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

Kari Pulli

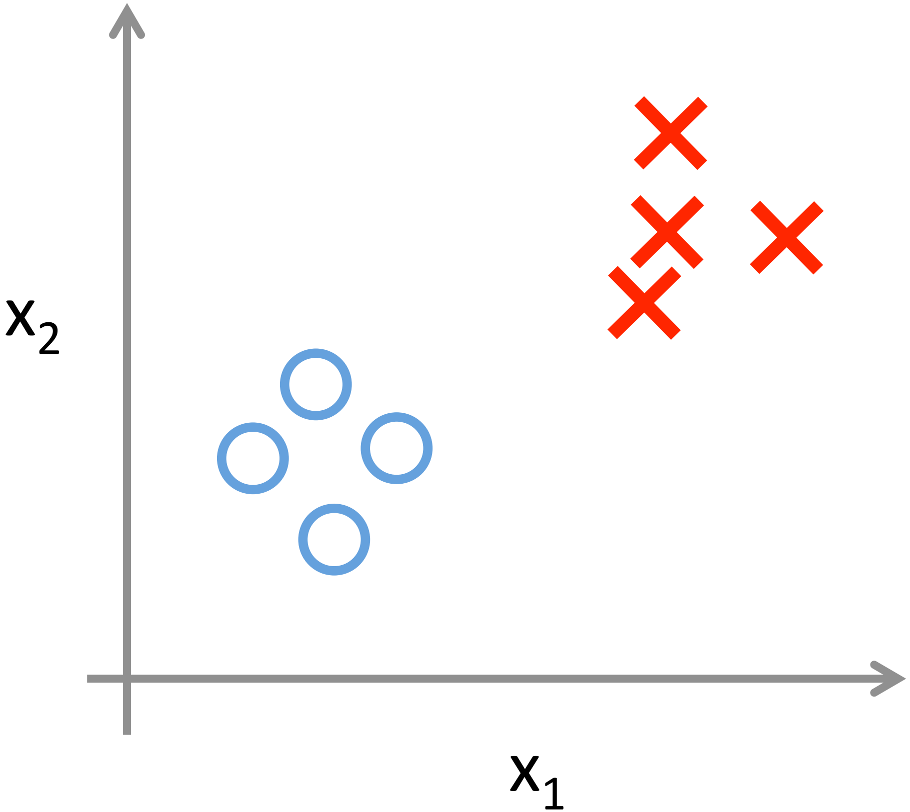
VP Computational Imaging

Light

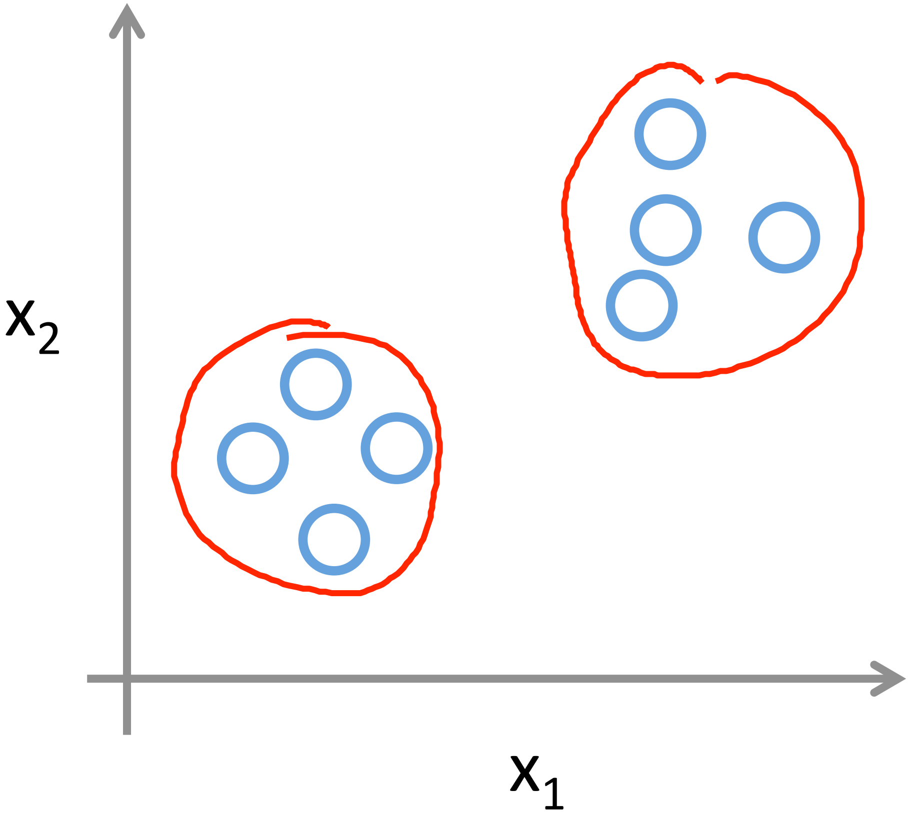
Material sources

- 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

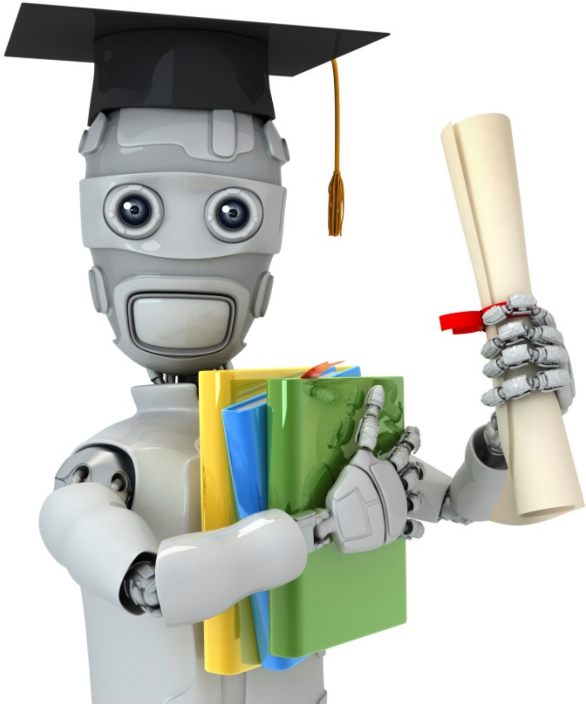
Supervised Learning



Unsupervised Learning



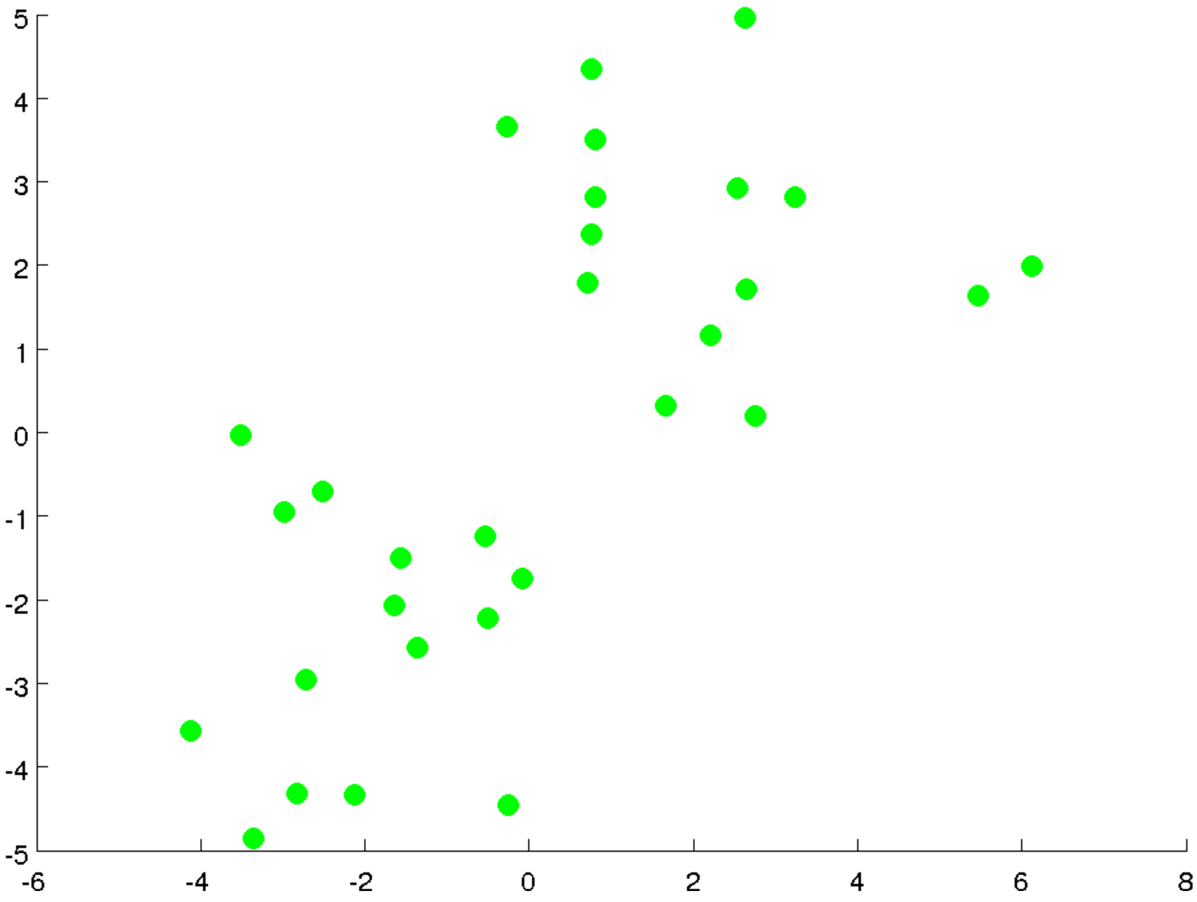
Andrew Ng



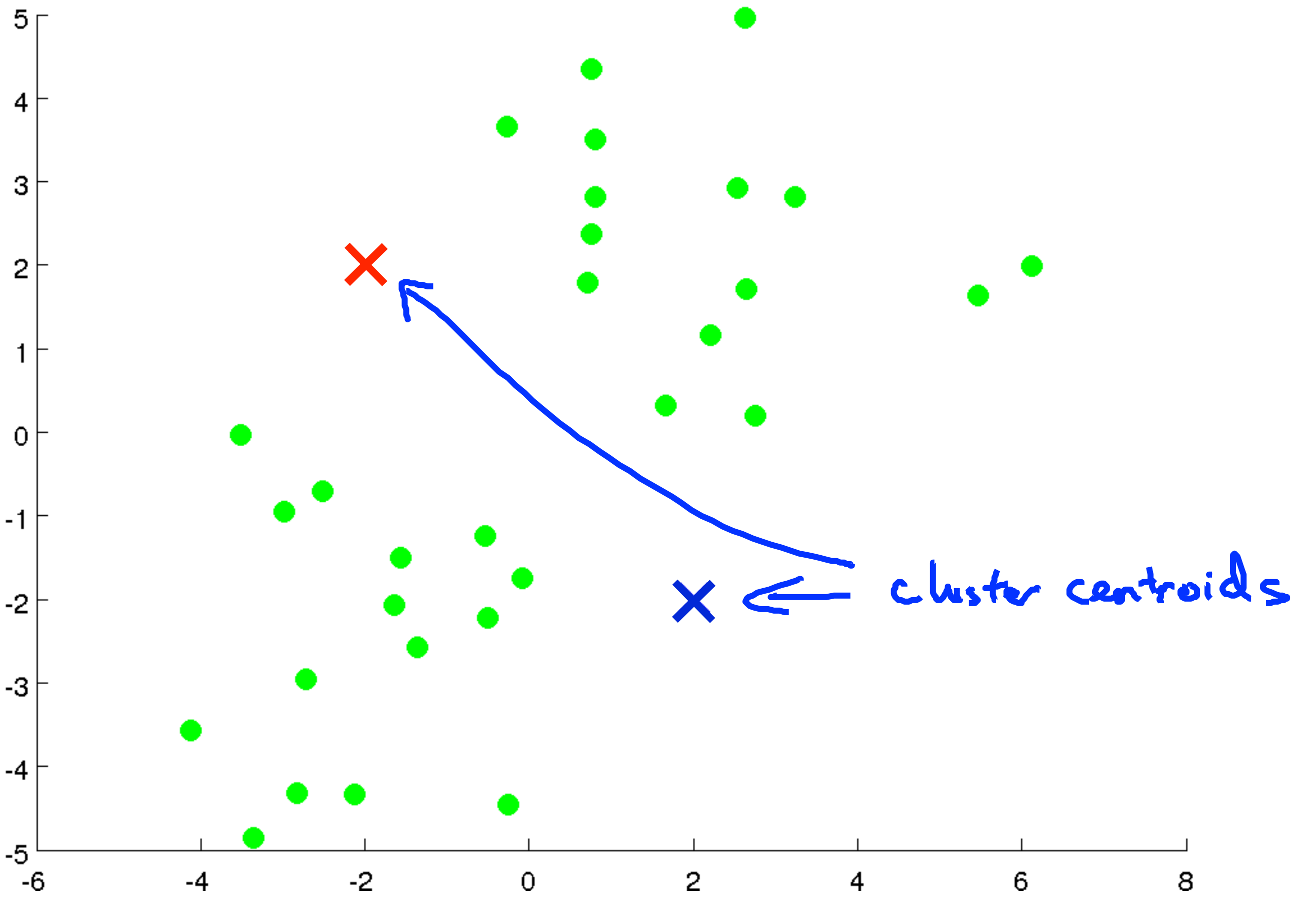
Machine Learning

Clustering

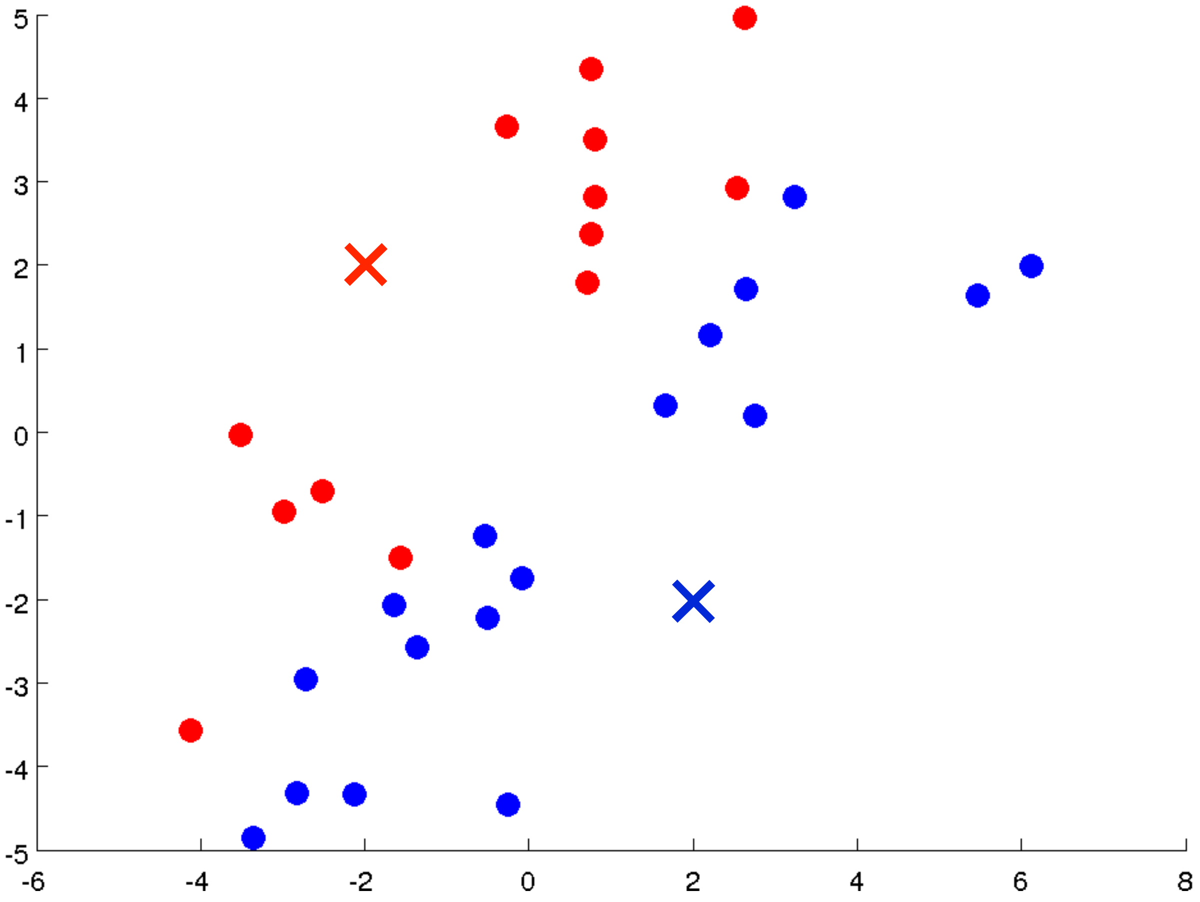
K-means algorithm



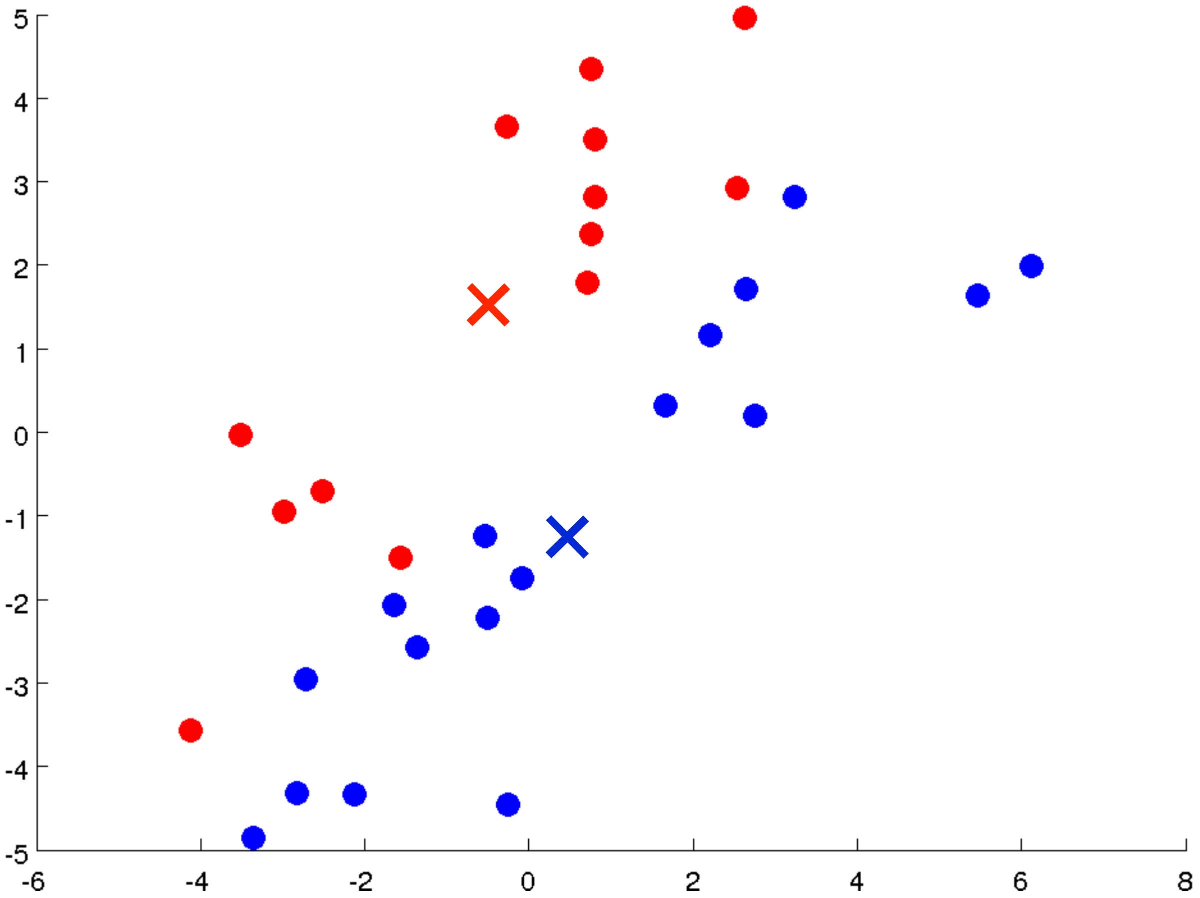
Andrew Ng

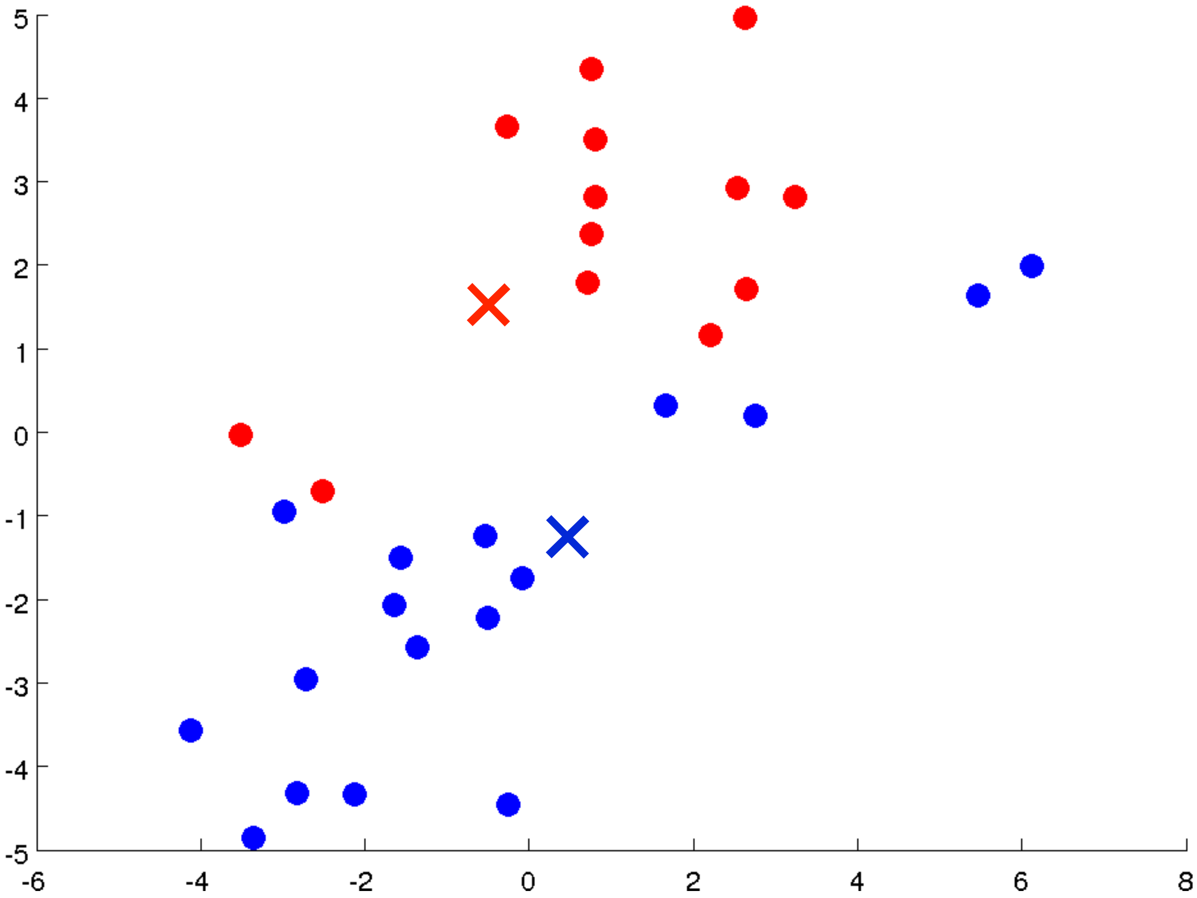


cluster centroids

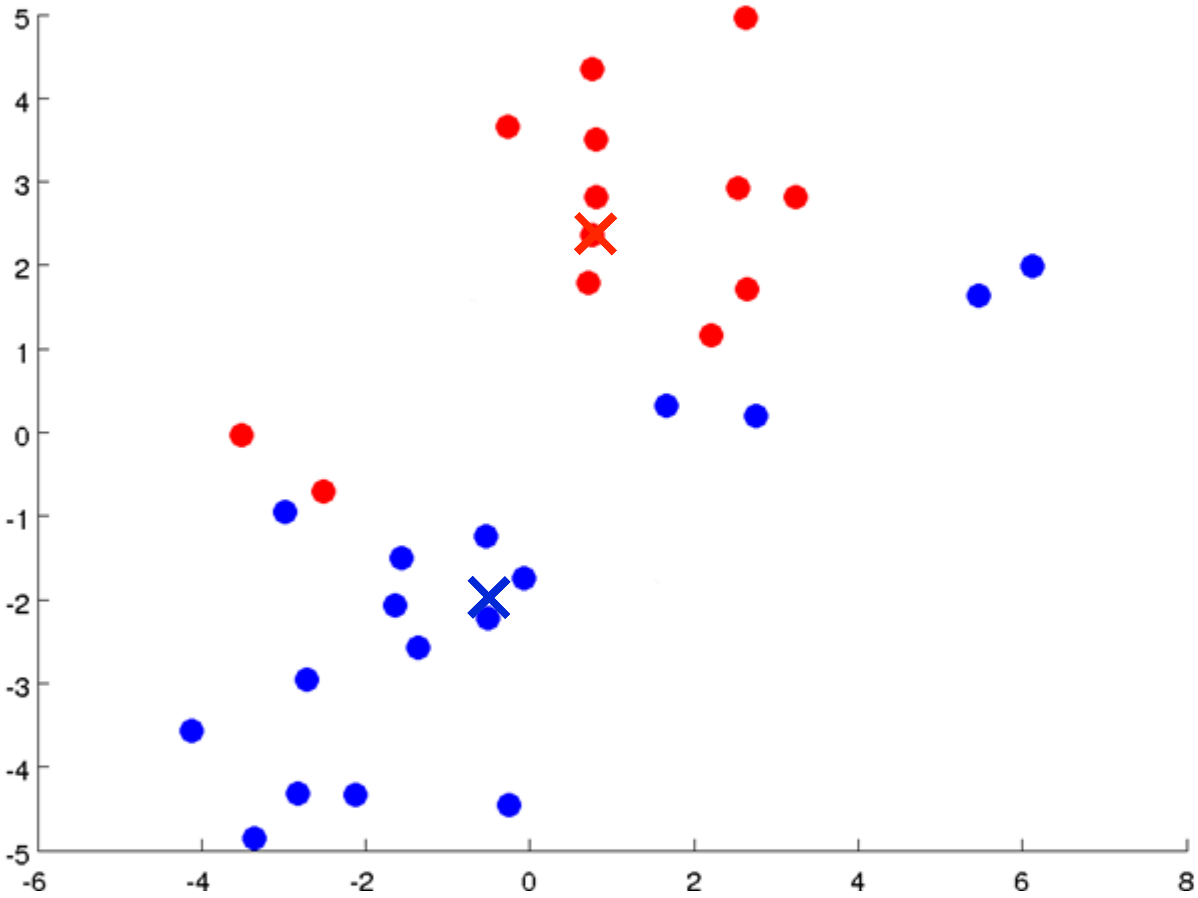


5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8

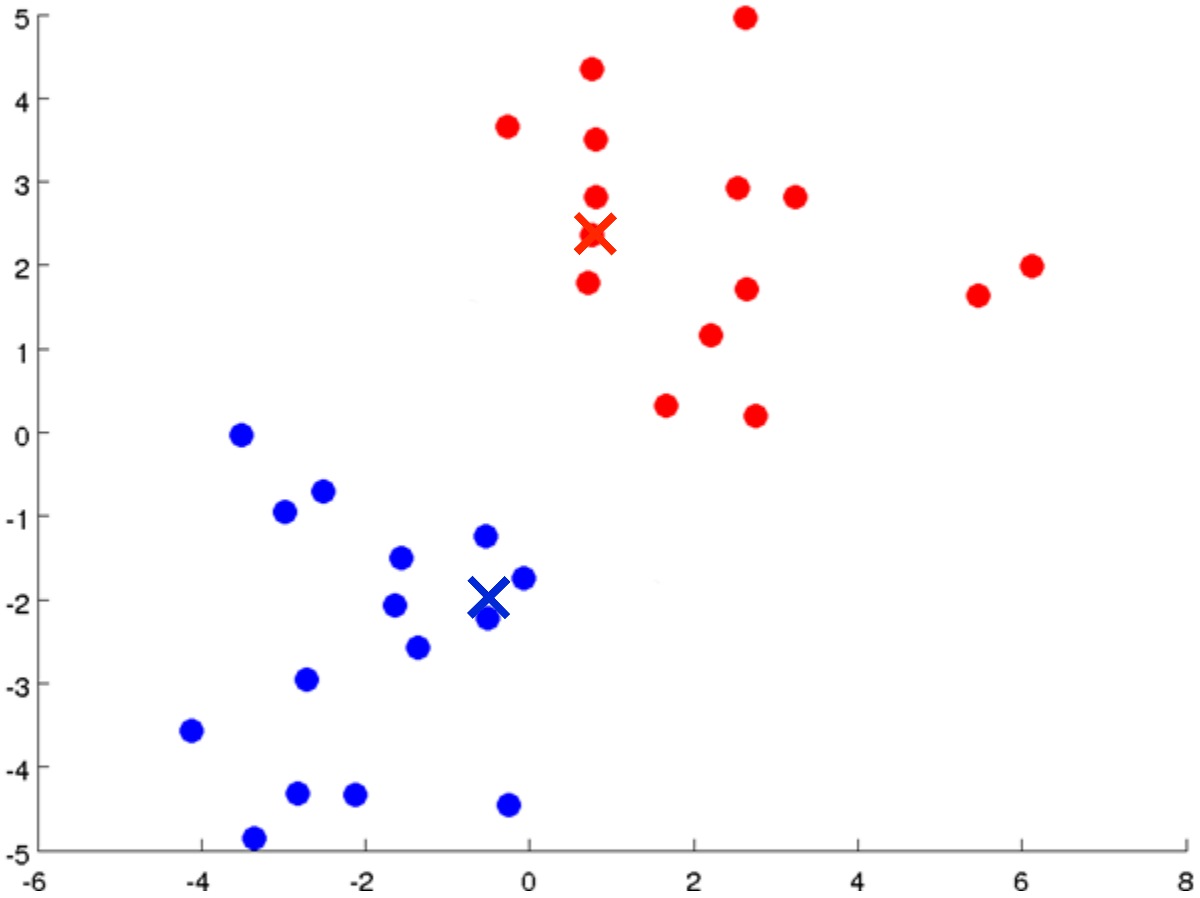

5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8



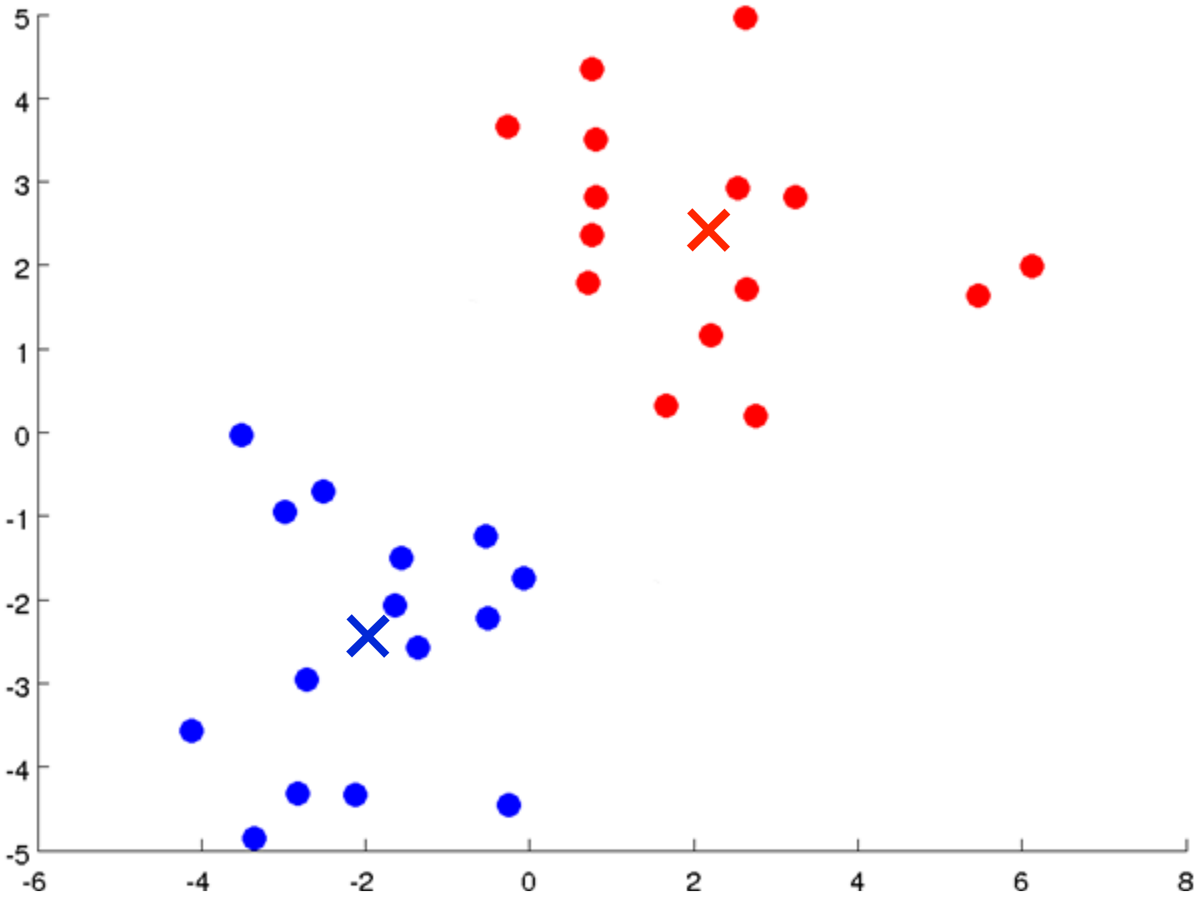
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8



Andrew Ng



Andrew Ng



5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8

Microsoft Research Cambridge

Decision Forests

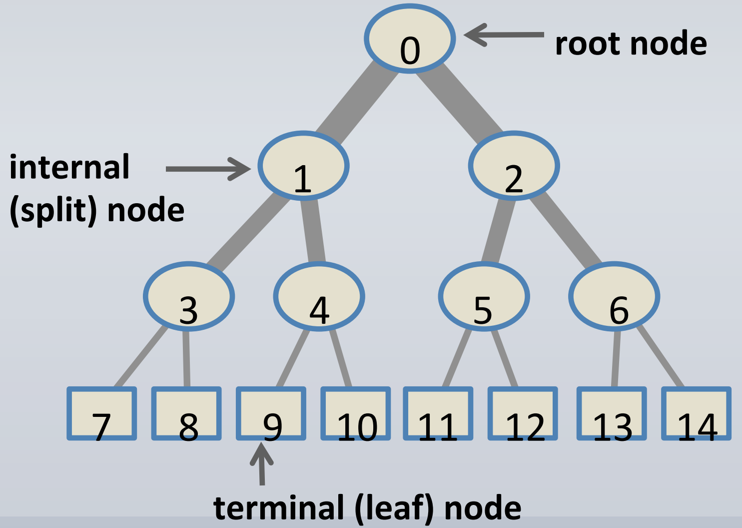
for computer vision and medical image analysis

A. Criminisi, J. Shotton and E. Konukoglu

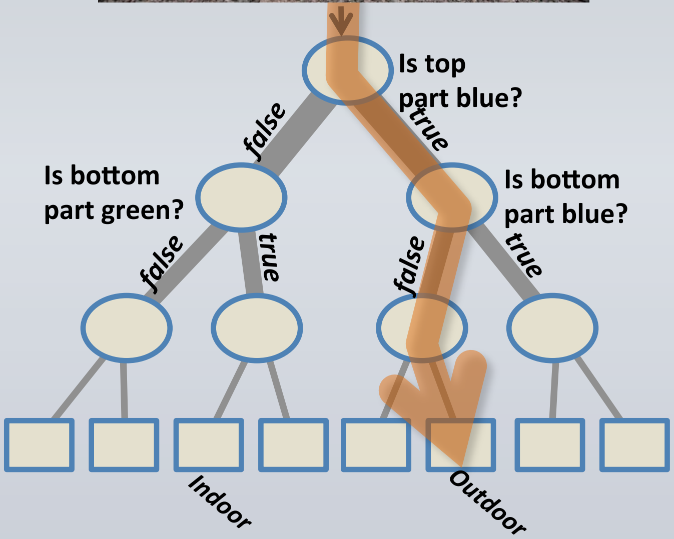
http://research.microsoft.com/projects/decisionforests

Decision trees and decision forests

A general tree structure

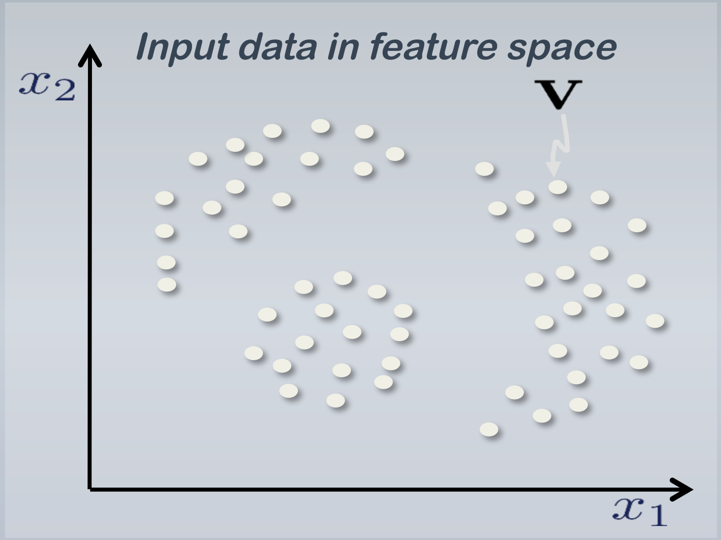


A decision tree

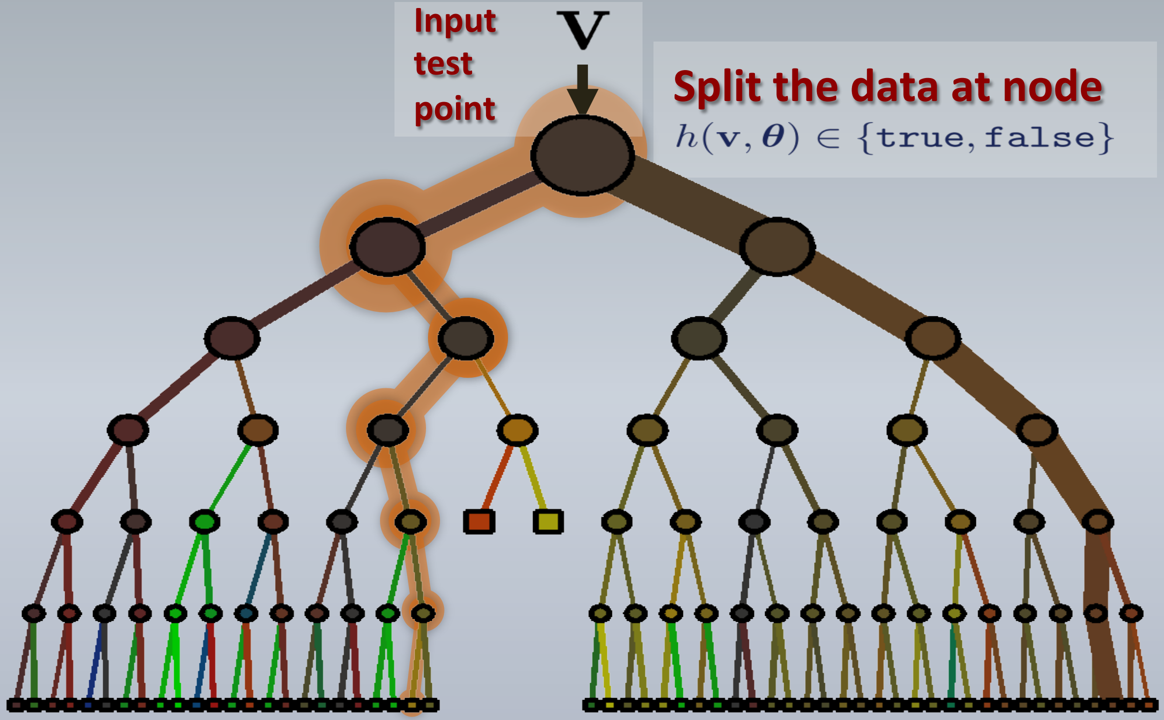


A forest is an ensemble of trees. The trees are all slightly different from one another.

Decision tree testing (runtime)

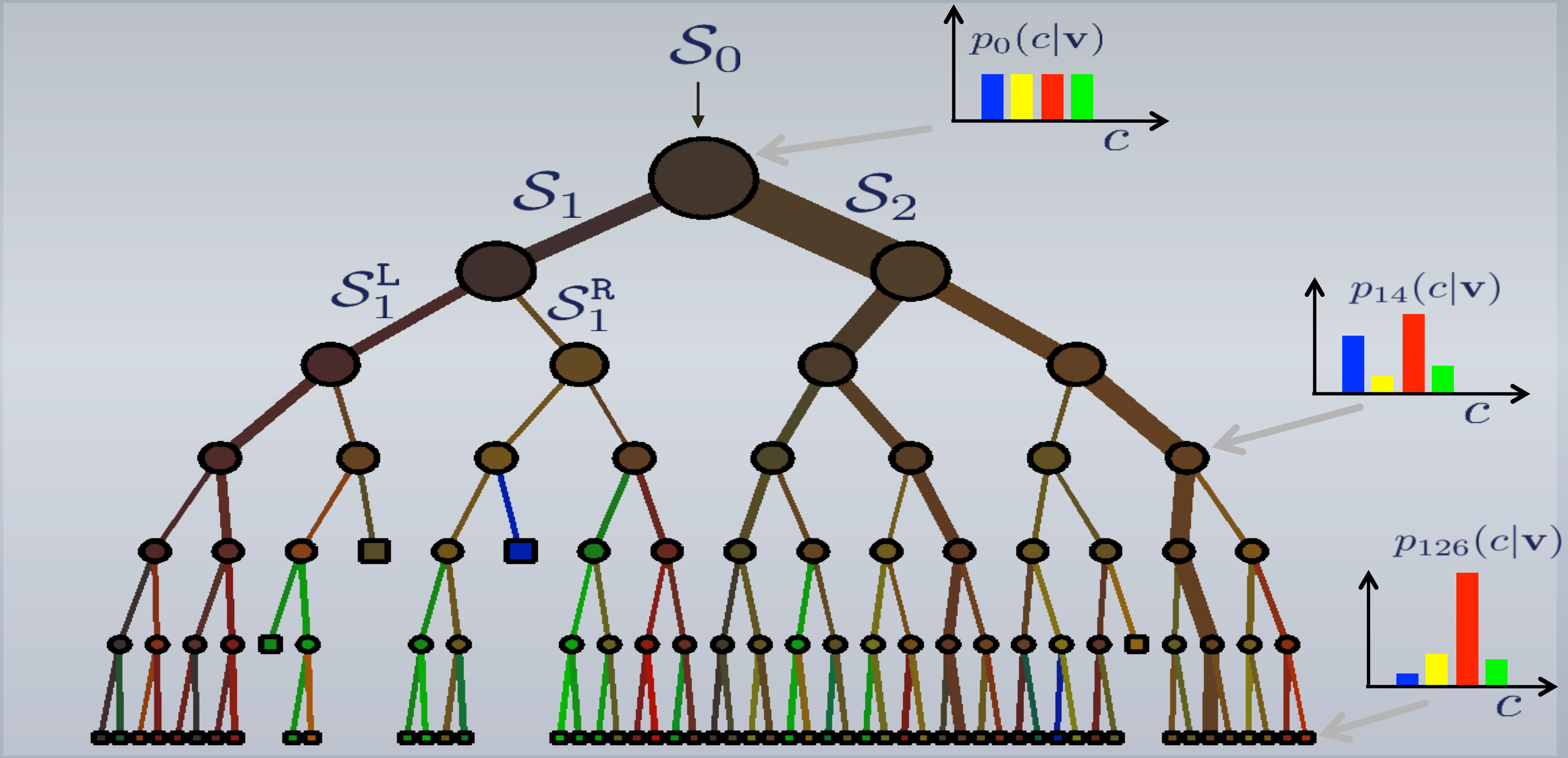


$$\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$$



Split the data at node

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



$\mathcal{S}_0$
$\mathcal{S}_1$
$\mathcal{S}_2$
$\mathcal{S}_1^{\mathrm{L}}$
$\mathcal{S}_1^{\mathrm{R}}$
$p_0(c|\mathbf{v})$
c
$p_{14}(c|\mathbf{v})$
c
$p_{126}(c|\mathbf{v})$
c

Decision tree training (off-line)

How to split the data?

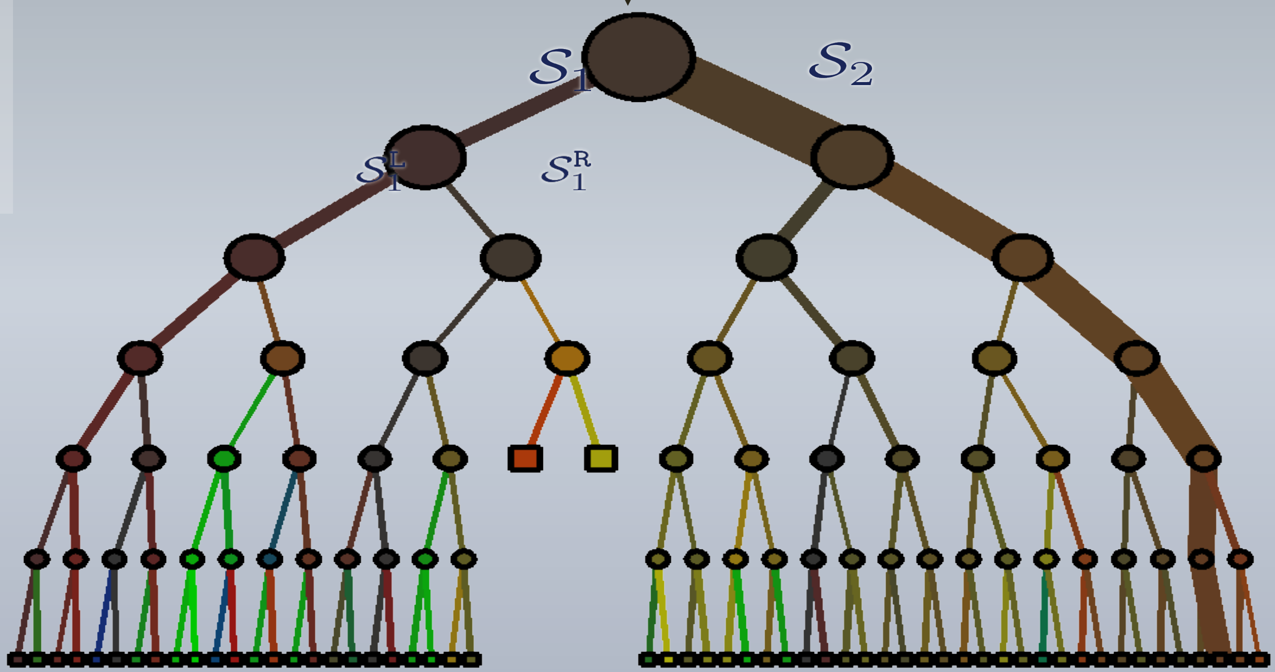
$$h(\mathbf{v}, \boldsymbol{\theta}) \in \{\texttt{true}, \texttt{false}\}$$

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

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

Input training data

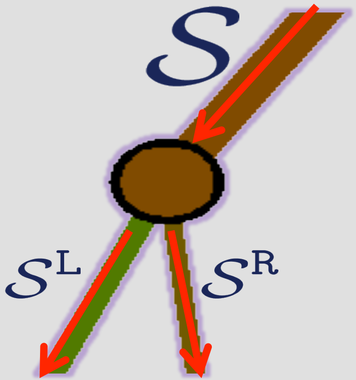
$\mathcal{S}_0 = \{\mathbf{v}, c\}$



Binary tree? Ternary?
How big a tree?

Training and information gain

(for categorical, non-parametric distributions)



Information gain

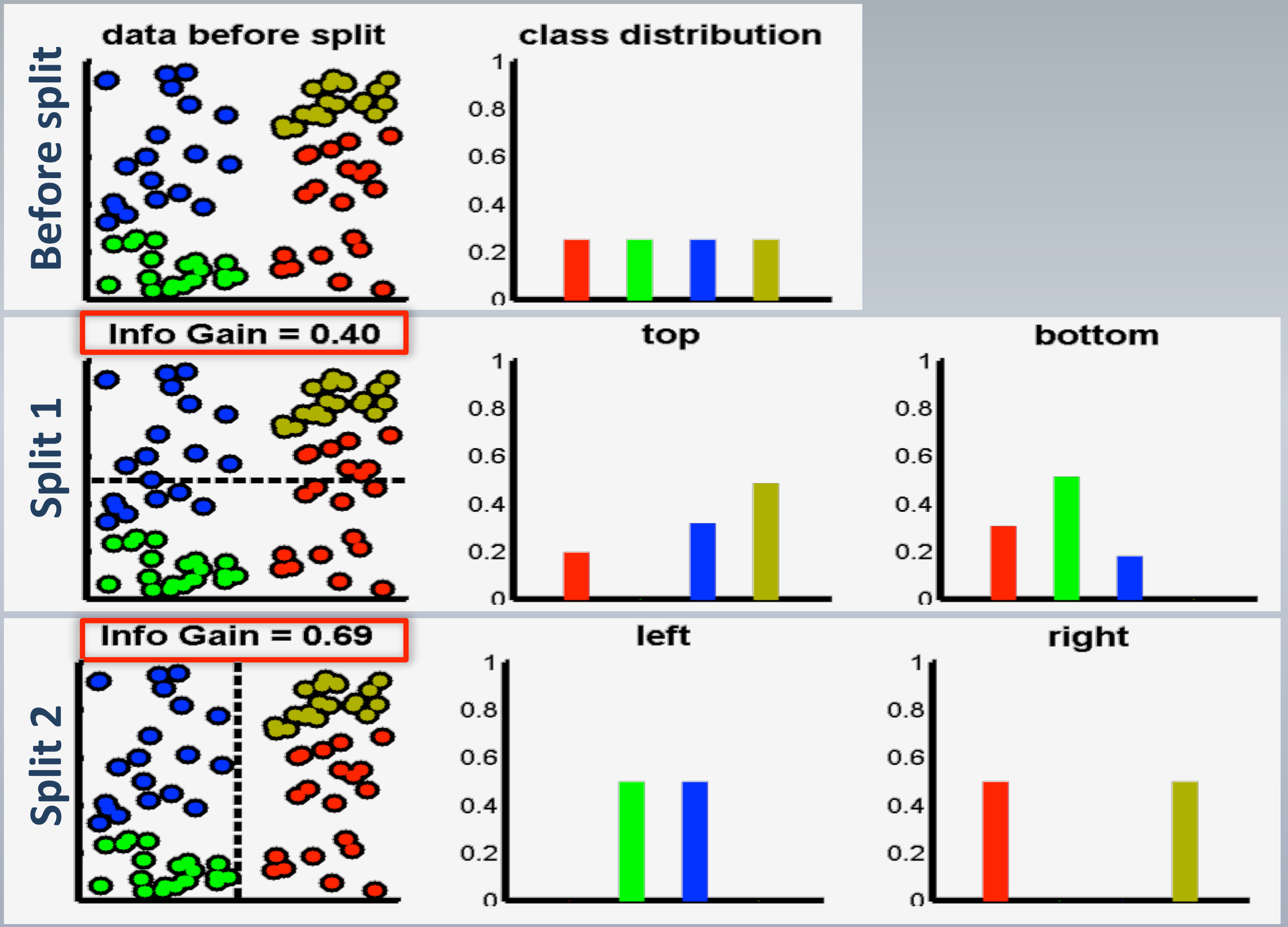
$$I(\mathcal{S},\boldsymbol{\theta}) = H(\mathcal{S}) - \sum_{i\in\{\mathrm{L},\mathrm{R}\}} \frac{|\mathcal{S}^i|}{|\mathcal{S}|} H(\mathcal{S}^i)$$

Shannon's entropy

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

Node training

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

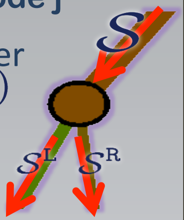


The weak learner model

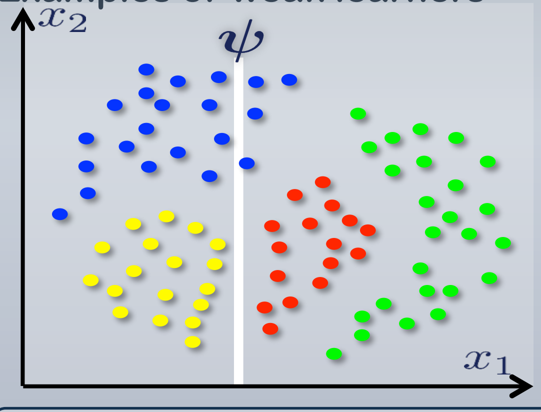
Splitting data at node j

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

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



Examples of weak learners

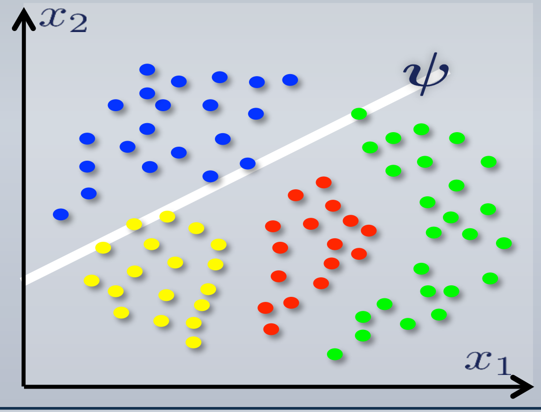


Weak learner: axis aligned

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

Feature response for 2D example. $\boldsymbol{\phi}(\mathbf{v}) = (x_1 \ x_2 \ 1)^\top$

With $\psi = (1 \ 0 \ \psi_3)$ or $\psi = (0 \ 1 \ \psi_3)$

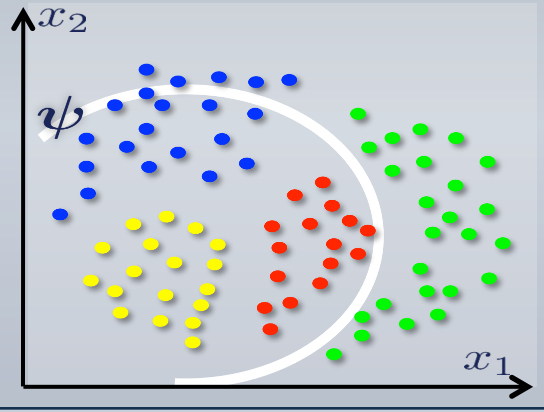


Weak learner: oriented line

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

Feature response for 2D example. $\boldsymbol{\phi}(\mathbf{v}) = (x_1 \ x_2 \ 1)^\top$

With $\psi \in \mathbb{R}^3$ a generic line in homog. coordinates.



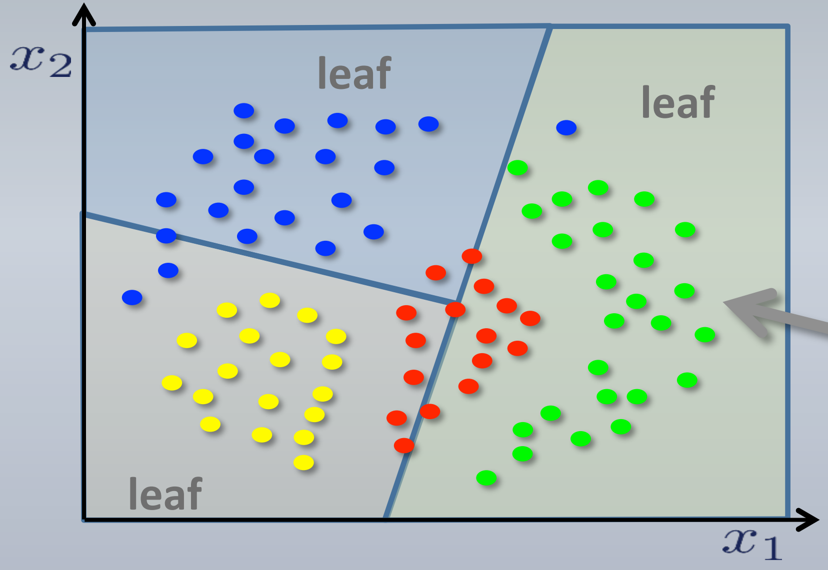
Weak learner: conic section

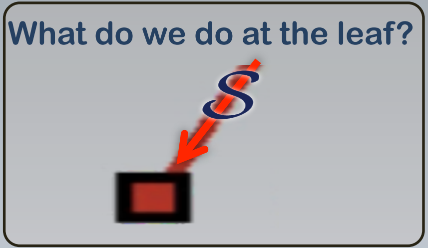
$$h(\mathbf{v}, \boldsymbol{\theta}) = \left[\tau_1 > \boldsymbol{\phi}^\top(\mathbf{v}) \ \boldsymbol{\psi} \ \boldsymbol{\phi}(\mathbf{v}) > \tau_2\right]$$

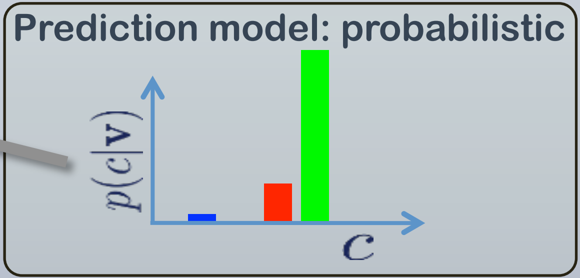
Feature response for 2D example. $\boldsymbol{\phi}(\mathbf{v}) = (x_1 \ x_2 \ 1)^\top$

With $\psi \in \mathbb{R}^{3\times 3}$ a matrix representing a conic.

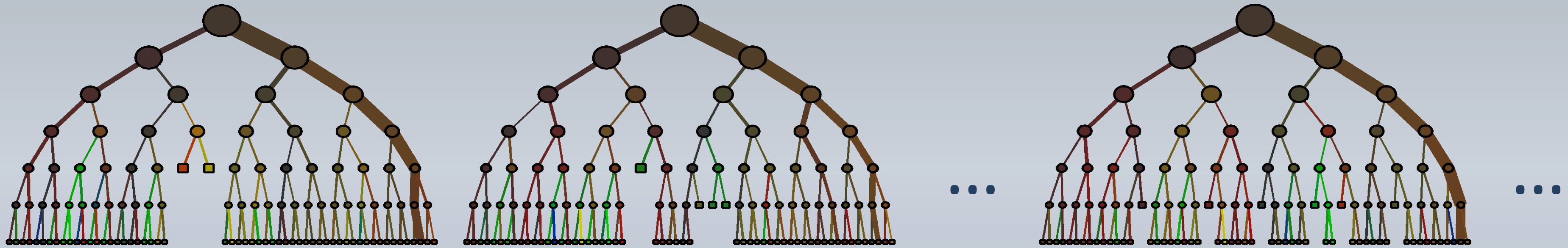
The prediction model







Decision forest training (off-line)



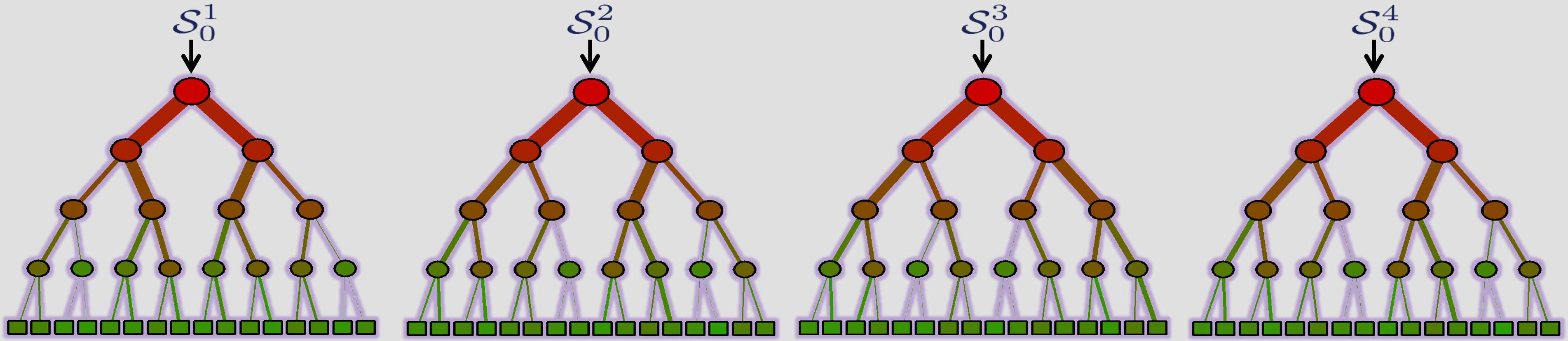
How many trees?
How different?
How to fuse their outputs?

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



Efficient training

Decision forest model: the randomness model

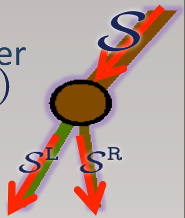
2) Randomized node optimization (RNO)

$\mathcal{T}$	The full set of all possible node test parameters
$\mathcal{T}_j \subset \mathcal{T}$	For each node the set of randomly sampled features
$\rho = \lvert\mathcal{T}_j\rvert$	Randomness control parameter. For $\rho = \lvert\mathcal{T}\rvert$ no randomness and maximum tree correlation. For $\rho = 1$ max randomness and minimum tree correlation.

Node training

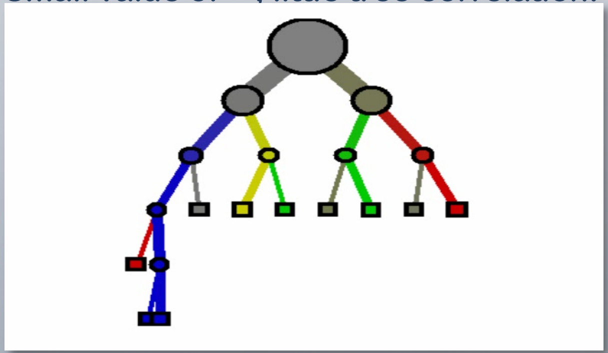
Node weak learner $h(\mathbf{v}, \boldsymbol{\theta}_j)$

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

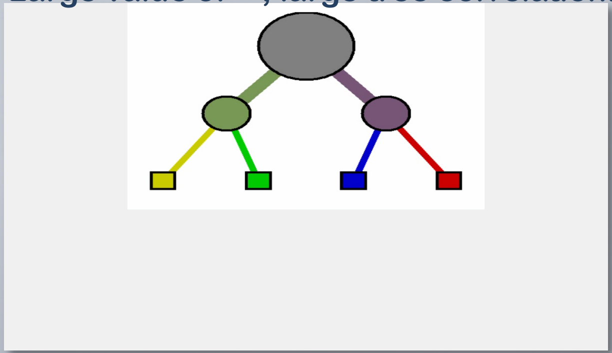


The effect of ρ

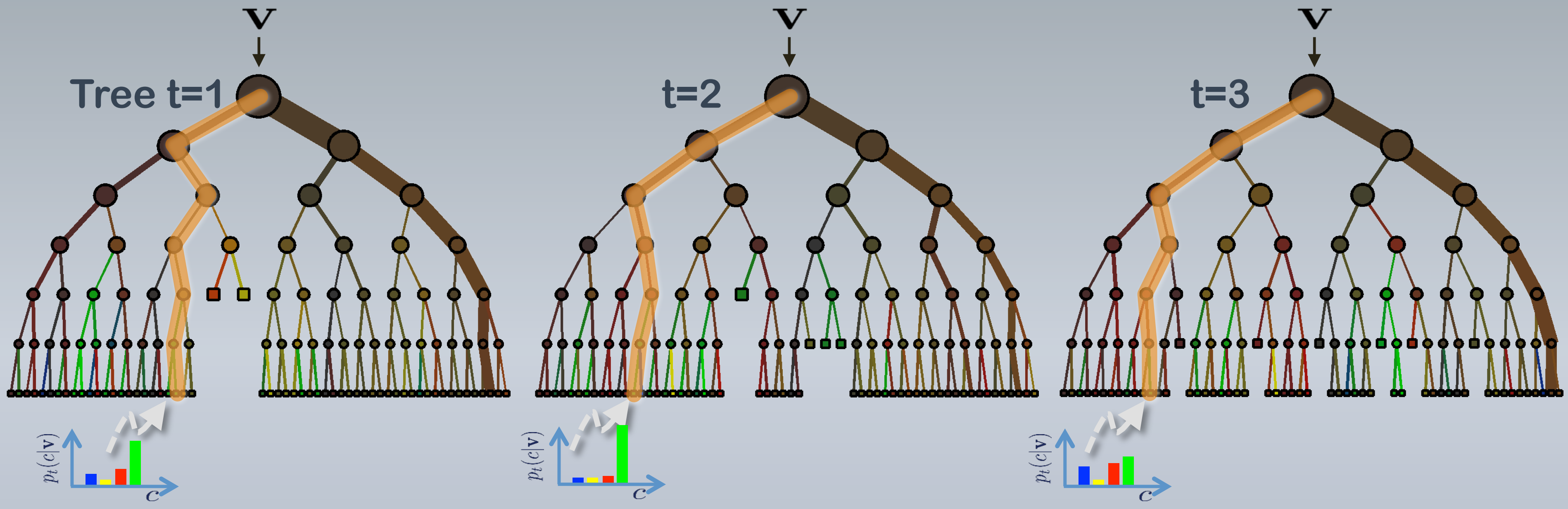
Small value of ρ; little tree correlation.



Large value of ρ; large tree correlation.

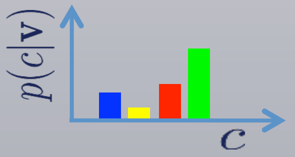


Classification forest: the ensemble model

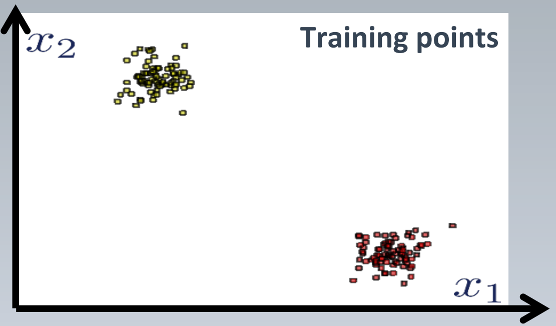


The ensemble model

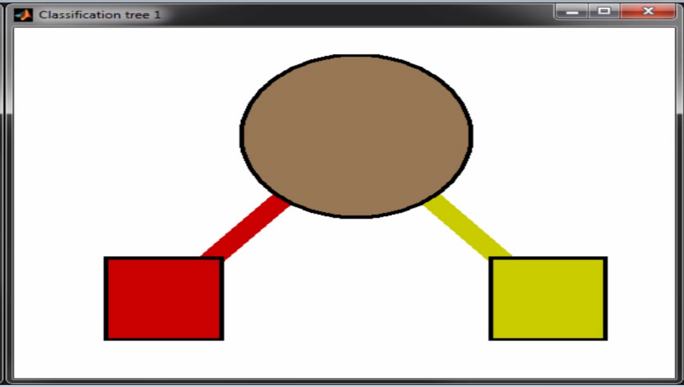
Forest output probability $$p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$$



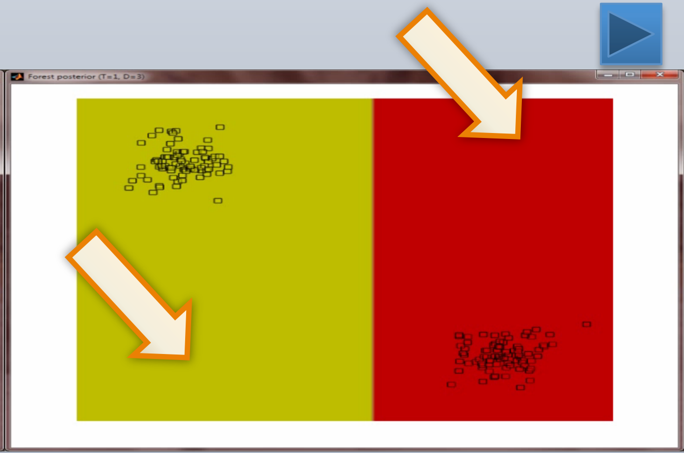
Classification forest: effect of the weak learner model



Training different trees in the forest



Testing different trees in the forest



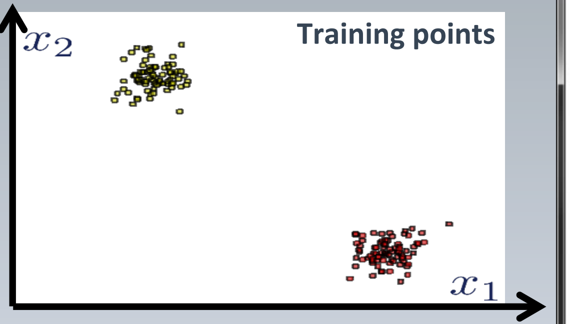
Three concepts to keep in mind:

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

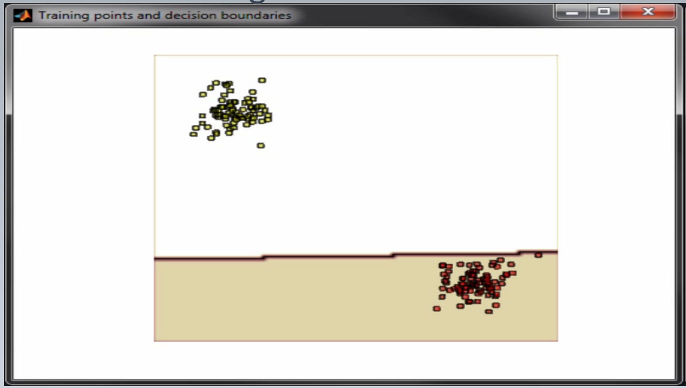
(2 videos in this page)

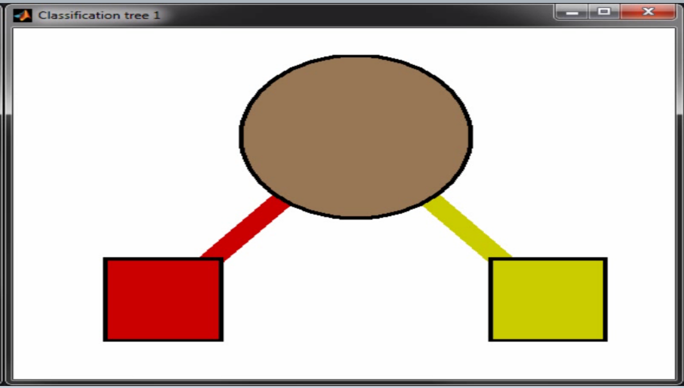
Parameters: T=200, D=2, weak learner = aligned, leaf model = probabilistic

Classification forest: effect of the weak learner model

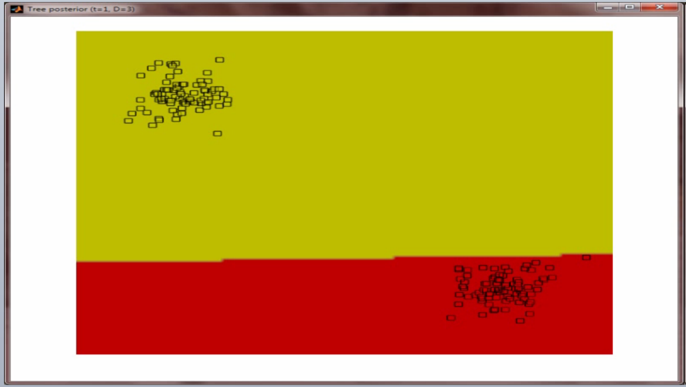


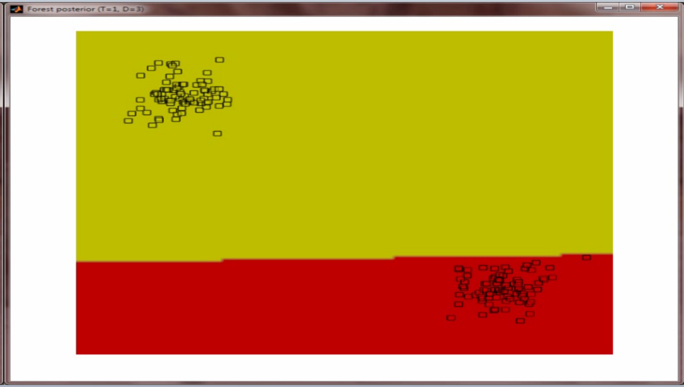
Training different trees in the forest





Testing different trees in the forest

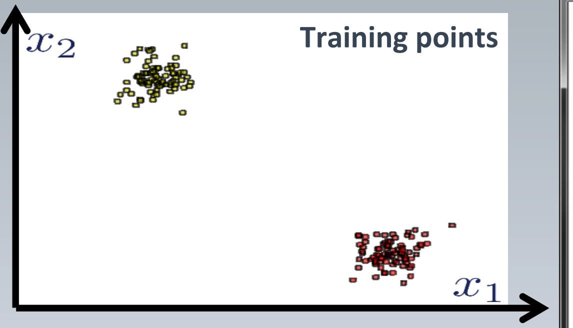




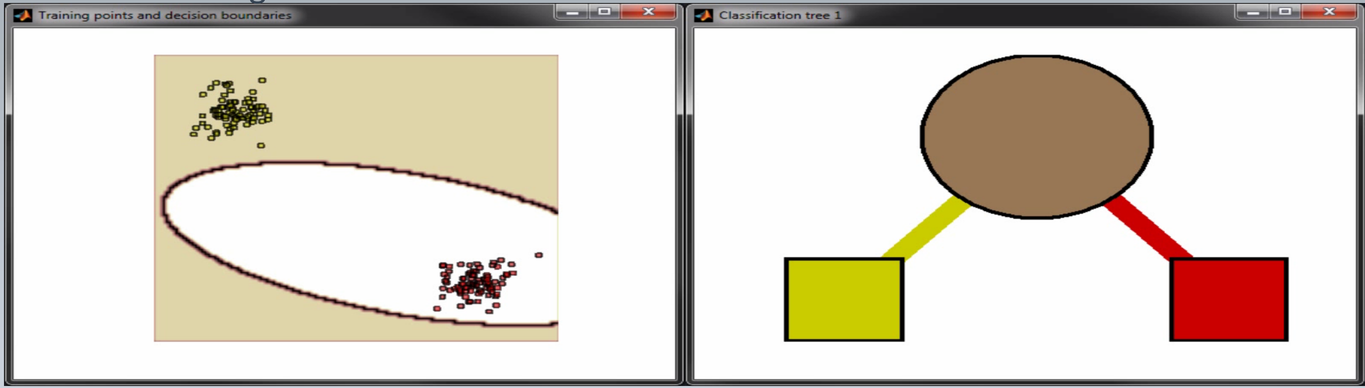
(2 videos in this page)

Parameters: T=200, D=2, weak learner = linear, leaf model = probabilistic

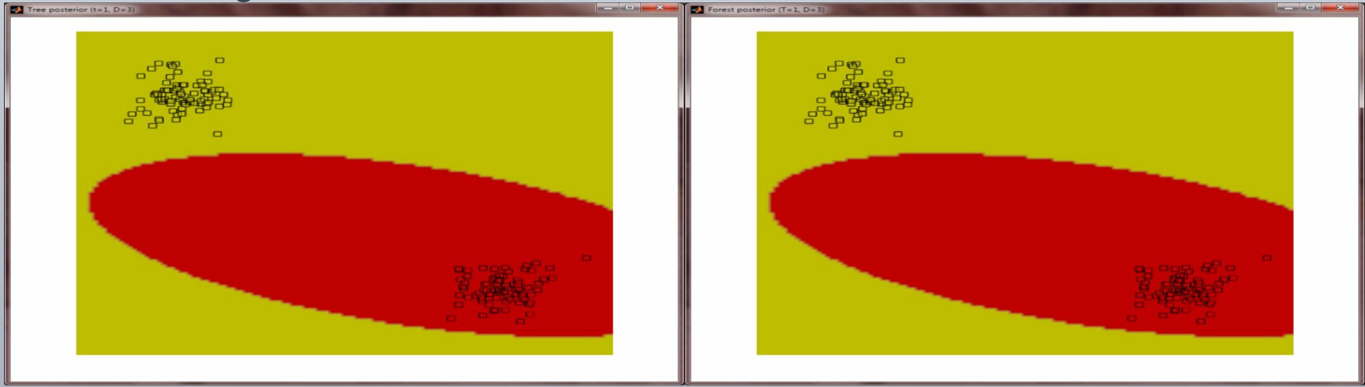
Classification forest: effect of the weak learner model



Training different trees in the forest



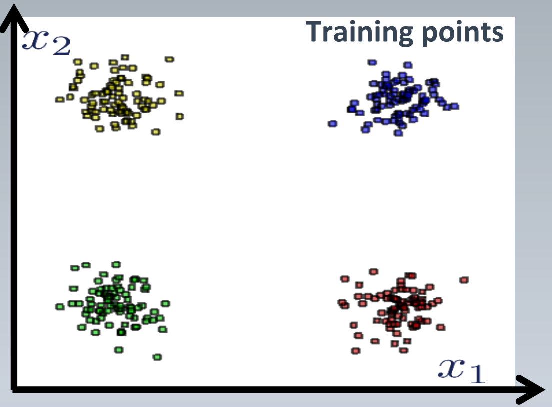
Testing different trees in the forest



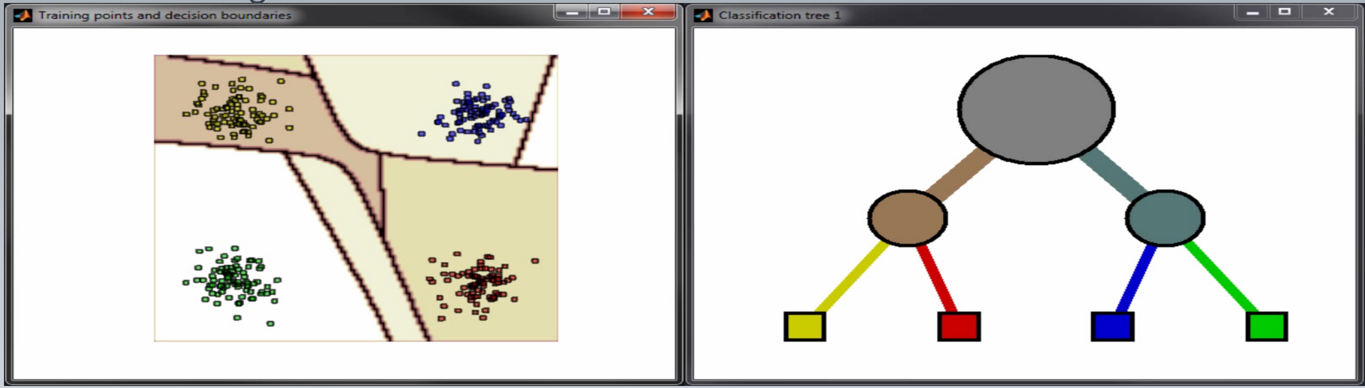
(2 videos in this page)

Parameters: T=200, D=2, weak learner = conic, leaf model = probabilistic

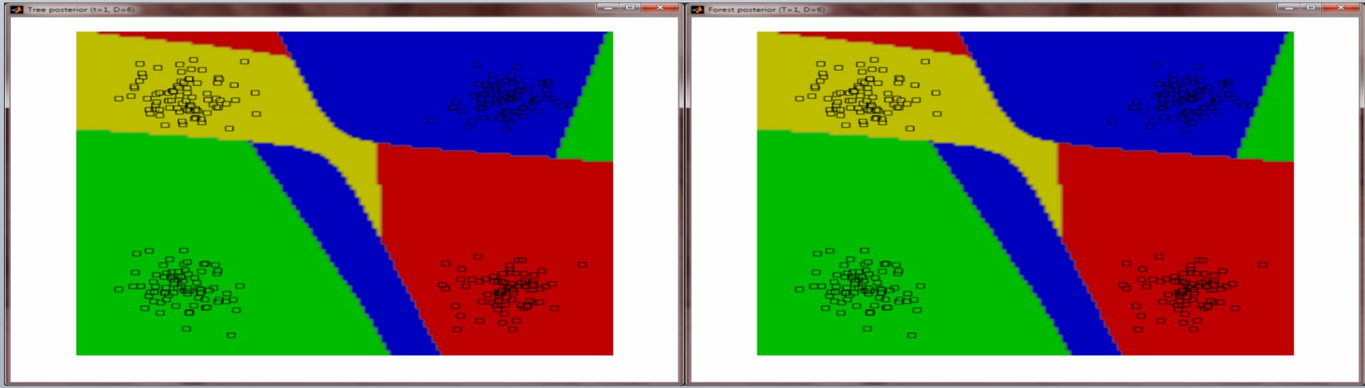
Classification forest: with >2 classes



Training different trees in the forest

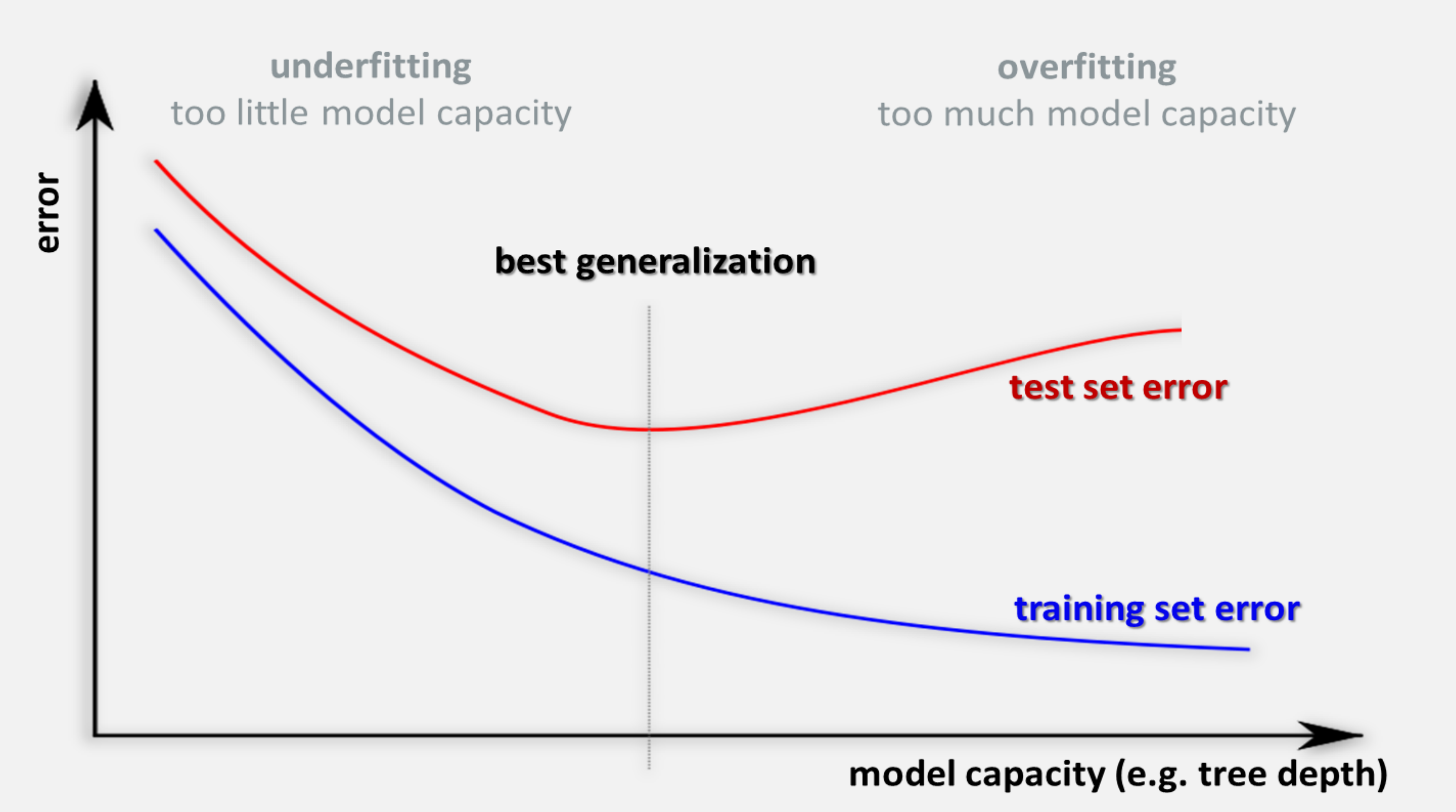


Testing different trees in the forest

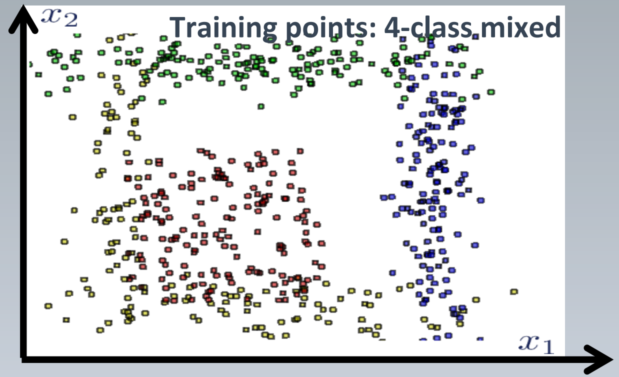


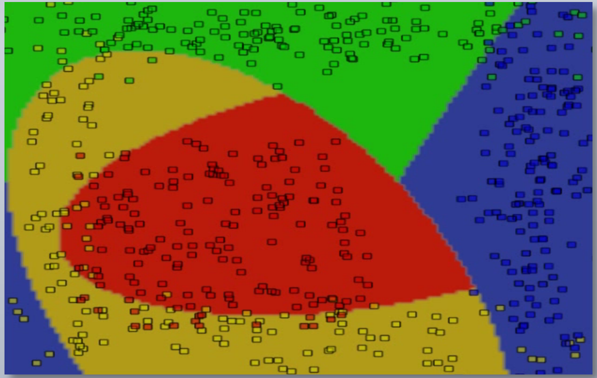
(2 videos in this page)

Parameters: T=200, D=3, weak learner = conic, leaf model = probabilistic

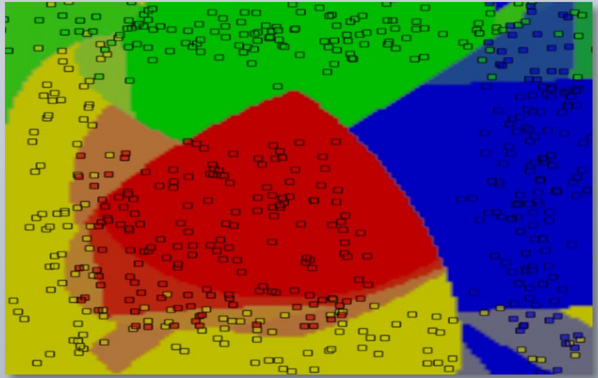

underfitting
too little model capacity
overfitting
too much model capacity
error
best generalization
test set error
training set error
model capacity (e.g. tree depth)

Classification forest: effect of tree depth

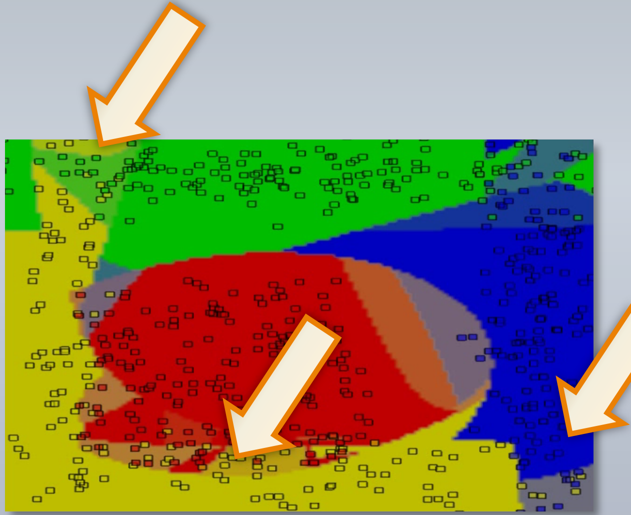




T=200, D=3, w. l. = conic



T=200, D=6, w. l. = conic



T=200, D=15, w. l. = conic

max tree depth, D

underfitting

overfitting

(3 videos in this page)

Predictor model = prob.

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

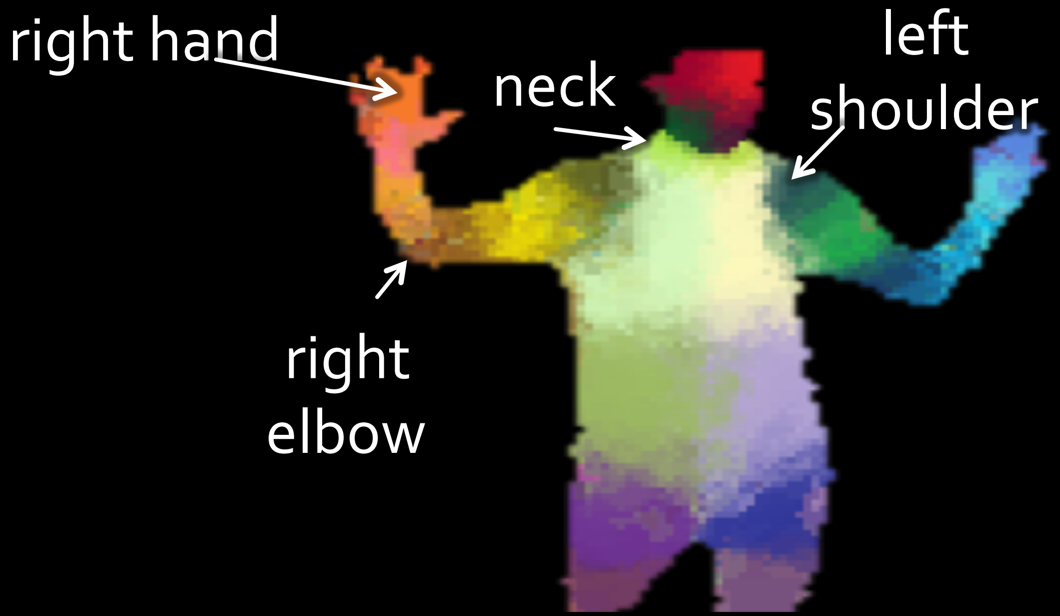
Jamie Shotton, Andrew Fitzgibbon, Mat Cook,
Toby Sharp, Mark Finocchio, Richard Moore,
Alex Kipman, Andrew Blake

CVPR 2011

Microsoft Research

XBOX 360

Body part recognition



Body part recognition

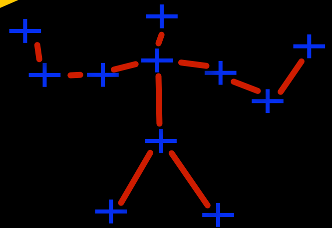
- No temporal information
  - frame-by-frame
- Local pose estimate of parts
  - each pixel & each body joint treated independently
- Very fast
  - simple depth image features
  - parallel decision forest classifier

The Kinect pose estimation pipeline

capture
depth image &
remove bg

infer
body parts
per pixel

cluster pixels to
hypothesize
body joint
positions



fit model &
track skeleton

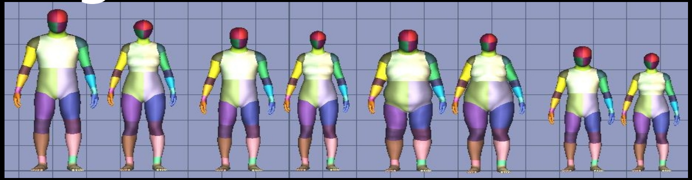
Synthetic training data

Record mocap

500k frames

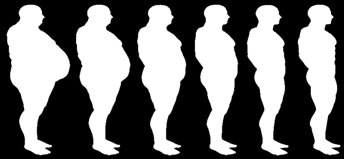
distilled to 100k poses

Retarget to several models



Render (depth, body parts) pairs

Train invariance to:



Synthetic vs. real data

synthetic
(train & test)

real
(test)

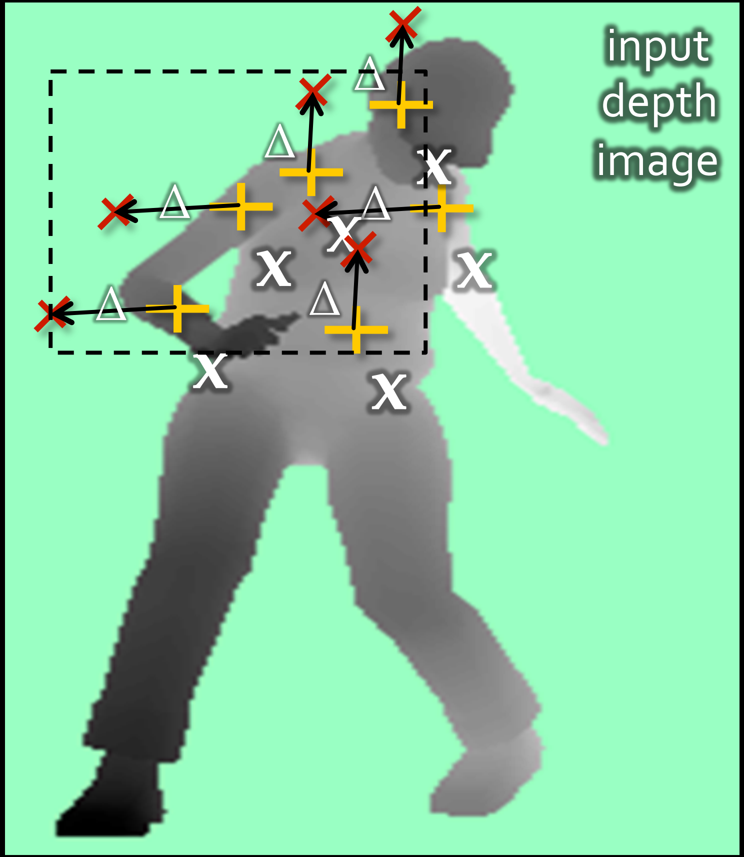
Fast depth image features

- Depth comparisons
  - very fast to compute

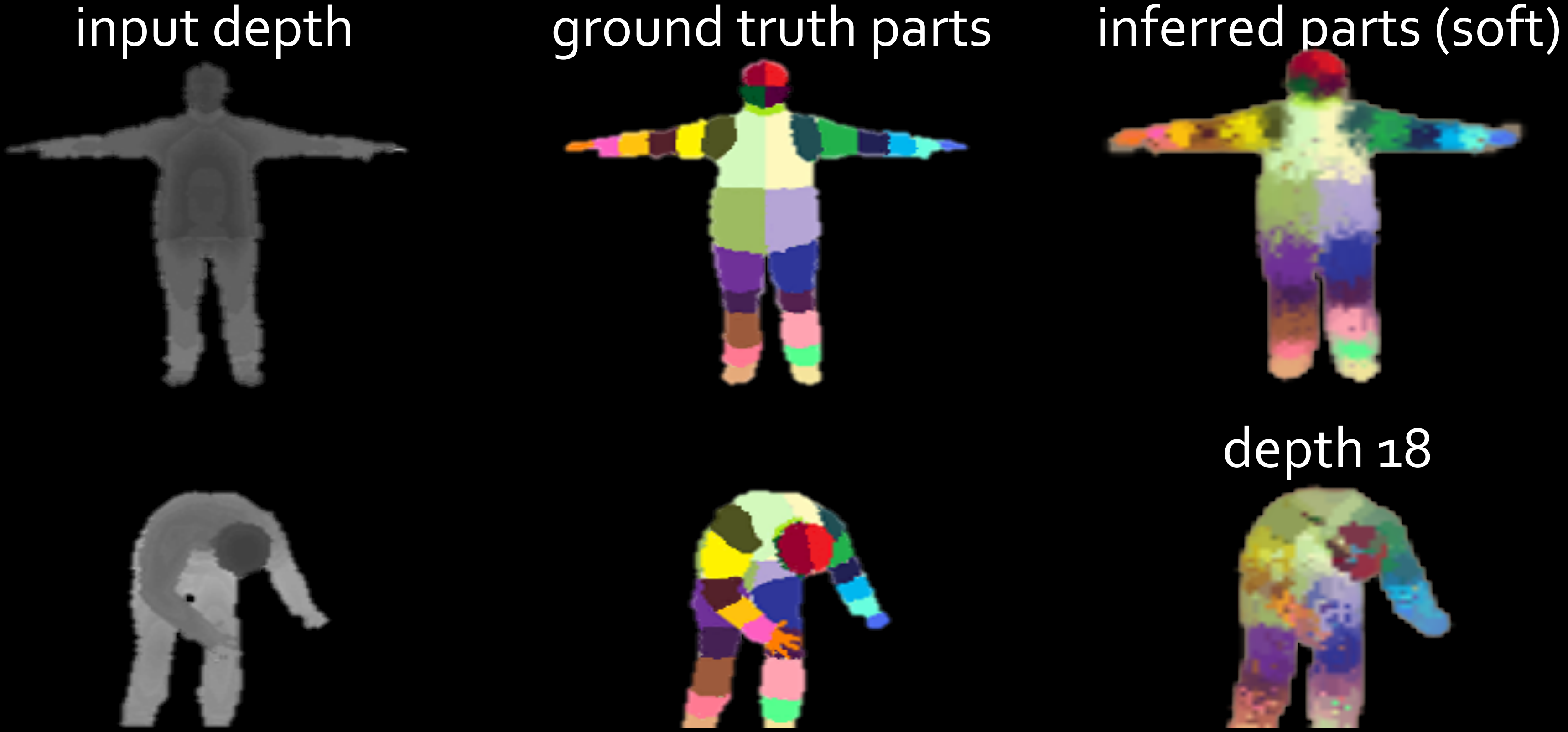
$$\underbrace{f(\overbrace{I}^{\text{image}}, \underbrace{x}_{\text{image coordinate}})}_{\text{feature response}} = \overbrace{d_I(x)}^{\text{depth}} - \overbrace{d_I(x+\Delta)}^{\text{offset depth}}$$

$$\Delta = \underbrace{v/d_I(x)}_{\text{scales inversely with depth}}$$

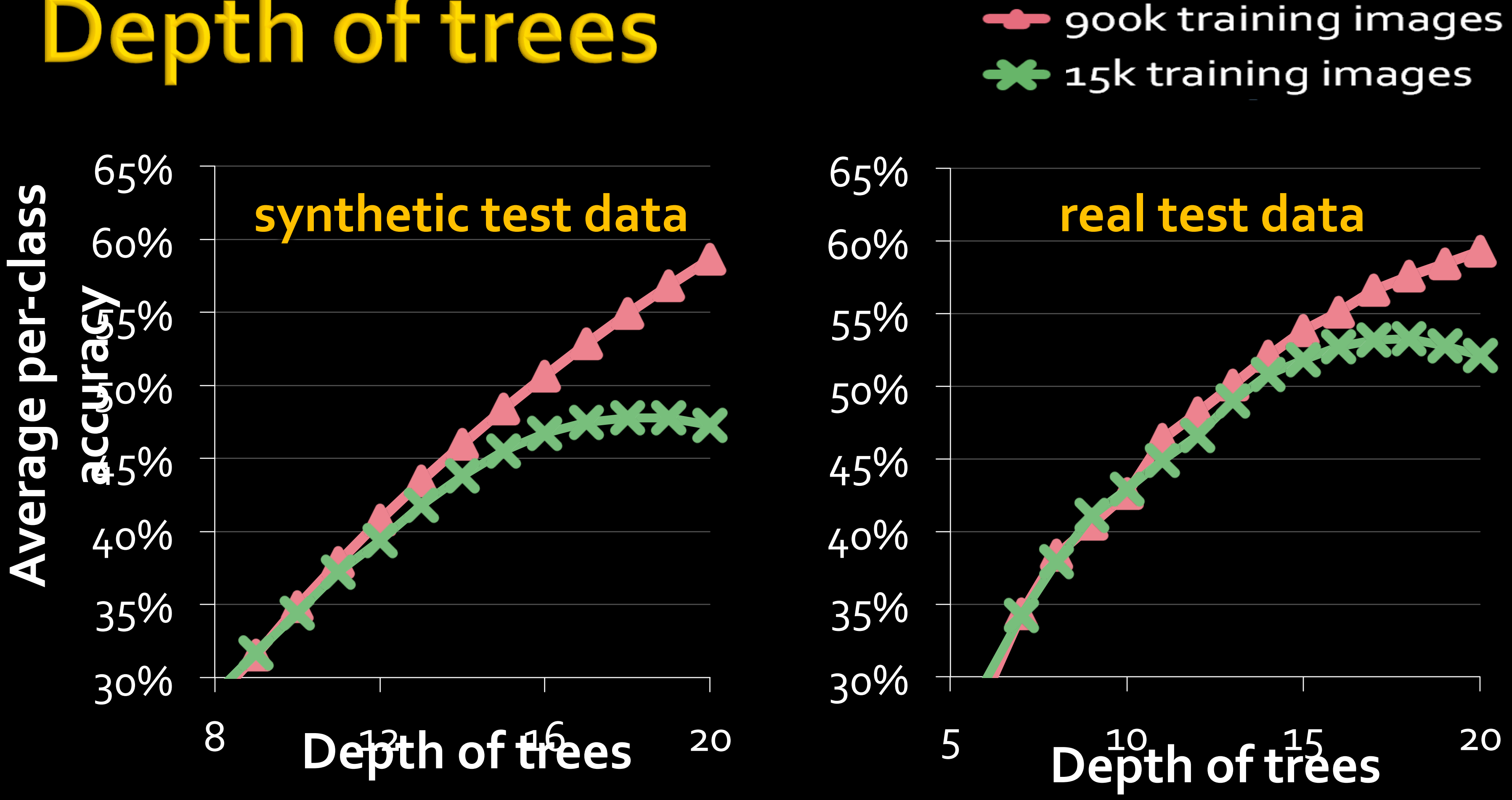
Background pixels
d = large constant



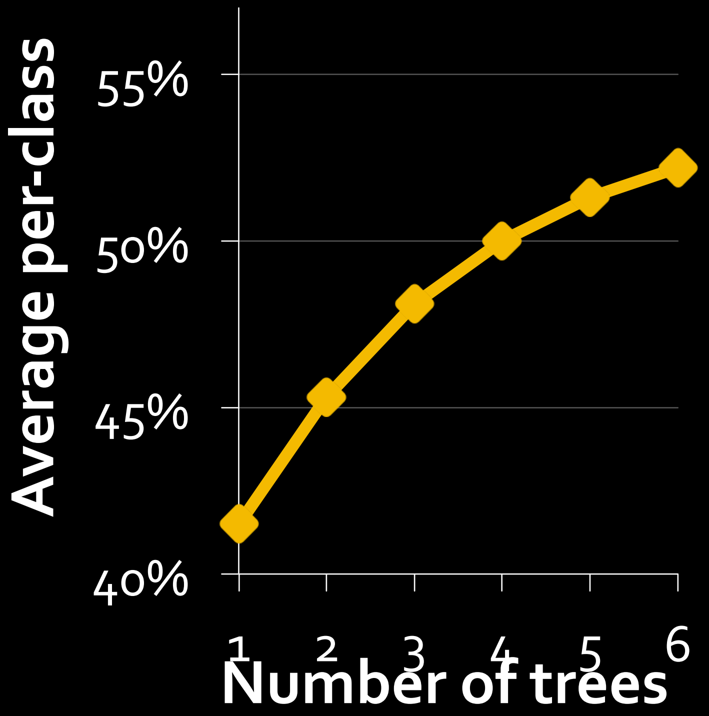
Depth of trees

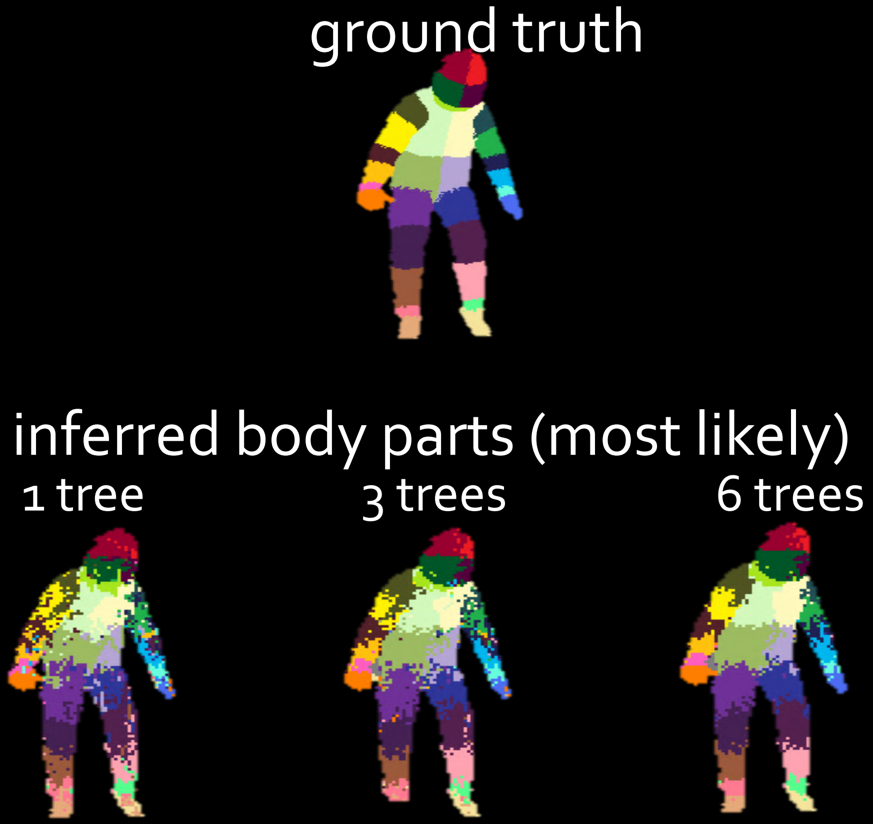


Depth of trees

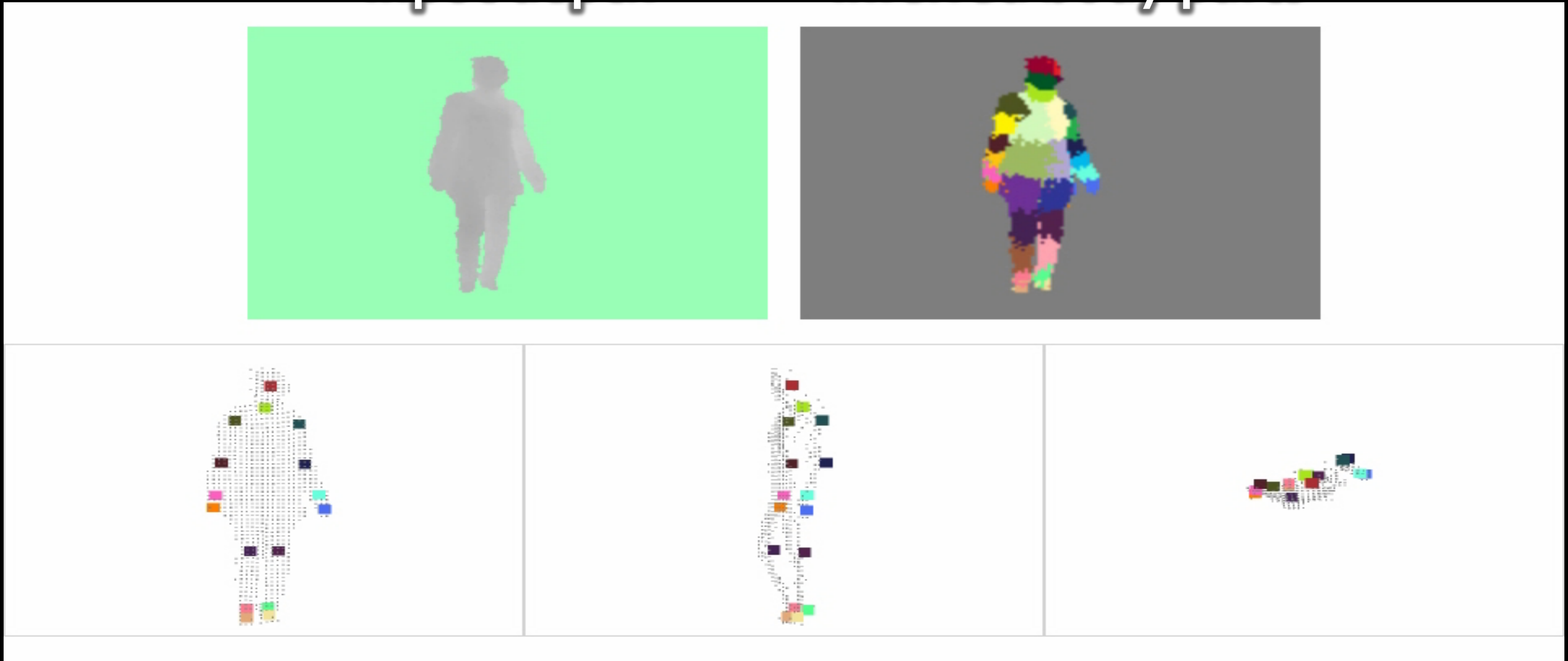


Number of trees





input depth inferred body parts

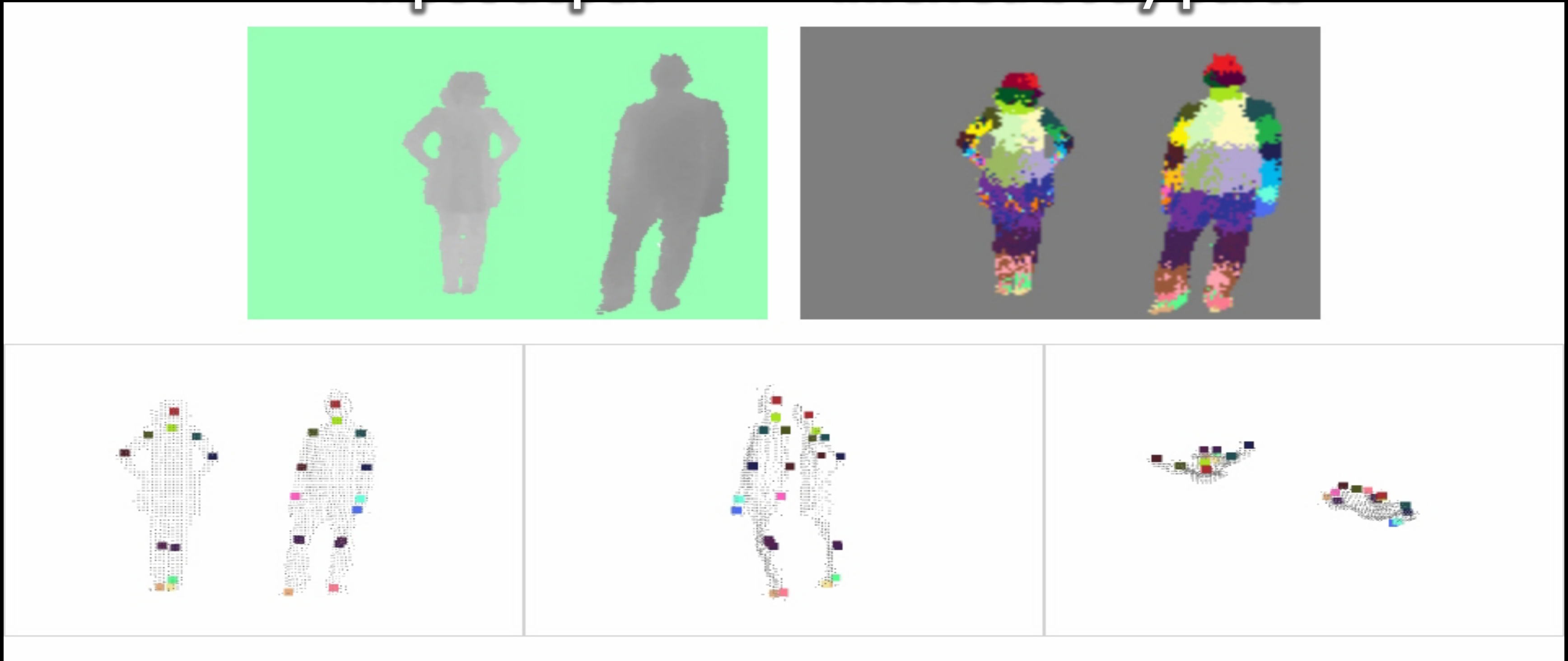


front view side view top view

inferred joint positions

no tracking or smoothing

input depth inferred body parts



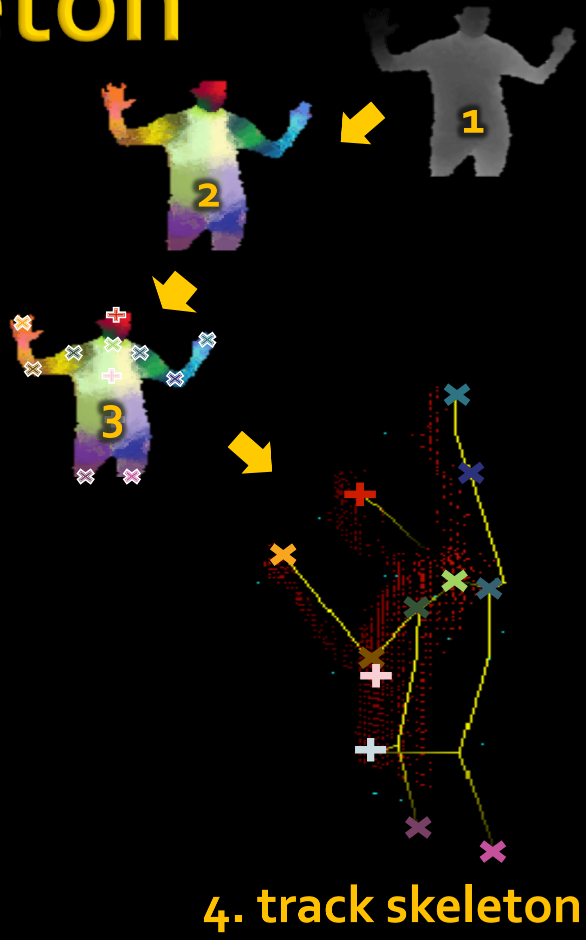
front view side view top view

inferred joint positions

no tracking or smoothing

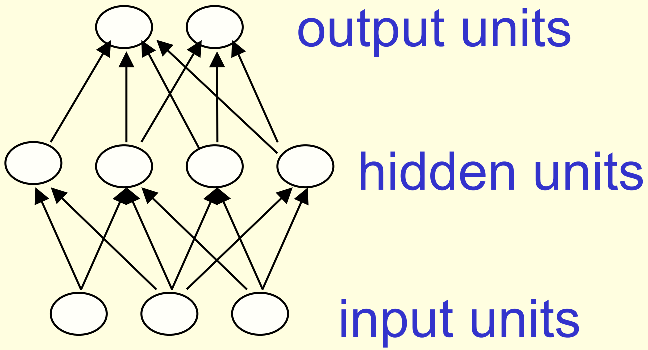
From proposals to skeleton

- 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

bias

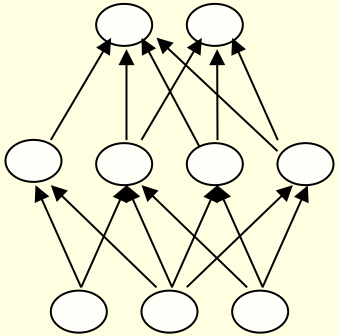
i th input

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

output

index over input connections

weight on i th input

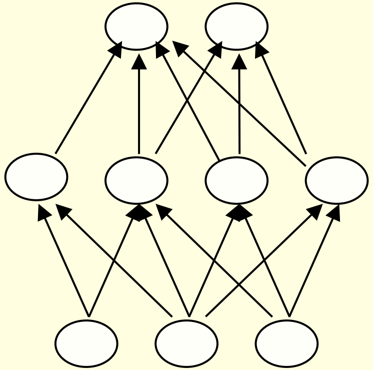


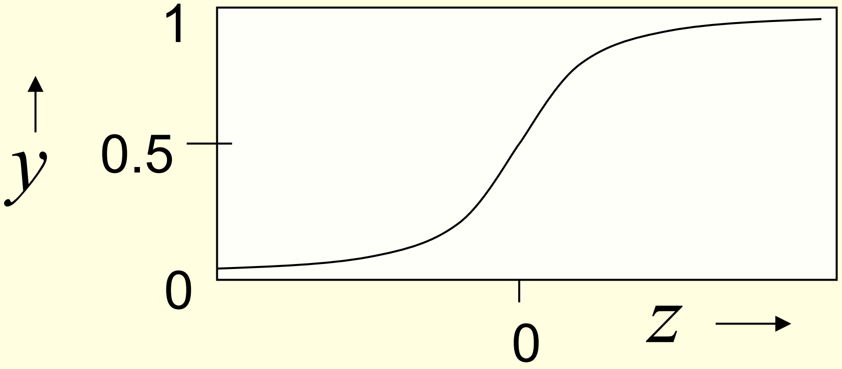
Sigmoid neurons

- 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

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

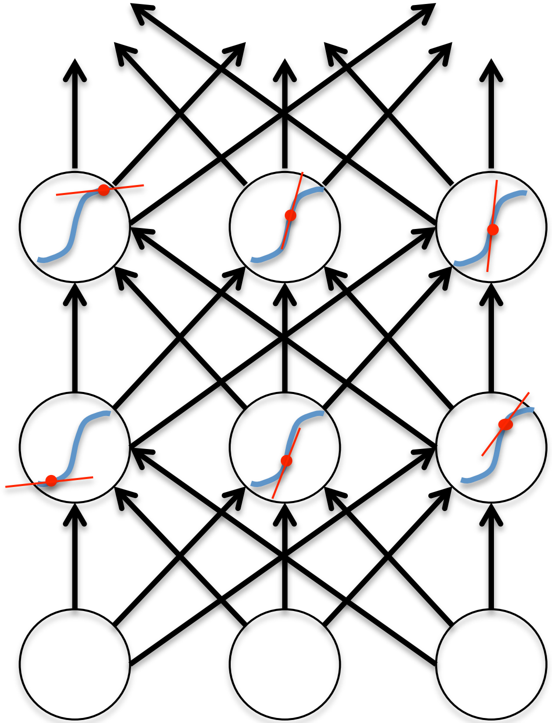
$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{\partial z}{\partial x_i} = w_i \qquad \frac{dy}{dz} = y\,(1-y)$$





Finding weights with backpropagation

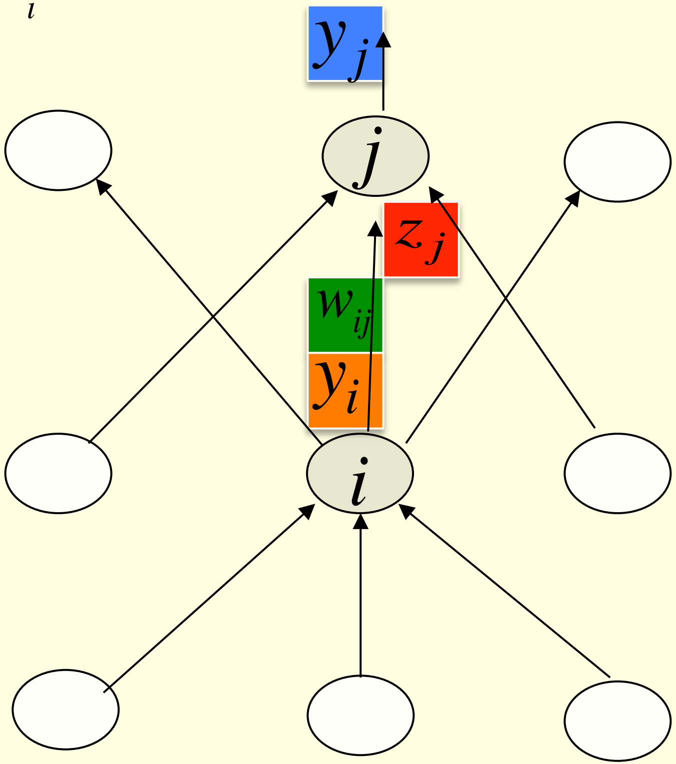
- There is a big difference between the forward and backward passes.
- In the forward pass we use squashing functions to prevent the activity vectors from exploding.
- The backward pass, is completely linear.
  - The forward pass determines the slope of the linear function used for backpropagating through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



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

Backpropagating dE/dy

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



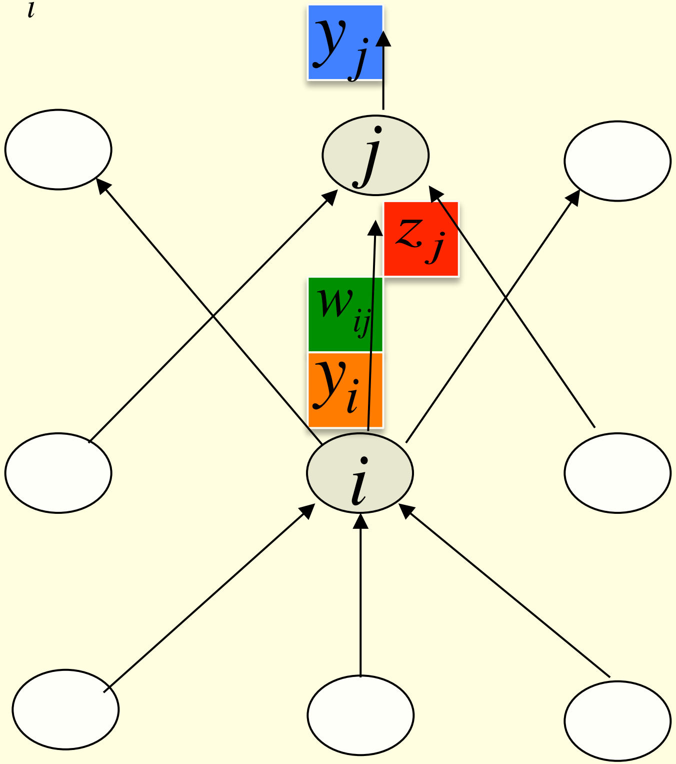
- Find squared error
- Propagate error to the layer below
- Compute error derivative w.r.t. weights
- Repeat

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

Backpropagating dE/dy

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

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



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

Propagate error across non-linearity

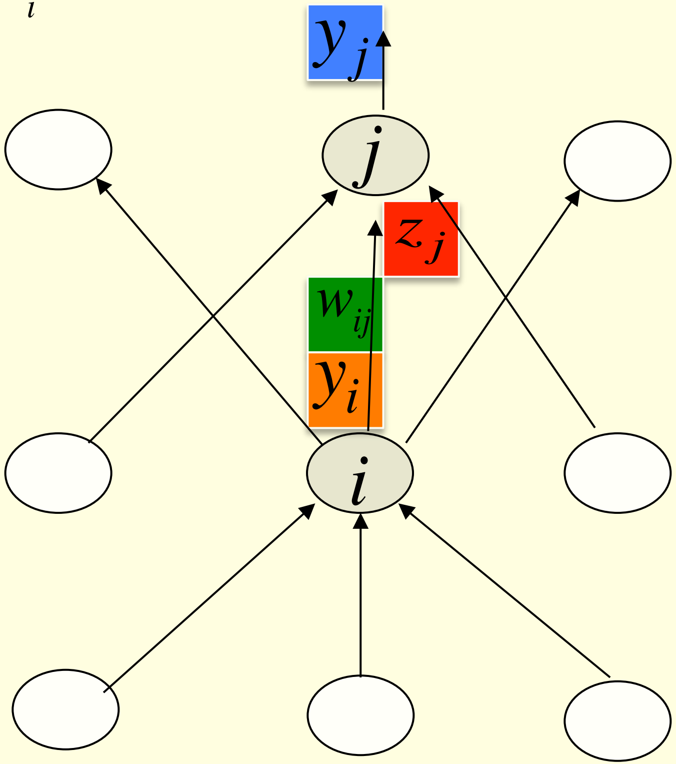
$$y = \frac{1}{1+e^{-z}}$$

Backpropagating dE/dy

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

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

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



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

Propagate error across non-linearity

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

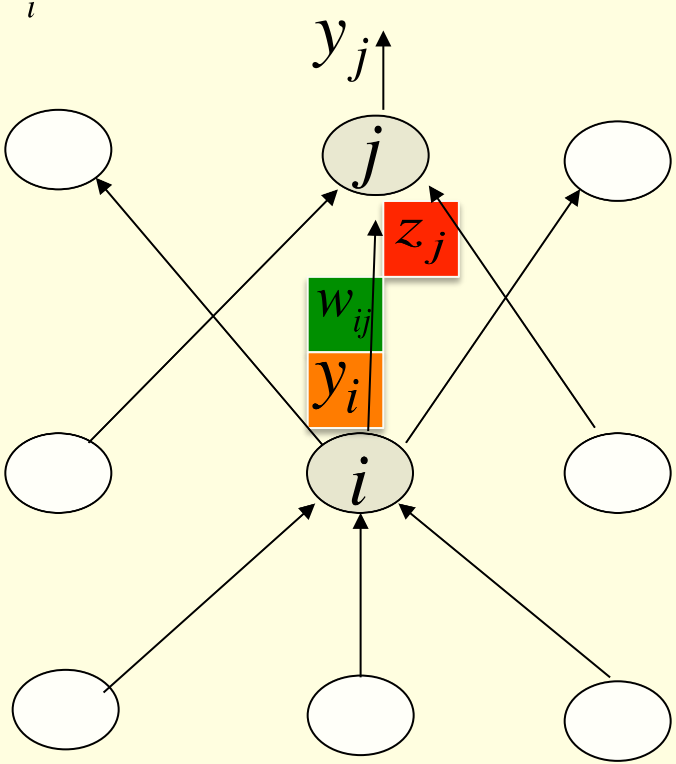
$$\frac{dy}{dz} = y\,(1-y)$$

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

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

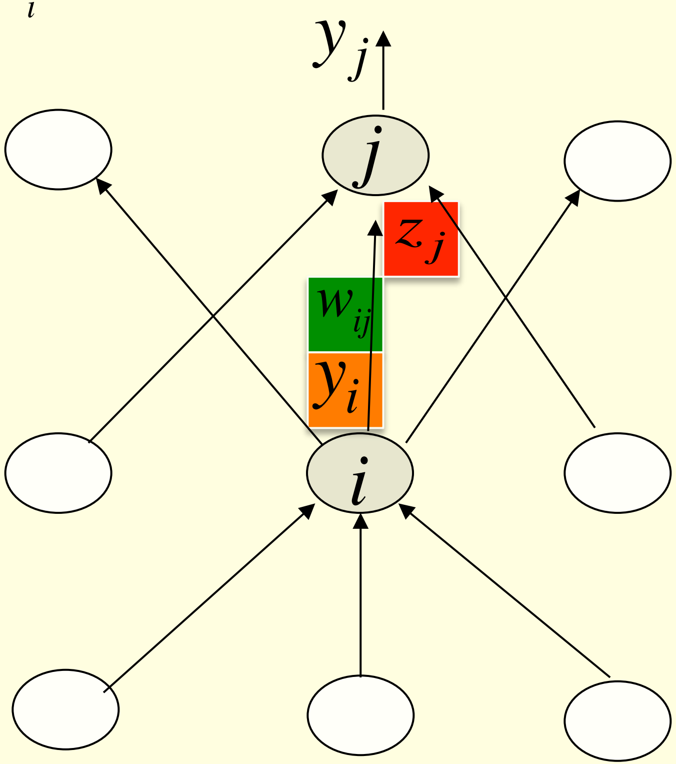
$$y = \frac{1}{1 + e^{-z}} \qquad \frac{dy}{dz} = y\,(1 - y)$$

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

Backpropagating dE/dy

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

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



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

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

Propagate error to the next activation across connections

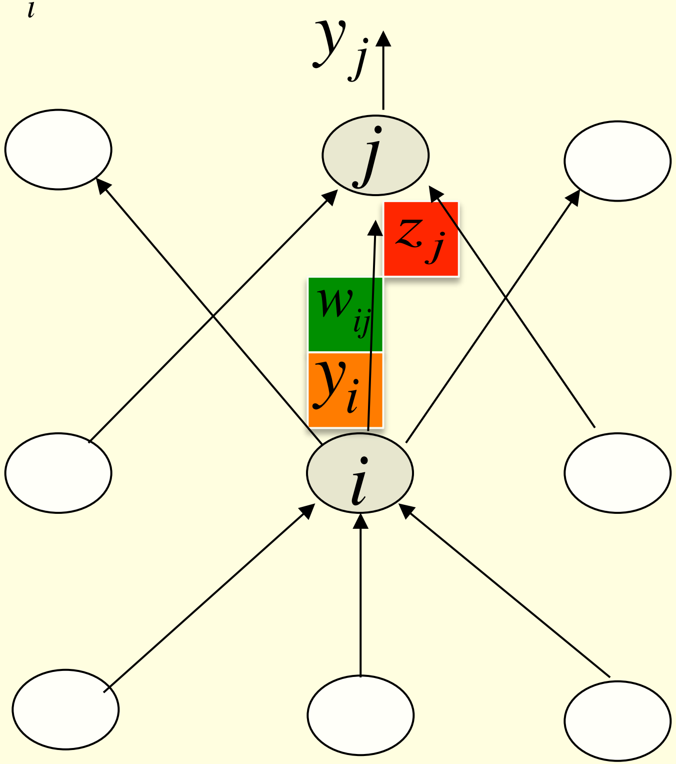
$$y = \frac{1}{1+e^{-z}} \qquad \frac{dy}{dz} = y\,(1-y)$$

Backpropagating dE/dy

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

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

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



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

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

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

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

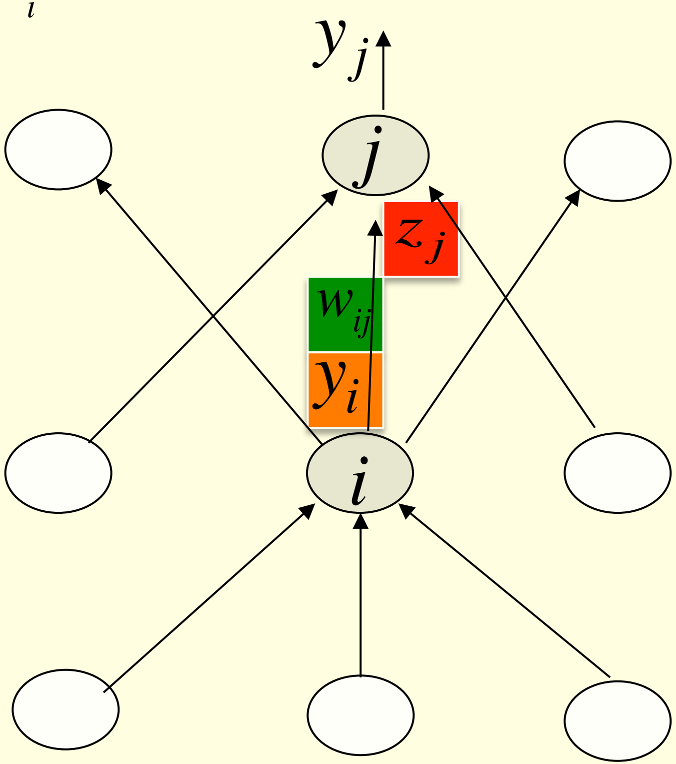
Error gradient w.r.t. weights

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

Backpropagating dE/dy

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

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



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

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

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

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

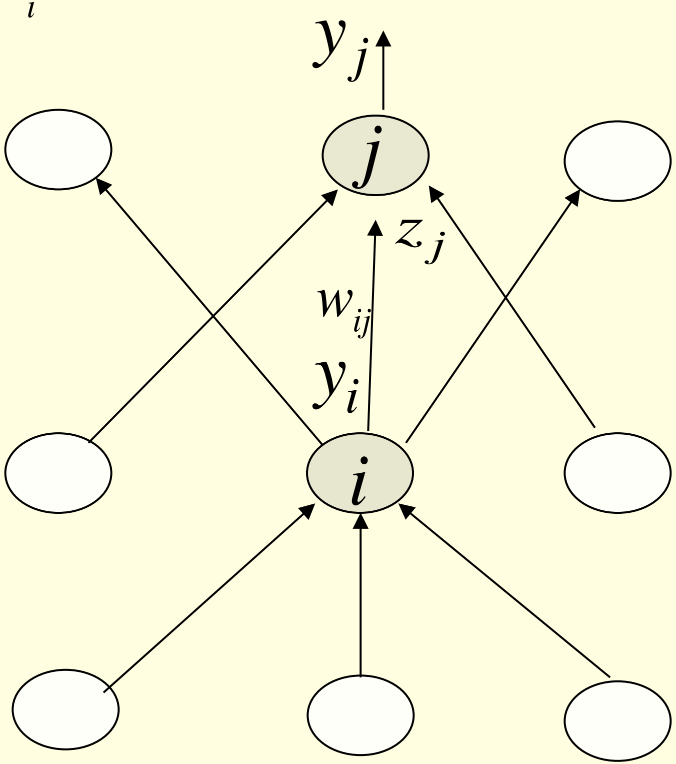
Error gradient w.r.t. weights

Backpropagating dE/dy

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

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

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



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

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

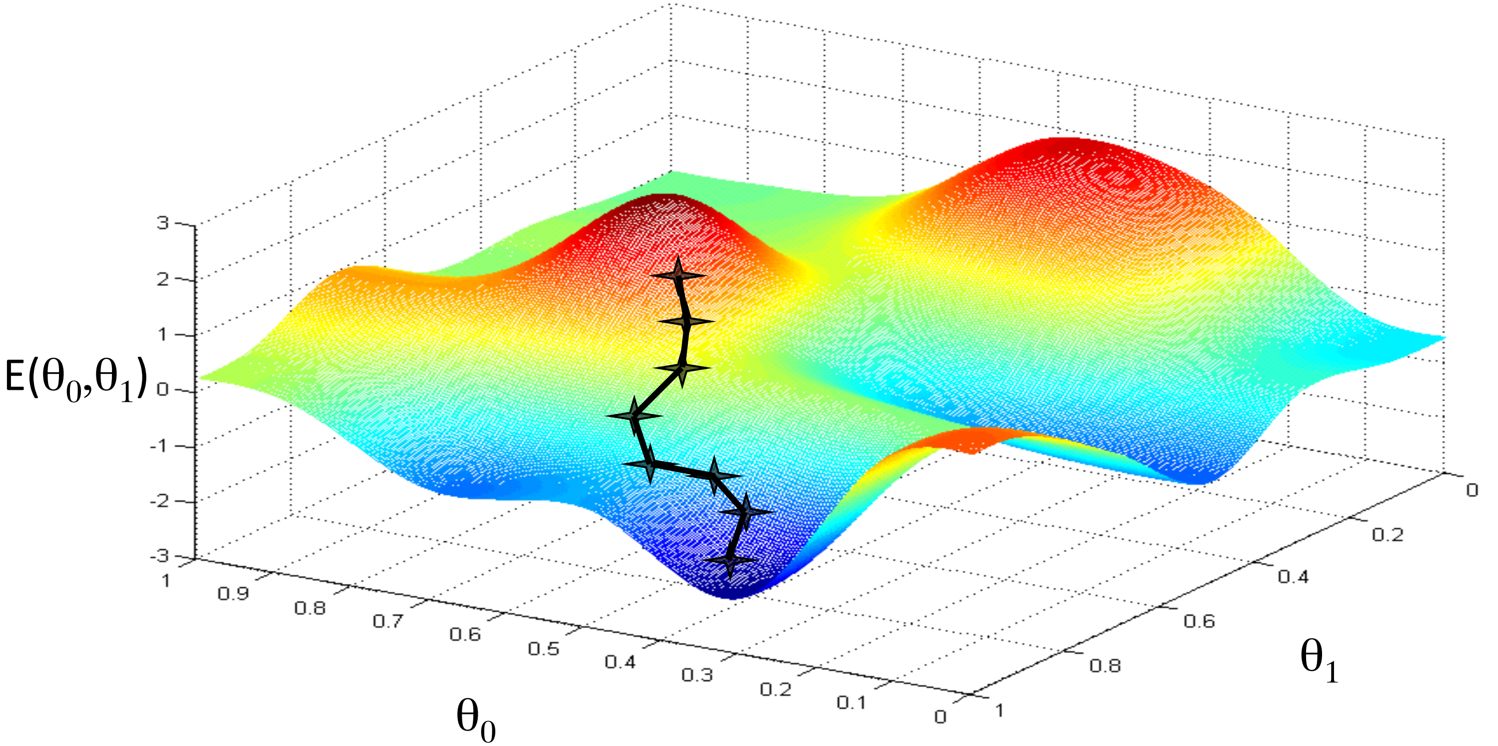
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = y_i\frac{\partial E}{\partial z_j}$$

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

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?

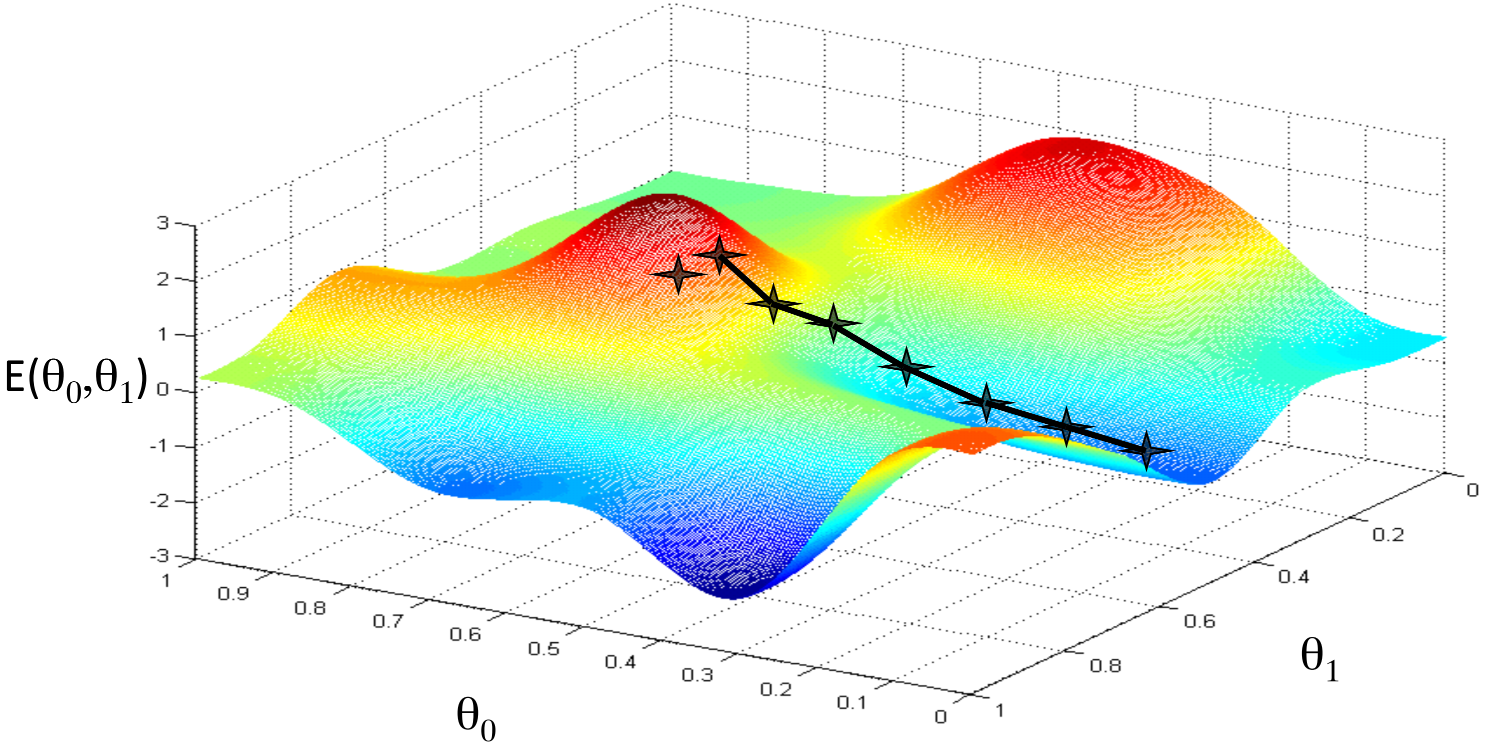
Gradient descent algorithm



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

Andrew Ng

Gradient descent algorithm

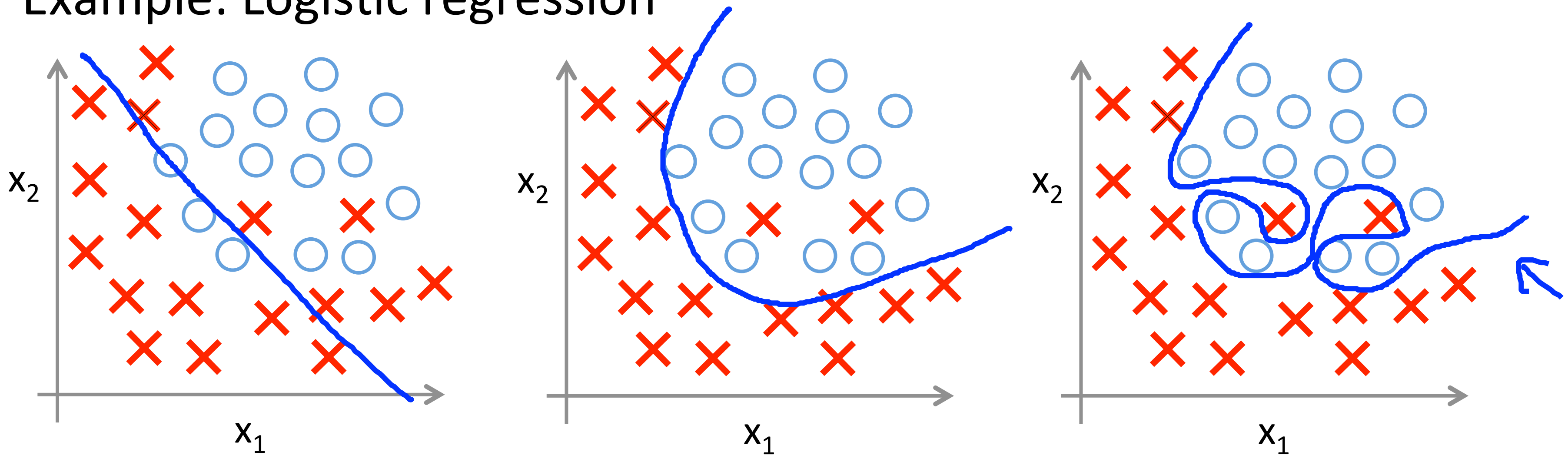


$$\text{repeat until convergence } \{ \; W := W - \alpha \frac{\partial E}{\partial W} \; \}$$

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



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

(g = sigmoid function)

"Underfit"

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

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

"Overfit"

Preventing overfitting

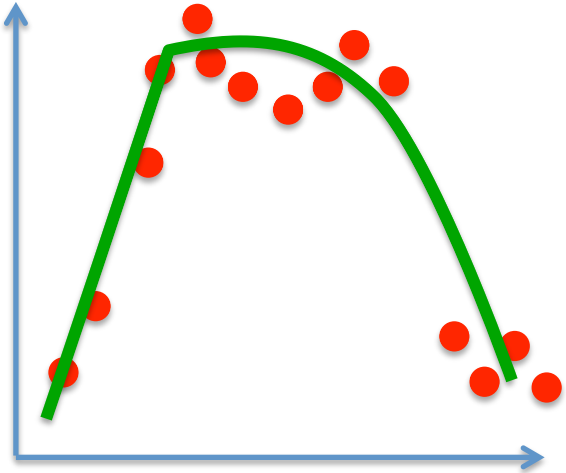
- Approach 1: Get more data!
  - almost always the best bet if you have enough compute power to train on more data
- Approach 2: Use a model that has the right capacity:
  - enough to fit the true regularities.
  - not enough to also fit spurious regularities (if they are weaker)
- Approach 3: Average 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

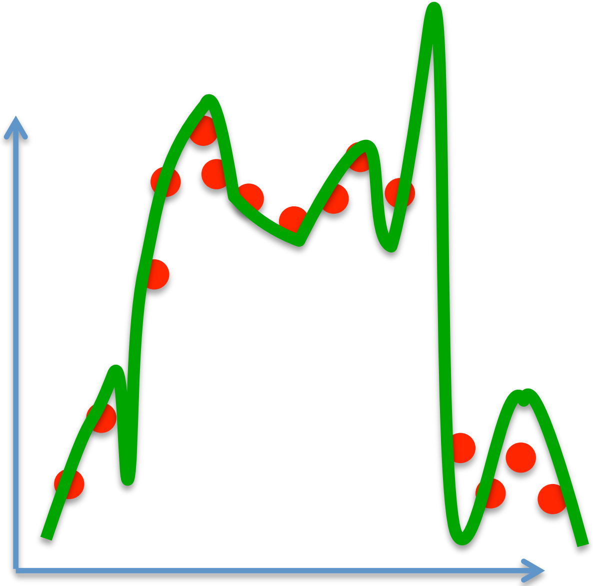
- The capacity can be controlled in many ways:
  - Architecture: Limit the number of hidden layers and the number of units per layer.
  - Early stopping: Start with small weights and stop the learning before it overfits.
  - Weight-decay: Penalize large weights using penalties or constraints on their 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

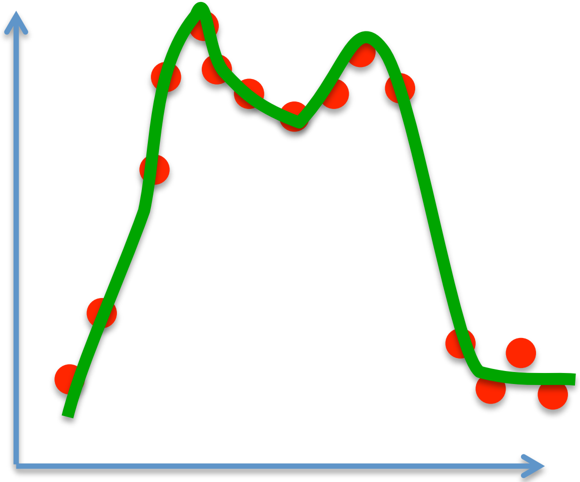
Small model



Big model

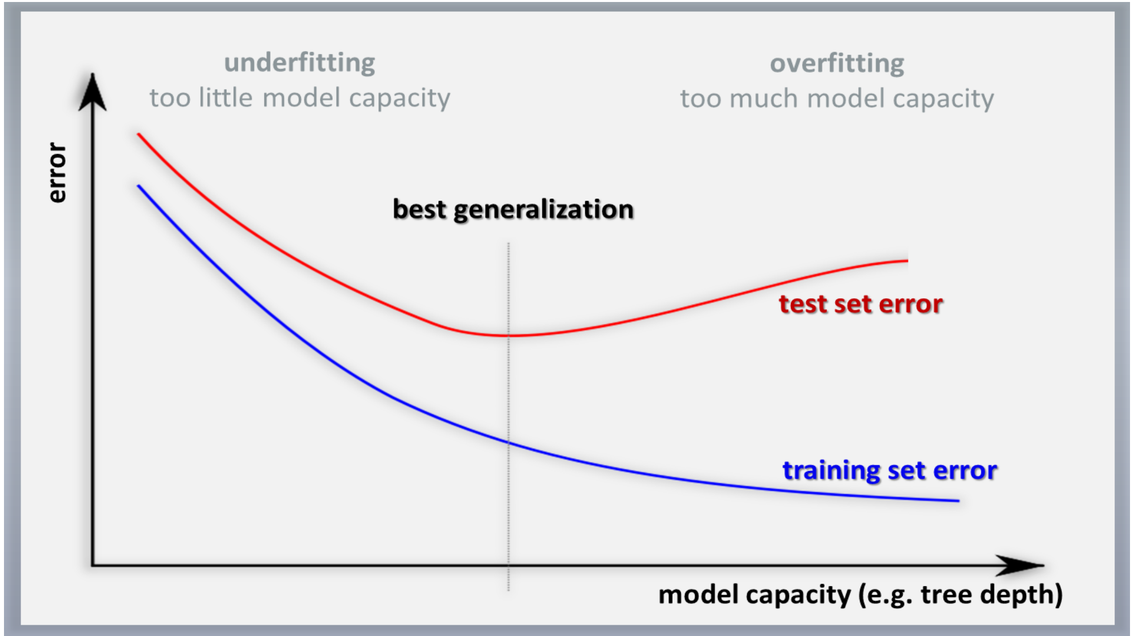


Big model + regularize



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.

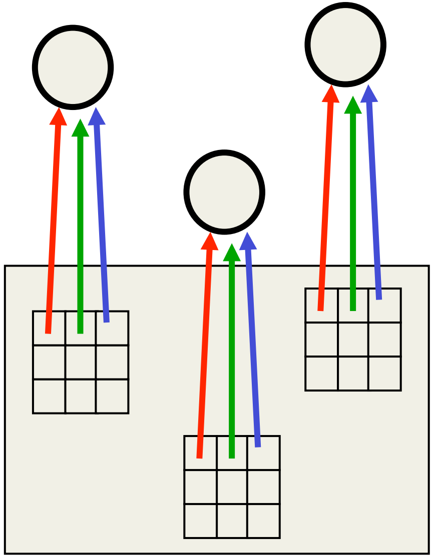


Convolutional Neural Networks
(currently the dominant approach for neural networks)

- 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.

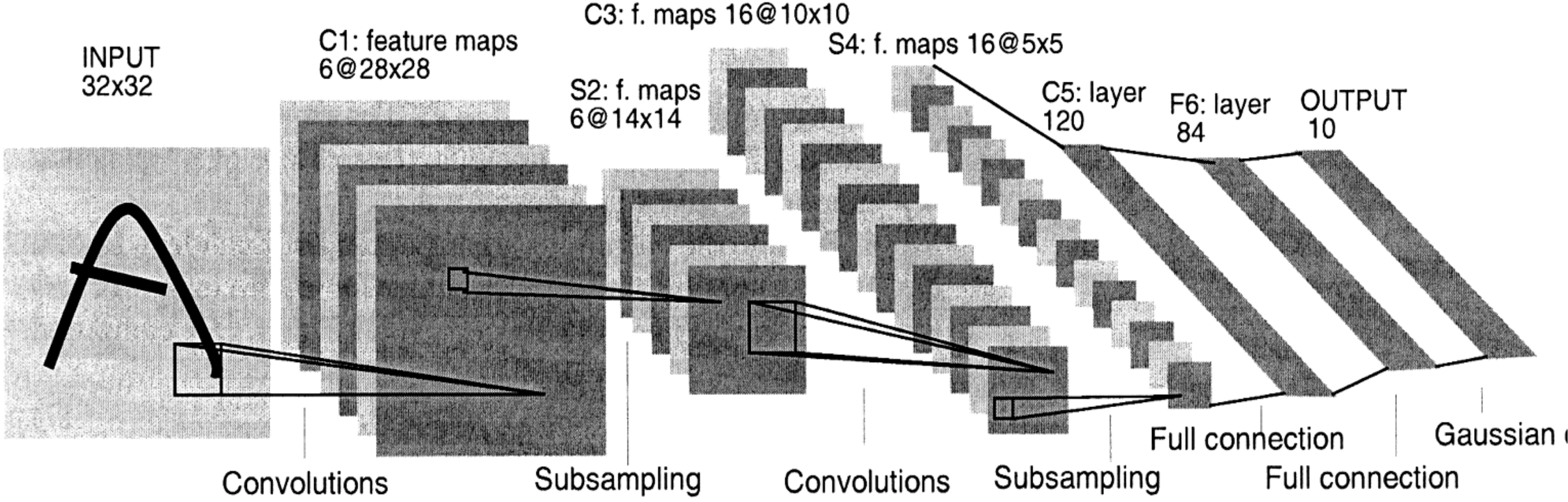


Le Net

- 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

AT&T LeNet 5 RESEARCH
answer: 0
AT&T LeNet 5 RESEARCH
answer: 12
AT&T LeNet 5 RESEARCH
answer: 31
AT&T LeNet 5 RESEARCH
answer: 2
AT&T LeNet 5 RESEARCH
answer: 5
AT&T LeNet 5 RESEARCH
answer: 30
AT&T LeNet 5 RESEARCH
AT&T LeNet 5 RESEARCH
AT&T LeNet 5 RESEARCH

The architecture of LeNet5

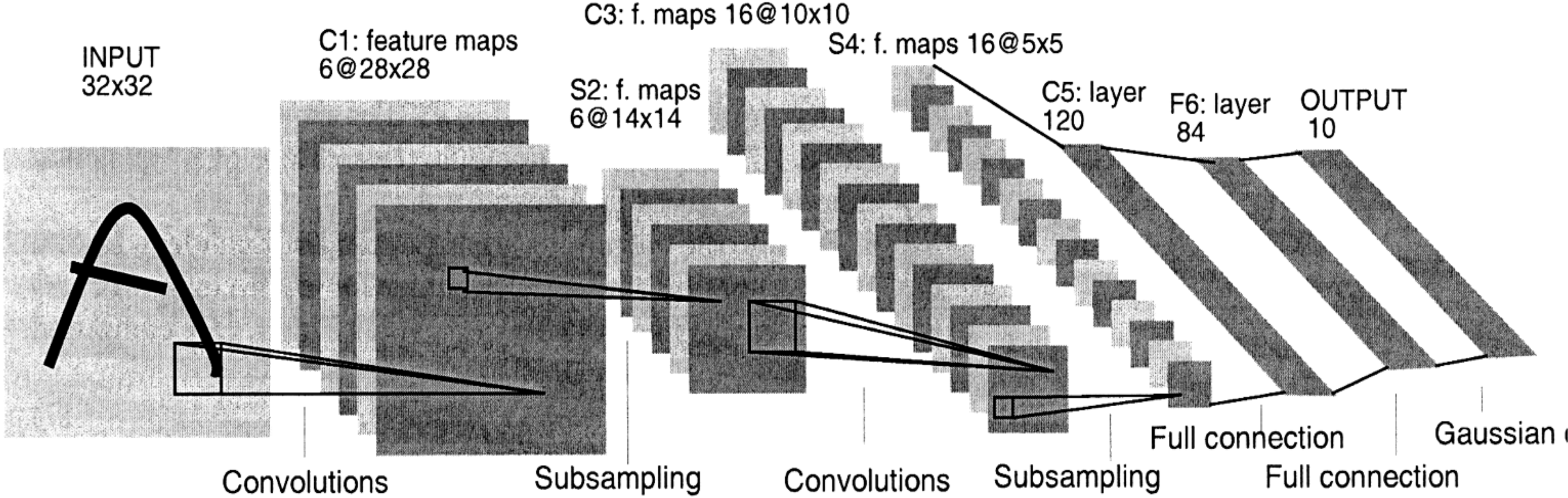


Pooling the outputs of replicated feature detectors

- 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.

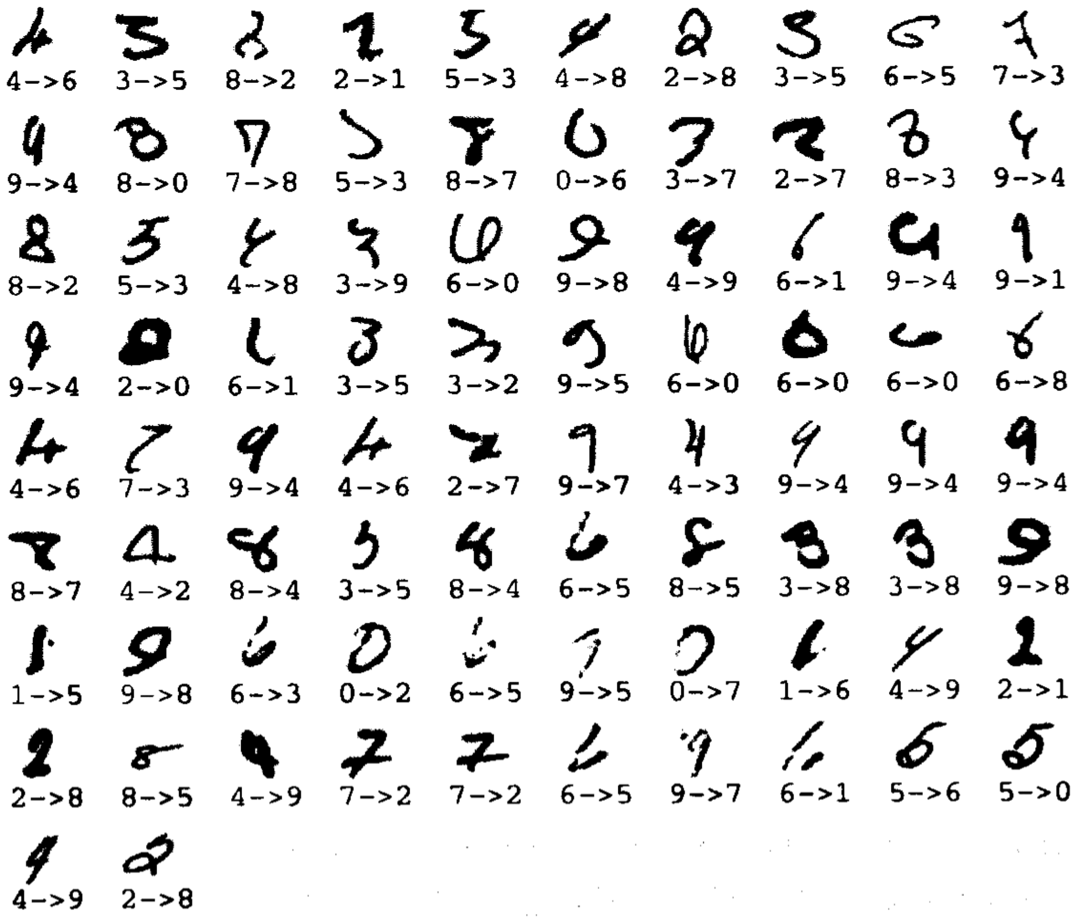
- Problem: After several levels of pooling, we have lost information about the precise positions of things.

The architecture of LeNet5




AT&T
LeNet 5
RESEARCH
answer: 0
AT&T
LeNet 5
RESEARCH
answer: 12
AT&T
LeNet 5
RESEARCH
answer: 31
AT&T
LeNet 5
RESEARCH
answer: 2
AT&T
LeNet 5
RESEARCH
answer: 5
AT&T
LeNet 5
RESEARCH
answer: 30
AT&T
LeNet 5
RESEARCH
AT&T
LeNet 5
RESEARCH
AT&T
LeNet 5
RESEARCH

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

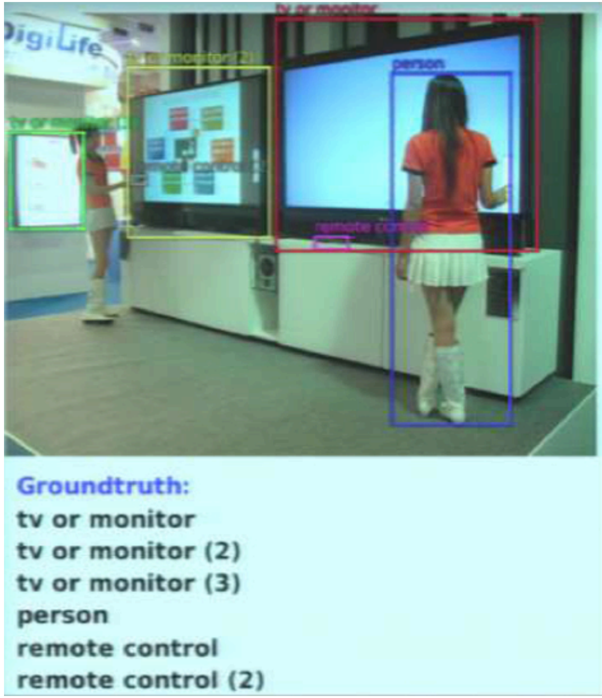
- LeNet uses knowledge about the invariances to design:
  - the local connectivity
  - the weight-sharing
  - the pooling.
- This achieves about 80 errors.
- Ciresan *et al.* (2010) inject knowledge of invariances by creating a huge amount of carefully designed extra training data:
  - For each training 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

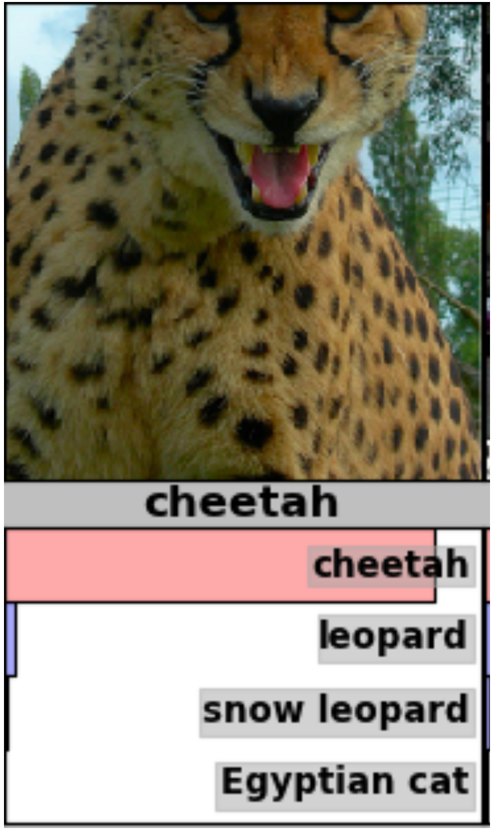
- Recognizing real objects in color photographs downloaded from the web is much more complicated than recognizing hand-written digits:
  - Hundred times as many classes (1000 vs. 10)
  - Hundred times as many pixels (256 x 256 color vs. 28 x 28 gray)
  - Two dimensional image of three-dimensional scene.
  - 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.

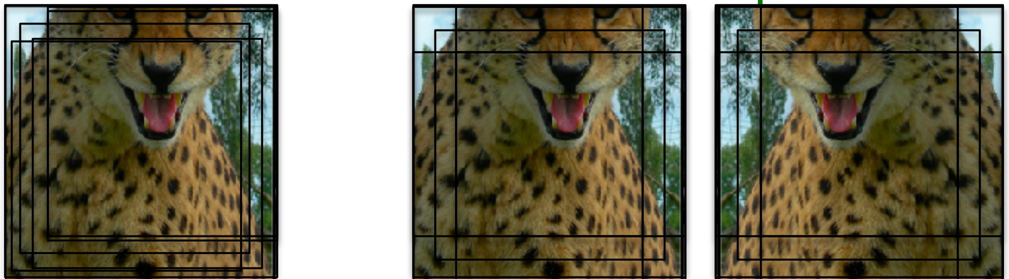


Examples from the test set (with the network's guesses)



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.

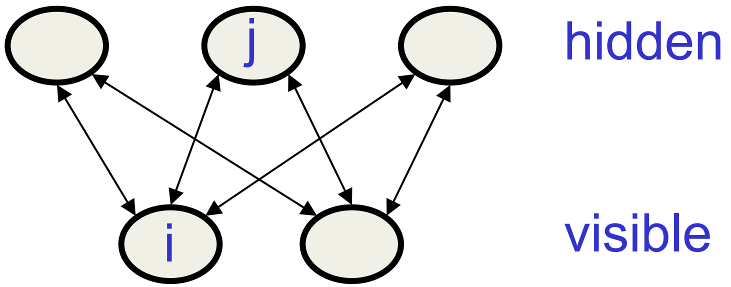


- Use "dropout" to regularize the weights in the globally connected layers (which contain most of the parameters).
    - Dropout means that half of the hidden units in a layer are randomly removed for each training example.
    - This stops hidden units from relying too much on other hidden units.

Auto-Encoders

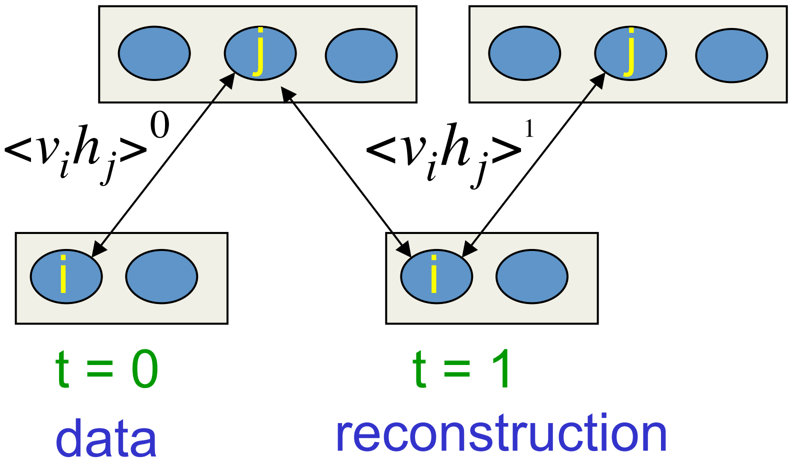
Restricted Boltzmann Machines

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



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

Contrastive divergence to train an RBM



$$\Delta w_{ij} = \varepsilon \, (<v_i h_j>^0 - <v_i h_j>^1)$$

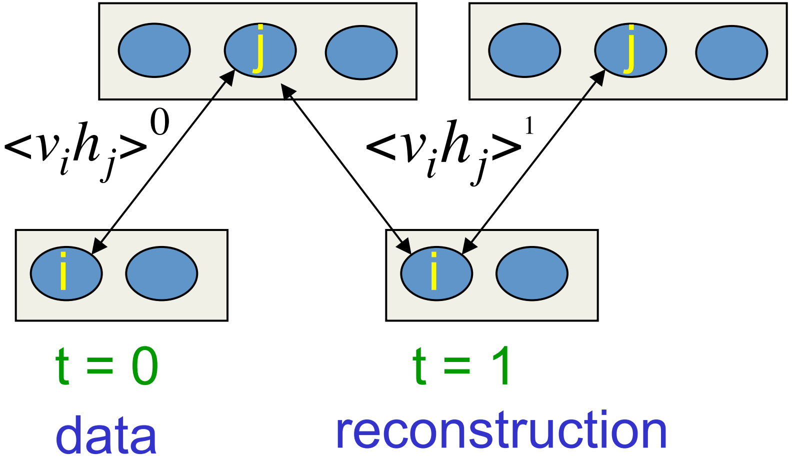
Start with a training vector on the visible units.

Update all the hidden units in parallel.

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

Update the hidden units again.

Explanation



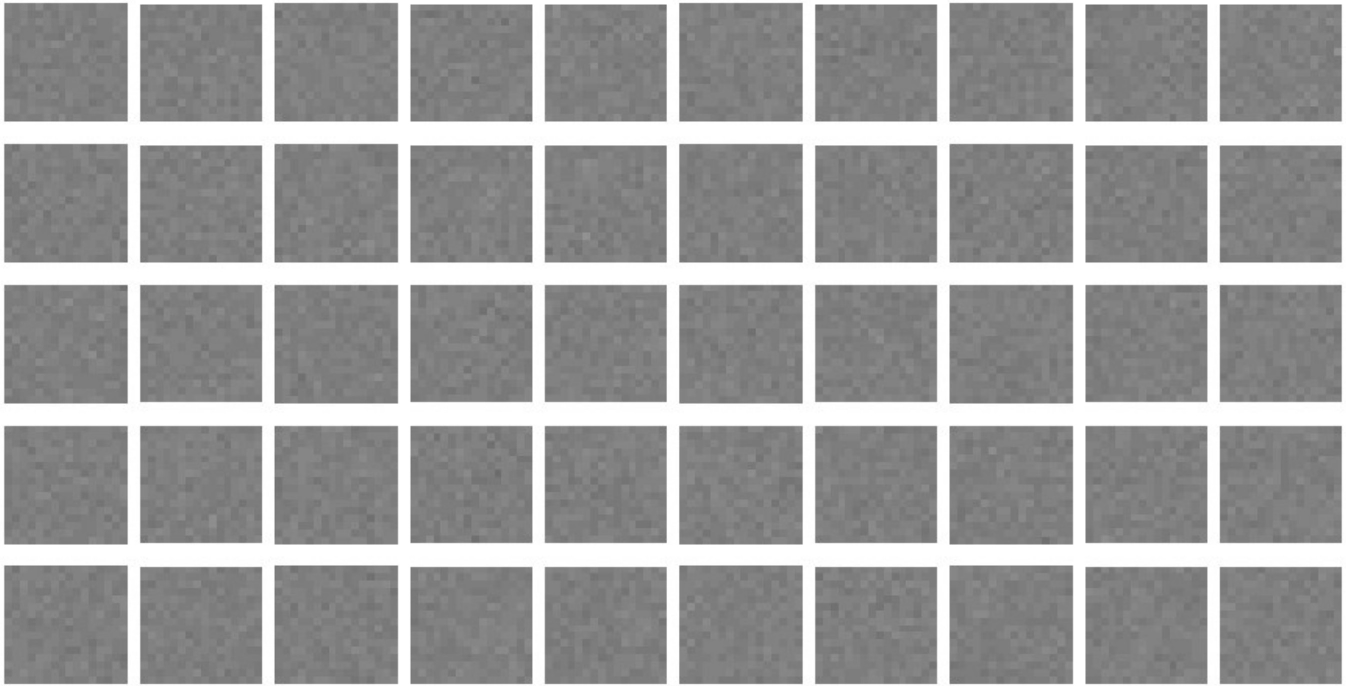
$$\Delta w_{ij} = \varepsilon \left(<v_i h_j>^0 - <v_i h_j>^1 \right)$$

Ideally, hidden layers re-generate the input.

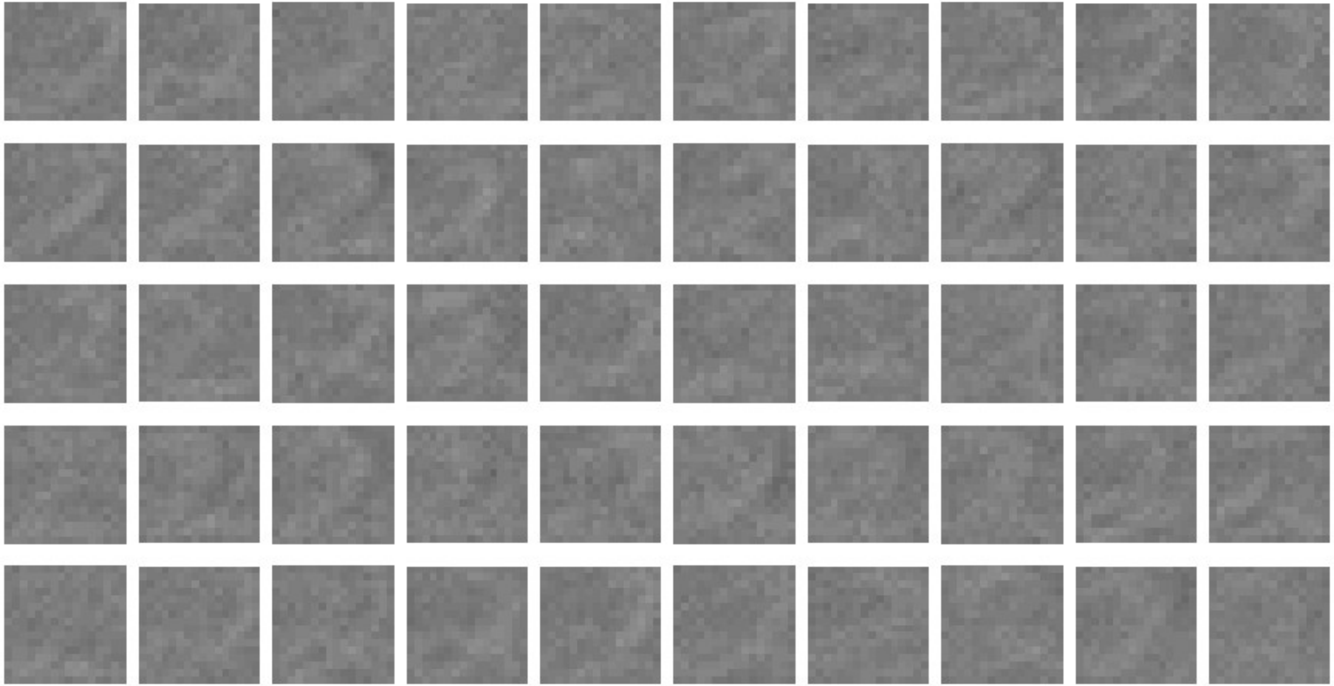
If that's not the case, the hidden layers generate something else.

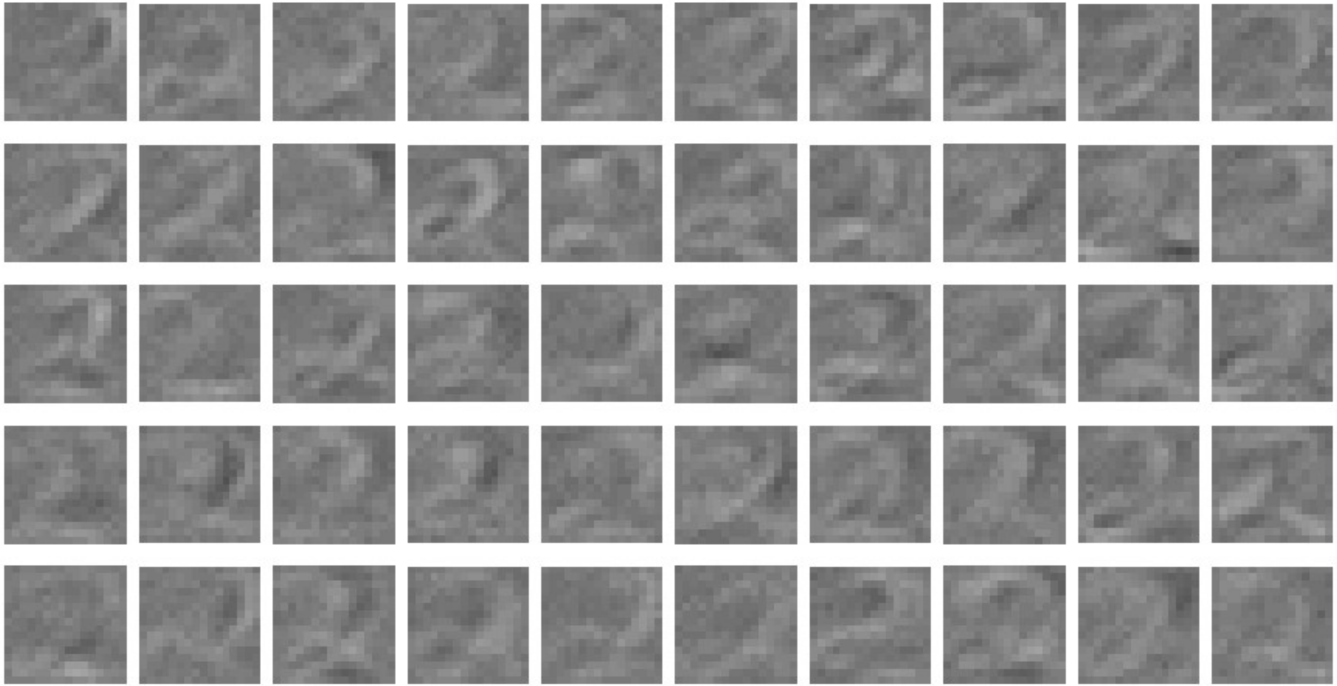
Change the weights so that this wouldn't happen.

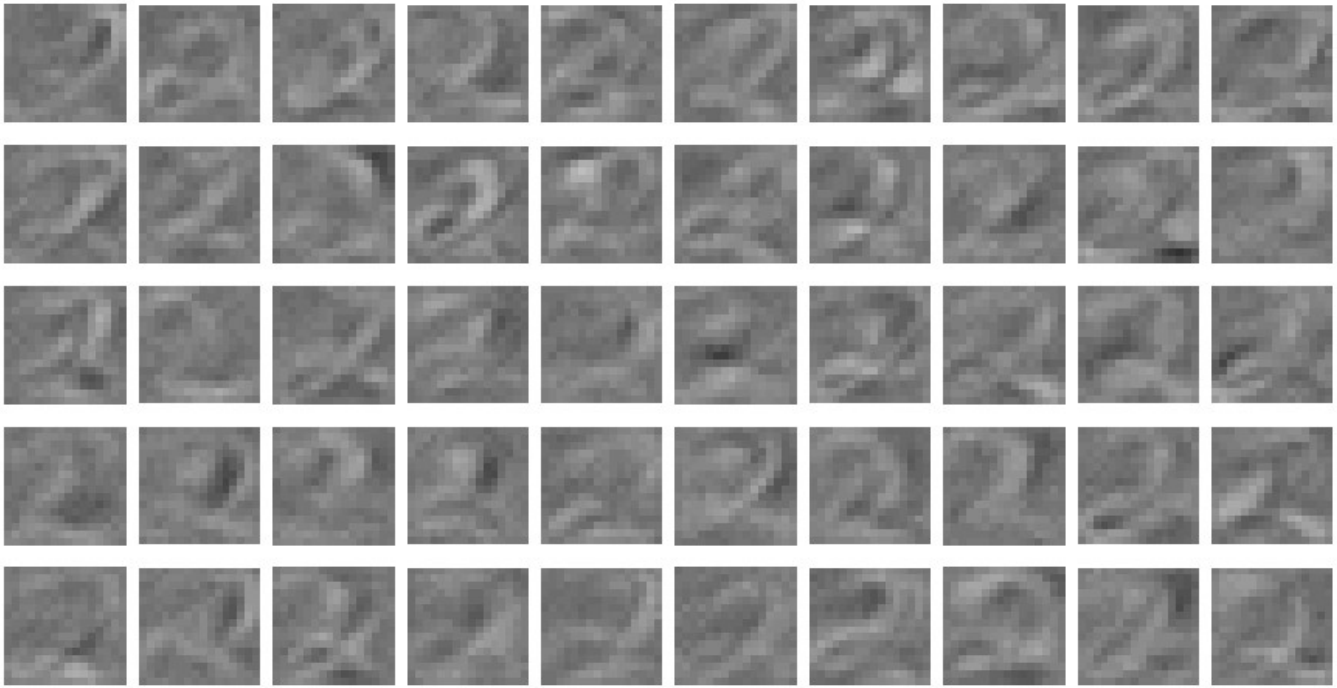
The weights of the 50 feature detectors

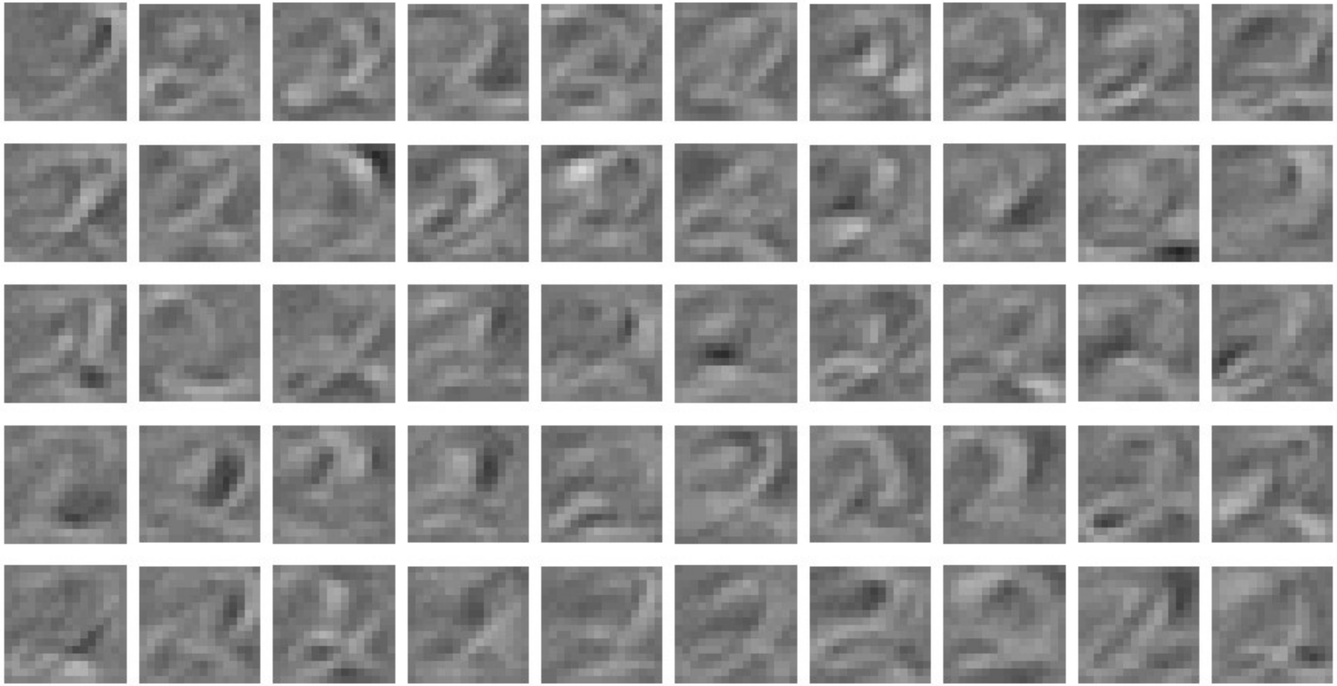


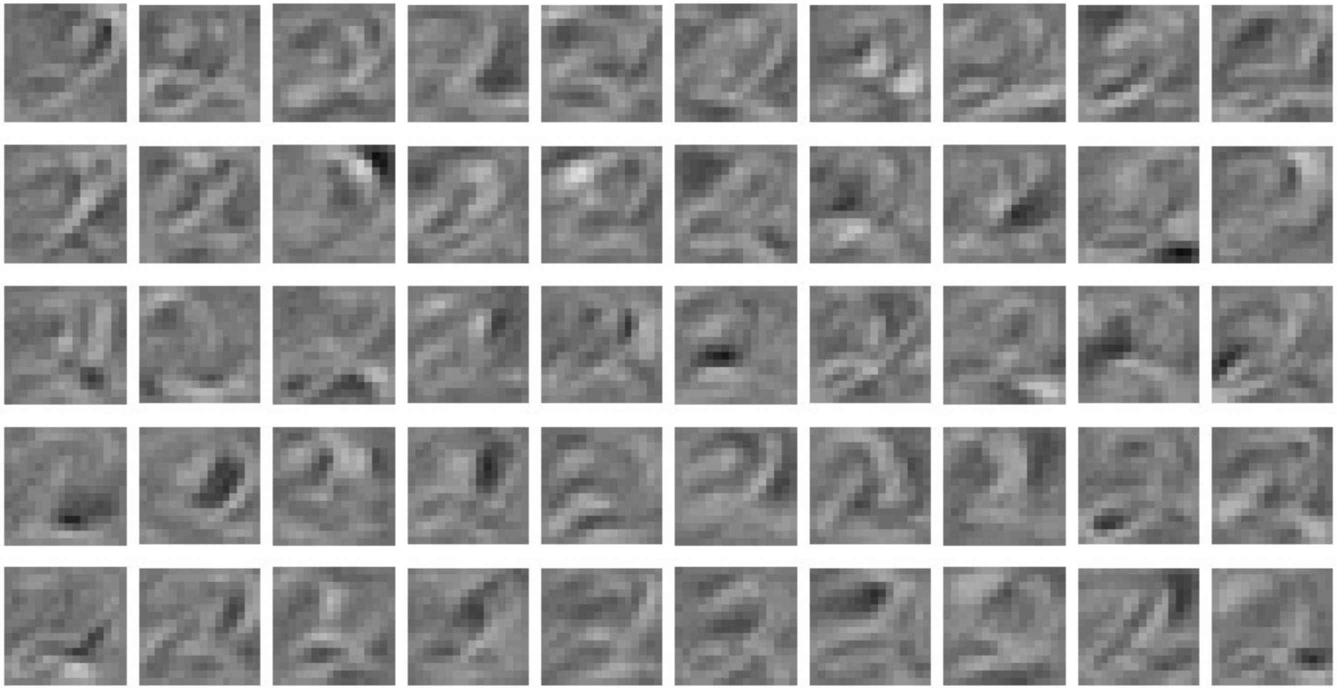
We start with small random weights to break symmetry

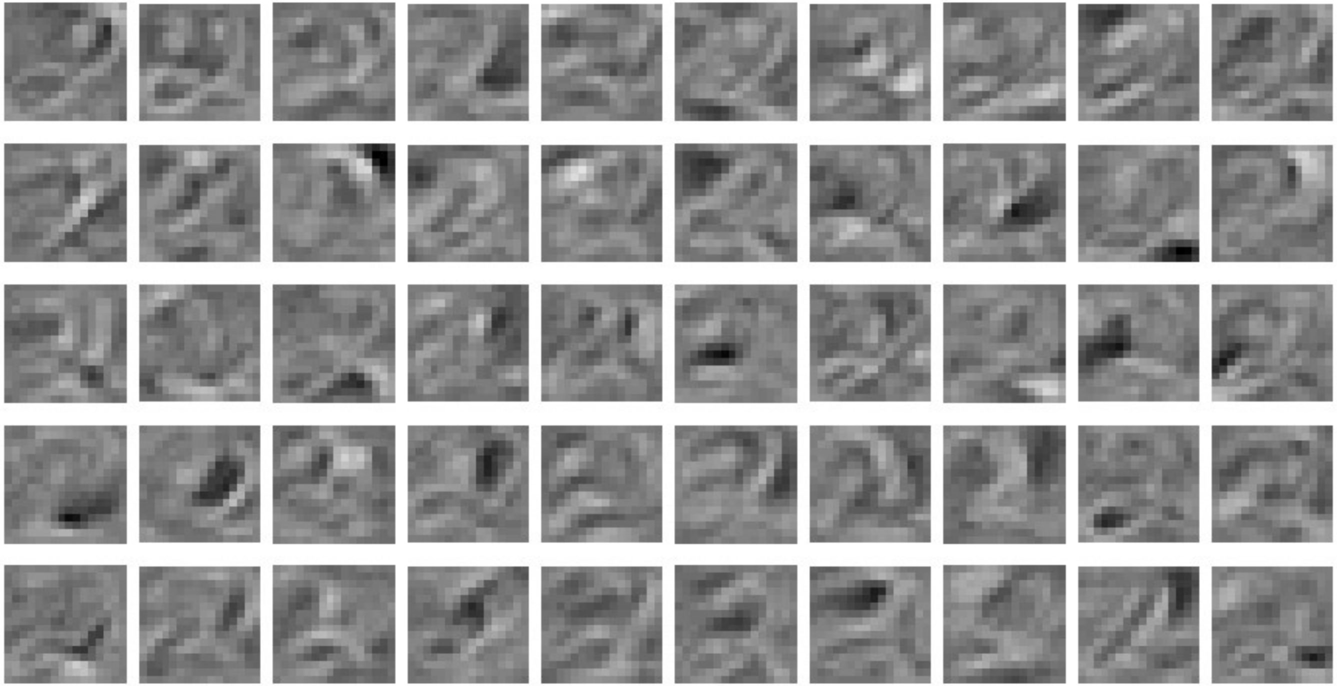


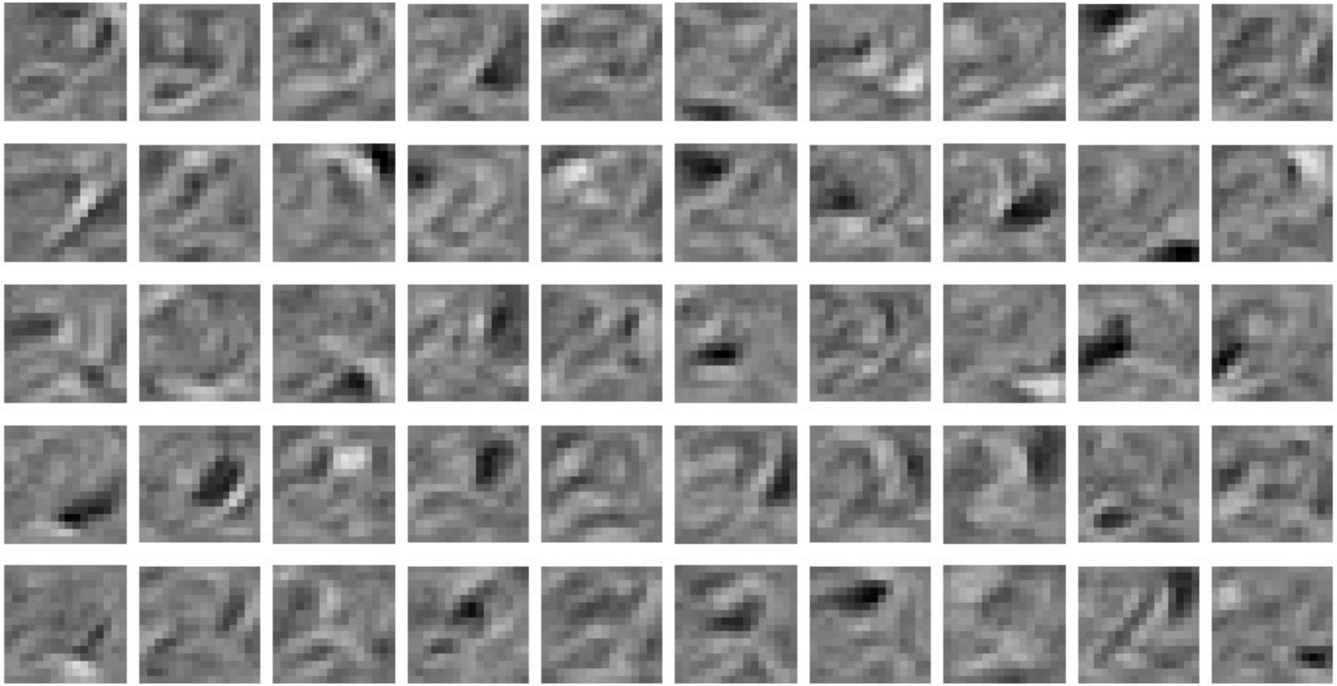


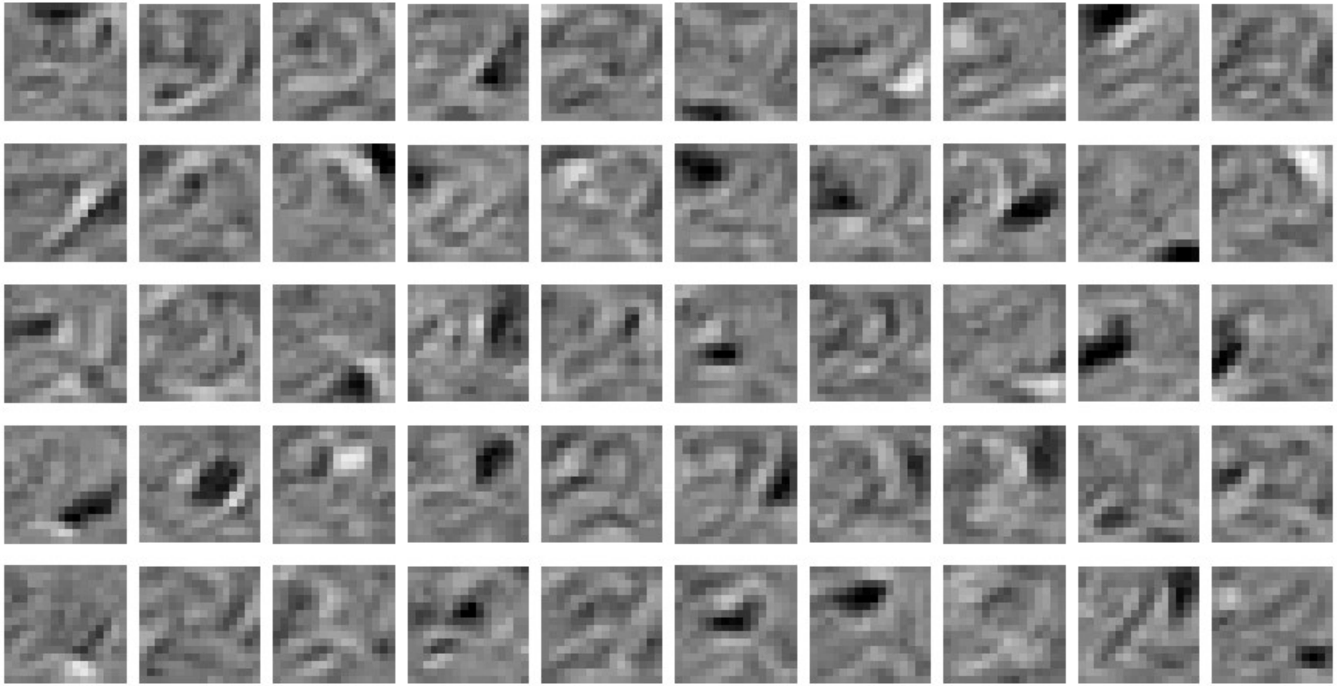




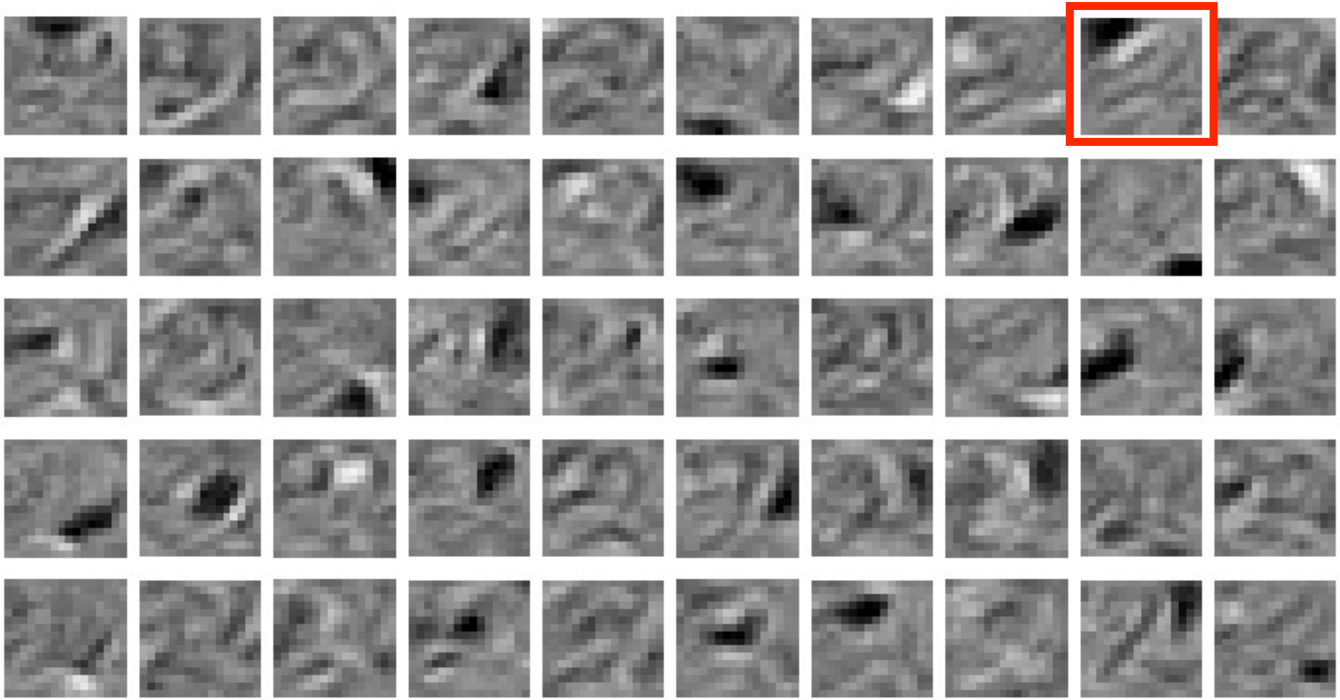








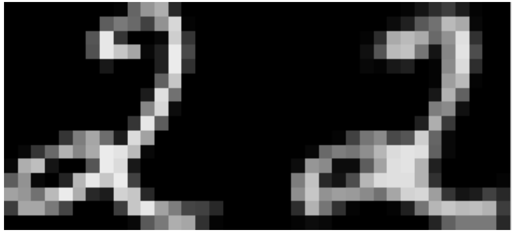
The final 50 x 256 weights: Each neuron grabs a different feature



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

Data

Reconstruction from activated binary features



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

Data

Reconstruction from activated binary features

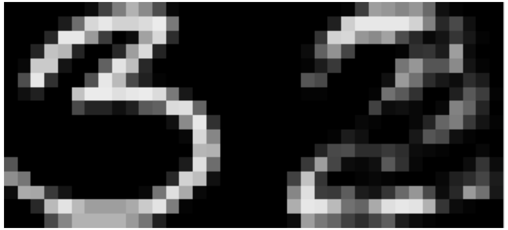
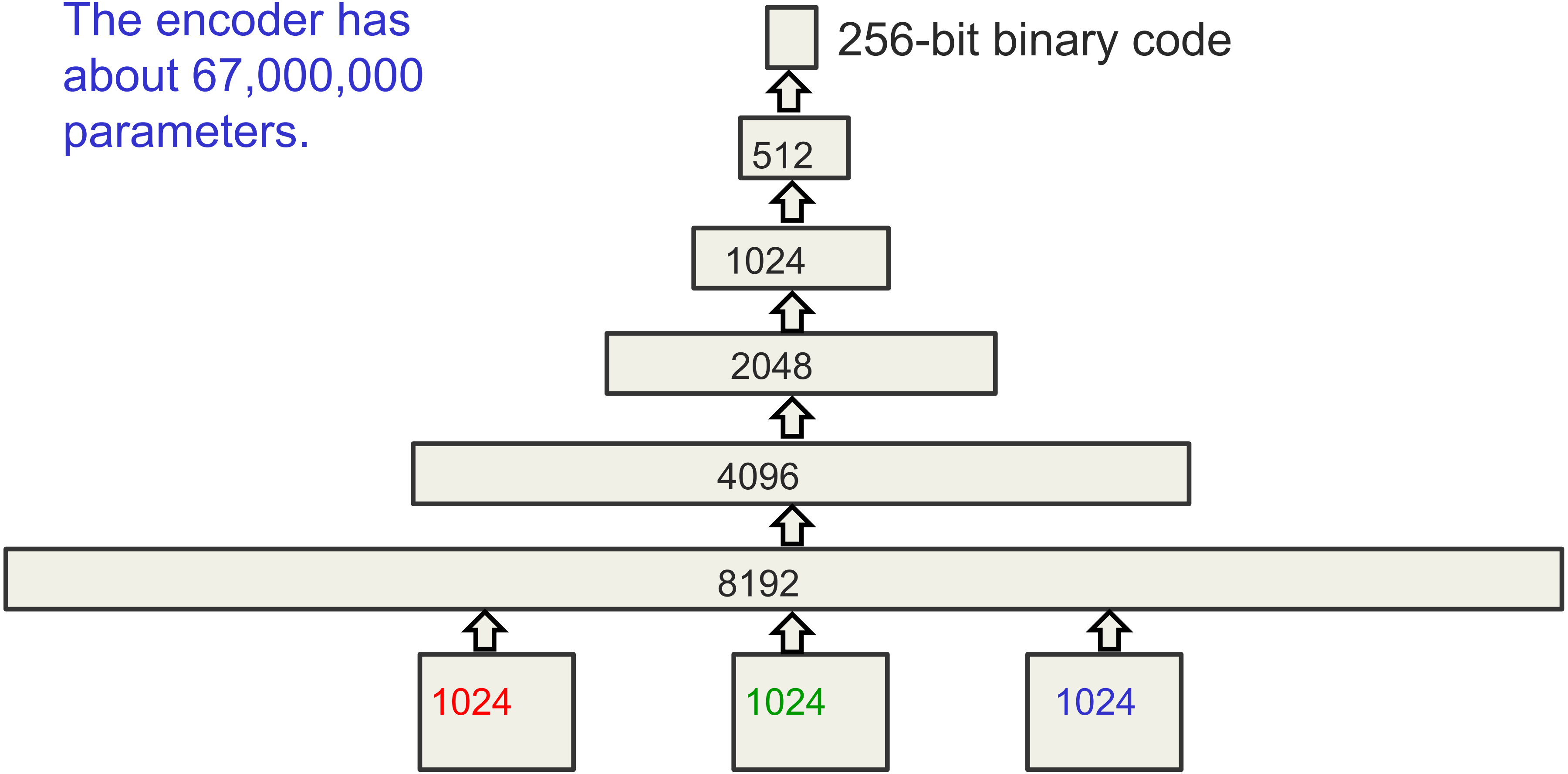


Image from an unfamiliar digit class

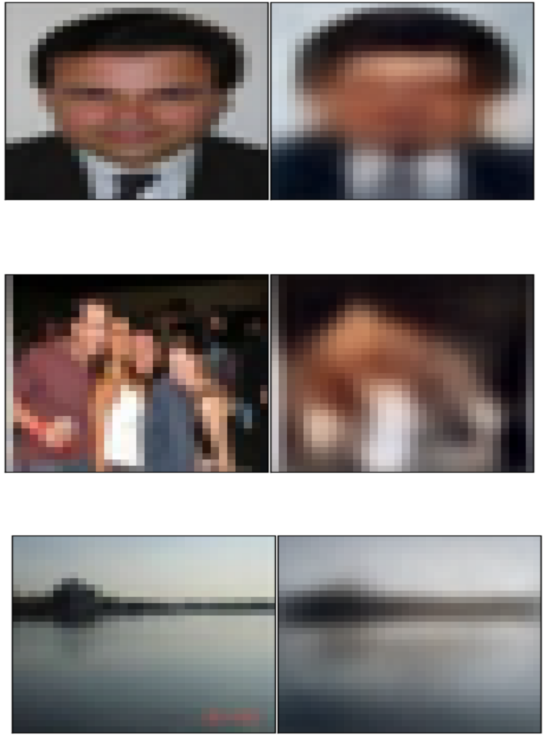
The network tries to see every image as a 2.

Krizhevsky’s deep autoencoder

The encoder has about 67,000,000 parameters.



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