Machine Learning

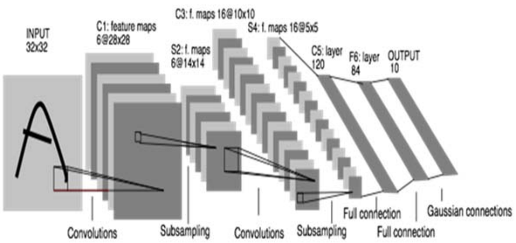
10-701, Fall 2015

Deep Learning

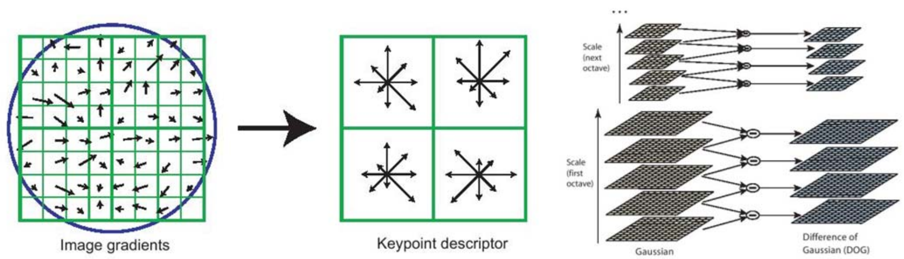
Eric Xing

(and Pengtao Xie)

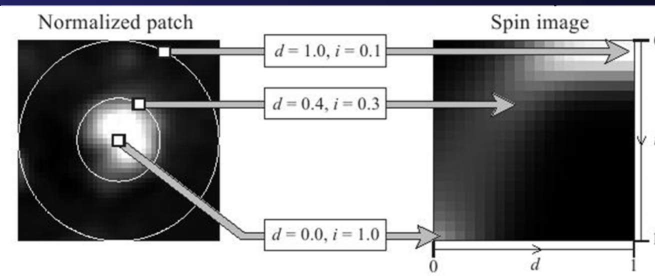
Lecture 8, October 6, 2015



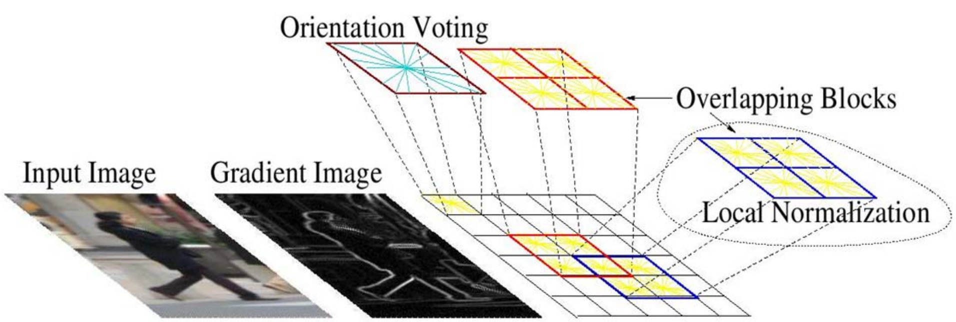
A perennial challenge in computer vision: feature engineering

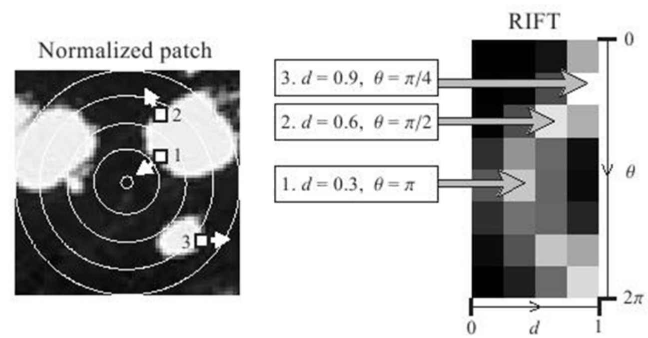


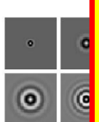
SIFT



Spin image

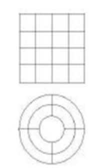






Drawbacks of feature engineering

1. Needs expert knowledge
2. Time consuming hand-tuning

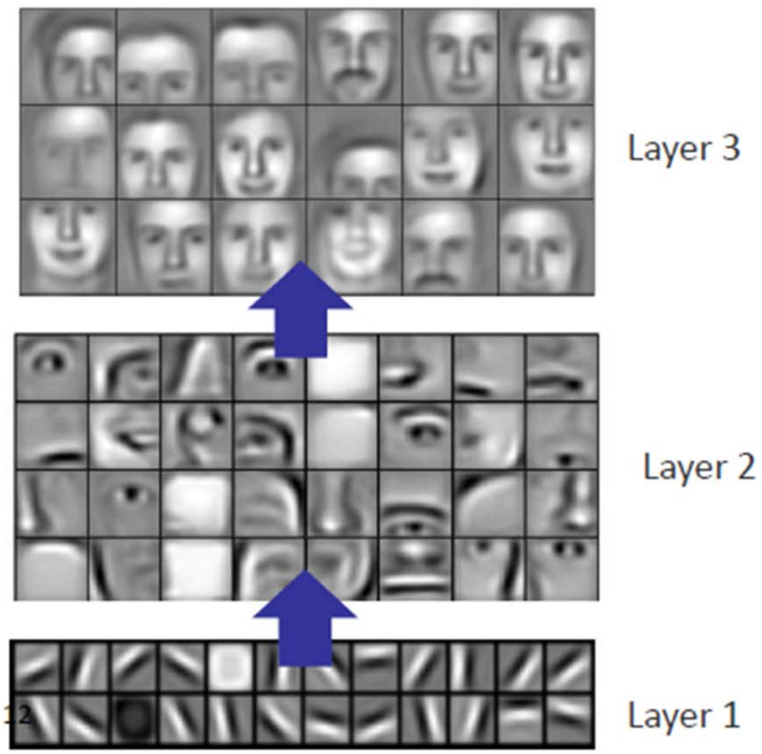


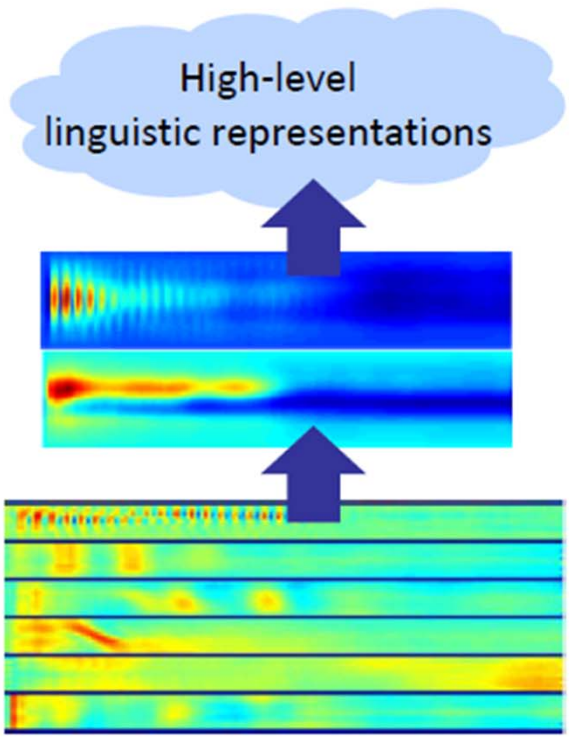
Textons

and Ng

Automatic feature learning?

- Successful learning of intermediate representations
  [Lee et al ICML 2009, Lee et al NIPS 2009]

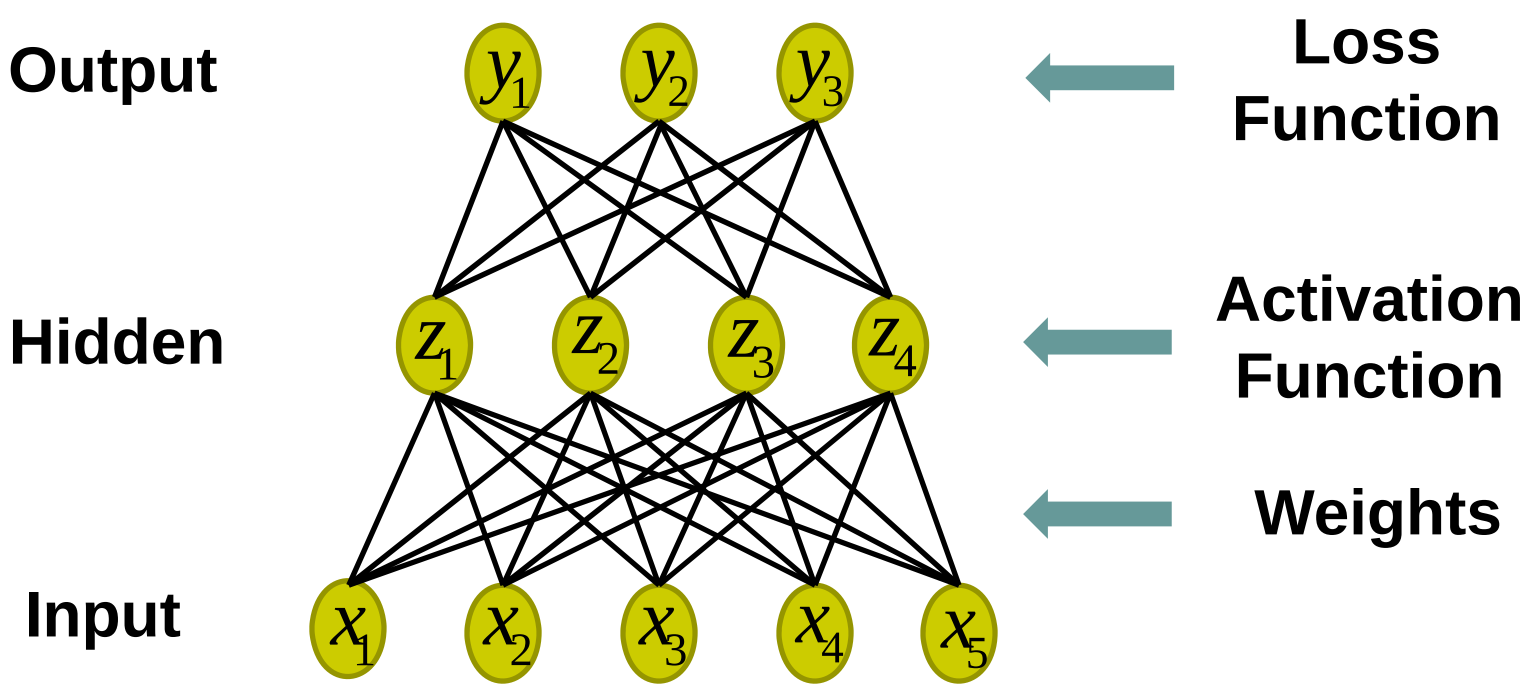




“Deep” models

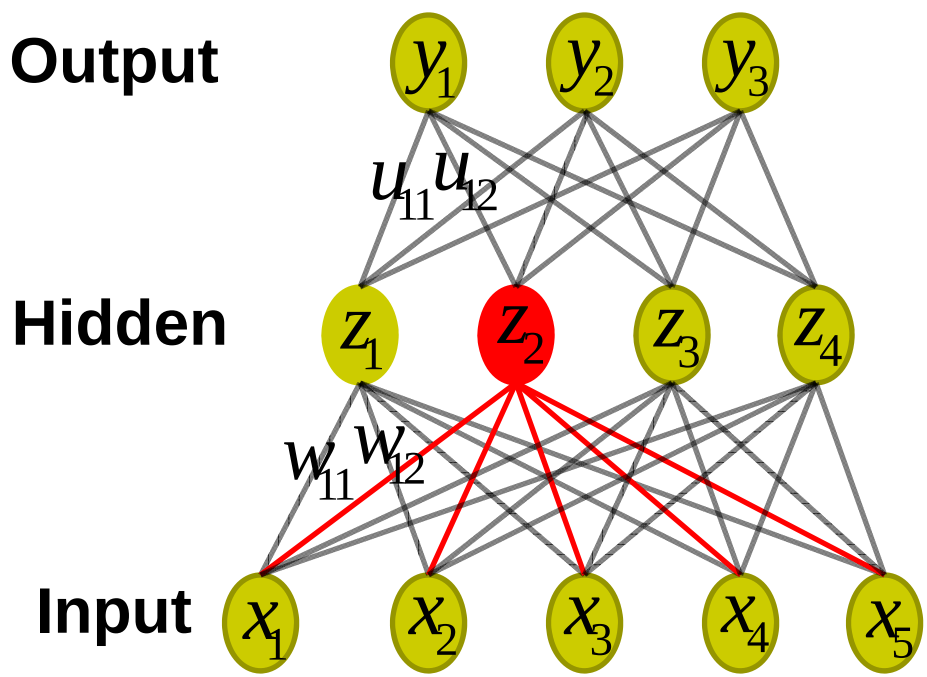
- Neural Networks: Feed-forward*
  - You have seen it
- Autoencoders (multilayer neural net with target output = input)
  - Non-probabilistic -- Directed: PCA, Sparse Coding
  - Probabilistic -- Undirected: MRFs and RBMs*
- Convolutional Neural Nets
- Recursive Neural Networks*

Neural Network



Local Computation At Each Unit

Linear Combination + Nonlinear Activation



Activation Function

units in the lower layer

$$z_2 = \sigma\left(\sum_{i=1}^{5} w_{2i} x_i\right)$$

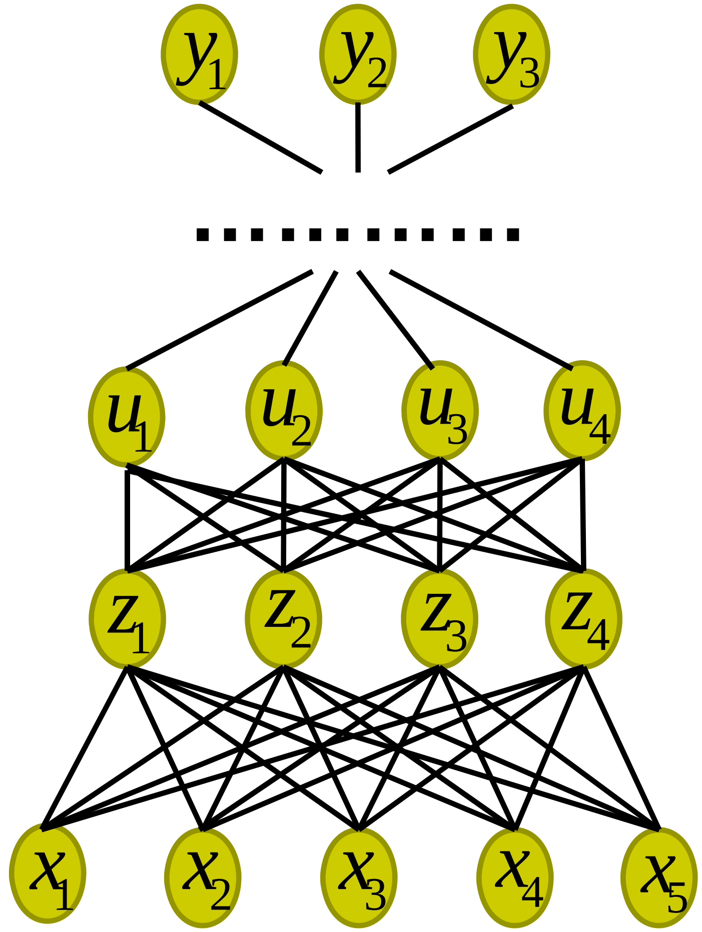
Weights connecting to units in the lower layer

Deep Neural Network

Output

More Hidden Layers

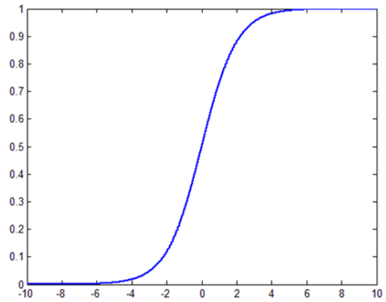
Input



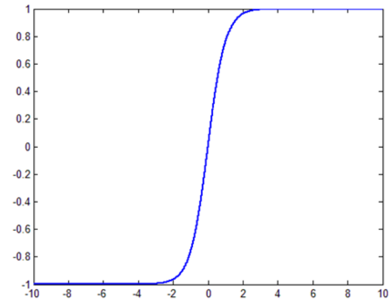
Activation Functions

- Applied on the hidden units
- Achieve nonlinearity
- Popular activation functions

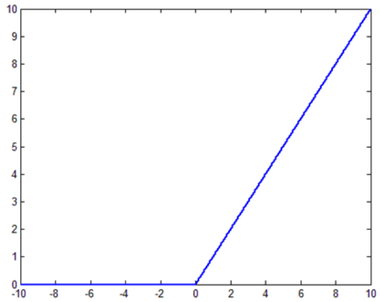
Sigmoid



Tanh



Rectified Linear



Loss Functions

- Squared loss for regression

$$(\boldsymbol{y} - \boldsymbol{t})^2$$

Prediction True value

- Cross entropy loss for classification

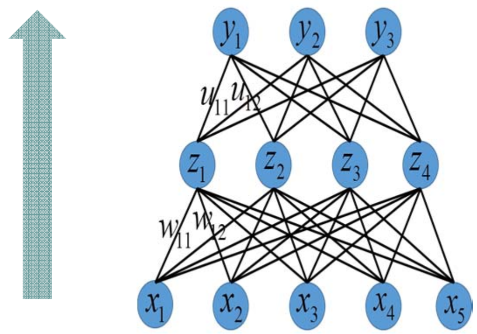
$$-\sum_{k=1}^{K} t_k \ln a_k \quad a_k = \frac{\exp(y_k)}{\sum_{j=1}^{K} \exp(y_j)}$$

Class label

Prediction

Neural Network Prediction

- Compute unit values layer by layer in a forward manner



- Prediction function

Activation Function

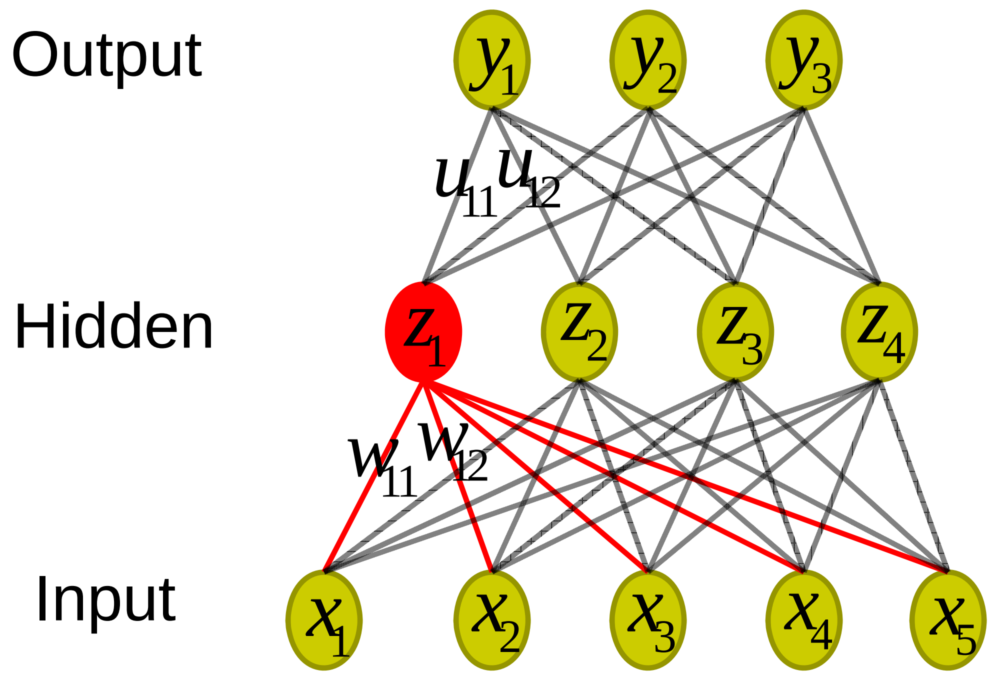
Input

$$y_k = \sum_{j=1}^{4} u_{kj}\,\sigma\left(\sum_{i=1}^{5} w_{ji} x_i\right)$$

Output

Weights

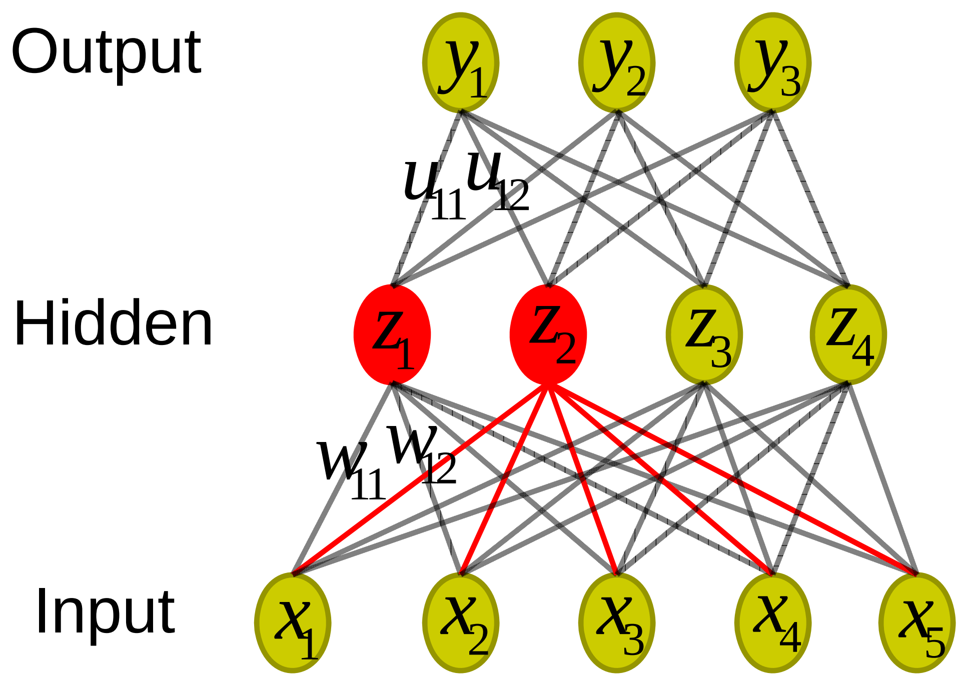
Neural Network Prediction



$$z_1 = \sigma\left(\sum_{i=1}^{5} w_{1i} x_i\right)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

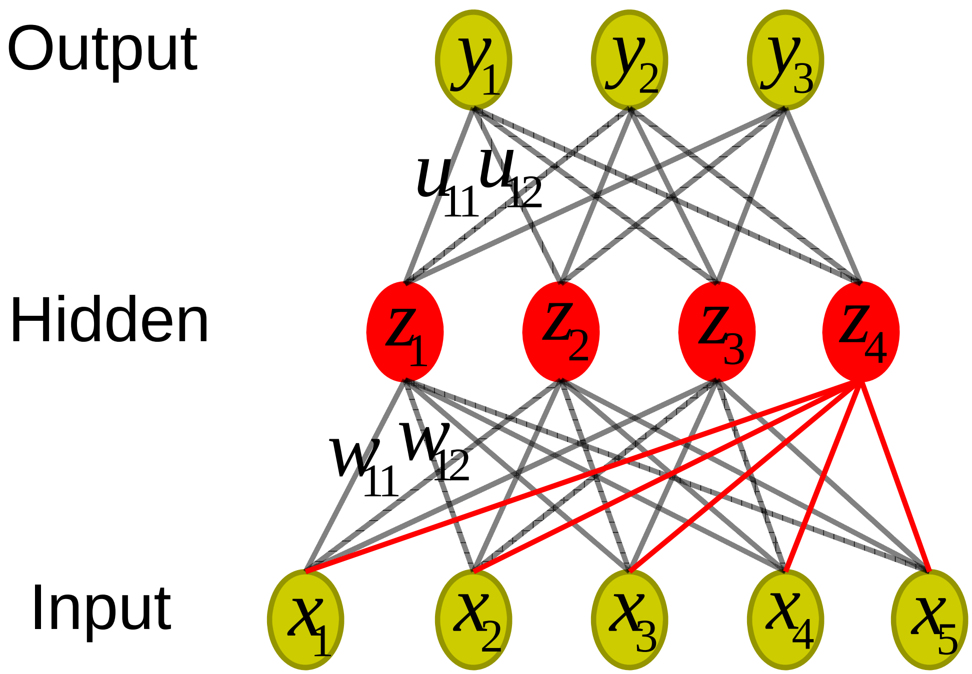
Neural Network Prediction



$$z_2 = \sigma\left(\sum_{i=1}^{5} w_{2i} x_i\right)$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

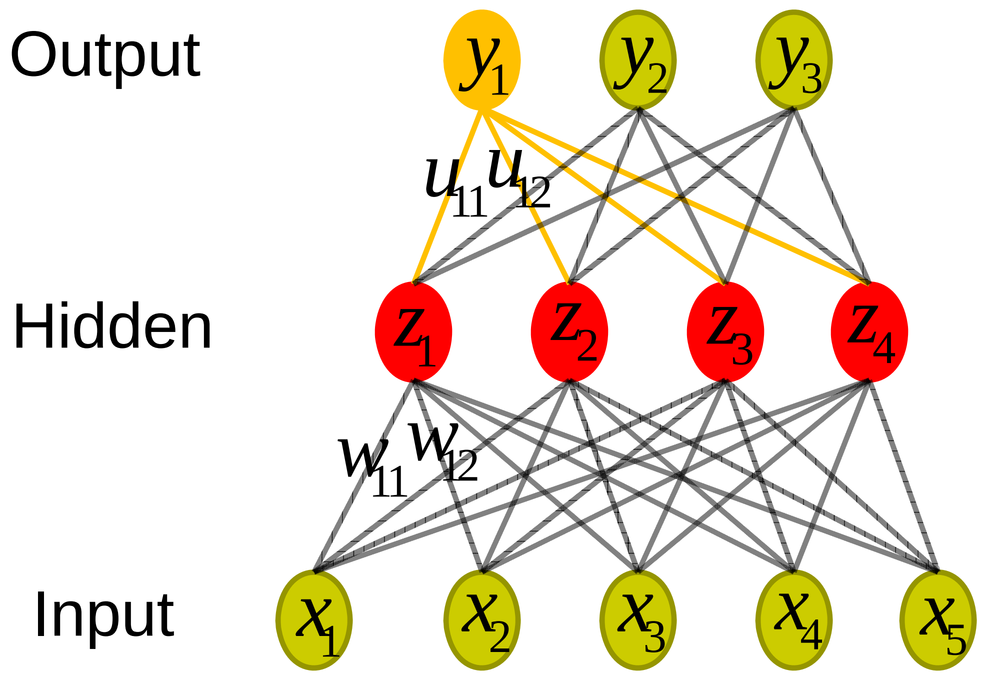
Neural Network Prediction



$$z_4 = \sigma\left(\sum_{i=1}^{5} w_{4i} x_i\right)$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

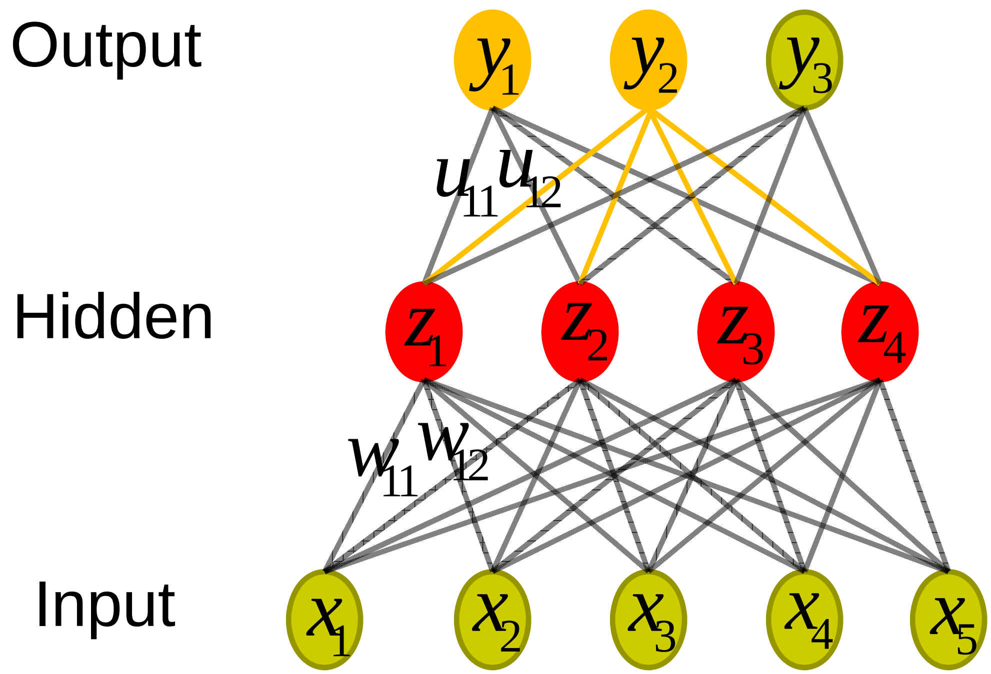
Neural Network Prediction



$$y_1 = \sigma\left(\sum_{i=1}^{4} u_{1i} z_i\right)$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

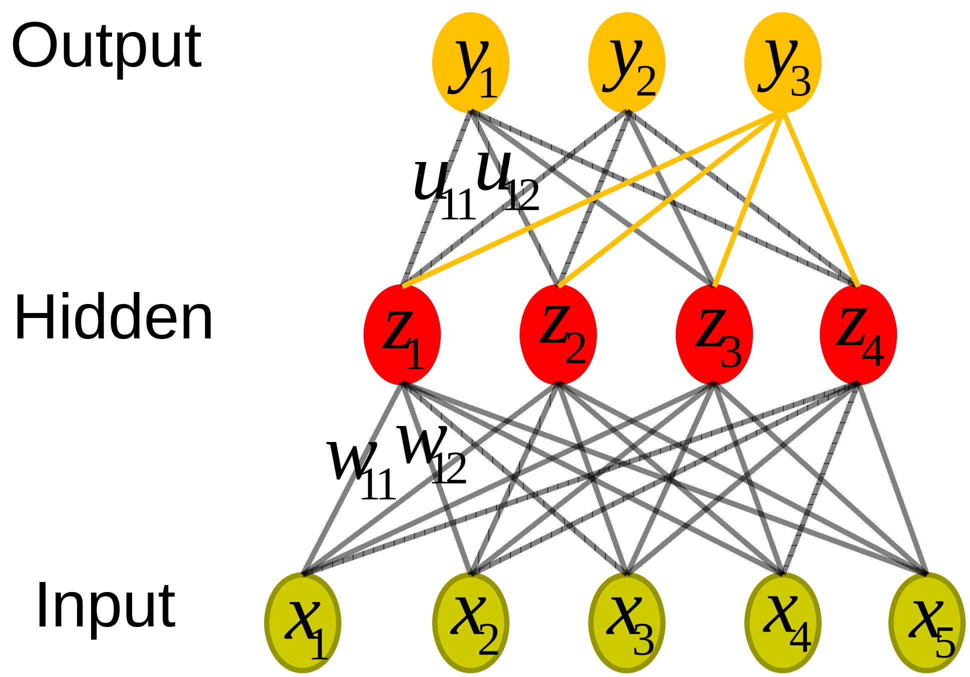
Neural Network Prediction



$$y_2 = \sigma\left(\sum_{i=1}^{4} u_{2i} z_i\right)$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Neural Network Prediction



$$y_3 = \sigma\left(\sum_{i=1}^{4} u_{3i} z_i\right)$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Neural Network Training

- Gradient descent
- Back-Propagation (BP)
  - A routine to compute gradient
  - Use chain rule of derivative

Neural Network Training

- Goal: compute gradient

$$\frac{\partial L}{\partial w_{ij}}$$

Training loss

Weight between unit i and j

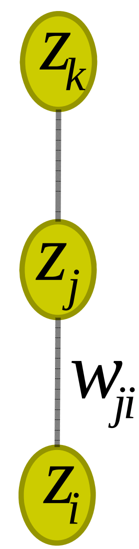
- Apply chain rule

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial a_j}\frac{\partial a_j}{w_{ji}}$$

Linear combination value $a_j = \sum_i w_{ji} z_i$

$$\boxed{\frac{\partial L}{\partial a_j}} = \sum_k \boxed{\frac{\partial L}{\partial a_k}}\frac{\partial a_k}{a_j}$$

Called error, computed recursively in a backward manner



Neural Network Training

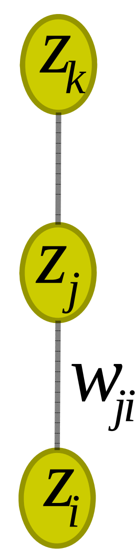
- Apply chain rule (cont'd)

$$\frac{\partial a_j}{w_{ji}} = z_i \longleftarrow$$ Derivative of activation function

$$\frac{\partial a_k}{\partial a_j} = \frac{\partial a_k}{\partial z_j}\frac{\partial z_j}{\partial a_j} = w_{kj}\sigma'(a_j)$$

$$\boxed{\text{gradient}=\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial a_j}\frac{\partial a_j}{w_{ji}}=\text{backward error x forward activation}}$$

- Pseudo code of BP



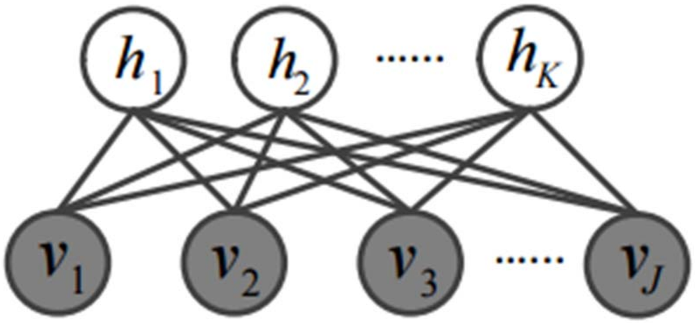
While not converge

1. compute forward activations
2. compute backward errors
3. compute gradients of weights
4. perform gradient descent

Pretraining

- A better initialization strategy of weight parameters
  - Based on Restricted Boltzmann Machine
  - **An auto-encoder model**
  - Unsupervised
  - Layer-wise, greedy
- Useful when training data is limited
- Not necessary when training data is rich

Restricted Boltzmann Machine

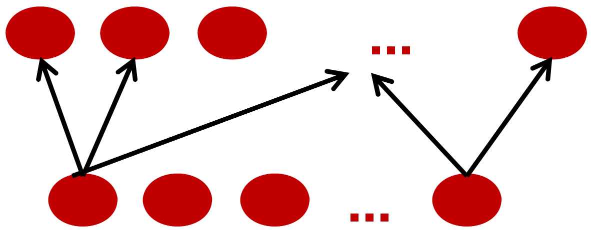


$$E(\mathbf{h}, \mathbf{v}) = -\sum_{j=1}^{J} \alpha_j v_j - \sum_{k=1}^{K} \beta_k h_k - \sum_{j=1}^{J} \sum_{k=1}^{K} A_{jk} v_j h_k$$

Layer-wise Unsupervised Pre-training

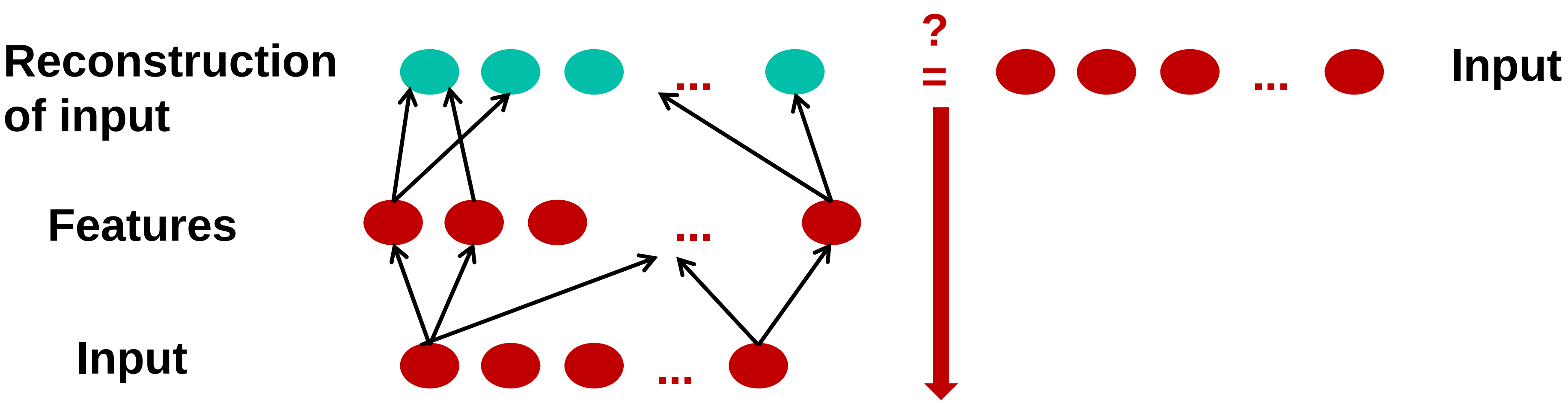
Features

Input



Layer-wise Unsupervised Pre-training

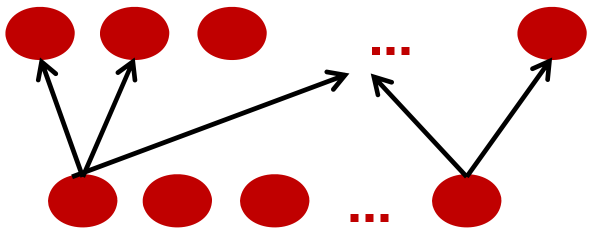
Auto-encoder:



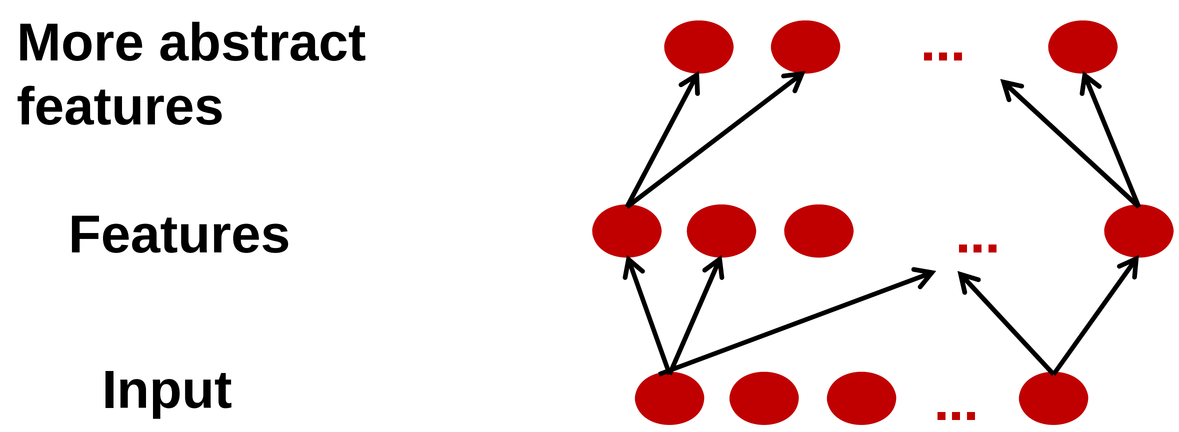
Layer-wise Unsupervised Pre-training

Features

Input

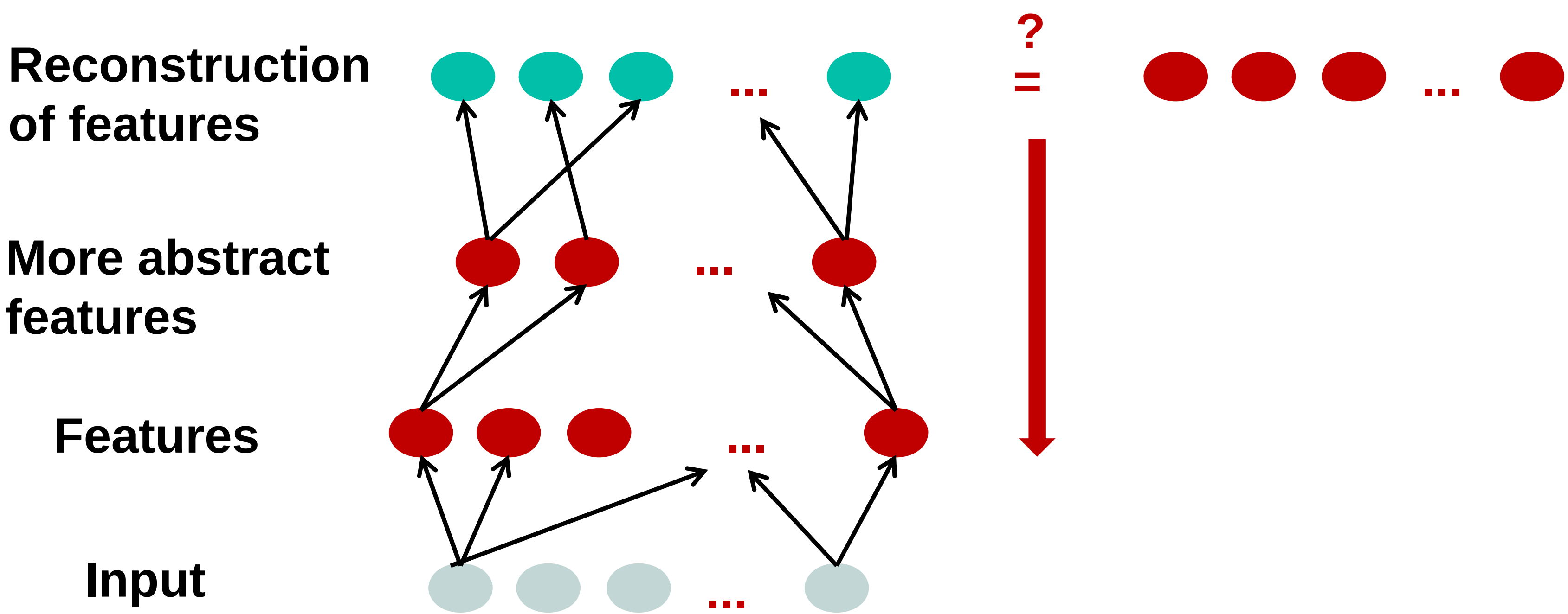


Layer-wise Unsupervised Pre-training

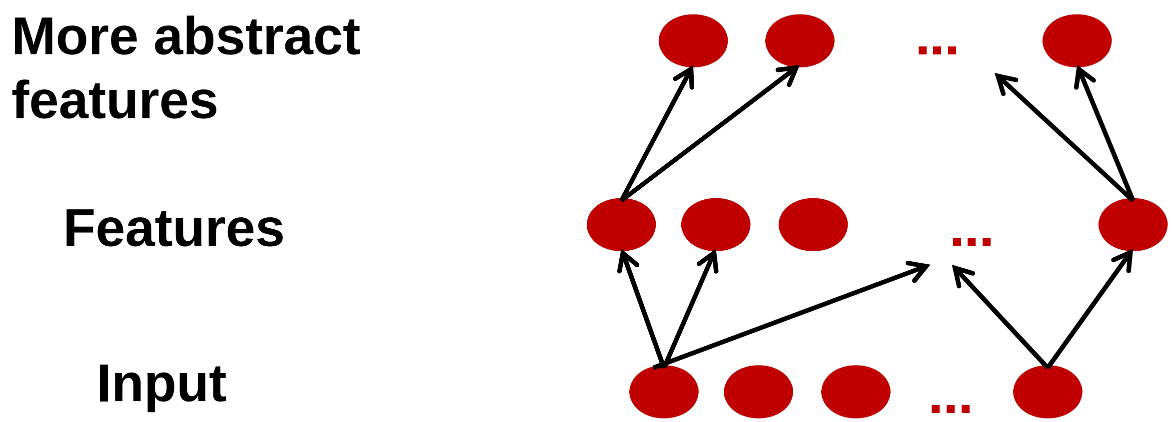


Layer-wise Unsupervised Pre-training

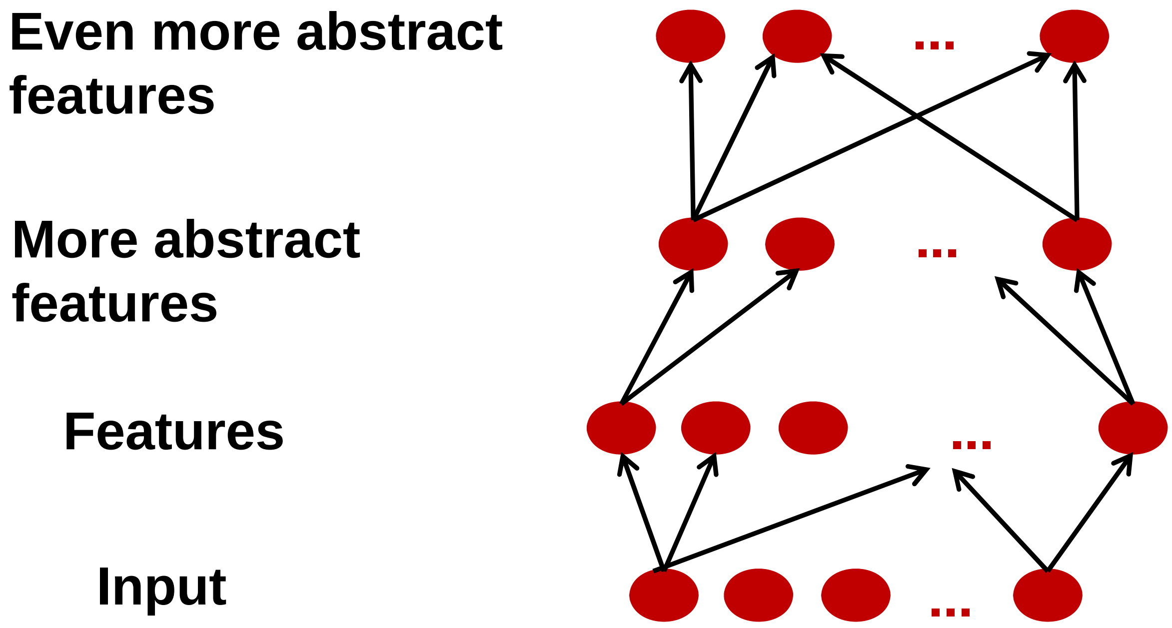
Auto-encoder:



Layer-wise Unsupervised Pre-training

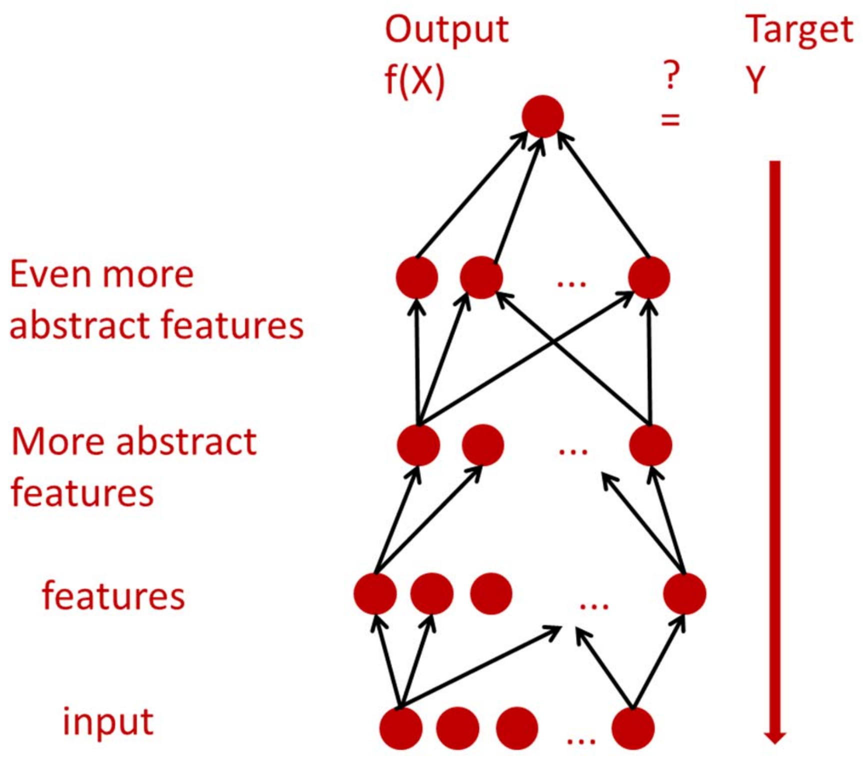


Layer-wise Unsupervised Pre-training



Supervised Fine-Tuning

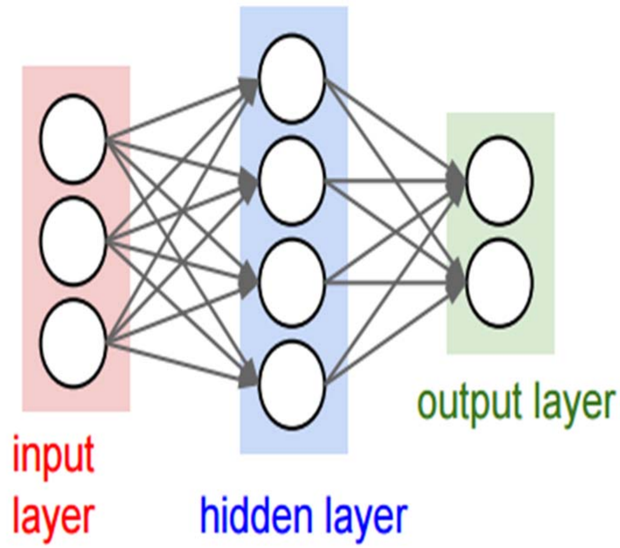
- Use the weights learned in unsupervised pretraining to initialize the network
- Then run BP in supervised setting



Convolutional Neural Network

- Some contents are borrowed from Rob Fergus, Yan Lecun and Stanford's course

Ordinary Neural Network



Now

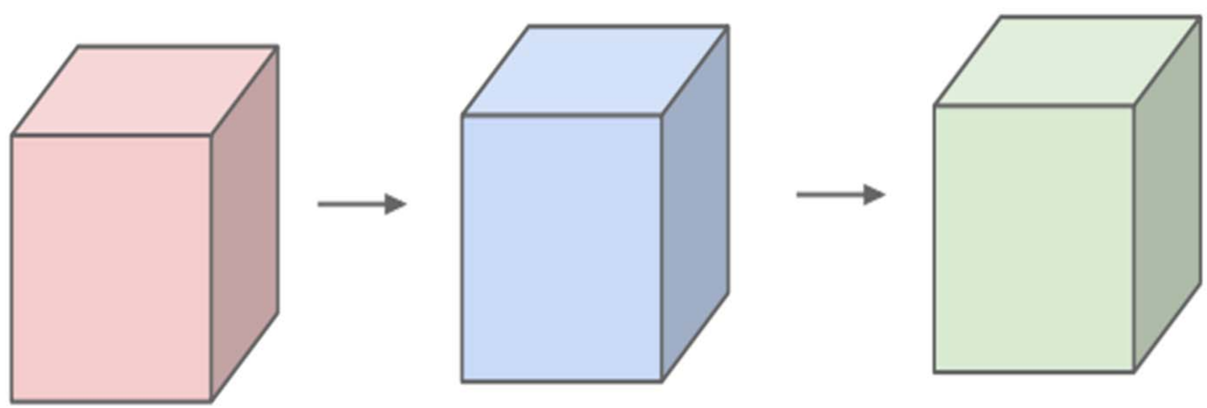
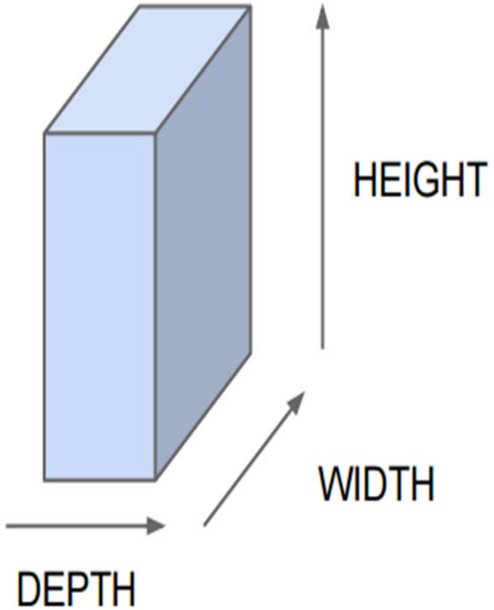


Figure courtesy, Fei-Fei, Andrej Karpathy

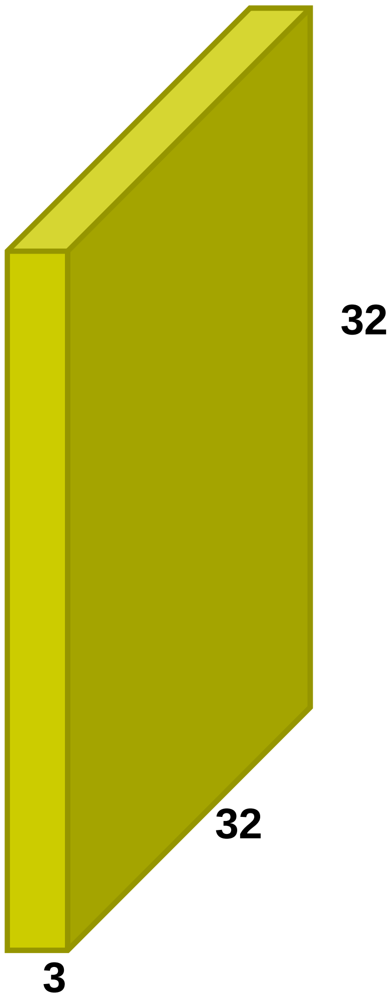
All Neural Net activations arranged in 3 dimensions

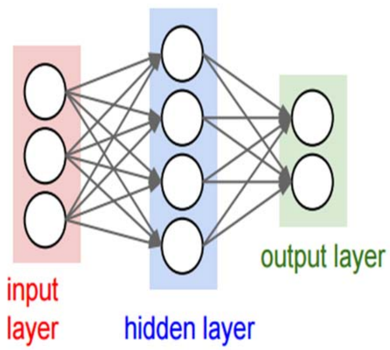


For example, a CIFAR-10 image is a 32*32*3 volume: 32 width, 32 height, 3 depth (RGB)

Figure courtesy, Fei-Fei, Andrej Karpathy

Local connectivity





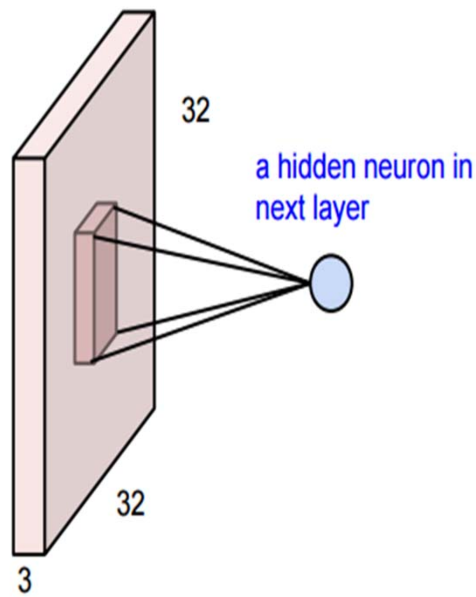


image: 32 * 32 * 3 volume

before: full connectivity: 32 * 32 * 3 weights for each neuron

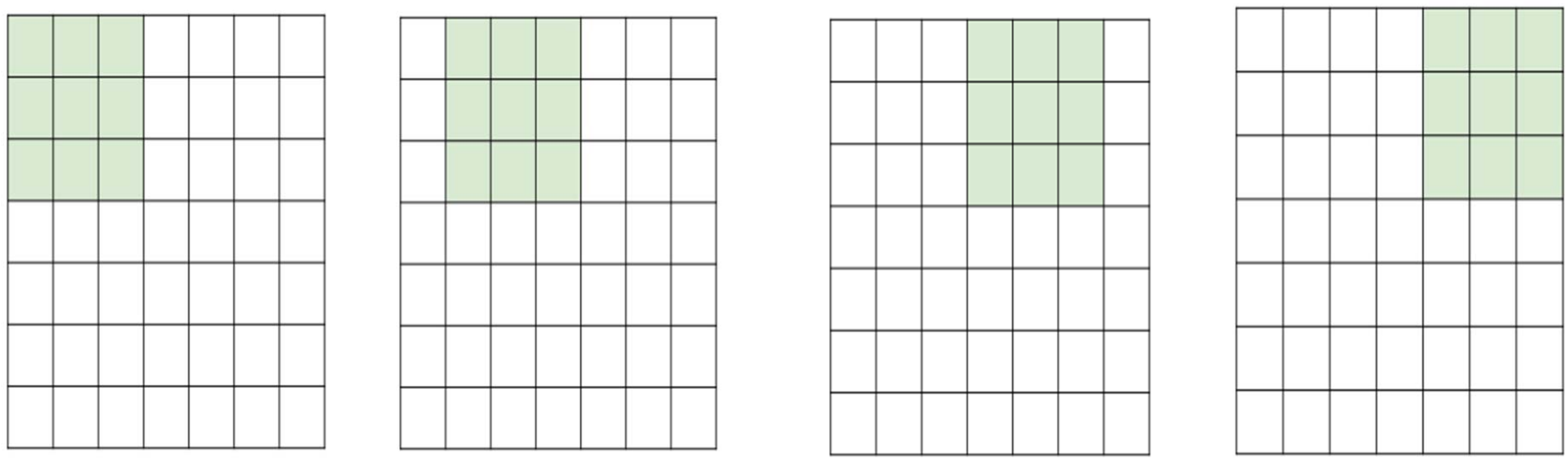
now: one unit will connect to, e.g. 5*5*3 chunk and only have 5*5*3 weights

Note the connectivity is:

- **local in space**
- **full in depth**

Convolution

- One local region only gives one output
- Convolution: Replicate the column of hidden units across space, with some stride

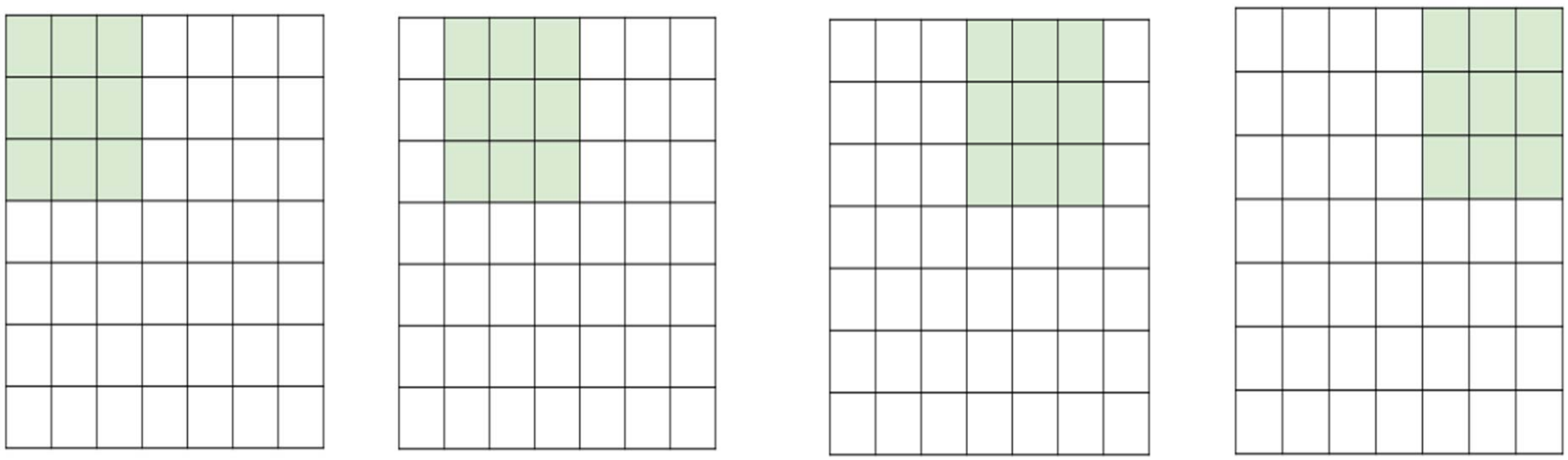


- 7 * 7 Input
- Assume 3*3 connectivity, stride = 1

- Produce a map
- What’s the size of the map? 5 * 5

Convolution

- One local region only gives one output
- Convolution: Replicate the column of hidden units across space, with some stride

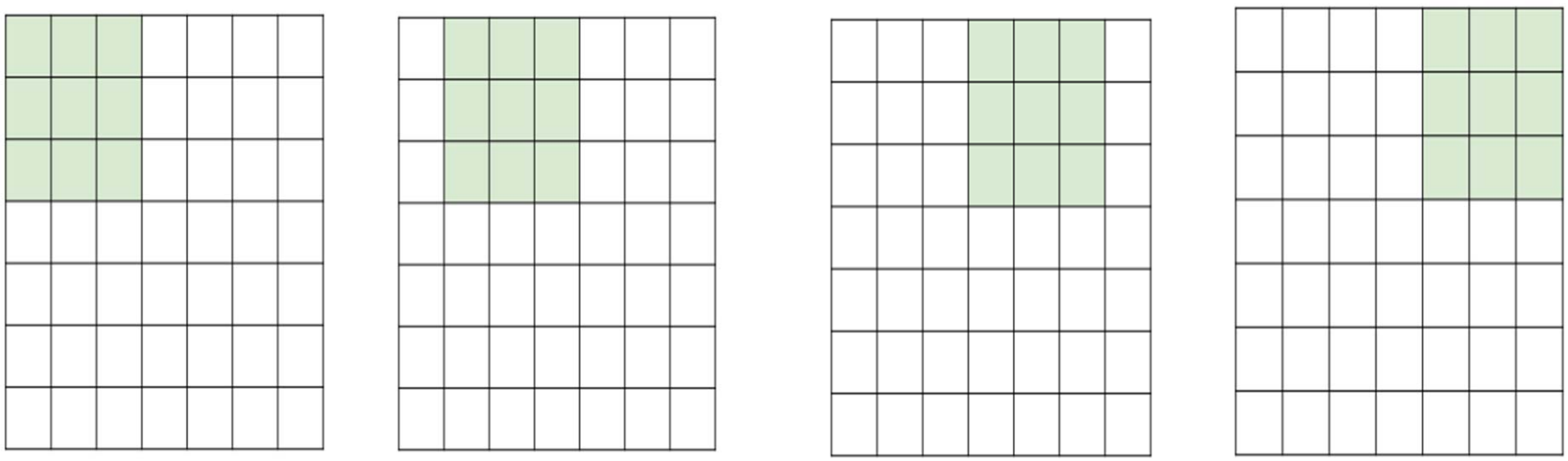


- 7 * 7 Input
- Assume 3*3 connectivity, stride = 1

- What if stride = 2?

Convolution

- One local region only gives one output
- Convolution: Replicate the column of hidden units across space, with some stride



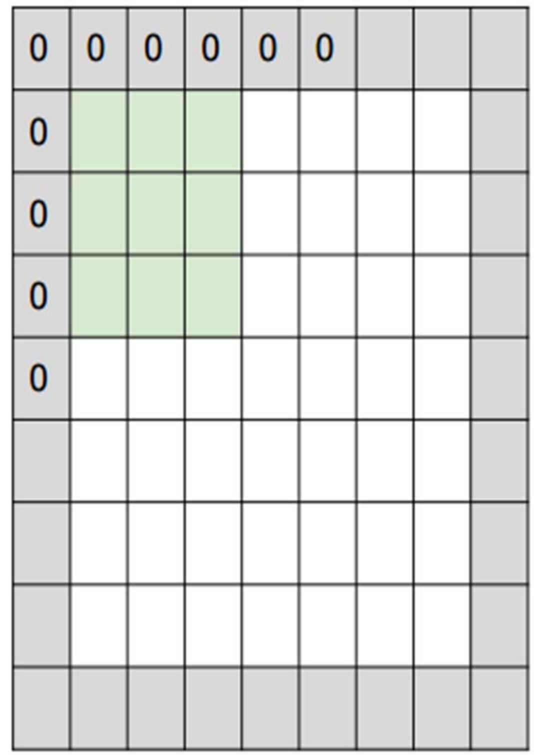
- 7 * 7 Input
- Assume 3*3 connectivity, stride = 1

- What if stride = 3?

Convolution: In Practice

- Zero Padding
    - Input size: 7 * 7
    - Filter Size: 3*3, stride 1
    - Pad with 1 pixel border

    - Output size?
        - 7 * 7 => preserved size!



Slide courtesy, Fei-Fei, Andrej Karpathy

Convolution: Summary

- Zero Padding
  - Input volume of size [W1 * H1 * D1]
  - Using K units with receptive fields F x F and applying them at strides of S gives

Output volume: [W2, H2, D2]

- $W2 = (W1 – F)/S + 1$
- $H2 = (H1 - F) / S + 1$
- $D2 = k$

Slide courtesy, Fei-Fei, Andrej Karpathy

Convolution: Problem

- Assume input [32 * 32 * 3]
- 30 units with receptive field 5 * 5, applied at stride 1/pad 1

  => Output volume: [30 * 30 * 30]

At each position of the output volume, we need 5 * 5 * 3 weights

=> Number of weights in such layer: 27000 * 75 = 2 million ☹

Idea:
Weight sharing!

Learn one unit, let the unit convolve across all local receptive fields!

Convolution: Problem

- Assume input [32 * 32 * 3]
- 30 units with receptive field 5 * 5, applied at stride 1/pad 1

  => Output volume: [30 * 30 * 30] = 27000 units

Weight sharing

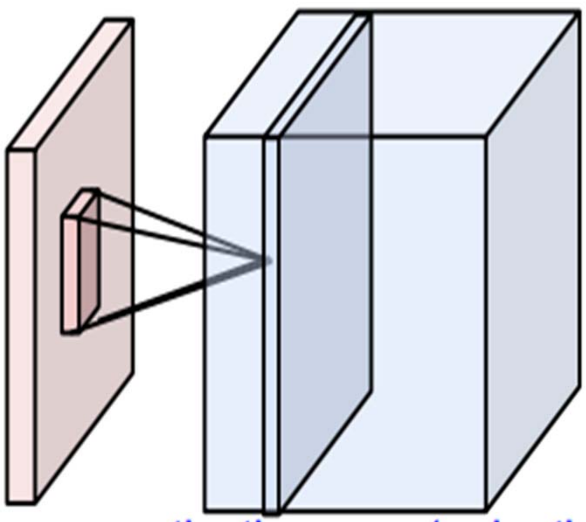
=> Before: Number of weights in such layer: 27000 * 75 = 2 million ☹

=> After: weight sharing: 30 * 75 = 2250 ☺

But also note that sometimes it's not a good idea to do weight sharing! When?

Convolutional Layers

- Connect units only to local receptive fields
- Use the same unit weight parameters for units in each "depth slice" (i.e. across spatial positions)



one activation map (a depth slice), computed with one set of weights

Can call the units "**filters**"

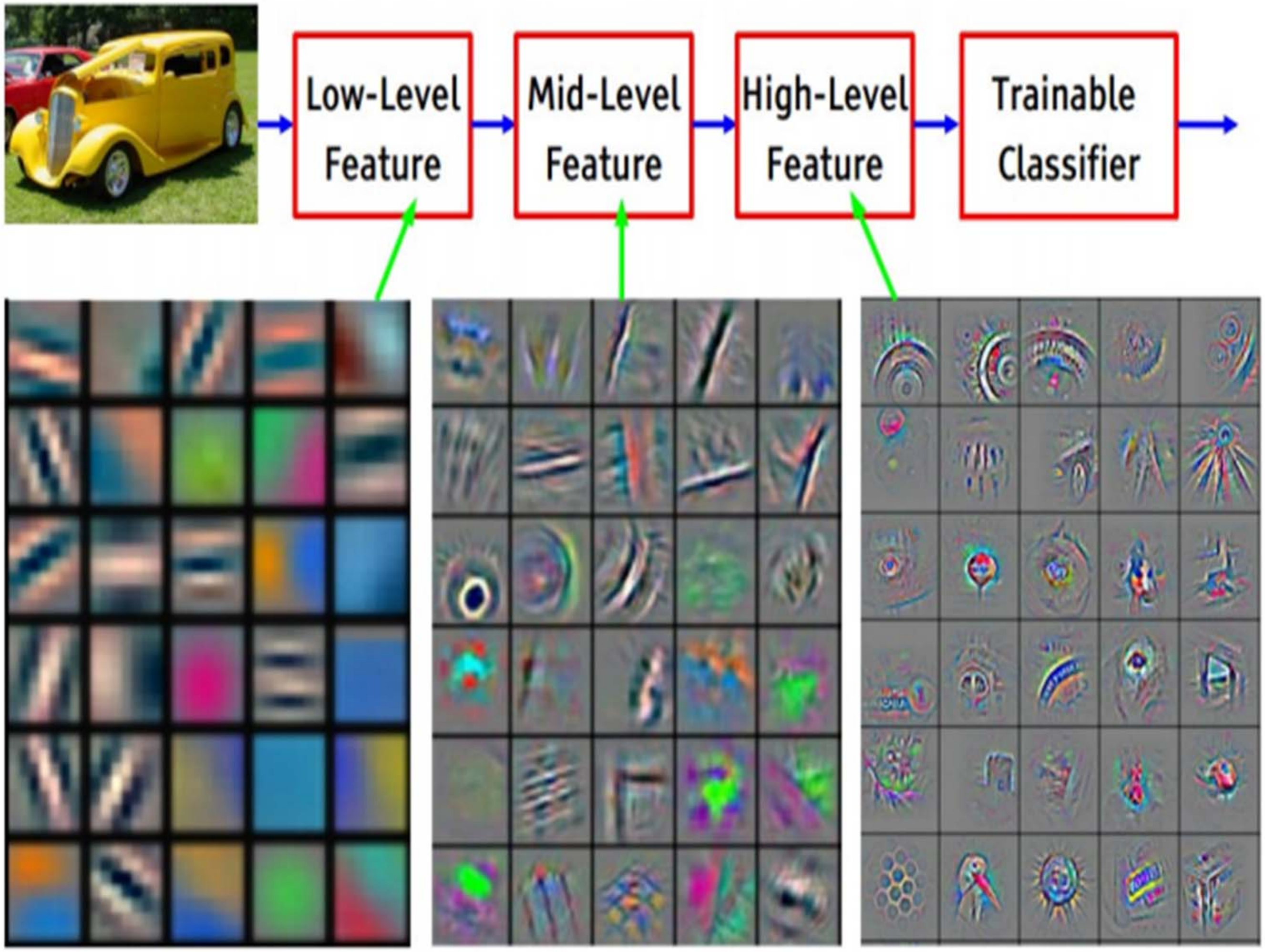
We call the layer convolutional because it is related to convolution of two signals

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

Sometimes we also add a bias term b, y = Wx + b, like what we have done for ordinary NN

Short question: Will convolution layers introduce nonlinearity?

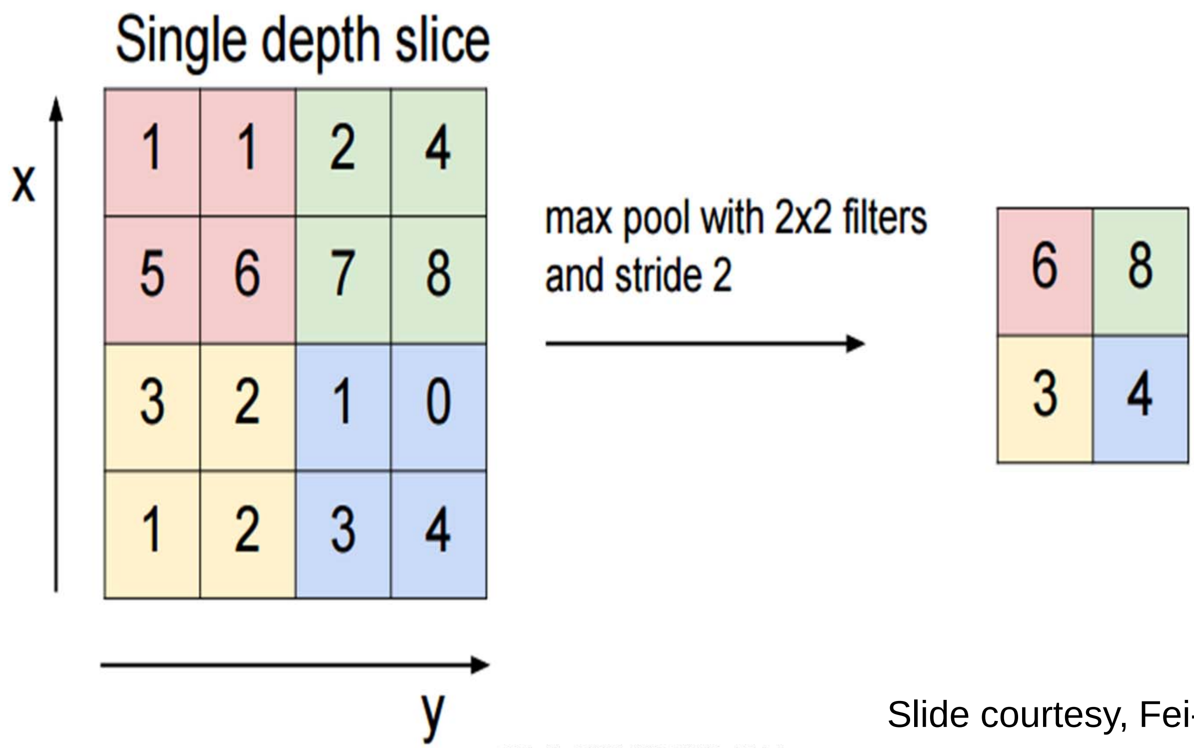
Stacking Convolutional Layers



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Pooling Layers

- In ConvNet architectures, Conv layers are often followed by Pool layers
  - makes the representations smaller and more manageable without losing too much information. Computes MAX operation (most common)



Slide courtesy, Fei-Fei, Andrej Karpathy

Pooling Layers

- In ConvNet architectures, Conv layers are often followed by Pool layers
  - makes the representations smaller and more manageable without losing too much information. Computes MAX operation (most common)
- Input volume of size [W1 x H1 x D1]
- Pooling unit receptive fields F x F and applying them at strides of S gives
- Output volume: [W2, H2, D1]: depth unchanged!

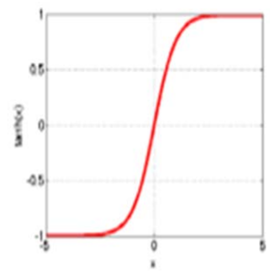
$$W2 = (W1-F)/S+1,$$

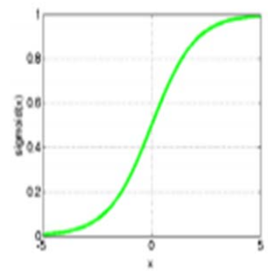
$$H2 = (H1-F)/S+1$$

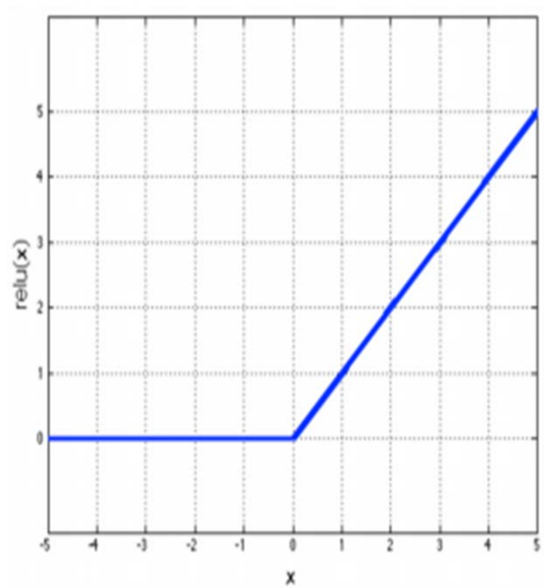
Short question: Will pooling layer introduce nonlinearity?

Nonlinerity

- Similar to NN, we need to introduce nonlinearity in CNN
  - Sigmoid
  - Tanh
  - RELU: Rectified Linear Units -
    - Simplifies backpropagation
    - Makes learning faster
    - Avoids saturation issues

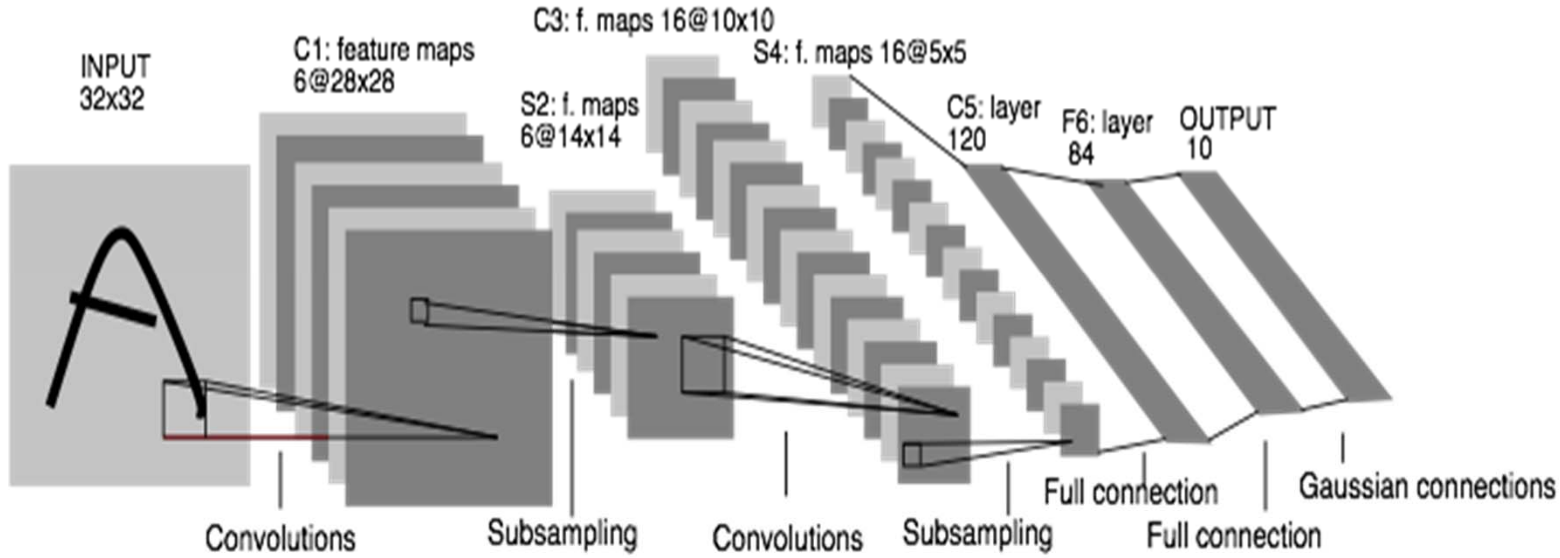






Slide courtesy, Yan Lecun

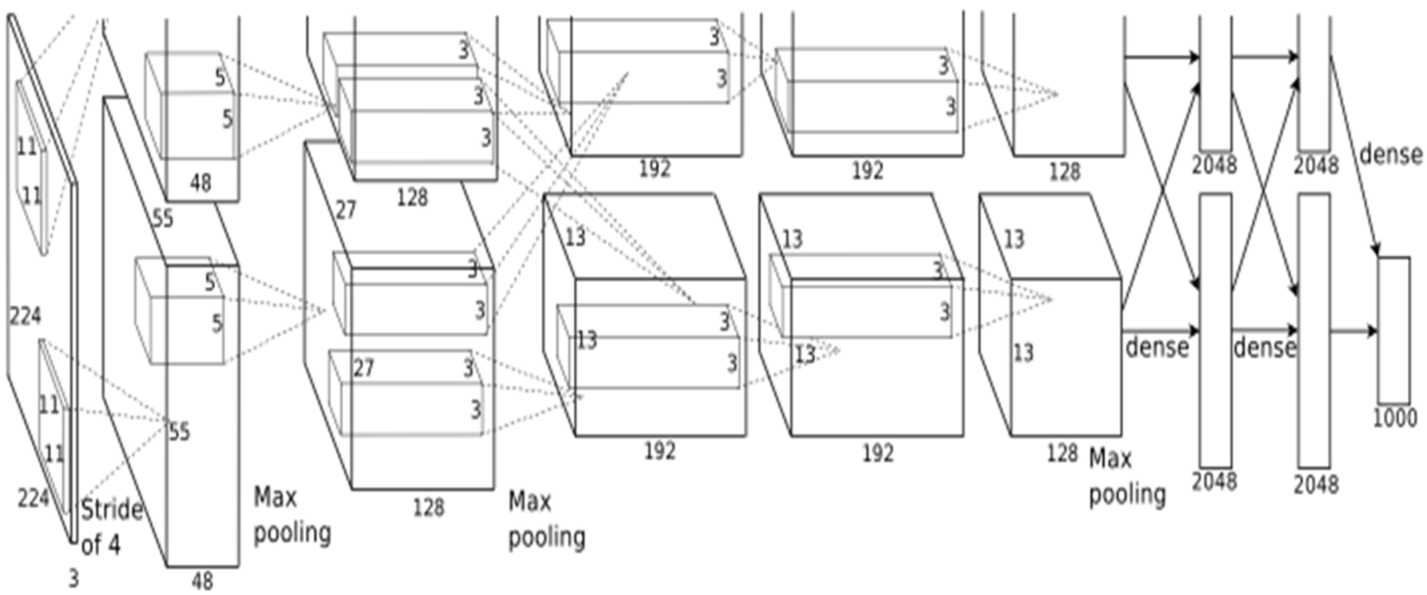
Convolutional Networks: 1989



LeNet: a layered model composed of convolution and subsampling operations followed by a holistic representation and ultimately a classifier for handwritten digits. [LeNet]

Slide courtesy, Yangqing Jia

Convolutional Nets: 2012

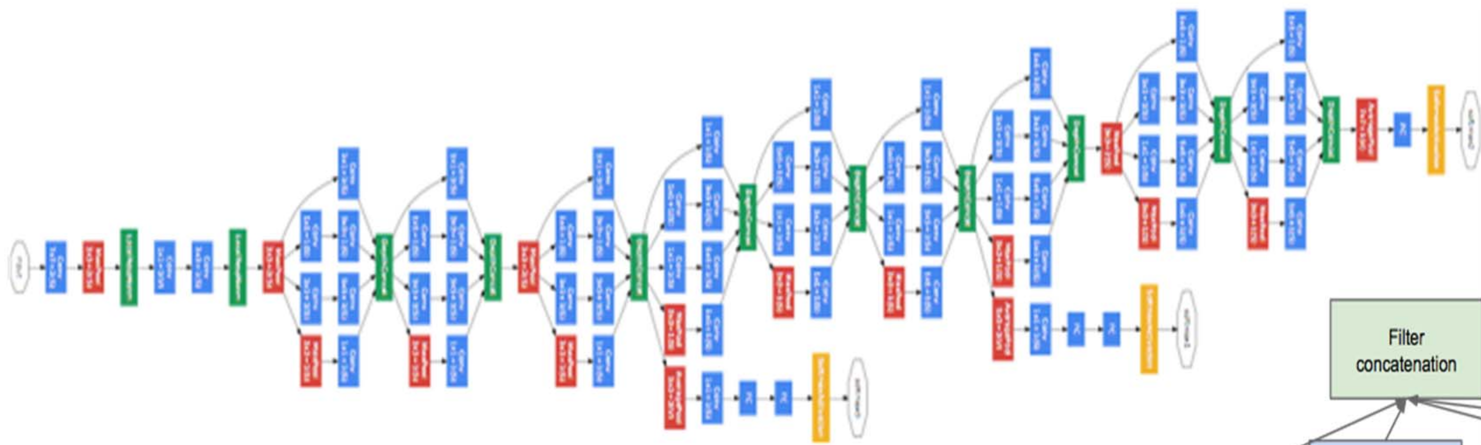


AlexNet: a layered model composed of convolution, subsampling, and further operations followed by a holistic representation and all-in-all a landmark classifier on
ILSVRC12. [AlexNet]

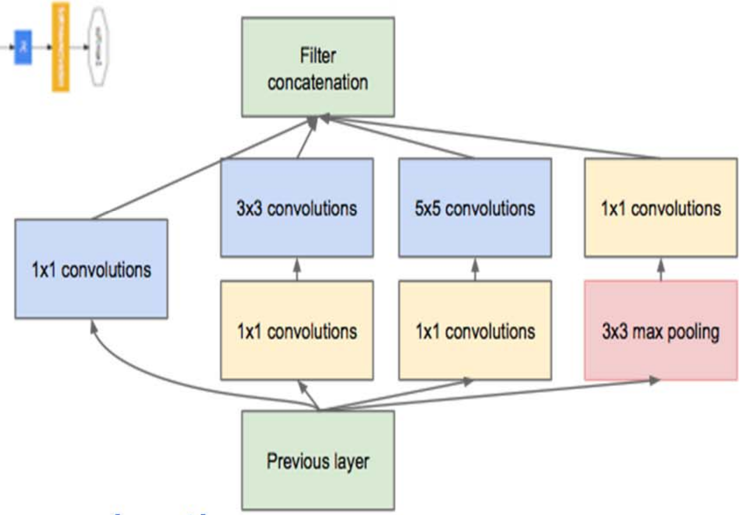
+ data
+ gpu
+ non-saturating nonlinearity
+ regularization

Slide courtesy, Yangqing Jia

Convolutional Nets: 2014



ILSVRC14 Winners: ~6.6% Top-5 error
- GoogLeNet: composition of multi-scale dimension-reduced modules (pictured)
- VGG: 16 layers of 3x3 convolution interleaved with max pooling + 3 fully-connected layers

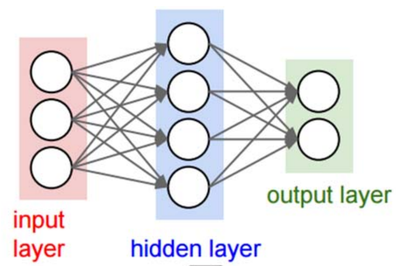


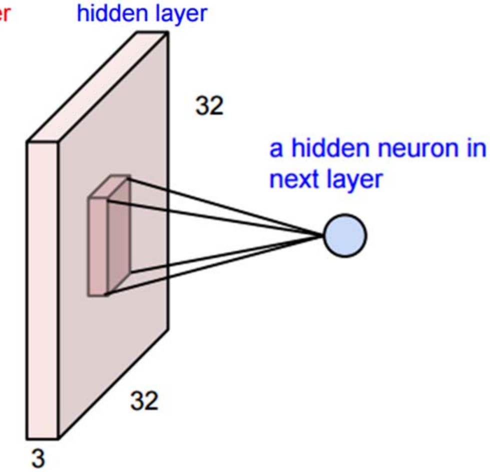
+ depth
+ data
+ dimensionality reduction

Slide courtesy, Yangqing Jia

Training CNN: Use GPU

- Convolutional layers
  - Reduce parameters BUT Increase computations
- FC layers
  - each neuron has more weights
  - but less computations





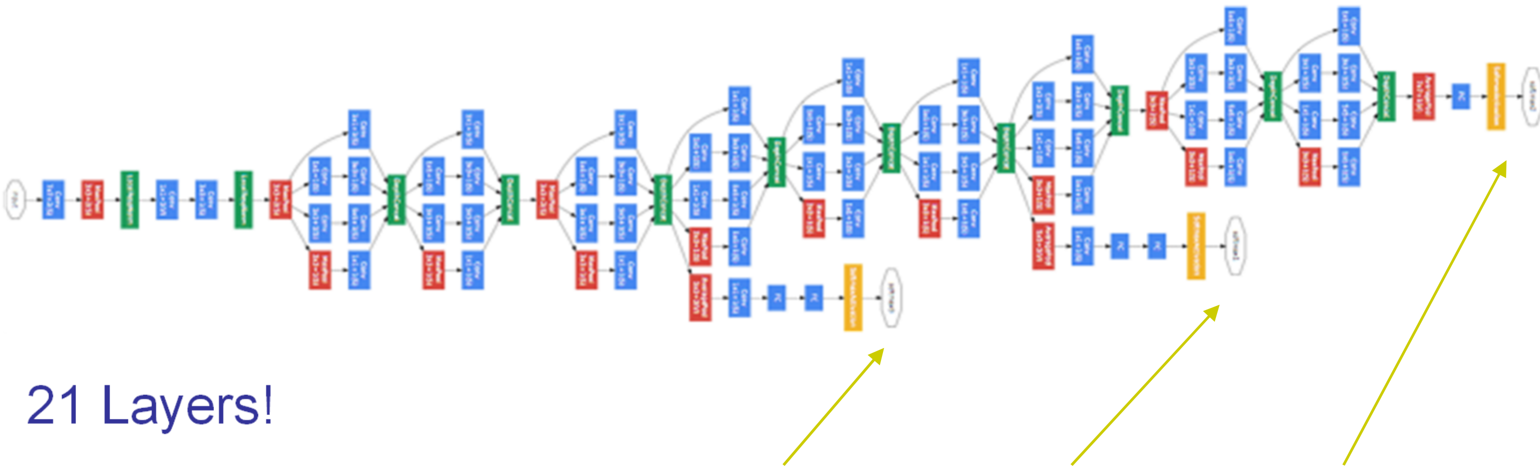
- Conv layers
  - each neuron has less weights
  - but more computations. Why?

    because

    it will

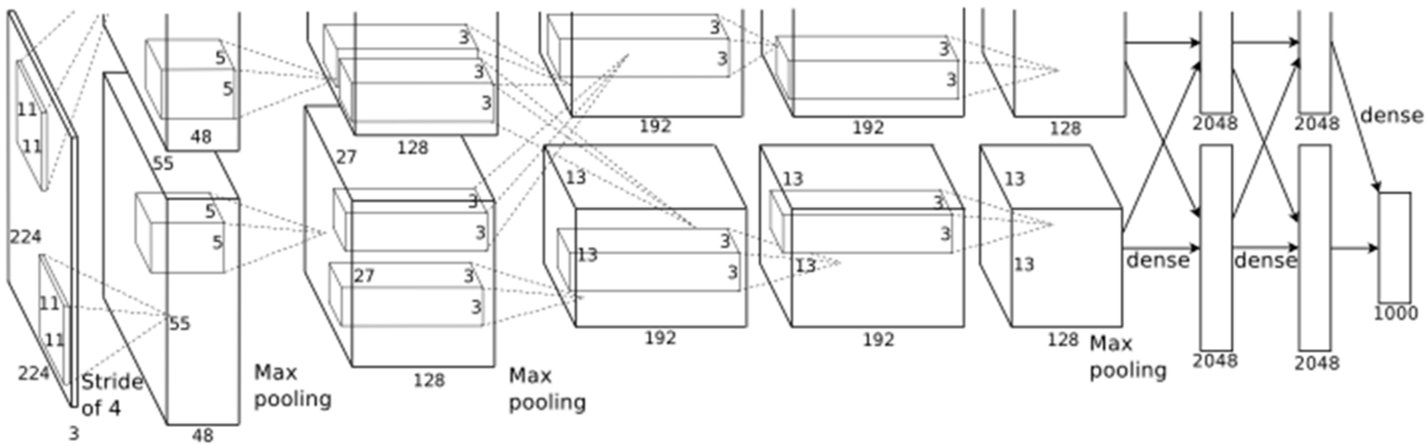
GPU is good at convolution!

Training CNN: depth cares!



- 21 Layers!
- Gradient vanishes when the network is too deep: Lazy to learn!
- Add intermediate loss layers to produce error signals!
- Do contrast normalization after each conv layer!
- Use ReLU to avoid saturation!

Training CNN: Huge model needs more data!



IMAGENET

- Only 7 layers, 60M parameters!
- Need more labeled data to train!
- Data augmentation: crop, translate, rotate, add noise!

Training CNN: highly nonconvex objective

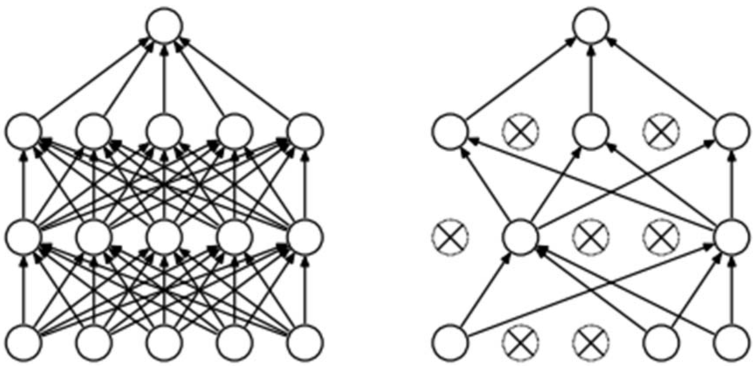
- Demand more advanced optimization techniques
  - Add momentum as we have done for NN
  - Learning rate policy
    - decrease learning rate regularly!
    - different layers use different learning rate!
    - observe the trend of objective curve more often!
  - Initialization really cares!
    - Supervised pretraining
    - Unsupervised pretraining

Training CNN: avoid overfitting

- More data are always the best way to avoid overfitting
  - data augmentation

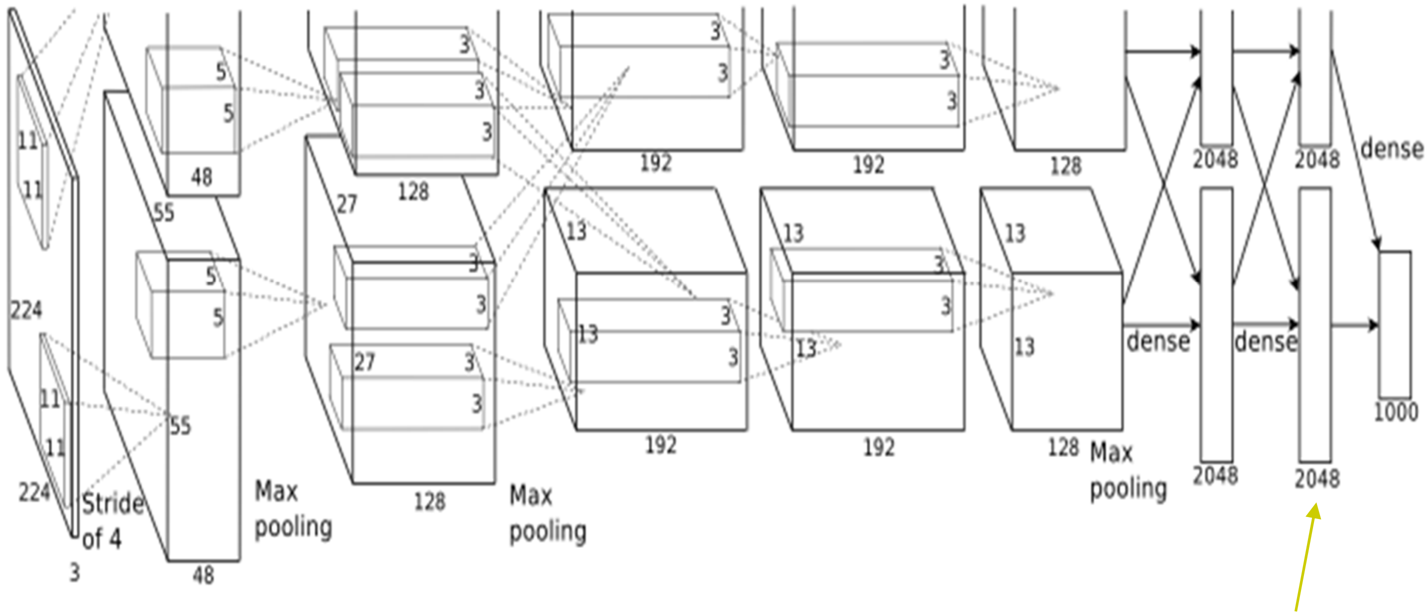
- Add regualizations: recall what we have done for linear regression

- Dropout



(a) Standard Neural Net (b) After applying dropout.

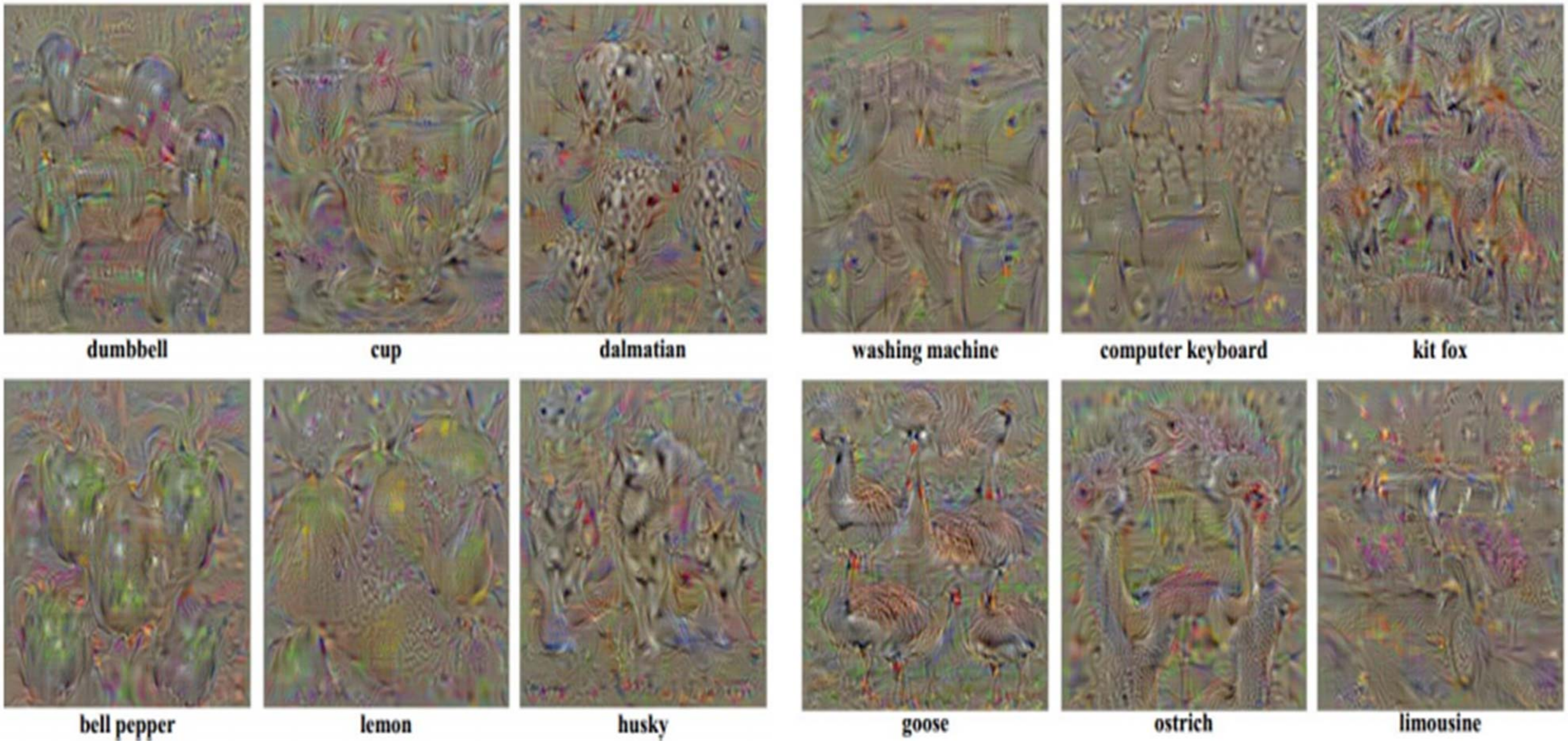
Visualize and Understand CNN



A CNN transforms the image to 4096 numbers that are then linearly classified.

Visualize and Understand CNN

- Find images that maximize some class score:



Yes, Google Inceptionism!

Visualize and Understand CNN

- More visualizations

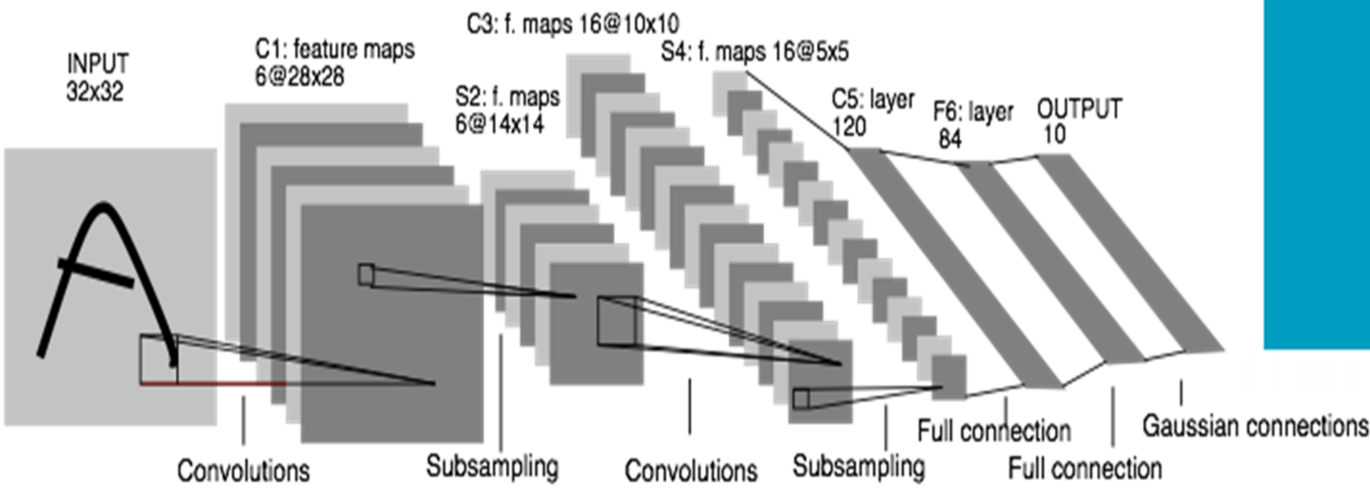
https://www.youtube.com/watch?v=AgkfIQ4I GaM&feature=youtu.be

Limitations

- Supervised Training
  - Need huge amount of labeled data, but label is scarce!
- Slow Training
  - Train an AlexNet on a single machine need one week!
- Optimization
  - Highly nonconvex objective
- Parameter tuning is hard
  - The parameter space is so large…

Summary

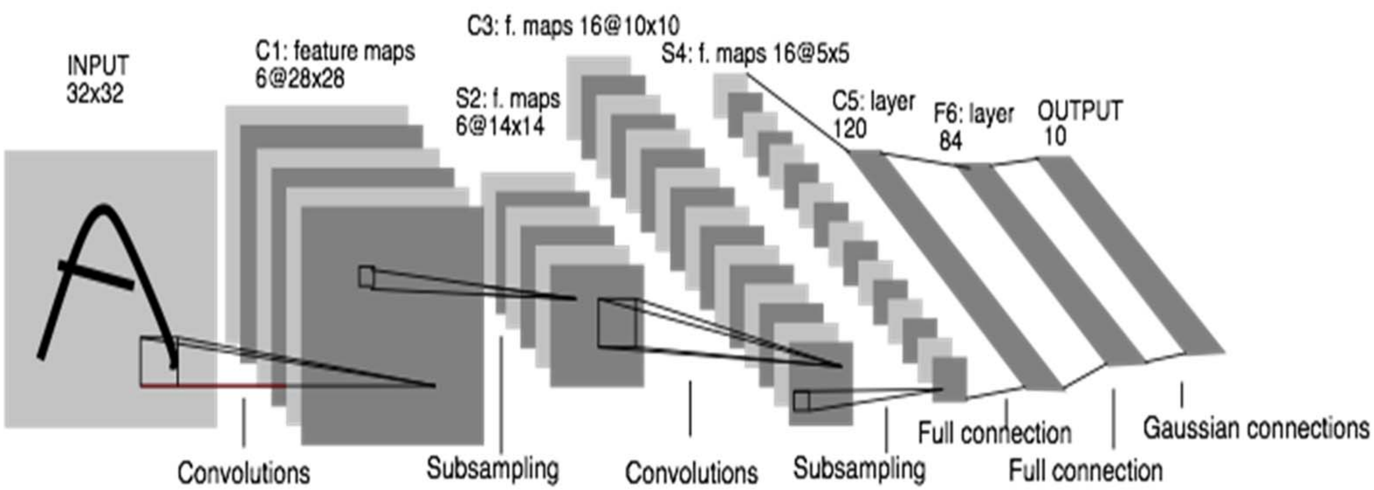
- Neural network with specialized connectivity structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant features
- Classification layer at the end

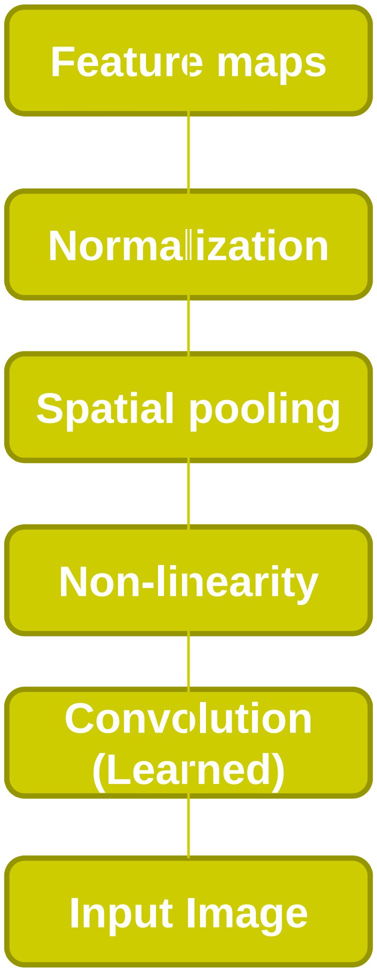


Slide courtesy, Rob Fergus

Summary

- Feed-forward
  - Convolve input
  - Non-linearity (rectified linear)
  - Pooling (local max, mean)
- Supervised
- Train convolutional filter s by back-propagation classification error at the end





Further reading

- Andrej Karpathy: The Unreasonable Effectiveness of Recurrent Neural Networks (http://karpathy.github.io/2015/05/21/rnn-effectiveness)
- Recurrent Neural Networks Tutorial (http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns)