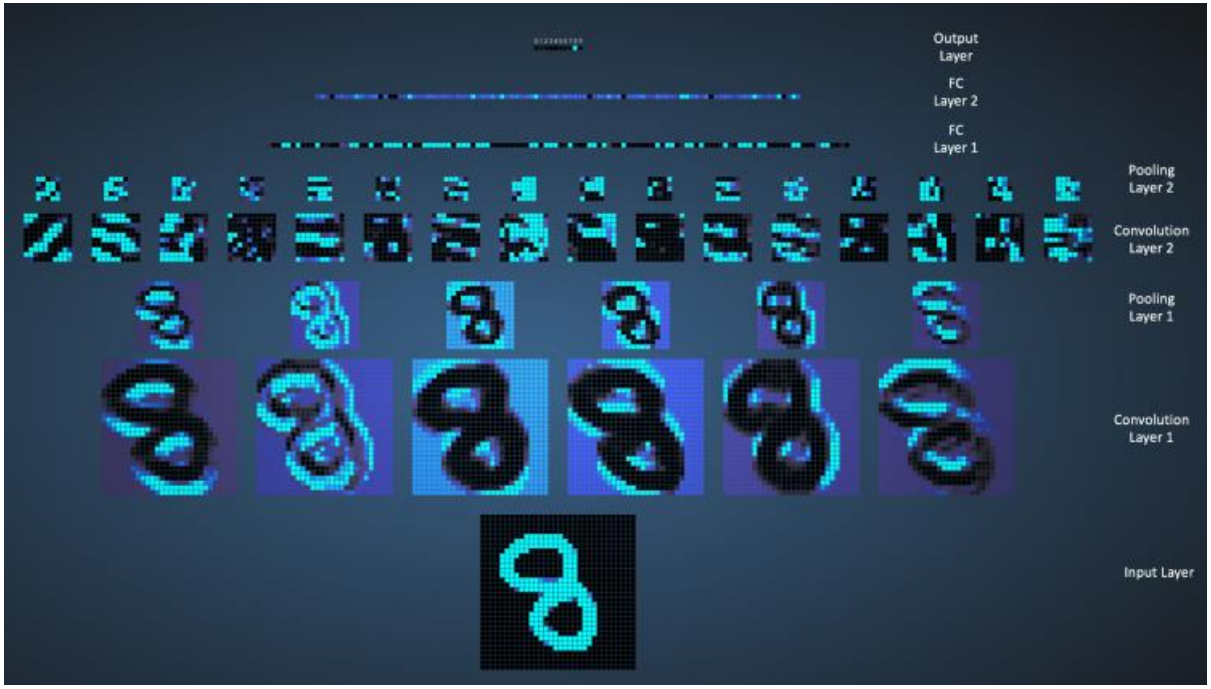
OA4118
Neural Networks: Special Architectures



http://scs.ryerson.ca/~aharley/vis

Special Architectures

- Deep NNs, with many layers help with **automatic feature generation,** but more nodes and layers add complexity, which brings its own problems
- A fully connected NN can have ‘way too many weights to train in any reasonable time

Special Architectures

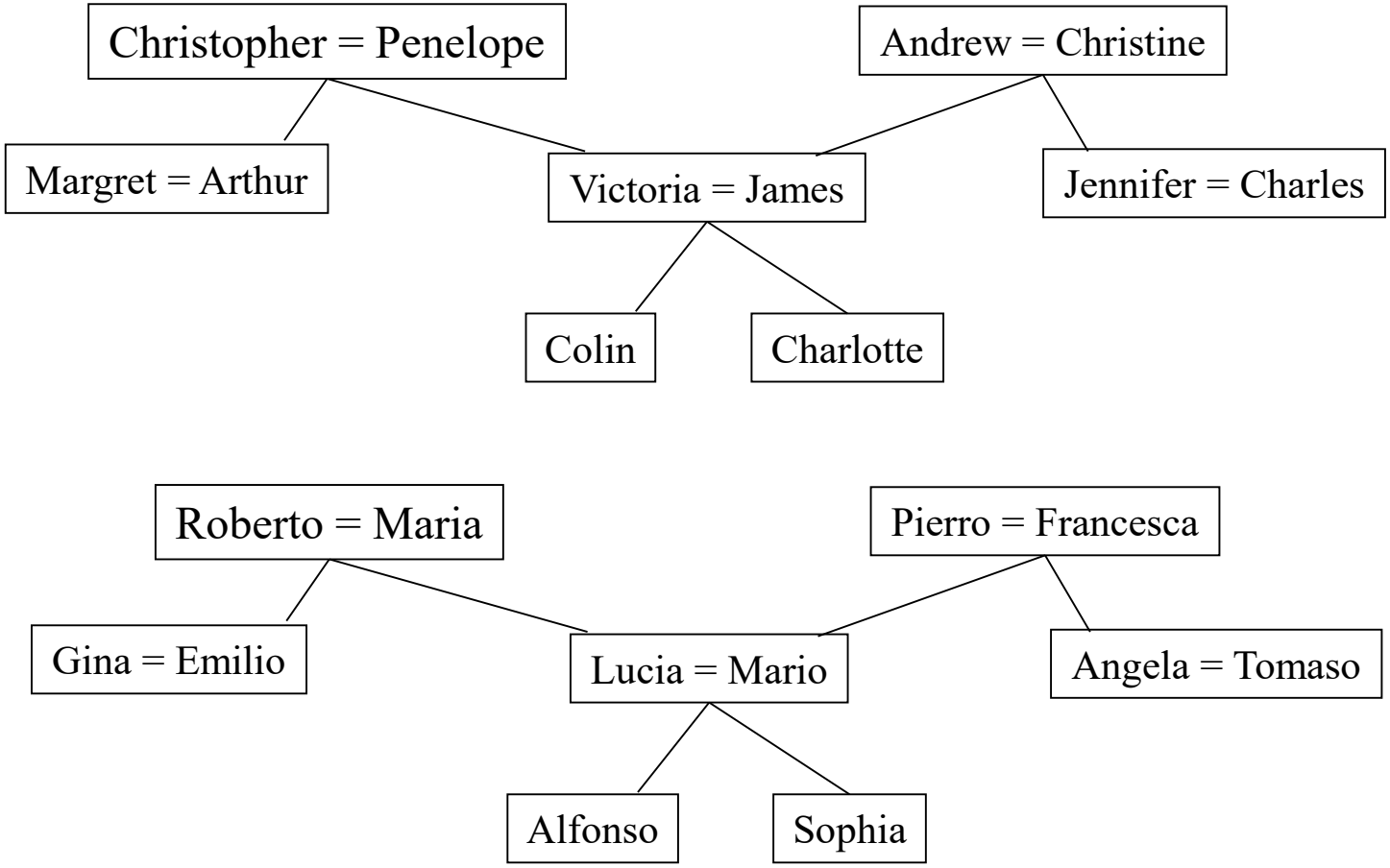
- In a number of specialized domains – like image, voice or language processing, one useful plan seems to be to
  - Design the **architecture tailored to the application** (e.g. CNN and RNNs) to promote the right kind of feature generation
  - …And use more (and more, and more) training data
  - …And use transfer learning
  - …And use dropout and batch-normalization

The Plan

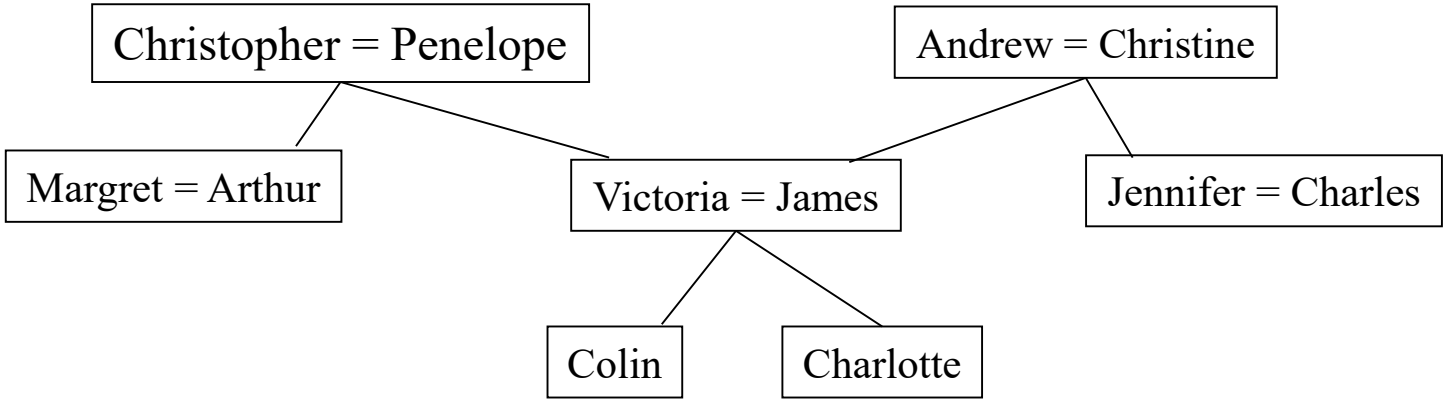
- Show a toy example (not an image!) of partially connected NN (or skip this?)
- Introduce Convolutional Neural Networks (CNN) using handwritten digits.
  - Handwritten digit recognition
- Recurrent Neural Networks (RNN) for variable length (usually time series data e.g. language processing).
  - Predict currency values
  - CNN + RNN used to generate image labels.
- AutoEncoders

A Toy Example
Paccanaro and Hinton (2000)

1980's Example of Learning Relationships with a Deep Neural Network (Paccanaro and Hinton, 2000)

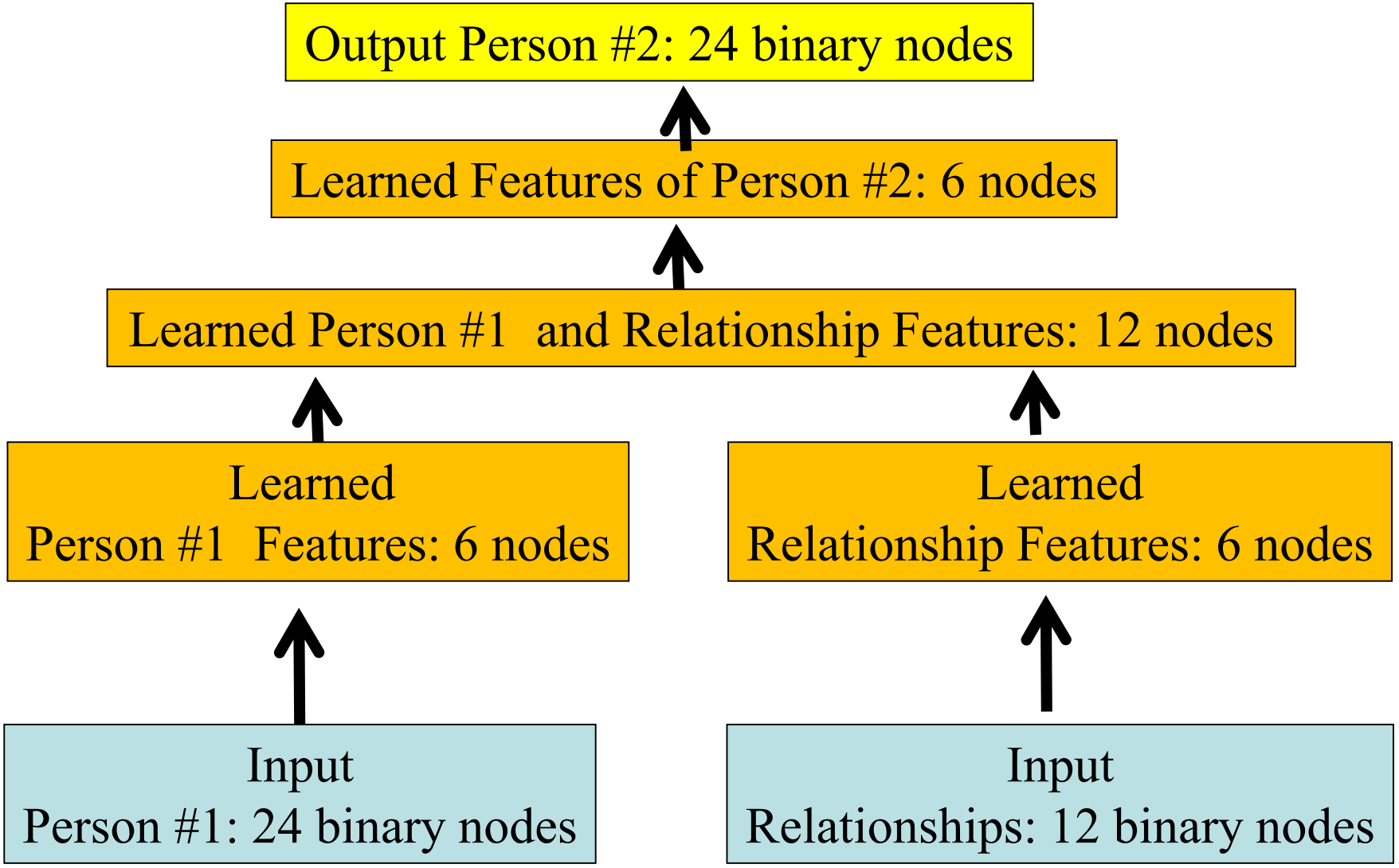


Express Relationships as Triples

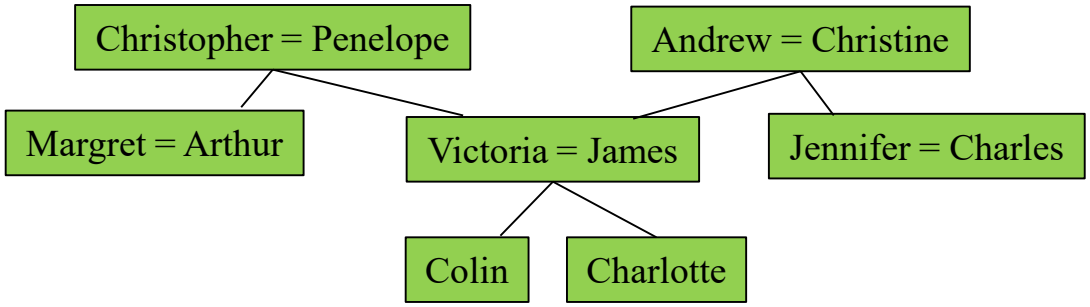


- There are 12 relationship types: uncle, niece, mother, …
- Express each relationship as a triples (112):
  Christopher has wife Penelope
- Train a neural network to predict the second person from the name of the first person and the relationship

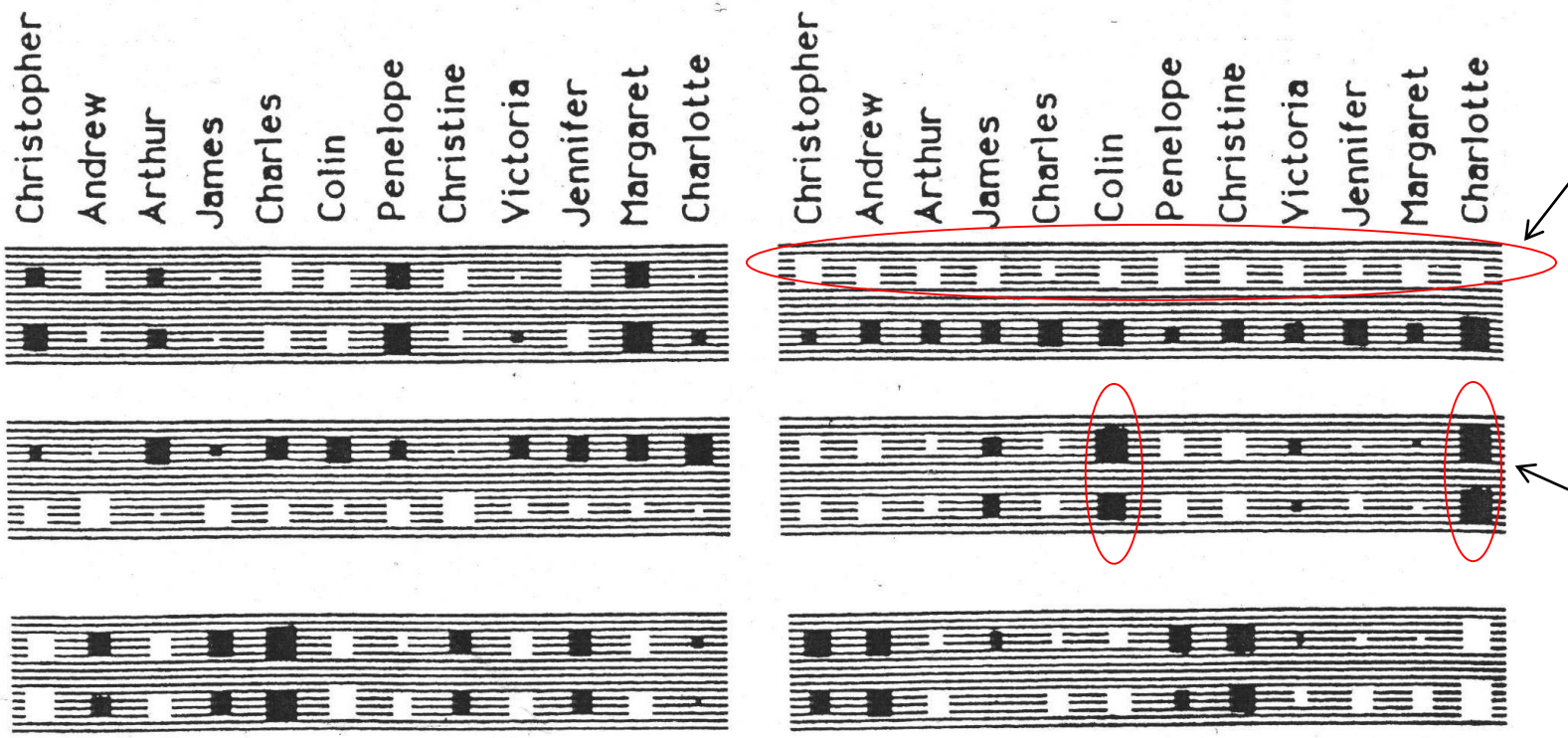
Architecture: Local Connectivity



6 x 24 Weights From Person Input to First Hidden Layer 6 Nodes (Hinton '12)



white squares = positive weights
black squares = negative weights



24 Weights to Node 4: Identifies English

Node 5 Identifies 3rd Generation

Deep Neural Networks applied to computer vision

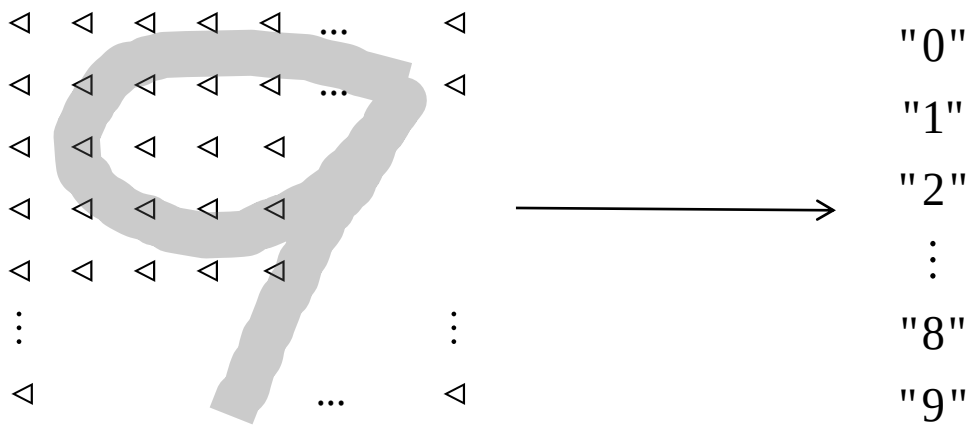
- Using a fully connected NN for image recognition can cause problems with recognizing different sizes, translations, etc. (unless you have tons and tons and tons of data and build a deep NN and regularize the heck out of it).
- Another solution is to use a deep network that starts with **local connections** that learn the same feature for small patches of an image, no matter where that patch is. (Convolutional Neural Network, **CNN**).
- LeCun (1989) was first to recognize a sequence of digits (once used for 10% of recognizing all handwritten checks in North America) is by

http://yann.lecun.com/exdb/lenet/index.html

Traditional Neural Networks for Handwritten Digit Recognition

16 x 16 pixel input nodes

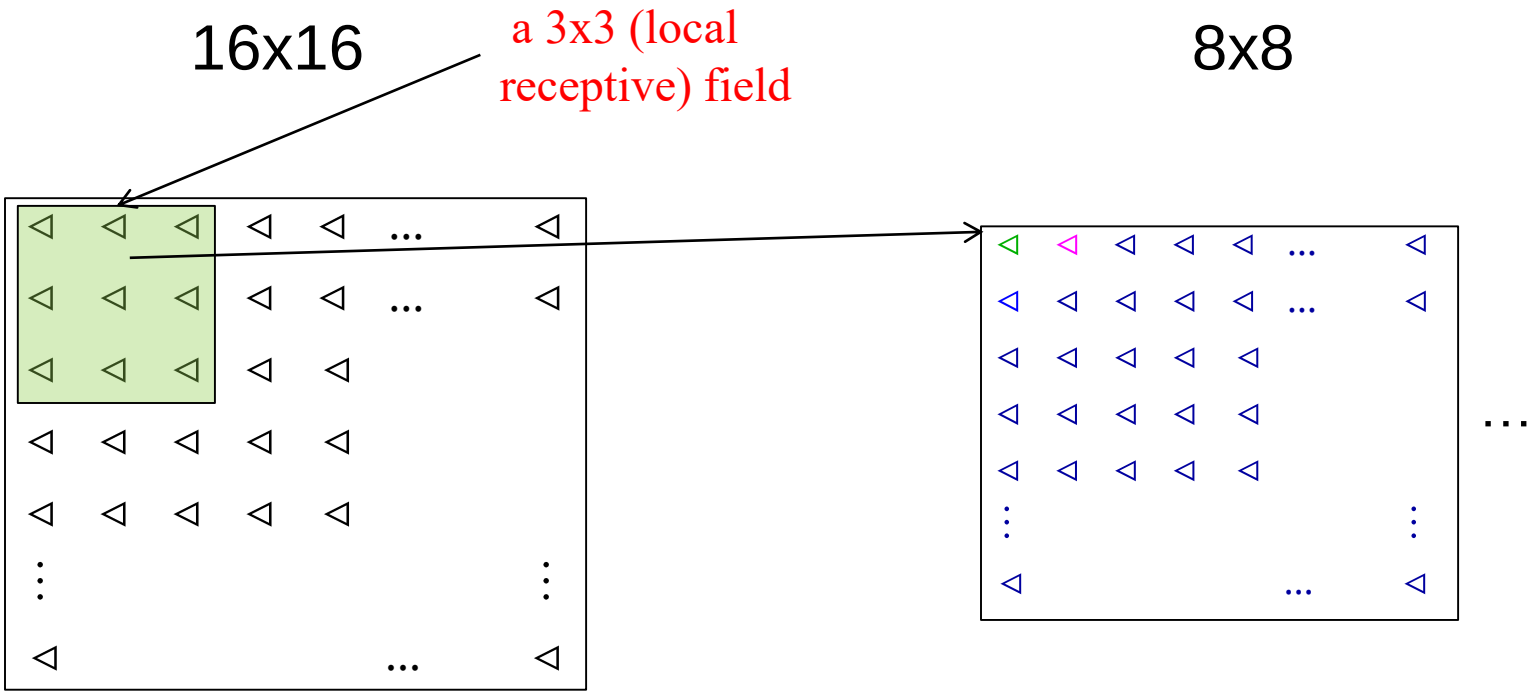
10 output nodes



Model	#weights	Test Accuracy
Net-1: No hidden layer	2570 = [(16)16+1] 10	80% correct
Net-2: Add one hidden layer with 12 nodes	3214	87% correct

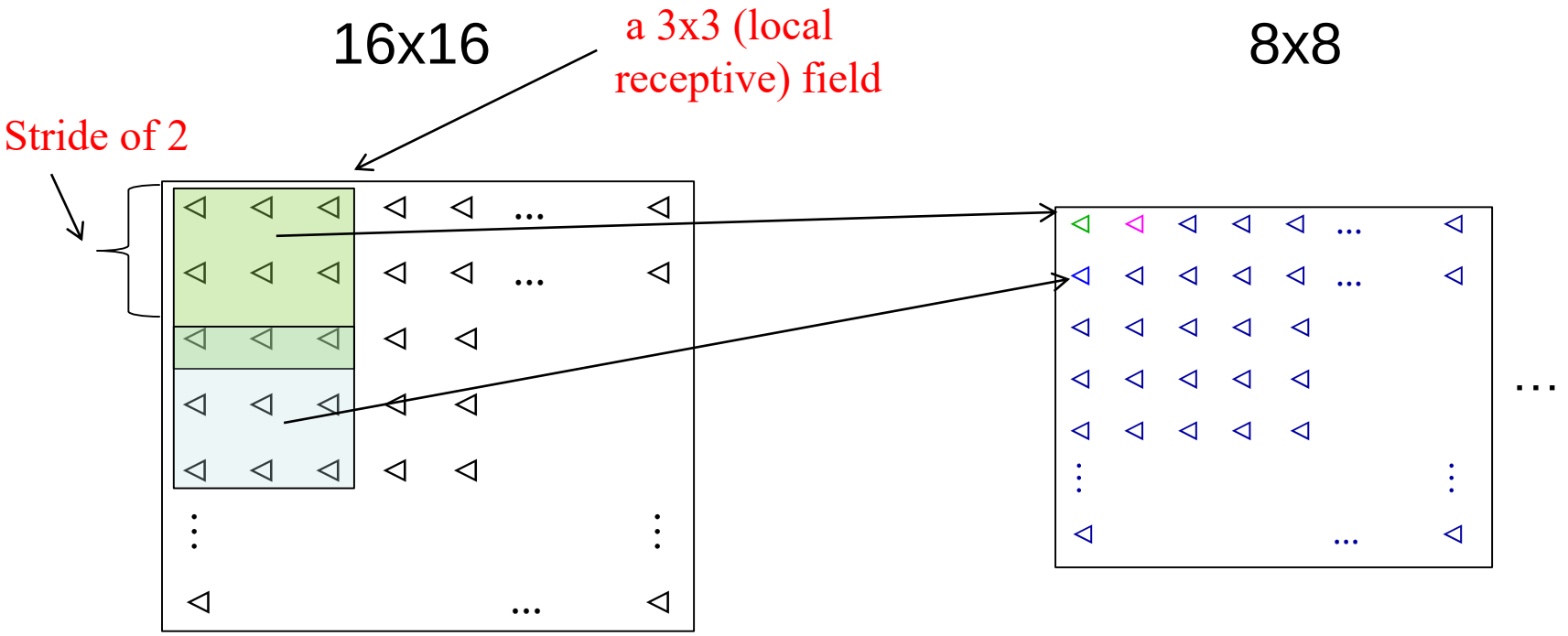
Use Local Connections as in the previous example.

Learn 8x8 local features for small 3x3 overlapping patches.
With local connectivity many weighs are set to 0.



Use Local Connections as in the previous example.

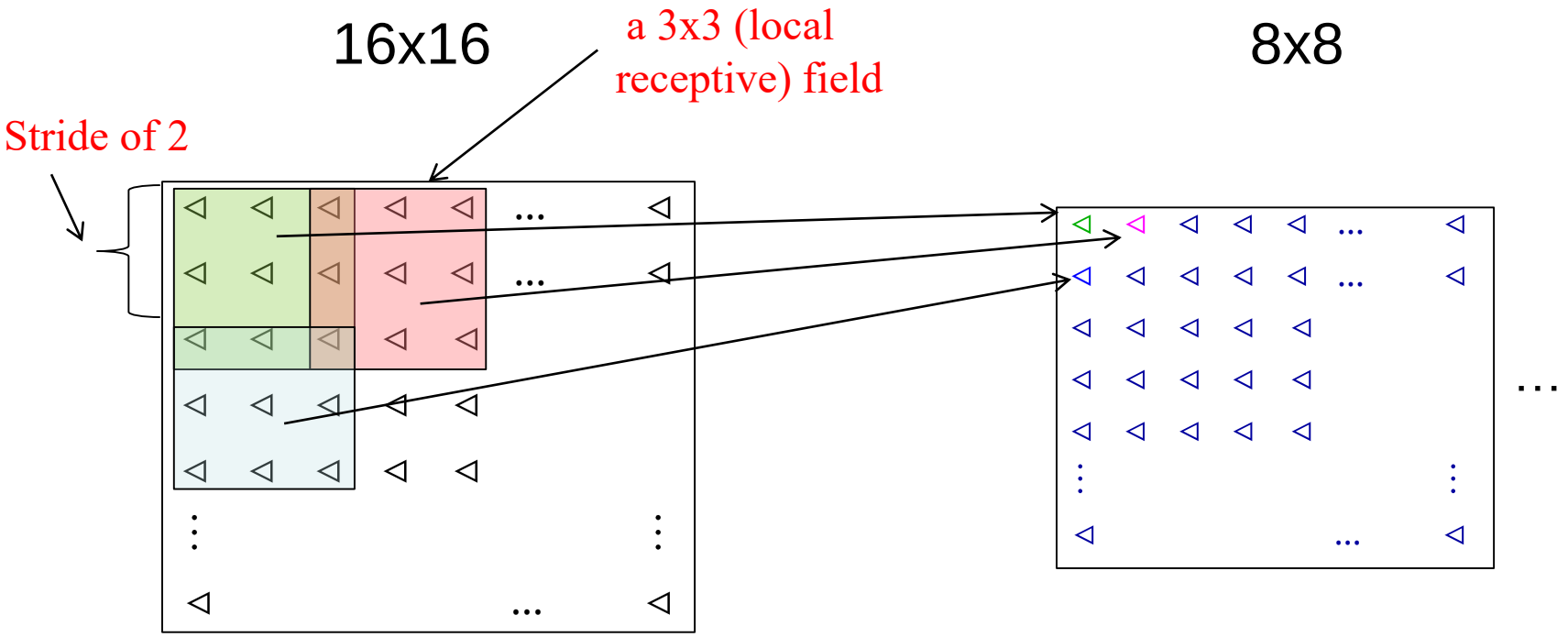
Learn 8x8 local features for small 3x3 overlapping patches.
With local connectivity many weighs are set to 0.



Use Local Connections as in the previous example.

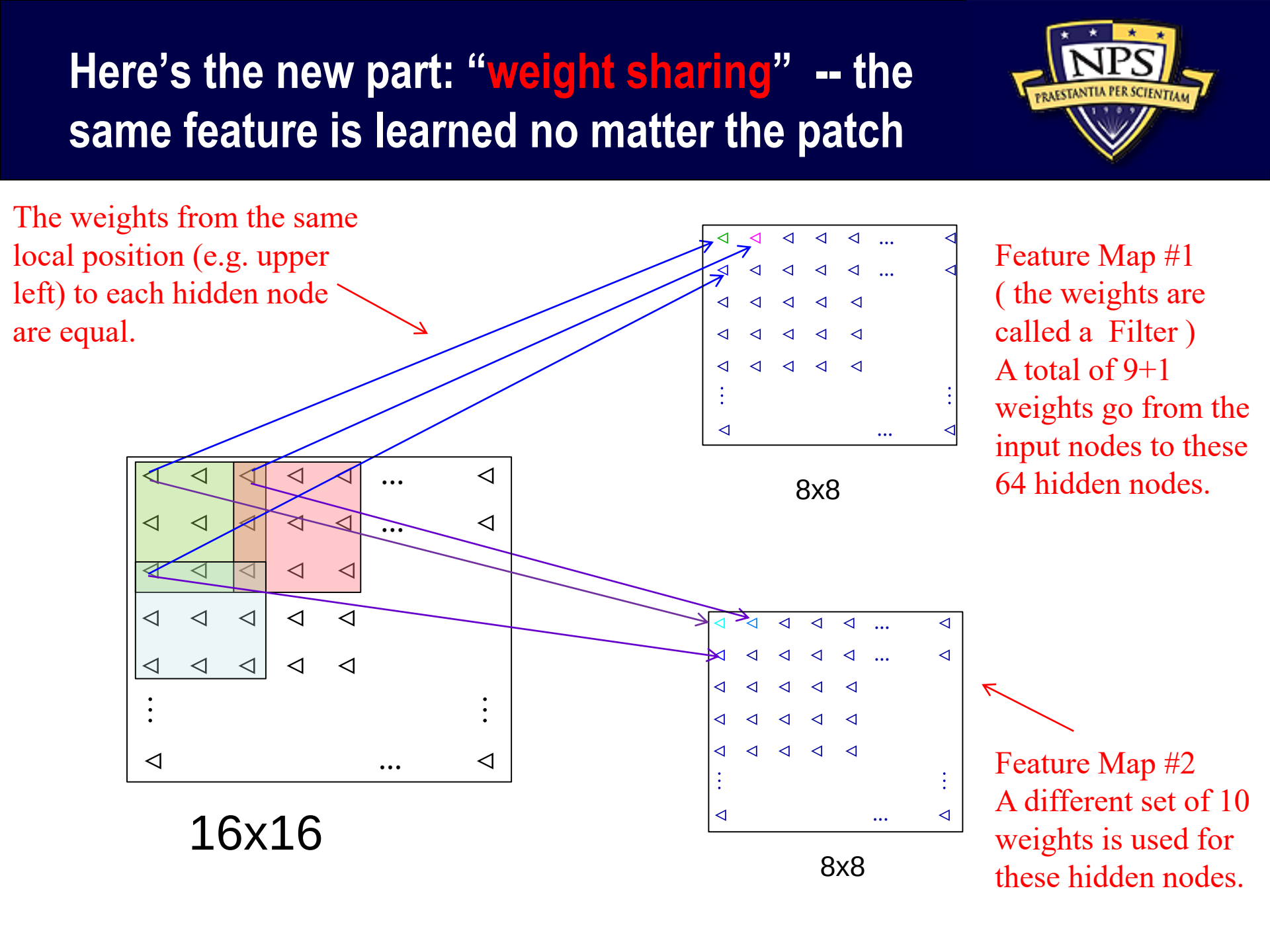
Learn 8x8 local features for small 3x3 overlapping patches.

With local connectivity many weighs are set to 0.



Nnet-3: 2 locally connected hidden layers has **1226** weights (the last hidden layer is fully connected to the 10 output nodes) and 88.5% accuracy

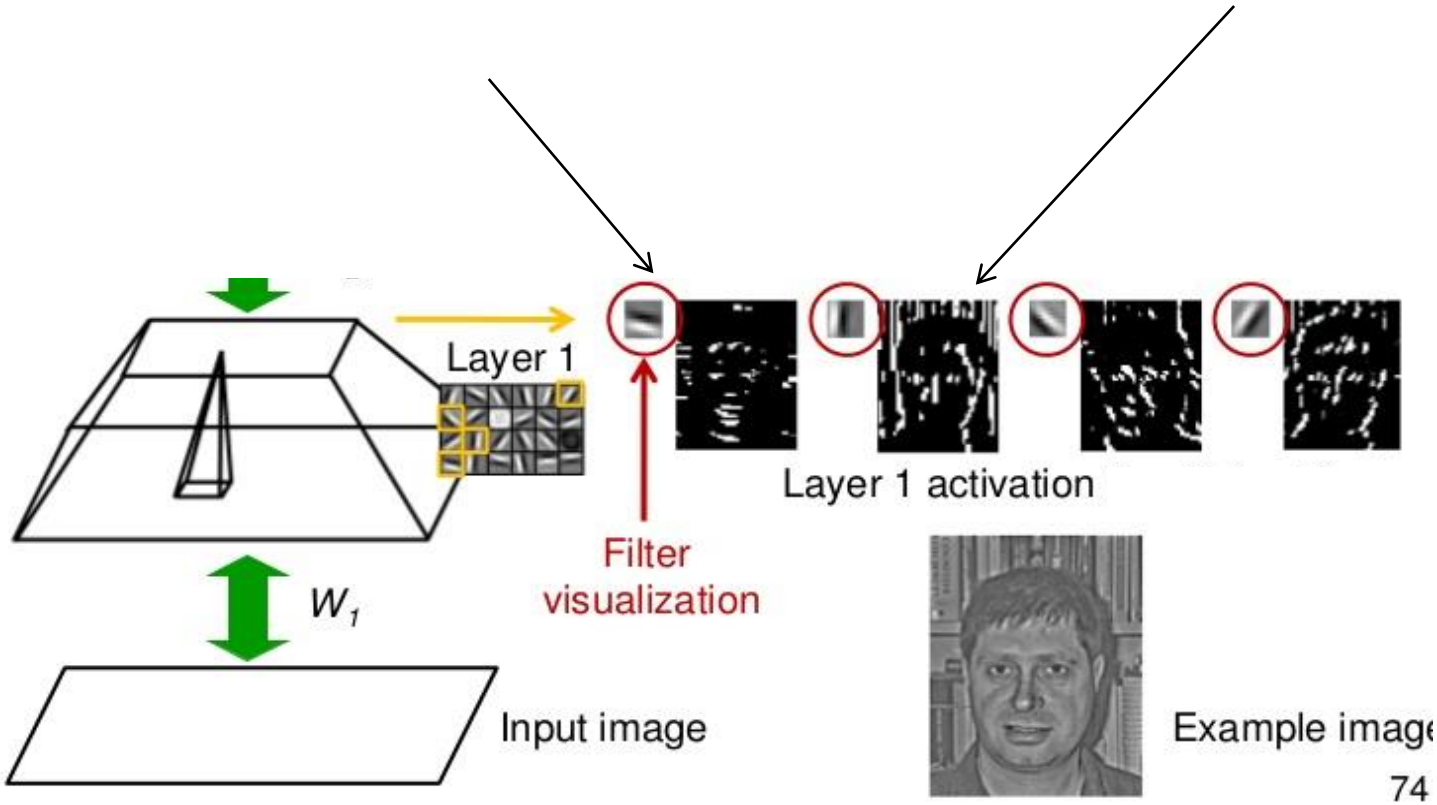
Here's the new part: "weight sharing" -- the same feature is learned no matter the patch



Visualization

A picture of the field of pixels that maximally activates this particular node

White indicates a node with a large activation.



http://www.slideshare.net/zukun/p04-restricted-boltzmann-machines-cvpr2012-deep-learning-methods-for-vision

Considerations

- The default "stride" in keras is 1
- "Receptive fields" are usually square with od sides – 3×3 or 5×5, say
- Stride 1 on a 3×3 turns an $n \times p$ matrix into an $(n - 2) \times (p - 2)$; sometimes the input is "padded" with two extra rows and columns of zeros to produce $n \times p$ output
  - In our example, stride 2 on 3×3 didn't quite work for a 16×16 matrix

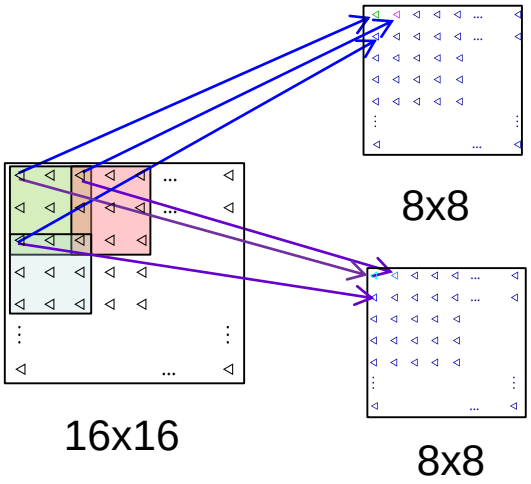
Convolutional Neural Network (CNN)

- Nnet- 4 one convolutional layer hidden layer.
- Nnet-5 two convolutional layers.

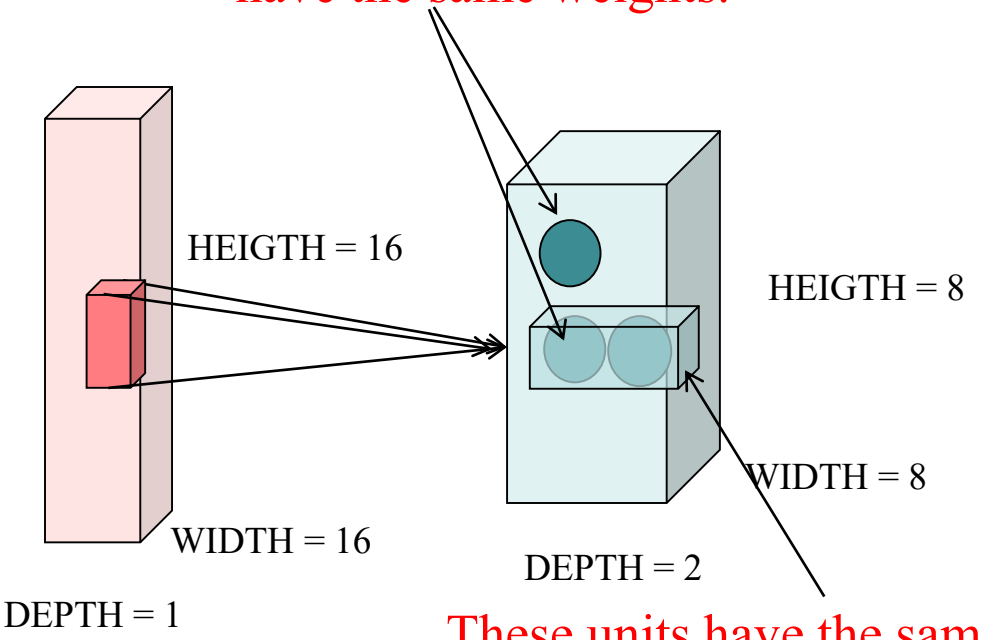
Model	#Links	#Wgts	Test Accuracy
Net-1:no hidden layer	2570	2570	80.0% correct
Net-2: one hidden layer	3214	3214	87.0% correct
Net-3: Locally Connected	1226	1126	88.5% correct
Net-4:Covolutional	2266	1132	94.0% correct
Net-5:Convolutional	5194	1060	98.4% correct

(More nodes in layer 2, but with weight sharing)

Representing CNN Layers as Blocks



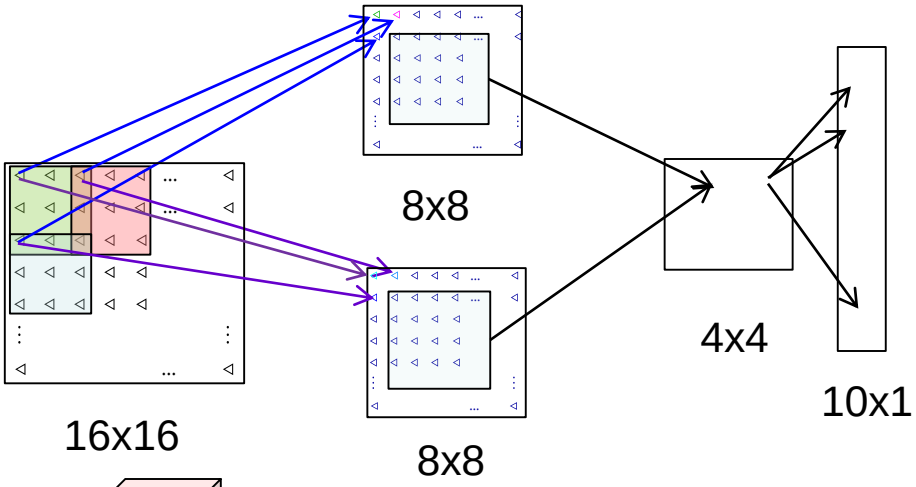
Nodes of the same depth have the same weights.



These units have the same local receptive fields in the input layer.

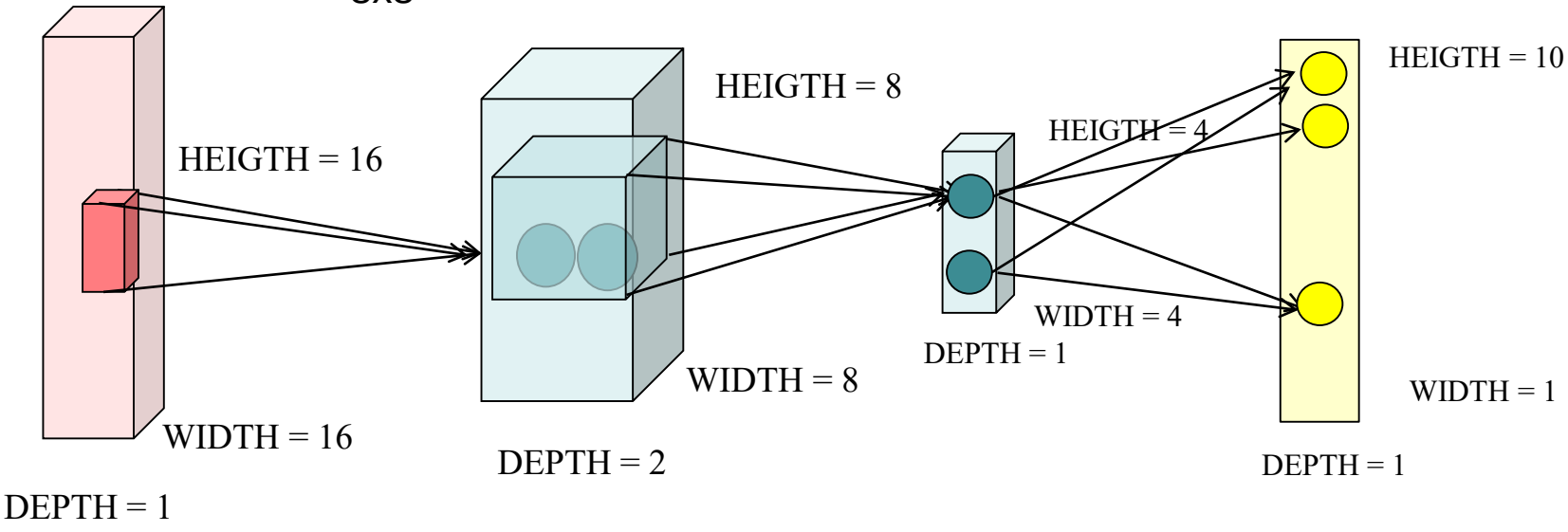
Color image would have depth 3

Representing CNN Layers as Blocks

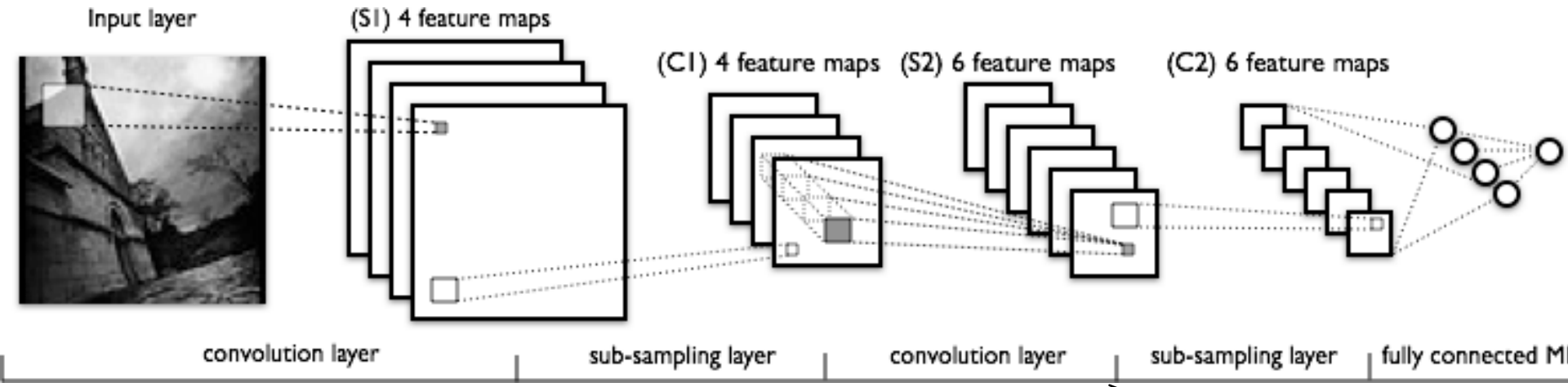


Second 4 x 4 Hidden Layer: Local 5 x 5 (x2) Receptive Field with stride 1, but no weight sharing.

Output Layer: Fully Connected



A picture of a CNN

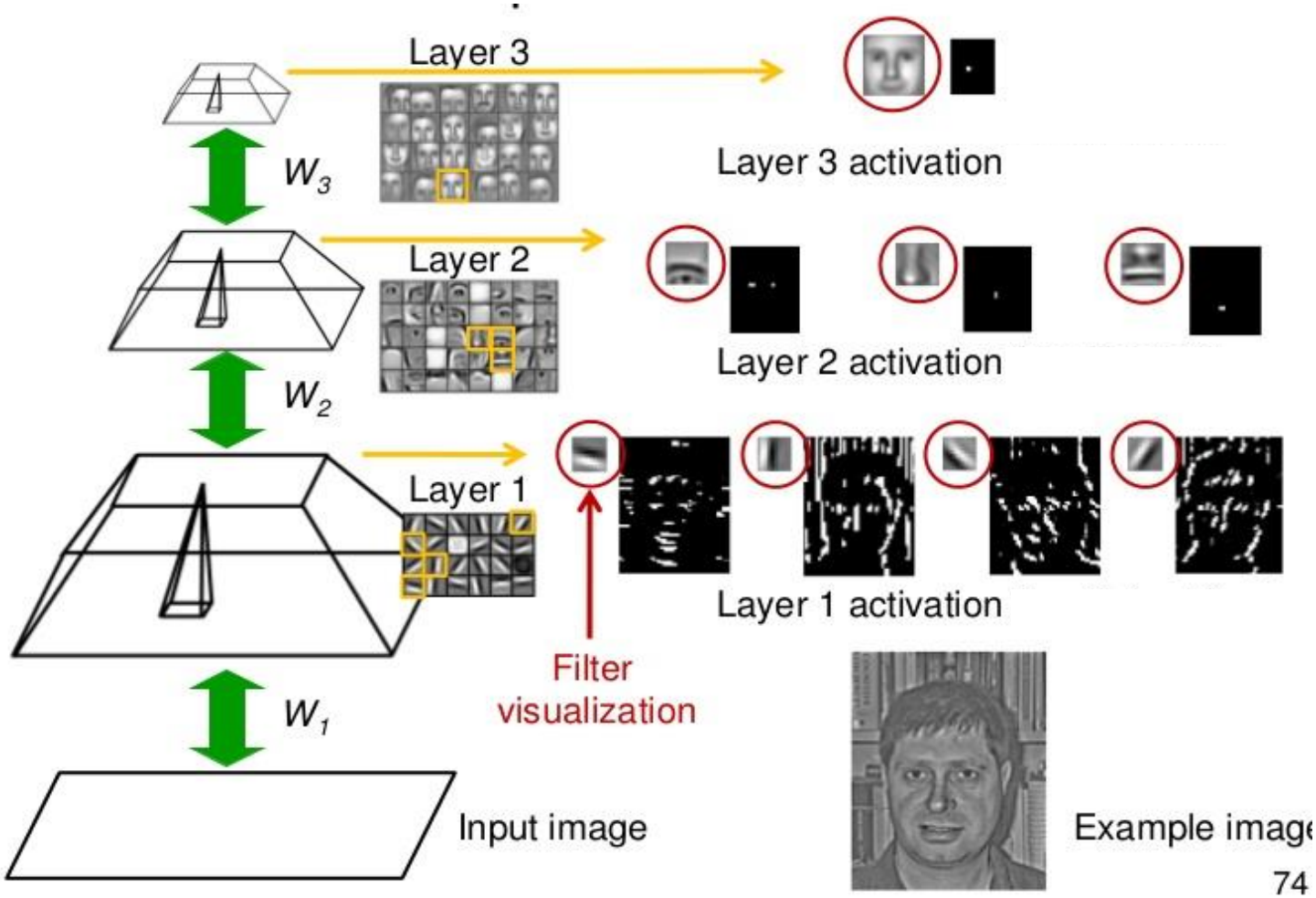


Max pooling (sub-sampling layer). Replaces squares of pixels by a single value, the max pixel in the square:
Reduces computation
Helps translation invariance

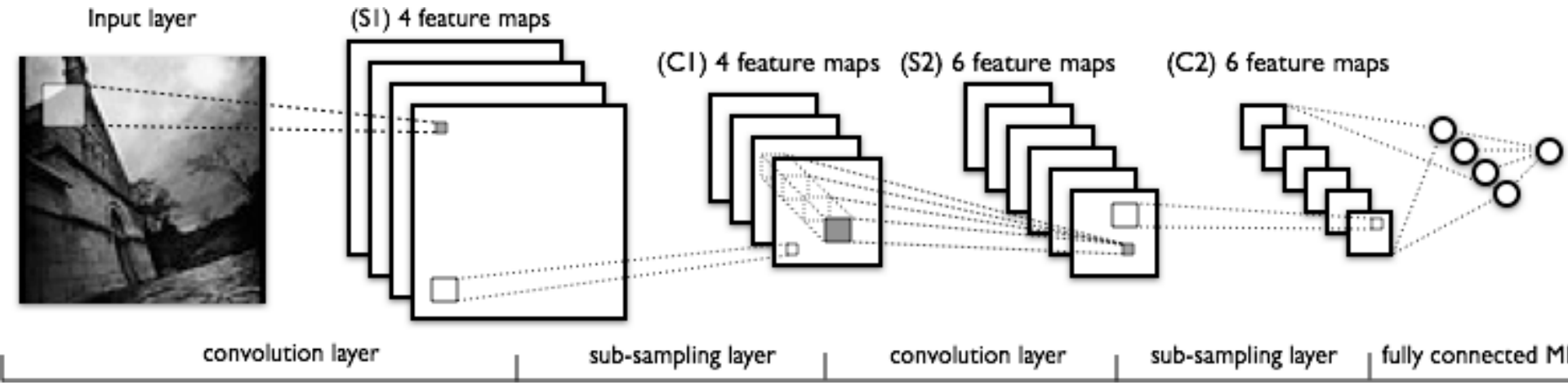
At the end "flatten" the features into a vector of features, fully connected to the output units.

http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

Visualizing the output of nodes from deeper layers



Trends



- Deep
- Small Filter Sizes 3x3 and stride = 1; ReLu activation
- Small Pooling (“sub-sampling layer”) Sizes 2x2
- Padding so that height and width don’t decrease too fast

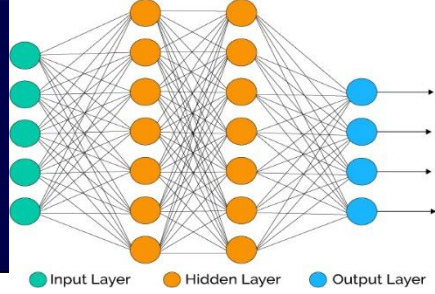
This is a nice demo of an MNIST CNN

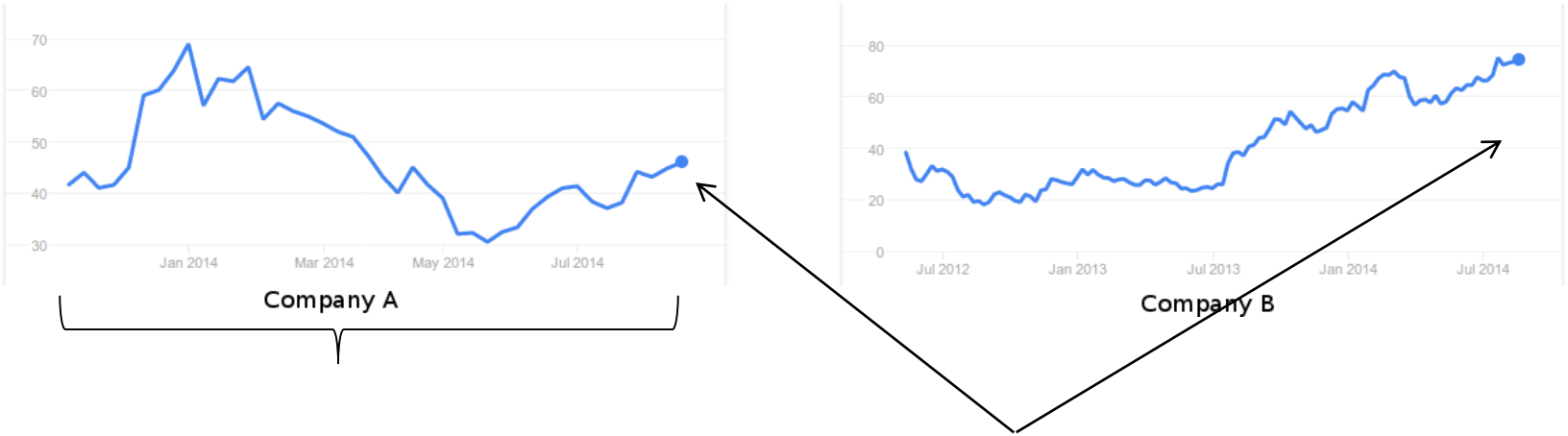
http://scs.ryerson.ca/~aharley/vis/

Recurrent Neural Networks

- An **RNN** is a network designed to capture information that changes with time
- Often the input and output will be in different domains – e.g., input is a sequence of sounds, output is a sequence of words
- It uses a layered network that **reuses** weights across time steps
- The network can “maintain state” – which means it can “remember” past relationships

Predict Stock Prices





Target? y = today's stock price

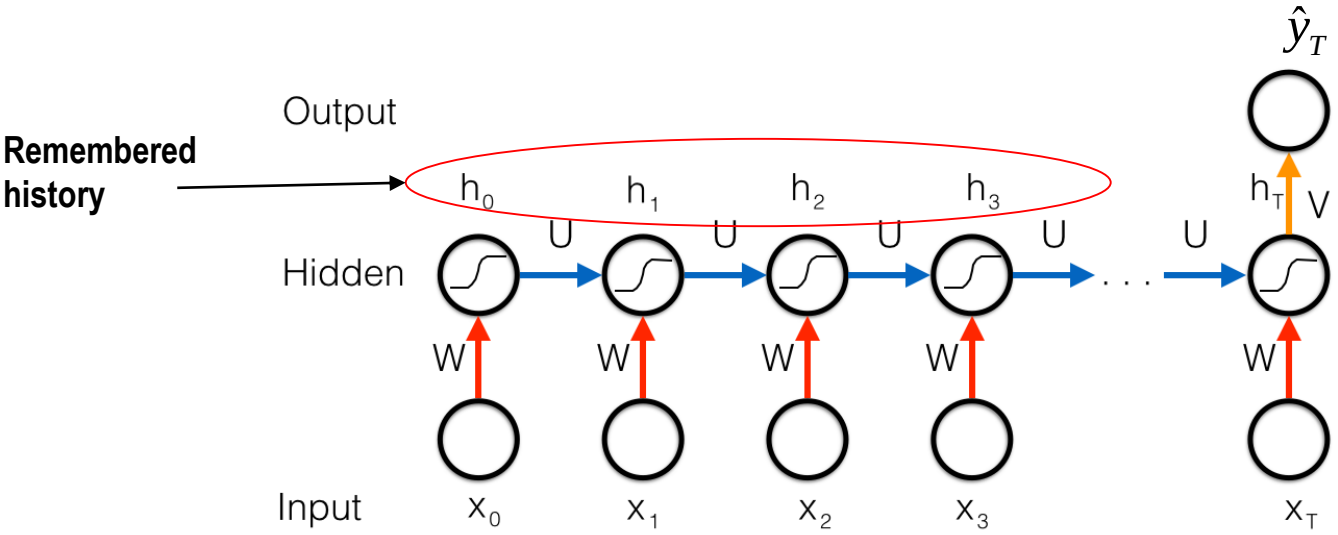
Input? x's are prices since IPO,
but Company B's history is longer than Company A.

Recurrent Neural Network
Weights to be trained are W, U, V

We want to predict the stock price for today, day T

$\hat{y}_T$ = RNN output for today;

y_T = target for today (unkown until the end of the day)



Simple Inputs:

x_0 = input for the initial day,

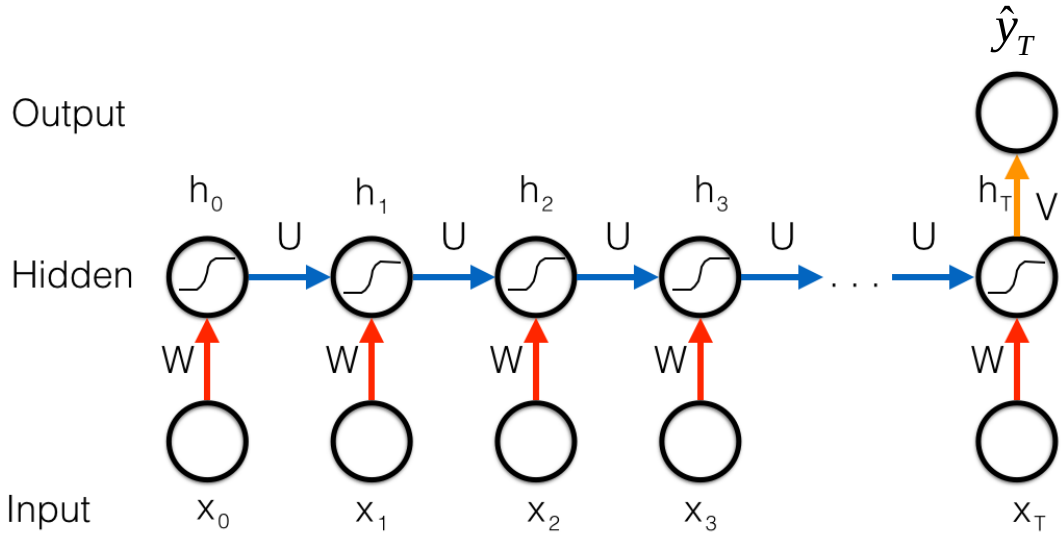
x_1 = input for "day" 1, will be y_0

⋮

x_T = input for today, T will be y_{T-1}

Recurrent Neural Network
The RNN is more general that it looks

- "T" can be any length needed for compay A or B.
- The inputs are typically a vector of inputs.
- A "day" is a window of time. E.g. here if we may wish to update weekly. Then each t may be the Moday of that week; the input for week t could be the vector of stock prices for the previous week.

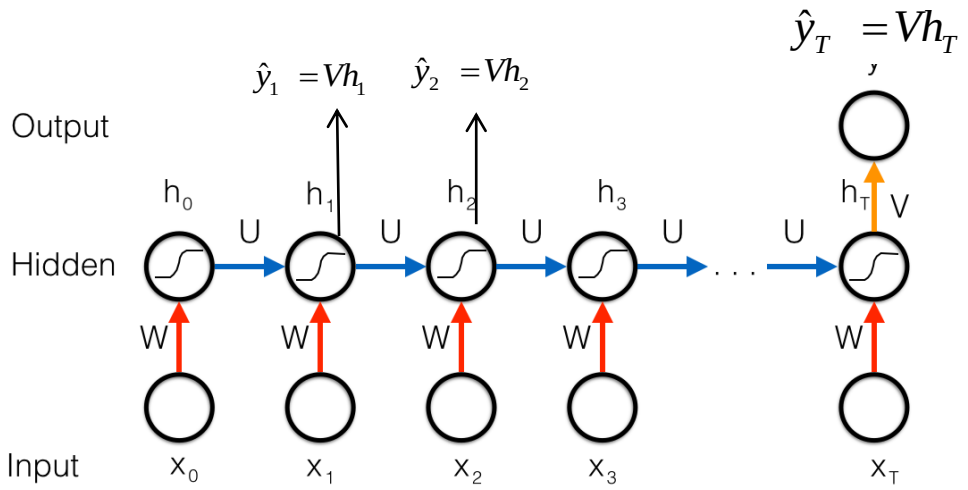


Training the RNN, use BPTT (back propagation through time)

Find W, U, V to minimize $E = \sum_{t=1}^{T-1} (y_t - \hat{y}_t)^2$

(with one example series, but in general, you will have more).

$$h_0 = f(Wx_0)$$
$$h_1 = f(Uh_0 + Wx_1)$$
$$\vdots$$
$$h_t = f(Uh_{t-1} + Wx_t)$$
$$\vdots$$
$$\hat{y}_T = Vh_T$$



$\hat{y}_T$ = today's predicted price (output)

Example that combines a CNN and a RNN

Vinyals, Toshev, Bengio, Erhan (2016) (also Karpathy, Fei-Fei (2015))

Better Image Model

InitialModel: A close up of a plate of food on a table

BestModel: A bunch of bananas and a bottle of wine.

Colors

InitialModel: A train that is sitting on the tracks.

BestModel: A blue and yellow train traveling down train tracks.

Better Image Model

InitialModel: A close up of a person eating a hot dog.

BestModel: A woman holding a banana up to her face.

Counting

InitialModel: A brown bear is swimming in the water.

BestModel: Two brown bears sitting on top of rocks.

Training Set

- Start with supervised images, i.e. images *I*, with a one sentence description, $S = (S_0,...,S_N)$ with *N-1* words.

  MicroSoft Common Objects in Context (MS COCO) 2014 data set

The man at bat readies to swing at the pitch while the umpire looks on.

A large bus sitting next to a very tall building.

- *N* varies from image to image and each sentence ends with a "stop word".

Represent the image and each word by vectors of the same length

- Run the I through a CNN, say one trained using ImageNet but, replace the output layer with a layer of p nodes.

  The output vector (x_{-1})will be used as the description of I.

  We have "embedded" the image into a p - dimensional space.

- Also embed each word (represented by a one - hot vector with length equal to the dictionary size, V) into a vector of length p.

  I.e. the embedded word vector x_t for the t^{th} word is :

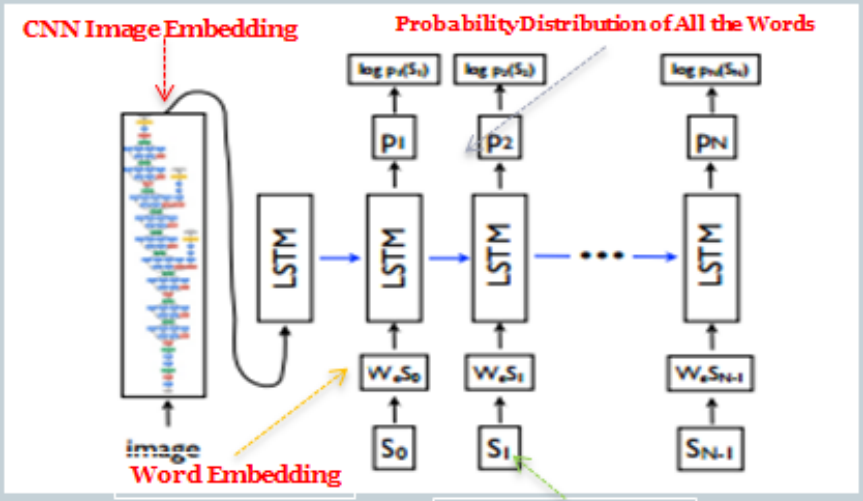
$$x_t = W_e^{p \times V} S_t$$

(they train the weights for the image embedding CNN and they train the weights for the word embedding)

Now put it all together
TensorFlow Implementation
https://github.com/tensorflow/models/tree/master/im2txt

- The embedded image vector is the input to the first RNN
- After that, each word embedding is used in turn as an input to each RNN node.

Here the RNN nodes are called "LSTM cells", more on that later.



$$
\begin{aligned}
x_{-1} &= \mathrm{CNN}(I) \\
x_t &= W_e S_t, \quad t \in \{0 \ldots N-1\} \\
p_{t+1} &= \mathrm{LSTM}(x_t), \quad t \in \{0 \ldots N-1\}
\end{aligned}
$$

The output is a vector of (softmax) probabilities of length V, used to choose the next word.

TensorFlow Implementation
https://github.com/tensorflow/models/tree/master/im2txt

- Turns out, using the image only at the beginning worked best.
- Start with CNN weights from a model trained on ImageNet.
  Update the RNN weights, holding the CNN weights fixed.
  Then fine - tune the CNN weights, because otherwise, the image embedding won't pick up e.g. colors.
- Training uses the true word in the caption at each step. These clearly are not known when it comes time to predict for a new image. So they train with a combination of actual words and predicted words.
- Their best model uses an ensemble of CNN - RNN trained models.
- *RNNs that translate from one language to another use a similar approach.*

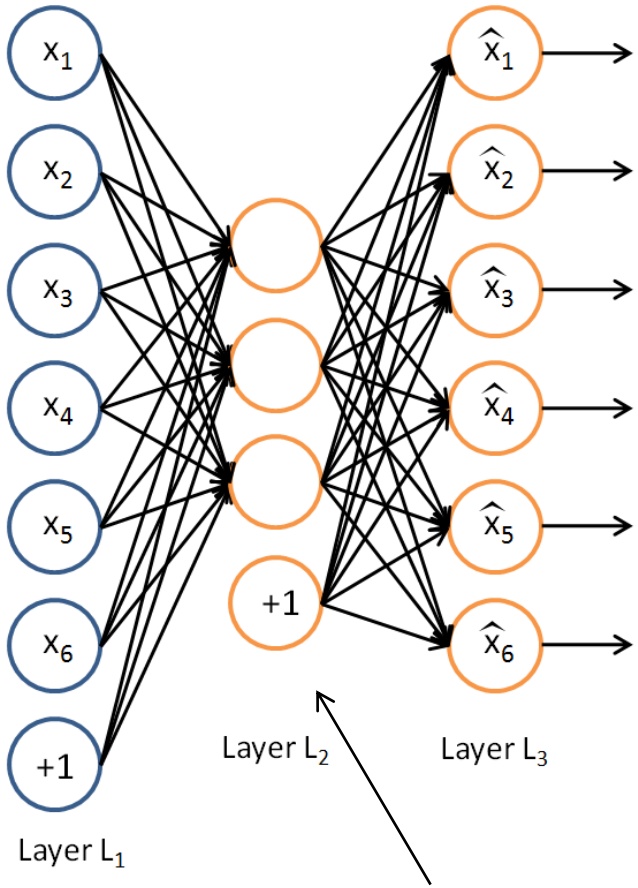
Autoencoding

- An **unsupervised** method to represent (embed or encode) high-dimensional examples in a lower-dimensional space
- We have seen examples of this operation from Principal Components and multidimensional scaling (e.g. t-SNE)
- Some of those methods made it tricky to assign new observations into the map

Autoencoding

- Auto-encoding uses Neural Networks to generate non-linear features from the input features
- It includes the mapping which can be used to encode new observations
- In one common implementation, we fit a supervised NN and then use one of the hidden layers as the “encoder”
- But what if you don’t have responses?

Unsupervised Data has no y's. To train, each observation's target y's will be equal to the input x's



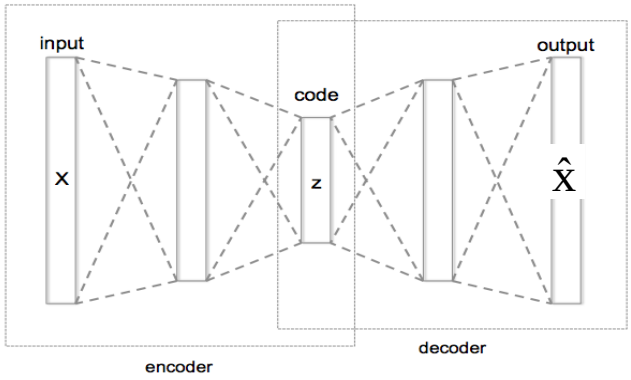
The targets (the y's) are the same as the input features (the x's).

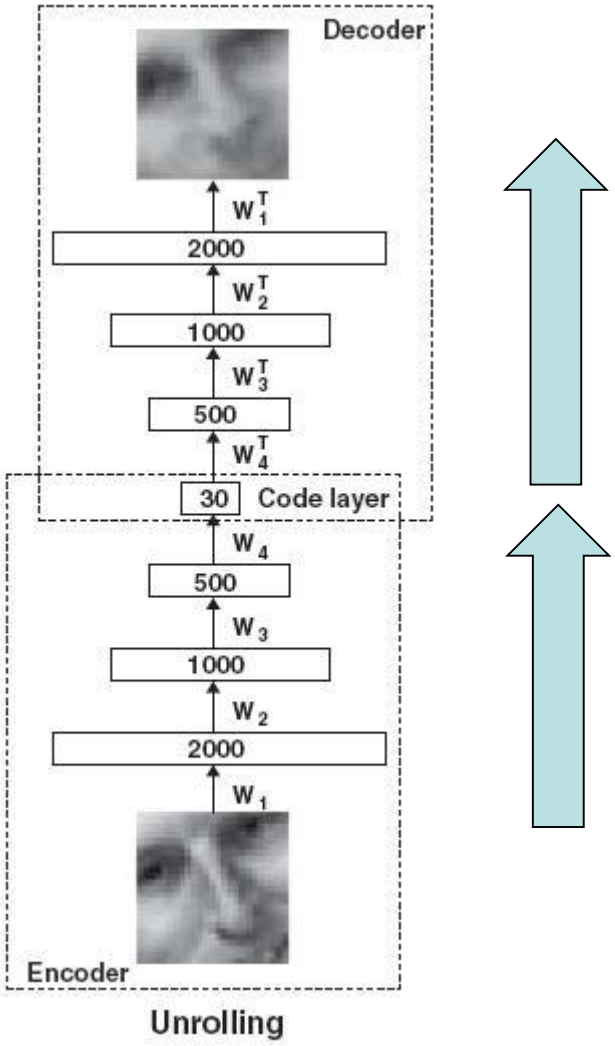
The NN outputs are estimates of the inputs

The hidden layer nodes are the new features (the codes). They are (non-linear) functions of the input x's

Deep Auto-encoding

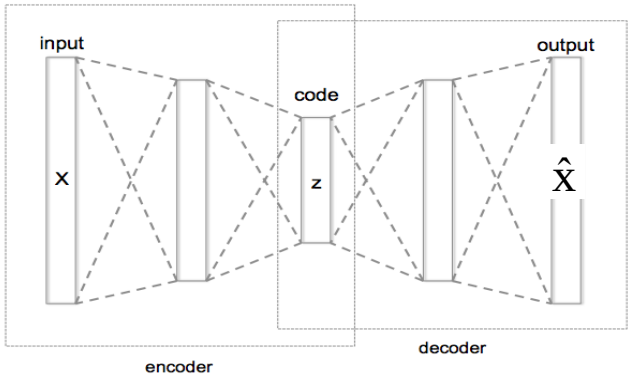
- The auto-encoder can be a deep neural network too.
- This way the codes (learned features) are more complicated non-linear functions of the original.

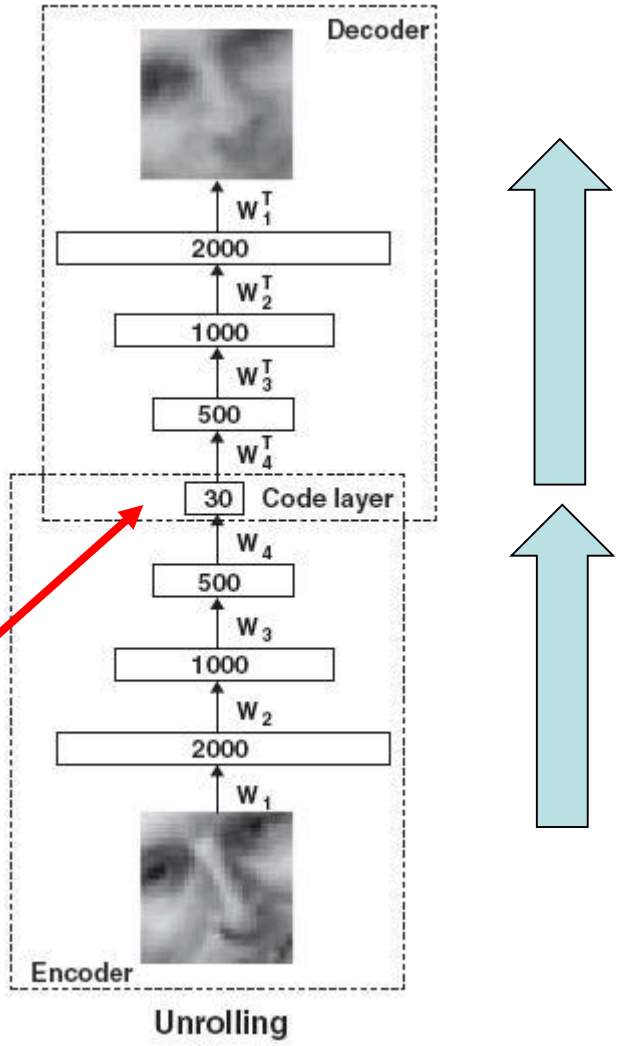




Deep Auto-encoding

- The auto-encoder can be a deep neural network too.
- This way the codes (learned features) are more complicated non-linear functions of the original.



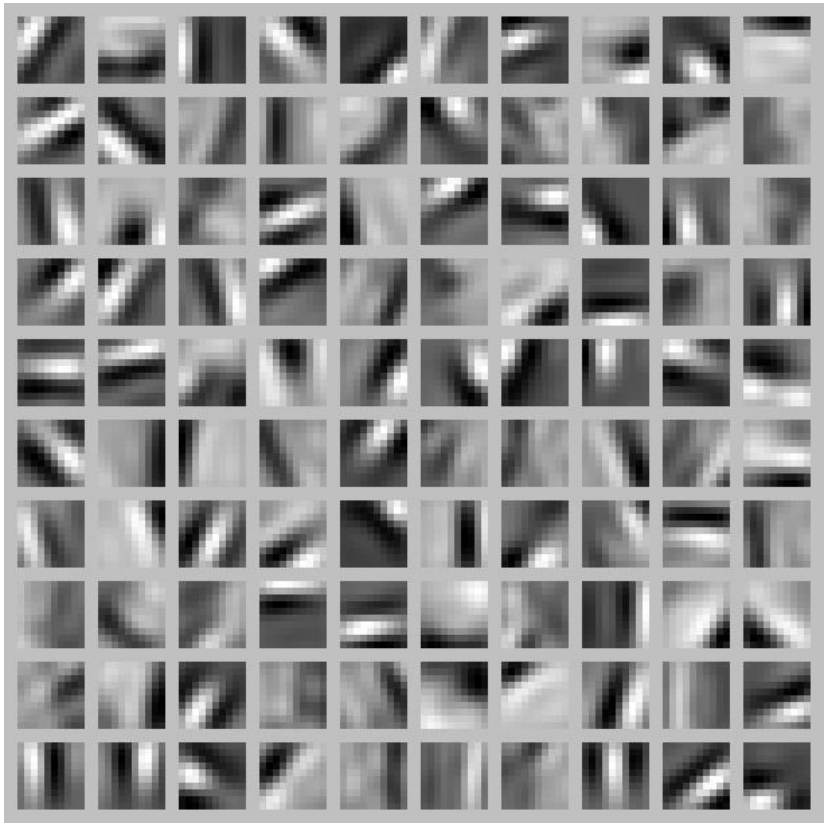


Codes

Training

- The new features, the **codes, are numeric**.
- The error function is usually a **squared error** function.
- Training is tricky. We don't want the auto-encoder to learn the identity function.
- An approach is to require the number of hidden units H, to be much less than the number of input units, p. Another is to let H >> p, but impose some type of regularization, etc. (Restricted Boltzmann Machines, RBM)

So what do auto-encoders learn for images?



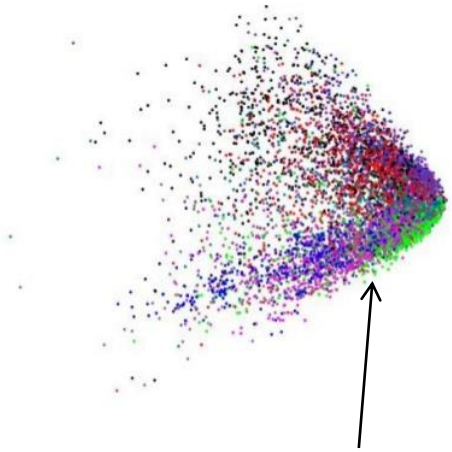
100 “learned” features taken from an auto-encoder. They look a lot like the first sets of features learned using supervised image data!

Uses of Auto-encoding

- Anomaly detection: if observed and predicted are far apart…
- Dimension reduction – maybe for visualization
- To train the first several layers of a supervised NN using unsupervised examples. This increases the effective size of the training set for supervised learning with neural networks (especially for computer vision tasks and text mining tasks)

Application: Use the learned features (codes) to reduce dimension for visualization.

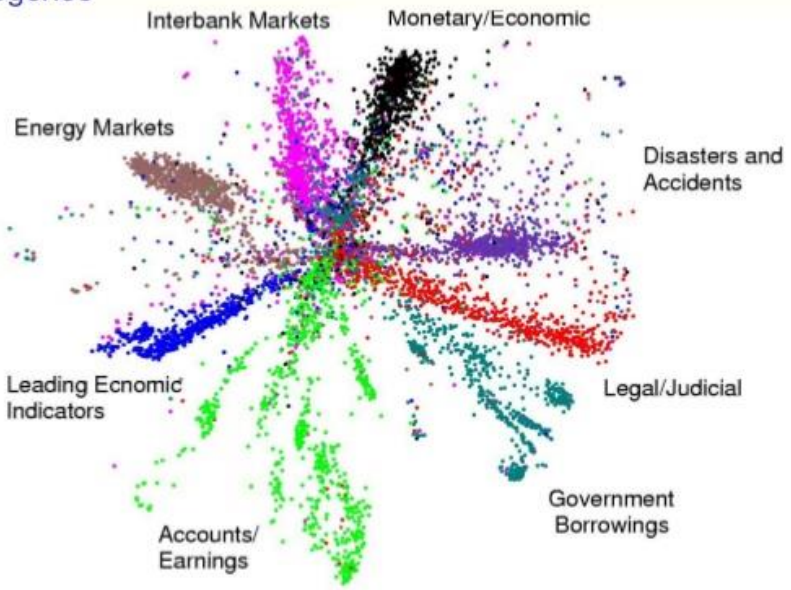
First compress all documents to 2 numbers using a type of PCA
Then use different colors for different document categories



In general, PC's are not great for identifying clusters

2000 – 250 – 50 – **2** – 50 – 250 – 2000
Auto-encoded Network

First compress all documents to 2 numbers with an autoencoder
Then use different colors for different document categories



Is this a good data reduction technique?

- A shallow network with a linear activation function is more like PC. Deeper networks, with non-linear activation functions seem to do better here.

- We really like the fact that unlike many dimension reduction techniques (e.g. multi-dimensional scaling), we get an encoder that can be used on **new** observations.

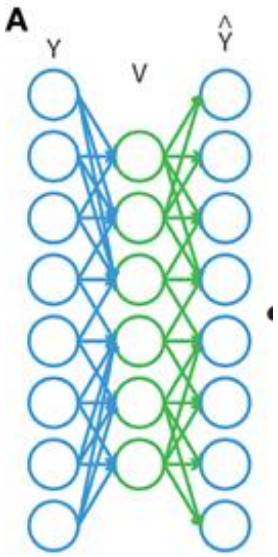
- And, since NN are already easily scaled to very large problems, we don't need new software or ideas to use NN to reduce dimensions.

- Is this the best way to reduce the dimension of the original features for visualization? t-SNE is more popular.

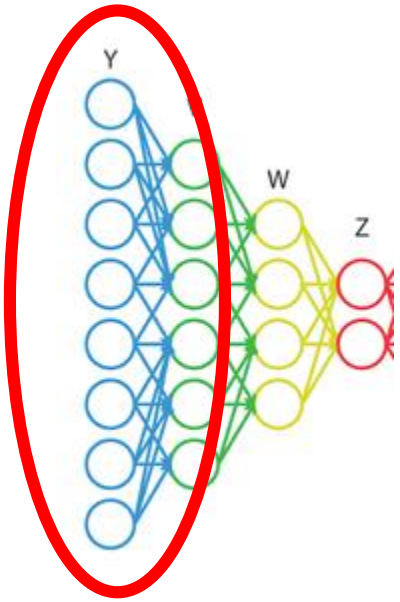
Application: Use the weights from unsupervised auto-encoding to pre-training supervised neural networks

- Deep neural networks take lots of supervised data which is typically much harder to come by than unsupervised data (e.g. images, text, etc.)

A. Unsupervised training. Use the large unsupervised data set to train the weights from the input (blue) to the code layer (green).



B. Start with the weights trained in A. Add to the NN, and use the **supervised data.** Add the outputs and train the entire network.



Neural Network Recap

- A neural network is a statistical model with **lots** of parameters
  - With a continuous activation we can fit w/backprop
- NNs can be used for regression or classification, or serve as an auto-encoder for use in clustering, MDS or something else
- Neural networks are hard to fit, so transfer learning is great if you can get it
- CNNs are specific types of net used for images; RNNs are specific types used for time series data
- This is a rich and fast-evolving area!