

Machine Learning for Vision: Random Decision Forests and Deep Neural Networks

Kari Pulli

Senior Director

NVIDIA Research





Material sources

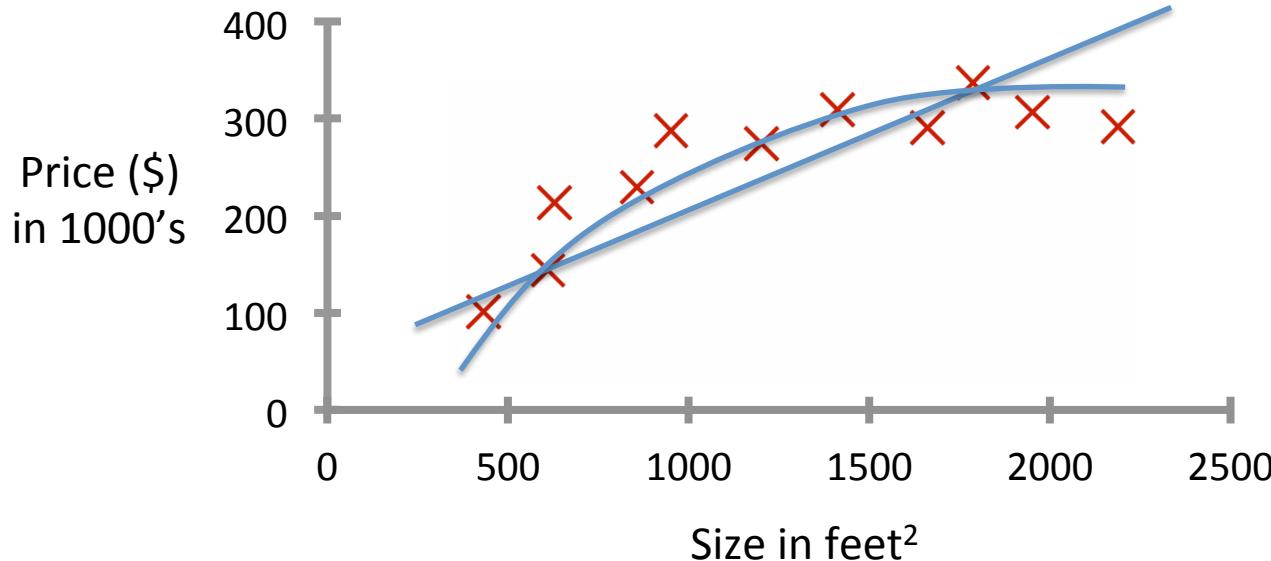
- **A. Criminisi & J. Shotton: Decision Forests Tutorial**
 - <http://research.microsoft.com/en-us/projects/decisionforests/>
- **J. Shotton et al. (CVPR11): Real-Time Human Pose Recognition in Parts from a Single Depth Image**
 - <http://research.microsoft.com/apps/pubs/?id=145347>
- **Geoffrey Hinton: Neural Networks for Machine Learning**
 - <https://www.coursera.org/course/neuralnets>
- **Andrew Ng: Machine Learning**
 - <https://www.coursera.org/course/ml>
- **Rob Fergus: Deep Learning for Computer Vision**
 - <http://media.nips.cc/Conferences/2013/Video/Tutorial1A.pdf>
 - <https://www.youtube.com/watch?v=qgx57X0fBdA>

Two types of supervised learning

- Each training case consists of
 - an input vector x
 - a target output t
- Regression: The target output is
 - real numbers
- Classification: The target output is
 - a class label

Our focus
today

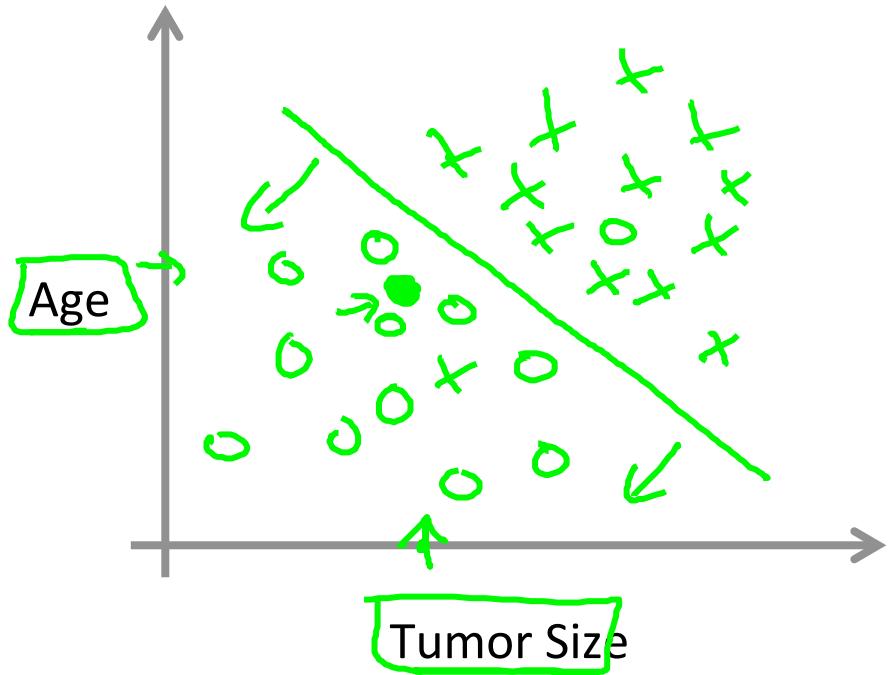
Housing price prediction



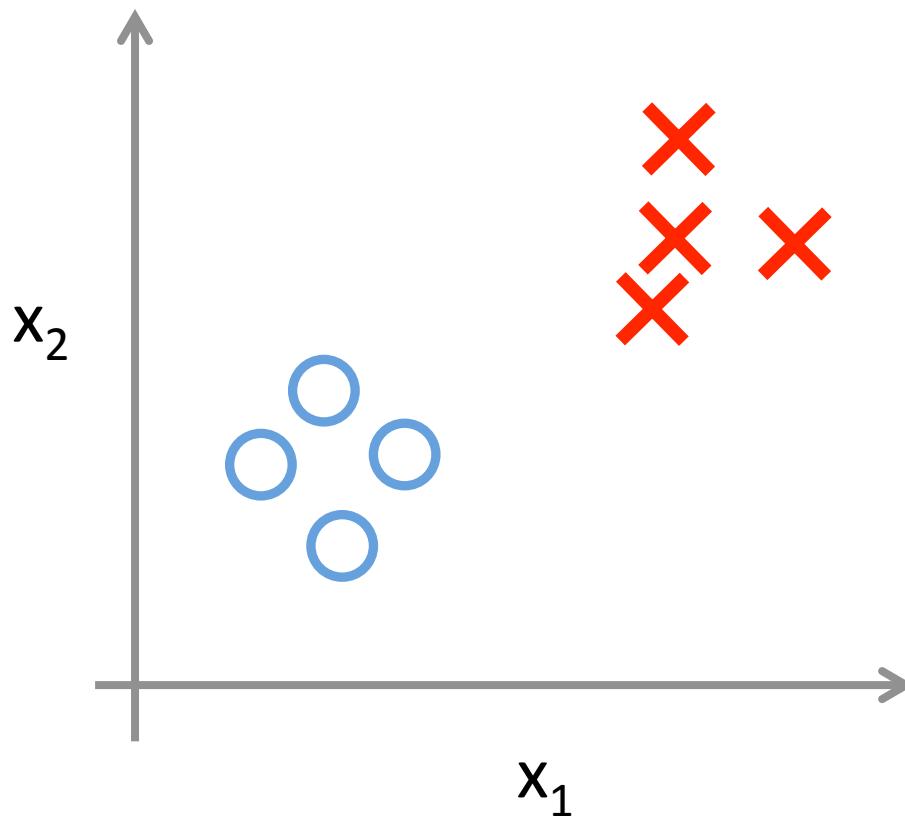
Supervised Learning
“right answers” given

Regression: Predict continuous
valued output (price)

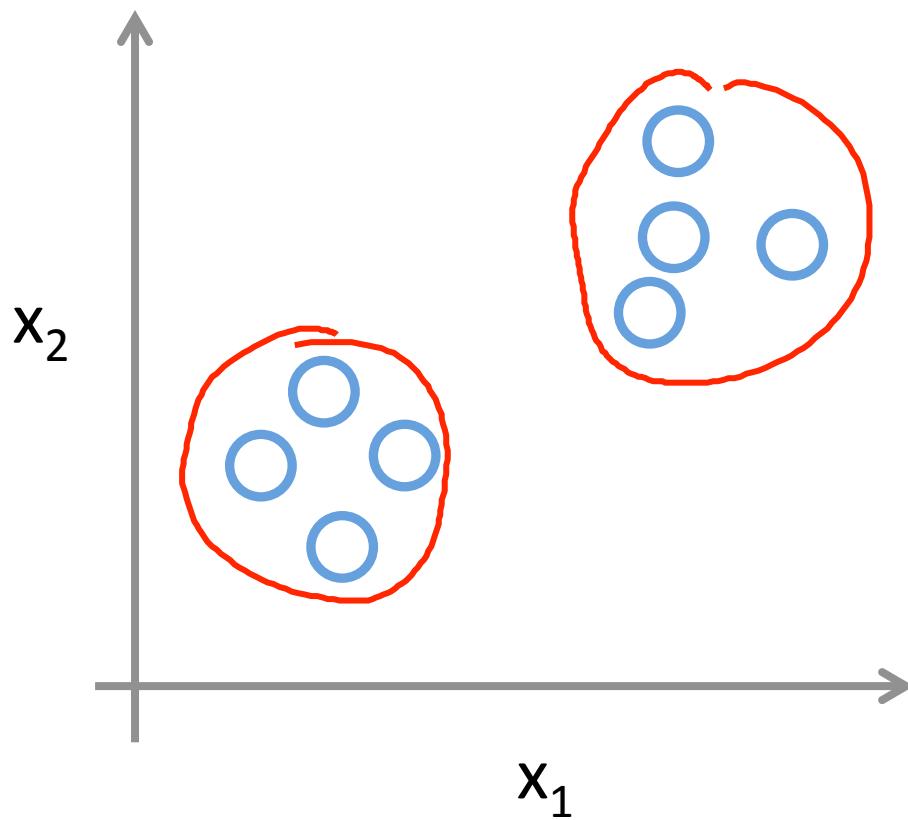
Breast cancer (malignant, benign)

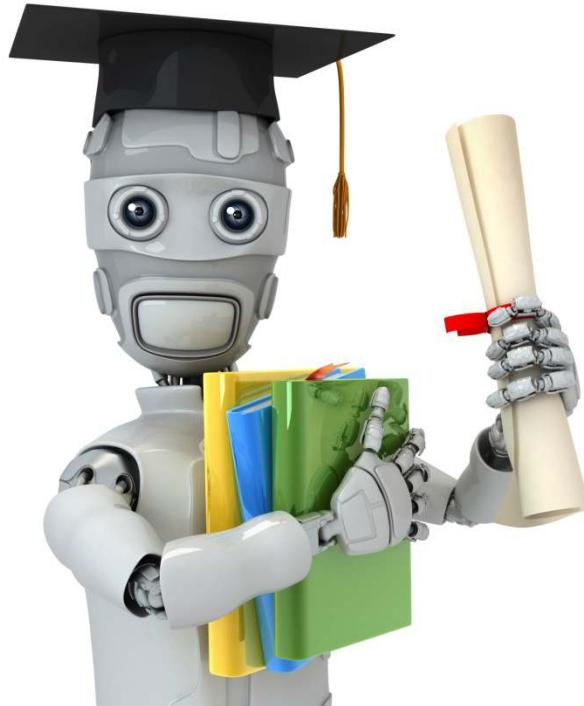


Supervised Learning



Unsupervised Learning

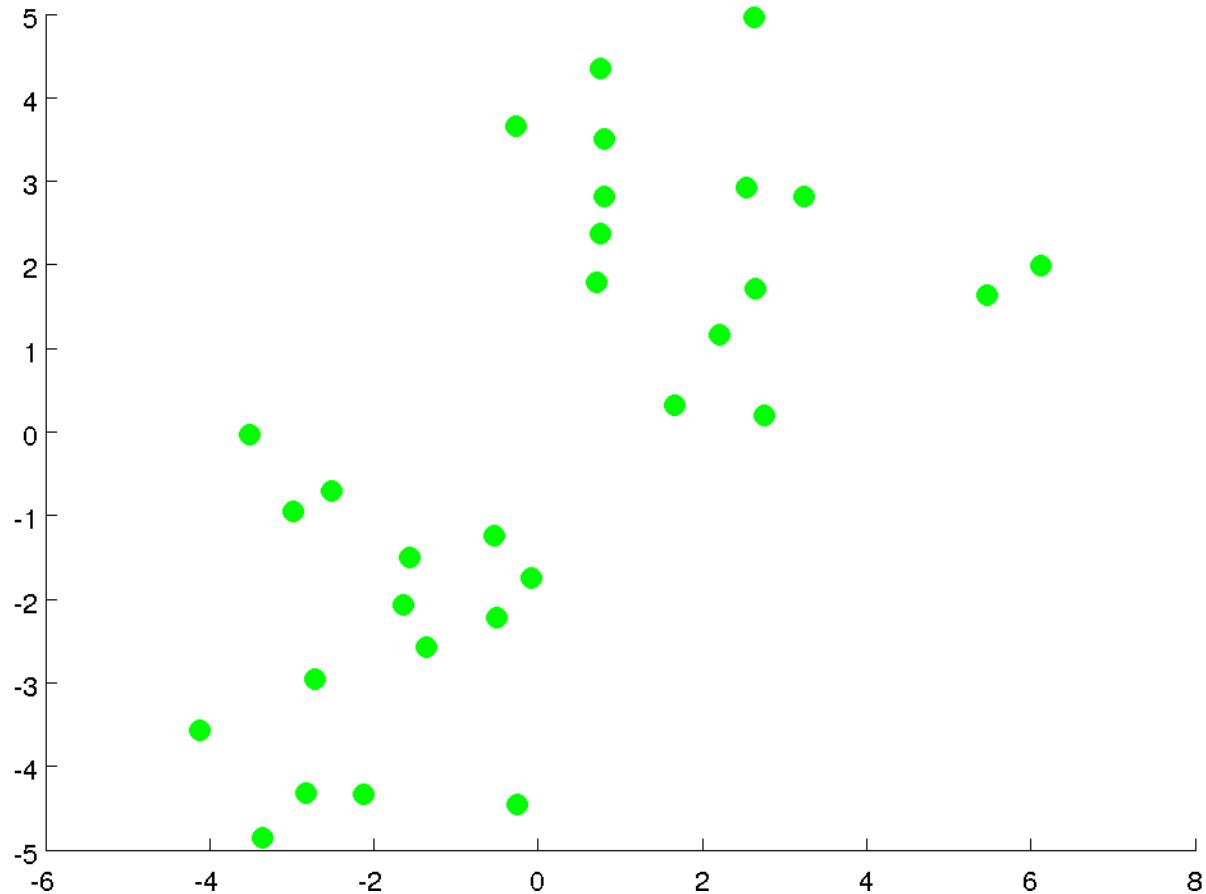


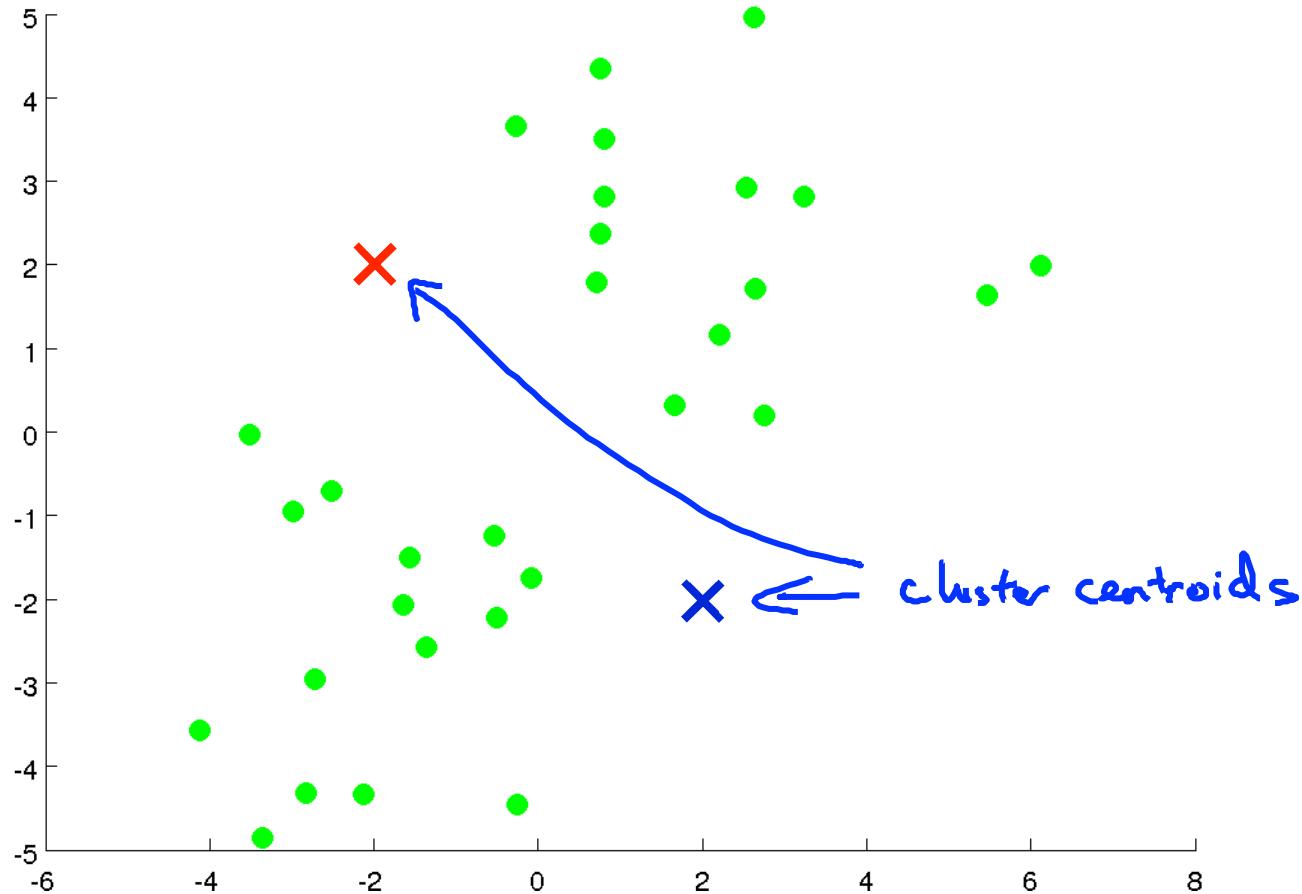


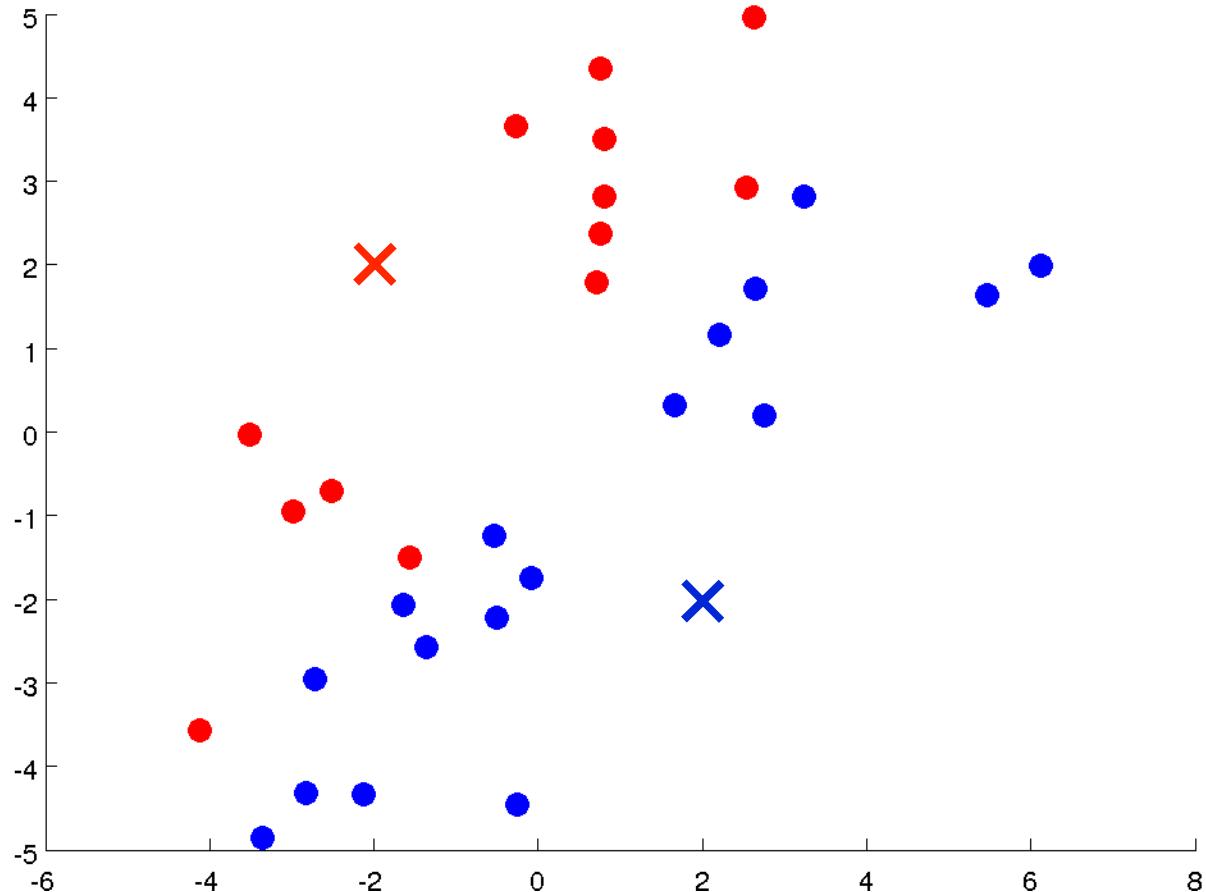
Machine Learning

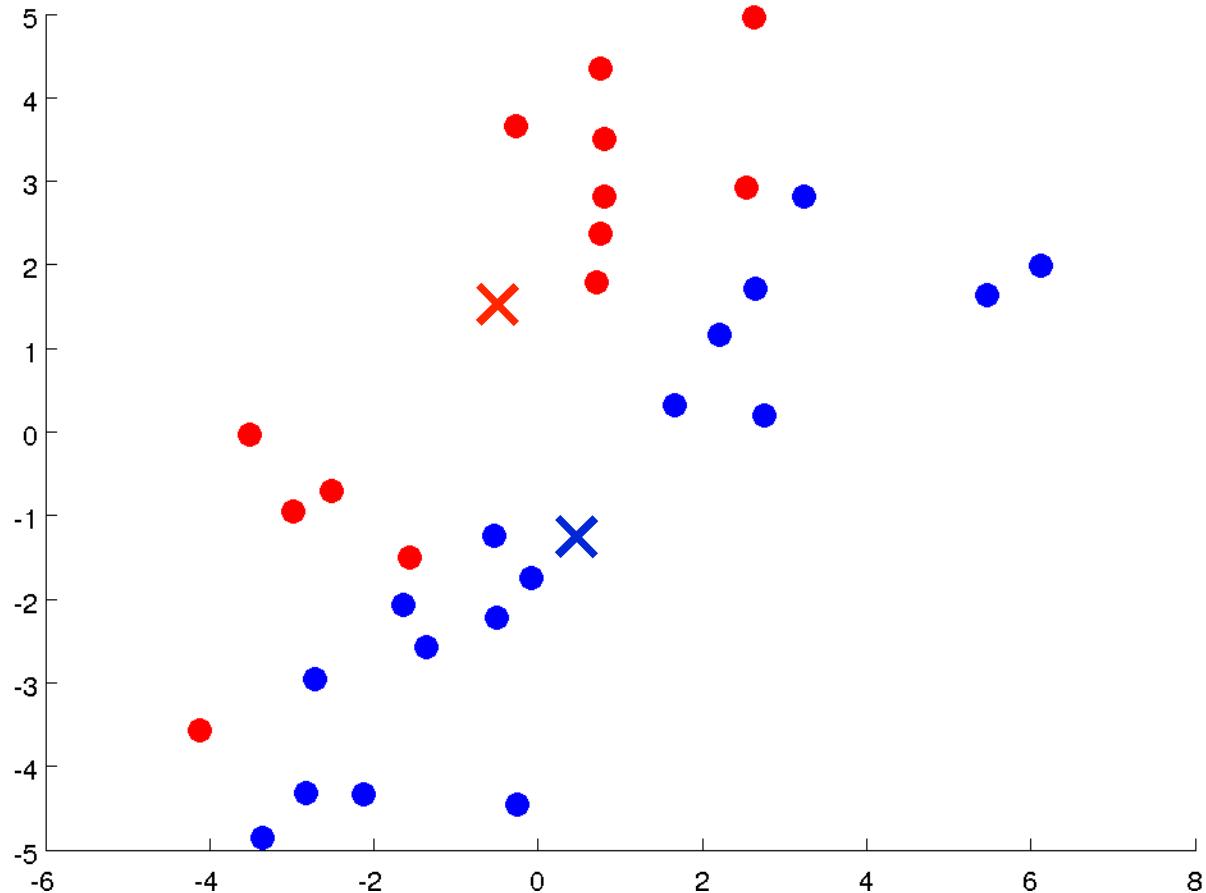
Clustering

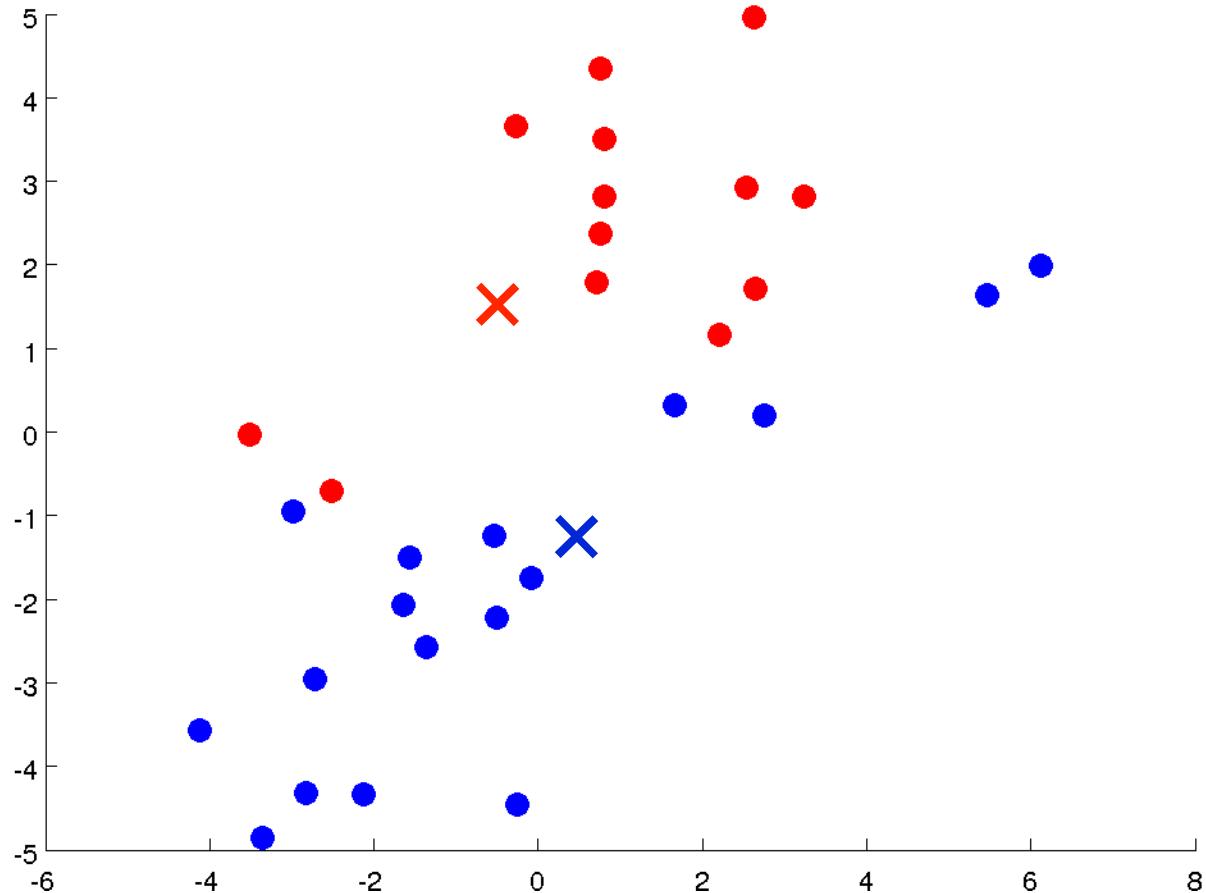
K-means algorithm

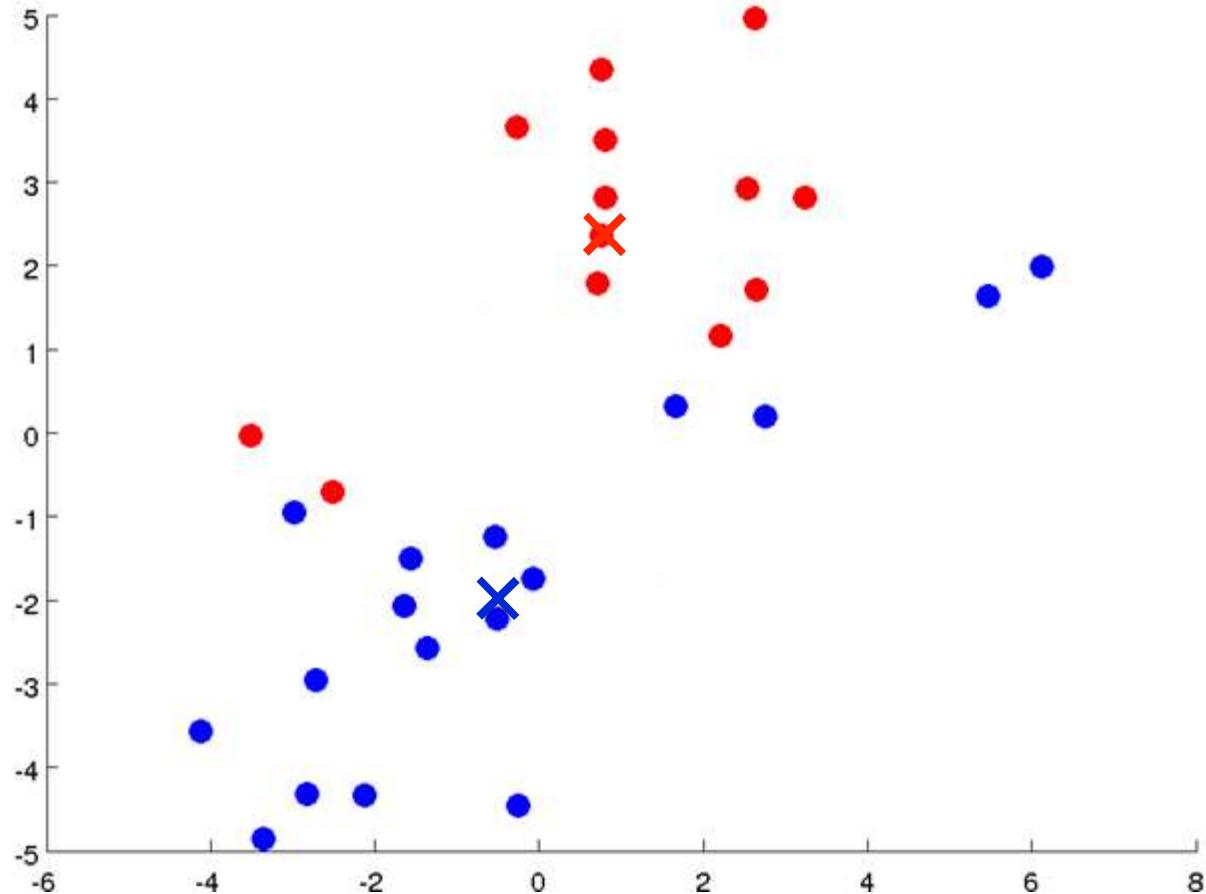


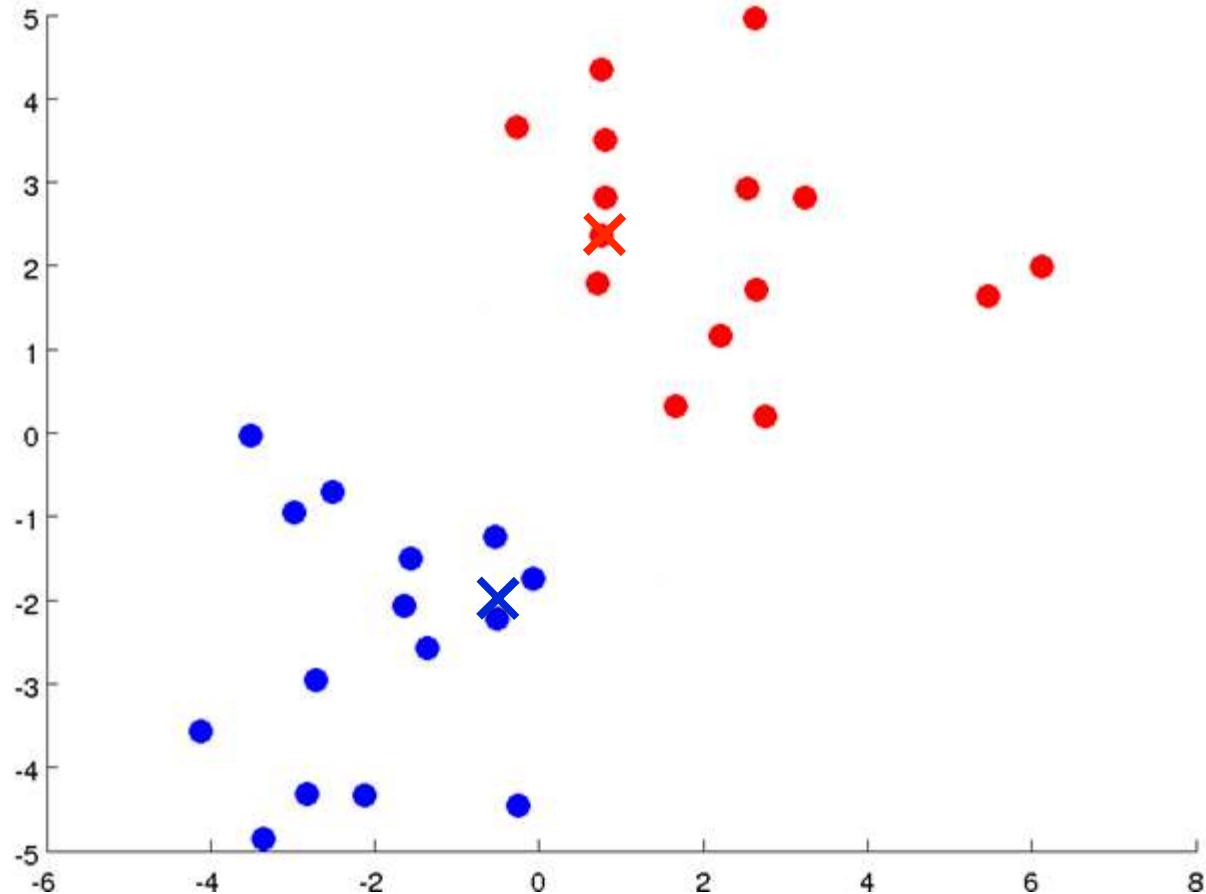


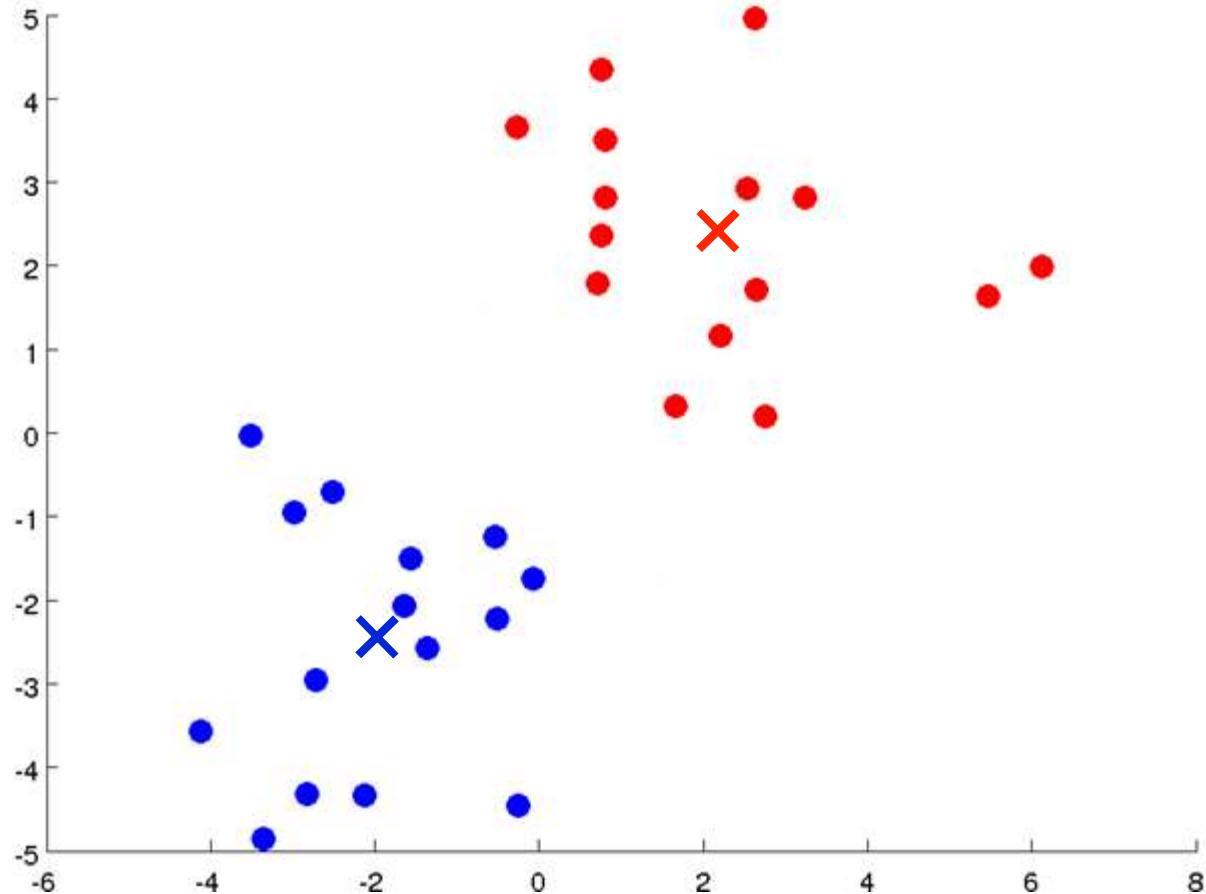












Decision Forests

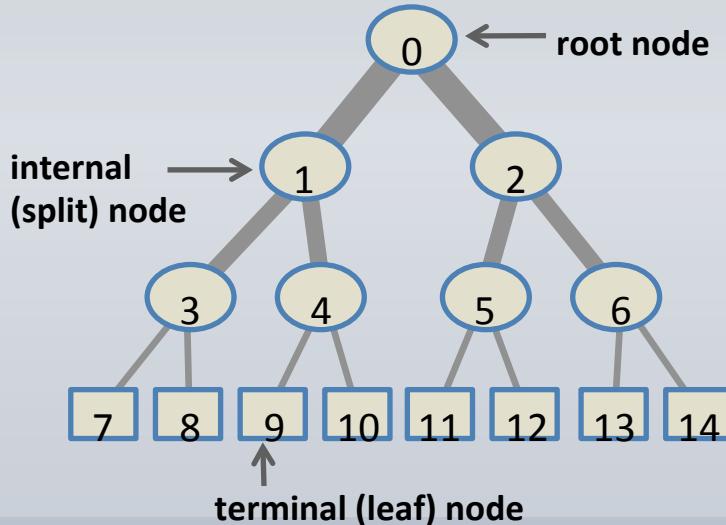
for computer vision and medical image analysis



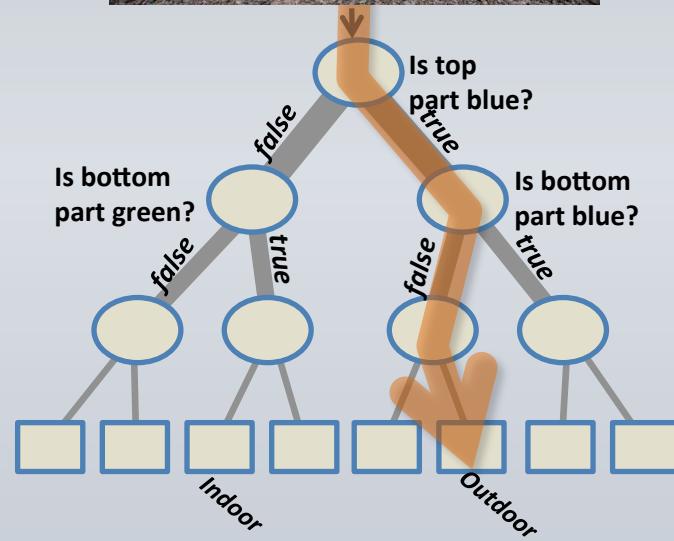
A. Criminisi, J. Shotton and E. Konukoglu

Decision trees and decision forests

A general tree structure

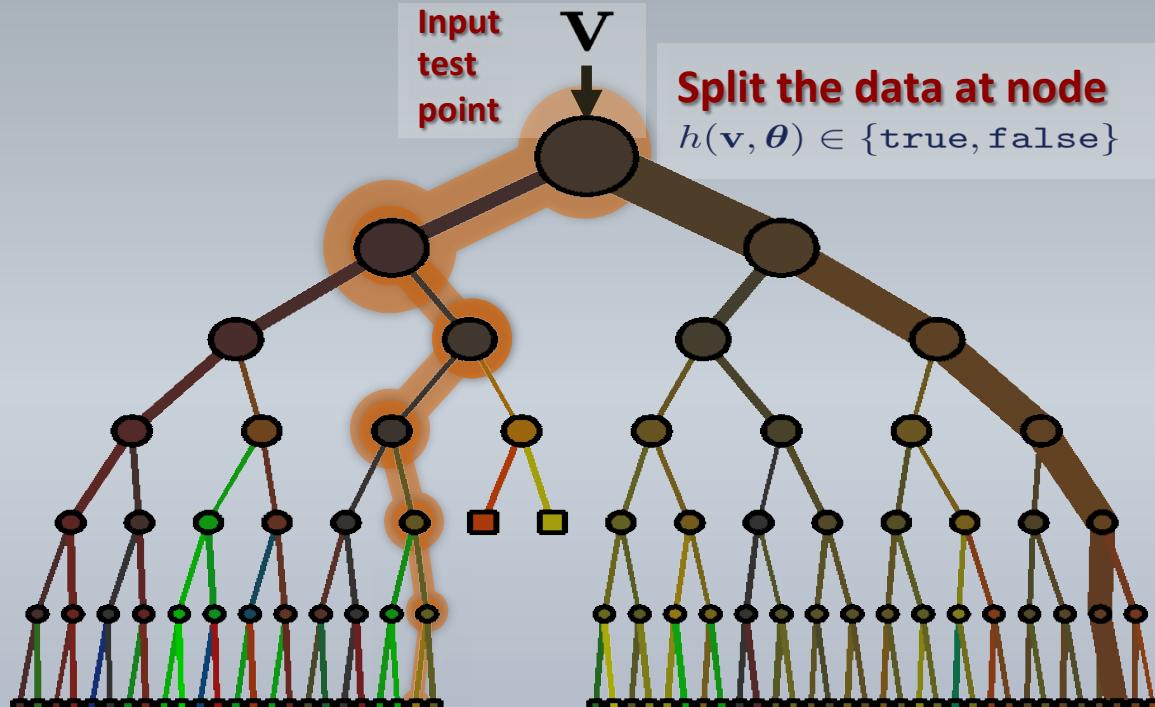
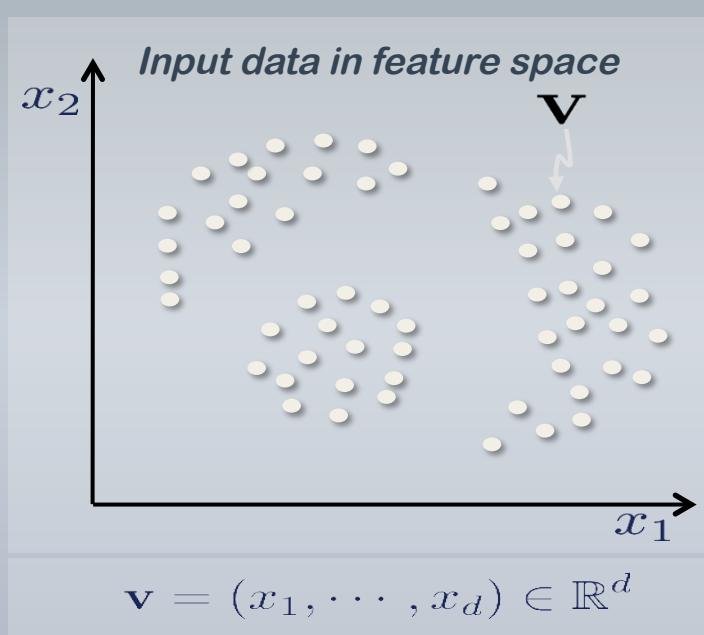


A decision tree



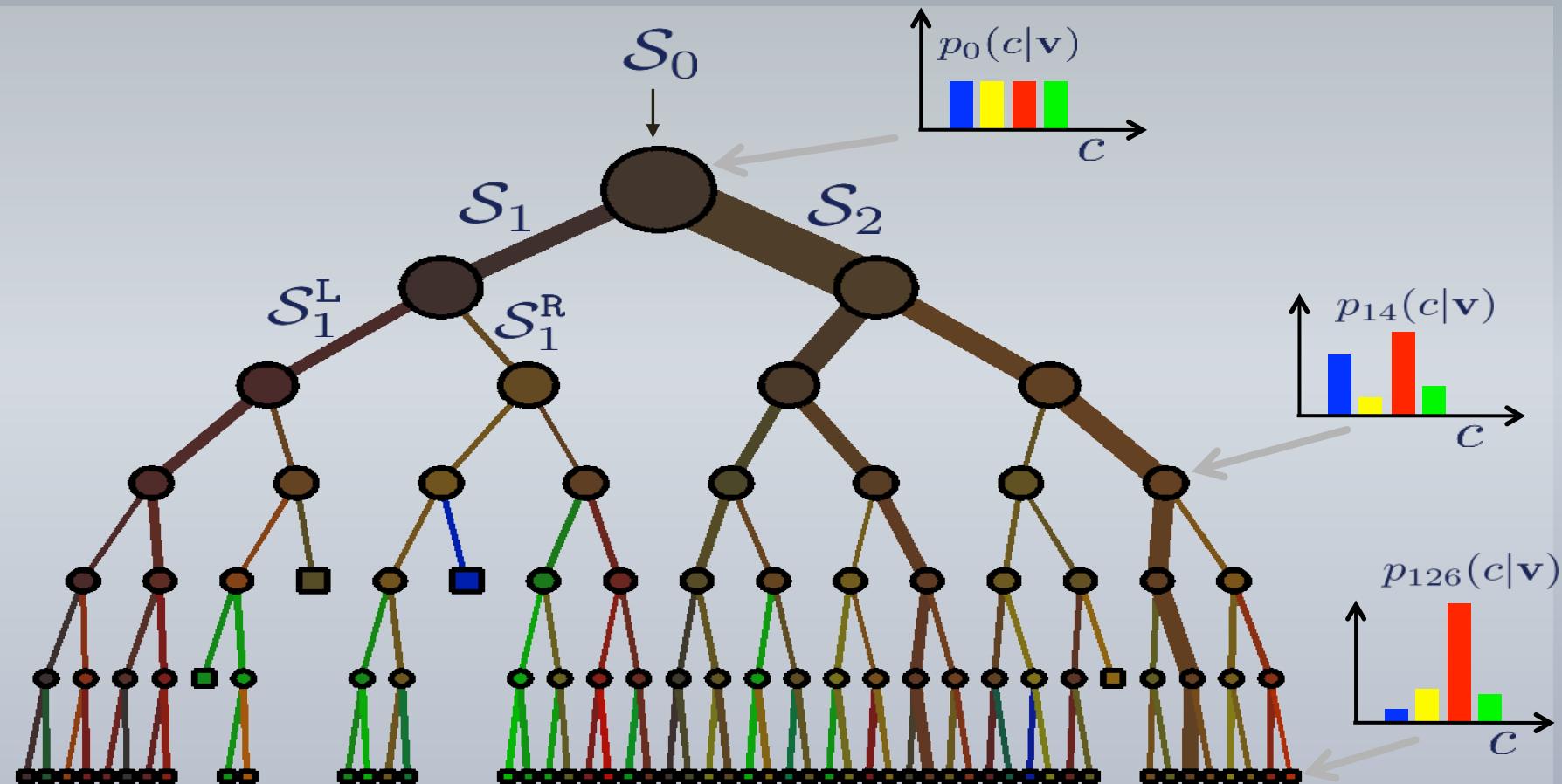
A forest is an ensemble of trees. The trees are all slightly different from one another.

Decision tree testing (runtime)



Prediction at leaf

$$p(c|\mathbf{v}) = \sum_j p(c|j)p(j|\mathbf{v})$$



Decision tree training (off-line)

How to split the data?

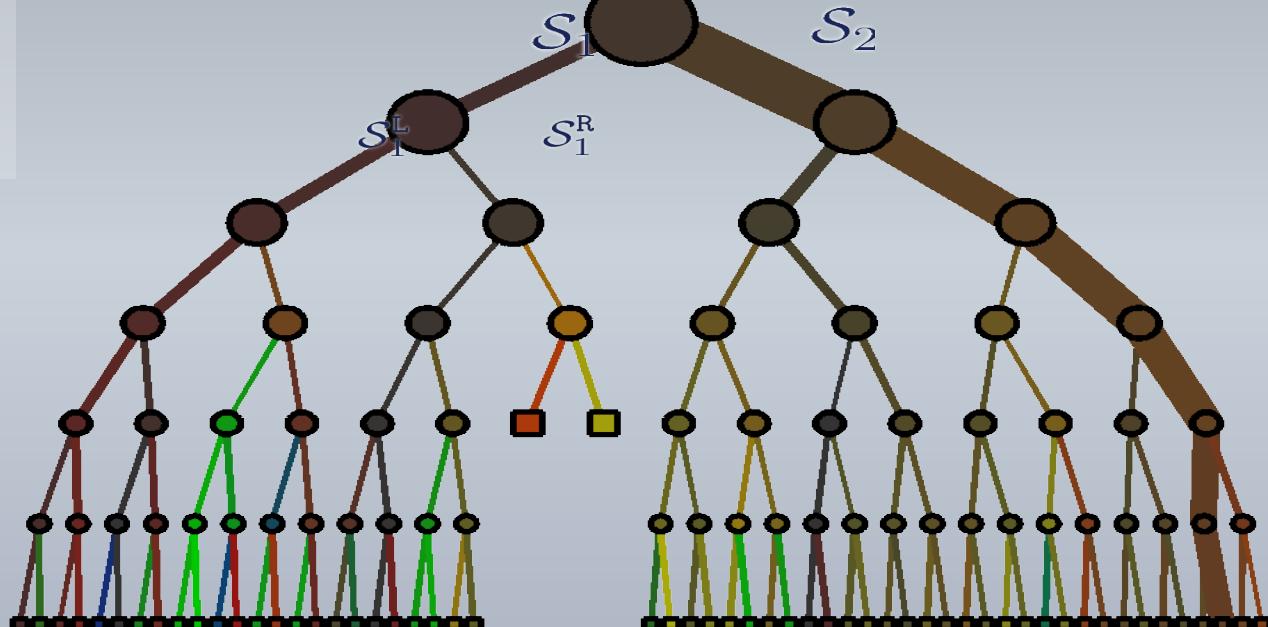
$$h(\mathbf{v}, \theta) \in \{\text{true, false}\}$$

$$\theta_j = \arg \max_{\theta \in \mathcal{T}_j} I$$

$$I = I(\mathcal{S}_j, \theta)$$

Input training data

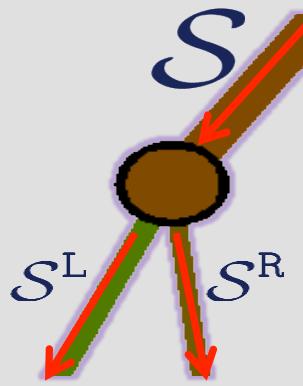
$$\mathcal{S}_0 = \{\mathbf{v}, c\}$$



Binary tree? Ternary?
How big a tree?

Training and information gain

(for categorical, non-parametric distributions)



Information gain

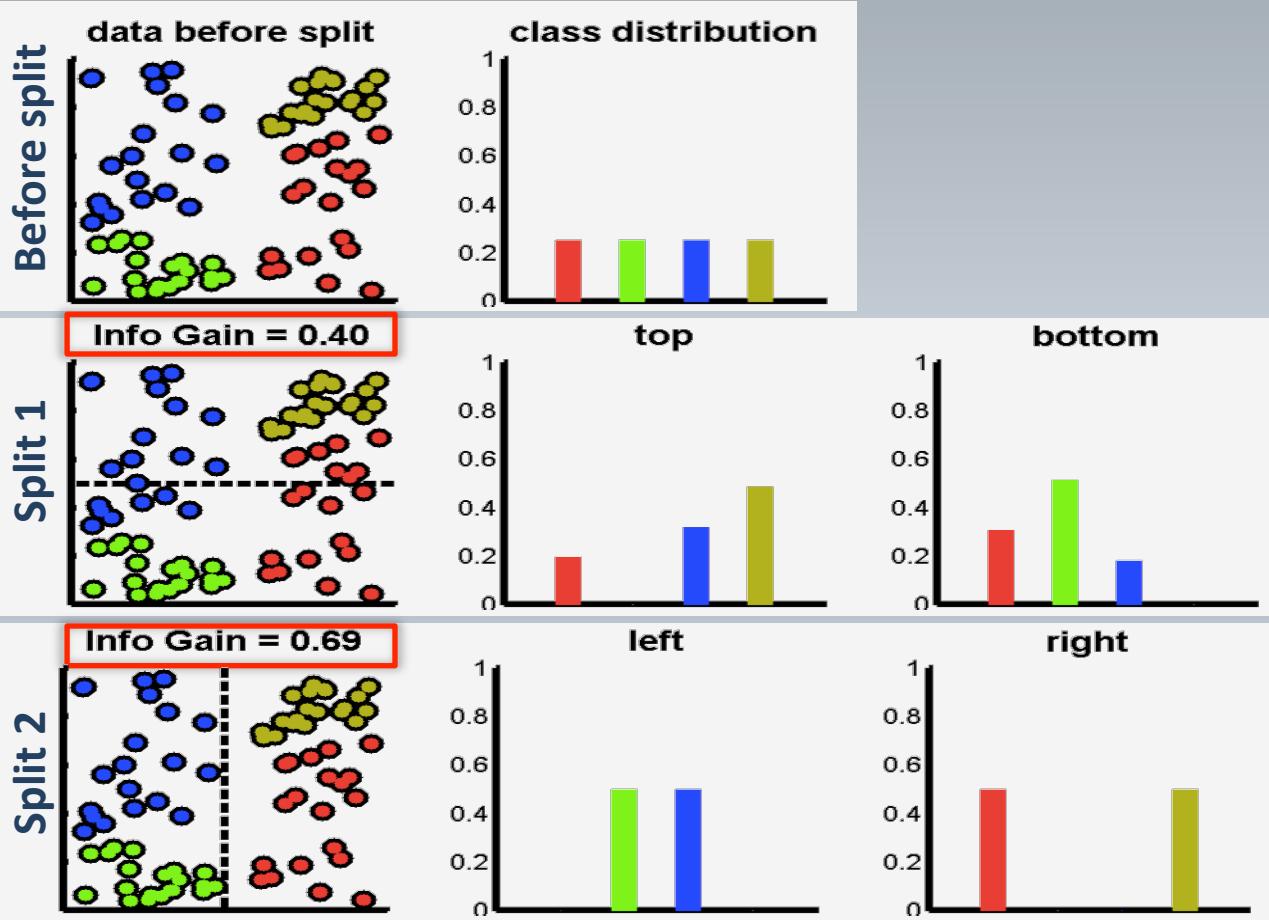
$$I(S, \theta) = H(S) - \sum_{i \in \{L, R\}} \frac{|S^i|}{|S|} H(S^i)$$

Shannon's entropy

$$H(S) = - \sum_{c \in \mathcal{C}} p(c) \log(p(c))$$

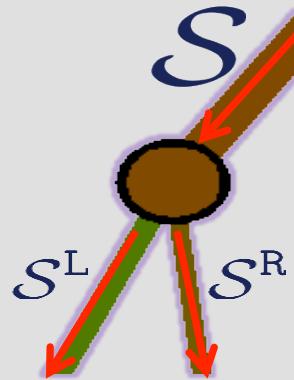
Node training

$$\theta = \arg \max_{\theta \in \mathcal{T}_j} I(S_j, \theta)$$



Training and information gain

(for continuous, parametric densities)



Information gain

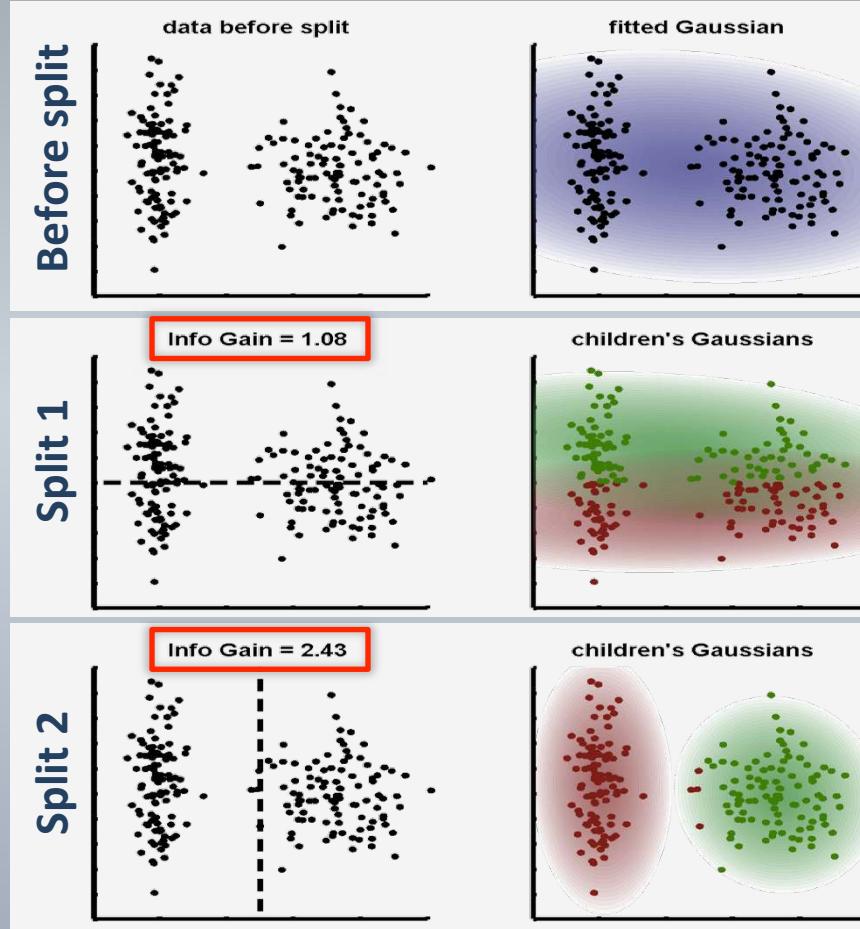
$$I(\mathcal{S}, \theta) = H(\mathcal{S}) - \sum_{i \in \{L,R\}} \frac{|\mathcal{S}^i|}{|\mathcal{S}|} H(\mathcal{S}^i)$$

Differential entropy of Gaussian

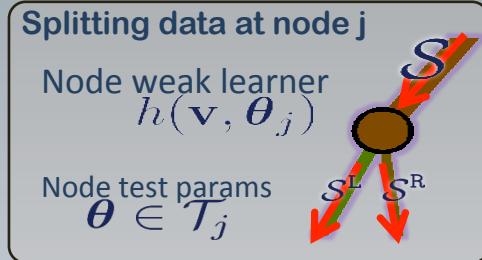
$$H(\mathcal{S}) = \frac{1}{2} \log ((2\pi e)^d |\Lambda(\mathcal{S})|)$$

Node training

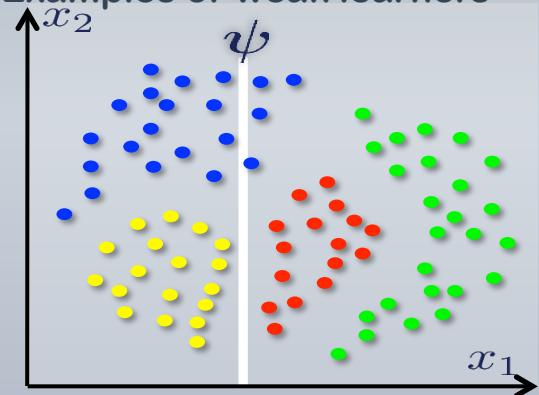
$$\theta = \arg \max_{\theta \in \mathcal{T}_j} I(\mathcal{S}_j, \theta)$$



The weak learner model



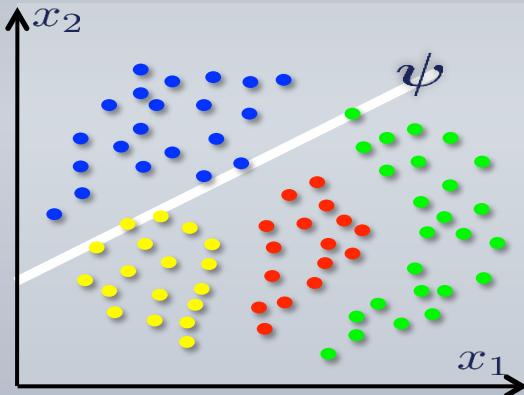
Examples of weak learners



Weak learner: axis aligned

$$h(\mathbf{v}, \boldsymbol{\theta}) = [\tau_1 > \phi(\mathbf{v}) \cdot \psi > \tau_2]$$

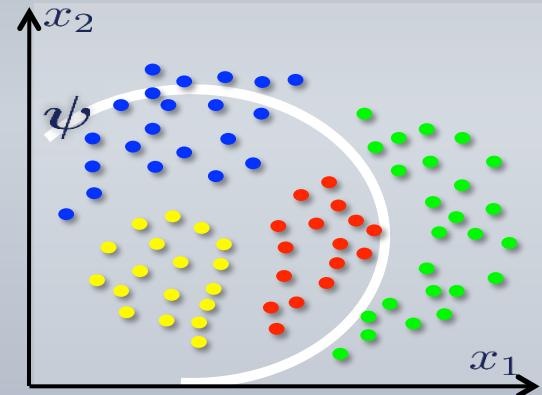
Feature response $\phi(\mathbf{v}) = (x_1 \ x_2 \ 1)^\top$
for 2D example.
With $\psi = (1 \ 0 \ \psi_3)$ or $\psi = (0 \ 1 \ \psi_3)$



Weak learner: oriented line

$$h(\mathbf{v}, \boldsymbol{\theta}) = [\tau_1 > \phi(\mathbf{v}) \cdot \psi > \tau_2]$$

Feature response $\phi(\mathbf{v}) = (x_1 \ x_2 \ 1)^\top$
for 2D example.
With $\psi \in \mathbb{R}^3$ a generic line in homog. coordinates.

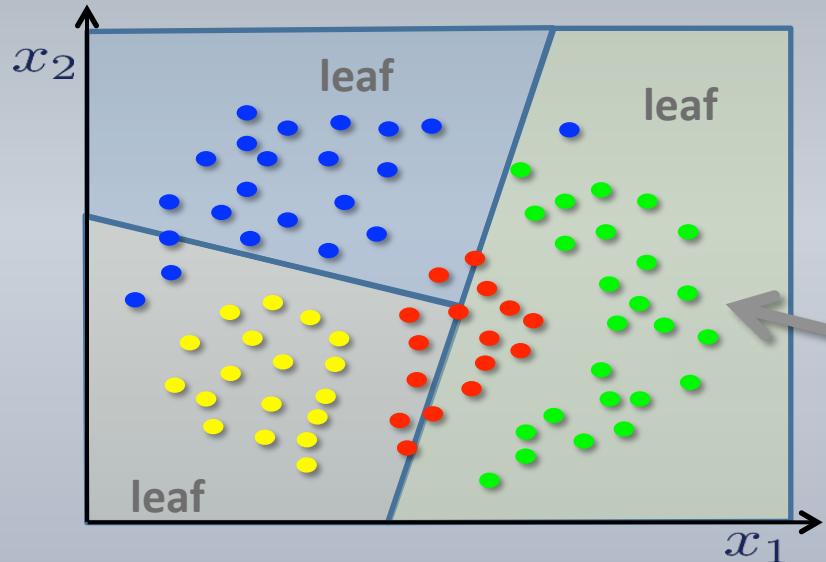


Weak learner: conic section

$$h(\mathbf{v}, \boldsymbol{\theta}) = [\tau_1 > \phi^\top(\mathbf{v}) \psi \phi(\mathbf{v}) > \tau_2]$$

Feature response $\phi(\mathbf{v}) = (x_1 \ x_2 \ 1)^\top$
for 2D example.
With $\psi \in \mathbb{R}^{3 \times 3}$ a matrix representing a conic.

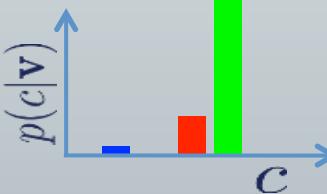
The prediction model



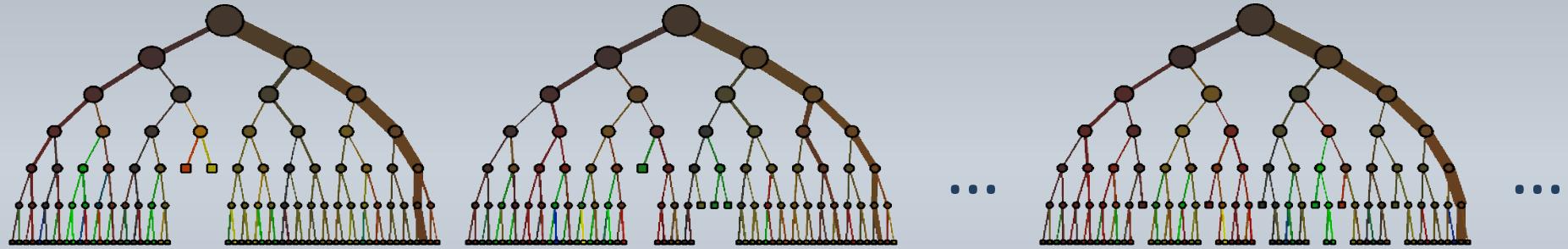
What do we do at the leaf?



Prediction model: probabilistic



Decision forest training (off-line)



How many trees?
How different?
How to fuse their outputs?

Decision forest model: the randomness model

1) Bagging (randomizing the training set)

\mathcal{S}_0

The full training set

$\mathcal{S}_0^t \subset \mathcal{S}_0$

The randomly sampled subset of training data made available for the tree t

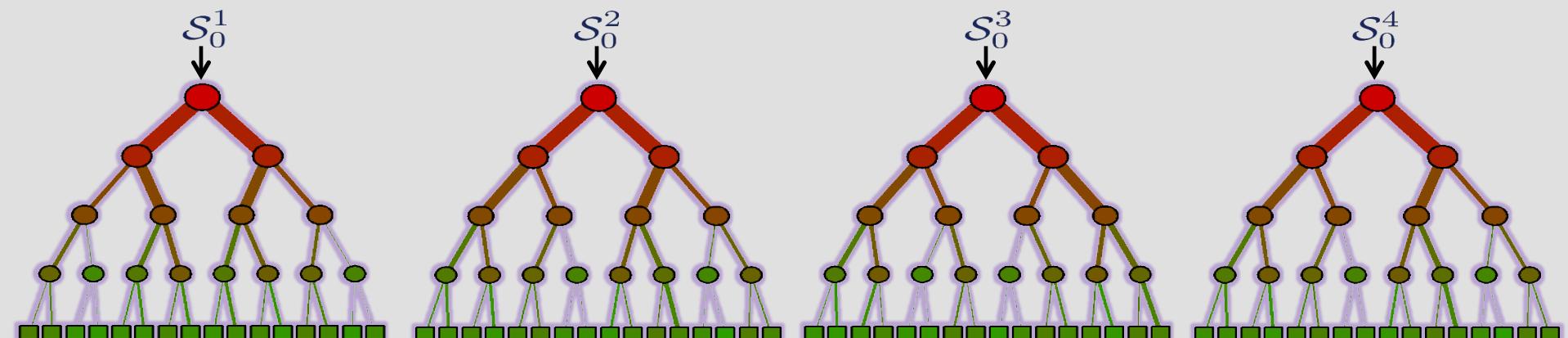
Forest training

\mathcal{S}_0^1

\mathcal{S}_0^2

\mathcal{S}_0^3

\mathcal{S}_0^4



Efficient training

Decision forest model: the randomness model

2) Randomized node optimization (RNO)

 \mathcal{T}

The full set of all possible node test parameters

 $\mathcal{T}_j \subset \mathcal{T}$

For each node the set of randomly sampled features

 $\rho = |\mathcal{T}_j|$

Randomness control parameter.

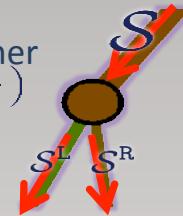
For $\rho = |\mathcal{T}|$ no randomness and maximum tree correlation.

For $\rho = 1$ max randomness and minimum tree correlation.

Node training

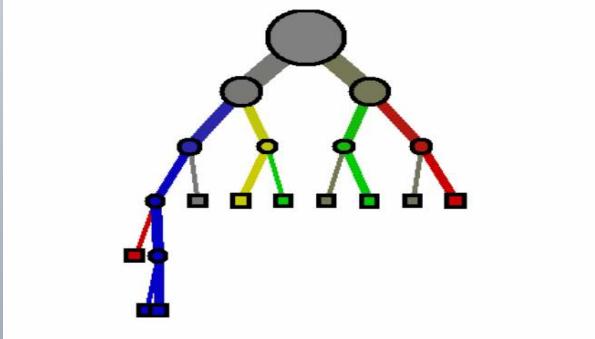
Node weak learner
 $h(\mathbf{v}, \boldsymbol{\theta}_j)$

Node test params
 $\boldsymbol{\theta} \in \mathcal{T}_j$

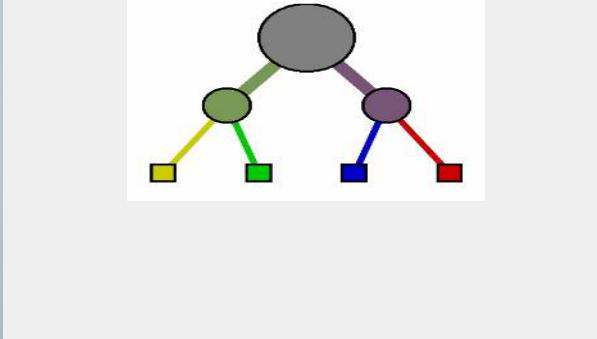


The effect of ρ

Small value of ρ ; little tree correlation.

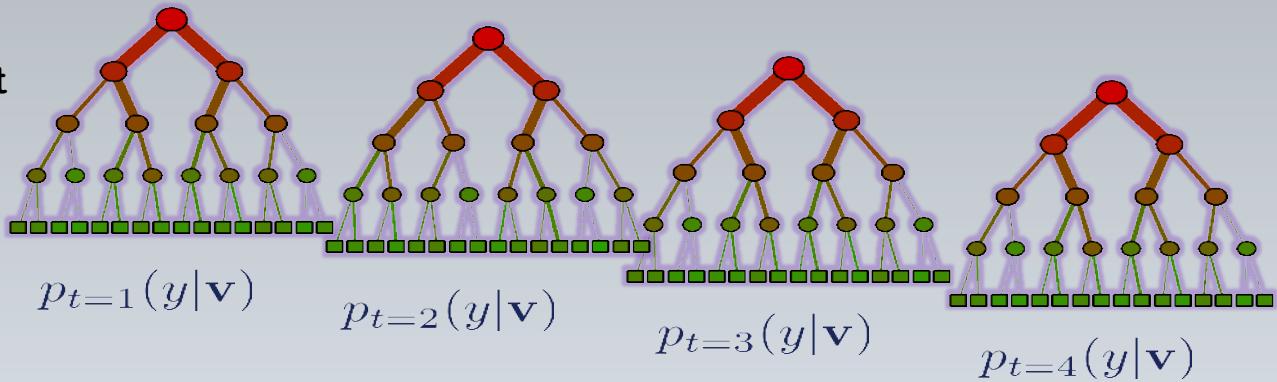


Large value of ρ ; large tree correlation.

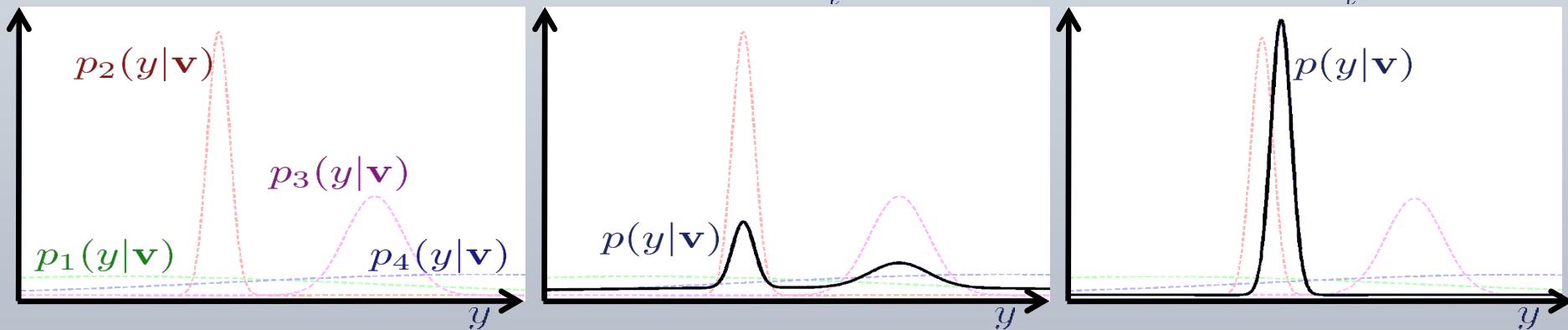


Decision forest model: the ensemble model

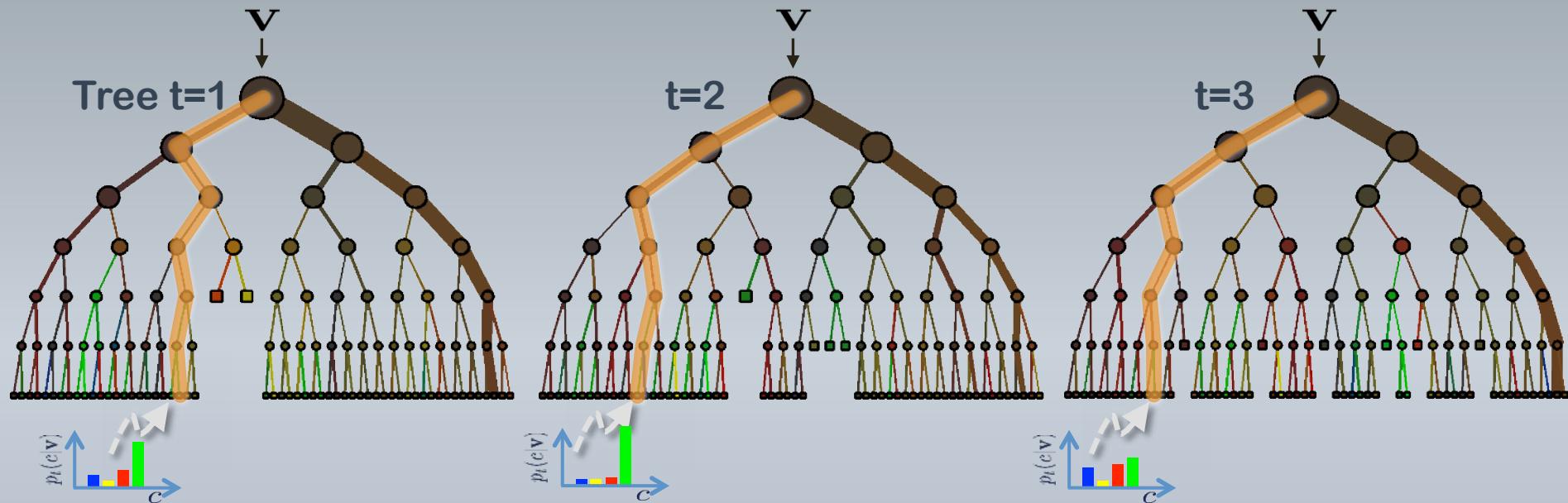
An example forest to predict continuous variables



$$p(y|\mathbf{v}) = \frac{1}{T} \sum_t^T p_t(y|\mathbf{v}) \quad p(y|\mathbf{v}) = \frac{1}{Z} \prod_t^T p_t(y|\mathbf{v})$$

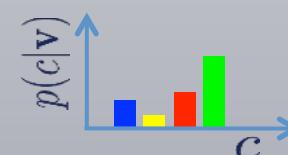


Classification forest: the ensemble model



The ensemble model

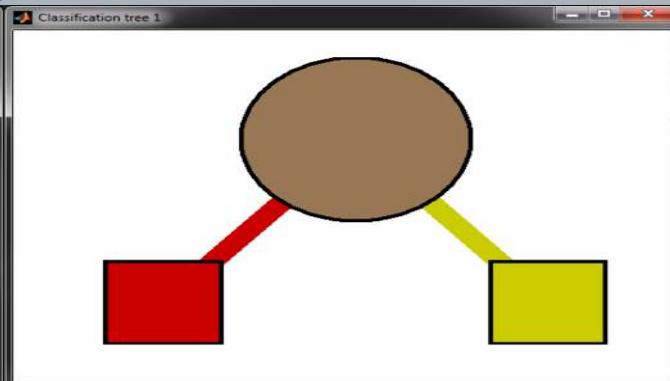
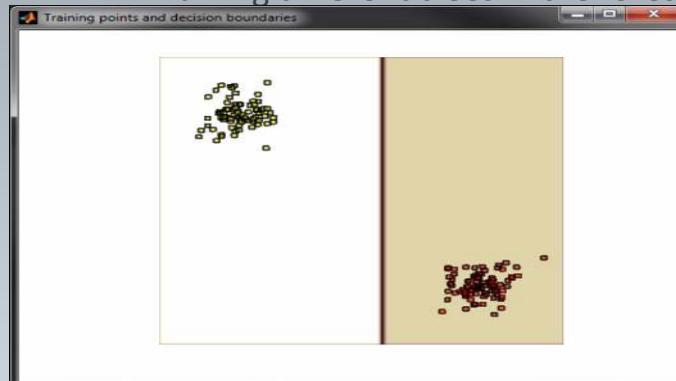
Forest output probability
$$p(c|\mathbf{v}) = \frac{1}{T} \sum_t^T p_t(c|\mathbf{v})$$



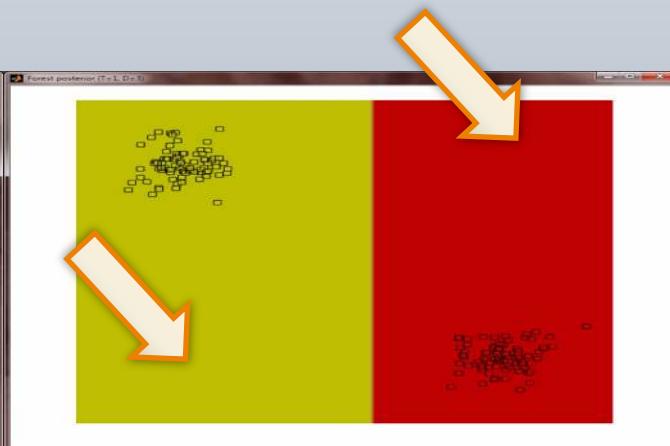
Classification forest: effect of the weak learner model



Training different trees in the forest



Testing different trees in the forest



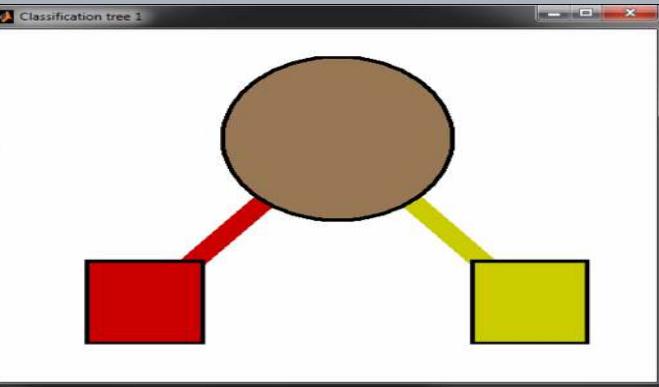
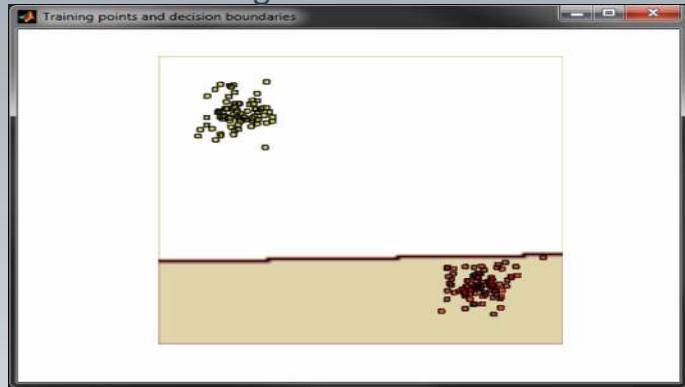
Three concepts to keep in mind:

- “Accuracy of prediction”
- “Quality of confidence”
- “Generalization”

Classification forest: effect of the weak learner model



Training different trees in the forest



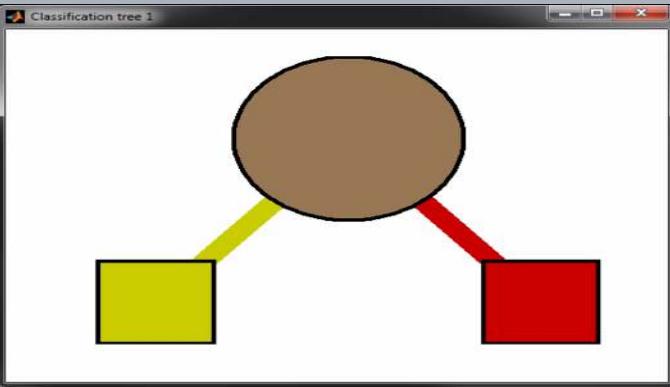
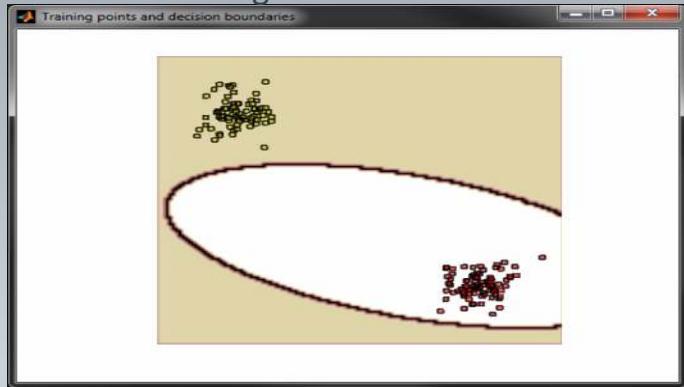
Testing different trees in the forest



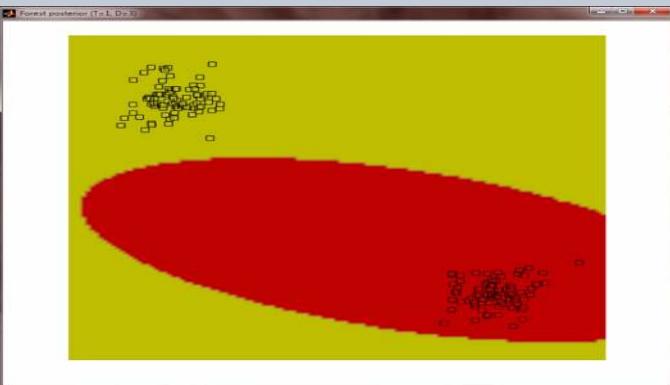
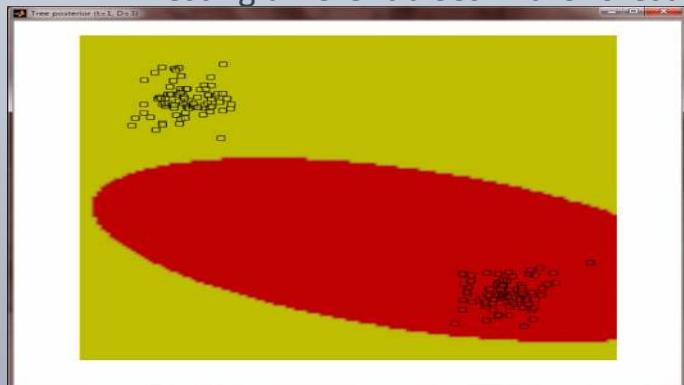
Classification forest: effect of the weak learner model



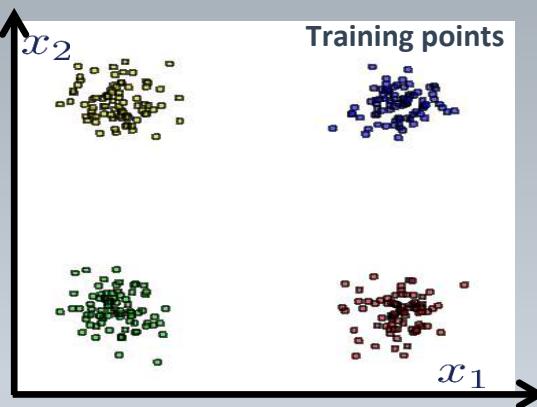
Training different trees in the forest



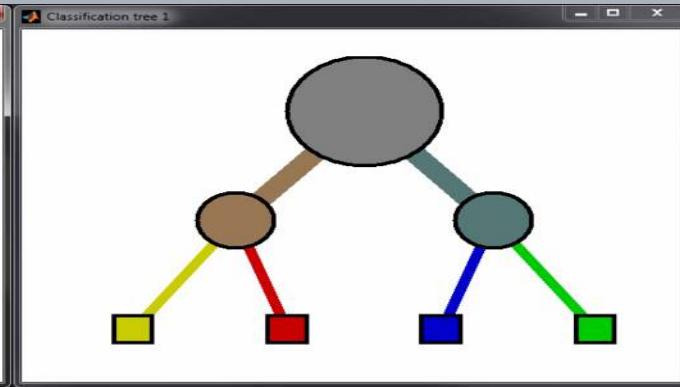
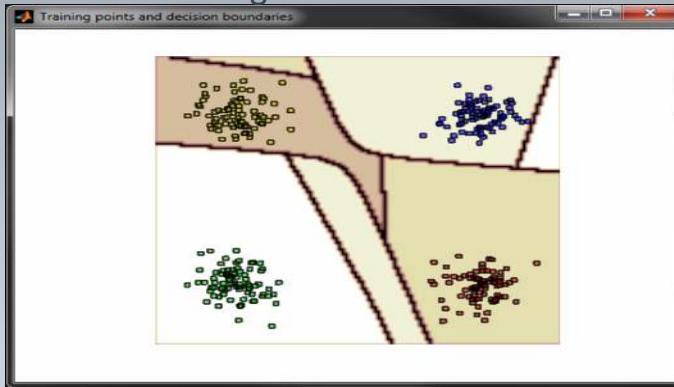
Testing different trees in the forest



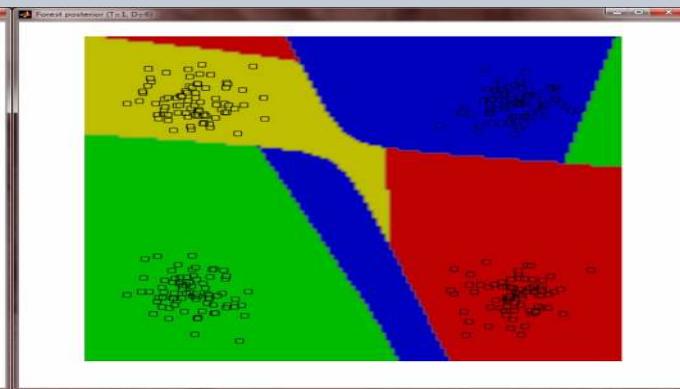
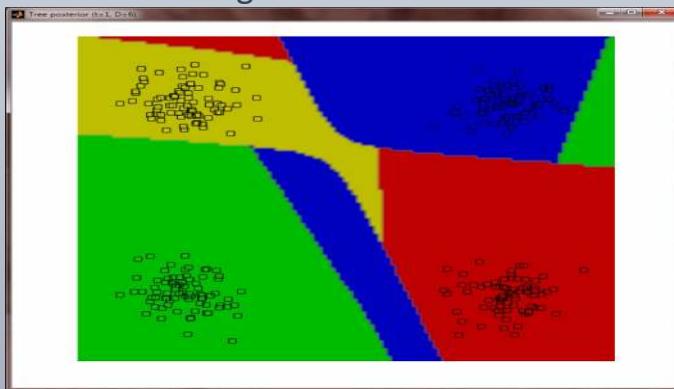
Classification forest: with >2 classes

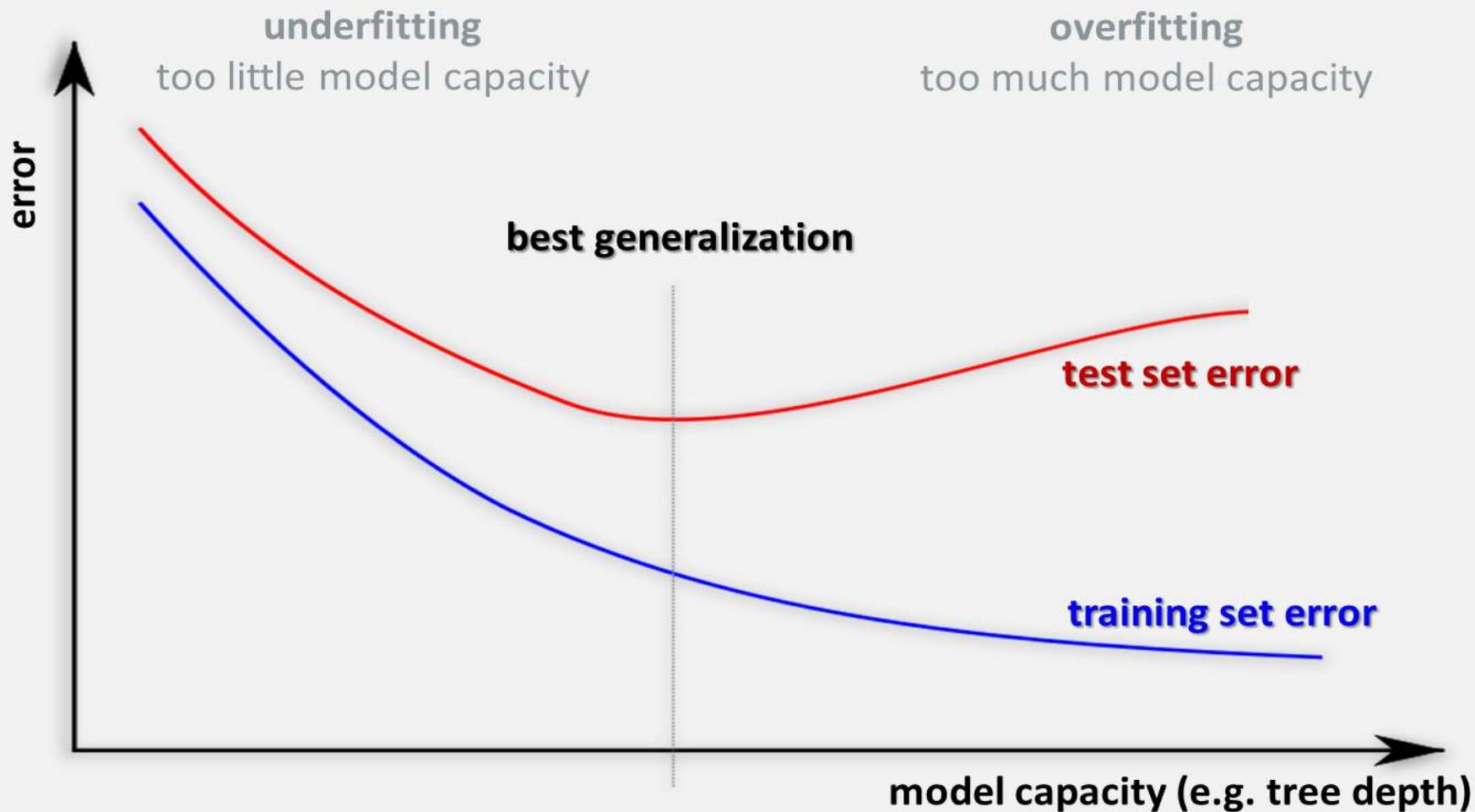


Training different trees in the forest

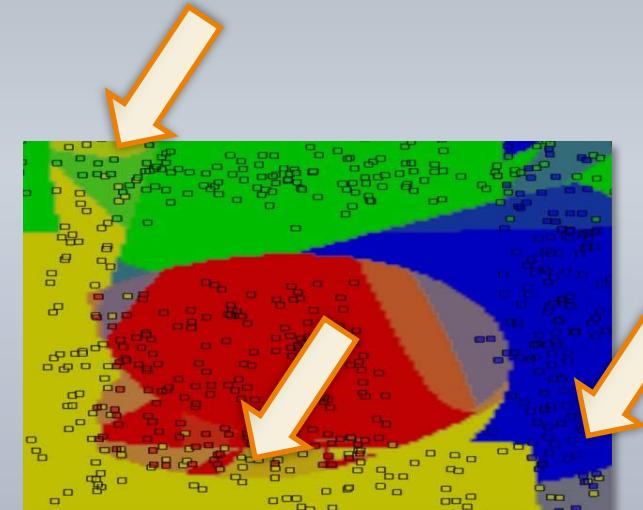
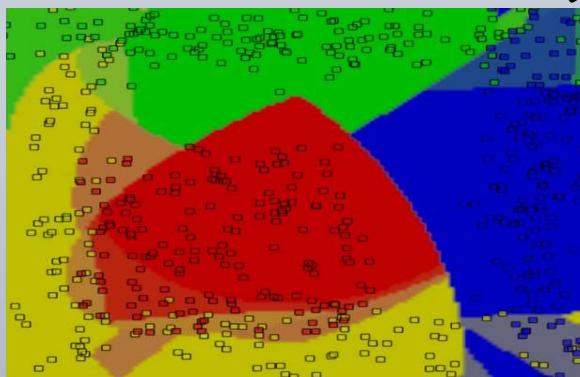
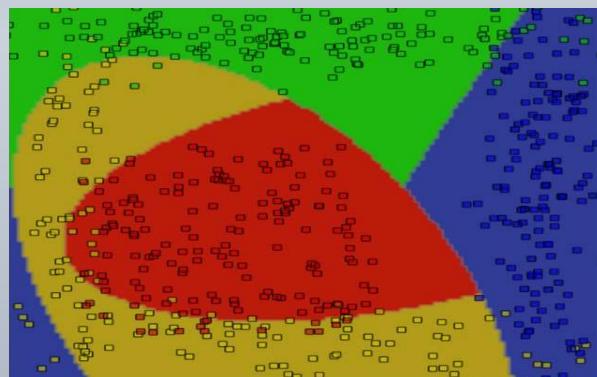
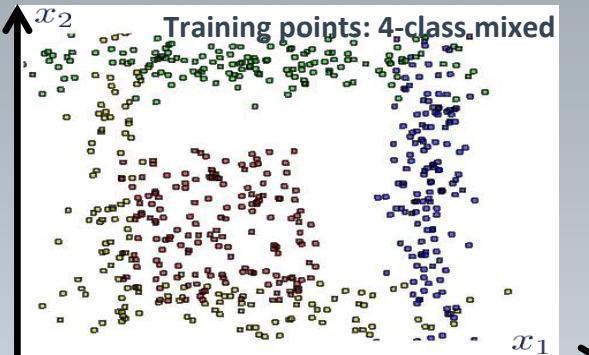


Testing different trees in the forest





Classification forest: effect of tree depth



max tree depth, D
overfitting

Predictor model = prob.

underfitting

(3 videos in this page)

Real-Time Human Pose Recognition in Parts from Single Depth Images

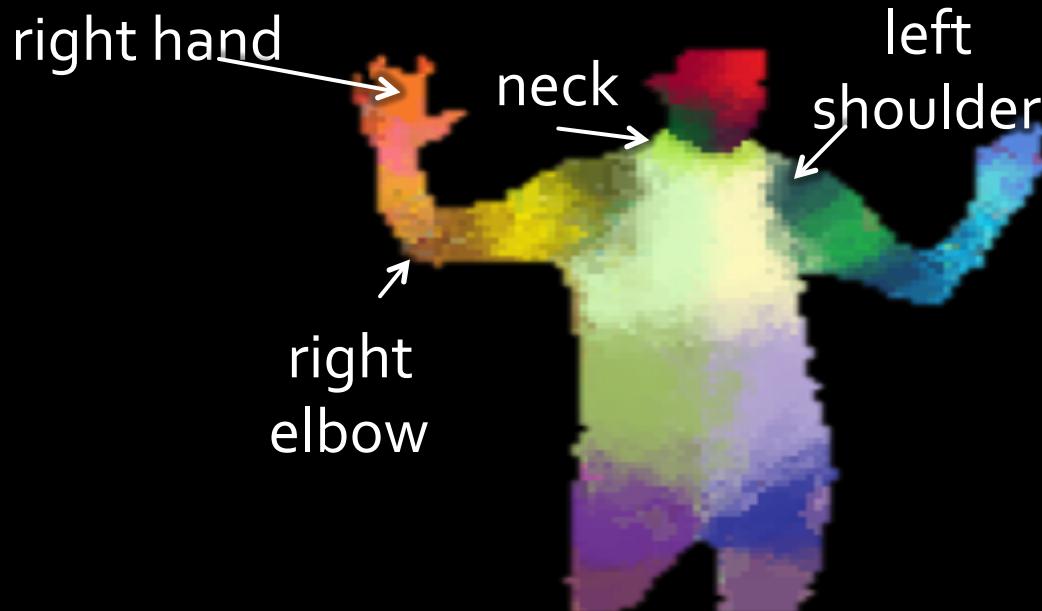
Jamie Shotton, Andrew Fitzgibbon, Mat Cook,
Toby Sharp, Mark Finocchio, Richard Moore,
Alex Kipman, Andrew Blake

CVPR 2011

Microsoft[®]
Research



Body part recognition

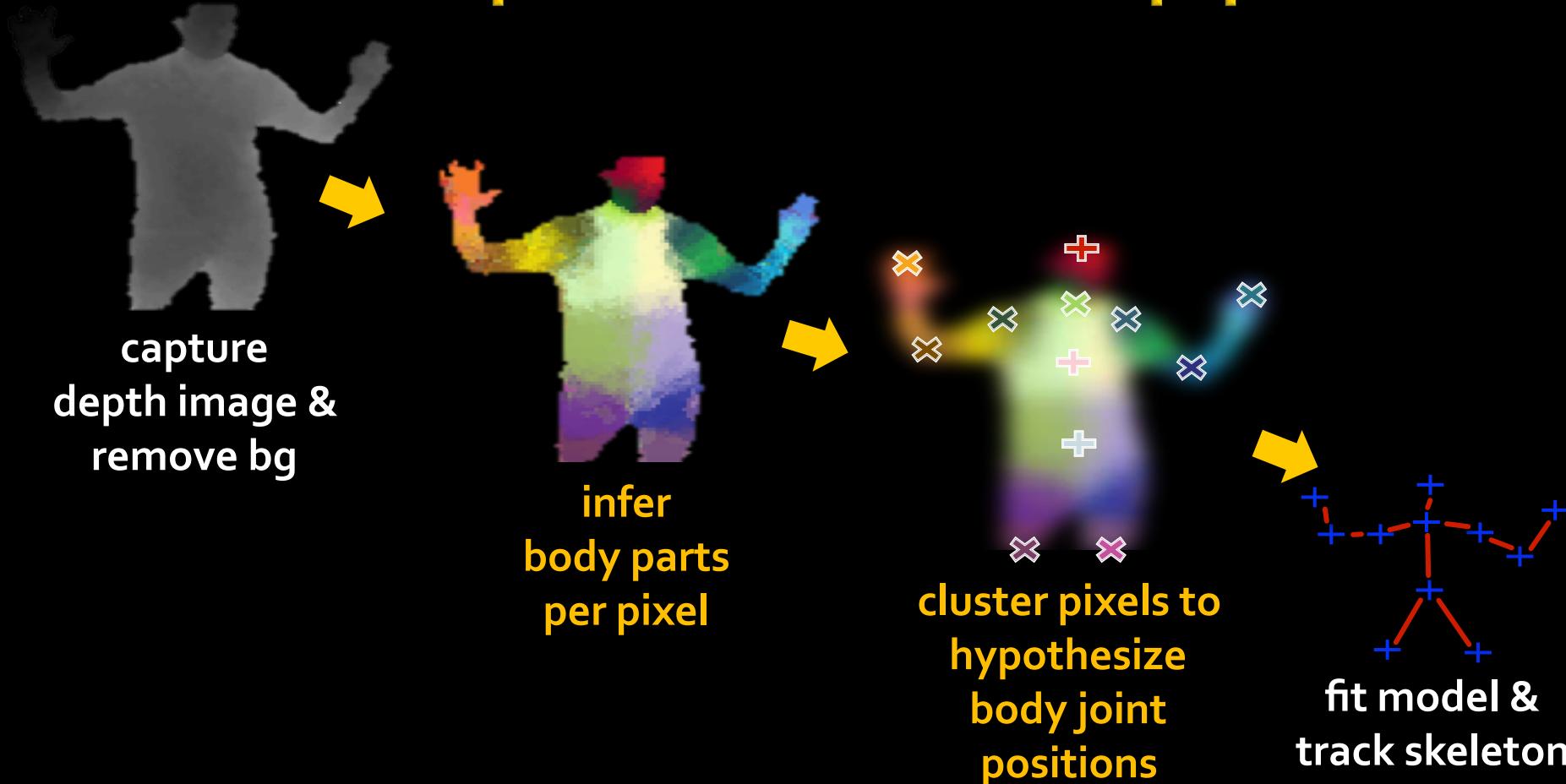


Body part recognition

- No temporal information
 - frame-by-frame
- Local pose estimate of parts
 - each pixel & each body joint treated independently
- Very fast
 - simple depth image features
 - parallel decision forest classifier



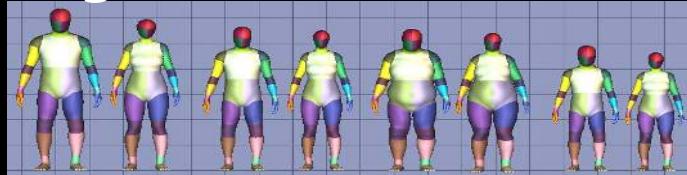
The Kinect pose estimation pipeline



Synthetic training data

Record mocap
500k frames
distilled to 100k poses

Retarget to several models



Render (depth, body parts) pairs



Train invariance to:



Synthetic vs. real data



synthetic
(train & test)



real
(test)

Fast depth image features

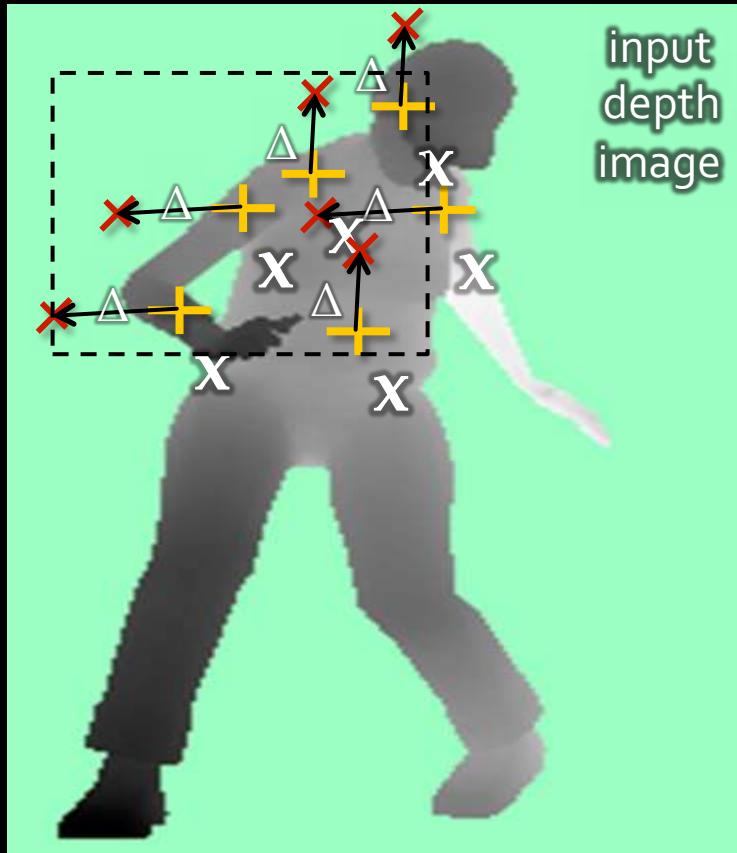
- Depth comparisons
 - very fast to compute

feature response $f(I, x) = \underbrace{d_I(x)}_{\text{image coordinate}} - \underbrace{d_I(x + \Delta)}_{\text{offset depth}}$

$$\Delta = \underbrace{v/d_I(x)}$$

scales inversely with depth

Background pixels
 $d = \text{large constant}$



Depth of trees

input depth



ground truth parts



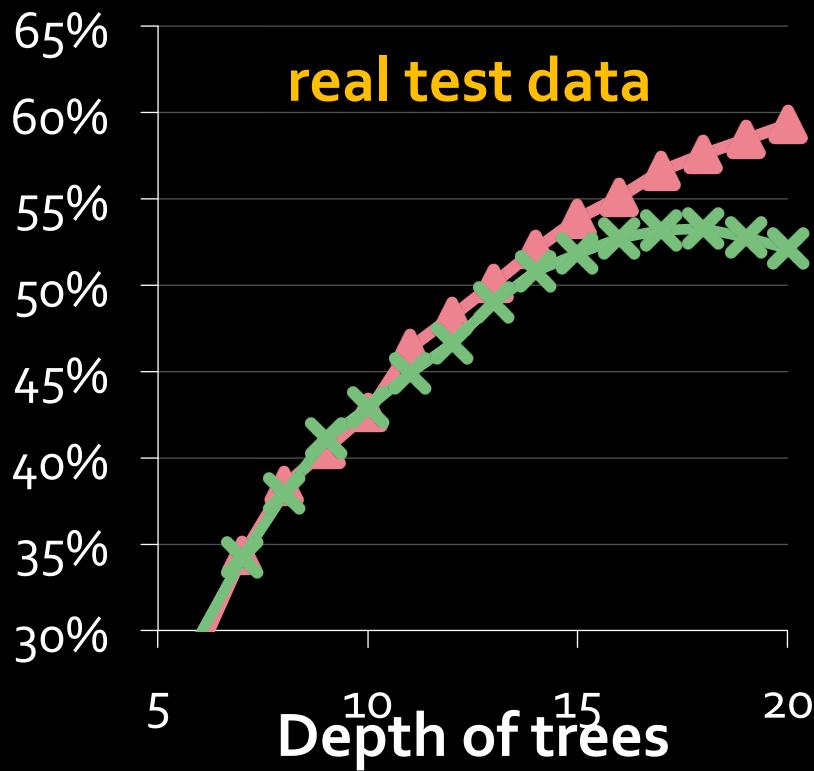
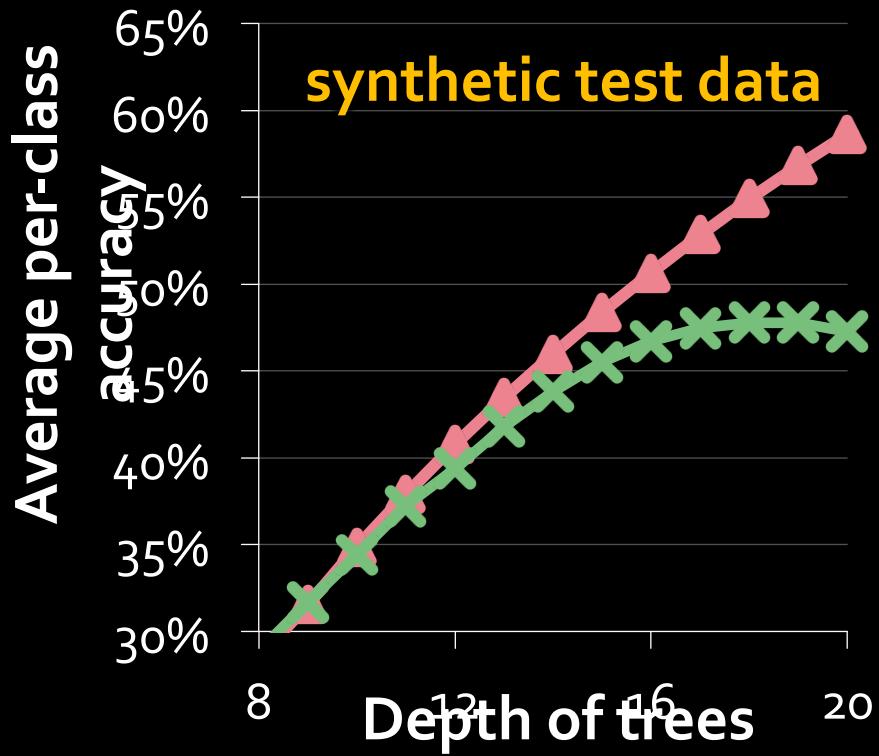
inferred parts (soft)



depth 18

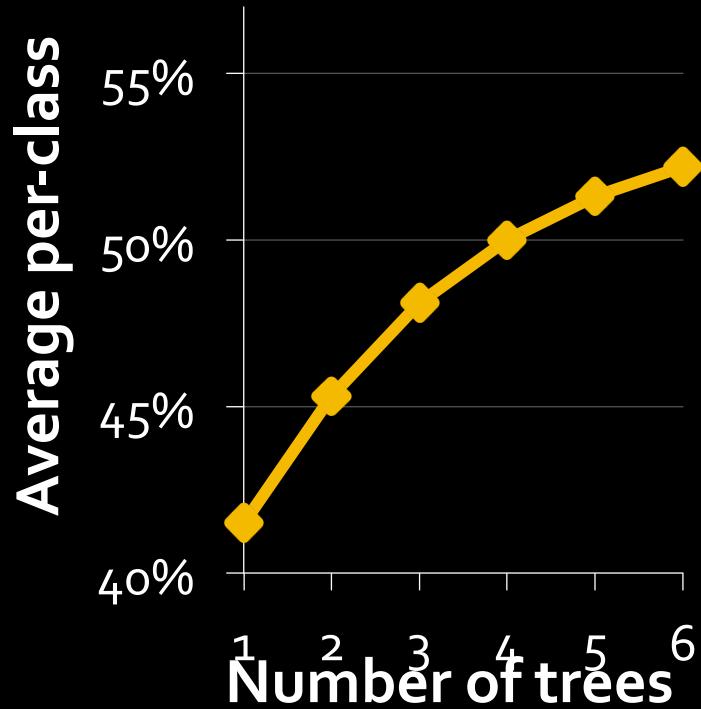


Depth of trees



—♦— 900k training images
—×— 15k training images

Number of trees



ground truth



inferred body parts (most likely)
1 tree 3 trees 6 trees



input depth



inferred body parts



front view



side view

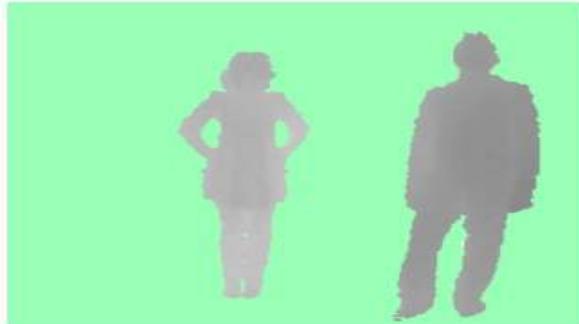


top view

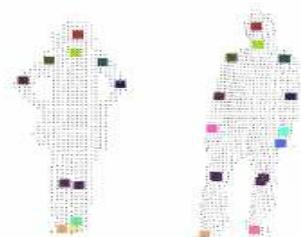
inferred joint positions

no tracking or smoothing

input depth



inferred body parts



front view



side view



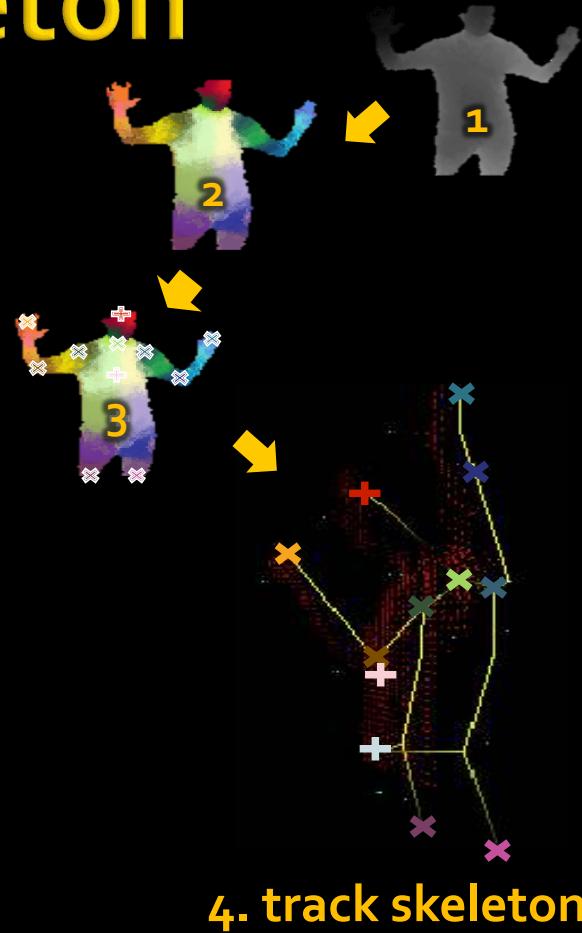
top view

inferred joint positions

no tracking or smoothing

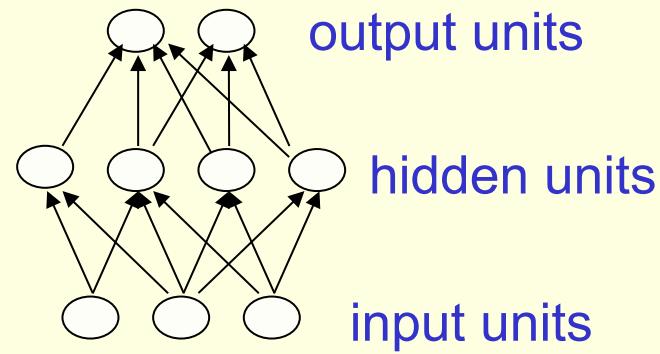
From proposals to skeleton

- Use...
 - 3D joint hypotheses
 - kinematic constraints
 - temporal coherence
- ... to give
 - full skeleton
 - higher accuracy
 - invisible joints
 - multi-player



Feed-forward neural networks

- These are the most common type of neural network in practice
 - The first layer is the input and the last layer is the output.
 - If there is more than one hidden layer, we call them “deep” neural networks.
 - Hidden layers learn complex features, the outputs are learned in terms of those features.

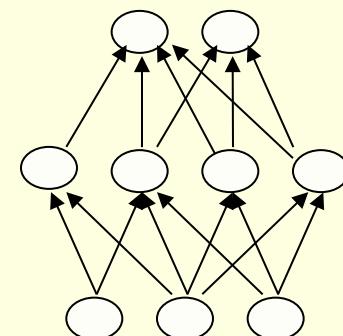


Linear neurons

- These are simple but computationally limited

$$y = b + \sum_i x_i w_i$$

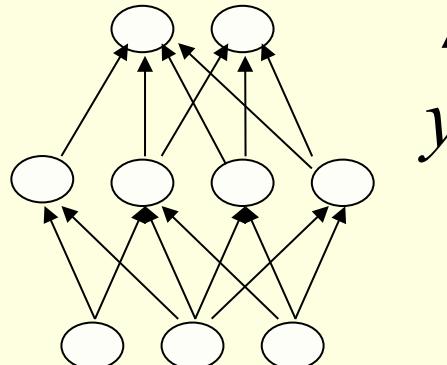
bias
output
 i^{th} input
index over
input connections
weight on
 i^{th} input



Sigmoid neurons

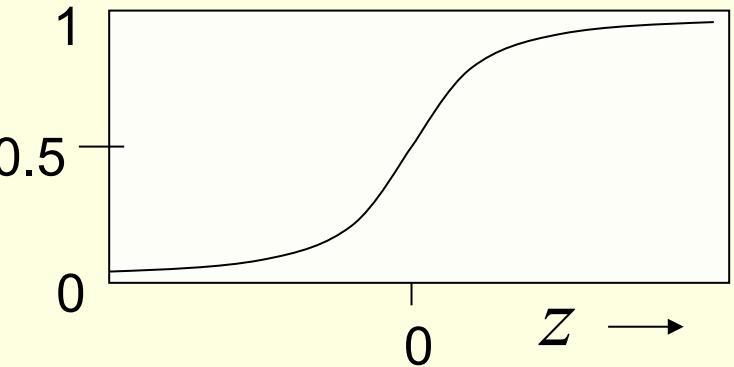
- These give a real-valued output that is a smooth and bounded function of their total input.

– They have nice derivatives which make learning easy



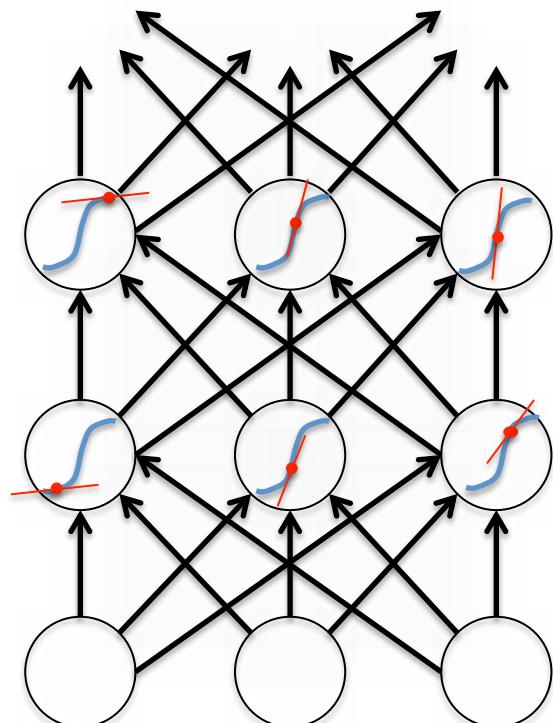
$$z = b + \sum_i x_i w_i \quad y = \frac{1}{1 + e^{-z}}$$

$$\frac{\partial z}{\partial w_i} = x_i \quad \frac{\partial z}{\partial x_i} = w_i \quad \frac{dy}{dz} = y(1-y)$$



Finding weights with backpropagation

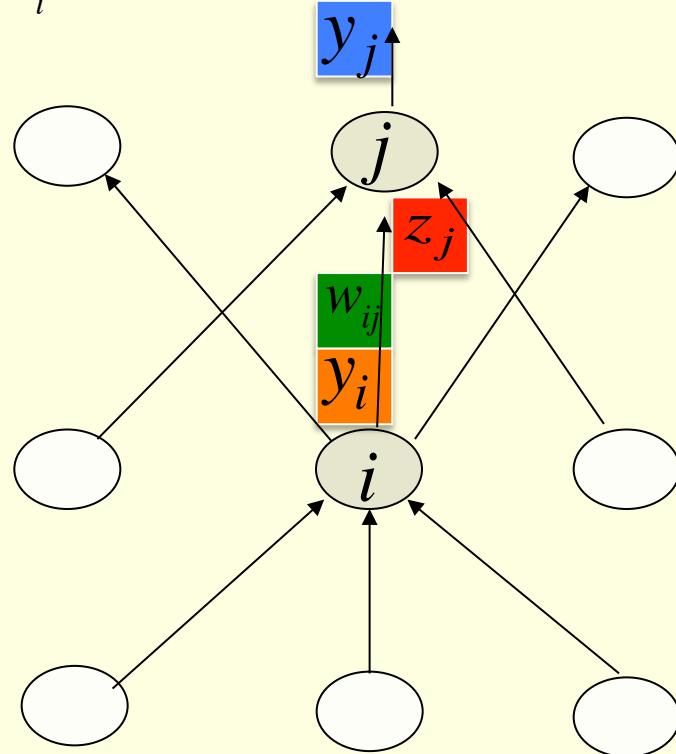
- There is a big difference between the forward and backward passes.
- In the forward pass we use squashing functions to prevent the activity vectors from exploding.
- The backward pass, is completely **linear**.
 - The forward pass determines the slope of the **linear** function used for backpropagating through each neuron.



$$z_j = \sum_i y_i w_{ij}$$

Backpropagating dE/dy

$$E_j = \frac{1}{2} (y_j - t_j)^2$$



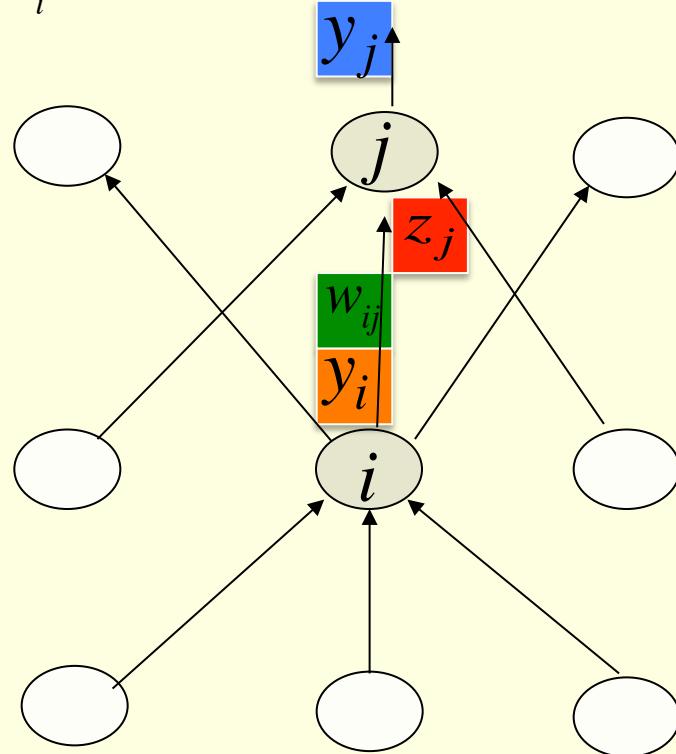
- Find squared error
- Propagate error to the layer below
- Compute error derivative w.r.t. weights
- Repeat

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

$$z_j = \sum_i y_i w_{ij}$$

Backpropagating dE/dy

$$E_j = \frac{1}{2} (y_j - t_j)^2$$



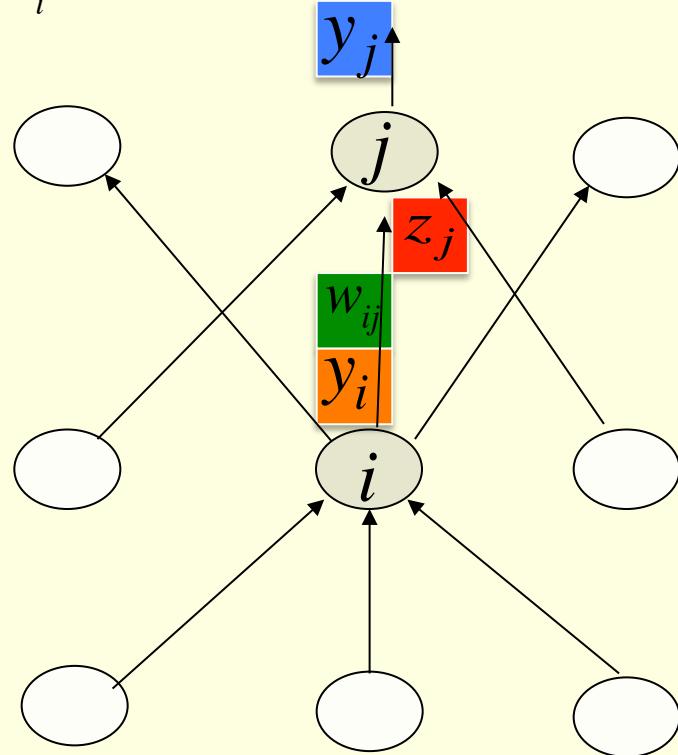
$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j} \frac{\partial E}{\partial y_j}$$

Propagate error across
non-linearity

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

$$z_j = \sum_i y_i w_{ij}$$

Backpropagating dE/dy



$$E_j = \frac{1}{2} (y_j - t_j)^2$$
$$\frac{\partial E}{\partial y_j} = y_j - t_j$$

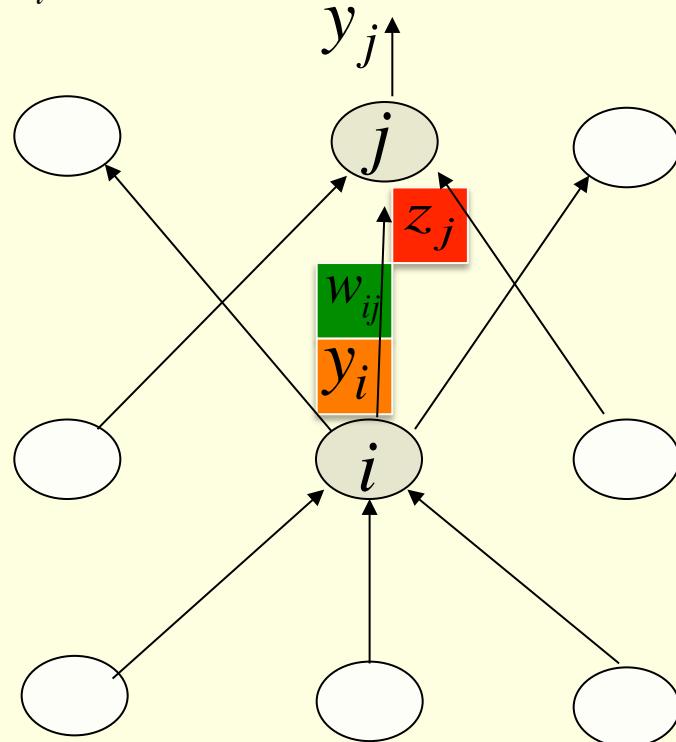
$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j} \frac{\partial E}{\partial y_j} = y_j (1 - y_j) \frac{\partial E}{\partial y_j}$$

Propagate error across
non-linearity

$$y = \frac{1}{1 + e^{-z}}$$
$$\frac{dy}{dz} = y (1 - y)$$

$$z_j = \sum_i y_i w_{ij}$$

Backpropagating dE/dy



$$E_j = \frac{1}{2} (y_j - t_j)^2$$
$$\frac{\partial E}{\partial y_j} = y_j - t_j$$

$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j} \frac{\partial E}{\partial y_j} = y_j (1 - y_j) \frac{\partial E}{\partial y_j}$$

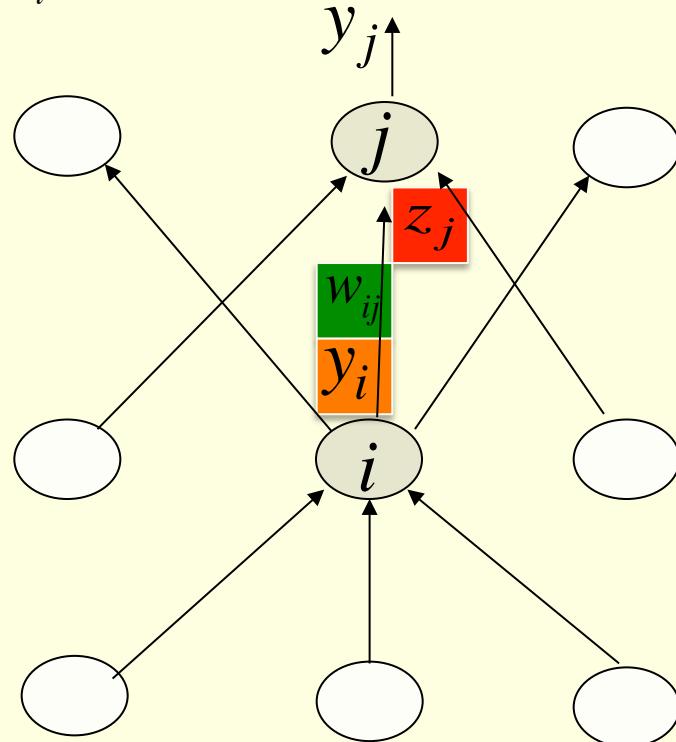
$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i} \frac{\partial E}{\partial z_j}$$

Propagate error to the next activation across connections

$$y = \frac{1}{1 + e^{-z}} \quad \frac{dy}{dz} = y (1 - y)$$

$$z_j = \sum_i y_i w_{ij}$$

Backpropagating dE/dy



$$y = \frac{1}{1 + e^{-z}} \quad \frac{dy}{dz} = y(1 - y)$$

$$E_j = \frac{1}{2} (y_j - t_j)^2$$

$$\frac{\partial E}{\partial y_j} = y_j - t_j$$

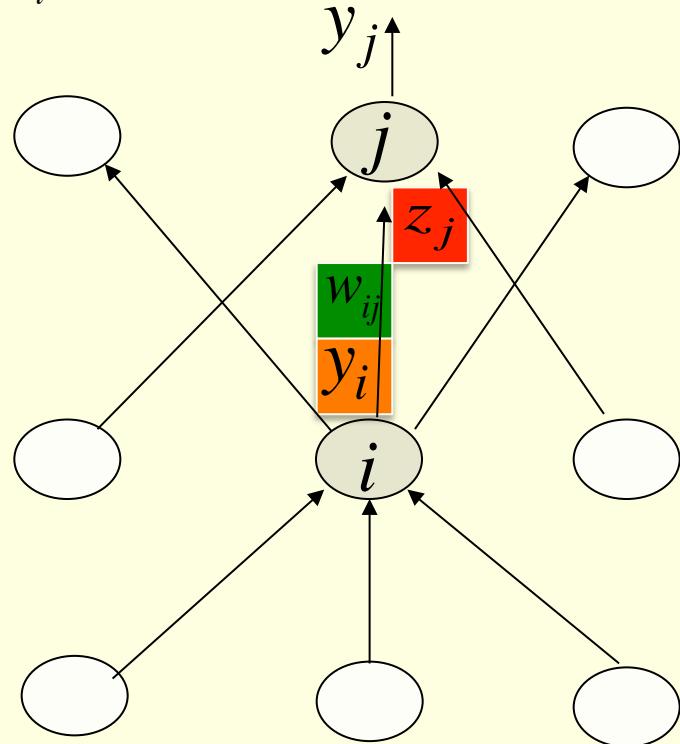
$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j} \frac{\partial E}{\partial y_j} = y_j (1 - y_j) \frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i} \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \frac{\partial E}{\partial z_j}$$

Propagate error to the next activation across connections

$$z_j = \sum_i y_i w_{ij}$$

Backpropagating dE/dy



$$E_j = \frac{1}{2} (y_j - t_j)^2$$

$$\frac{\partial E}{\partial y_j} = y_j - t_j$$

$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j} \frac{\partial E}{\partial y_j} = y_j (1 - y_j) \frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i} \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \frac{\partial E}{\partial z_j}$$

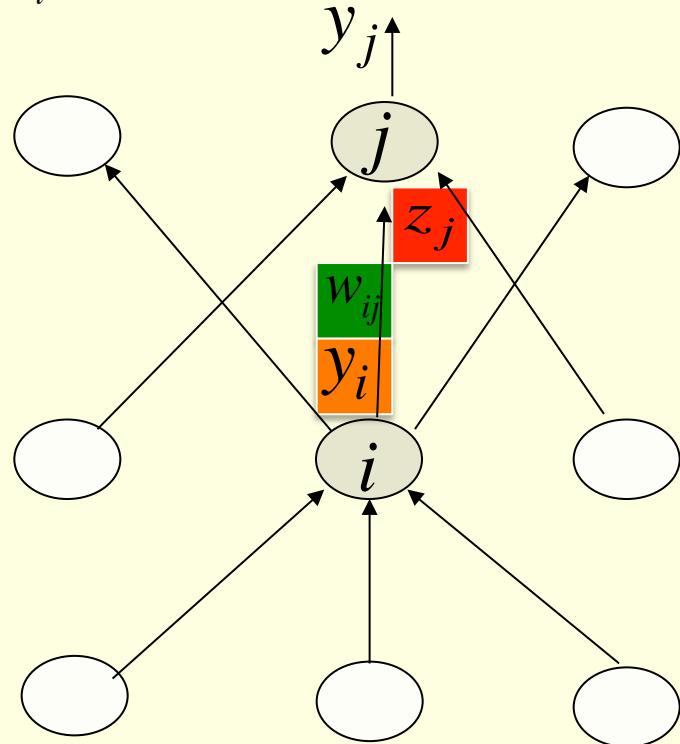
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j}$$

Error gradient w.r.t. weights

$$y = \frac{1}{1 + e^{-z}} \quad \frac{dy}{dz} = y(1 - y)$$

$$z_j = \sum_i y_i w_{ij}$$

Backpropagating dE/dy



$$E_j = \frac{1}{2} (y_j - t_j)^2$$

$$\frac{\partial E}{\partial y_j} = y_j - t_j$$

$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j} \frac{\partial E}{\partial y_j} = y_j (1 - y_j) \frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i} \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \frac{\partial E}{\partial z_j}$$

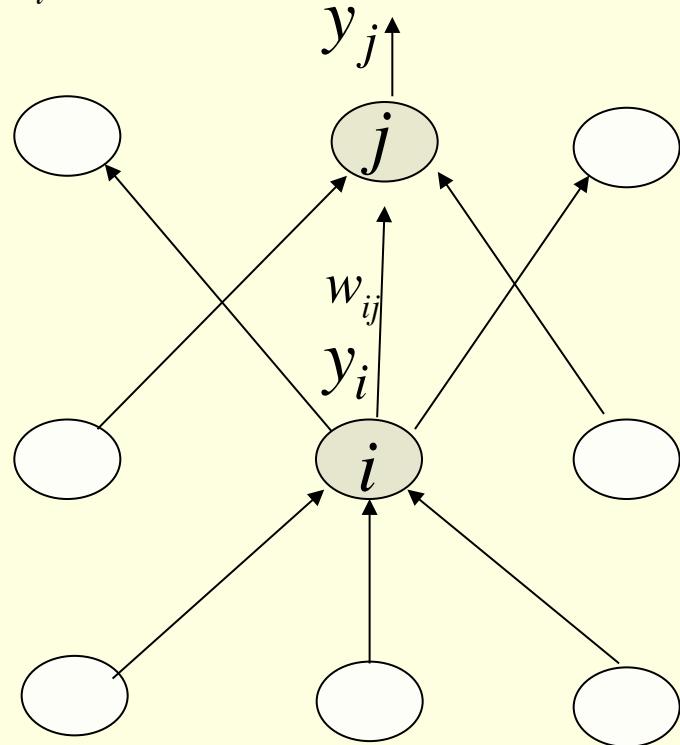
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i \frac{\partial E}{\partial z_j}$$

Error gradient w.r.t. weights

$$y = \frac{1}{1 + e^{-z}} \quad \frac{dy}{dz} = y(1 - y)$$

$$z_j = \sum_i y_i w_{ij}$$

Backpropagating dE/dy



$$y = \frac{1}{1 + e^{-z}} \quad \frac{dy}{dz} = y(1 - y)$$

$$E_j = \frac{1}{2}(y_j - t_j)^2$$

$$\frac{\partial E}{\partial y_j} = y_j - t_j$$

$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j} \frac{\partial E}{\partial y_j} = y_j(1 - y_j) \frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i} \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \frac{\partial E}{\partial z_j}$$

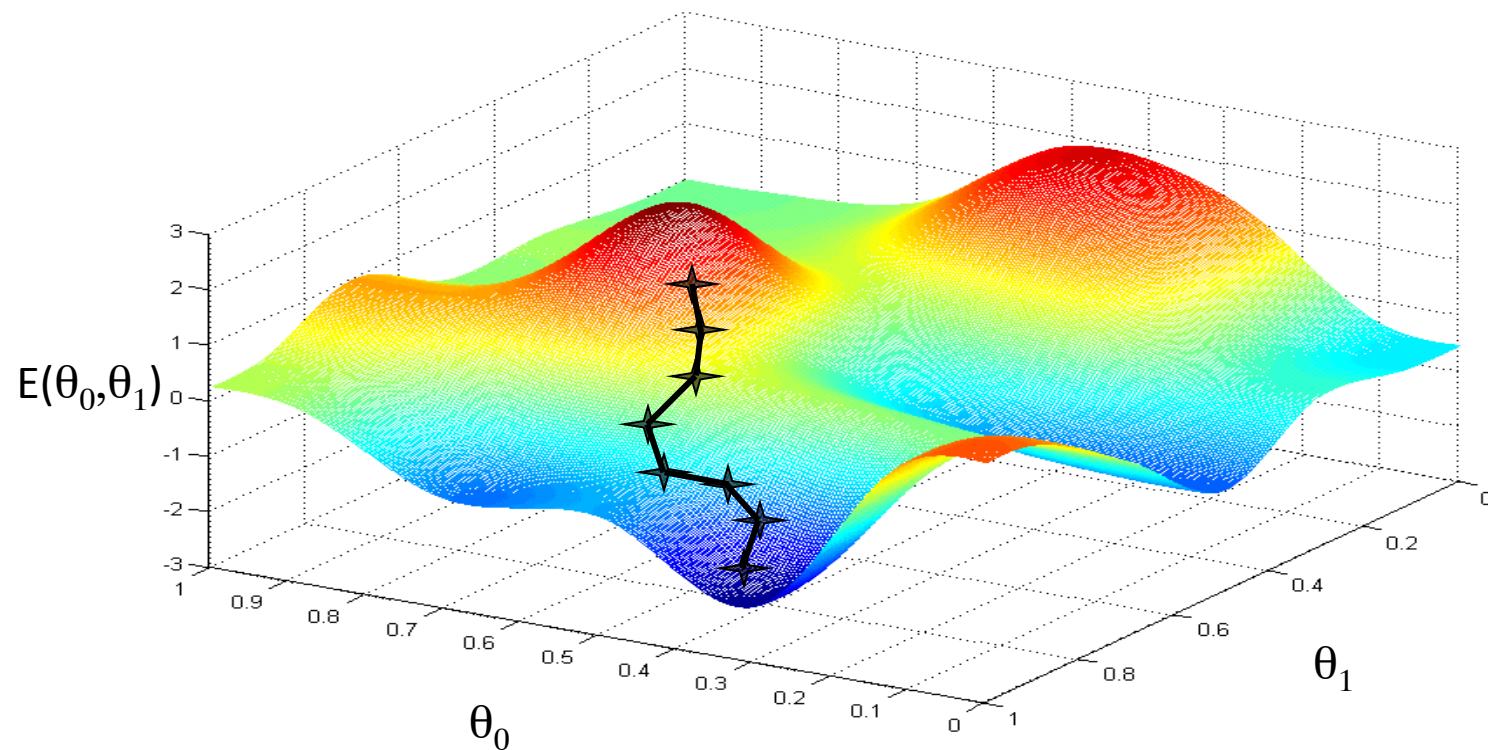
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i \frac{\partial E}{\partial z_j}$$

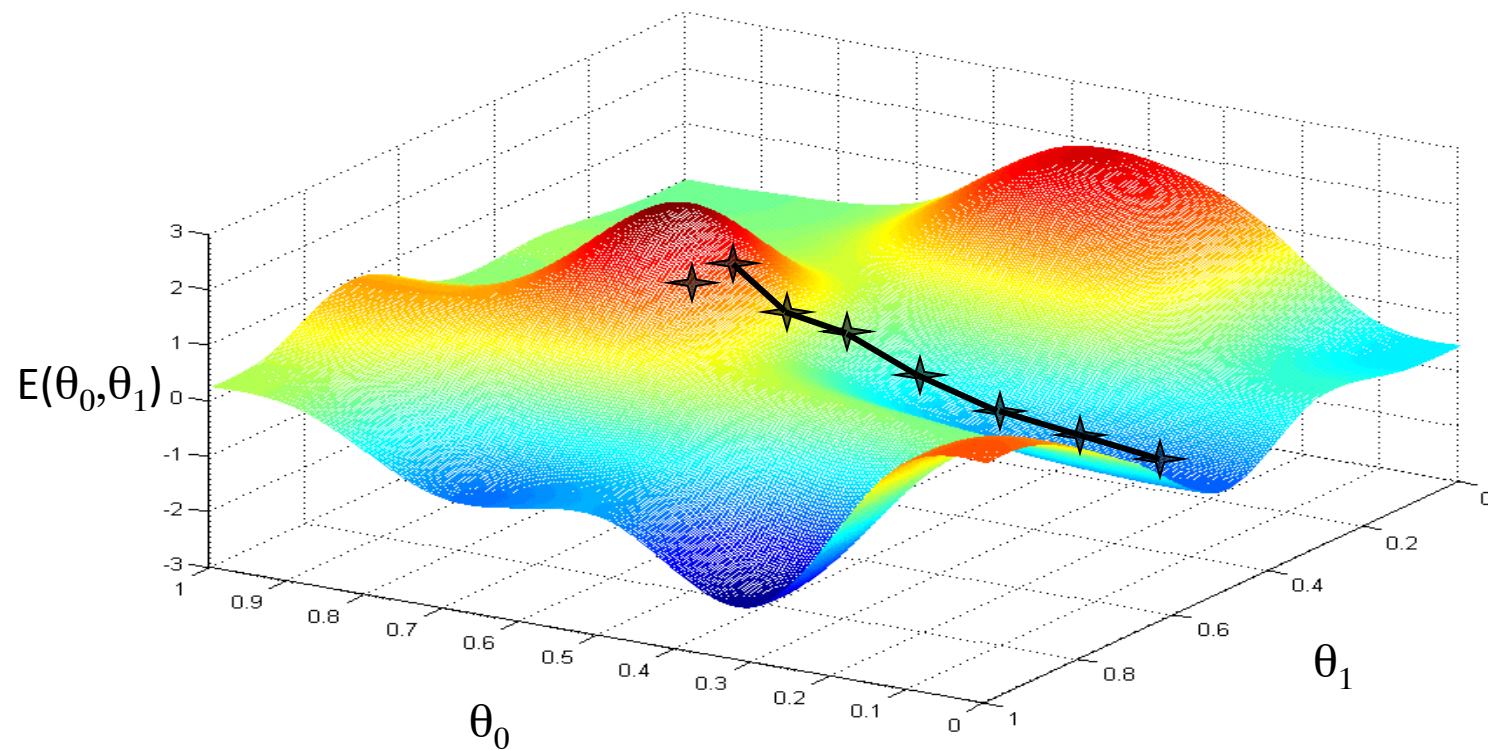
Converting error derivatives into a learning procedure

- The backpropagation algorithm is an efficient way of computing the error derivative dE/dw for every weight on a single training case.
- To get a fully specified learning procedure, we still need to make a lot of other decisions about how to use these error derivatives:
 - Optimization issues: How do we use the error derivatives on individual cases to discover a good set of weights?
 - Generalization issues: How do we ensure that the learned weights work well for cases we did not see during training?

Gradient descent algorithm

repeat until convergence { $W := W - \alpha \frac{\partial E}{\partial W}$ }

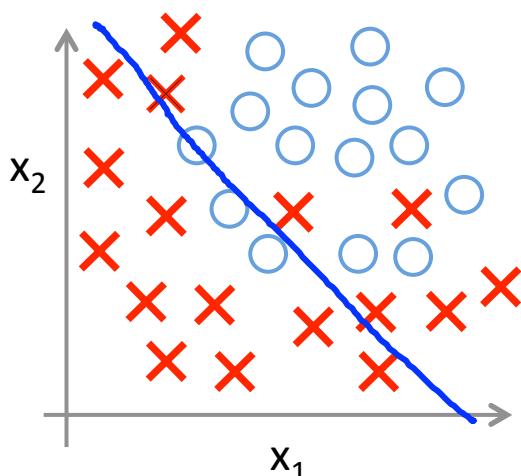




Overfitting: The downside of using powerful models

- The training data contains information about the regularities in the mapping from input to output. But it also contains two types of noise.
 - The target values may be unreliable
 - There is sampling error:
accidental regularities just because of the particular training cases that were chosen.
- When we fit the model, it cannot tell which regularities are real and which are caused by sampling error.
 - So it fits both kinds of regularity.
 - If the model is very flexible it can model the sampling error really well. This is a disaster.

Example: Logistic regression

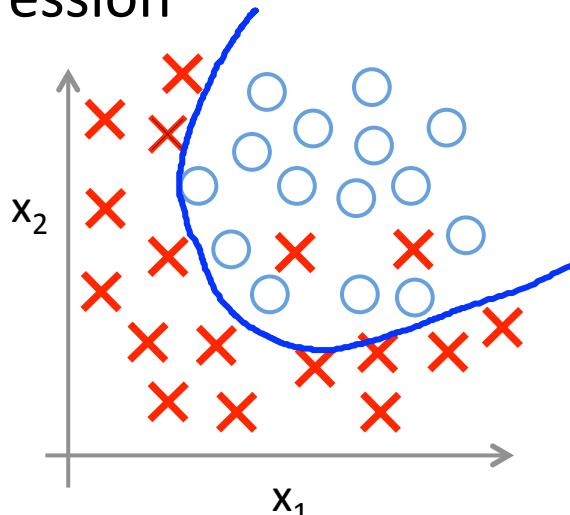


$$\rightarrow h_{\theta}(x) = g(\underline{\theta_0 + \theta_1 x_1 + \theta_2 x_2})$$

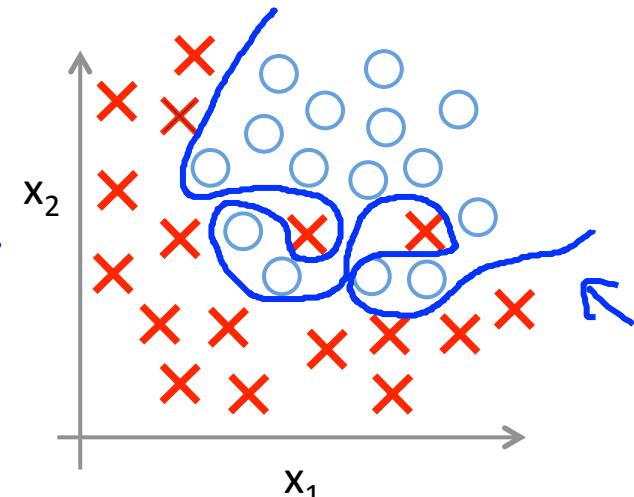
(g = sigmoid function)



"Underfit"



$$g(\underline{\theta_0 + \theta_1 x_1 + \theta_2 x_2} \\ + \underline{\theta_3 x_1^2} + \underline{\theta_4 x_2^2} \\ + \underline{\theta_5 x_1 x_2})$$



$$g(\underline{\theta_0 + \theta_1 x_1 + \theta_2 x_1^2} \\ + \underline{\theta_3 x_1^2 x_2} + \underline{\theta_4 x_1^2 x_2^2} \\ + \underline{\theta_5 x_1^2 x_2^3} + \underline{\theta_6 x_1^3 x_2} + \dots)$$

"Overfit"

Preventing overfitting

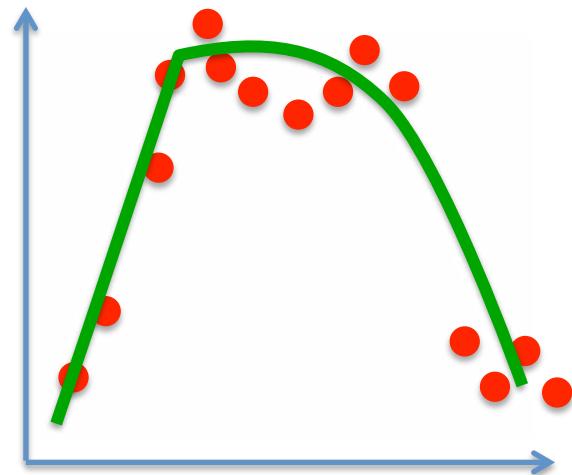
- Approach 1: Get more data!
 - Almost always the best bet if you have enough compute power to train on more data.
- Approach 2: Use a model that has the right capacity:
 - enough to fit the true regularities.
 - not enough to also fit spurious regularities (if they are weaker).
- Approach 3: Average many different models.
 - Use models with different forms.
- Approach 4: (Bayesian) Use a single neural network architecture, but average the predictions made by many different weight vectors.
 - Train the model on different subsets of the training data (this is called “bagging”).

Some ways to limit the capacity of a neural net

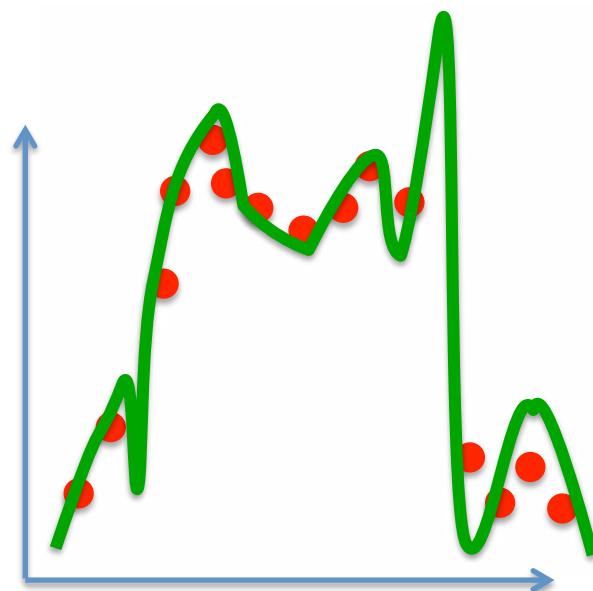
- The capacity can be controlled in many ways:
 - Architecture: Limit the number of hidden layers and the number of units per layer.
 - Early stopping: Start with small weights and stop the learning before it overfits.
 - Weight-decay: Penalize large weights using penalties or constraints on their squared values (L2 penalty) or absolute values (L1 penalty).
 - Noise: Add noise to the weights or the activities.
- Typically, a combination of several of these methods is used.

Small Model vs. Big Model + Regularize

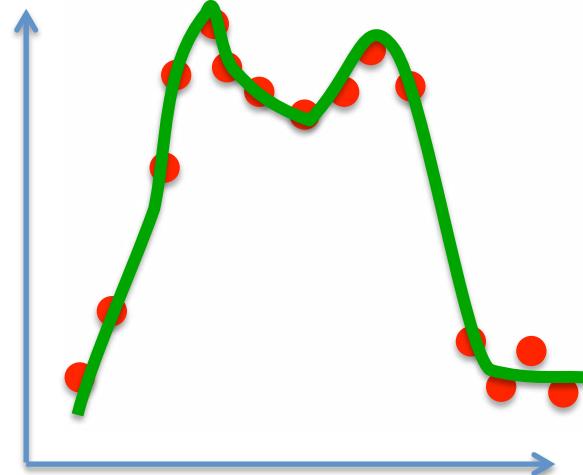
Small model



Big model

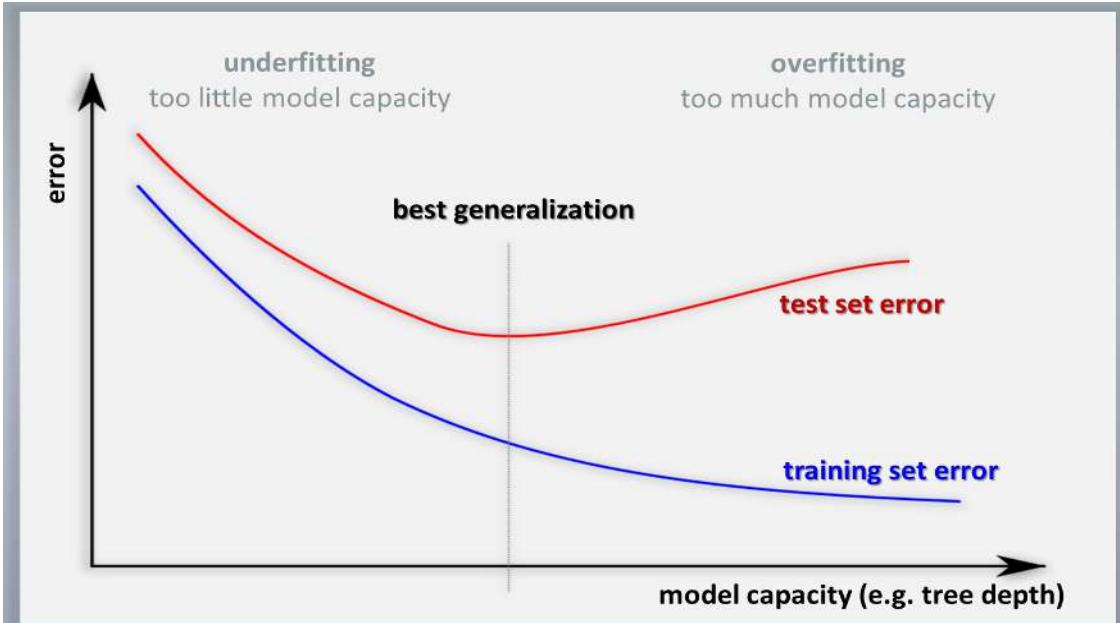


Big model +
regularize



Cross-validation for choosing meta parameters

- Divide the total dataset into three subsets:
 - Training data is used for learning the parameters of the model.
 - Validation data is not used for learning but is used for deciding what settings of the meta parameters work best.
 - Test data is used to get a final, unbiased estimate of how well the network works.

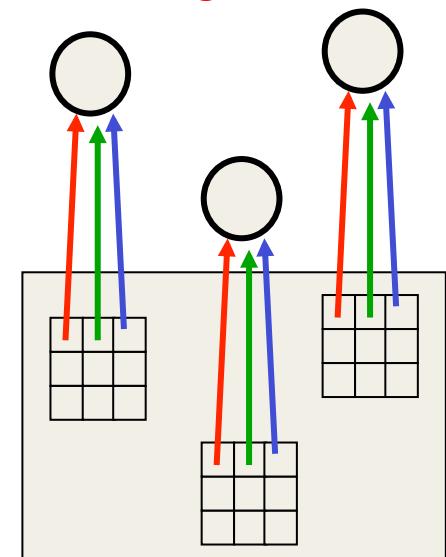


Convolutional Neural Networks

(currently the dominant approach for neural networks)

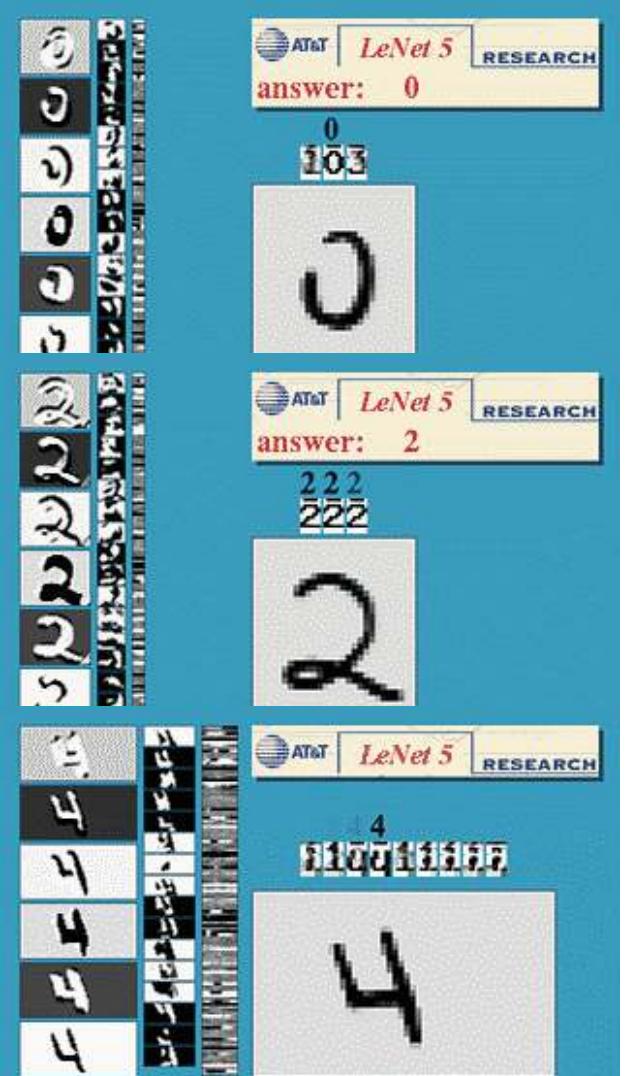
- Use many different copies of the same feature detector with different positions.
 - Replication greatly reduces the number of free parameters to be learned.
- Use several different feature types, each with its own map of replicated detectors.
 - Allows each patch of image to be represented in several ways.

The similarly colored connections all have the same weight.

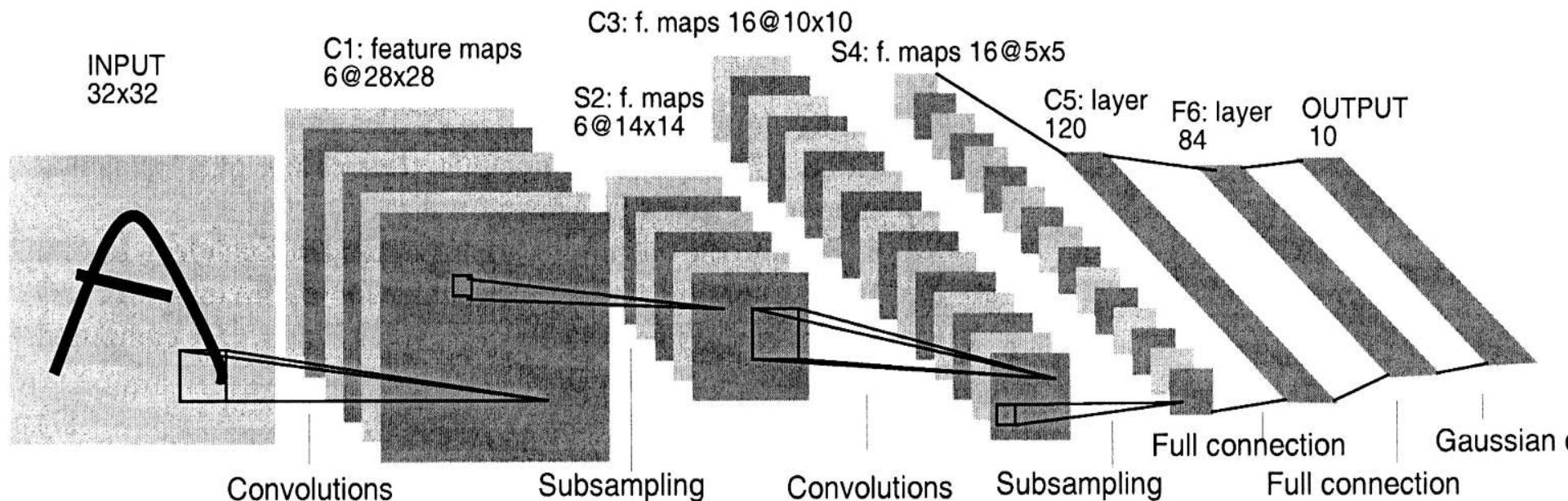


Le Net

- Yann LeCun and his collaborators developed a really good recognizer for handwritten digits by using backpropagation in a feedforward net with:
 - Many hidden layers
 - Many maps of replicated convolution units in each layer
 - Pooling of the outputs of nearby replicated units
 - A wide input can cope with several digits at once even if they overlap
- This net was used for reading ~10% of the checks in North America.
- Look the impressive demos of LENET at <http://yann.lecun.com>



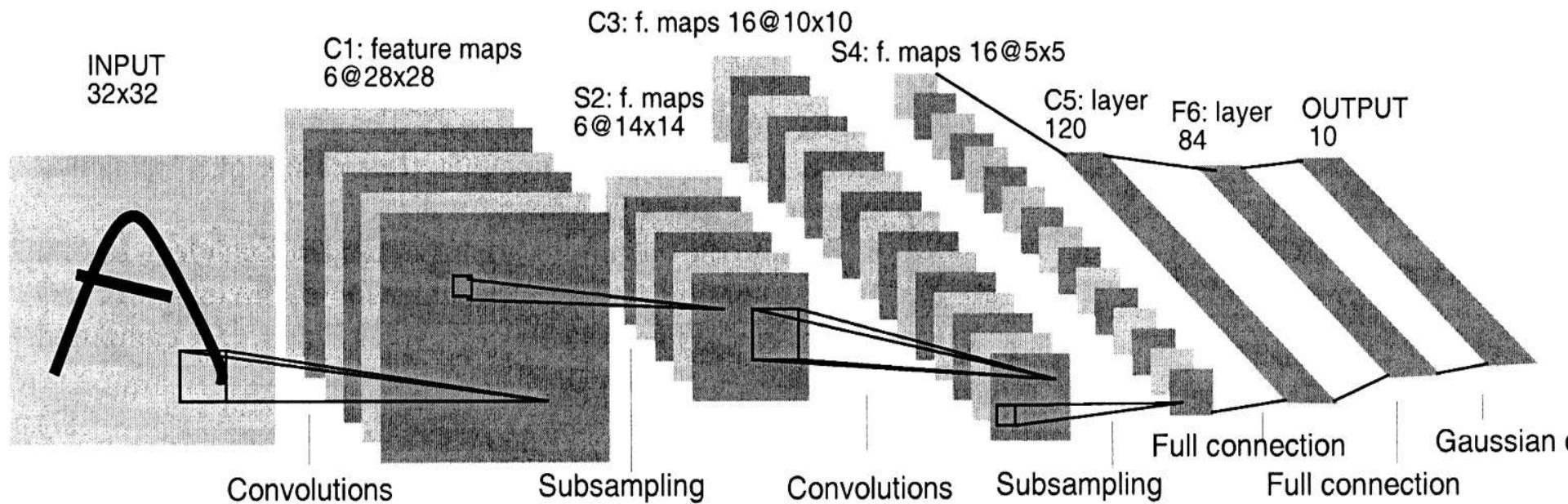
The architecture of LeNet5

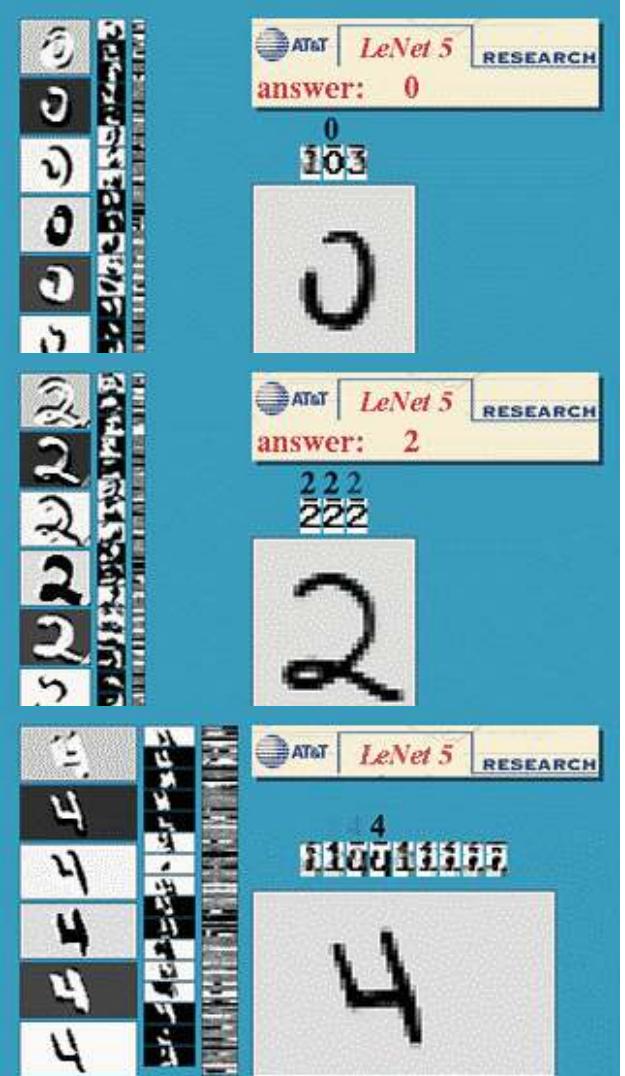


Pooling the outputs of replicated feature detectors

- Get a small amount of translational invariance at each level by averaging four neighboring outputs to give a single output.
 - This reduces the number of inputs to the next layer of feature extraction.
 - Taking the maximum of the four works slightly better.
- **Problem:** After several levels of pooling, we have lost information about the precise positions of things.

The architecture of LeNet5





4	3	2	1	5	8	2	3	6	1
4->6	3->5	8->2	2->1	5->3	4->8	2->8	3->5	6->5	7->3
9	8	7	5	7	6	7	3	3	4
9->4	8->0	7->8	5->3	8->7	0->6	3->7	2->7	8->3	9->4
8	3	8	3	0	9	9	1	4	1
8->2	5->3	4->8	3->9	6->0	9->8	4->9	6->1	9->4	9->1
9	0	1	3	3	9	6	0	2	6
9->4	2->0	6->1	3->5	3->2	9->5	6->0	6->0	6->0	6->8
4	7	9	4	2	7	4	9	9	9
4->6	7->3	9->4	4->6	2->7	9->7	4->3	9->4	9->4	9->4
7	4	8	3	8	6	8	3	3	9
8->7	4->2	8->4	3->5	8->4	6->5	8->5	3->8	3->8	9->8
1	9	6	0	6	7	0	1	4	1
1->5	9->8	6->3	0->2	6->5	9->5	0->7	1->6	4->9	2->1
2	8	4	2	2	1	9	1	6	5
2->8	8->5	4->9	7->2	7->2	6->5	9->7	6->1	5->6	5->0
1	2								
4->9	2->8								

The 82 errors made by LeNet5

Notice that most of the errors are cases that people find quite easy.

The human error rate is probably 20 to 30 errors but nobody has had the patience to measure it.

Test set size is 10000.

The brute force approach

- LeNet uses knowledge about the invariances to **design**:
 - the local connectivity
 - the weight-sharing
 - the pooling.
- This achieves about 80 errors.
- Ciresan *et al.* (2010) inject knowledge of invariances by creating a huge amount of carefully designed extra training data:
 - For each training image, they produce many new training examples by applying many different transformations.
 - They can then train a large, deep, dumb net on a GPU without much overfitting.
- They achieve about 35 errors.

From hand-written digits to 3-D objects

- Recognizing real objects in color photographs downloaded from the web is much more complicated than recognizing hand-written digits:
 - Hundred times as many classes (1000 vs. 10)
 - Hundred times as many pixels (256×256 color vs. 28×28 gray)
 - Two dimensional image of three-dimensional scene.
 - Cluttered scenes requiring segmentation
 - Multiple objects in each image.
- Will the same type of convolutional neural network work?

The ILSVRC-2012 competition on ImageNet

- The dataset has 1.2 million high-resolution training images.
- The classification task:
 - Get the “correct” class in your top 5 bets.
There are 1000 classes.
- The localization task:
 - For each bet, put a box around the object.
Your box must have at least 50% overlap with the correct box.



Groundtruth:
tv or monitor
tv or monitor (2)
tv or monitor (3)
person
remote control
remote control (2)

Examples from the test set (with the network's guesses)



cheetah

cheetah

leopard

snow leopard

Egyptian cat



bullet train

bullet train

passenger car

subway train

electric locomotive



hand glass

scissors

hand glass

frying pan

stethoscope

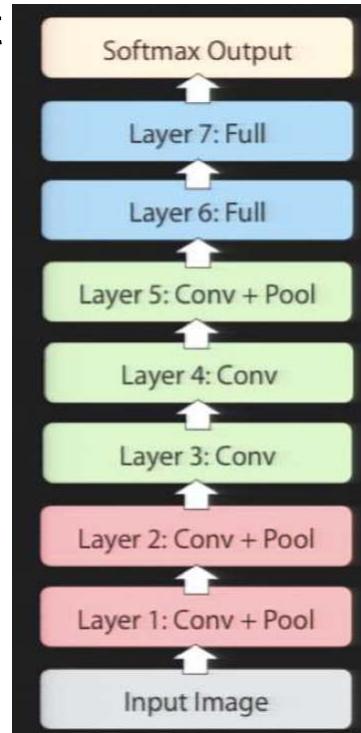
- University of Toronto (Alex Krizhevsky) • 16.4% 34.1%

Error rates on the ILSVRC-2012 competition

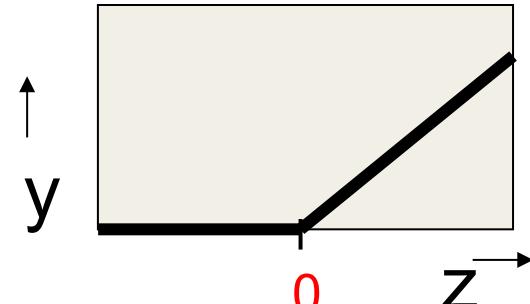
	classification	classification & localization
• University of Tokyo	• 26.1%	53.6%
• Oxford University Computer Vision Group	• 26.9%	50.0%
• INRIA (French national research institute in CS) + XRCE (Xerox Research Center Europe)	• 27.0%	
• University of Amsterdam	• 29.5%	

A neural network for ImageNet

- Alex Krizhevsky (NIPS 2012) developed a very deep convolutional neural net
- Its architecture:



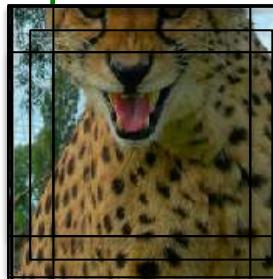
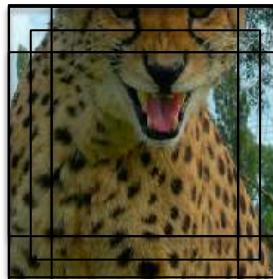
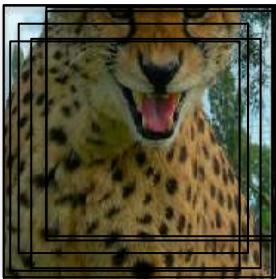
- The activation functions were:
 - Rectified linear units in every hidden layer.
 - These train much faster.
 - More expressive than logistic.



- Competitive normalization to suppress hidden activities when nearby units have stronger activities.
 - This helps with variations in intensity.

Tricks that significantly improve generalization

- Train on random 224x224 patches from the 256x256 images to get more data. Also use left-right reflections of the images.
 - At test time, combine the opinions from ten different patches: The four 224x224 corner patches plus the central 224x224 patch plus the reflections of those five patches.
- Use “dropout” to regularize the weights in the globally connected layers (which contain most of the parameters).
 - Dropout means that half of the hidden units in a layer are randomly removed for each training example.
 - This stops hidden units from relying too much on other hidden units.



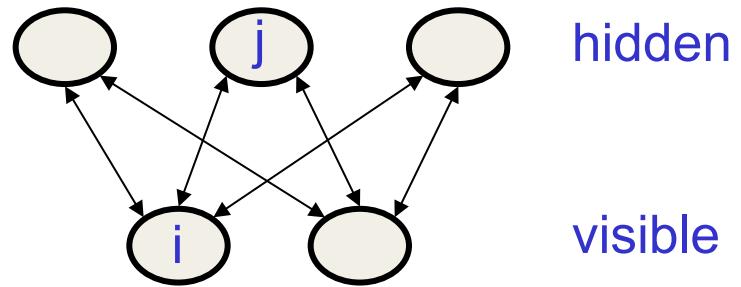
The hardware required for Alex's net

- He uses a very efficient implementation of convolutional nets on two Nvidia GTX 580 Graphics Processor Units (over 1000 fast little cores)
 - GPUs are very good for matrix-matrix multiplies.
 - GPUs have very high bandwidth to memory.
 - This allows him to train the network in a week.
 - It also makes it quick to combine results from 10 patches at test time.
- We can spread a network over many cores if we can communicate the states fast enough.
- As cores get cheaper and datasets get bigger, big neural nets will improve faster than old-fashioned (*i.e.*, pre Oct 2012) computer vision systems.

Auto-Encoders

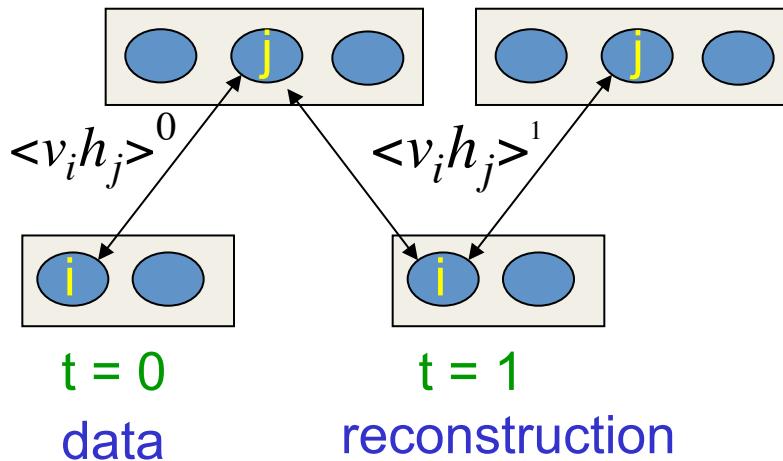
Restricted Boltzmann Machines

- Simple recursive neural net
 - Only one layer of hidden units.
 - No connections between hidden units.
- Idea:
 - The hidden layer should “auto-encode” the input.



$$p(h_j = 1) = \frac{1}{1 + e^{-(b_j + \sum_{i \in vis} v_i w_{ij})}}$$

Contrastive divergence to train an RBM



$$\Delta w_{ij} = \varepsilon (\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1)$$

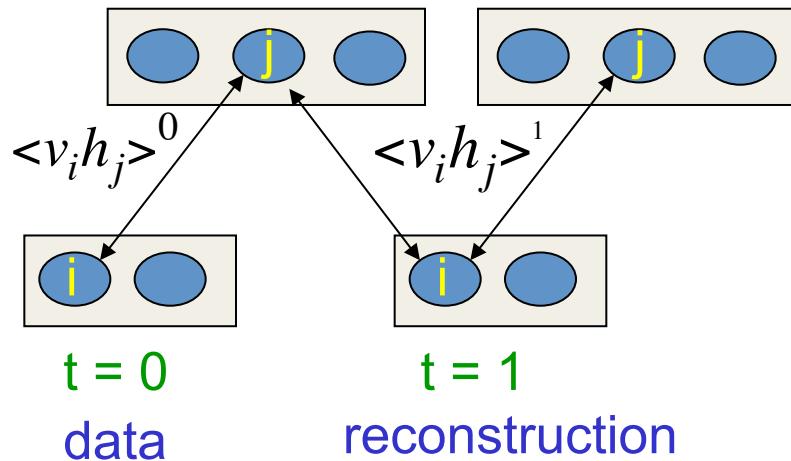
Start with a training vector on the visible units.

Update all the hidden units in parallel.

Update all the visible units in parallel to get a “reconstruction”.

Update the hidden units again.

Explanation



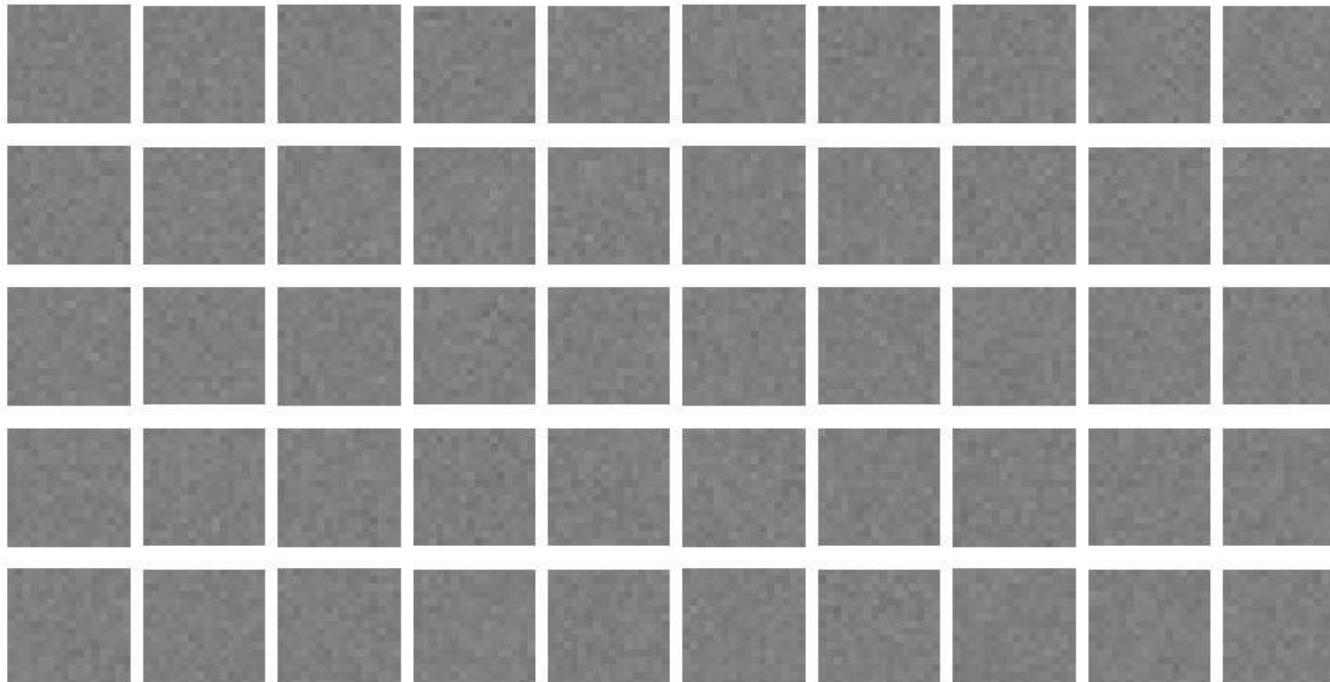
Ideally, hidden layers re-generate the input.

If that's not the case, the hidden layers generate something else.

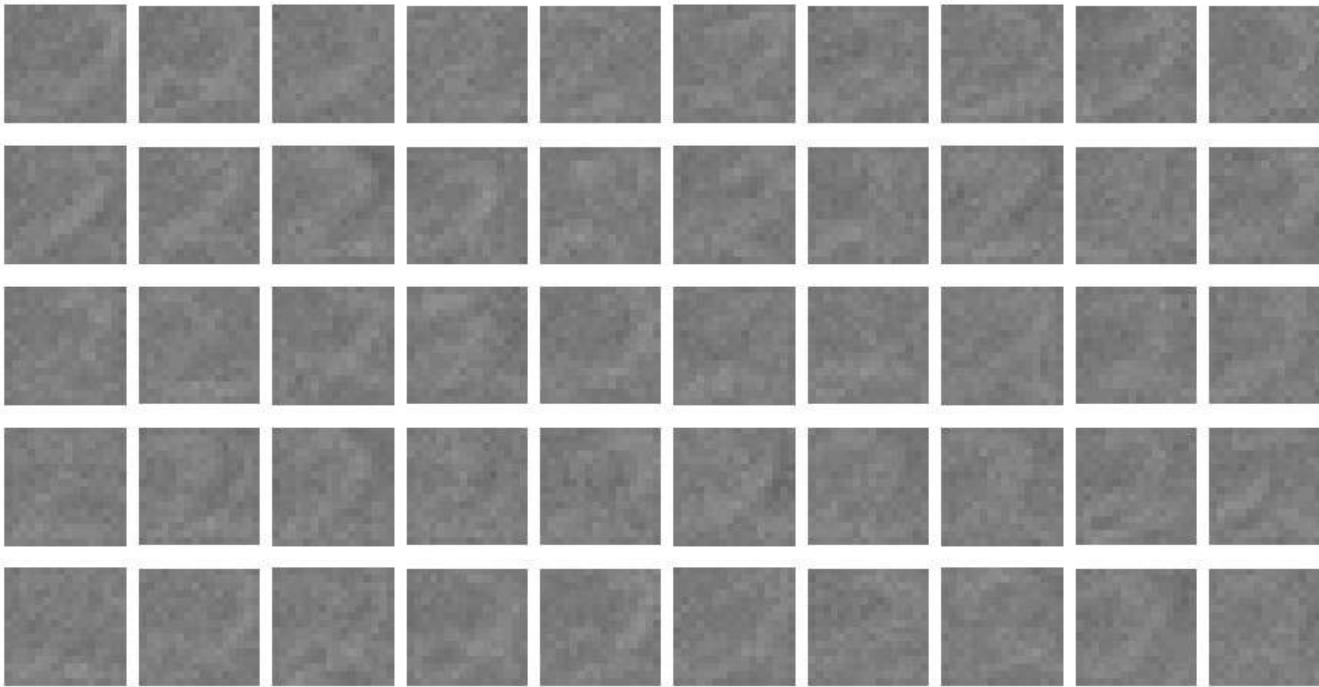
Change the weights so that this wouldn't happen.

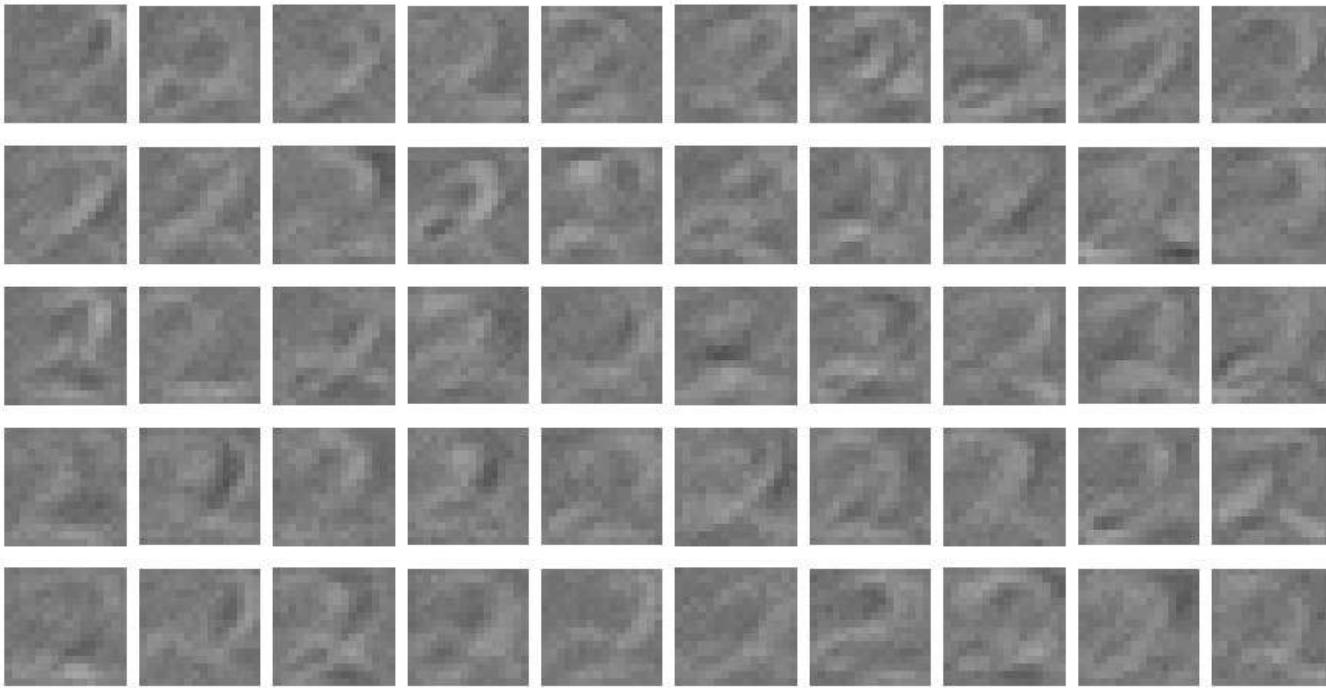
$$\Delta w_{ij} = \varepsilon (\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1)$$

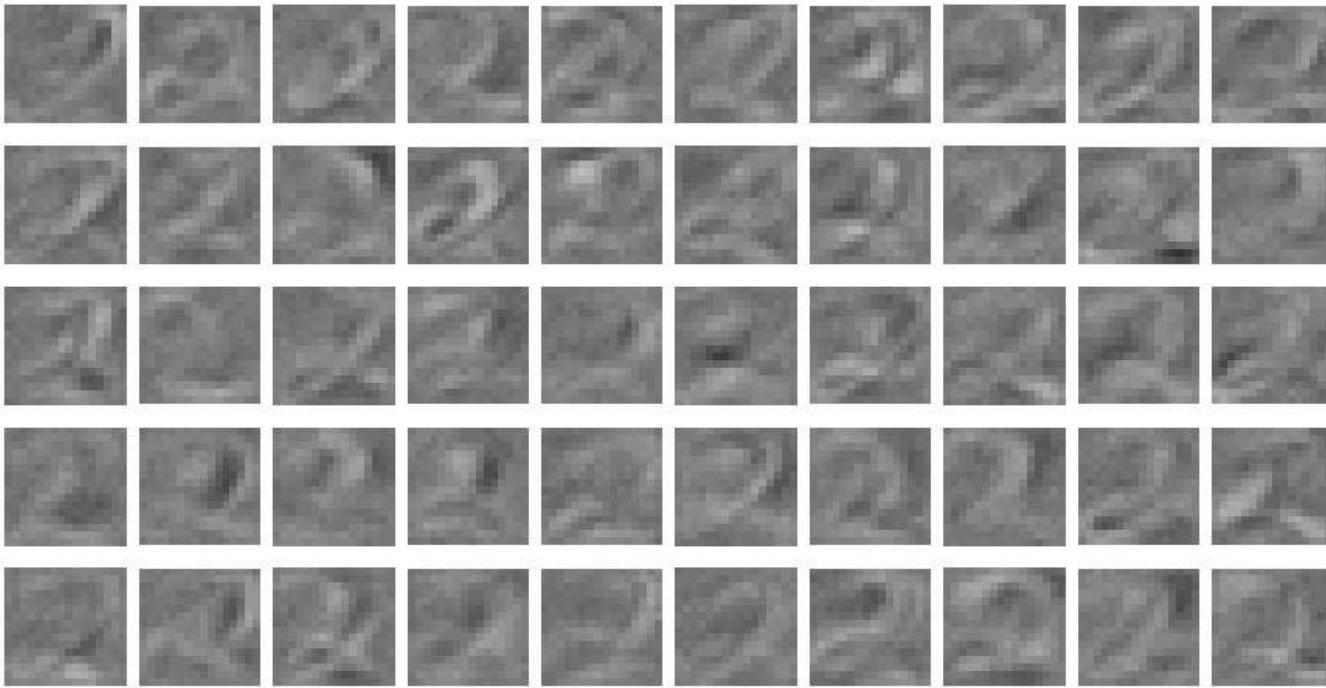
The weights of the 50 feature detectors

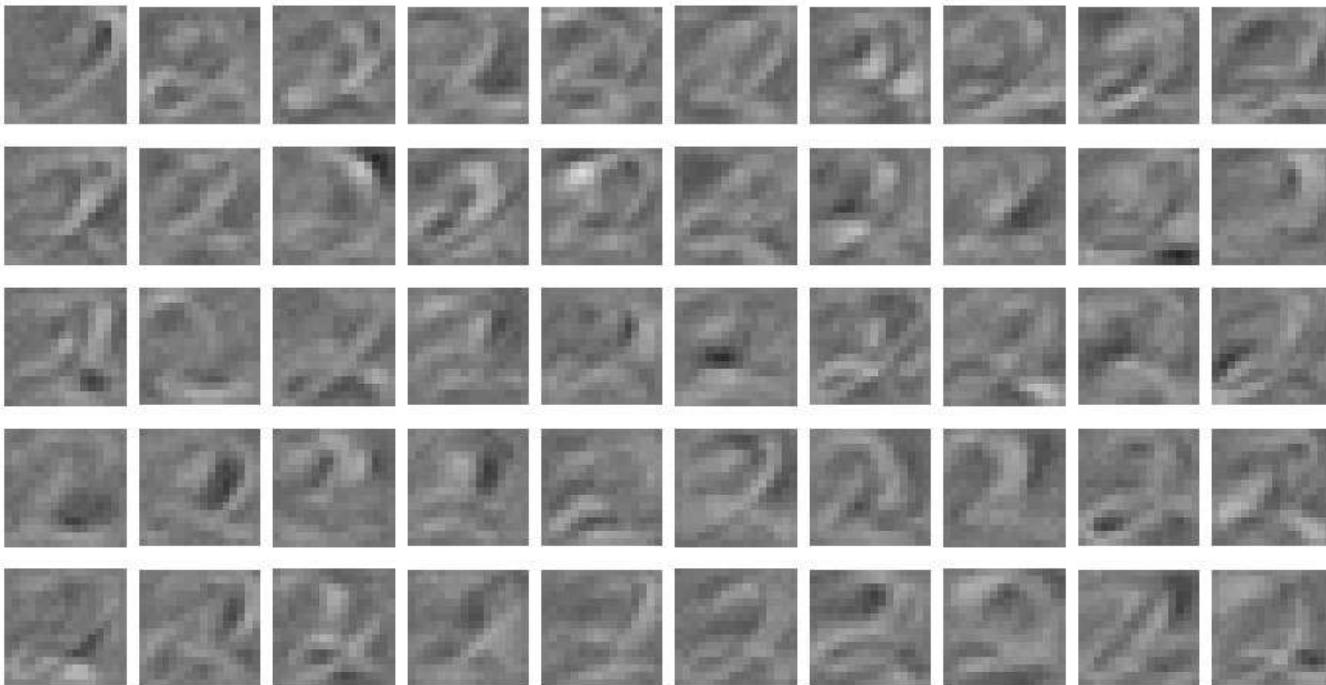


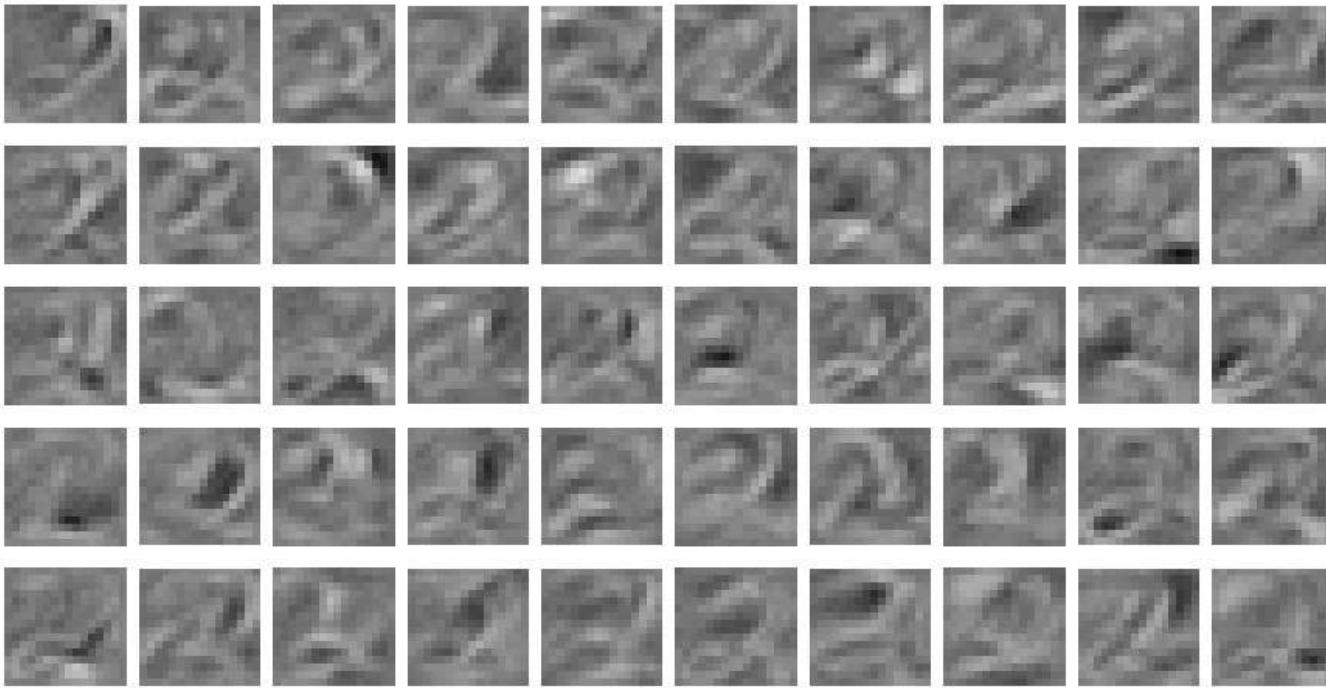
We start with small random weights to break symmetry

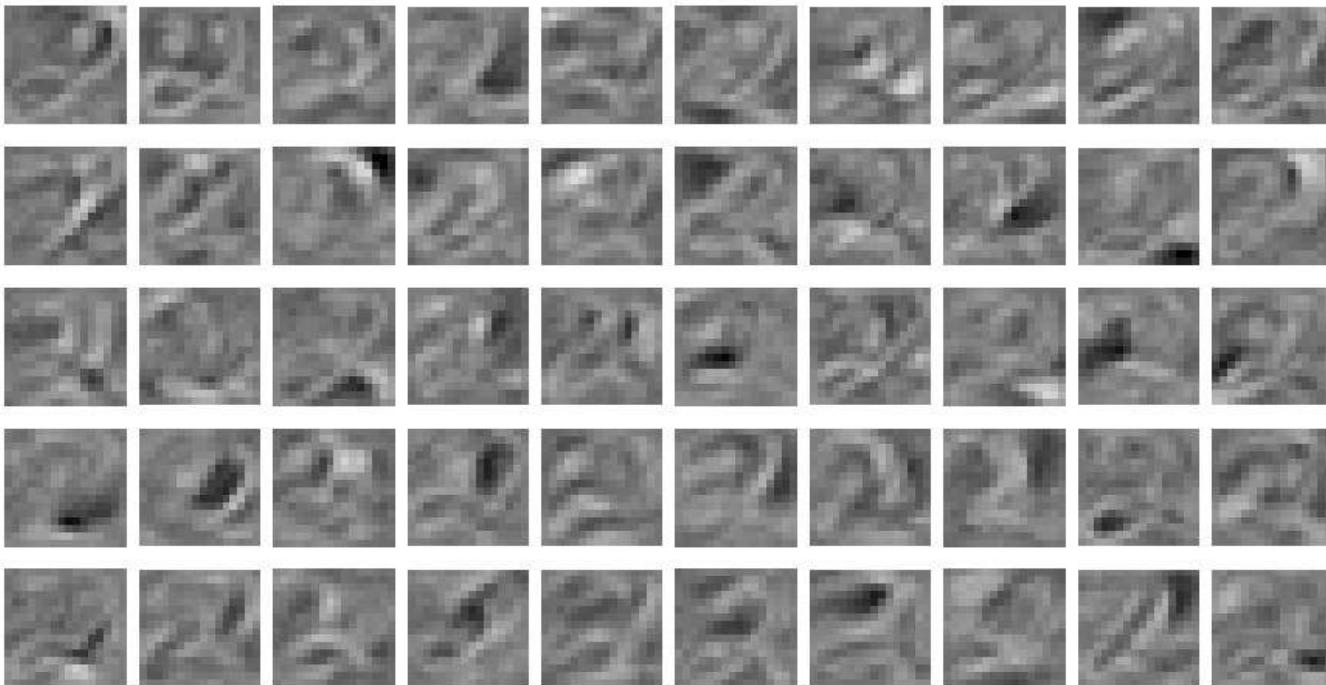


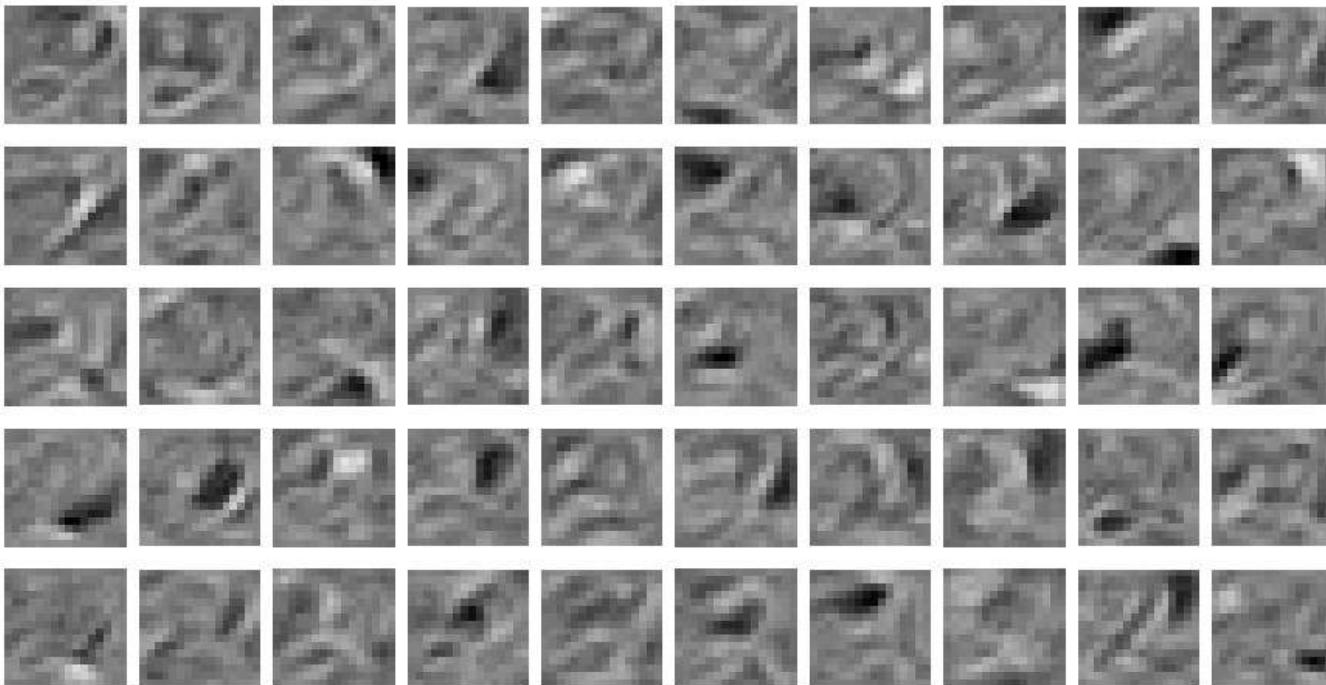


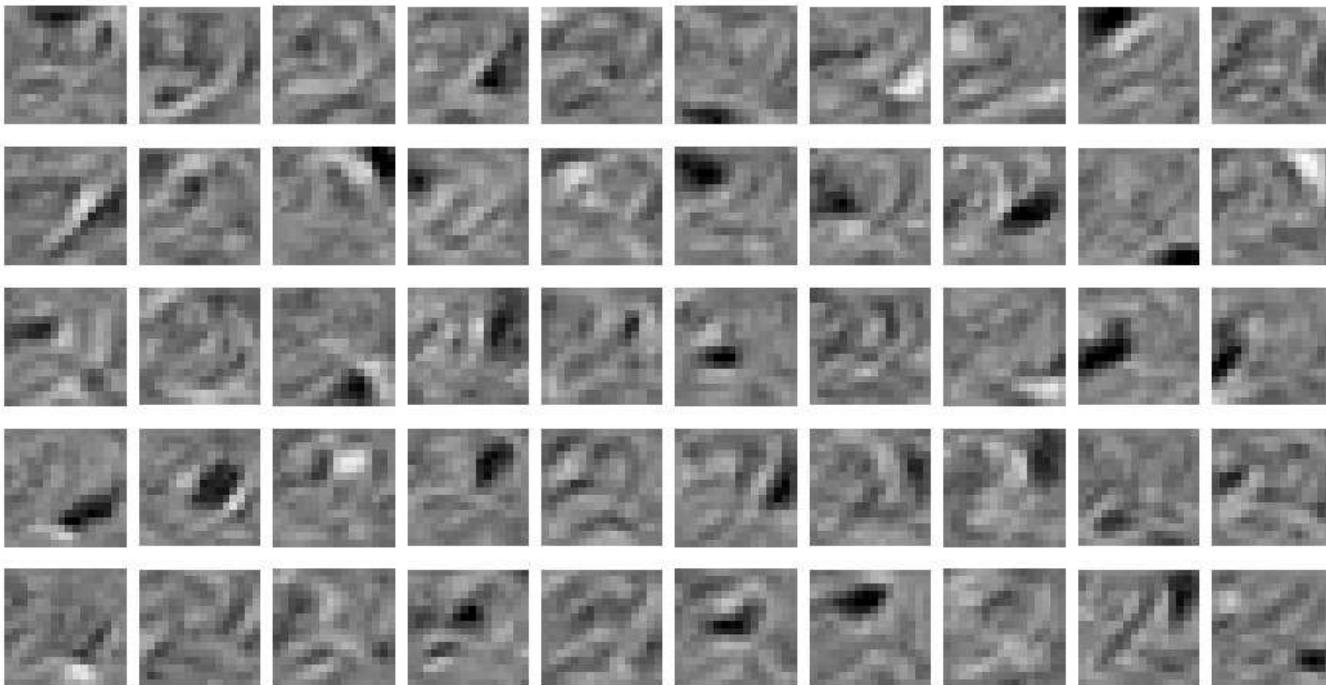




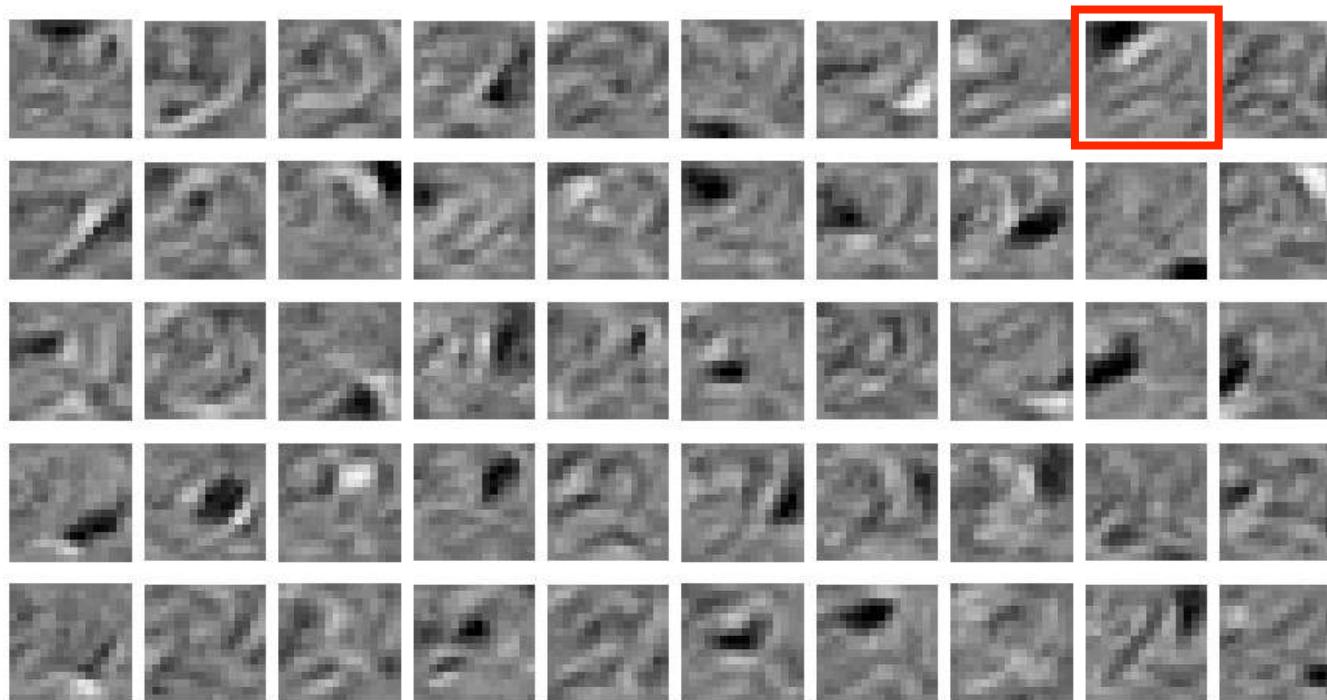




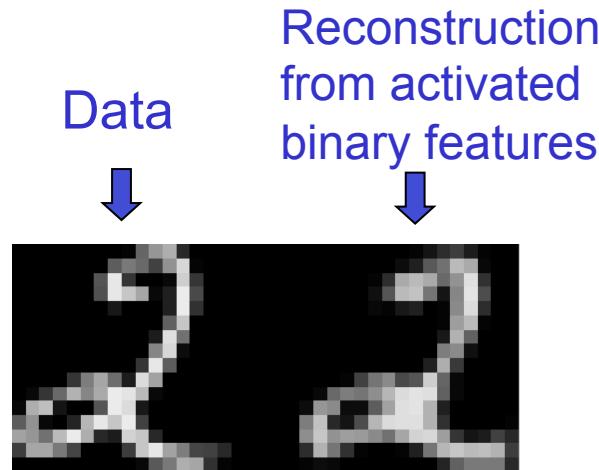




The final 50×256 weights: Each neuron grabs a different feature



How well can we reconstruct digit images from the binary feature activations?



New test image from the digit class that the model was trained on

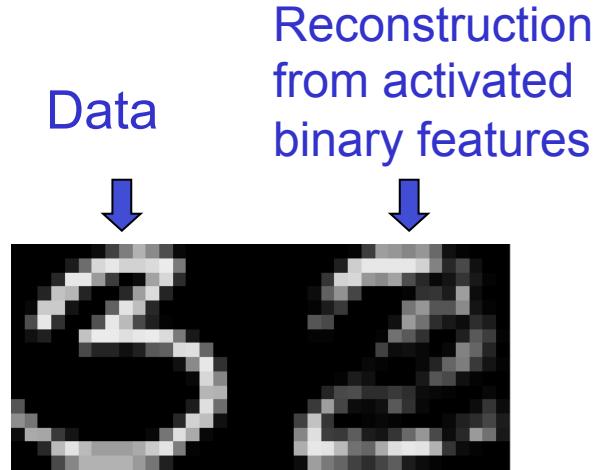
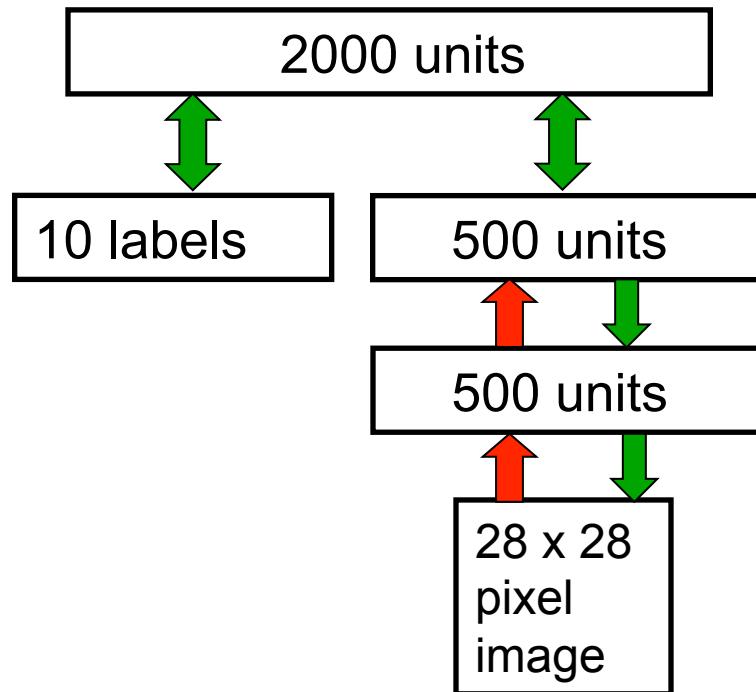


Image from an unfamiliar digit class
The network tries to see every image as a 2.

The DBN used for modeling the joint distribution of MNIST digits and their labels

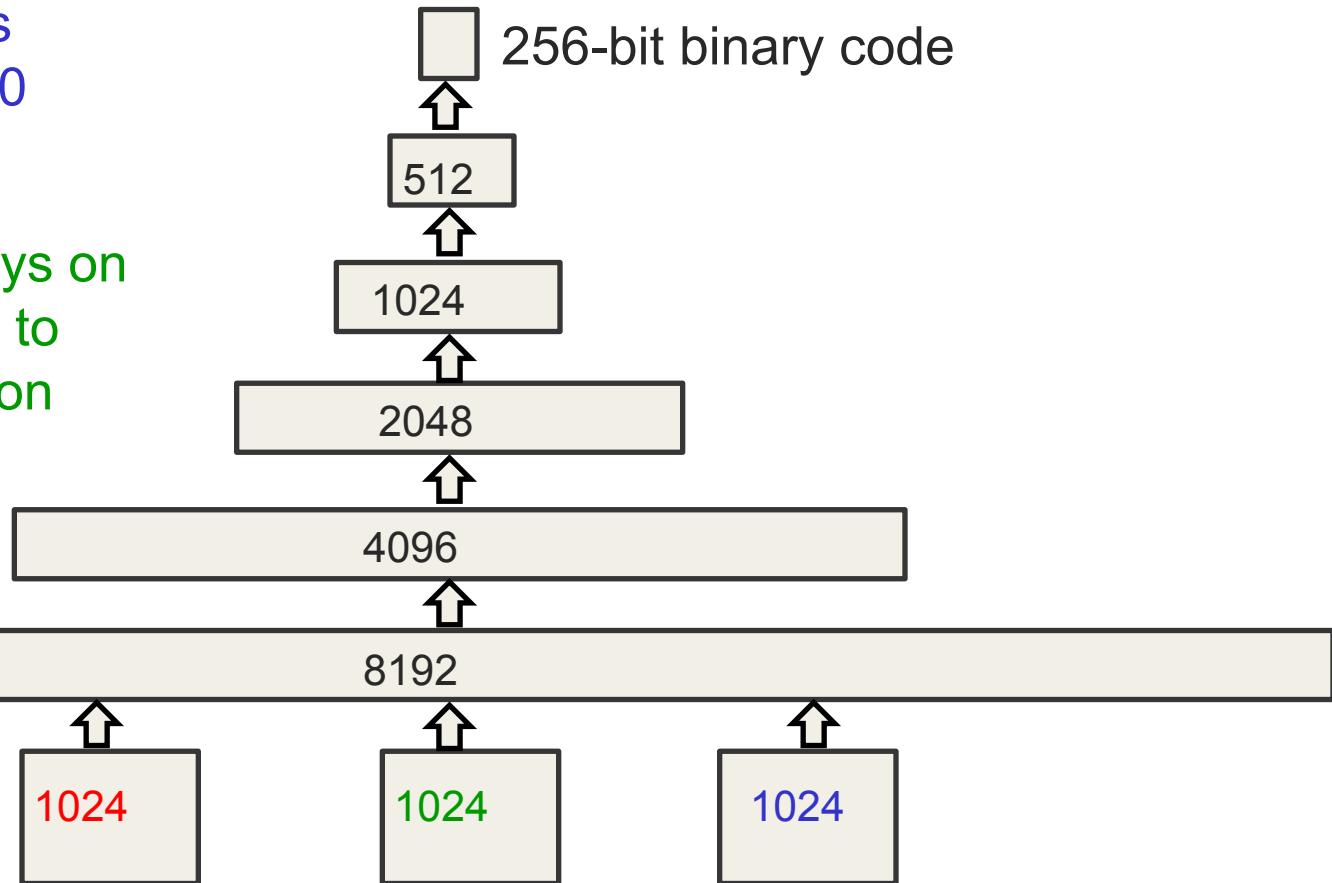
- The first two hidden layers are learned without using labels.
- The top layer connects the labels to the features in the second hidden layer.
- The weights are then fine-tuned to be a better generative model using backpropagation.



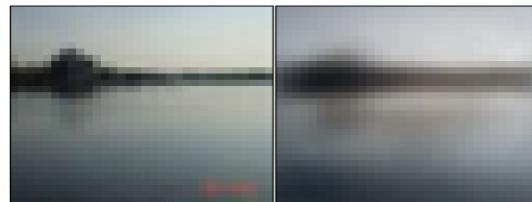
Krizhevsky's deep autoencoder

The encoder has about 67,000,000 parameters.

It takes a few days on a GTX 285 GPU to train on two million images.



Reconstructions of 32x32 color images from 256-bit codes



retrieved using 256 bit codes



retrieved using Euclidean distance in pixel intensity space



retrieved using 256 bit codes



retrieved using Euclidean distance in pixel intensity space

