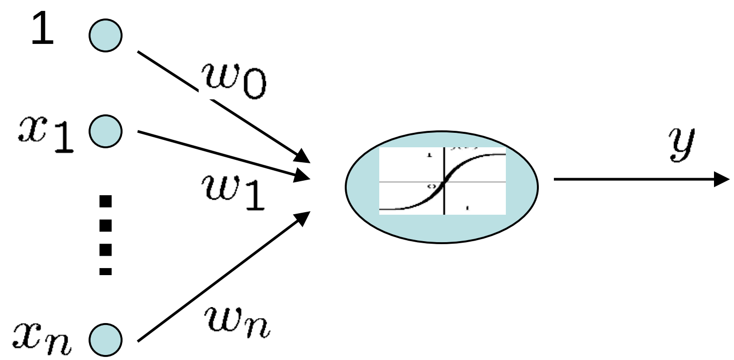
Neural Networks

Neural Network Neurons

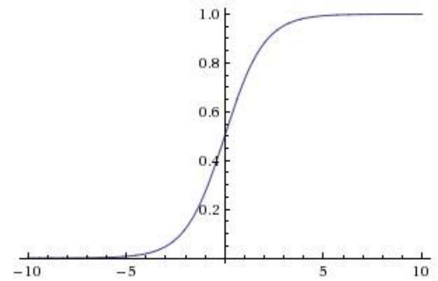


- Biologically inspired
- Receives n inputs (plus a bias term)
- Multiplies each input by its weight
- Applies activation function to the sum of results
- Outputs result

Commonly Used Activation Functions

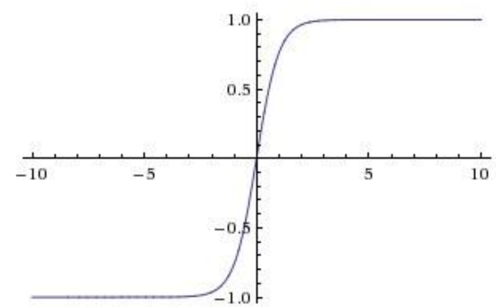
- **Sigmoid function:**

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



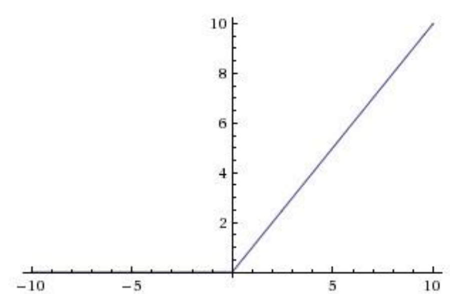
- **Tanh function:**

$$\tanh(x) = 2\sigma(2x) - 1$$

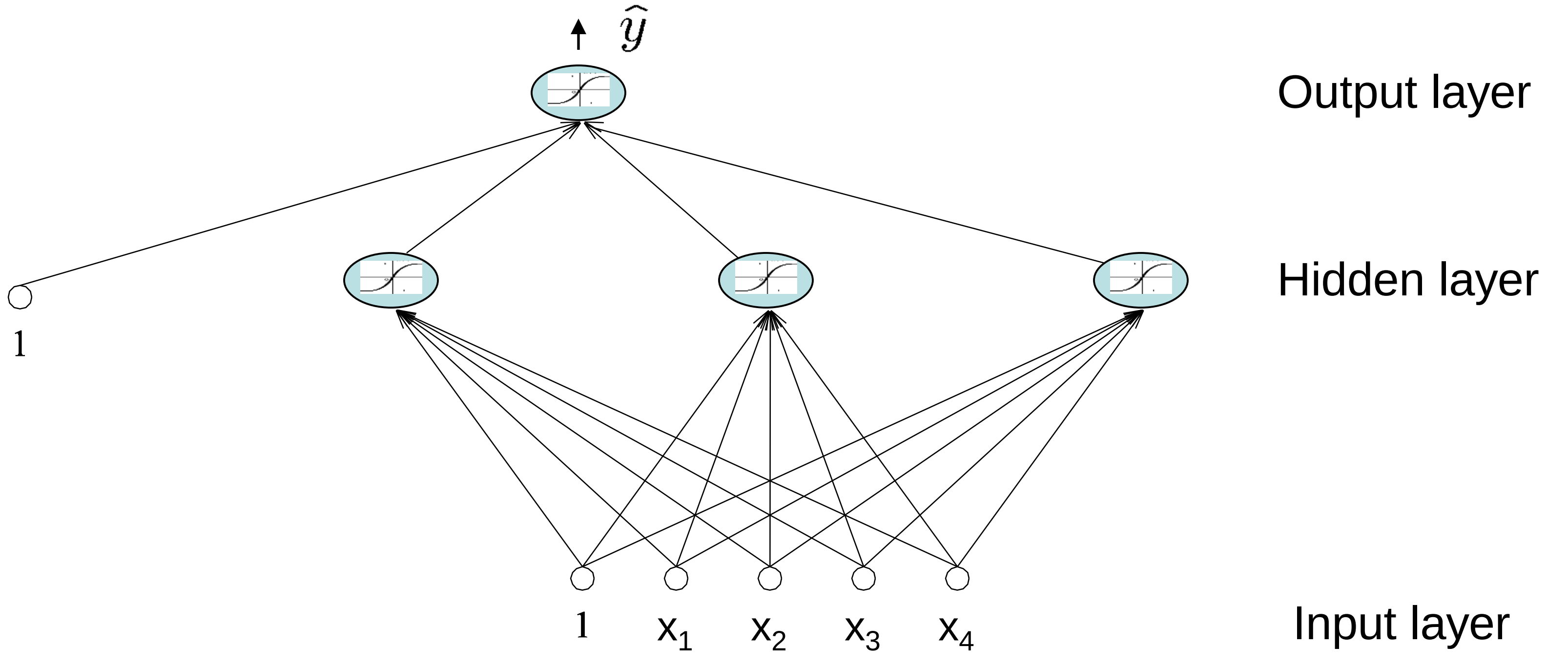


- **Rectified Linear Unit (ReLu)**:

$$f(x) = \max(0, x)$$



Basic Multilayer Neural Network

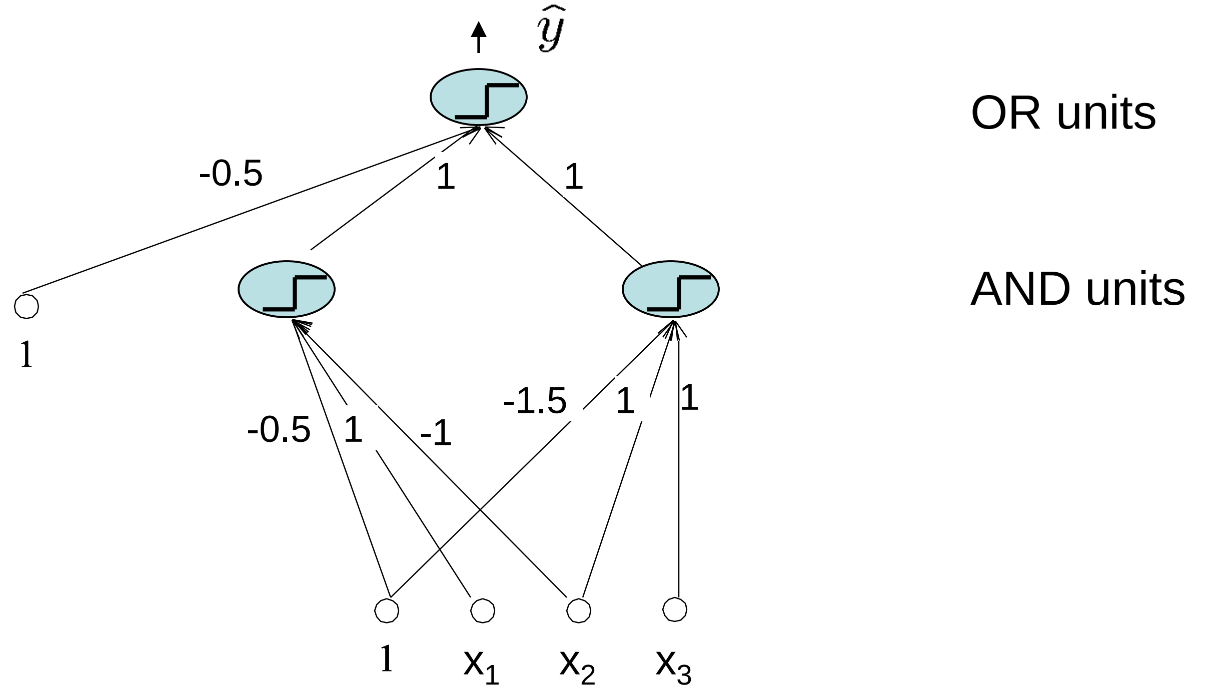


- Each layer receives its inputs from the previous layer and forwards its outputs to the next – <u>feed forward</u> structure
- Output layer: sigmoid activation function for classification, and linear activation function for regression
- Referred to as a two-layer network (2 layer of weights)

Representational Power

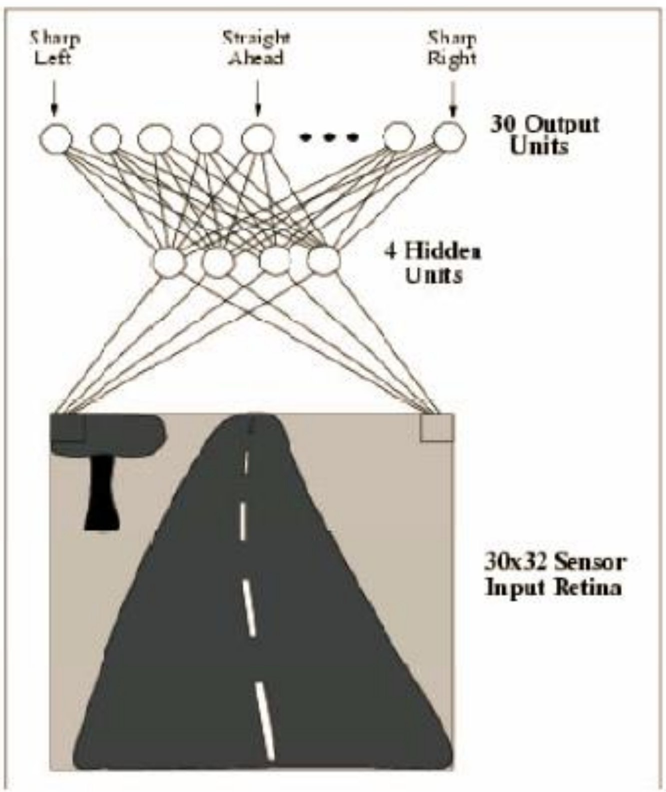
- <u>Any Boolean Formula</u>
  - Consider a formula in disjunctive normal form:

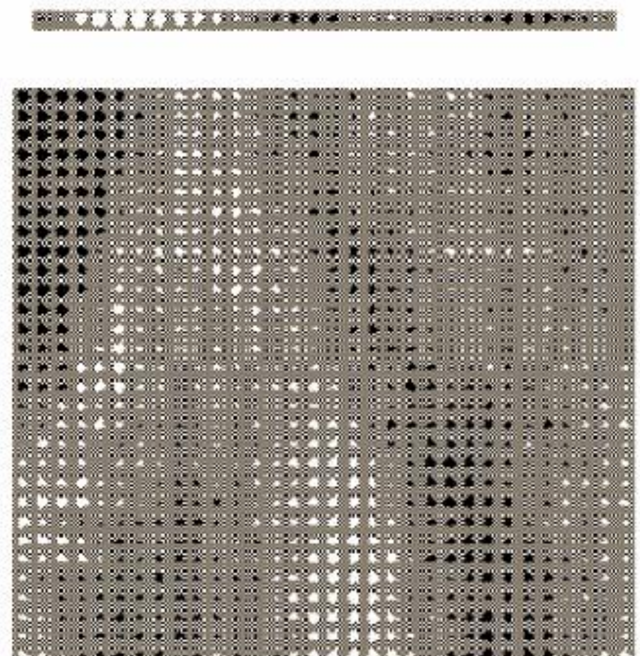
$$(x_1 \wedge \neg x_2) \vee (x_2 \wedge x_3)$$

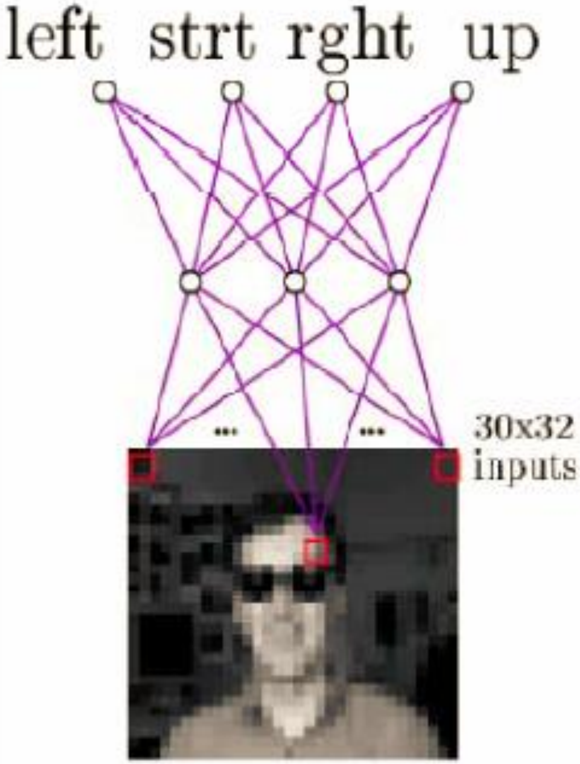


Example

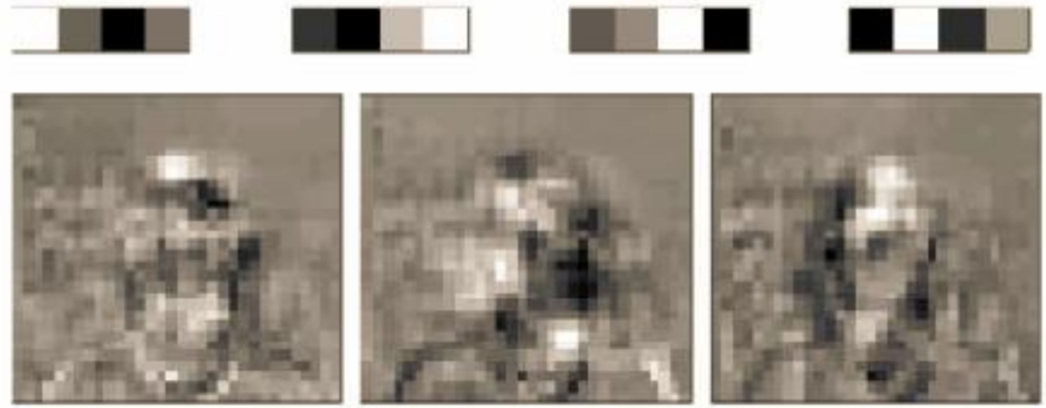
Neural net is one of the most effective methods when the data include complex sensory inputs such as images.

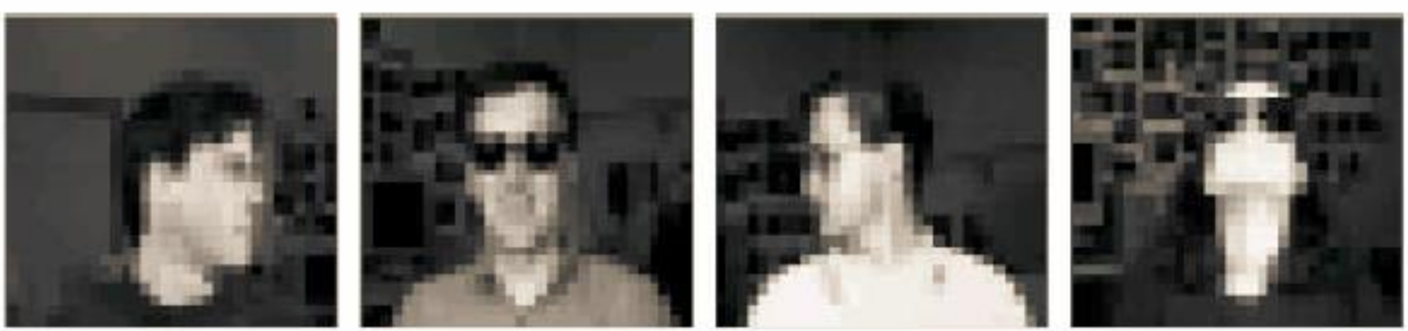






Learned Weights





Typical input images

Training: Backpropagation

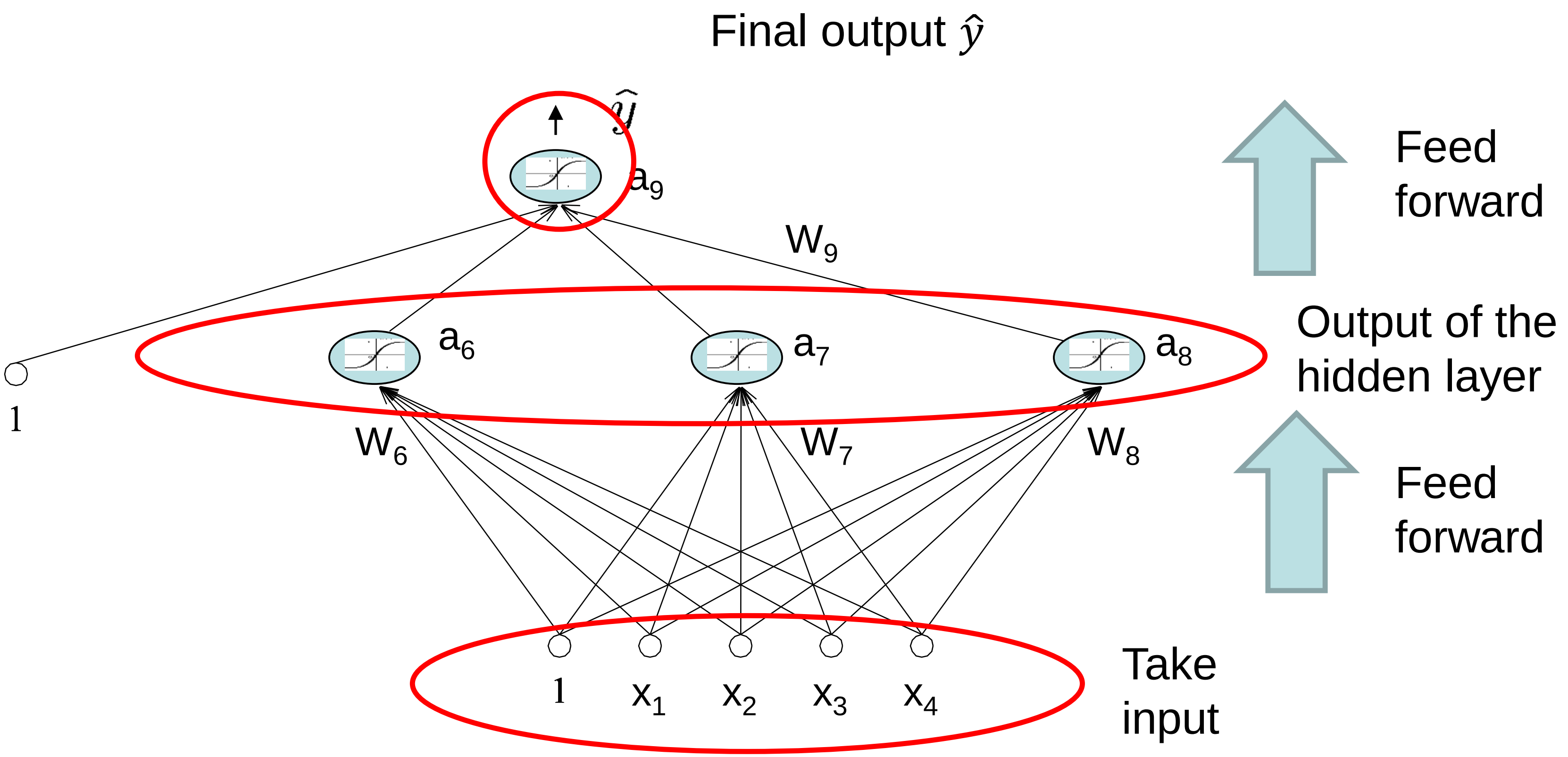
- Training of the neural net aims to find weights that minimize some loss function
- For example, for regression problem, denoting the network output for input x as $\hat{y}(x)$

$$L(\mathrm{w}) = \sum_{i=1}^{n} (\hat{y}(x_i, w) - y_i)^2$$

- For classification problems the loss can be different, e.g., negative log-likelihood
- Use gradient descent to iteratively improve the weights
- This is done from layer to layer, applying the chain rule to compute the gradient for each layer

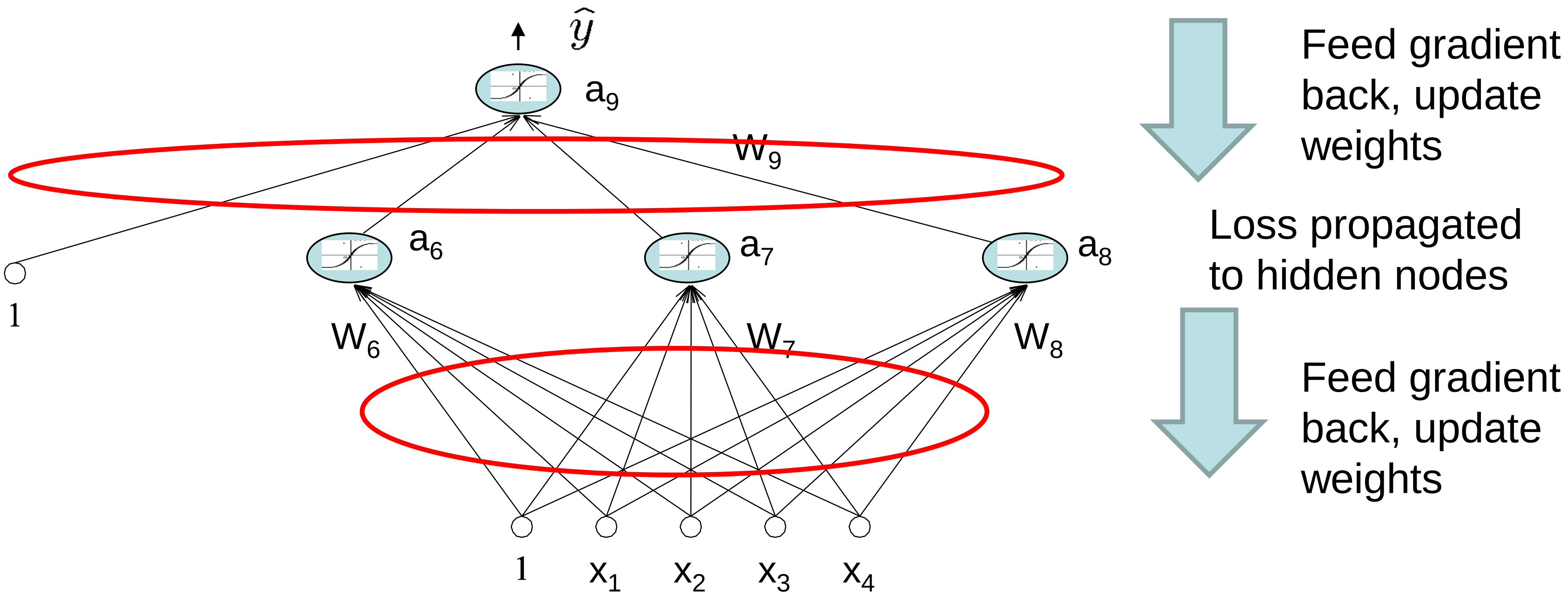
Chain rule for gradient: $\frac{df}{dx} = \frac{df}{dy}\frac{dy}{dx}$

Training: the forward pass



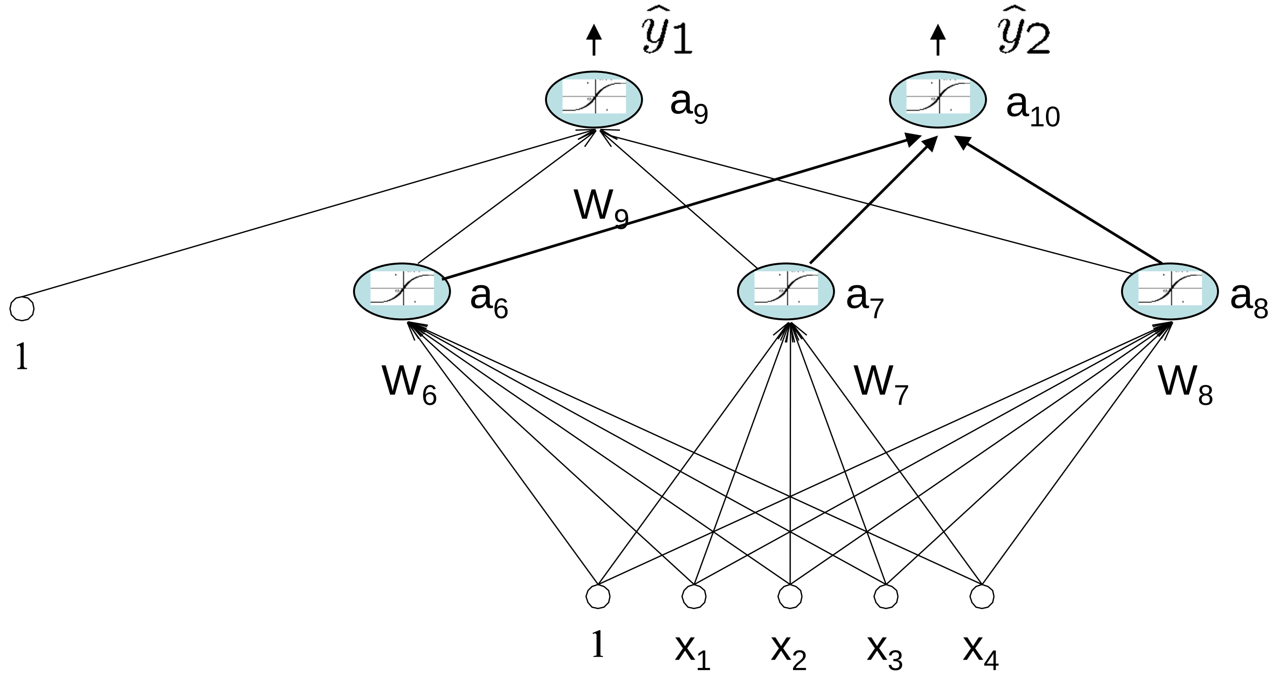
Training: the backward pass

Compute Loss $L(\hat{y}(x), y)$



The calculation of the gradient will depend on the loss function and the activation function – but often it is not complicated
E.g., if we use the same loss as logistic regression, we have the same update rule for updating the outer most weight layer

Networks with Multiple Output Units



Backpropagation Training

- Initialize all the weights with small random values
- Repeat
  - For all training examples, do

    Begin Epoch

      For each training example do

      - Compute the network output
      - Compute loss
      - Backpropagate this loss from layer to layer and adjust weights to decrease this loss

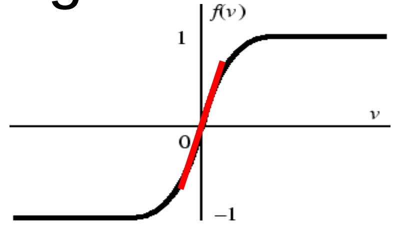
    End Epoch

Remarks on Training

- Not guaranteed to convergence, may oscillate or reach a local minima.
- However, in practice many large networks can be adequately trained on large amounts of data for realistic problems, e.g.,
  - Driving a car
  - Recognizing handwritten zip codes
  - Play world championship level Backgammon
- Many epochs (thousands) may be needed for adequate training, large data sets may require hours or days of training
- Termination criteria can be:
  - Fixed number of epochs
  - Threshold on training set error
  - Increased error on a validation set
- To avoid local minima problems, can run several trials starting from different initial random weights and:
  - Take the result with the best training or validation performance.
  - Build a committee of networks that vote during testing, possibly weighting vote by training or validation accuracy

Notes on Proper Initialization

- Start in the "linear" regions
  - keep all weights near zero, so that all sigmoid units are in their linear regions. This makes the whole net the equivalent of one linear threshold unit—a relatively simple function.
  - This will also avoid having very small gradient



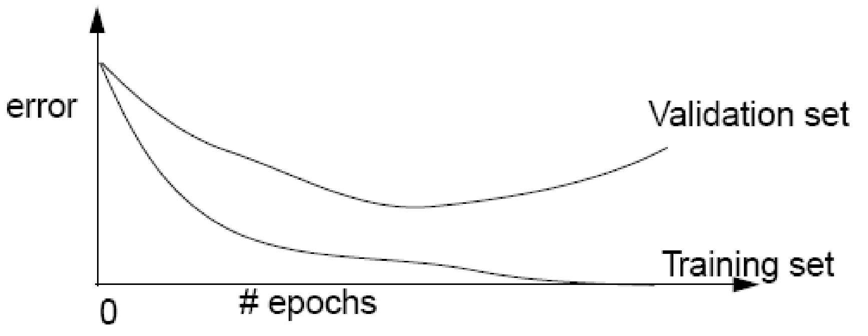
- Break symmetry
  - If we start with all the weights equal, what would happen?
  - Ensure that each hidden unit has different input weights so that the hidden units move in different directions.

Batch, Online and Online with Momentum

- <u>Batch.</u> Sum up the gradient for a batch of examples and take a combined gradient step
- <u>Online:</u> Take a gradient step for each example
- <u>Momentum:</u> each update linearly combines the current gradient with the previous update direction to ensure smoother convergence

Overtraining Prevention

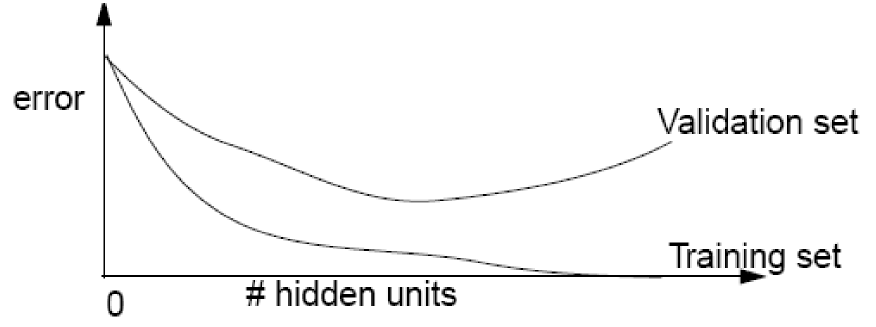
- Running too many epochs may overtrain the network and result in overfitting.



- Keep a validation set and test accuracy after every epoch. Maintain weights for best performing network on the validation set and return it when performance decreases significantly beyond this.

Over-fitting Prevention

- Too few hidden units underfit the data and fail to learn the concept.
- Too many hidden units over-fit



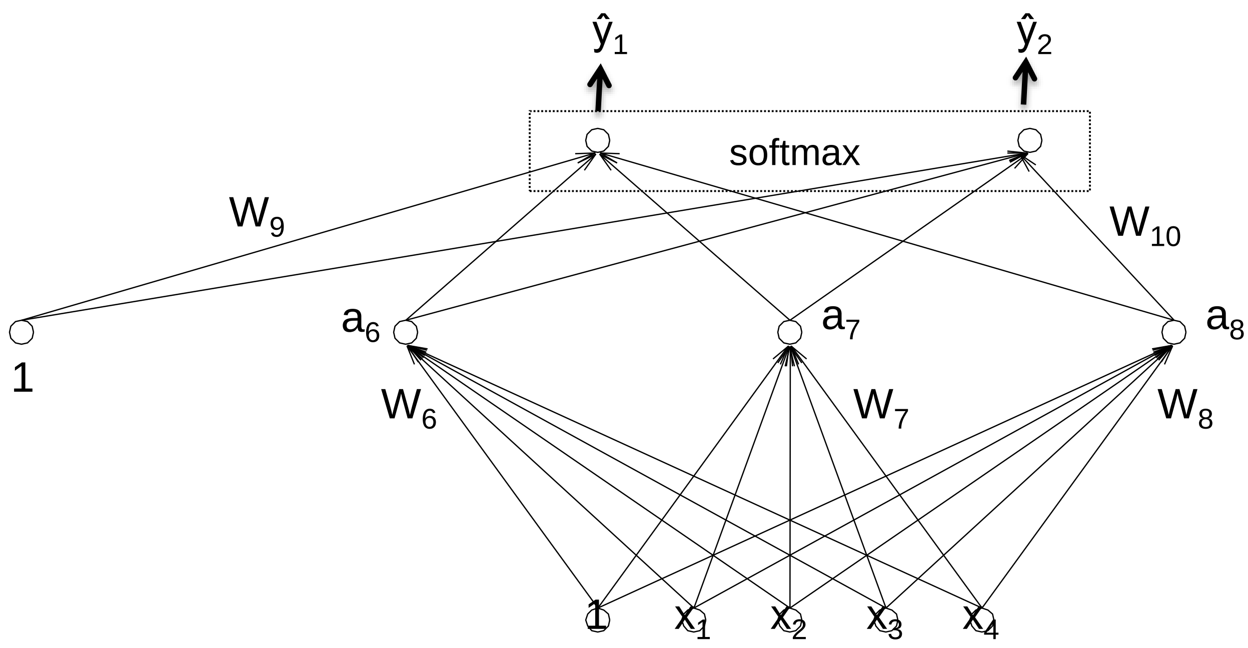
- Cross-validation can be used to decide the right number of hidden units.
- ***Weight decay*** multiplies all weights by some fraction between 0 and 1 after each epoch.
  - Encourages smaller weights and less overfitting
  - Equivalent to including a regularization term to the loss

Input/Output Coding

- Appropriate coding of inputs/outputs can make learning easier and improve generalization.
- Best to encode discrete multi-category features using multiple input units and include one binary unit per value
- Continuous inputs can be handled by a single input unit, but scaling them between 0 and 1
- For classification problems, best to have one output unit per class.
  - Continuous output values then represent certainty in various classes.
  - Assign test instances to the class with the highest output.
- Use target values of 0.9 and 0.1 for binary problems rather than forcing weights to grow large enough to closely approximate 0/1 outputs.
- Continuous outputs (regression) can also be handled by scaling to the range between 0 and 1

Softmax for multi-class classification

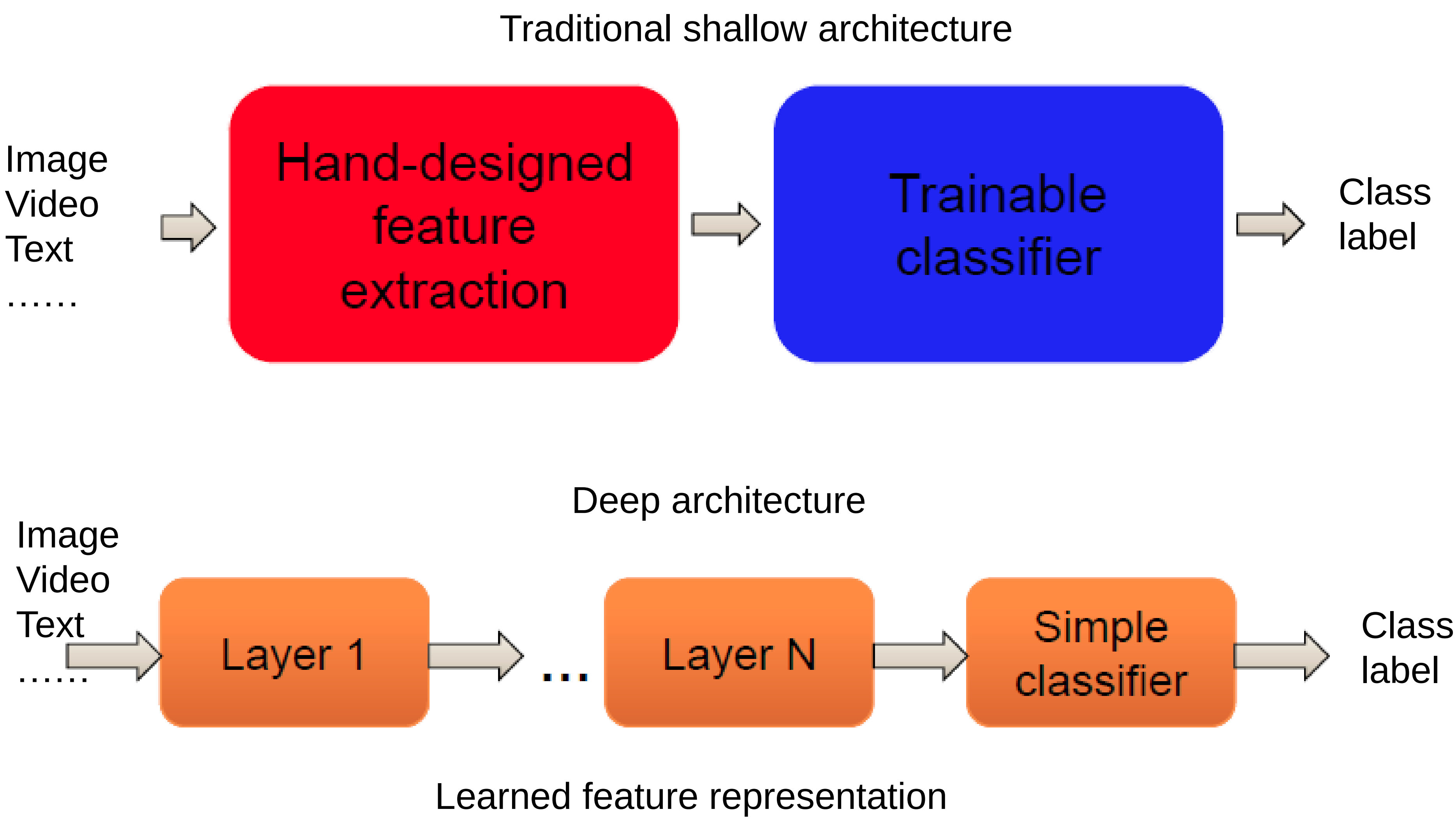
- For K classes, we have K nodes in the output layer, one for each class
- Let a_k be the output of the class-k node, i.e. $a_k = (w_k \cdot A)$, where A is the output of the hidden layer, and w_k is the weight vector leading into the class-k node
- We define: $P(y = k|\mathbf{x}) = \frac{\exp a_k}{\sum_{i=1}^{K} \exp a_i}$



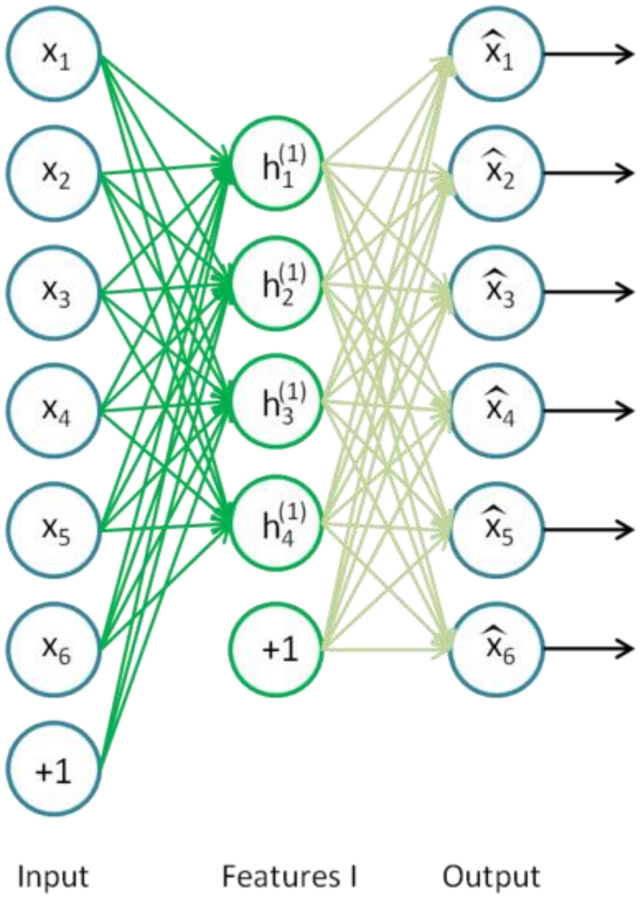
Recent Development

- A recent trend in ML is deep learning, which learns feature hierarchies from large amounts of unlabeled data
- The feature hierarchies are expected to capture the inherent structure in the data
- Can often lead to better classification when used the learned features to train with labeled data
- Neural networks provide one approach for deep learning

Shallow vs Deep Architectures

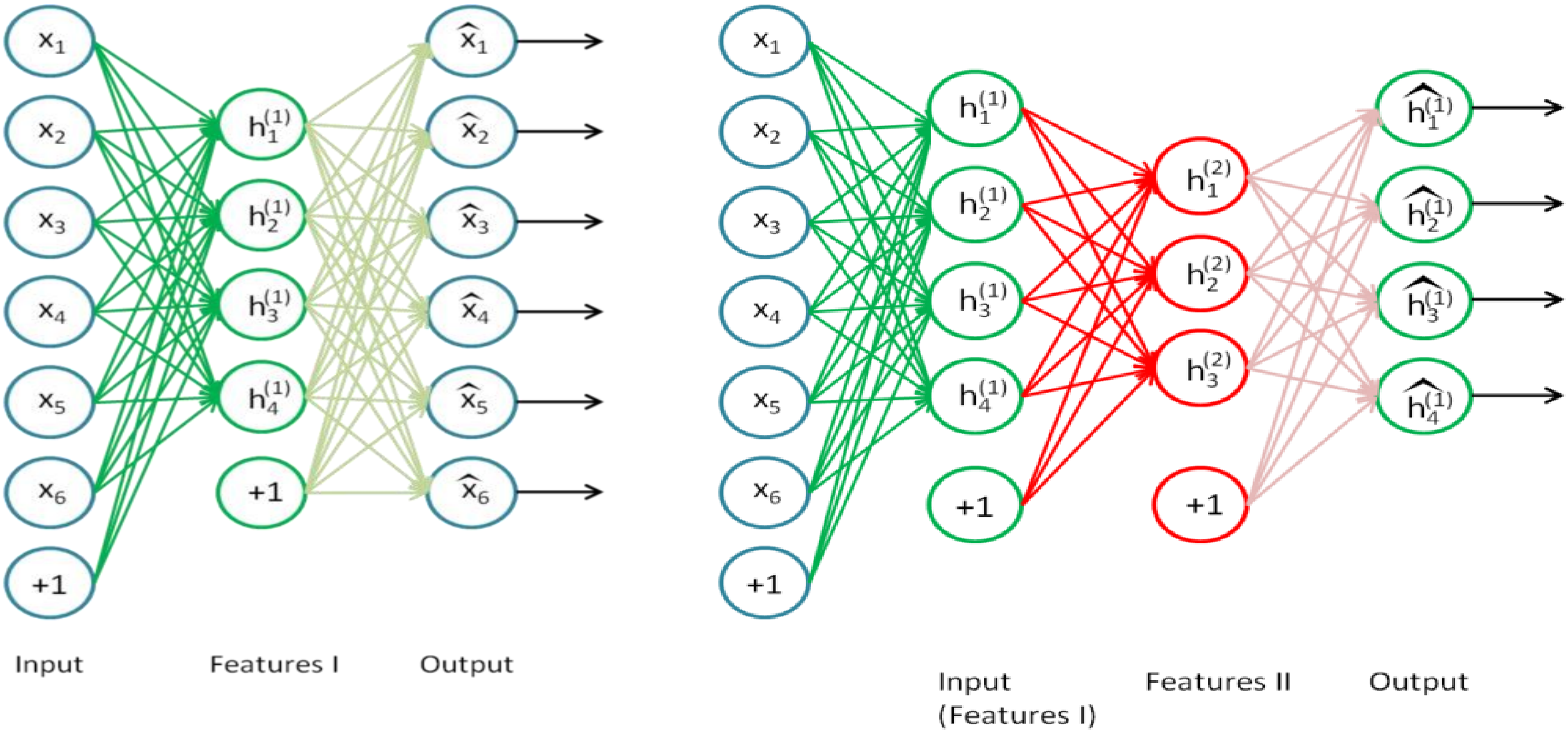


Deep learning with Auto-encoders



- Network is trained to output the input
- The hidden layer serves to “extract” features from the input that is essential for representation
  - **Constraining layer 2 to be sparse**
- Training is carried out with the same back-prop procedure

Deep structure: Stacked Auto-encoders



- This continues … and learns a feature hierarchy
- The final feature layer can then be used as features for supervised learning
- Can also do supervised training on the entire network to fine-tune all weights

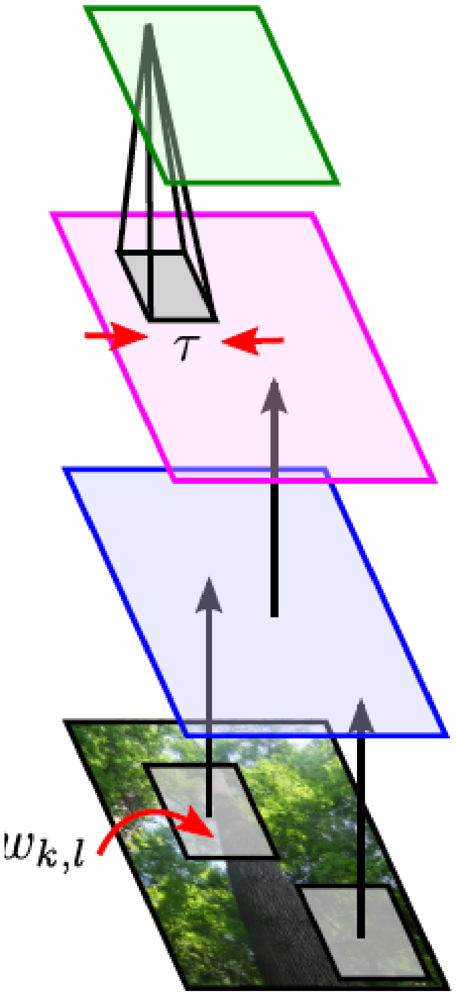
Convolutional neural networks

- A different network structure that has been extremely successful in handling visual, textual and audio data
- Current state of the art on many computer vision tasks

Key ideas behind convolutional neural networks

- Image statistics are translation invariant
  - Need to build translation invariance into the model
  - Tie parameters together in the network
  - Reduce number of parameters
- Low level features/patterns should be local
  - Network should have only local connectivity
  - Reduce # of parameters
- High-level features/patterns will be coarser
  - We can zoom out by subsampling and still capture the high level patterns well

Building blocks of CNN



$$x_{i,j} = \max_{|k|<\tau, |l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$

non-linear stage

$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

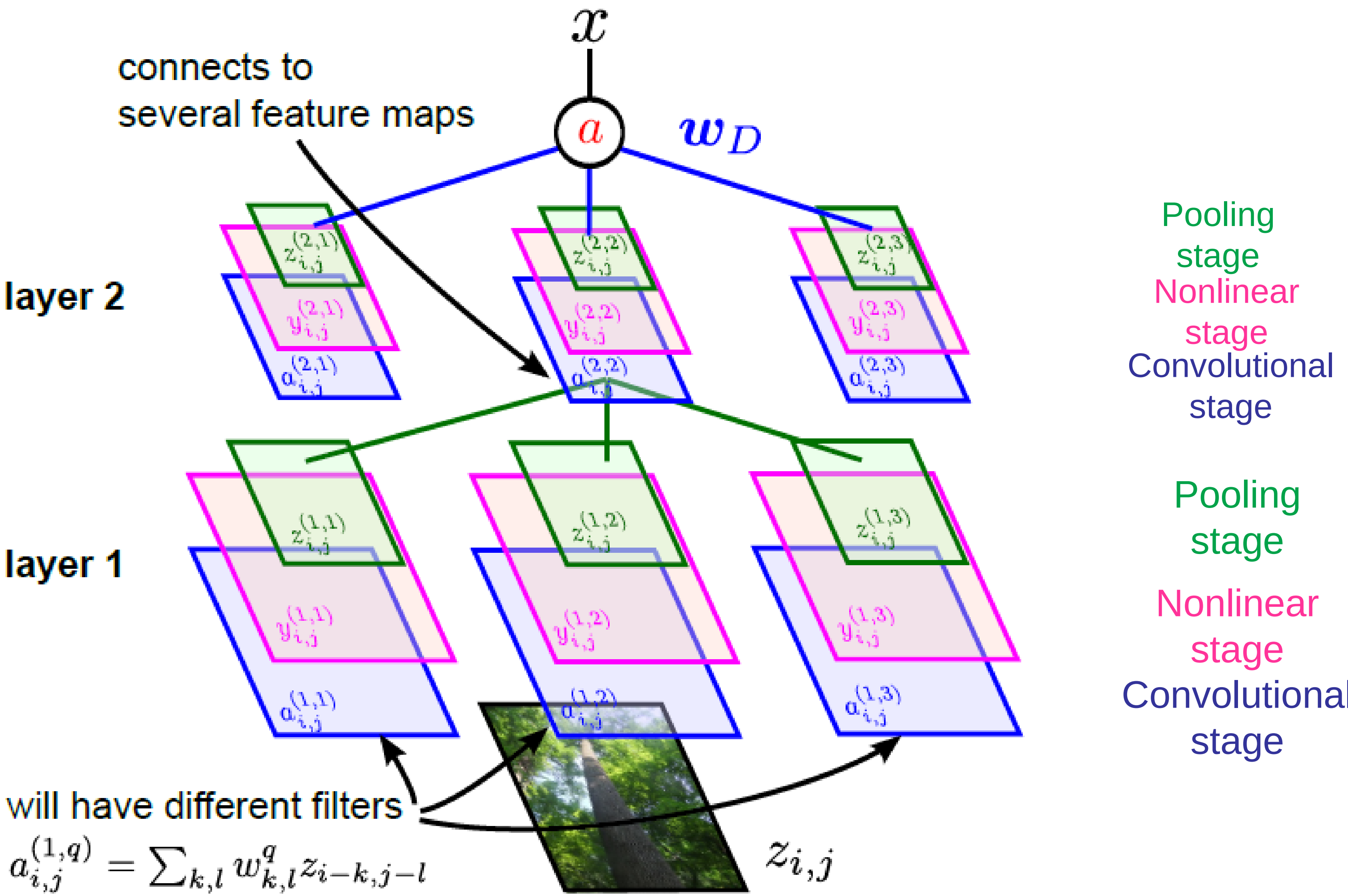
only parameters

convolutional stage

$z_{i,j}$

input image

Full CNN



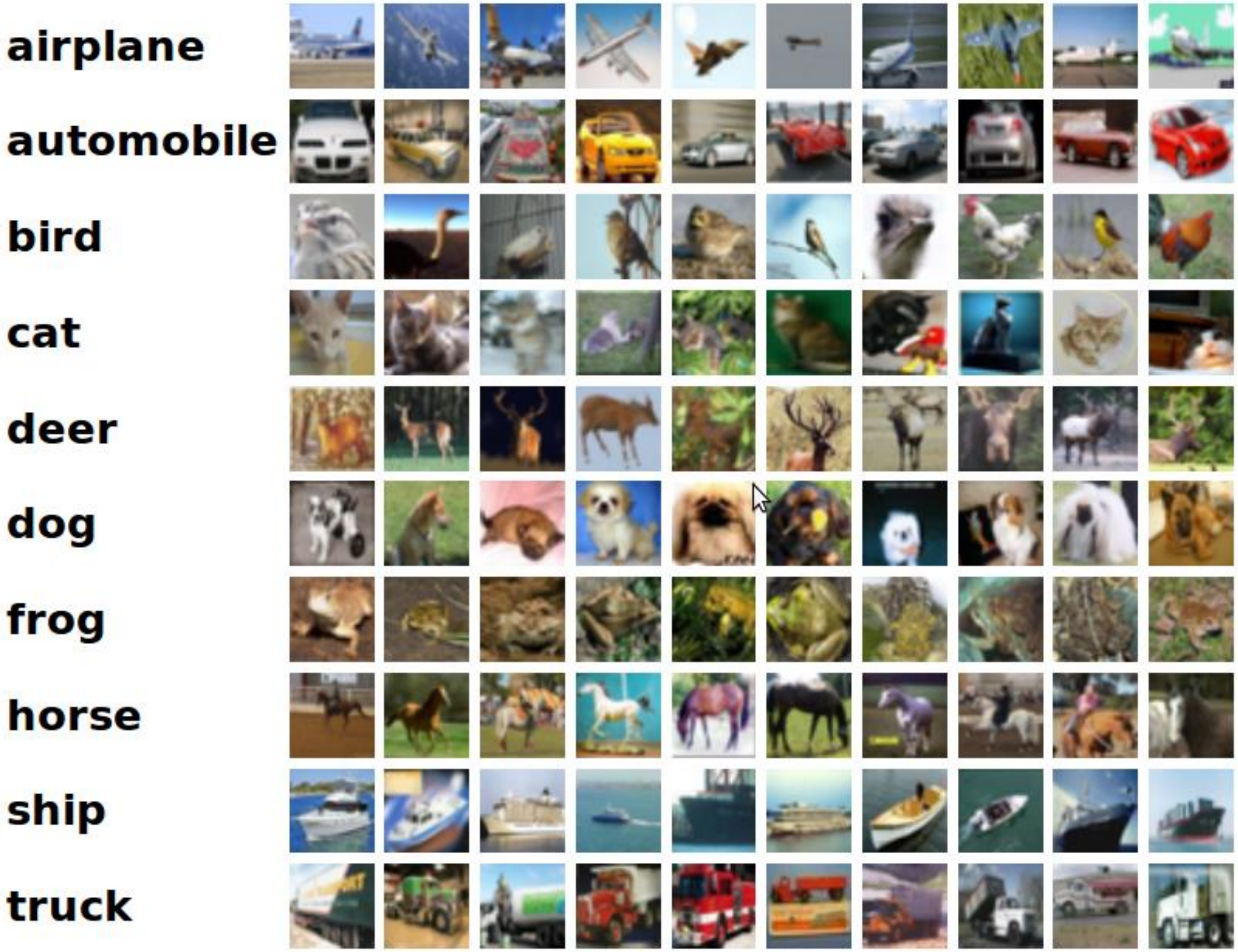
x
a
w_D
connects to
several feature maps
layer 2
layer 1
$z_{i,j}^{(2,1)}$
$y_{i,j}^{(2,1)}$
$a_{i,j}^{(2,1)}$
$z_{i,j}^{(2,2)}$
$y_{i,j}^{(2,2)}$
$a_{i,j}^{(2,2)}$
$z_{i,j}^{(2,3)}$
$y_{i,j}^{(2,3)}$
$a_{i,j}^{(2,3)}$
$z_{i,j}^{(1,1)}$
$y_{i,j}^{(1,1)}$
$a_{i,j}^{(1,1)}$
$z_{i,j}^{(1,2)}$
$y_{i,j}^{(1,2)}$
$a_{i,j}^{(1,2)}$
$z_{i,j}^{(1,3)}$
$y_{i,j}^{(1,3)}$
$a_{i,j}^{(1,3)}$
will have different filters
$a_{i,j}^{(1,q)} = \sum_{k,l} w_{k,l}^{q} z_{i-k,j-l}$
$z_{i,j}$
Pooling stage
Nonlinear stage
Convolutional stage
Pooling stage
Nonlinear stage
Convolutional stage

Training

- back-propagation for training
- data-augmentation: include shifted, rotations, mirroring, locally distorted versions of the training data
  - Often improves performance substantially
- typical numbers:
  - 5 convolutional layers, 3 fully connected layers in the top
  - 500,000 neurons
  - 50,000,000 parameters
  - 1 week to train (GPUs)

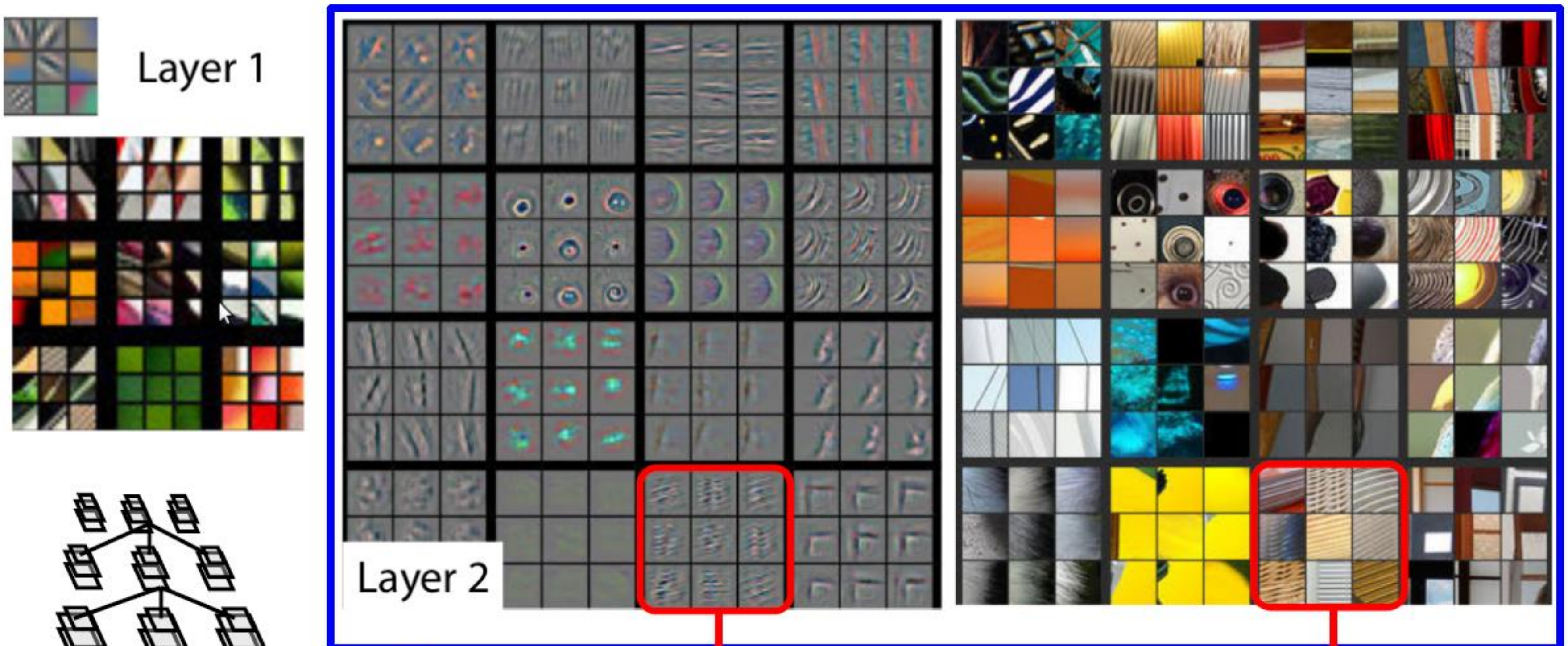
Demo

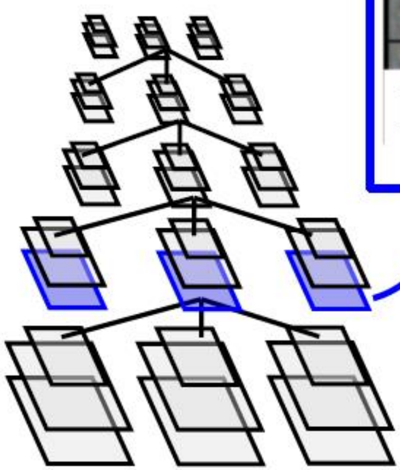
http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html



CIFAR 10 dataset: 50,000 training images, 10,000 test images

Looking inside

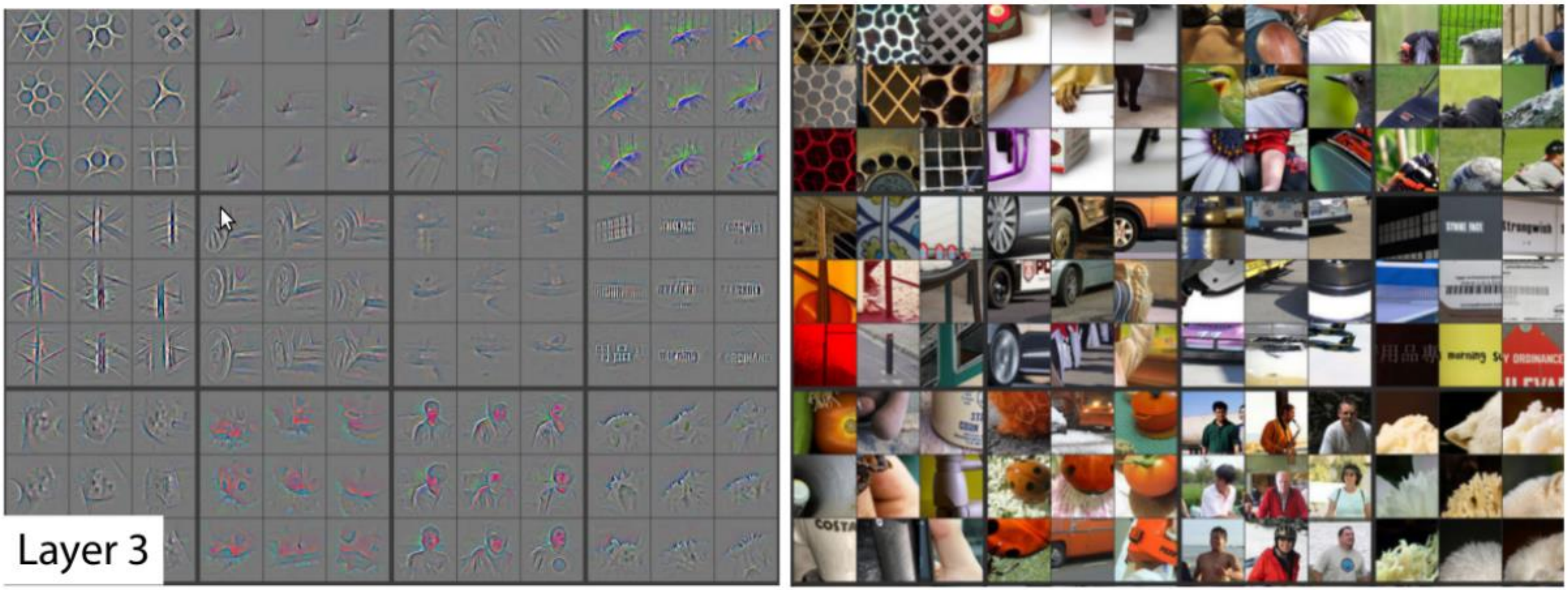




reconstruction of image patches from that unit
(indicates aspect of patches which unit is sensitive to)

top 9 image patches that cause maximal activation in layer 2 unit

Looking inside



- Higher level layers encode more abstract features
- Automatically learn useful features for differentiating different classes

Summary

- That’s a basic intro
- There are many many types of deep learning
- Different kinds of autoencoder, variations on architectures and training algorithms, etc …
- Various packages: Theano, Caffe, Torch, Convnet …
- Tremendous impact in vision, speech and natural language processing
- Very fast growing area …