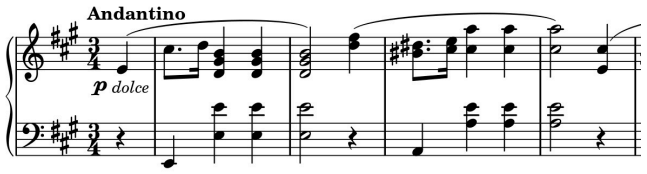
Consider domains with structured or latticed input

- **Spatial structure**

  E.g., images

- **Temporal structure**

  E.g, speech, music



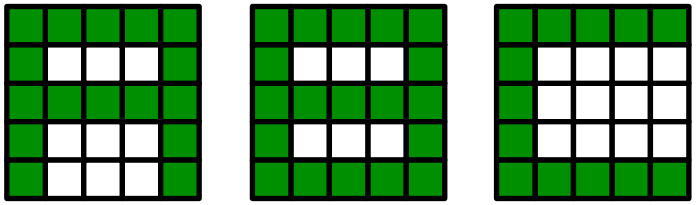
- **Sequential structure**

  E.g., language, video, DNA

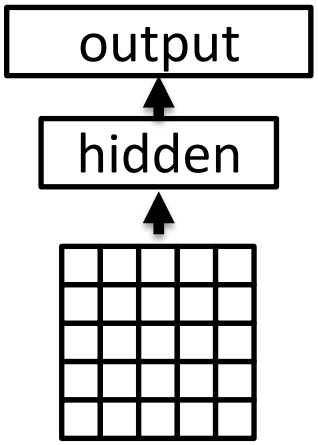
A convolutional neural net (ConvNet) takes advantage of this structure through a specialized architecture.

Recognizing An Image

Input is 5x5 pixel array

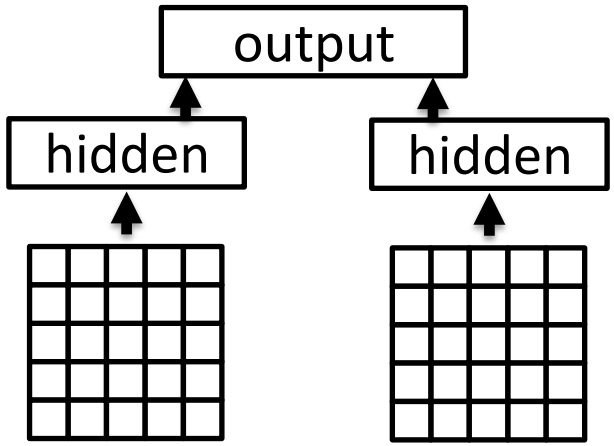


Simple back propagation net like you used in Assignment 3



Recognizing An Object With Unknown Location

Object can appear either in the left image or in the right image

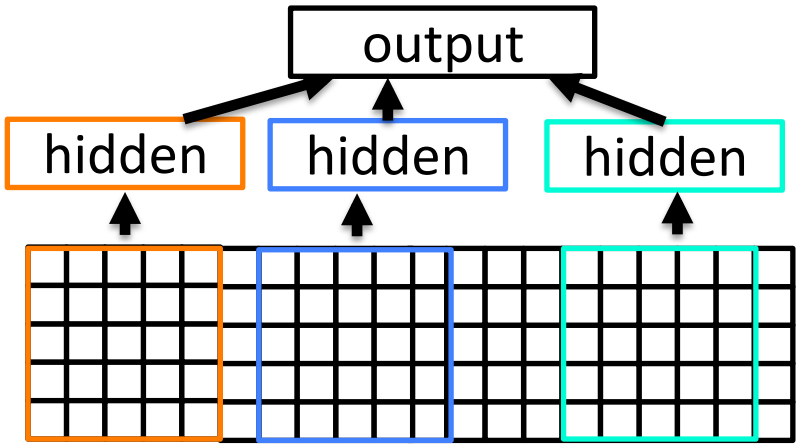


Output indicates presence of object regardless of position

What do we know about the weights?

Generalizing To Many Locations

Each possible location the object can appear in has its own set of hidden units



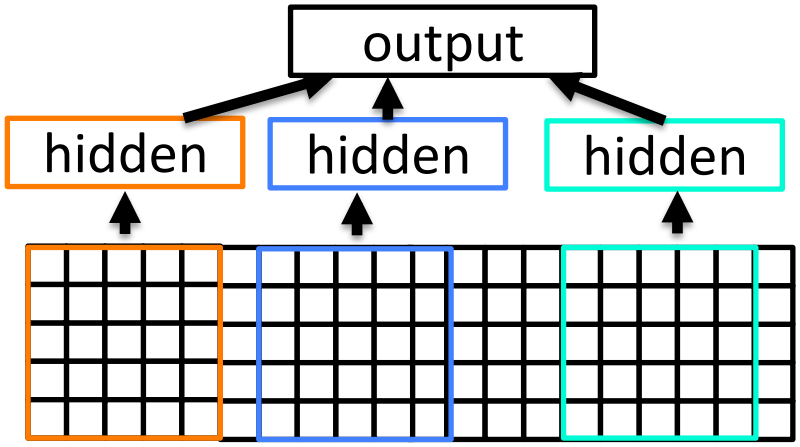
Each set detects the same features except in a different location

Locations can overlap

Convolutional Neural Net

Each patch of the image is processed by a different set of hidden units ('detectors', 'features')

But mapping from patch to hidden is the same everywhere



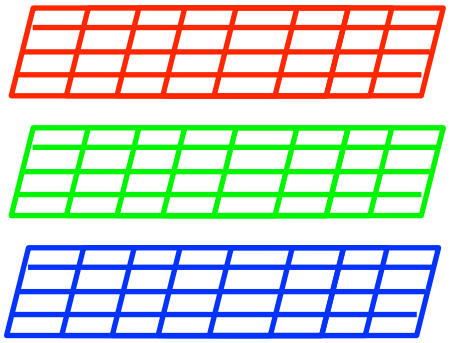
Can be in 2D as well as 1D

Achieves translation invariant recognition

The Input Layer

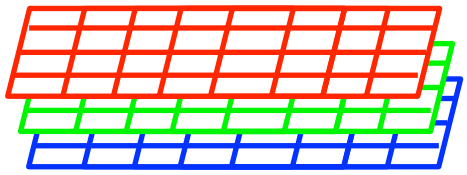
Input layer typically represents pixels present

- **at a given (x,y) location**
- **of a particular color (R, G, B)**



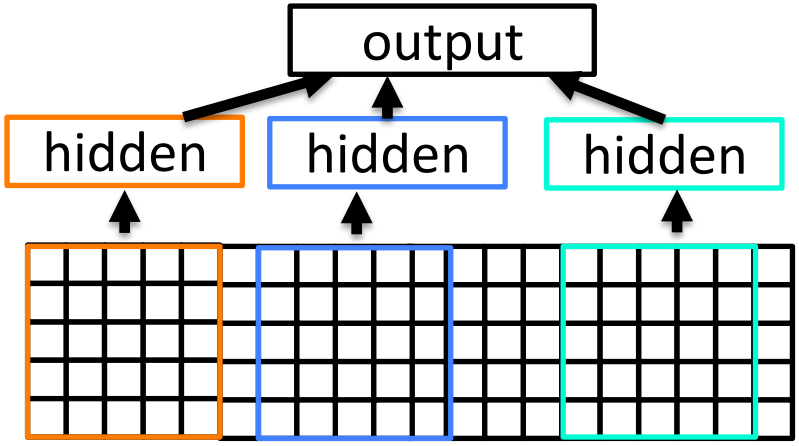
3D lattice

- **height X width X # channels**



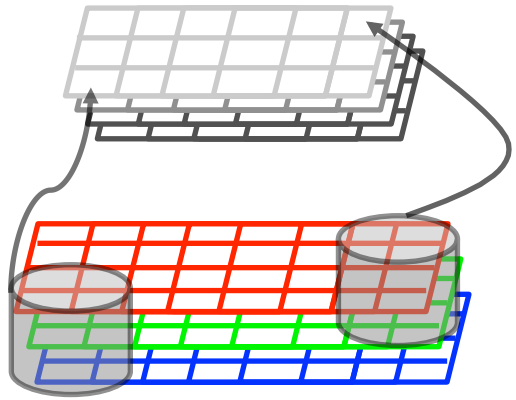
The Hidden Layer

Each hidden unit (i) is replicated across lattice



Instead of drawing pools of hidden, draw one (x,y) lattice for each hidden unit type (i)

- **Hidden unit i at (x,y) gets input from a cylinder of activity from its corresponding input patch**
- **Each hidden unit in a map has the same incoming weights**



Jargon

Each hidden unit lattice

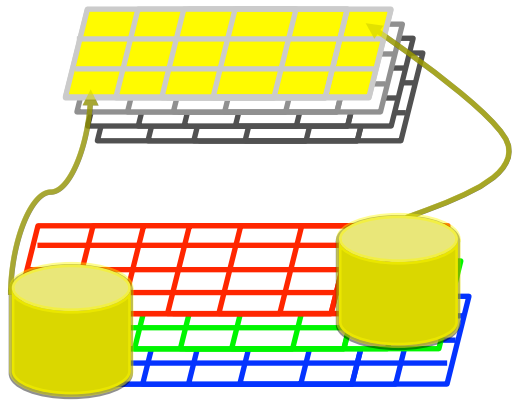
- **also called map, feature, feature type, dimension, channel**

Weights for each channel

- **also called kernels**

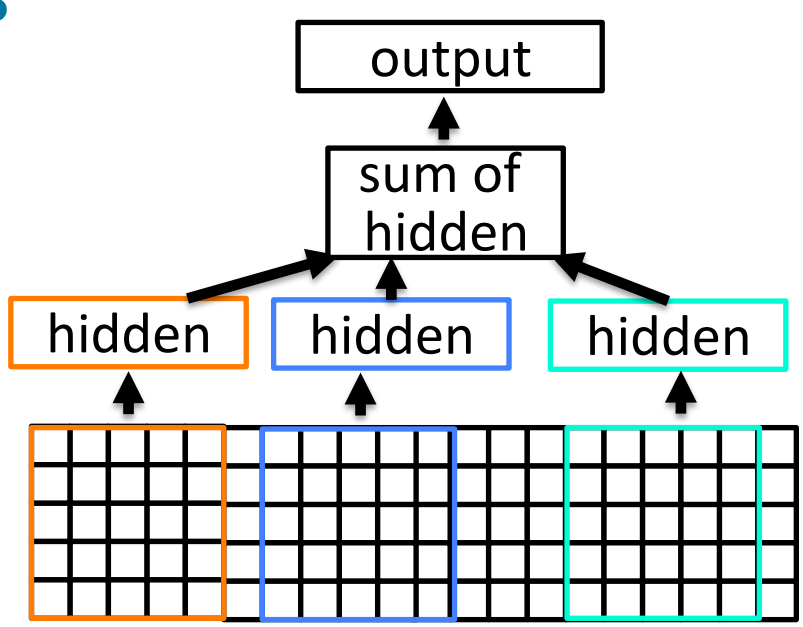
Input patch to a hidden unit at

- **also called receptive field**



Pooling / Subsampling

If all the hidden units are detecting the same features, and we simply want to determine whether the object appeared in any location, we can combine hidden representations



Sum pooling vs. max pooling

Transformation types

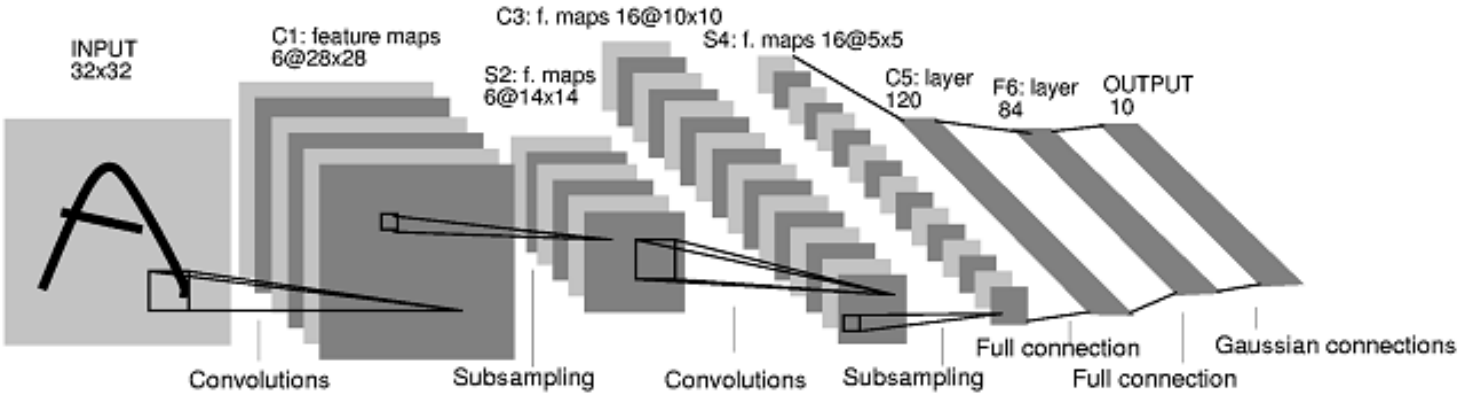
Each layer in a convolutional net has a 3D lattice structure

- **width X height X feature type**

Two types of transformations between layers

- **convolution + activation function**
- **pooling / subsampling**

Full blown convolutional net performs these transformations repeatedly -> Deep net



Videos And Demos

LeNet-5

Javascript convolutional net demo

ImageNet

ImageNet

- > 15M high resolution images
- over 22K categories
- labeled by Mechanical Turk workers

ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)

- 2010-2015

2010 ImageNet Competition

1.2 M training images, 1000 categories (general and specific)

200K test images

output a list of 5 object categories in descending order of confidence

two error rates: top-1 and top-5

Krizhevsky, Sutskever, & Hinton (2012)

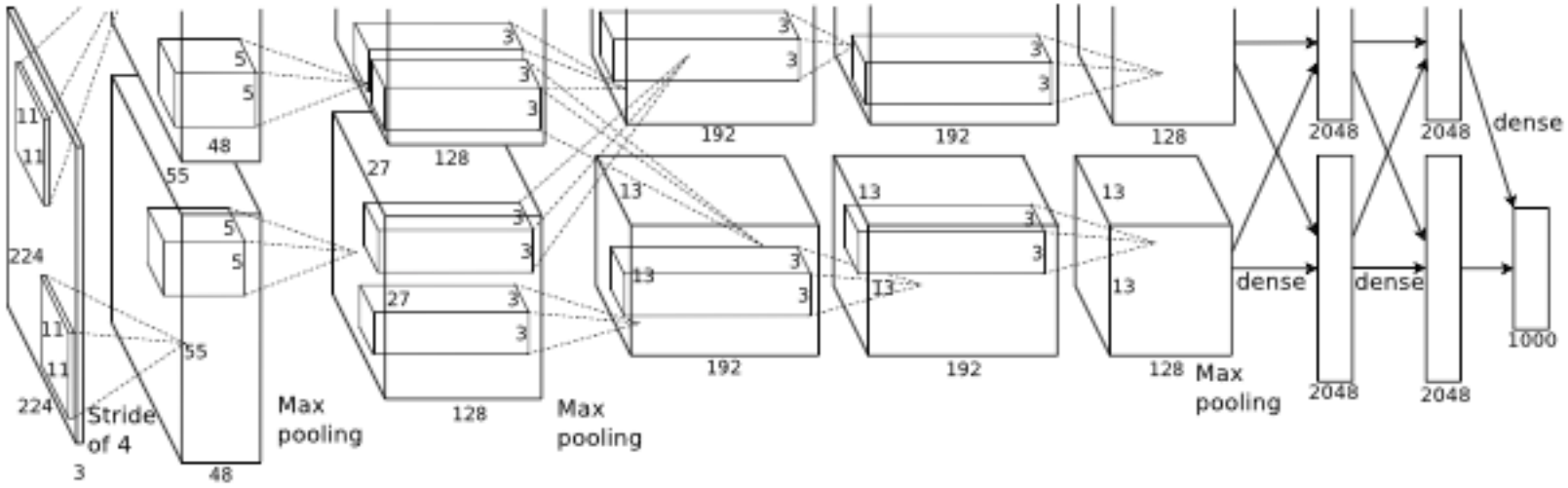


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

5 convolutional layers, split across two GPUs

3 fully connected layers

1000-way softmax output layer

650k units, 60M parameters, 630M connections

Trained on 2 GPUs for about a week with stochastic gradient descent

Key Ideas

ReLU instead of logistic or tanh units

Training on multiple GPUs

- cross talk only in certain layers
- balance speed vs. connectivity

Data set augmentation

- Vary image intensity, color
- Translate images
- Horizontal reflections
- Increases size of training set by a factor of 2048

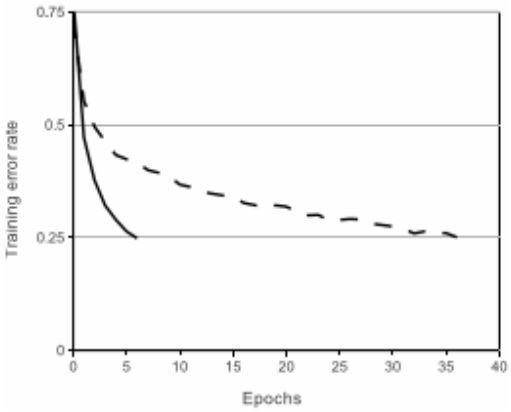


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**. The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

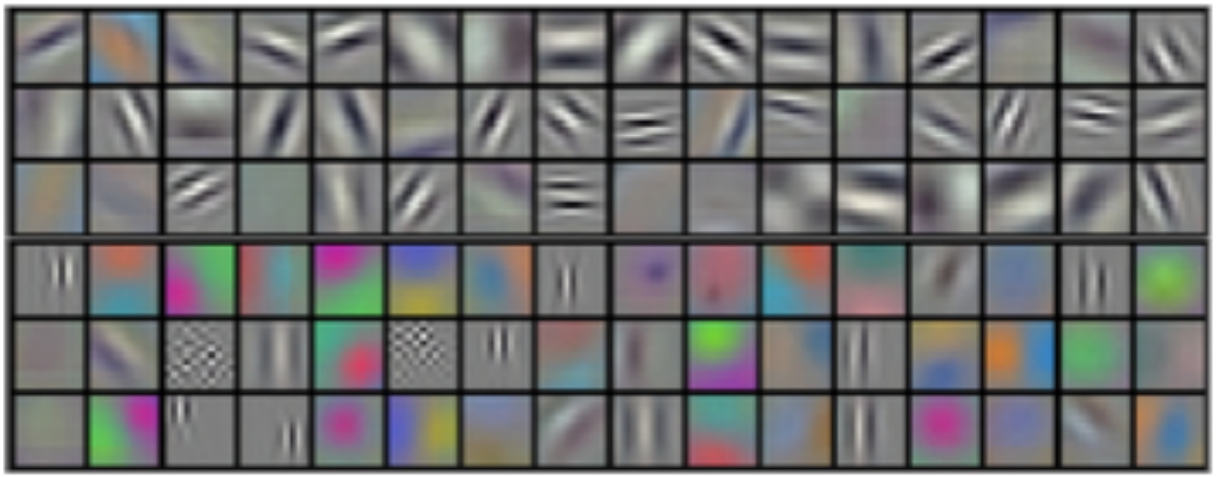
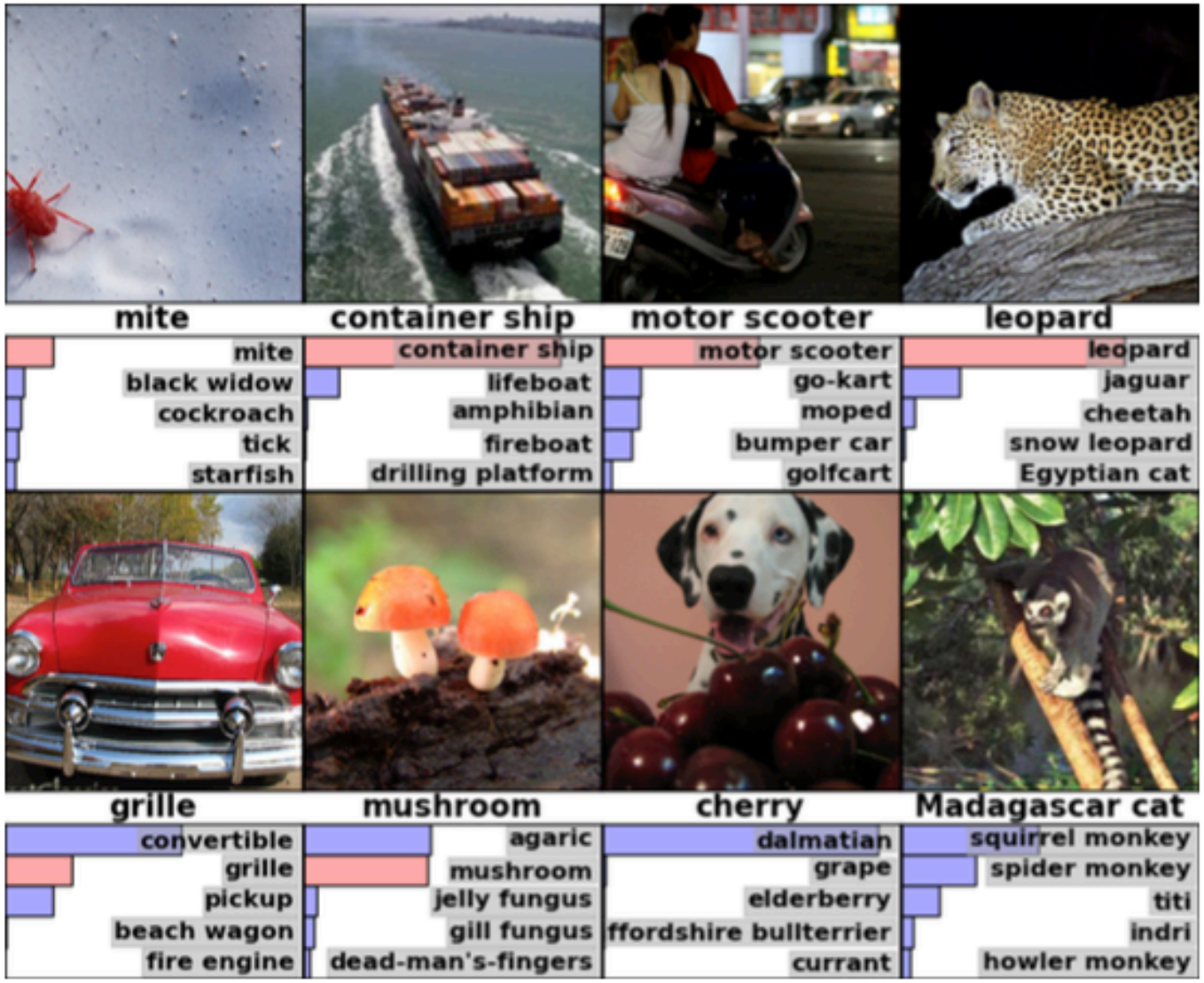


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.


mite
mite
black widow
cockroach
tick
starfish
container ship
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
motor scooter
go-kart
moped
bumper car
golfcart
leopard
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
convertible
grille
pickup
beach wagon
fire engine
mushroom
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
cherry
dalmatian
grape
elderberry
ffordshire bullterrier
currant
Madagascar cat
squirrel monkey
spider monkey
titi
indri
howler monkey

2010 Results

Team	Score
SuperVision	**0.15315**
ISI	0.26172
OXFORD_VGG	0.26979
XRCE/INRIA	0.27058
University of Amsterdam	0.29576
LEAR-XRCE	0.34464

2013: Down to 11% error

Image Captioning: The Latest Fad

Deep Visual-Semantic Alignments for Generating Image Descriptions

- **Karpathy & Fei-Fei (Stanford)**

Show and Tell: A Neural Image Caption Generator

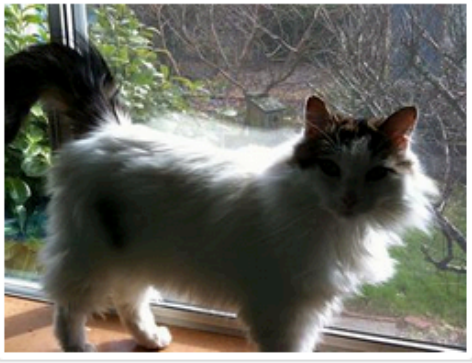
- **Vinyals, Toshev, Bengio, Erhan (Google)**

Deep Captioning with Multimodal Recurrent Nets

- **Mao, Xu, Yang, Wang, Yuille (UCLA, Baidu)**

Four more at end of class…

Below are some examples of the generated sentences from Microsoft COCO dataset. It is generated by sampling the maximum likelihood word. It seems that in this dataset, "a" is the most common first word in the sentence, so all the generated sentences begin with "a". The results are expected to be further improved by using beam search.



a cat sitting on a bench in front of a window

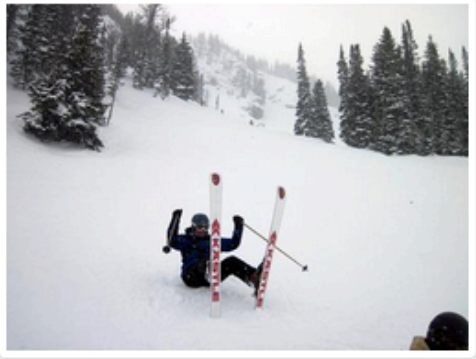
a man riding a snowboard down a snow covered slope

a train traveling down a train track next to a forest

a laptop computer sitting on top of a table

a man riding a wave on top of a surfboard

a clock tower with a clock on top of it



a person on skis is skiing down a hill

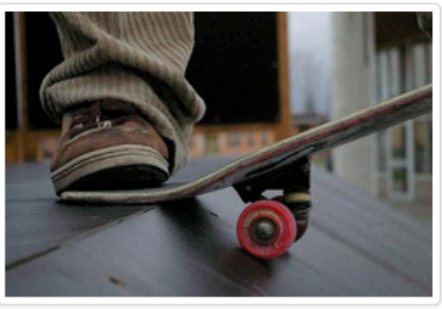
a pizza sitting on top of a table next to a box of pizza

a red fire hydrant sitting in the middle of a forest

a train traveling down a bridge over a river

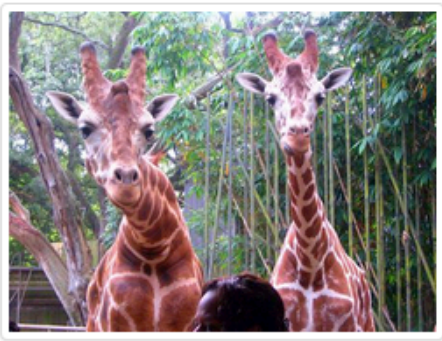
a man riding a surfboard on top of a wave

a bus is driving down the street in front of a building



a person is holding a skateboard on a street

a close up of a bowl of food on a table

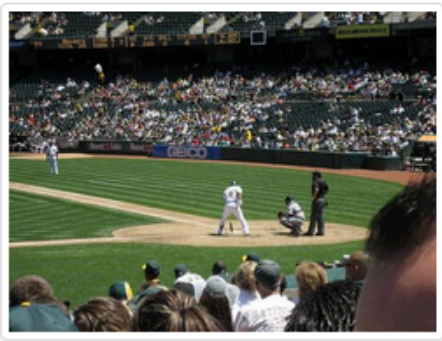


a giraffe standing next to a tree in a zoo

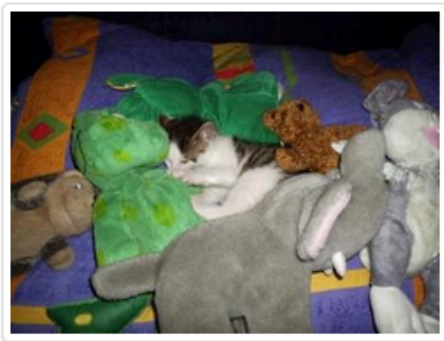
a bunch of oranges are on a table

a group of planes flying in the sky

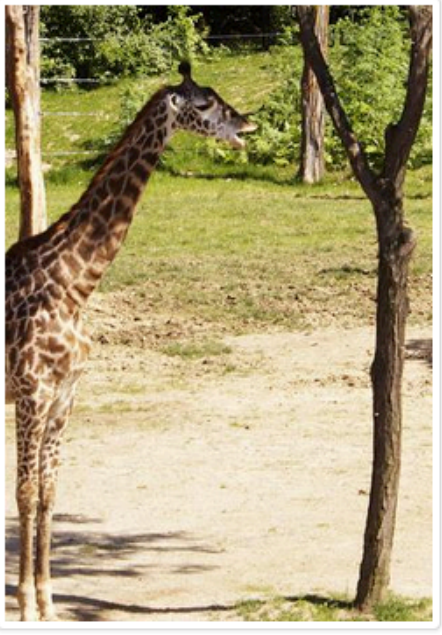
a train is traveling down the tracks in a city



a baseball player is swinging at a ball



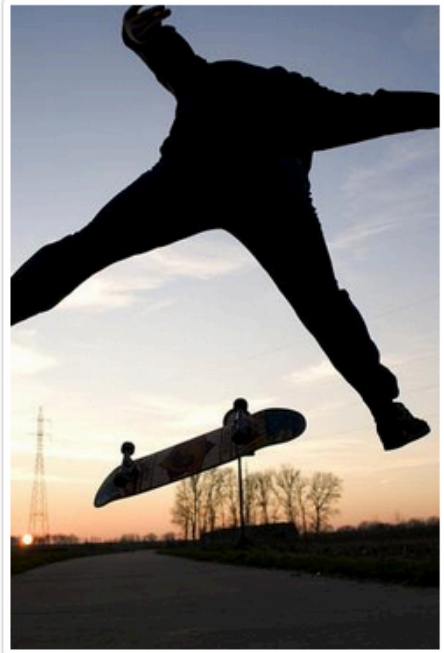
a cat laying on a bed with a stuffed animal



a giraffe standing in a field with trees in the background

a young girl brushing his teeth with a toothbrush

a group of people flying kites in a field



a man is doing a trick on a skateboard

Failures

a woman standing in front of a table with a cake

a kitchen with a stove , stove , and a refrigerator

a bird sitting on top of a tree branch

a zebra standing on a dirt road next to a tree

a man in a baseball uniform is holding a baseball bat

a man is playing tennis on a tennis court

a woman sitting at a table with a plate of food

a person holding a remote control in a hand

END

Conditioning The Input Vectors

If m_i is the mean activity of input unit i over the training set and s_i is the std dev over the training set

For each input (in both training and test sets), normalize by

$$x_i^\alpha \leftarrow \frac{x_i^\alpha - \overline{x}_i}{s_i}$$

- where $\overline{x}_i$ is the training set mean activity and s_i is the std deviation of the training set activities

Conditioning The Hidden Units

If you're using logistic units, then replace logistic output with function scaled from -1 to +1

$$y = \frac{2}{1 + e^{-net}} - 1 \qquad \frac{\partial y}{\partial net} = \frac{1}{2}(1 + y)(1 - y)$$

- **With net=0, y=0**
- **Will tend to cause biases to be closer to zero and more on the same scale as other weights in network**
- **Will also satisfy assumption I make to condition initial weights and weight updates for the units in the next layer**
- **tanh function**

Setting Learning Rates I

Initial guess for learning rate

- If error doesn't drop consistently, lower initial learning rate and try again
- If error falls reliably but slowly, increase learning rate.

Toward end of training

- Error will often jitter, at which point you can lower the learning rate down to 0 gradually to clean up weights

Remember, plateaus in error often look like minima

- be patient
- have some idea a priori how well you expect your network to be doing, and print statistics during training that tell you how well it's doing
- plot epochwise error as a function of epoch, even if you're doing minibatches

$$\text{NormalizedError} = \frac{\sum (t^{\alpha} - y^{\alpha})^2}{\sum (t^{\alpha} - \bar{t})^2}$$

Setting Learning Rates II

Momentum

- $\Delta w_{t+1} = \theta \Delta w_t - (1-\theta)\varepsilon \frac{\partial E}{\partial w_t}$

Adaptive and neuron-specific learning rates

- Observe error on epoch t-1 and epoch t
- If decreasing, then increase *global* learning rate, $\varepsilon_{\text{global}}$, by an additive constant
- If increasing, decrease *global* learning rate by a multiplicative constant
- If fan-in of neuron j is f_j, then $\varepsilon_j = \varepsilon_{\text{global}} \Big/ \sqrt{f_j}$

Setting Learning Rates III

Mike's hack

Initialization

```
epsilon = .01
inc = epsilon / 10
if (batch_mode_training)
   scale = .5
else
   scale = .9
```

Update

```
if (current_epoch_error < previous_epoch_error)
   epsilon = epsilon + inc
   saved_weights = weights
else
   epsilon = epsilon * scale
   inc = epsilon / 10
   if (batch_mode_training)
      weights = saved_weights
```

Setting Learning Rates IV

rmsprop

- Hinton lecture

Exploit optimization methods using curvature

- Requires computation of Hessian

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

When To Stop Training

1. Train *n* epochs; lower learning rate; train *m* epochs

- bad idea: can't assume one-size-fits-all approach

2. Error-change criterion

- stop when error isn't dropping
- My recommendation: criterion based on % drop over a window of, say, 10 epochs

  1 epoch is too noisy

  absolute error criterion is too problem dependent

- Karl's idea: train for a fixed number of epochs after criterion is reached (possibly with lower learning rate)

When To Stop Training

3. Weight-change criterion

- Compare weights at epochs t-10 and t and test:

$$\max_i \left| w_i^t - w_i^{t-10} \right| < \theta$$

- Don't base on length of overall weight change vector
- Possibly express as a percentage of the weight
- Be cautious: small weight changes at critical points can result in rapid drop in error