Machine Learning

CSE 142

Xin (Eric) Wang

Wednesday, December 1, 2021

Today

- Neural networks and deep learning (cont.)

Notes

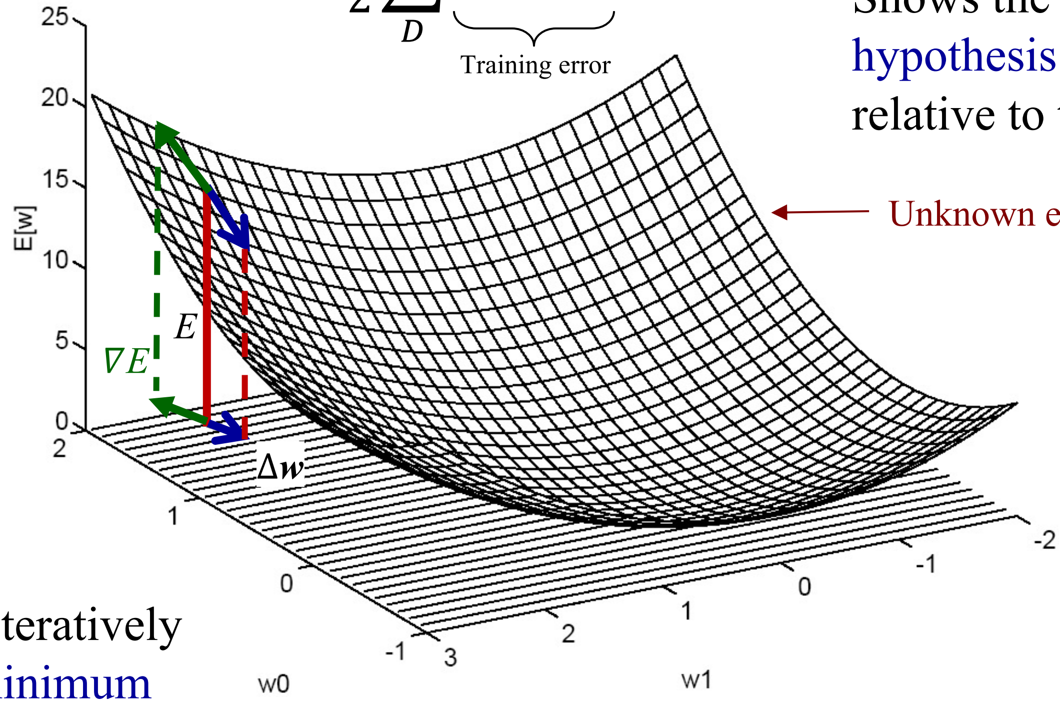
- HW3 grades out
- Final exam (Thursday, December 9, 4-6pm, here)
  - With camera on all the time. All the teaching staffs will be watching and check the roster from times to times.
  - No phones. No earphones. No talking. No internet search. No keyboard typing.
  - Write answers on a white paper using your pen.
  - You may have up to 5 additional minutes to take photos and upload them to Gradescope. Exams must be turned in by 6:05pm sharp.
  - In case of any Gradescope errors, email your answers to us before 6:05pm.
  - You may leave the Zoom meeting after turning in your solutions.
- Final exam practice and info sheet posted on Canvas
- Instructor OH today will be co-hosted by TA Jing Gu and Tutor Winson Chen

The hypothesis space and gradient descent

Review

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{D}\underbrace{(y_i - o_i)^2}_{\text{Training error}}$$

Training error $E(\boldsymbol{w})$



Shows the error E of the hypothesis $\boldsymbol{w} = (w_0, w_1)$ relative to the training data

Unknown error function $E(\boldsymbol{w})$

Hypothesis/weight space $\boldsymbol{w}$

Gradient descent iteratively searches for the minimum error over the complete training data by moving in the direction $(\delta w_0, \delta w_1)$, at each step, that most reduces the error.

Think globally, act locally!

So $\boldsymbol{w} = \boldsymbol{w} + \Delta\boldsymbol{w}$

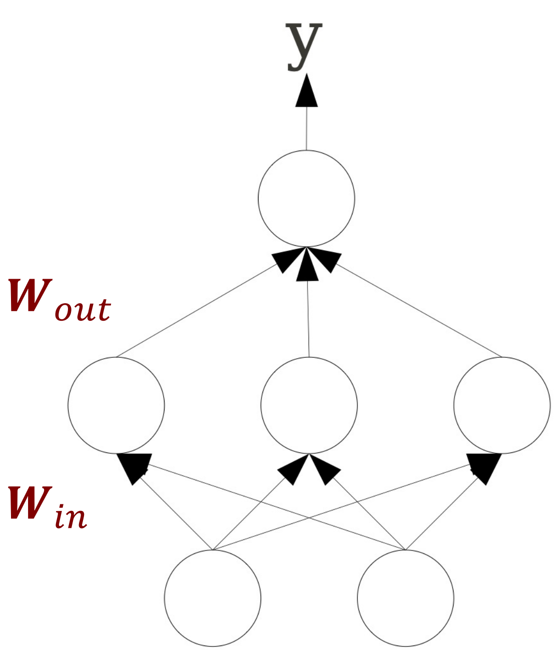
$$\Delta\boldsymbol{w} = -\eta\nabla E(\boldsymbol{w})$$

where $\nabla E(\boldsymbol{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n}\right)$ Gradient of E with respect to $\boldsymbol{w}$

Backpropagation

Review

- The backpropagation algorithm learns weights for a multilayer network
- Use gradient descent and chain rule to minimize the training loss



No activation functions:

$$\boldsymbol{y} = \boldsymbol{W}_{out}\boldsymbol{W}_{in}\boldsymbol{x}$$

sigmoid on the hidden layer and the output layer:

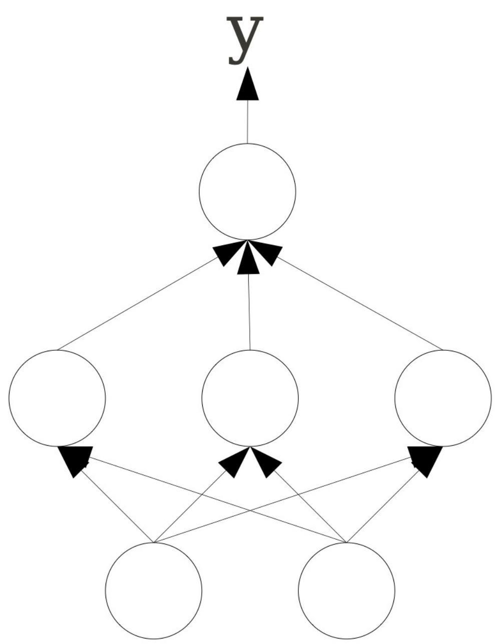
$$\boldsymbol{y} = S(\boldsymbol{W}_{out}\, S(\boldsymbol{W}_{in}\boldsymbol{x}))$$

Backprop trains the network by iteratively propagating errors backwards from output units

How to compute gradients?

- Use gradient descent to minimize the squared loss between the target values and the network output values:

$$E = \frac{1}{2}\sum_{p}(d^p - y^p)^2$$



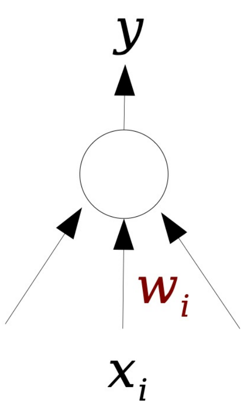
$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

Gradient descent on one single layer

$$y = \sum_i w_i x_i$$

$$E = \frac{1}{2}\sum_p (d^p - y^p)^2$$

$$\frac{dE}{dy} = y - d$$

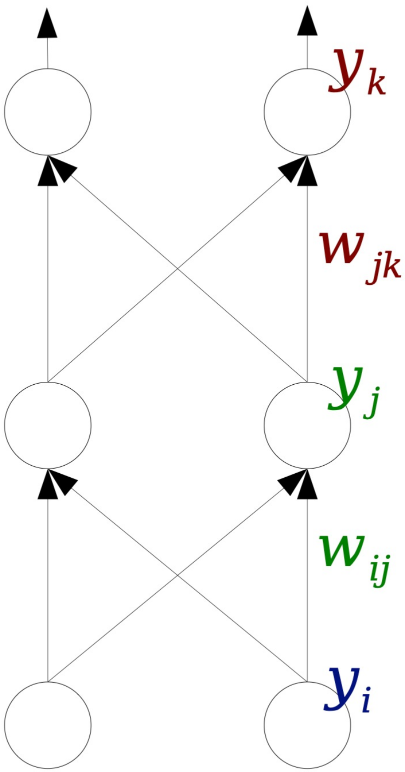


$$\frac{\partial E}{\partial w_i} = \frac{dE}{dy}\cdot\frac{\partial y}{\partial w_i} = (y-d)x_i$$

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i} = -\eta(y-d)x_i$$

How do we extend this to two (or even more) layers?

Chain rule to propagate errors backward



$$\frac{\partial E}{\partial y_k} = y_k - d_k$$

$$\delta_k = \frac{\partial E}{\partial y_k} = y_k - d_k$$

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial w_{jk}} = \delta_k \cdot y_j$$

$$\frac{\partial E}{\partial y_j} = \sum_k \left(\frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial y_j} \right)$$

Error term

Local gradient

$$\delta_j = \frac{\partial E}{\partial y_j}$$

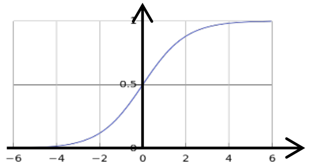
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial w_{ij}} = \delta_j \cdot y_i$$

$$\Delta w_{jk} = -\eta \cdot \frac{\partial E}{\partial w_{jk}} \qquad \Delta w_{ij} = -\eta \cdot \frac{\partial E}{\partial w_{ij}}$$

The Backpropagation algorithm

- Initialize all network weights $\boldsymbol{W}$ to small random numbers
- Until termination condition is met, do
  - For each training example, do
    - Propagate the input forward through the network
      - Input the training instance $\boldsymbol{x}$ and compute the outputs $\boldsymbol{y}$
        - » Using $\boldsymbol{x}$, $\boldsymbol{W}$, and sigmoid functions S
    - Propagate the errors backward through the network
      - Using chain rule
    - Update each network weight w_{ij}
      - $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$

The Backpropagation algorithm



- Implements a gradient descent search through the hypothesis space
  - Acts linearly early on, when the weights are small, since the sigmoid function for is approximately linear for small inputs
  - When the weights grow, the network starts to learn nonlinearly
- Errors propagate backwards
- Same process is followed for multiple layers
- Training can be quite slow
- Converges to local minimum (if given enough time)
- Methods to avoid local minimum trap problem include:
  - Run multiple times with different initial weights
  - Use a weight momentum term in the weight update rule
  - Stochastic gradient descent
  - Etc….

Backpropagation: comments

- What is the hypothesis space of Backpropagation?
  - The N-dimensional space of the N network weights
  - High-dimensional!
- Since the hypothesis space is continuous and the error function *E* is differentiable with respect to the weights, it makes an efficient gradient descent approach possible

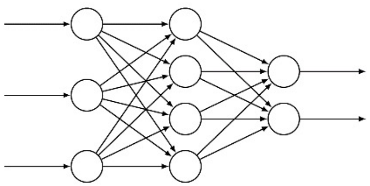
- What is the inductive bias of Backpropagation?
  (The way generalization is enforced beyond the data points)
  - It's hard to state precisely, but roughly it's *smooth interpolation between data points* or *convexity*
  - I.e., if two positive training example have no negative example between them, backprop tends to label the points in between as positive as well

Backpropagation: comments (cont.)

- There are multiple choices for the termination condition for updating weights
  - A fixed number of iterations (but how many?)
  - Until the error E falls below some predetermined threshold
- Backpropagation is susceptible to overfitting the training data, thus decreasing generalization to unseen examples
- Validation data comes in very useful here
- Typical strategy: Use the training data to train the network, but after every epoch (or a fixed number of iterations) measure error on the validation set
  - Choose the model (weights) that give the smallest error on the validation set

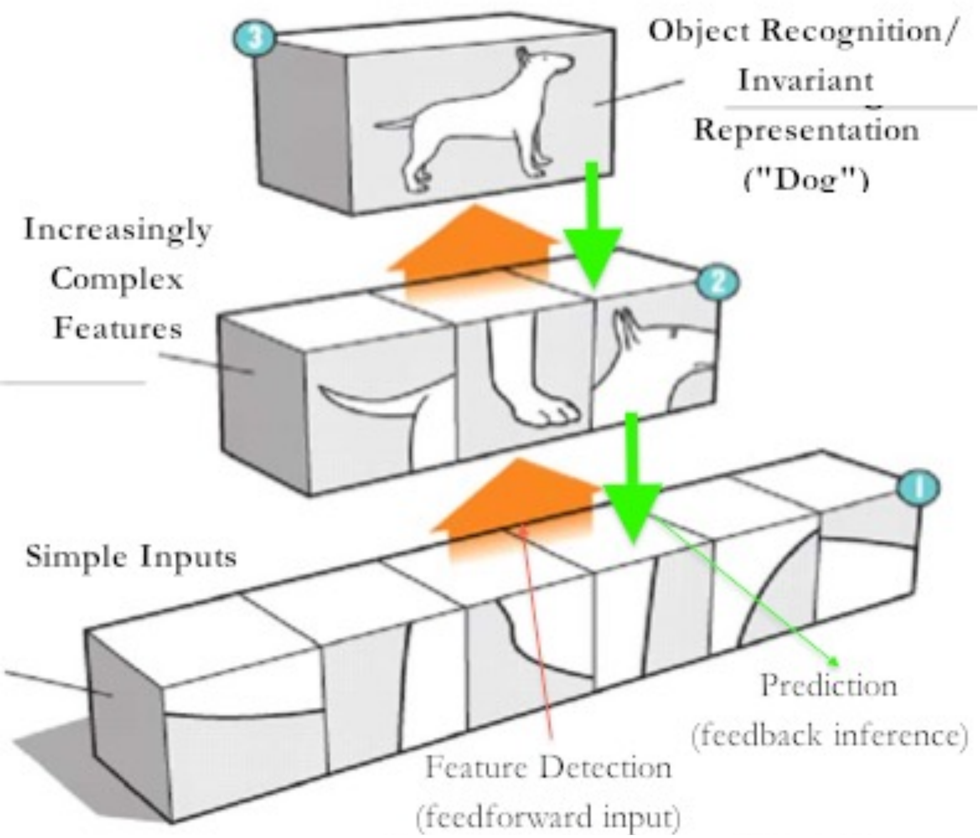
Deep learning

- Deep learning is about learning deep (many-layered) neural networks – multiple non-linear transformations from input to output
- Biological motivation: The human brain is a deep neural network, with many layers of neurons that act as feature detectors, detecting more and more abstract (high-level) features in deeper levels
- E.g., to classify or detect a cat in an image:
  - Bottom layers: Edge detectors, curves, corners straight lines
  - Middle layers: Fur patterns, eyes, ears
  - Higher layers: Body, head, legs
  - Top layer: Cat

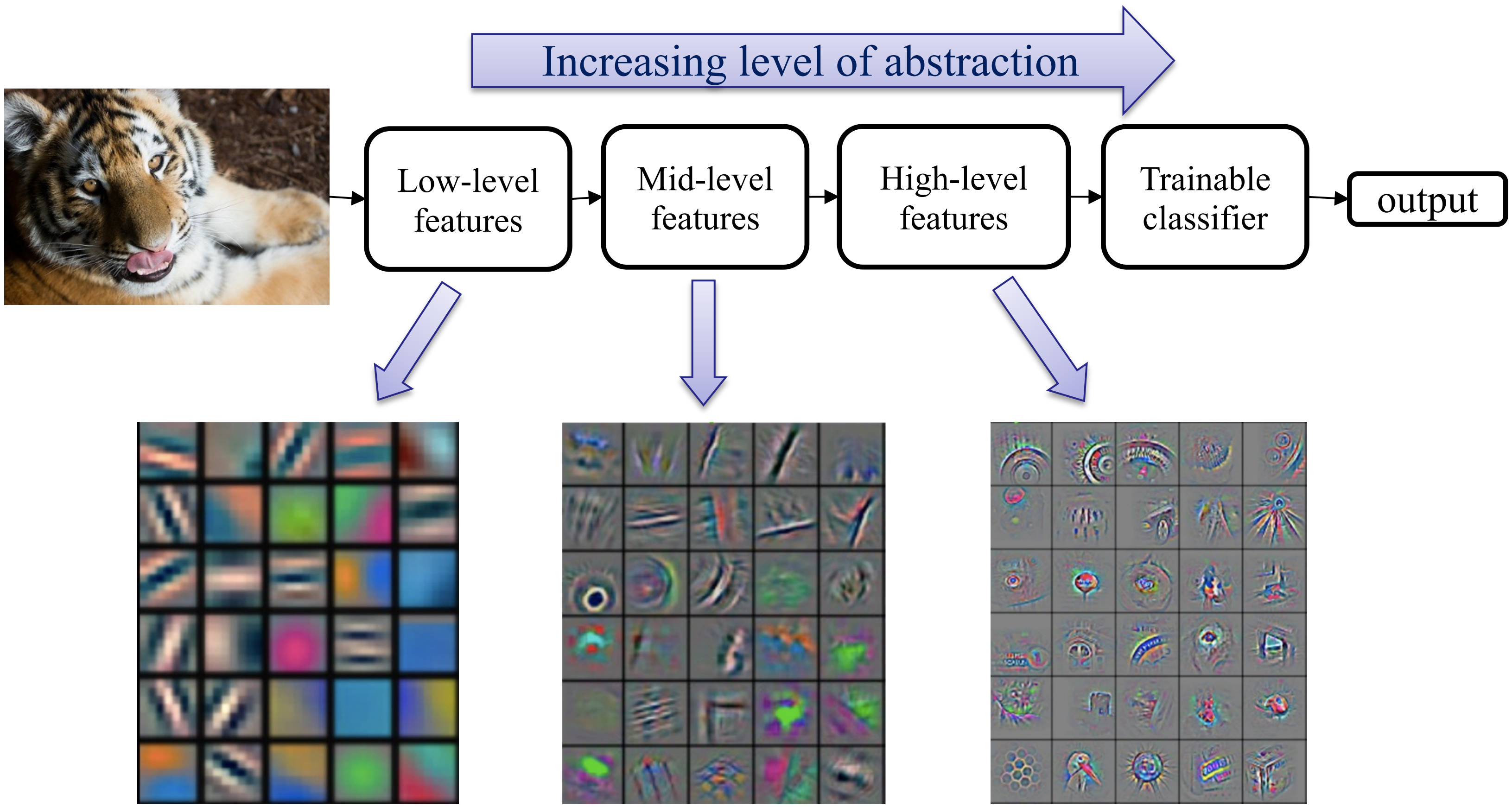


Deep learning

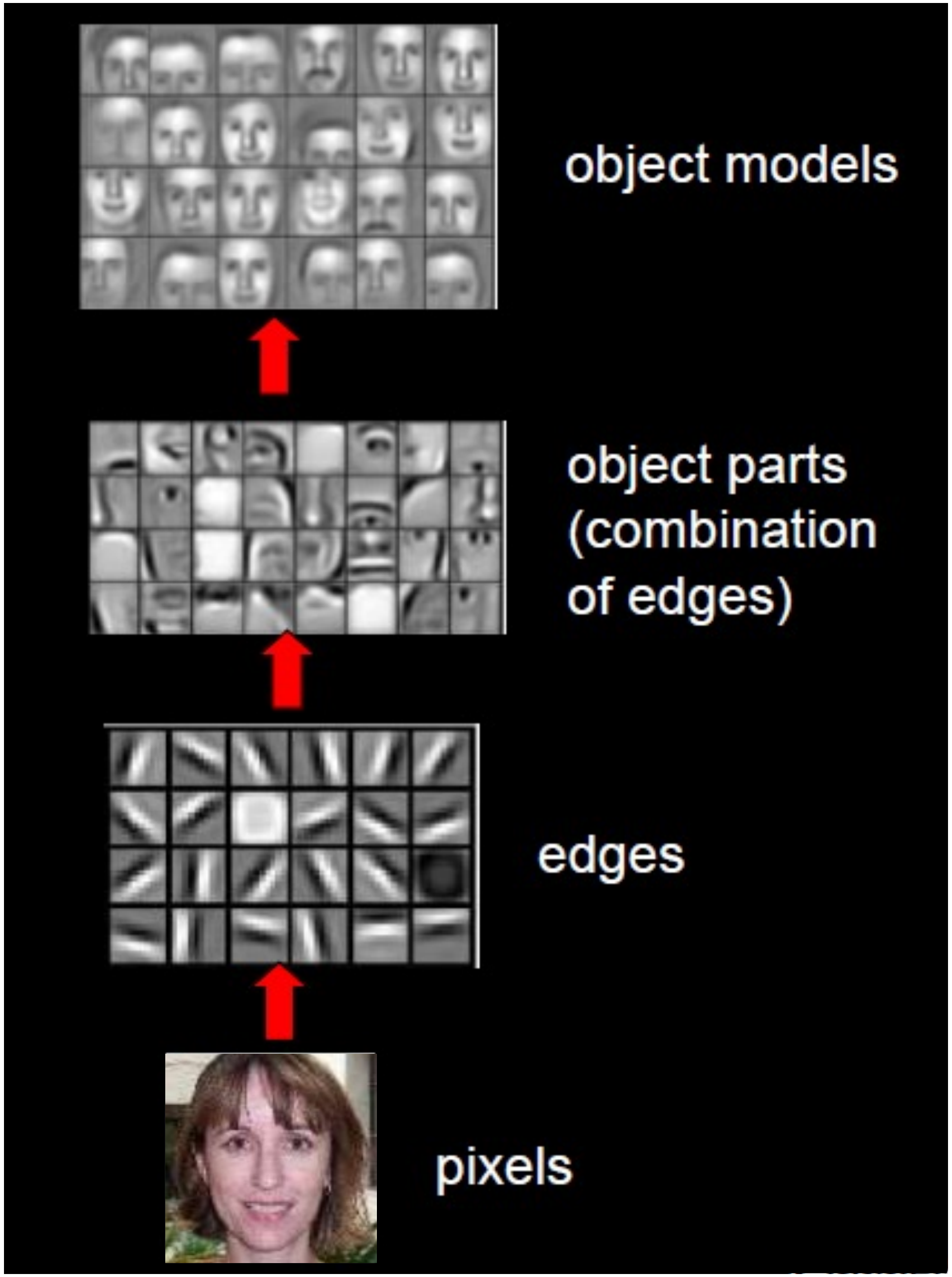
- Each level learns new (more complex or abstract) features from combinations of features from the level below



Deep learning

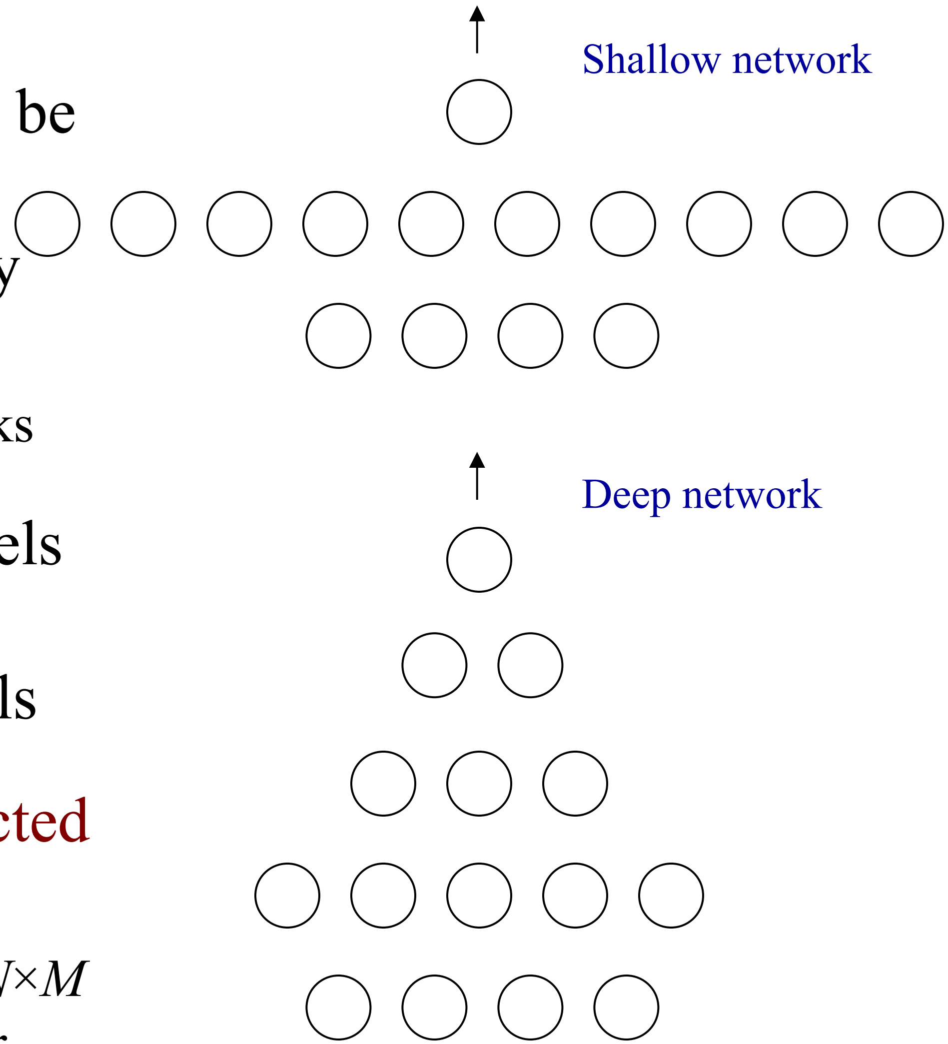


Feature visualization of convolutional net trained on ImageNet (Zeiler and Fergus, 2013)


object models
object parts
(combination
of edges)
edges
pixels

Deep network structure

- Although 2- and 3-layer networks have been shown to be able to approximate any continuous function, they may require exponential size
  - I.e., very wide, shallow networks
- In a deep network, higher levels can express combinations of features learned at lower levels
- Networks can be fully-connected or partially-connected
  - N nodes to M nodes = at most $N \times M$ connections (weights) in a layer



Convolutional neural networks (CNNs)