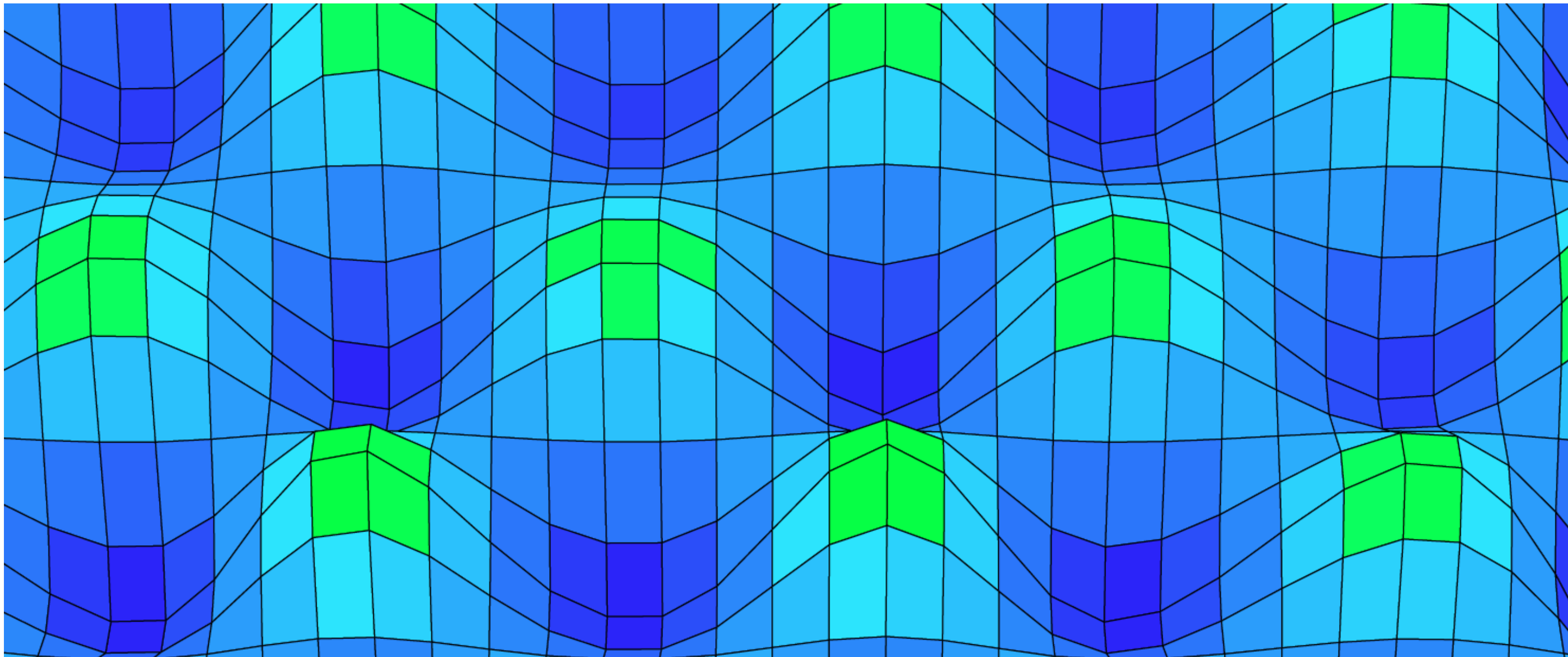
EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Philosophische Fakultät
Seminar für Sprachwissenschaft



Advanced regression models: classification

Peter Hendrix

Classification

“the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations whose category membership is known”

http://en.wikipedia.org/wiki/Statistical_classification

Classification

“All models are wrong, but some are useful”

George Box

Outline

- MNIST database
- Multinomial logistic regression
- Decision trees
- k-Nearest Neighbors
- Support vector machines
- Neural networks

Outline

- MNIST database
- Multinomial logistic regression
- Decision trees
- k-Nearest neighbors
- Support vector machines
- Neural networks

MNIST database

- MNIST database (Mixed National Institute of Standards and Technology database)
- Subset used as training data in Digit Recognizer competition on Kaggle (`http://www.kaggle.com`)
- Train on $32,000$ digits
- Classify $10,000$ unseen digits

MNIST database

```r
# Read training data
load("data/train.rda")
dim(train)
# [1] 32000   786
#
# Read test data
load("data/test.rda")
dim(test)
# [1] 10000   786
#
# See column names
colnames(train)[1:10]
#  [1] "label"  "pixel0" "pixel1" "pixel2" "pixel3" "pixel4" "pixel5"
#  [8] "pixel6" "pixel7" "pixel8"
```

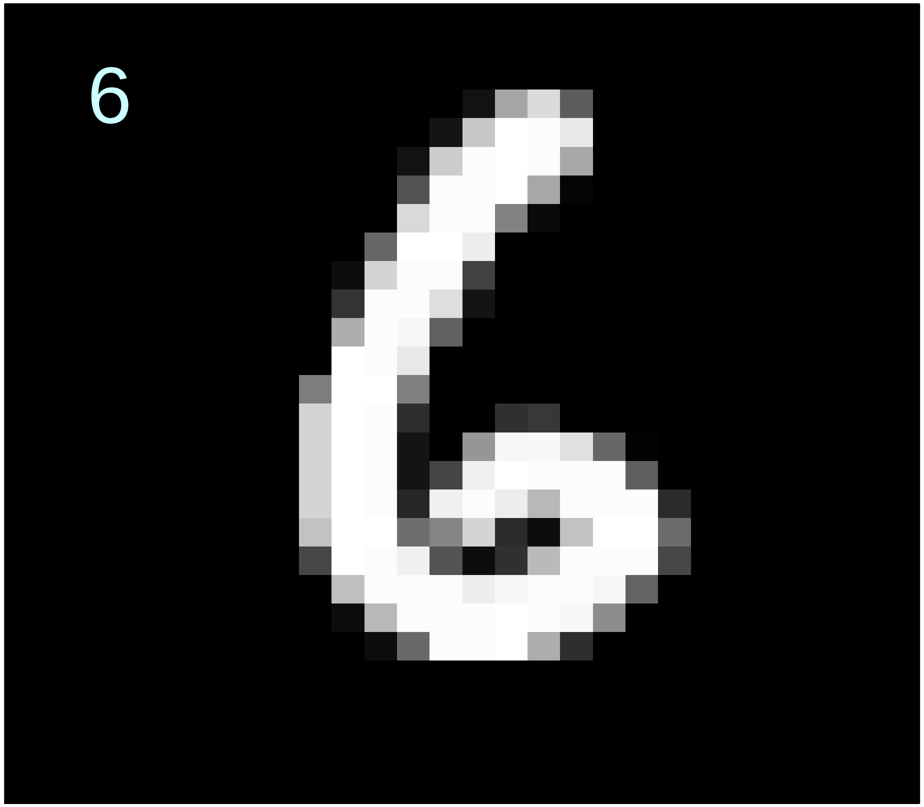
MNIST database

$$\begin{pmatrix} \texttt{pixel0} & \texttt{pixel1} & \texttt{pixel2} & \texttt{pixel3} & \dots & \texttt{pixel26} & \texttt{pixel27} \\ \texttt{pixel28} & \texttt{pixel29} & \texttt{pixel30} & \texttt{pixel31} & \dots & \texttt{pixel54} & \texttt{pixel55} \\ \texttt{pixel56} & \texttt{pixel57} & \texttt{pixel58} & \texttt{pixel59} & \dots & \texttt{pixel72} & \texttt{pixel73} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \texttt{pixel728} & \texttt{pixel729} & \texttt{pixel730} & \texttt{pixel731} & \dots & \texttt{pixel754} & \texttt{pixel755} \\ \texttt{pixel756} & \texttt{pixel757} & \texttt{pixel758} & \texttt{pixel759} & \dots & \texttt{pixel782} & \texttt{pixel783} \end{pmatrix}$$

MNIST database

```
# Pick a random image
set.seed(97)
num = sample(1:nrow(test),1)
#
# Turn into matrix
pic = as.numeric(test[num,2:785])
pic = matrix(pic,ncol=28,byrow=TRUE)
#
# Plot matrix
pic = t(apply(pic,2,rev))
image(pic,col=grey(level=seq(0,1,by=0.01)),xaxt="n",yaxt="n",
      useRaster=TRUE)
text(0.1,0.9,test$label[num],col="#CCFFFF",cex=2.5)
```

MNIST database



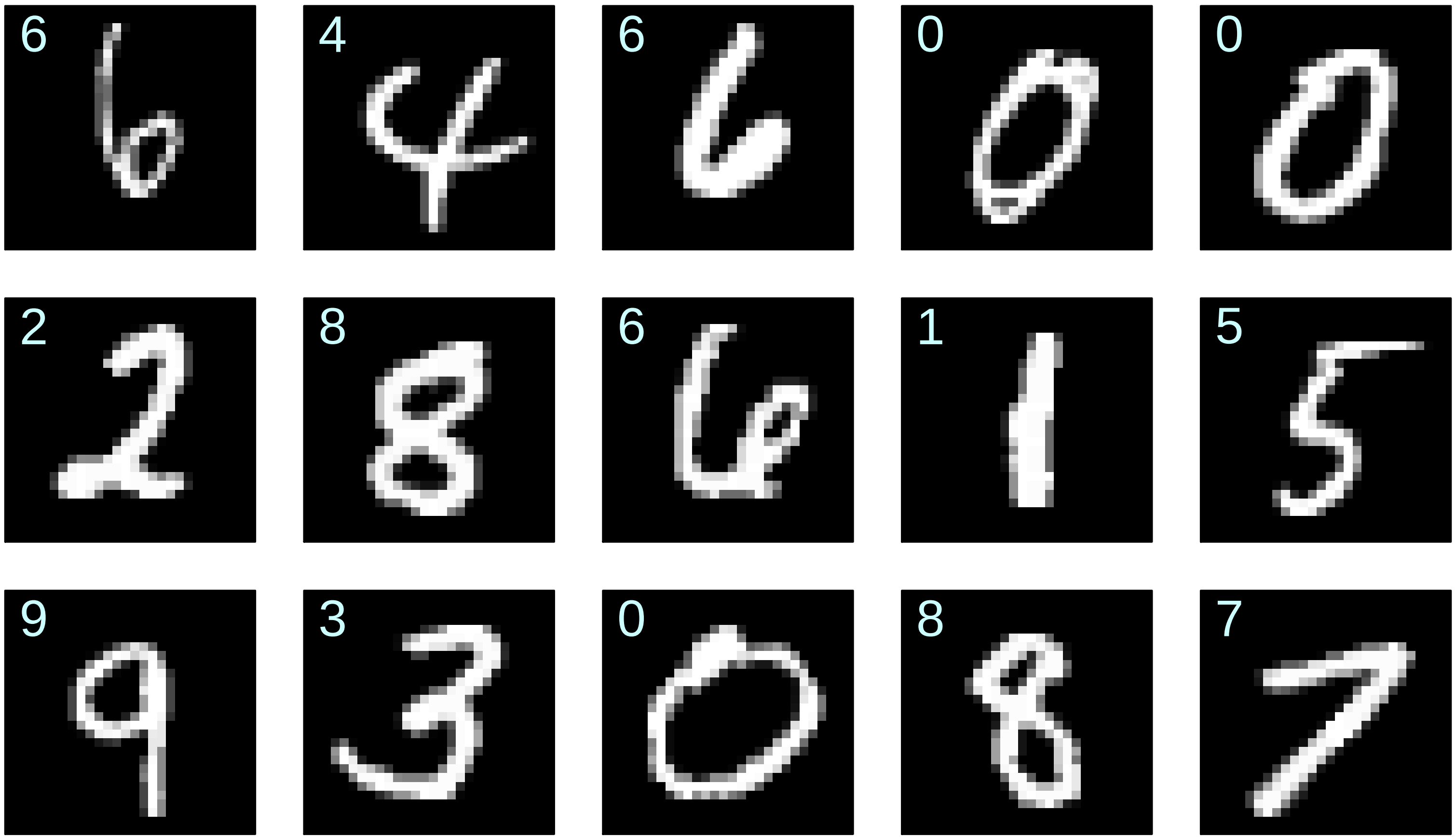
MNIST database

```r
# Create a general function for drawing
draw.fnc = function(num) {

  par(mar=c(1,1,1,1))
  pic = as.numeric(test[num,2:785])
  pic = matrix(pic,ncol=28,byrow=TRUE)
  pic = t(apply(pic,2,rev))
  image(pic,col=grey(level=seq(0,1,by=0.01)),xaxt="n",yaxt="n",
        useRaster=TRUE)
  text(0.1,0.9,test$label[num],col="#CCFFFF",cex=2.5)

}
```

MNIST database

```
# Pick random set of images
set.seed(416)
draw = sample(1:nrow(test),15,replace=F)
#
# Plot using function
par(mfrow=c(3,5))
par(oma=c(1,1,1,1))
sapply(draw,draw.fnc)
```

MNIST database



Outline

- MNIST database
- Multinomial logistic regression
- Decision trees
- k-Nearest neighbors
- Support vector machines
- Neural networks

Multinomial logistic regression

- Linear regression: numerical dependent variable
- Logistic regression: categorical dependent variable
- Convert binary outcome into continuous output
- Logit link function:

$$logit = \log\left(\frac{p}{1-p}\right)$$

Multinomial logistic regression

- Binary logistic regression: use 1 logit function to predict the odds of class 1 versus class 2
- Multinomial regression: use $k - 1$ logit functions to predict the odds of class 1 versus class 2, class 3, . . . , class k
- Assumption: IIA (indepence of irrelevant alternatives)
- The odds of predicting class 1 versus class 2 should not depend on the presence or absence of class 3

Multinomial logistic regression

```r
# Load library
library(nnet)
#
# Normalization function
normalize.fnc = function(x) {
  if(length(unique(x))>1) {
    return((x-min(x))/(max(x)-min(x)))
  } else {
    return(x)
  }
}
```

Multinomial logistic regression

```
# Normalize predictors
train.norm = train; test.norm = test
train.norm[,2:785] = apply(train.norm[,2:785],2,normalize.fnc)
test.norm[,2:785] = apply(test.norm[,2:785],2,normalize.fnc)
#
# Save normalized data
save(train.norm,file="data/train.norm.rda")
save(test.norm,file="data/test.norm.rda")
```

Multinomial logistic regression

```
# Load normalized data
load("data/train.norm.rda")
load("data/test.norm.rda")
# Run model
preds = paste(colnames(train)[2:785],collapse="+")
form = as.formula(paste("label ~",preds))
set.seed(234)
multinom = multinom(form,data=train.norm,MaxNWts=10000,maxit=50)
# # weights:  7860 (7065 variable)
# initial  value 73682.722976
# iter  10 value 19672.873857
# iter  20 value 13381.492378
# iter  30 value 11317.909415
# iter  40 value 9530.870468
# iter  50 value 8163.661850
# final  value 8163.661850
# stopped after 50 iterations
```

Multinomial logistic regression

```
# Predict
class_multinom = predict(multinom,newdata=test.norm)
table(class_multinom==test$label)
#
# FALSE  TRUE
#  1012  8988
```

Multinomial logistic regression

```
# Load bigger model
# Settings: maxit = 100
load("models/multinom.rda")
#
# Predict
class_multinom = predict(multinom,newdata=test.norm)
table(class_multinom==test$label)
#
# FALSE  TRUE
#   941  9059
```

Multinomial logistic regression

```
# Load even bigger model
# Settings: maxit = 1000
load("models/multinom1000.rda")
#
# Predict
class_multinom = predict(multinom,newdata=test.norm)
table(class_multinom==test$label)
#
# FALSE  TRUE
#  1059  8941
```

Multinomial logistic regression

- How to prevent overfitting?
  - cross-validation
  - penalization of regression coefficients

Multinomial logistic regression

```
# Load library
library(glmnet)
#
# Register cluster for parallel computation
library(doMC)
registerDoMC(4)
#
# Run model
glm = cv.glmnet(as.matrix(train[,2:785]),train$label,
                family="multinomial",nfolds=3,parallel=TRUE,
                maxit=1000)
```

Multinomial logistic regression

```
# Load bigger model (maxit=100,000, nfolds = 10)
load("models/glm.rda")
#
# Performance
class_glm = predict(glm,as.matrix(test[,2:785]),type="class")
table(class_glm==test$label)
#
# FALSE  TRUE
#   916  9084
```

Multinomial logistic regression

```
# Show confusion matrix
table(class_glm,test$label)
#
# class_glm    0    1    2    3    4    5    6    7    8    9
#         0  942    0    6    4    5   13    9    3    4   11
#         1    0 1087   11    7    7    7    4   11   23    8
#         2    5    3  886   21    6    8    6   15    8    3
#         3    3    2   16  915    3   31    0    5   27   13
#         4    2    1   15    1  883   12    8   14    5   21
#         5   14    4    2   30    4  765   15    3   29   10
#         6    9    1   12   11    8   23  931    1    9    1
#         7    0    4   19    8    2    8    2  963    4   41
#         8    9   11   23   27    6   25    9    1  835   12
#         9    0    2    5   12   45   12    1   32   23  877
```

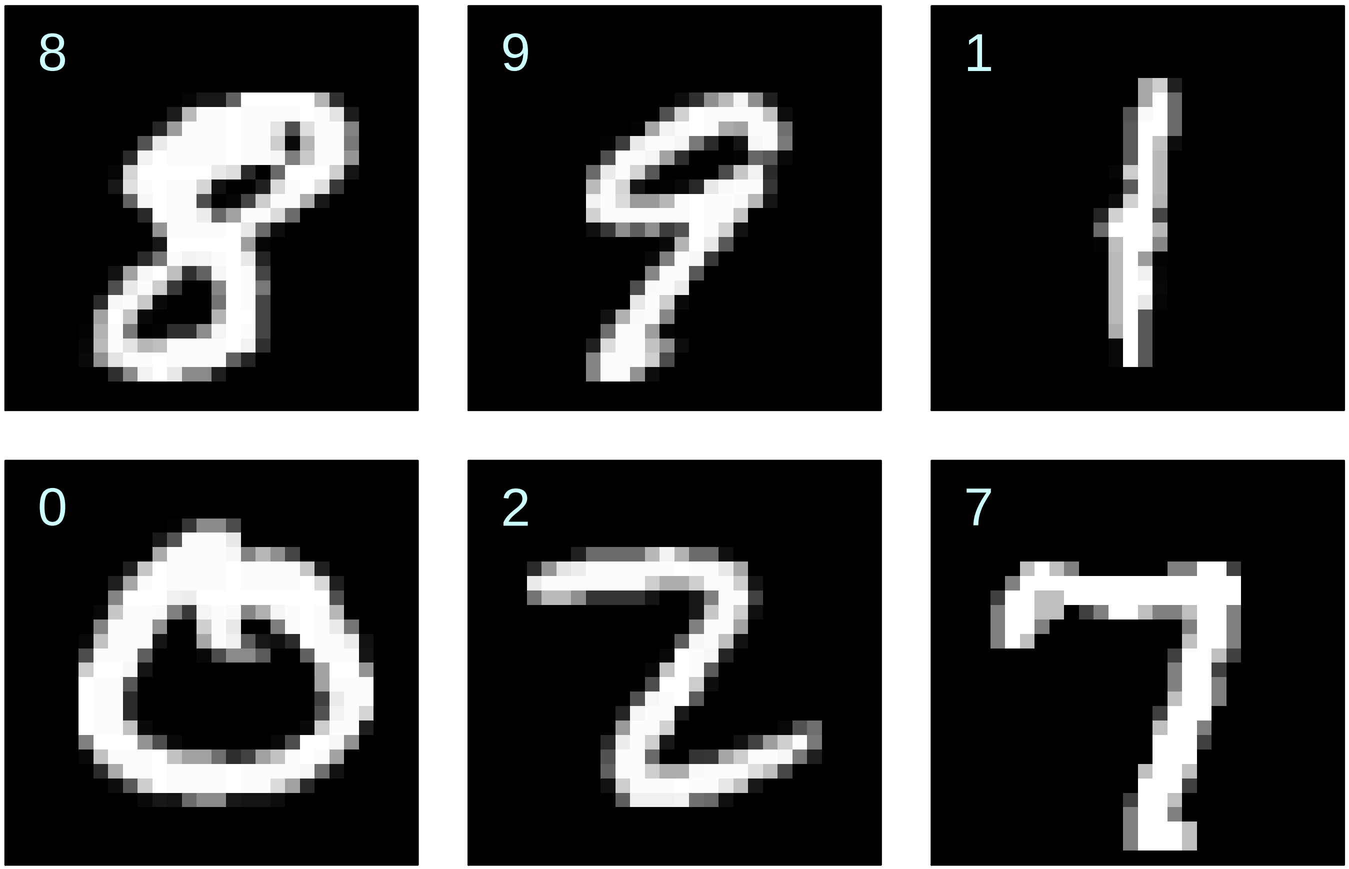
Multinomial logistic regression

```
# Get first 6 test set labels
test$label[1:6]
# [1] 8 9 1 0 2 7
# Levels: 0 1 2 3 4 5 6 7 8 9
#
# Predict probabilities
probs = predict(glm,as.matrix(test[,2:785]),type="response")
round(probs[1:6,1:10,],3)
#       0     1     2     3     4     5     6     7     8     9
# 1 0.000 0.000 0.000 0.002 0.000 0.001 0.000 0.004 0.946 0.047
# 2 0.000 0.000 0.001 0.002 0.025 0.009 0.001 0.018 0.009 0.934
# 3 0.000 0.948 0.007 0.009 0.000 0.006 0.005 0.001 0.016 0.007
# 4 0.991 0.000 0.001 0.003 0.000 0.000 0.000 0.002 0.002 0.000
# 5 0.001 0.074 0.570 0.130 0.000 0.004 0.002 0.040 0.179 0.001
# 6 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.999 0.000 0.001
```

Multinomial logistic regression

```
# Set parameters
par(mfrow=c(2,3))
par(oma=c(1,1,1,1))
draw = 1:6
#
# Draw
invisible(sapply(draw,draw.fnc))
```

Multinomial logistic regression



Multinomial logistic regression

- Advantages:
  - Decent basic classification technique
  - Computationally efficient
- Disadvantages:
  - Often outperformed quite a bit by alternative methods

Outline

- MNIST database
- Multinomial logistic regression
- Decision trees
- k-Nearest neighbors
- Support vector machines
- Neural networks

Decision trees

- Decision trees use a set of binary rules to predict a dependent variable
- The dependent variable can be categorical (classification trees) or numerical (regression trees)
- Different decision tree models use different algorithms to determine the "optimal" set of binary rules

How do decision trees work?

- Start with all observations
- Find the predictor and predictor value that best split the data into two groups
- Repeat until a stopping criterion is met
- Assign a class or value to terminal nodes

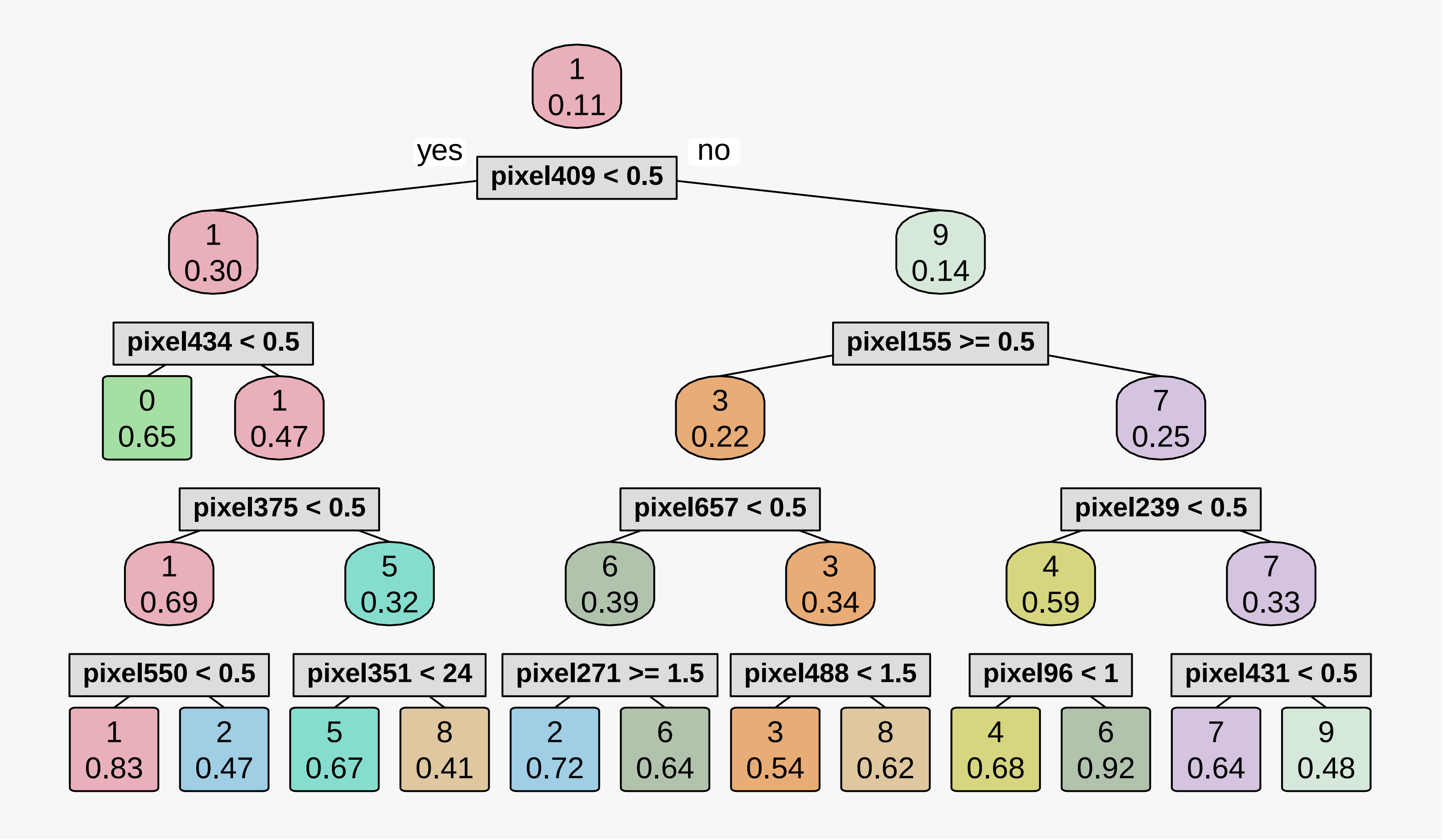
Recursive partitioning trees

```
# Load library
library(rpart)
#
# Run model
set.seed(910)
rpart = rpart(form,data=train,method="class",cp=0.01,maxdepth=4)
save(rpart,file="models/rpart_small.rda")
```

Recursive partitioning trees

```
# Load library
library(rpart.plot)
#
# Plot tree
par(bg = "#F7F7F7")
colors = c("#A6DFA3","#E9AFBA","#9FCEE5","#E8AC77","#D5D67F",
           "#86DDCE","#B1C3AD","#D4C4DF","#DFC7A0","#D6E8D9")
prp(rpart,type=2,cex=1.5,space=1.2,yspace=1.2,border.col="black",
    box.col=colors[rpart$frame$yval],leaf.round=0.3,branch.lty=1,
    split.cex=0.9,split.space=1.2,split.yspace=1.2,
    split.box.col="#DDDDDD",split.border.col="black",
    nn.box.col="#FFFFFF",nn.font=1,nn.cex=1.5, yesno.yshift=0.8,
    extra=8)
```

Recursive partitioning trees



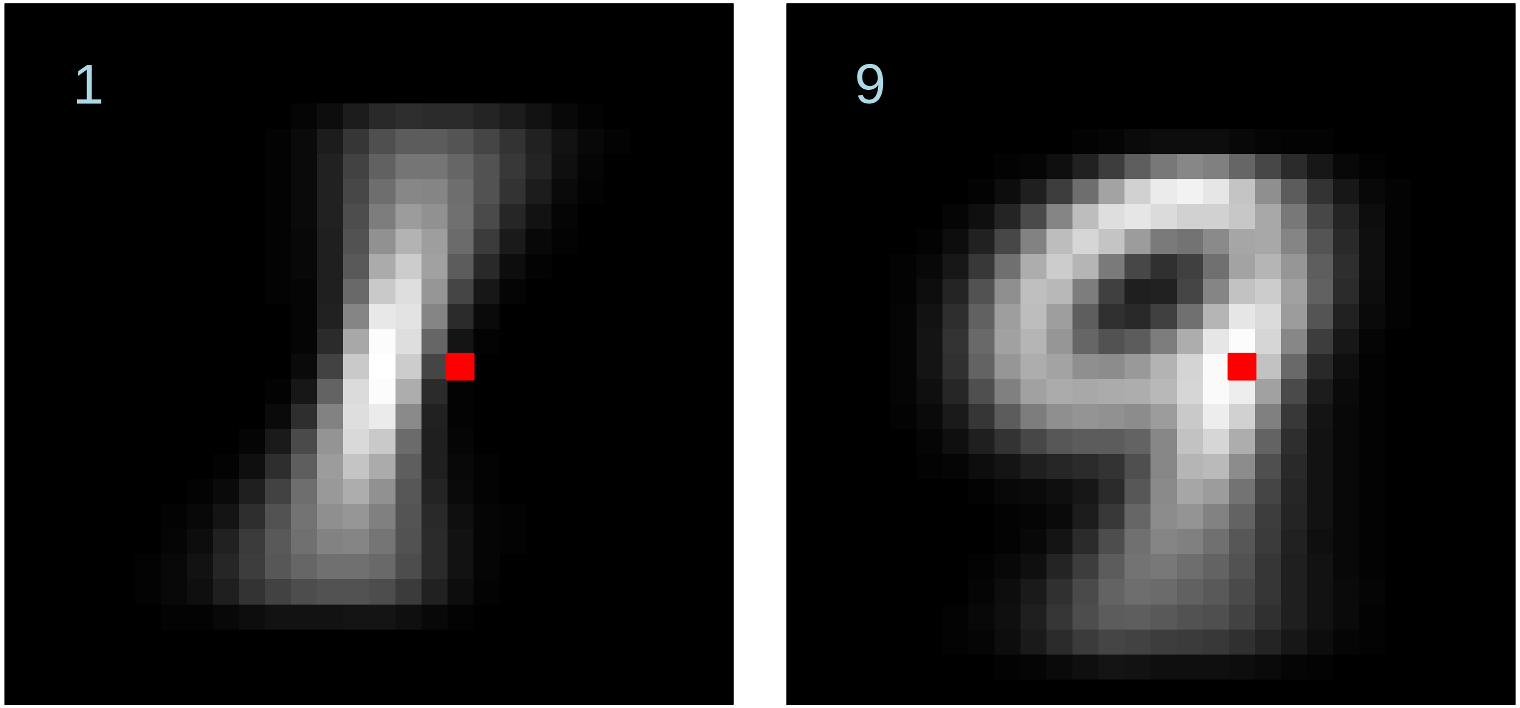
Recursive partitioning trees

```
draw.mean.fnc = function(num) {

  par(mar=c(1,1,1,1))
  pic = apply(train[train$label==num,2:785],2,mean)
  pic = matrix(pic,ncol=28,byrow=TRUE)
  pic = t(apply(pic,2,rev))
  image(pic,col=grey(level=seq(0,1,by=0.01)),xaxt="n",yaxt="n",
        useRaster=TRUE)
  text(0.1,0.9,num,col="lightblue",cex=2.5)

}
```

Recursive partitioning trees

```
# Set variables
par(mfrow=c(1,2))
par(oma=c(1,1,1,1))
left = ((409+1)%%28-1)/27-0.5*1/27
bottom = floor(28-(409+1)/28)/27-0.5*1/27
#
# Draw
draw.mean.fnc(1)
rect(left,bottom,left+1/27,bottom+1/27,col="red",border="red",lwd=2)
draw.mean.fnc(9)
rect(left,bottom,left+1/27,bottom+1/27,col="red",border="red",lwd=2)
```

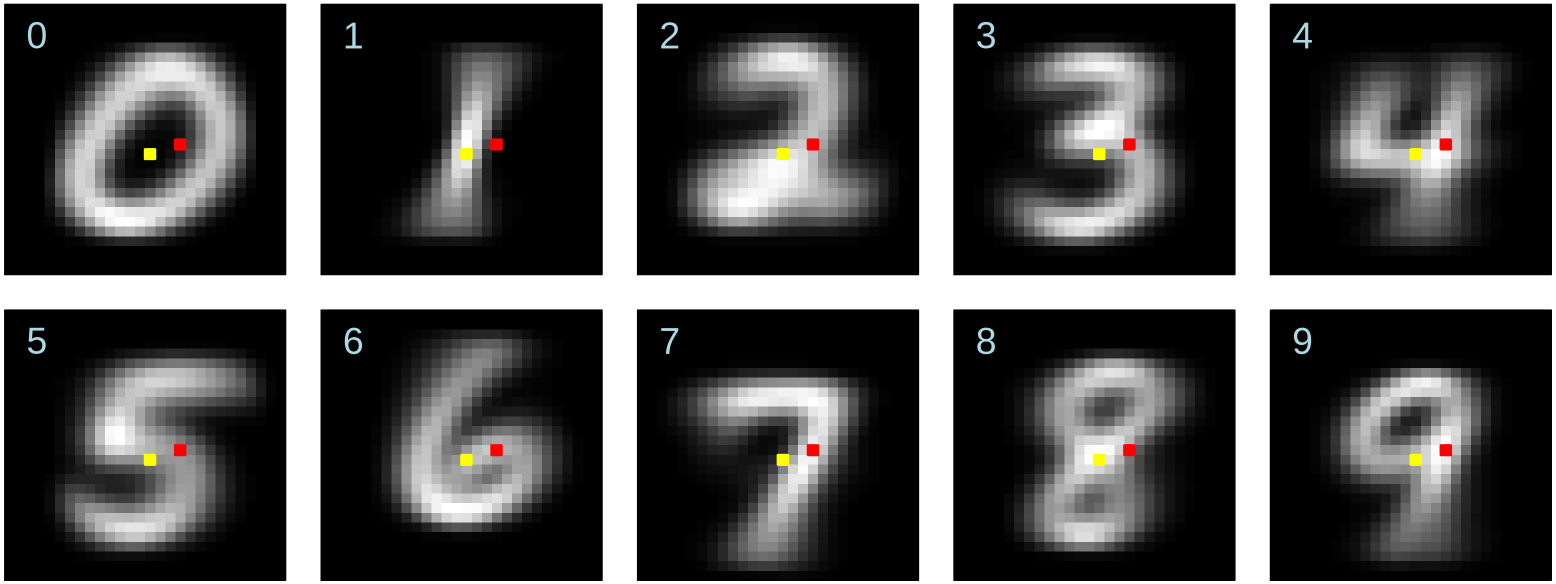

Recursive partitioning trees



Recursive partitioning trees

```
# Set variables
par(mfrow=c(2,5))
par(oma=c(1,1,1,1))
left = ((409+1)%%28-1)/27-0.5*1/27
bottom = floor(28-(409+1)/28)/27-0.5*1/27
left2 = ((434+1)%%28-1)/27-0.5*1/27
bottom2 = floor(28-(434+1)/28)/27-0.5*1/27
#
# Draw
for(i in 0:9) {
draw.mean.fnc(i)
rect(left,bottom,left+1/28,bottom+1/28,col="red",border="red",lwd=2)
rect(left2,bottom2,left2+1/28,bottom2+1/28,col="yellow",
     border="yellow",lwd=2)
}
```

Recursive partitioning trees



Recursive partitioning trees

```
# Evaluate performance
class_rpart = predict(rpart,newdata=test,type="class")
table(class_rpart==test$label)
#
# FALSE  TRUE
#  3771  6229
```

Recursive partitioning trees

```
# Load larger tree (cp = 0.00001, maxdepth = 30)
load("models/rpart.rda")
rpart$call
# rpart(formula = form, data = train, method = "class", cp = 1e-05)
#
# Performance
class_rpart = predict(rpart,newdata=test,type="class")
table(class_rpart==test$label)
#
# FALSE  TRUE
#  1447  8553
```

Recursive partioning trees

- Recursive partitioning trees tend to overfit the data
- Trees need to be pruned to increase prediction accuracy for new data
- Use cross-validation error scores to guide pruning

Recursive partitioning trees

```
# Inspect cp table
dfr = data.frame(rpart$cptable)
head(round(dfr,3))
#       CP nsplit rel.error xerror  xstd
# 1 0.092      0     1.000  1.000 0.002
# 2 0.091      1     0.908  0.918 0.002
# 3 0.072      2     0.817  0.817 0.003
# 4 0.066      3     0.745  0.745 0.003
# 5 0.063      4     0.679  0.679 0.003
# 6 0.049      5     0.616  0.616 0.003
```

Recursive partitioning trees

```
# Get cp where cross-validation error is minimal
min_num = which(dfr$xerror==min(dfr$xerror))
#
# Inspect cp table
dfr[(min_num-2):(min_num+2),]
#                CP nsplit rel.error    xerror        xstd
# 106 9.379433e-05    481 0.1014737 0.1648904 0.002224864
# 107 8.793219e-05    485 0.1010517 0.1646442 0.002223487
# 108 7.913897e-05    487 0.1008758 0.1645387 0.002222896
# 109 7.537045e-05    491 0.1005592 0.1653125 0.002227220
# 110 7.034575e-05    498 0.1000317 0.1653125 0.002227220
#
# Set cutoff value
cutoff = dfr$CP[min_num]
```

Recursive partitioning trees

```
# Prune tree
rpart = prune(rpart,cp=cutoff)
#
# Performance
class_rpart = predict(rpart,newdata=test,type="class")
table(class_rpart==test$label)
#
# FALSE  TRUE
#  1434  8566
```

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Random forests

Why plant a single tree if you can grow a forest?

Random forests

- Random forests consist of a large number of trees
- Each tree is trained on a random subset of the data and evaluated on the rest
- Each split decision is made on a random subset of the predictors
- Predictions are based on the average of all trees

Random forests

```r
# Library
library(randomForest)
# Register cluster for parallel computing
library(foreach)
library(doMC)
registerDoMC(4)
#
# Run model
rf <- foreach(ntree=rep(25, 4),.combine=combine,
              .packages="randomForest",.multicombine=TRUE) %dopar%
              randomForest(train[,2:785],train$label,ntree=ntree,
              strata=train$label,mtry=sqrt(784),
              sampsize = round(nrow(train)*0.8))
```

Random forests

```
# Load predictions from bigger model (10,000 trees)
# class_rf = predict(rf,newdata=test)
load("predictions/class_rf.rda")
# Performance
table(class_rf==test$label)
#
# FALSE  TRUE
#   348  9652
```

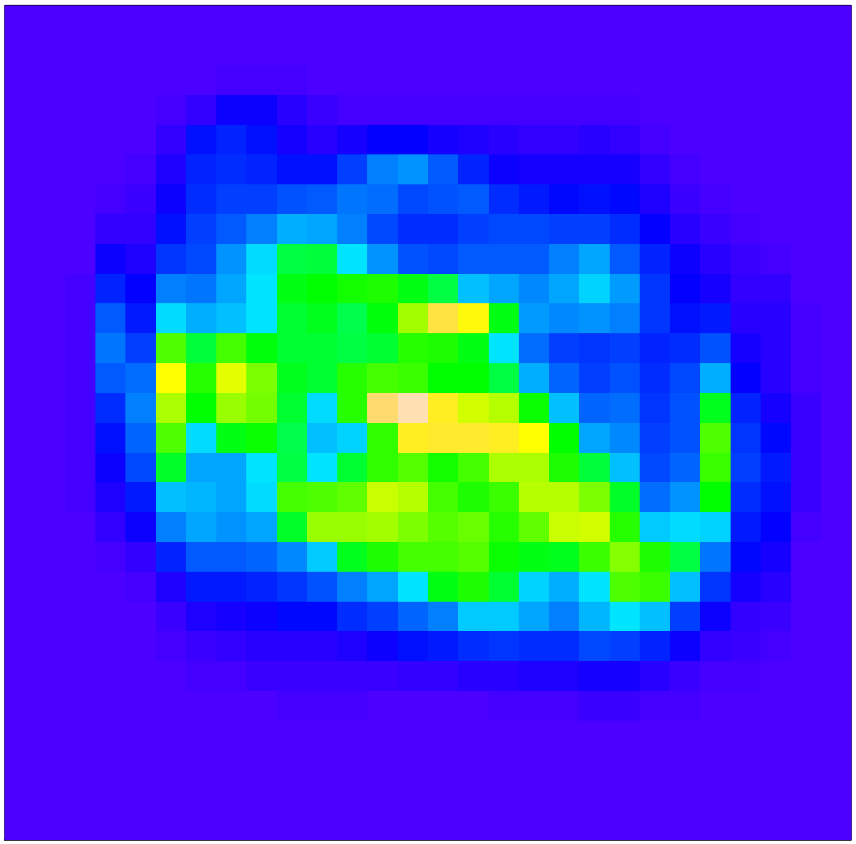
Random forests

```
# Get feature importances
# imp = importance(rf)
load("predictions/imp_rf.rda")
round(imp_rf[1:6,],3)
# pixel0 pixel1 pixel2 pixel3 pixel4 pixel5
#      0      0      0      0      0      0
#
round(imp_rf[101:106,],3)
# pixel100 pixel101 pixel102 pixel103 pixel104 pixel105
#   47.637   41.119   27.344   16.428    7.444    3.129
```

Random forests

```
# Turn into matrix
imp_rf.mat = matrix(imp_rf,ncol=28,byrow=TRUE)
imp_rf.mat = imp_rf.mat/max(imp_rf.mat)
#
# Plot matrix
par(mar=c(1,1,1,1))
pic = t(apply(imp_rf.mat,2,rev))
image(imp_rf.mat,col=topo.colors(100),xaxt="n",yaxt="n",
      useRaster=TRUE)
```

Random forests



Gradient boosting machines

- Trees in random forests are . . . random
- In gradient boosting machines trees are grown sequentially
- Each tree is an expert on the errors of the previous tree

Gradient boosting machines

- Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the data

Gradient boosting machines

- Repeat for $n = 1, 2, ..., N$:
  - Fit a tree $\hat{f}^n$ to the residuals
  - Update $\hat{f}$ by adding a shrunken version of the new tree (λ determines the amount of shrinkage):

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^n(x)$$

  - Update the residuals:

$$r_i \leftarrow r_i - \lambda \hat{f}^n(x)$$

Gradient boosting machines

- Output the gbm model: $\hat{f}(x) = \sum_{n=1}^{N} \lambda \hat{f}^{n}(x)$

Gradient boosting machines

- Parameters:
  - Number of trees (N)
  - Learning rate (λ)
  - Number of splits in the tree
  - ...

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Gradient boosting machines

```
# Library
library(gbm)
#
# Run model
gbm = gbm(form,data=train,distribution="multinomial",cv.folds=10,
          interaction.depth=20,n.trees=1000,shrinkage=0.025,
          bag.fraction=0.5,n.cores=12)
#
# Save model
save(gbm,file="models/gbm.rda")
```

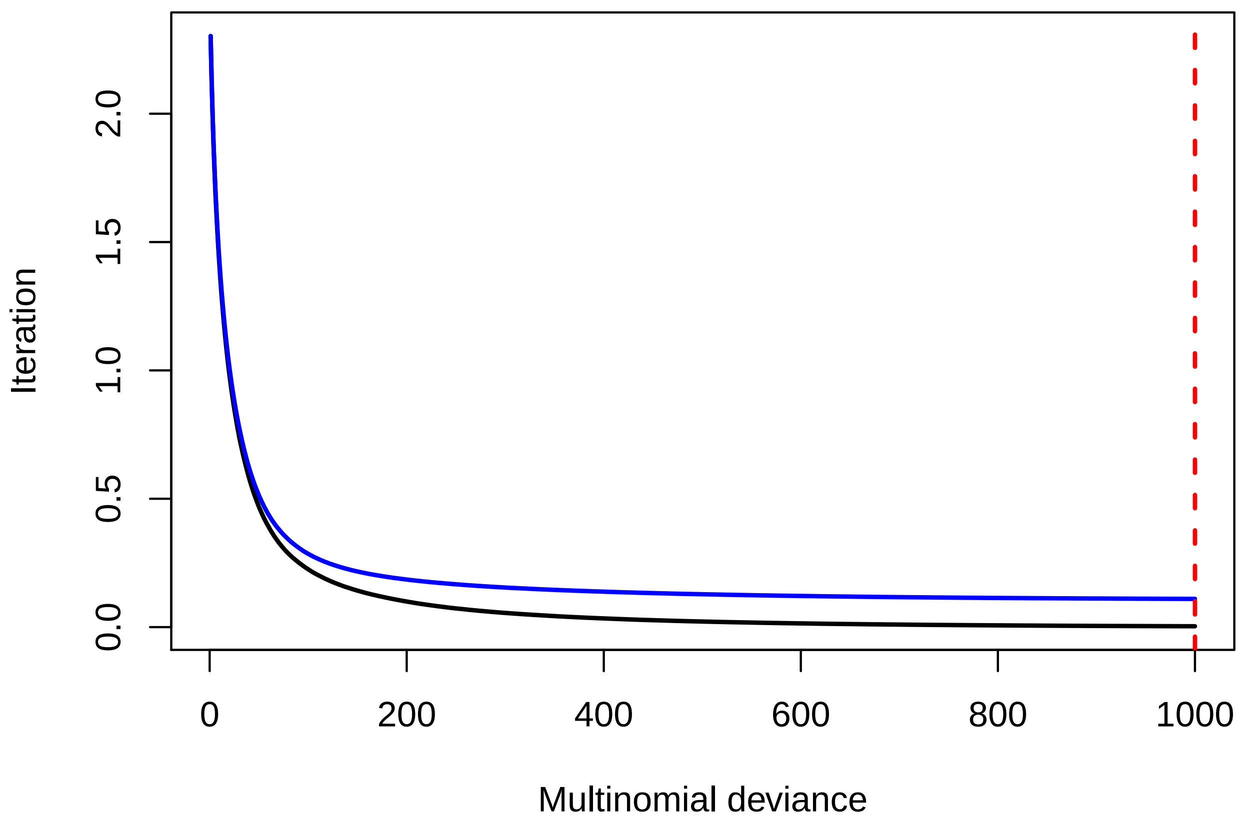
Gradient boosting machines

```
# Load model
load("models/gbm.rda")
#
# Performance
pred_gbm = predict(gbm,test,type="response")
# Using 1000 trees...
pred_gbm = matrix(pred_gbm,ncol=10)
pred_gbm = apply(pred_gbm,1,function(x) {which(x==max(x))})
pred_gbm = pred_gbm-1
table(pred_gbm==test$label)
#
# FALSE  TRUE
#   328  9672
```

Gradient boosting machines

```
# Performance plot
par(mar=c(4,4,1,1))
plot(1:1000,gbm$train.error,lwd=2,type="l",
     xlab="Multinomial deviance",ylab="Iteration")
lines(1:1000,gbm$cv.error,lwd=2,col="blue")
abline(v=1000,lty=2,lwd=2,col="red")
```

Gradient boosting machines



Gradient boosting machines

- Advantages:
  - Communication between trees
  - No overfitting
  - Excellent performance
- Disadvantage:
  - Computationally slow

Extra gradient boosting

Extra gradient boosting

```
# Xgboost works with numeric matrices as input
train.label = as.numeric(train$label)-1
train.matrix = as.matrix(sapply(train[,2:785], as.numeric))
test.matrix = as.matrix(sapply(test[,2:785], as.numeric))
```

Extra gradient boosting

```
# Library
library(xgboost)
#
# Set parameters
# Use xgb.cv to find nice parameter settings using cross-validation
param <- list("objective"="multi:softmax","eval_metric"="merror",
              "eta"=0.1, max_depth=6,"num_class"=10,"subsample"=0.8,
              "colsample_bytree"=0.90,"nthread"=10)
#
# Run model
xgb = xgboost(param=param,train.matrix,label=train.label,
              nrounds=1000)
#
# Save model
xgb.save(xgb,"models/xgb.rda")
```

Extra gradient boosting

```
# Load model
xgb = xgb.load("models/xgb.rda")
#
# Performance
# class_xgb = predict(xgb,newdata=test)
load("predictions/class_xgb.rda")
table(test$label==class_xgb)
#
# FALSE  TRUE
#   293  9707
```


Boosting

- Advantages:
    - Excellent performance
    - No overfitting
    - Computationally efficient
- Disadvantages:
    - Interpretability?

Boosting

“So, in conclusion: this is magic, you always want to use it.”

Patrick Winston
MIT Open Courseware

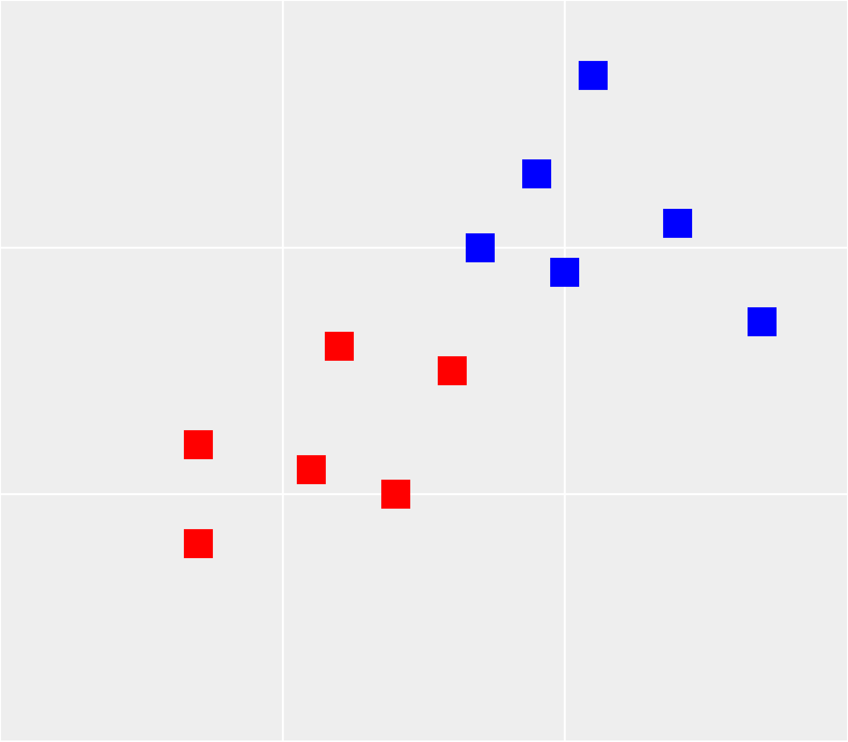
Outline

- MNIST database
- Multinomial logistic regression
- Decision trees
- k-Nearest neighbors
- Support vector machines
- Neural networks

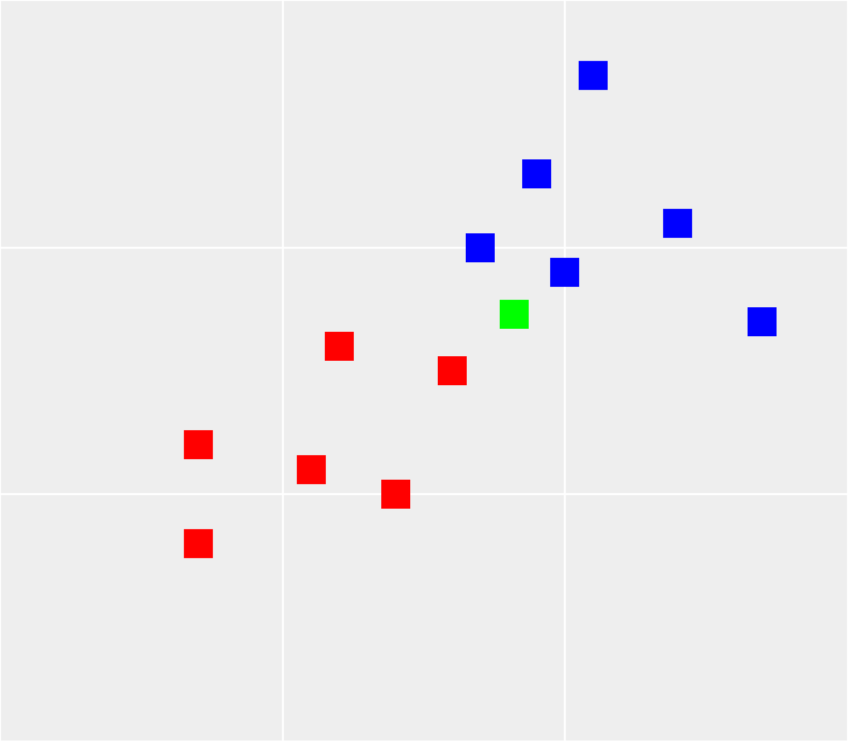
k-Nearest neighbors

- Store all instances encountered during training in memory
- Present new instance during test
- Find the k most similar example(s) in the training data using some distance metric $\Delta(X, Y)$
- Assign the most frequent class within the set of most similar example(s) (the k-nearest neighbours) to the new instance

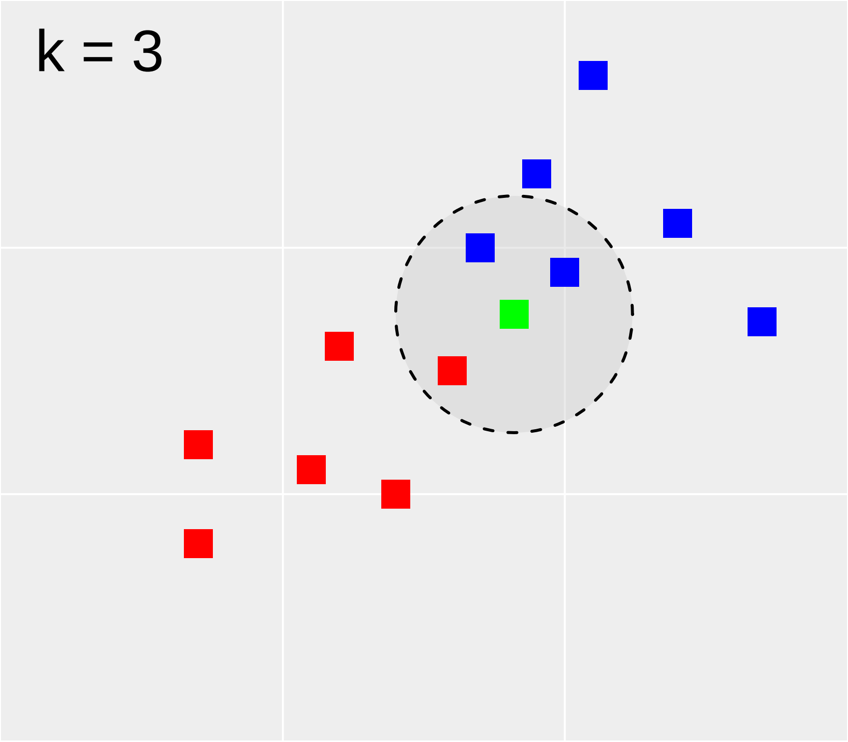
k-Nearest neighbors



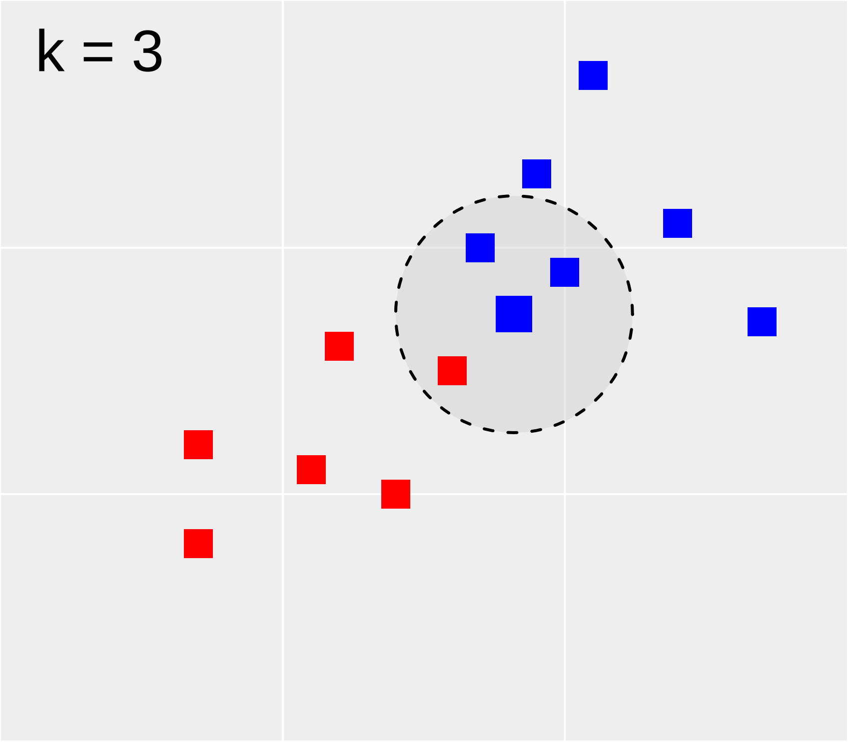
k-Nearest neighbors



k-Nearest neighbors



k-Nearest neighbors



k-Nearest neighbors

```
# Set a test item to look at
test.item = 38
#
# Define function to calculate distances
getDistance.fnc = function(num.train,num.test) {
  test = test[num.test,2:785]
  train = train[num.train,2:785]
  dist = sqrt(sum((test-train)^2))
  dist = round(dist,2)
  return(dist)
}
```

k-Nearest neighbors

```
# Calculate some distances
test$label[38]
# [1] 3
# Levels: 0 1 2 3 4 5 6 7 8 9
getDistance.fnc(test.item,which(train$label=="3")[1])
# [1] 2265.39
getDistance.fnc(test.item,which(train$label!="3")[1])
# [1] 2899.83
```

k-Nearest neighbors

```r
# Libraries:
library(doMC)
cluster = makeCluster(20)
#
# Define function to get neighbors
getNeighbors.fnc = function(num.test,k,train.rows=100) {
  distances = unlist(mclapply(1:train.rows,getDistance.fnc,
                      num.test=num.test,mc.cores=20))
  names(distances) = 1:train.rows
  neighbors = sort(distances)[1:k]
  neighbors = as.numeric(names(neighbors))
  return(neighbors)
}
```

k-Nearest neighbors

```
# Get neighbors
neighbors = getNeighbors.fnc(38,9,train.rows=500)
neighbors
# [1] 302 156  28 157 100  35 264 155 388
train$label[neighbors]
# [1] 3 3 3 5 3 3 3 2 8
# Levels: 0 1 2 3 4 5 6 7 8 9
```

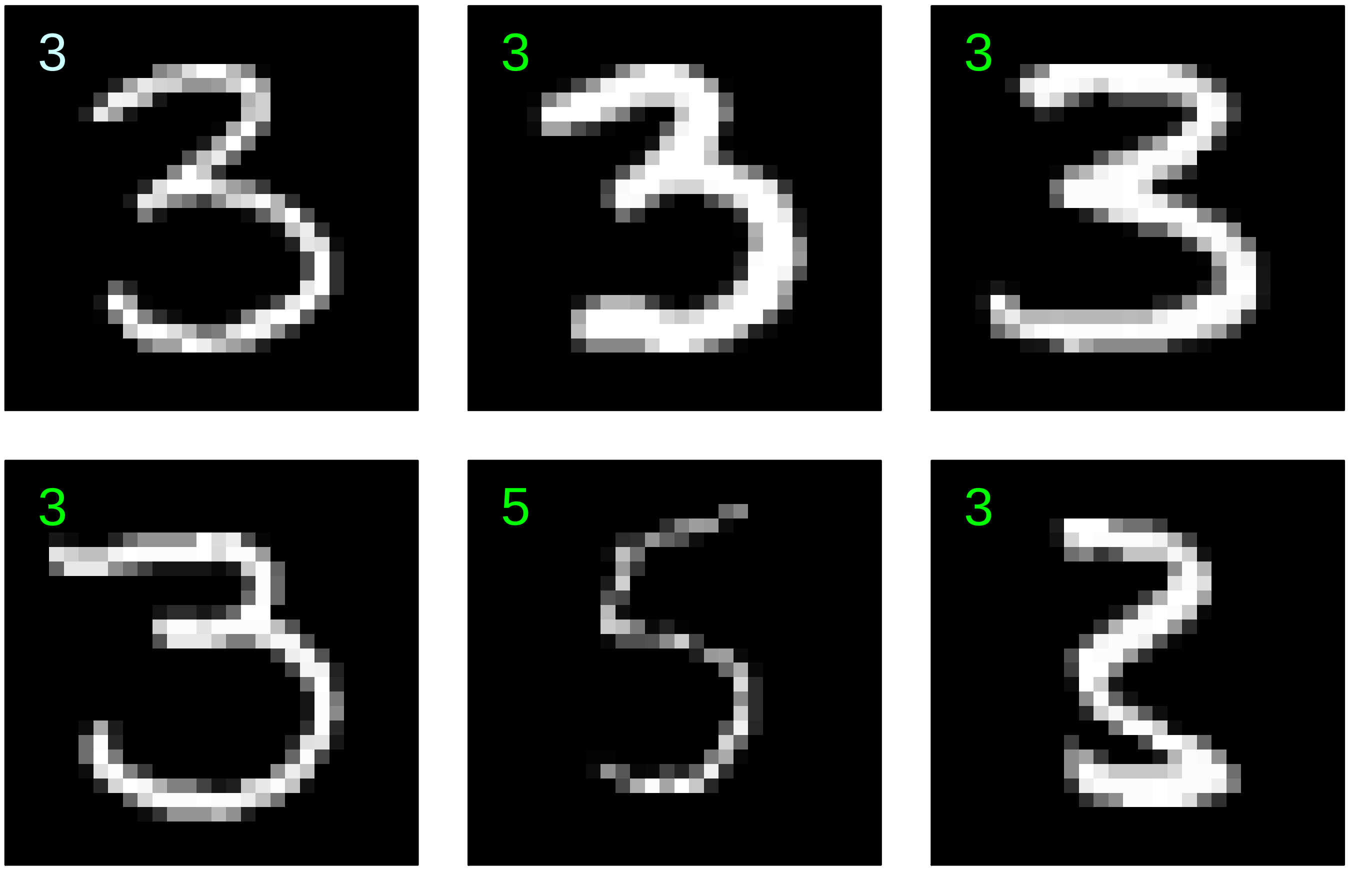
k-Nearest neighbors

```
# Create a general function for drawing
draw.train.fnc = function(num) {

  par(mar=c(1,1,1,1))
  pic = as.numeric(train[num,2:785])
  pic = matrix(pic,ncol=28,byrow=TRUE)
  pic = t(apply(pic,2,rev))
  image(pic,col=grey(level=seq(0,1,by=0.01)),xaxt="n",yaxt="n",
        useRaster=TRUE)
  text(0.1,0.9,train$label[num],col="green",cex=2.5)

}
```

k-Nearest neighbors

```
# Draw
par(mfrow=c(2,3))
par(oma=c(1,1,1,1))
invisible(sapply(test.item,draw.fnc))
invisible(sapply(neighbors[1:5],draw.train.fnc))
```

k-Nearest neighbors



k-Nearest neighbors

- What if the vote is a tie?
- Solutions:
  - random selection
  - increase k until the tie is broken
  - decrease k until the tie is broken
  - ...

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

k-Nearest neighbors

```r
# Library
library(class)
#
# Run model
knn = knn(train.norm[,2:785],test.norm[,2:785],cl=train$label,k=1)
```

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

k-Nearest neighbors

```
# Load model
load("models/knn.rda")
#
# Performance
table(knn==test$label)
#
# FALSE  TRUE
#   325  9675
```

Random k-Nearest neighbors

- Random forests build a collection of trees on random subsets of the data
- Many weak classifiers combined into a strong classifier
- The same idea can be applied to k-Nearest Neighbors

k-Nearest neighbors

```r
# Library
library(rknn)
#
# Set up cluster for parallel computing
library(doMC)
cluster = makeCluster(20)
#
# Run model
rknn = rknn(data=train.norm[,2:785],newdata=test.norm[,2:785],
            y=train$label,k=3,r=200,mtry=round(0.5*784),
            cluster=cluster,seed=31415)
#
# Save model
save(rknn,file="models/rknn.rda")
```

k-Nearest neighbors

```
# Load model
load("models/rknn.rda")
#
# Performance
table(rknn$pred==test$label)
#
# FALSE  TRUE
#   299  9701
```

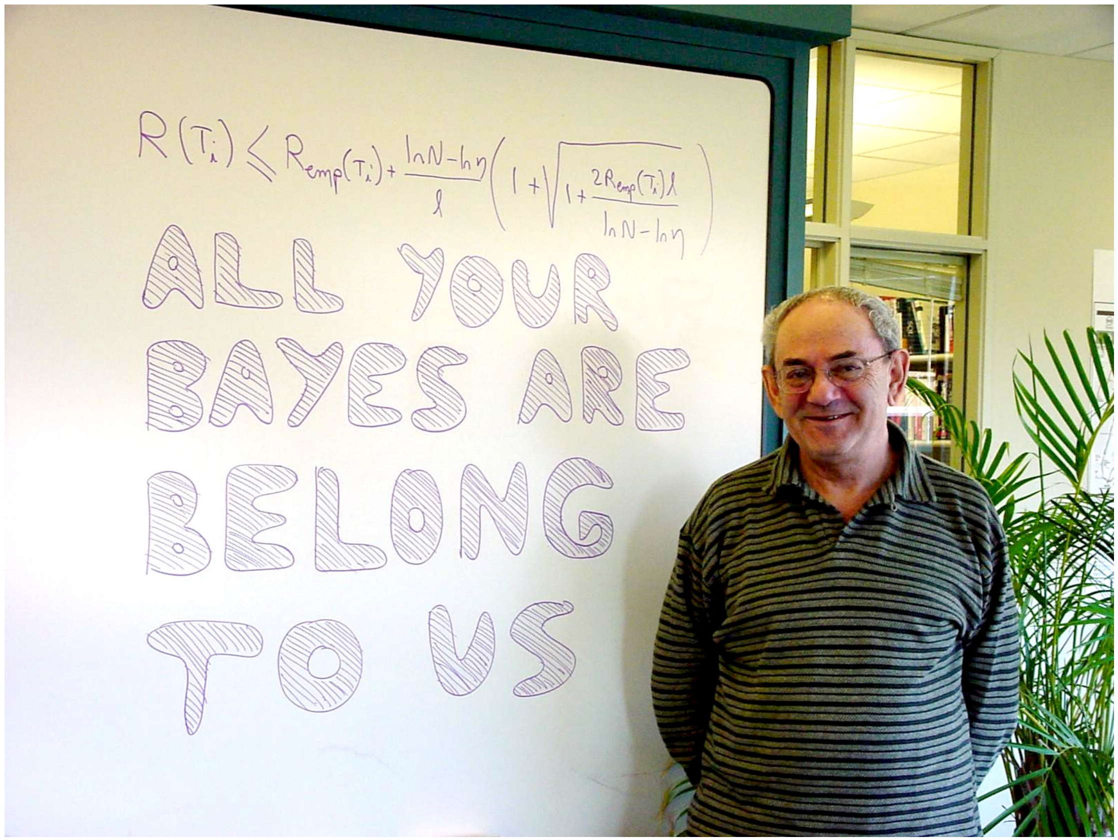

k-Nearest neighbors

- Advantages:
  - Competitive performance
- Disadvantages:
  - Computationally expensive
  - Intransparent

Outline

- MNIST database
- Multinomial logistic regression
- Decision trees
- k-Nearest neighbors
- Support vector machines
- Neural networks

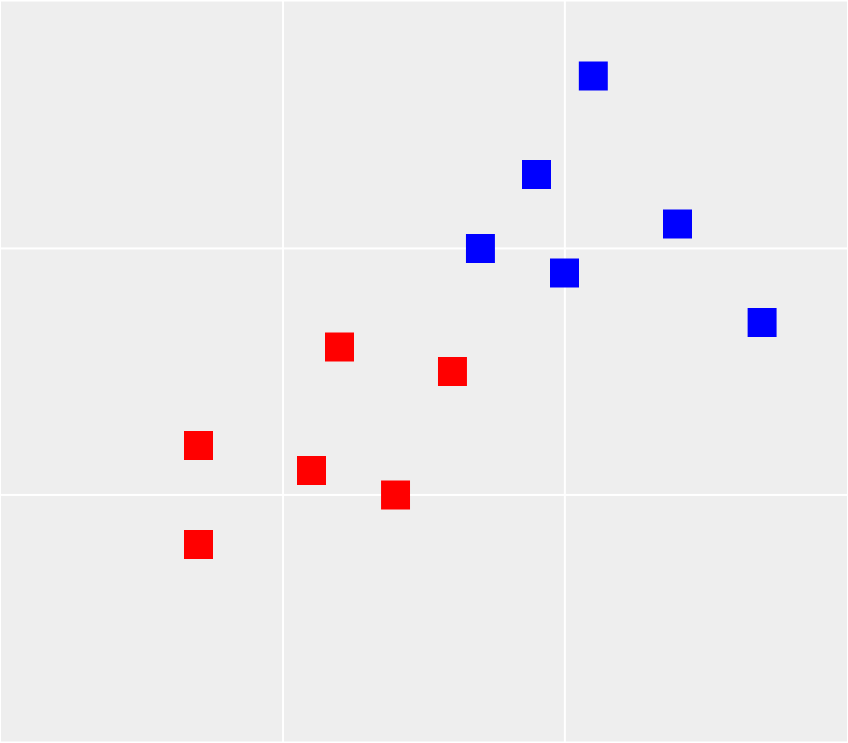
Support vector machines



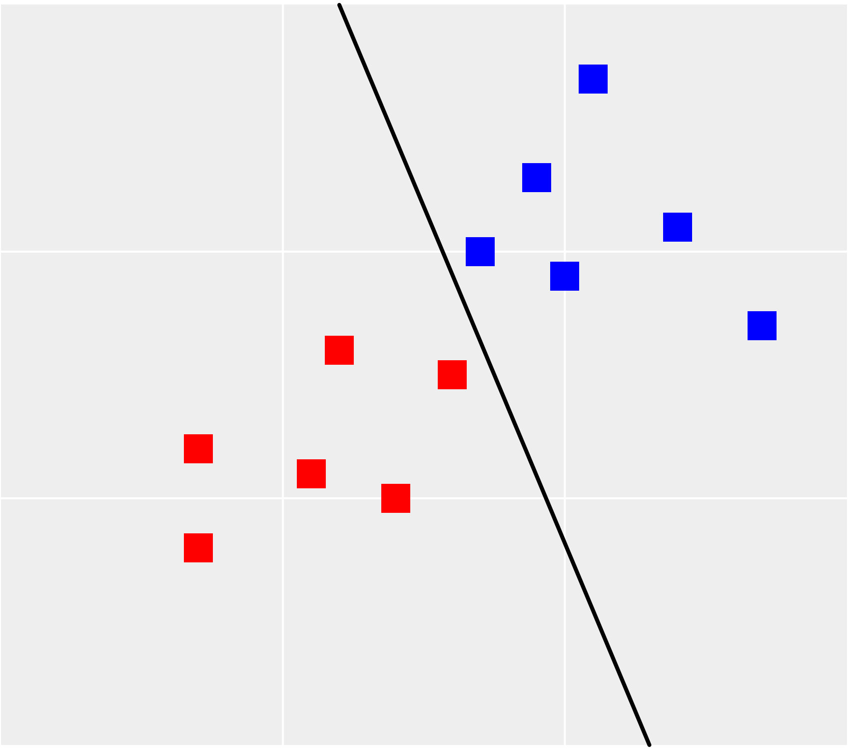
Support vector machines

- Find a hyperplane $\vec{w}$ that best seperates the classes
- Support vectors are the data points closest to $\vec{w}$
- The support vectors determine the location of $\vec{w}$
- Classify new instance based on its location relative to $\vec{w}$

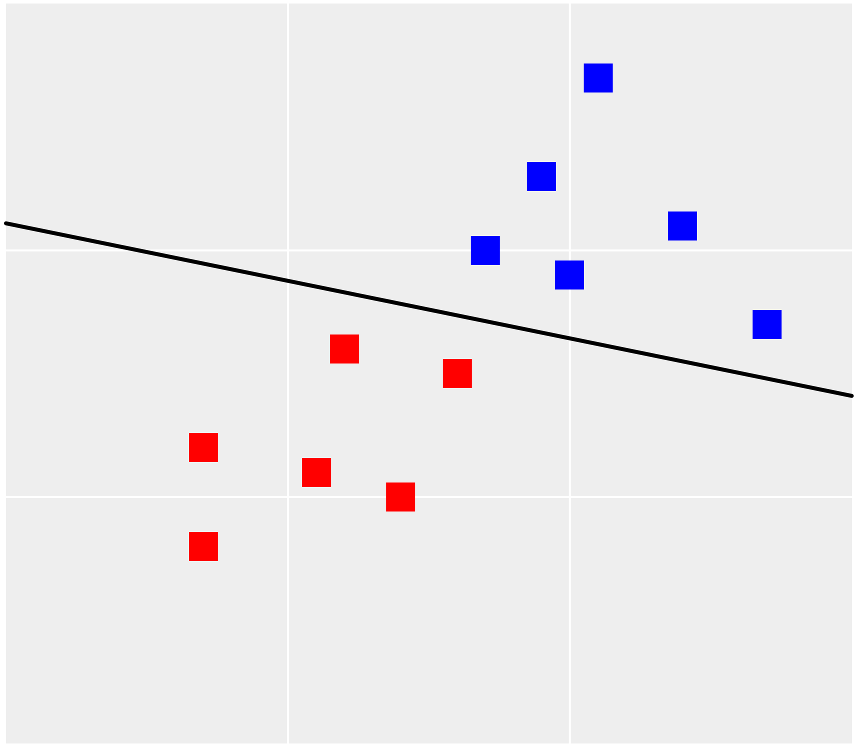
Support vector machines



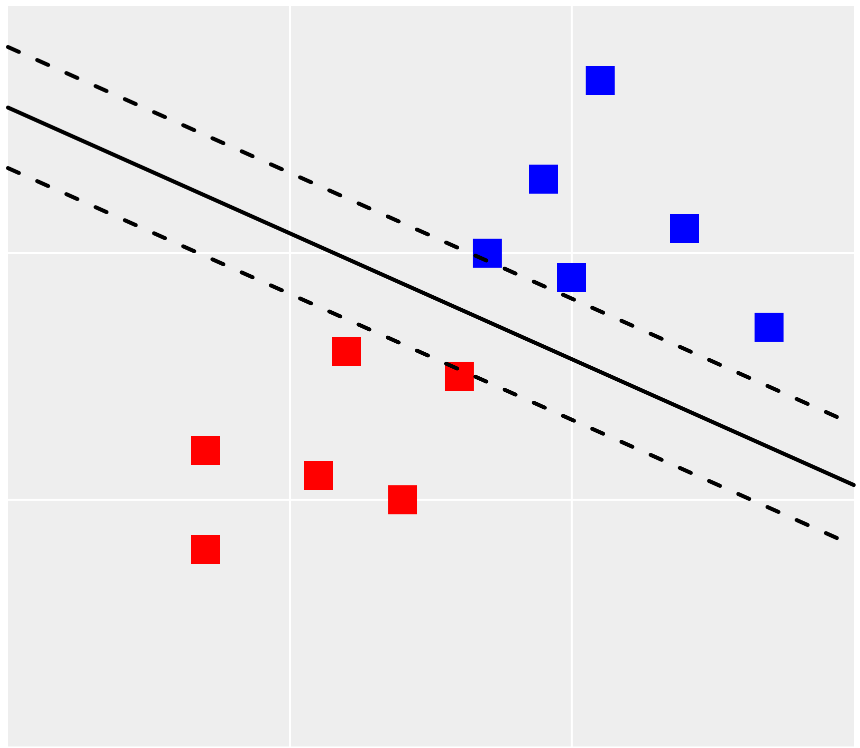
Support vector machines



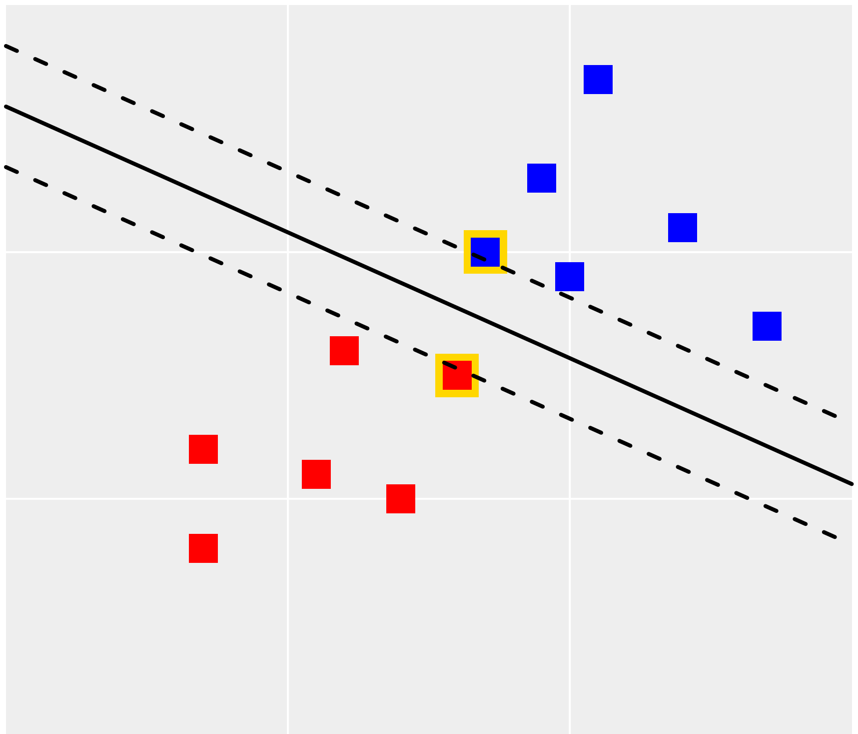
Support vector machines



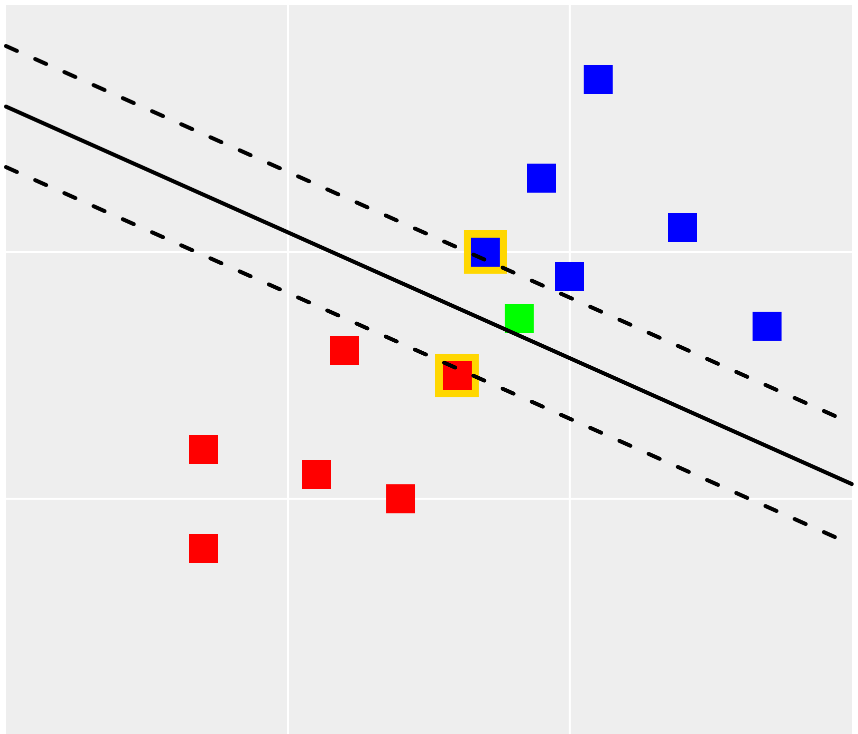
Support vector machines



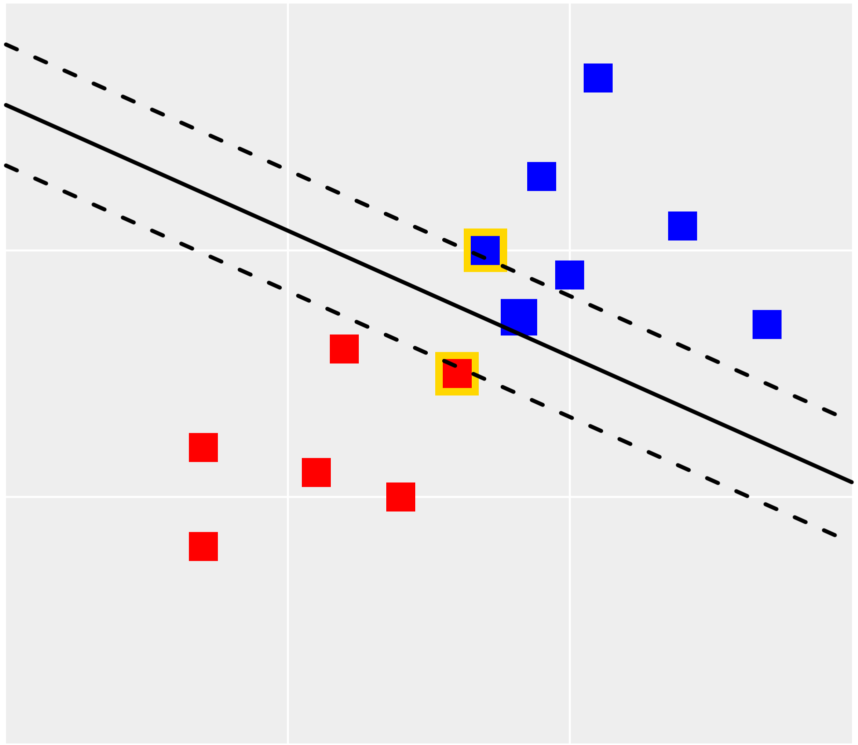
Support vector machines



Support vector machines



Support vector machines

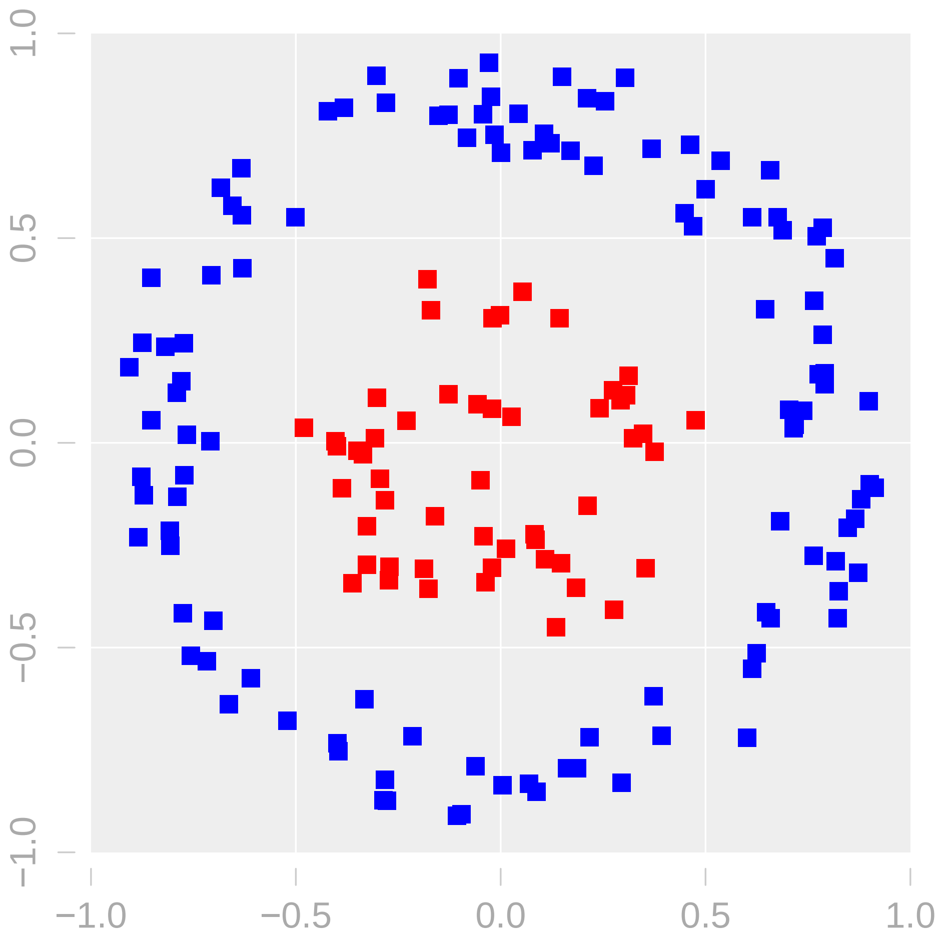


Support vector machines

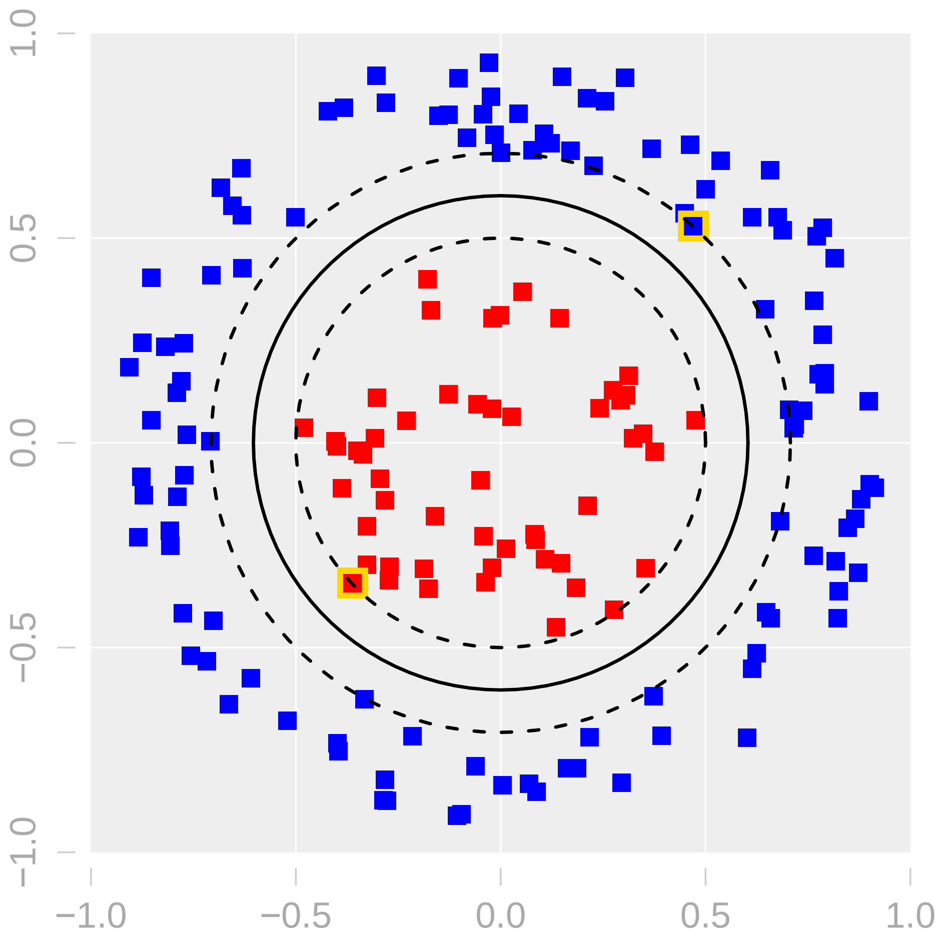
- Sometimes data are not linearly seperable in $\mathbb{R}^N$
- Use a transformation function to project the data to a higher dimension $\mathbb{R}^M$ in which they are linearly seperable
- Kernel functions (implicitly) transforms the data to $\mathbb{R}^M$
- This allows support vector machines to learn a separating hyperplane $\vec{w}$ that is linear in $\mathbb{R}^M$, but non-linear in $\mathbb{R}^N$

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

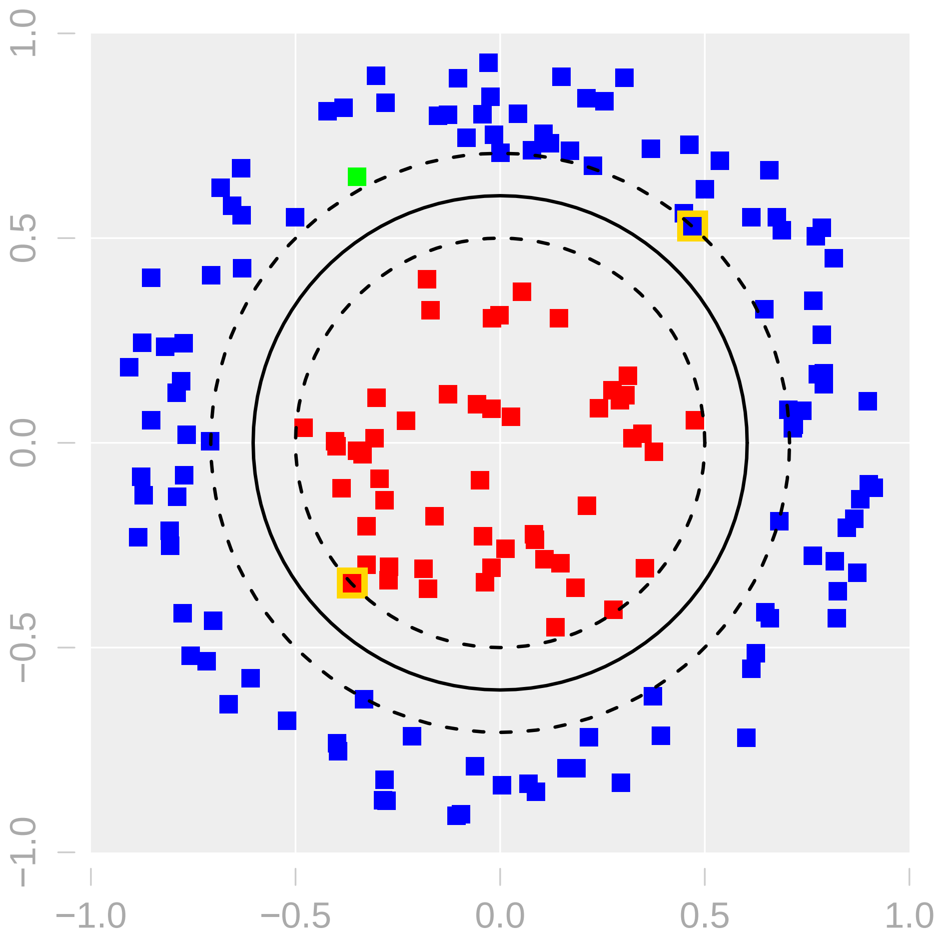
Support vector machines



Support vector machines



Support vector machines

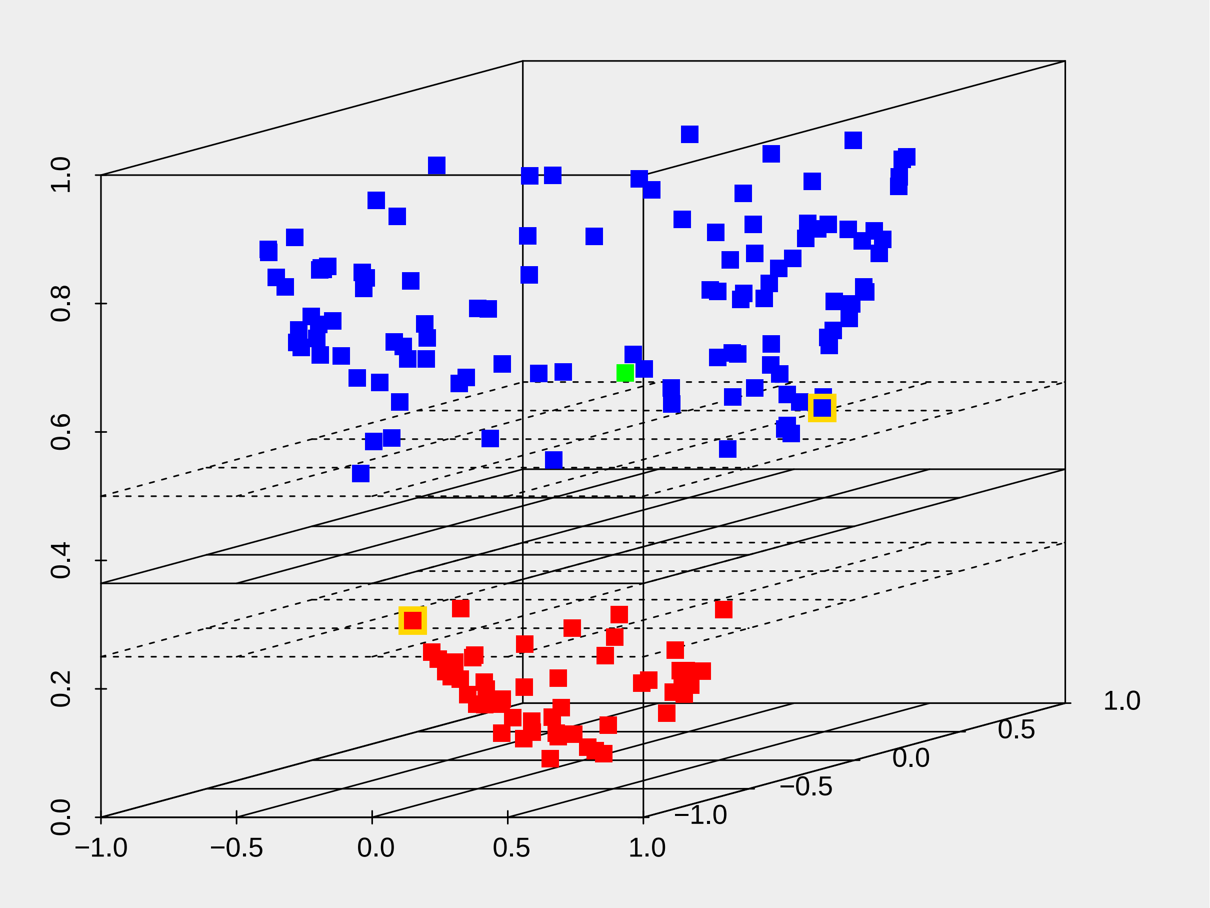


Support vector machines

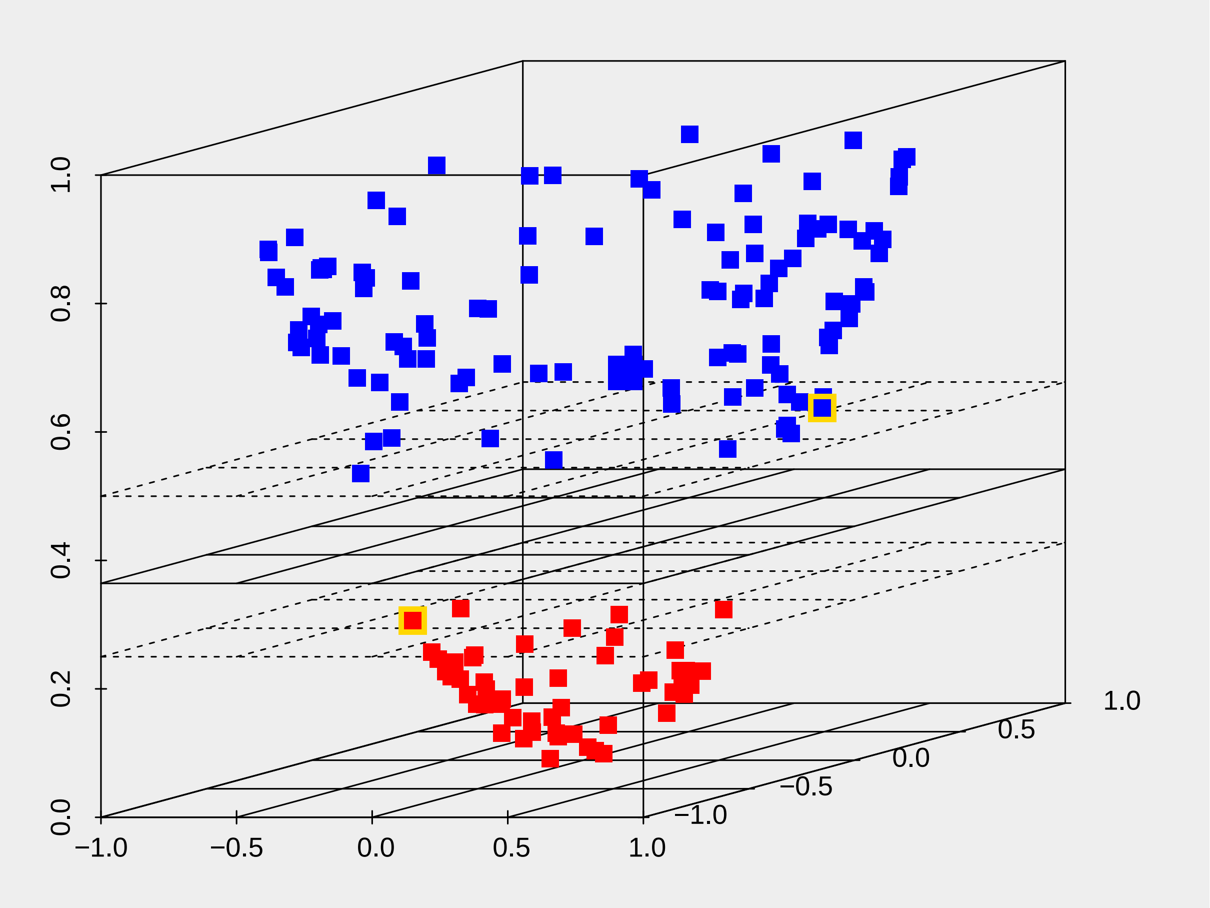
- Define a third dimension:

$$z = x^2 + y^2$$

Support vector machines



Support vector machines



Support vector machines

"- ich würde sagen: du bist ein Tor! Du suchst, was hienieden nicht zu finden ist! Aber ich habe sie gehabt, ich habe das Herz gefühlt, die große Seele, in deren Gegenwart ich mir schien mehr zu sein, als ich war, weil ich alles war, was ich sein konnte. Guter Gott! Blieb da eine einzige Kraft meiner Seele ungenutzt?"

Johann Wolfgang von Goethe
Die Leiden des jungen Werther

Support vector machines

- Support vector machines are designed for binary classification

- Extension to multiclass classification:
    - one versus all
    - one versus one

Support vector machines

```
# Library
library(e1071)
#
# Run model
svm <- svm(form,data.matrix(train.norm),type="C-classification",
           kernel="radial",cost=10,gamma=0.025)
#
# Save model
save(svm,file="models/svm.rda")
```

Support vector machines

```
# Load model
load("models/svm.rda")
#
# Performance
# class_svm = predict(svm,newdata=test.norm)
# class_svm = as.numeric(as.character(class_svm))-1
load("predictions/class_svm.rda")
table(class_svm==test$label)
#
# FALSE  TRUE
#   199  9801
```

Support vector machines

- Advantages:
  - Excellent performance
  - Kernels give flexibility
- Disadvantages:
  - Data are not always linearly separable, even in higher dimensions
  - Overfitting
  - Intransparent

Outline

- MNIST database
- Multinomial logistic regression
- Decision trees
- k-Nearest neighbors
- Support vector machines
- Neural networks

Neural networks

- Learning network
- Three components:
    - input units
    - hidden layer(s)
    - output units
- Connections between units in subsequent layers

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Neural networks

- Output of units determined by activation function
- Sigmoid activation function:

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

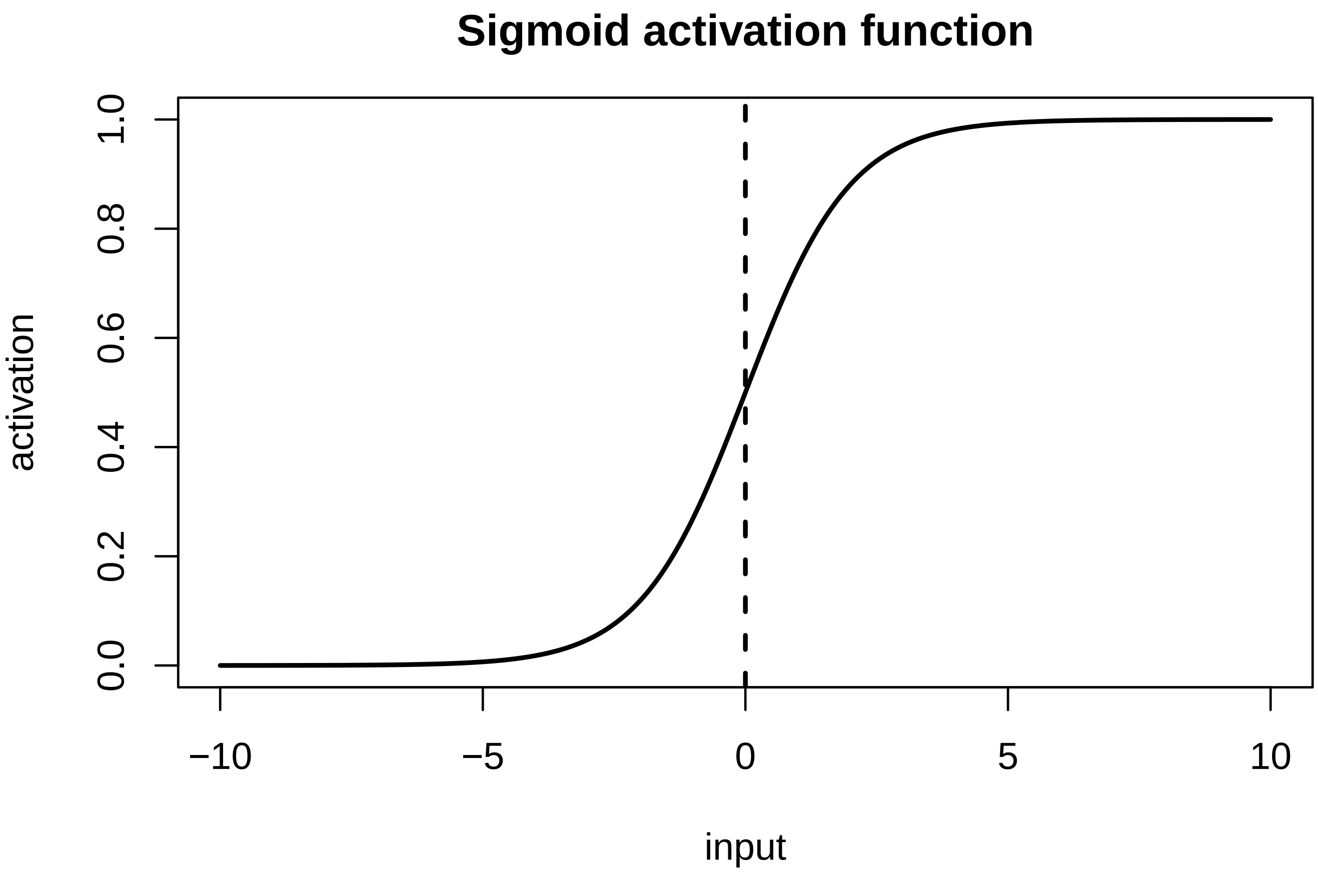
- The derivative of the activation function determines how the weights are adjusted
- Derivative of sigmoid activation function:

$$f'(x) = f(x) * (1 - f(x))$$

Neural networks

```
# Define sigmoid activation function
sigmoid.fnc = function(x) {1/(1+exp(1)^-x)}
# Plot
par(mar=c(4,4,3,1))
x = seq(-10,10,by=0.01)
y = sigmoid.fnc(x)
plot(x,y,type="l",lwd=2,xlab="input",ylab="activation",
     main = "Sigmoid activation function")
abline(v=0,lty=2,lwd=2)
```

Neural networks



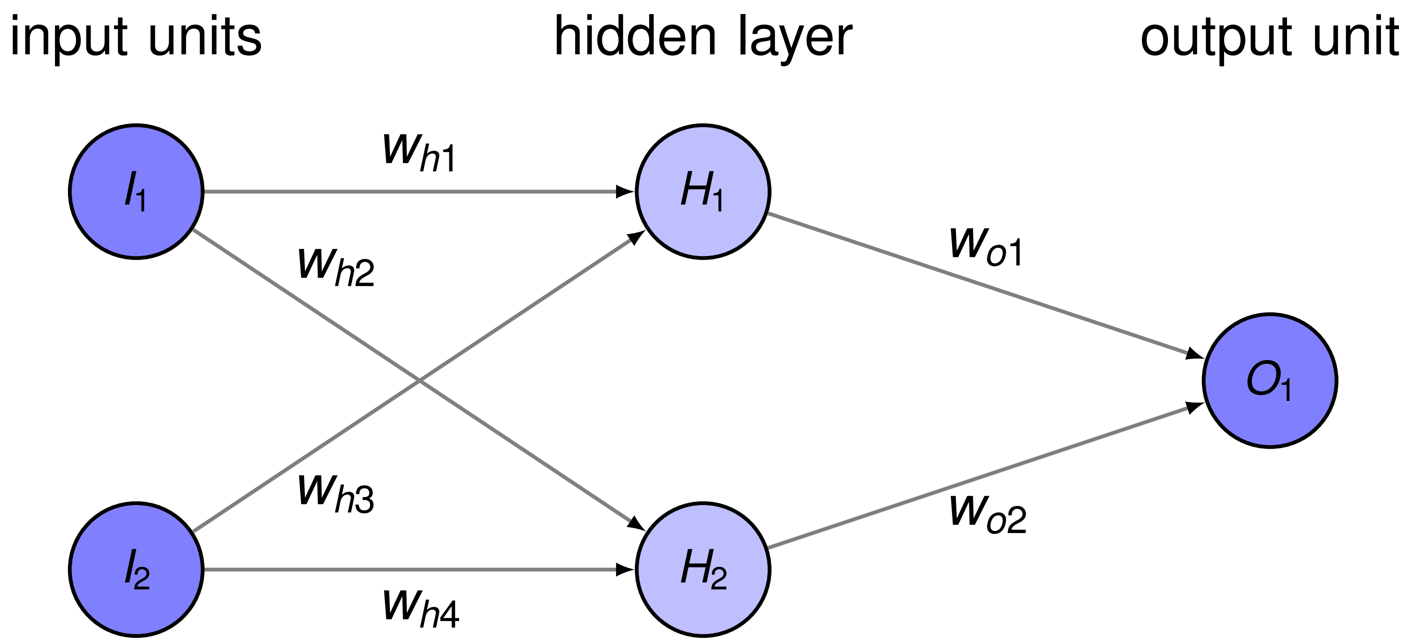
Neural networks

- In feed-forward neural networks a learning event consists of two steps:

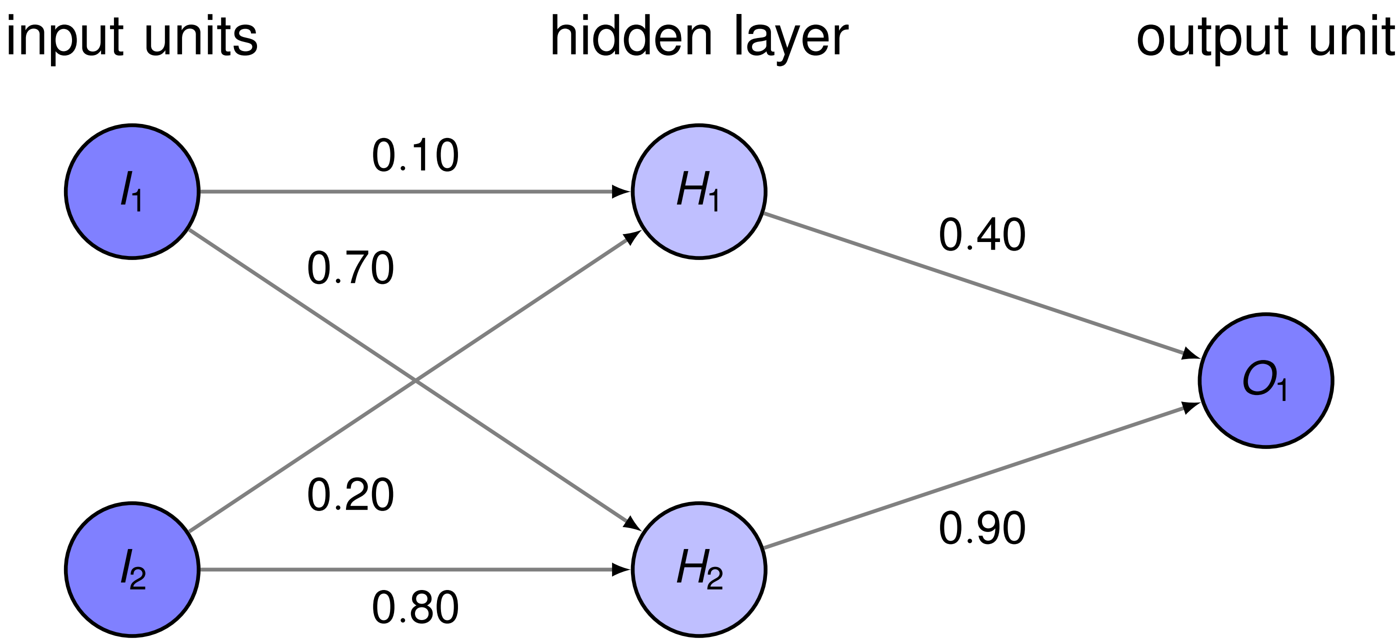
  1) Forward pass of input through the network

  2) Update connection weights on the basis of prediction error

Neural networks



Neural networks

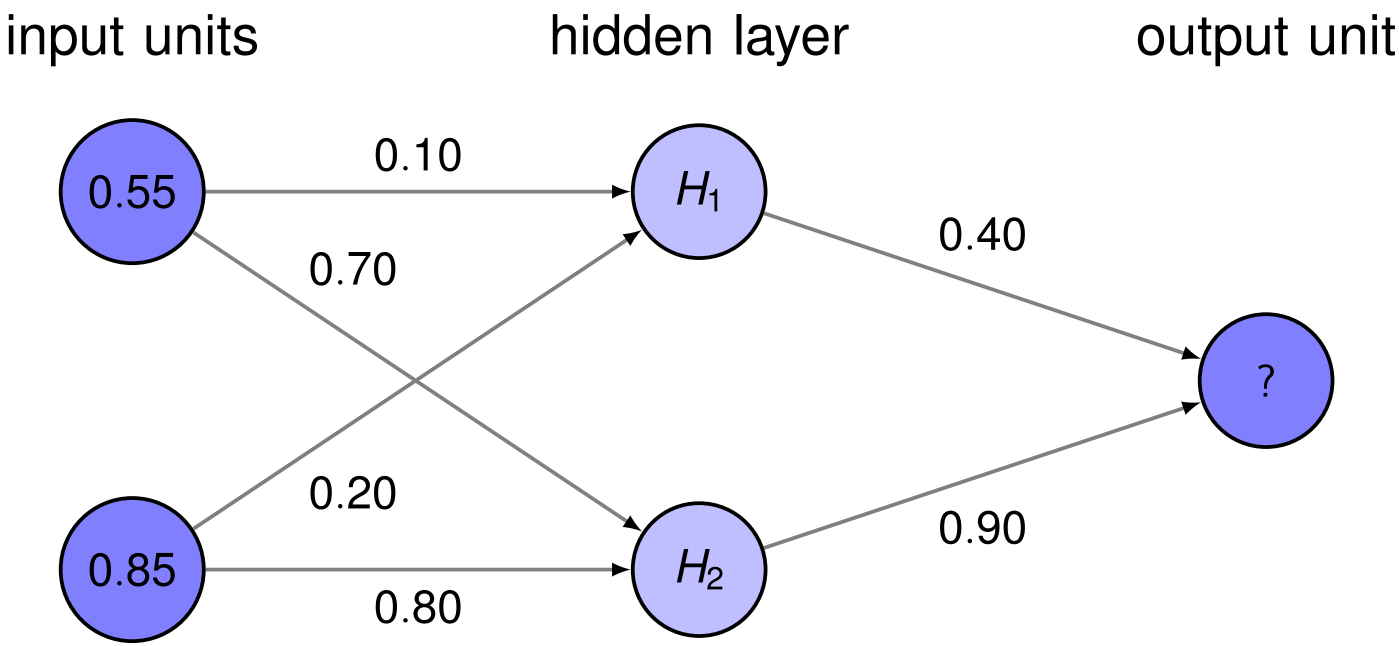


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Neural networks

- Event:
    - Feature 1: 0.55
    - Feature 2: 0.85
    - Output: 1

Neural networks



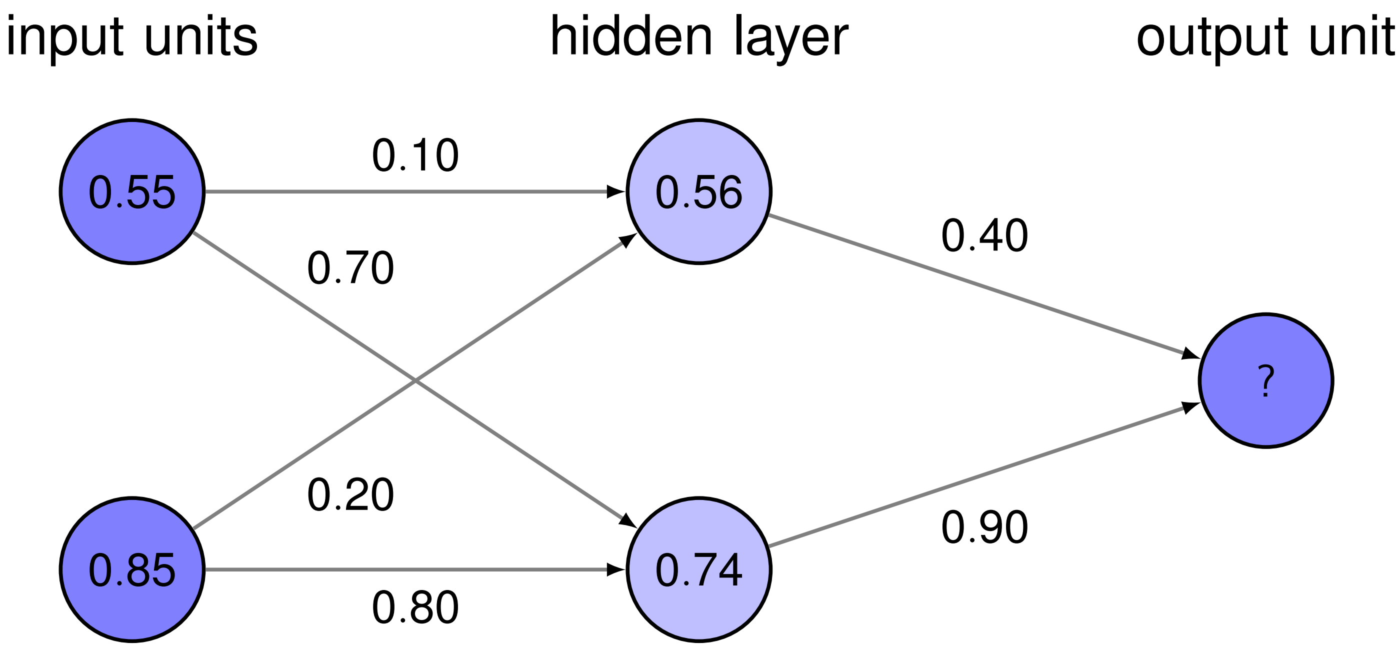
Neural networks

```
# Set input activations
act_i1= 0.55; act_i2 = 0.85
#
# Set weights for hidden layer connections
w_h1 = 0.10; w_h2 = 0.70; w_h3 = 0.20; w_h4 = 0.80
#
# Set weights for output connections
w_o1 = 0.40; w_o2 = 0.90
```

Neural networks

```
# Calculate activation of H1
act_h1 = act_i1*w_h1 + act_i2*w_h3
act_h1
# [1] 0.225
act_h1 = sigmoid.fnc(act_h1)
act_h1
# [1] 0.5560139
#
# Calculate activation of H2
act_h2 = act_i1*w_h2 + act_i2*w_h4
act_h2
# [1] 1.065
act_h2 = sigmoid.fnc(act_h2)
act_h2
# [1] 0.7436449
```

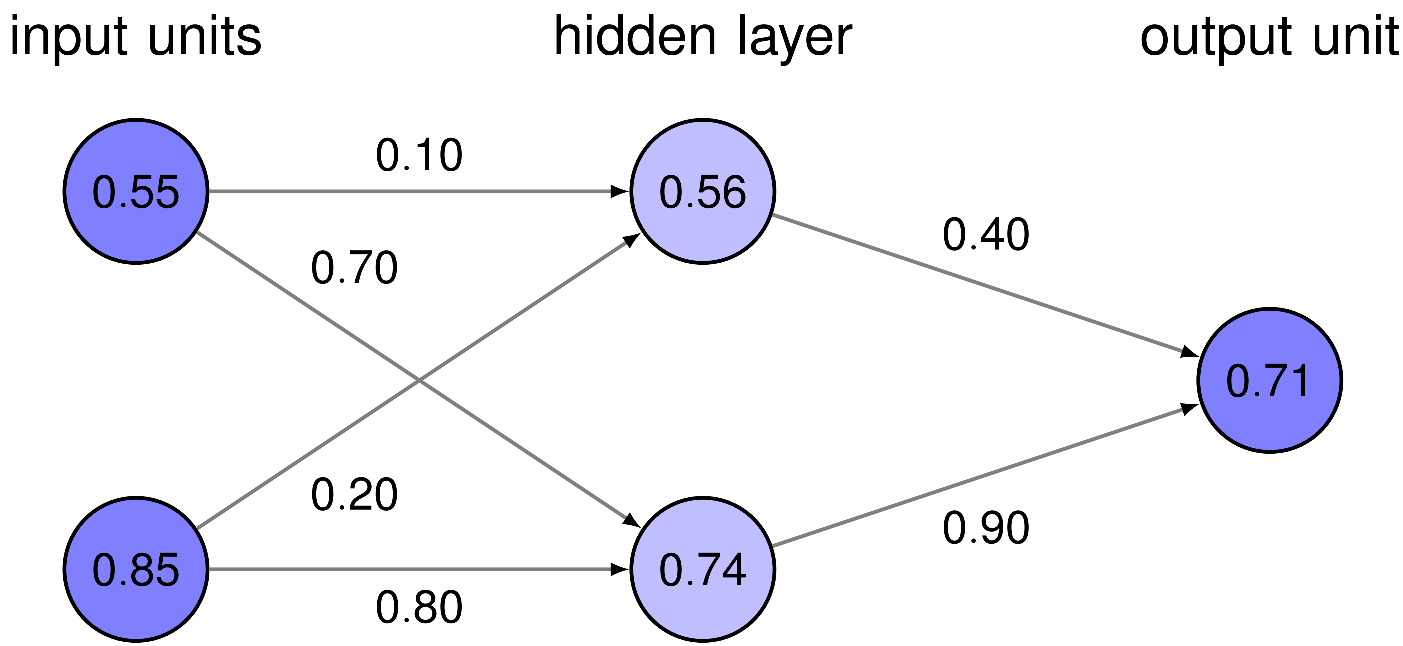
Neural networks



Neural networks

```
# Calculate activation of O1
act_o1 = act_h1*w_o1 + act_h2*w_o2
act_o1
# [1] 0.891686
act_o1 = sigmoid.fnc(act_o1)
act_o1
# [1] 0.709238
```

Neural networks



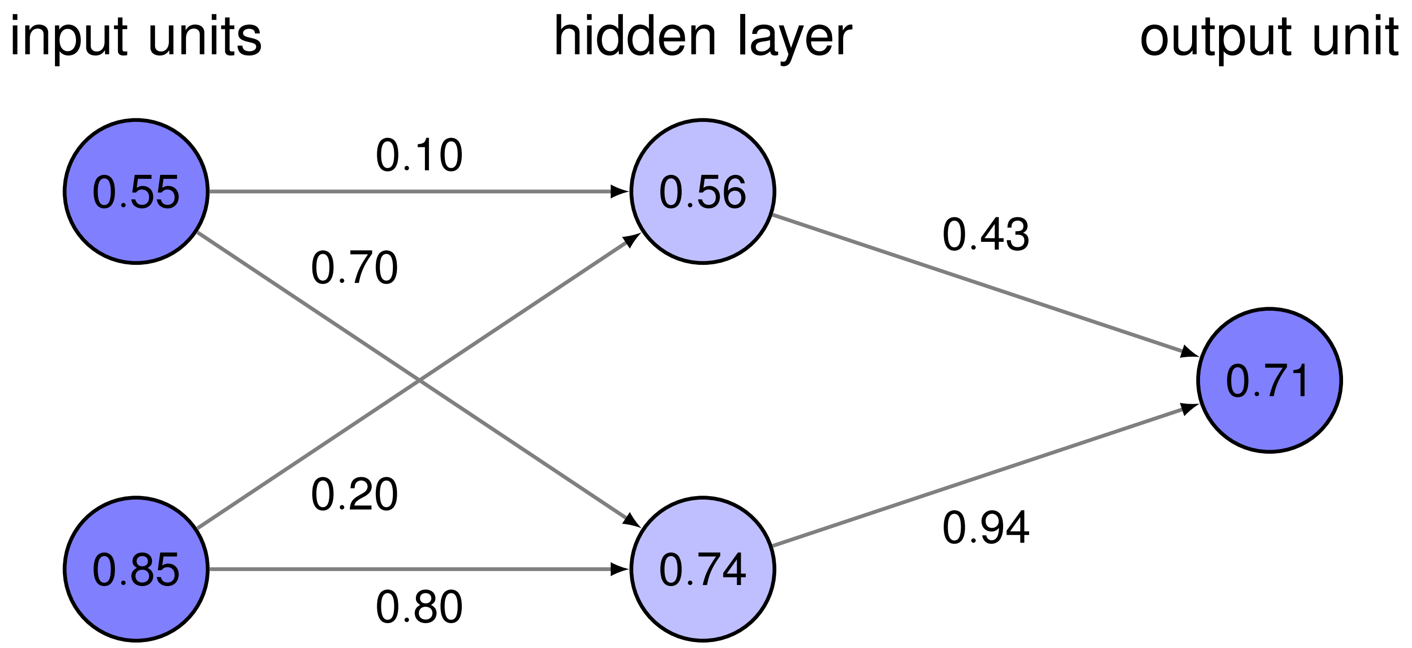
Neural networks

```
# Calculate output error
o_error_o1 = ((act_o1)*(1-act_o1)) * (1-act_o1)
o_error_o1
# [1] 0.05996079
```

Neural networks

```
# Calculate new w_o1
w_o1_new = w_o1 + (act_h1 * o_error_o1)
w_o1_new
# [1] 0.433339
# Calculate new w_o2
w_o2_new = w_o2 + (act_h2 * o_error_o1)
w_o2_new
# [1] 0.9445895
```

Neural networks



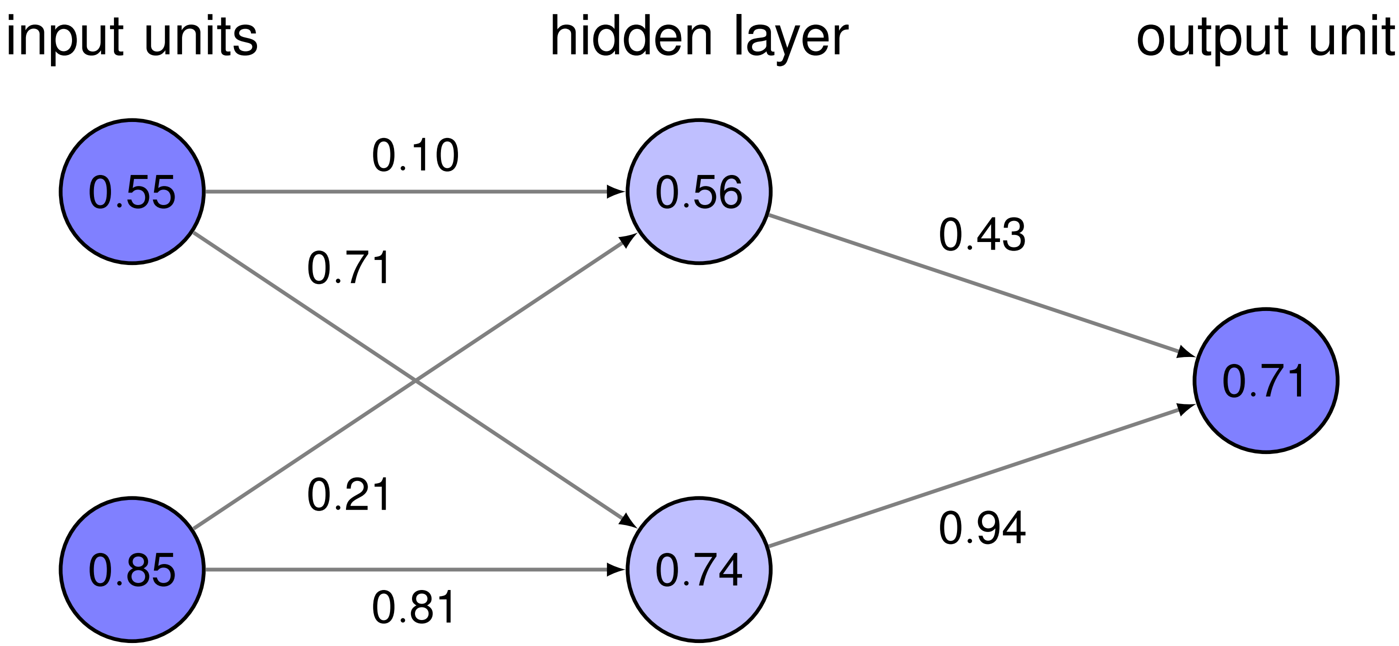
Neural networks

```
# Calculate output error H1
o_error_h1 = w_o1 * o_error_o1 * (act_h1 * (1-act_h1))
o_error_h1
# [1] 0.005920827
# Calculate output error H2
o_error_h2 = w_o2 * o_error_o1 * (act_h2 * (1-act_h2))
o_error_h2
# [1] 0.01028768
```

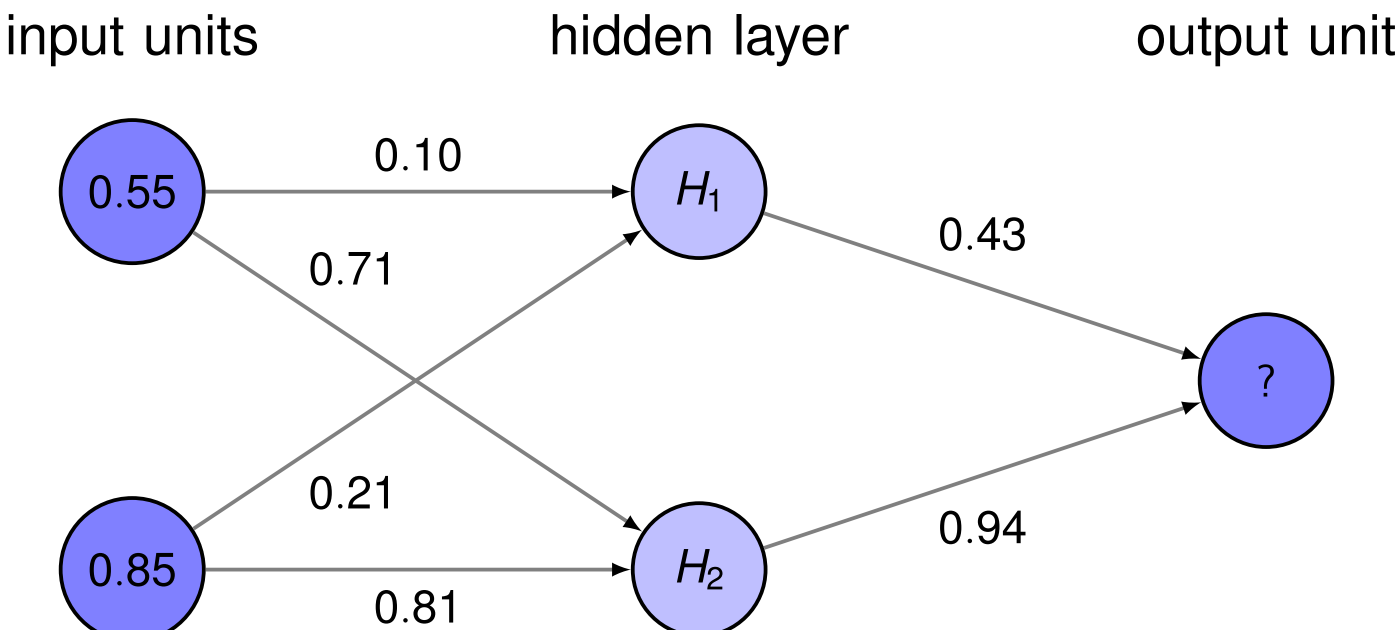
Neural networks

```
# Calculate new w_h1
w_h1_new = w_h1 + (act_i1 * o_error_h1)
w_h1_new
# [1] 0.1032565
# Calculate new w_h2
w_h2_new = w_h2 + (act_i1 * o_error_h2)
w_h2_new
# [1] 0.7056582
# Calculate new w_h3
w_h3_new = w_h3 + (act_i2 * o_error_h1)
w_h3_new
# [1] 0.2050327
# Calculate new w_h4
w_h4_new = w_h4 + (act_i2 * o_error_h2)
w_h4_new
# [1] 0.8087445
```

Neural networks



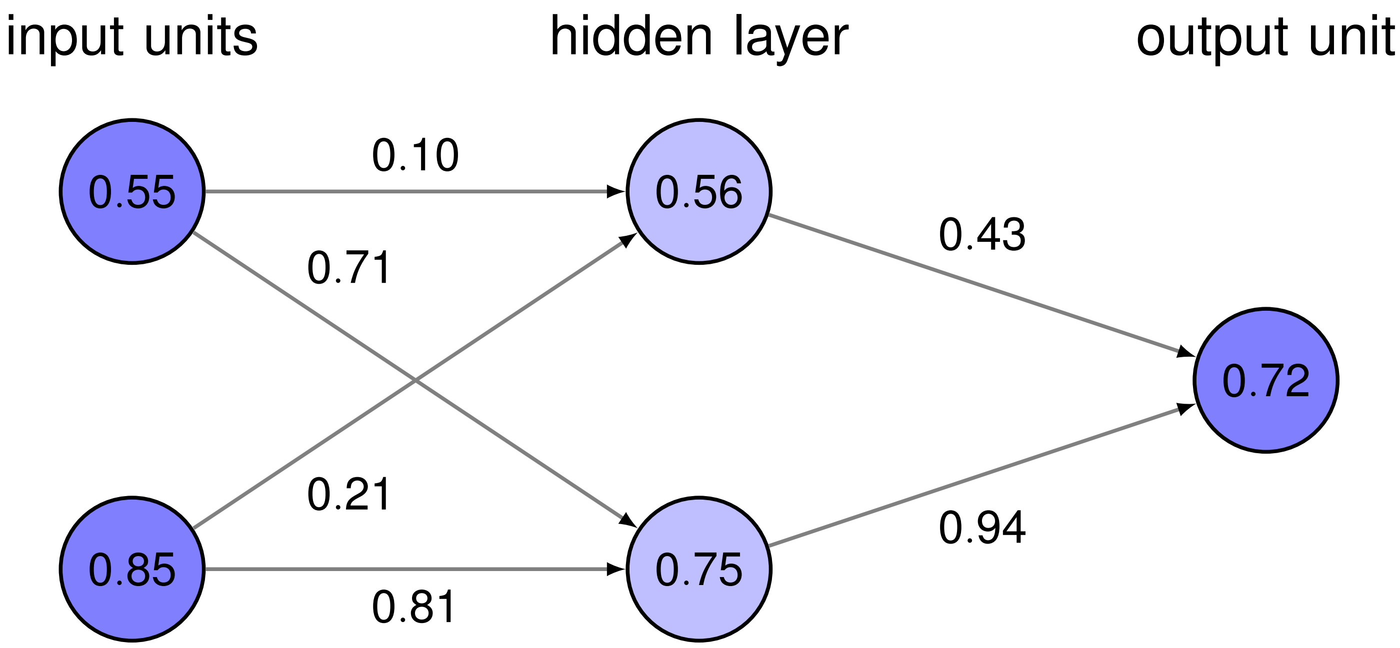
Neural networks



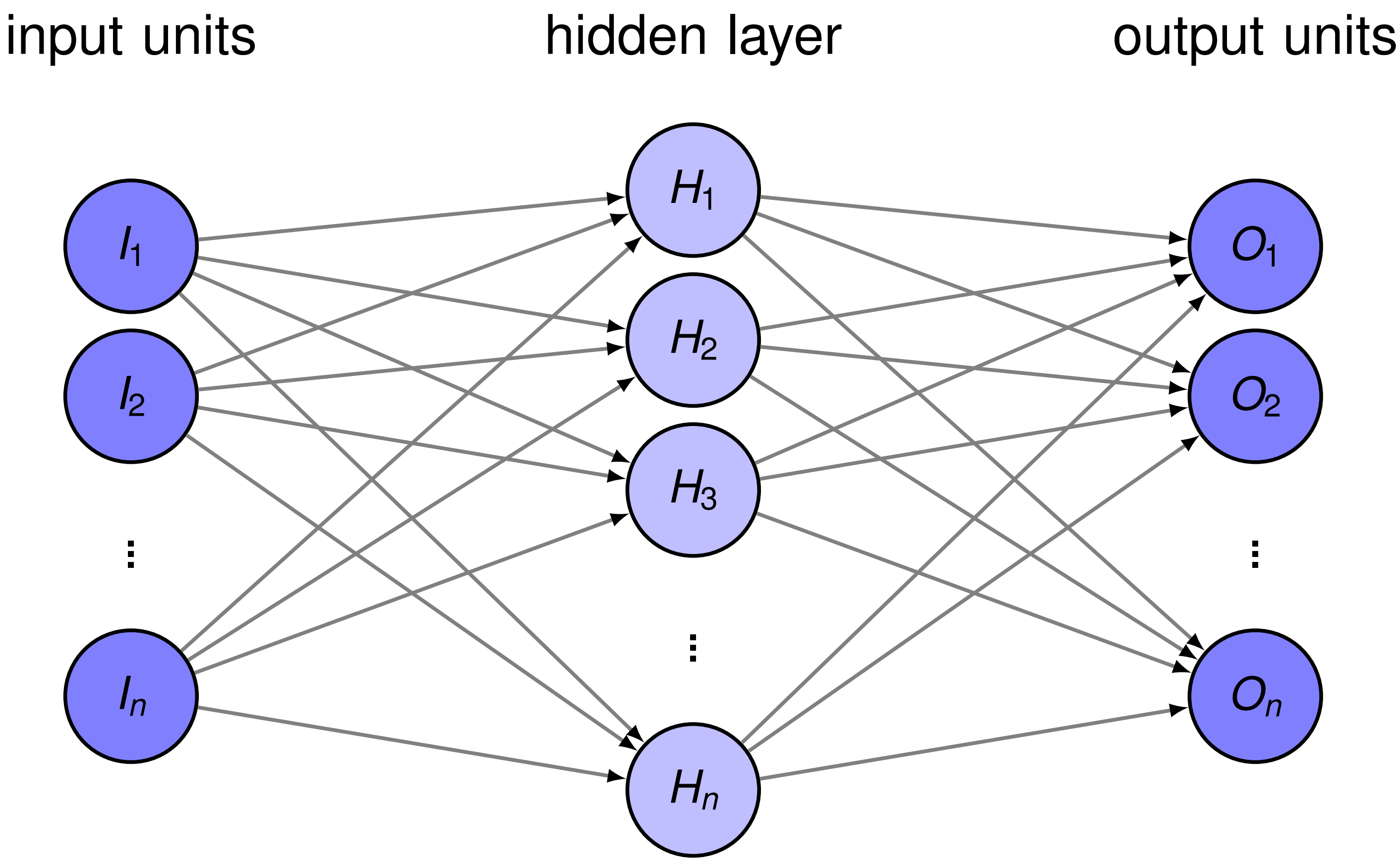
Neural networks

```
# Old output:
act_o1
# [1] 0.709238
#
# New output
act_h1_new = sigmoid.fnc(act_i1 * w_h1_new + act_i2 * w_h3_new)
act_h2_new = sigmoid.fnc(act_i1 * w_h2_new + act_i2 * w_h4_new)
act_o1_new = sigmoid.fnc(act_h1_new * w_o1_new + act_h2_new * w_o2_new)
act_o1_new
# [1] 0.7202949
```

Neural networks



Neural networks



Neural networks

```
# Library
library(h2o)
#
# Set up H2O cluster
localH2O = h2o.init(ip="localhost",port=54321,startH2O=TRUE,
                    max_mem_size='50g',nthreads=4)
# Import data into H2O cluster
train_h2o <- as.h2o(train,key='train')
test_h2o <- as.h2o(test,key='test')
```

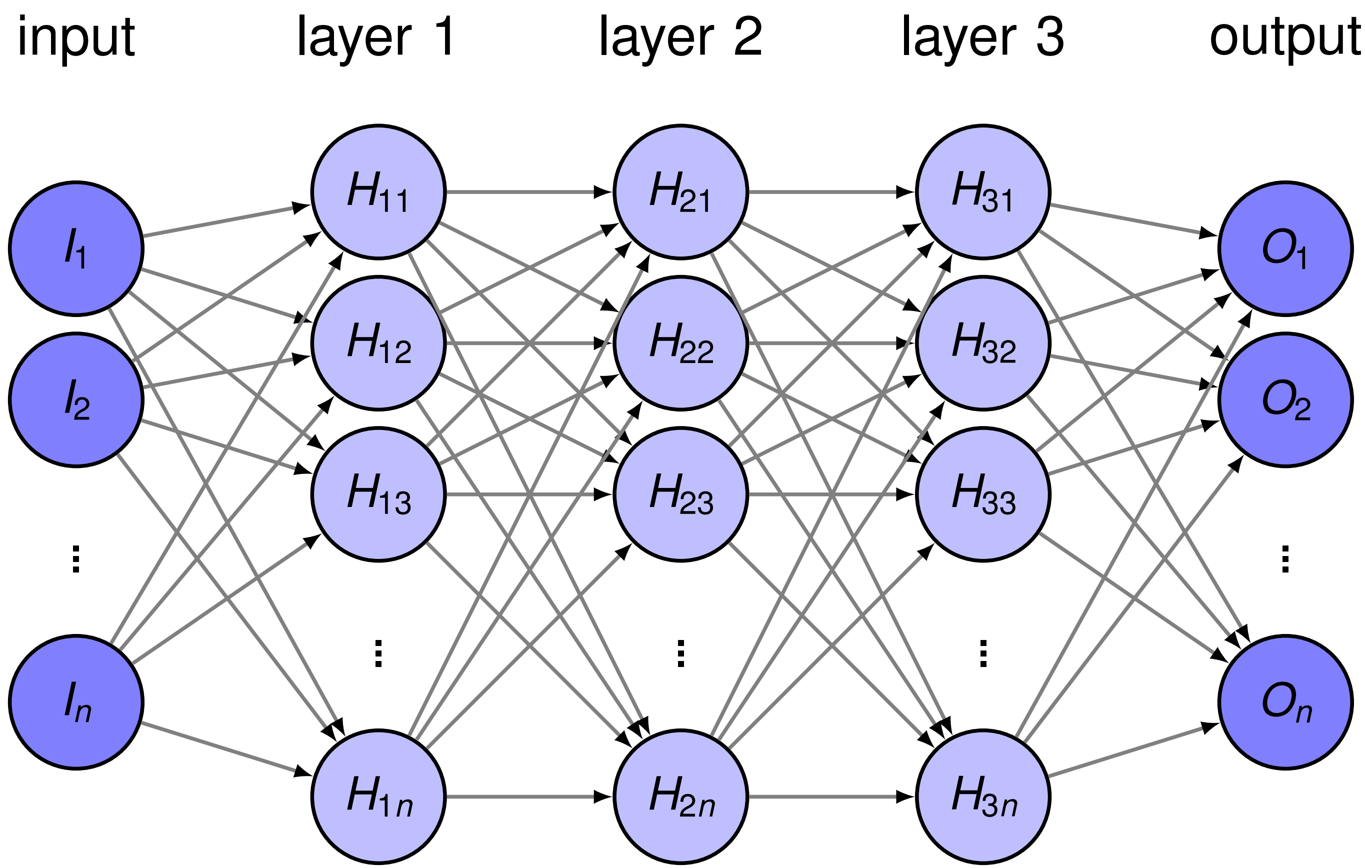
Neural networks

```
# Run model
nnet = h2o.deeplearning(x=2:785,y=1,train_h2o,
                        validation=test_h2o,hidden=10,
                        activation="RectifierWithDropout",
                        epochs=10,l1=1e-5,
                        input_dropout_ratio = 0.2)
#
# Predict
class_nnet = as.data.frame(h2o.predict(nnet,test_h2o))[,1]
#
# Performance:
table(class_nnet==test$label)
# FALSE  TRUE
#  1482  8518
```

Neural networks

```r
# Load predictions from bigger model
# hidden = 2048, epochs = 250
load("predictions/class_nnet.rda")
#
# Performance:
table(class_nnet==test$label)
#
# FALSE  TRUE
#   190  9810
```


Deep learning



Deep learning

```r
# Run model
deeplearning = h2o.deeplearning(x=2:785,y=1,data=train_h2o,
                                validation=test_h2o,
                                hidden=c(1024,1024,2048),
                                activation="RectifierWithDropout",
                                epochs=8000,l1=1e-5,
                                input_dropout_ratio=0.2,
                                train_samples_per_iteration=-1,
                                classification_stop=-1)
#
# Predict
class_deeplearning = as.data.frame(h2o.predict(deeplearning,
                                               test_h2o))[,1]
```

Deep learning

```
# Load predictions
load("predictions/class_deeplearning.rda")
#
# Performance
table(class_deeplearning==test$label)
#
# FALSE  TRUE
#   125  9875
```

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Deep learning

```
# Confusion matrix
table(class_deeplearning,test$label)
#
# class_deeplearning    0    1    2    3    4    5    6    7    8    9
#                  0  979    0    0    2    0    0    3    1    0    6
#                  1    0 1110    0    1    4    1    0    1    5    0
#                  2    1    1  983    7    1    0    1    4    0    0
#                  3    0    0    1 1015    1    2    0    1    5    2
#                  4    1    1    1    0  952    0    1    0    1    2
#                  5    1    0    1    5    1  895    0    0    5    4
#                  6    0    0    0    1    2    4  980    0    1    0
#                  7    0    1    7    0    3    1    0 1040    0    7
#                  8    2    1    2    5    0    0    0    0  949    4
#                  9    0    1    0    0    5    1    0    1    1  972
```

Deep learning

- Advantages:
  - Excellent performance
- Disadvantages:
  - Computationally expensive
  - Hard to tune
  - Black box?

Deep learning

- Taking a look inside the black box:

  1) Get the activation of a hidden layer unit given all training instances

  2) Calculate the correlation of these activations with each input unit

  3) Plot the result

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Deep learning

```
# Get deep features
features1 = h2o.deepfeatures(train_h2o,deeplearning,layer = 1)
features1 = as.data.frame(features1)
features2 = h2o.deepfeatures(train_h2o,deeplearning,layer = 2)
features2 = as.data.frame(features2)
features3 = h2o.deepfeatures(train_h2o,deeplearning,layer = 3)
features3 = as.data.frame(features3)
features = list(features1,features2,features3)
```

Deep learning

```
# Inspect deep features
dim(features[[1]])
# [1] 32000  1025
dim(features[[2]])
# [1] 32000  1025
dim(features[[3]])
# [1] 32000  2049
features[[1]][1:5,1:5]
#   label     DF.C1     DF.C2    DF.C3 DF.C4
# 1     9 0.4345101 0.0000000 1.039932     0
# 2     3 0.0000000 0.0000000 0.000000     0
# 3     9 0.8391911 0.1179328 0.000000     0
# 4     1 0.0000000 0.0000000 0.000000     0
# 5     8 0.0000000 0.0000000 0.000000     0
```

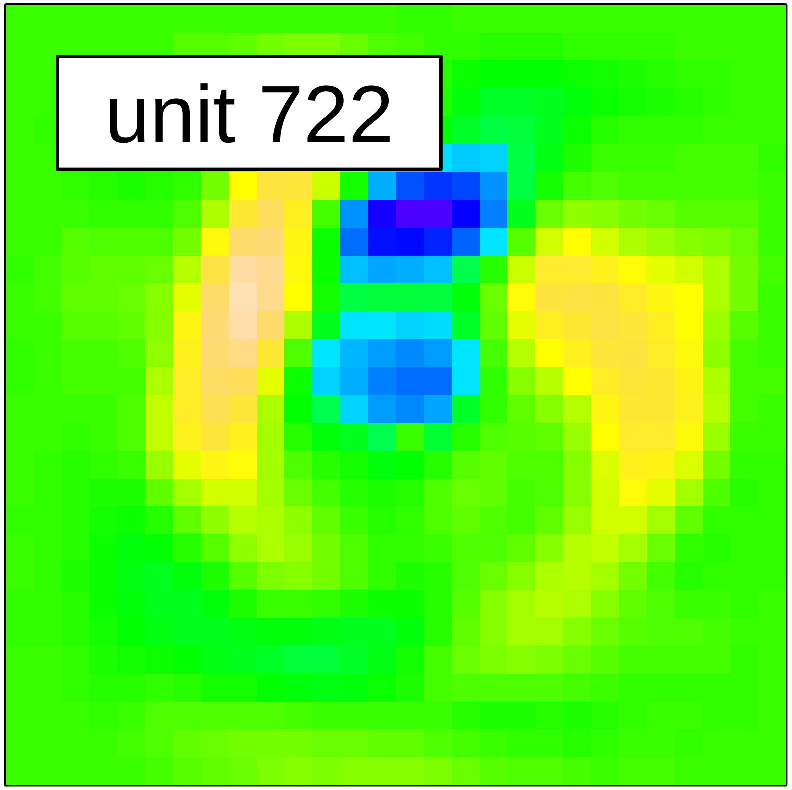
Deep learning

```
# Define plot function
feature.plot.fnc = function(feature=1,layer=1) {
  par(mar=c(1,1,1,1))

  cors = sapply(2:785,FUN=function(x) {
           cor(features[[layer]][,feature+1],train[,x])})
  cors[is.na(cors)] = 0

  pic = matrix(cors,ncol=28,byrow=TRUE)
  pic = t(apply(pic,2,rev))
  image(pic,col=topo.colors(100),xaxt="n",yaxt="n",useRaster=TRUE)

  width = 0.30 + nchar(feature)*0.07
  rect(0.05,0.8,0.05+width,0.95,col = "white", border="black",lwd=1)
  text(0.05 + width/2,0.874,paste("unit",feature),col="black",cex=1.5)
}
```

Deep learning

```
# Define plot function
feature.plot.fnc(feature=722,layer=1)
```

Deep learning



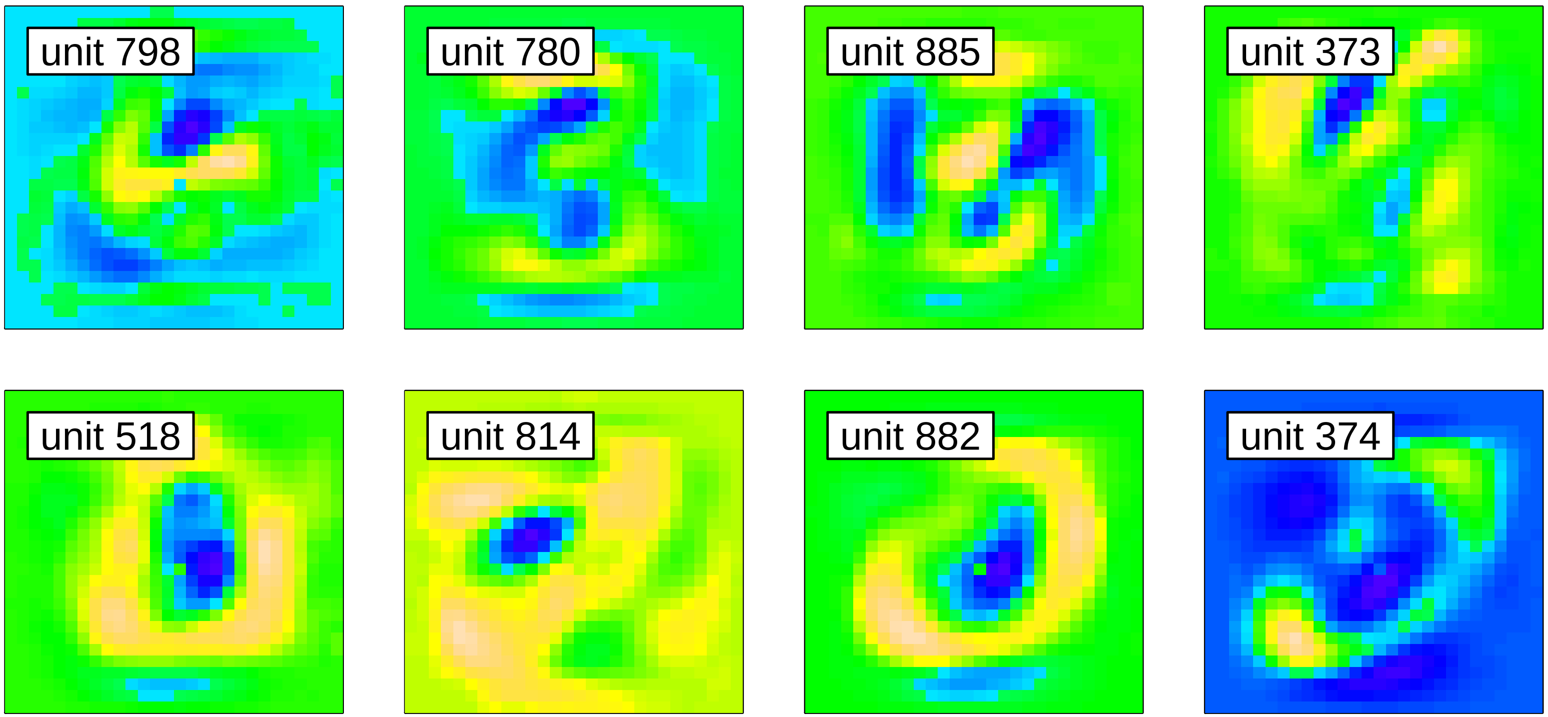
Deep learning

```
# Get average feature load for each digit
feature.load.fnc = function(feature=1,layer=1,
                            features=features) {
  tab = tapply(features[[layer]][,feature+1],
               features[[layer]][,1],mean)
  if(sum(tab) > 0) {tab = tab/sum(tab)} else {tab = tab}
  return(tab)
}
#
# Example
load = feature.load.fnc(722,1,features)
round(load,3)
#     0     1     2     3     4     5     6     7     8     9
# 0.169 0.025 0.076 0.011 0.238 0.092 0.135 0.156 0.086 0.012
```


Deep learning

```r
# Plot some features for layer 1
par(mfrow=c(2,4))
par(oma=c(1,1,1,1))
set.seed(3456)
for(feature in sample(1:1024,8,replace=F)) {
  feature.plot.fnc(feature,layer=1)
}
```

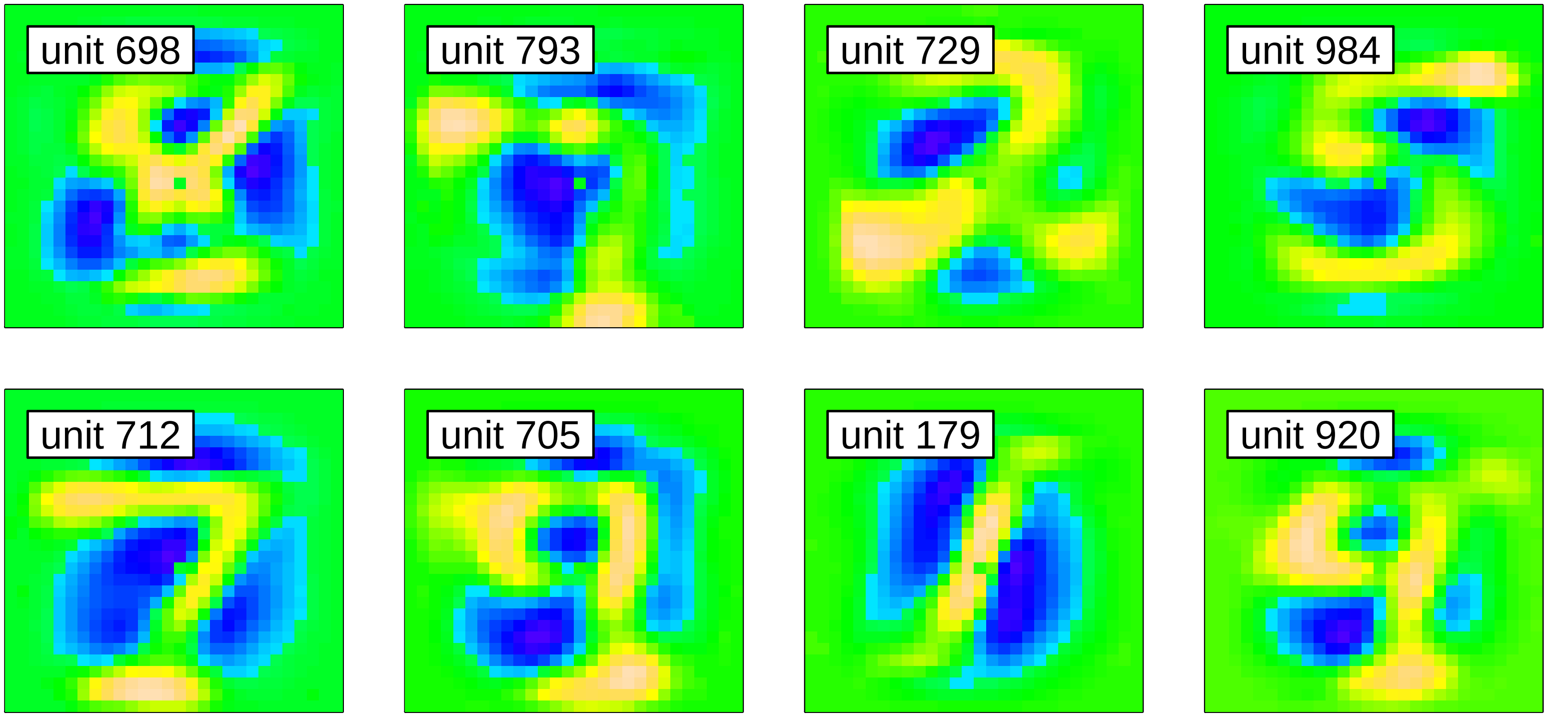
Deep learning



Deep learning

```r
# Plot some features for layer 2
par(mfrow=c(2,4))
par(oma=c(1,1,1,1))
set.seed(821)
for(feature in sample(1:1024,8,replace=F)) {
  feature.plot.fnc(feature,layer=2)
}
```

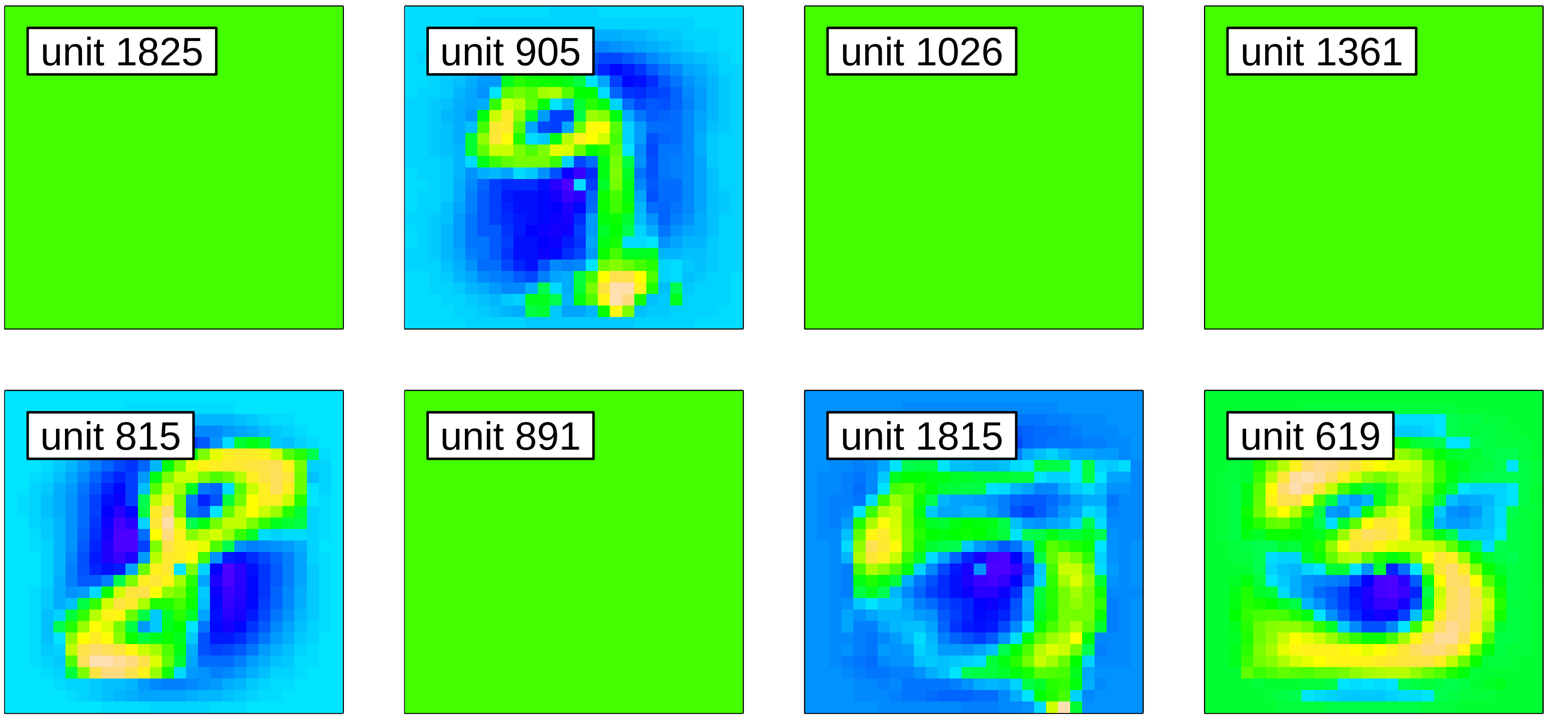
Deep learning



Deep learning

```
# Plot some features for layer 3
par(mfrow=c(2,4))
par(oma=c(1,1,1,1))
set.seed(727)
for(feature in sample(1:2408,8,replace=F)) {
  feature.plot.fnc(feature,layer=3)
}
```

Deep learning



Deep learning

```
# Get feature loads for each digit
round(feature.load.fnc(905,3,features),3)
#     0     1     2     3     4     5     6     7     8     9
# 0.000 0.000 0.000 0.000 0.000 0.011 0.000 0.007 0.000 0.981
round(feature.load.fnc(815,3,features),3)
#     0     1     2     3     4     5     6     7     8     9
# 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.996 0.004
round(feature.load.fnc(1815,3,features),3)
#     0     1     2     3     4     5     6     7     8     9
# 0.175 0.000 0.000 0.000 0.000 0.648 0.022 0.000 0.000 0.155
round(feature.load.fnc(619,3,features),3)
#     0     1     2     3     4     5     6     7     8     9
# 0.012 0.000 0.000 0.700 0.000 0.146 0.020 0.000 0.100 0.023
```

Deep learning

```r
# Define average maximum load function
layer.load.fnc = function(lay=layer,feats=features) {
  num.units = dim(features[[lay]])[2]-1
  dfr = sapply(1:num.units,feature.load.fnc,layer=lay,
               features=feats)

  maxes = apply(dfr,2,max)

  zero = length(maxes[maxes<=0])
  print(paste("Number of zero load nodes:", zero))

  maxes = maxes[maxes>0]
  measure = mean(maxes)
  return(measure)
}
```

Deep learning

```
# Get average maximum load for each layer
layer.load.fnc(1,features)
# [1] "Number of zero load nodes: 61"
# [1] 0.2912651
layer.load.fnc(2,features)
# [1] "Number of zero load nodes: 47"
# [1] 0.576322
layer.load.fnc(3,features)
# [1] "Number of zero load nodes: 1073"
# [1] 0.8023337
```

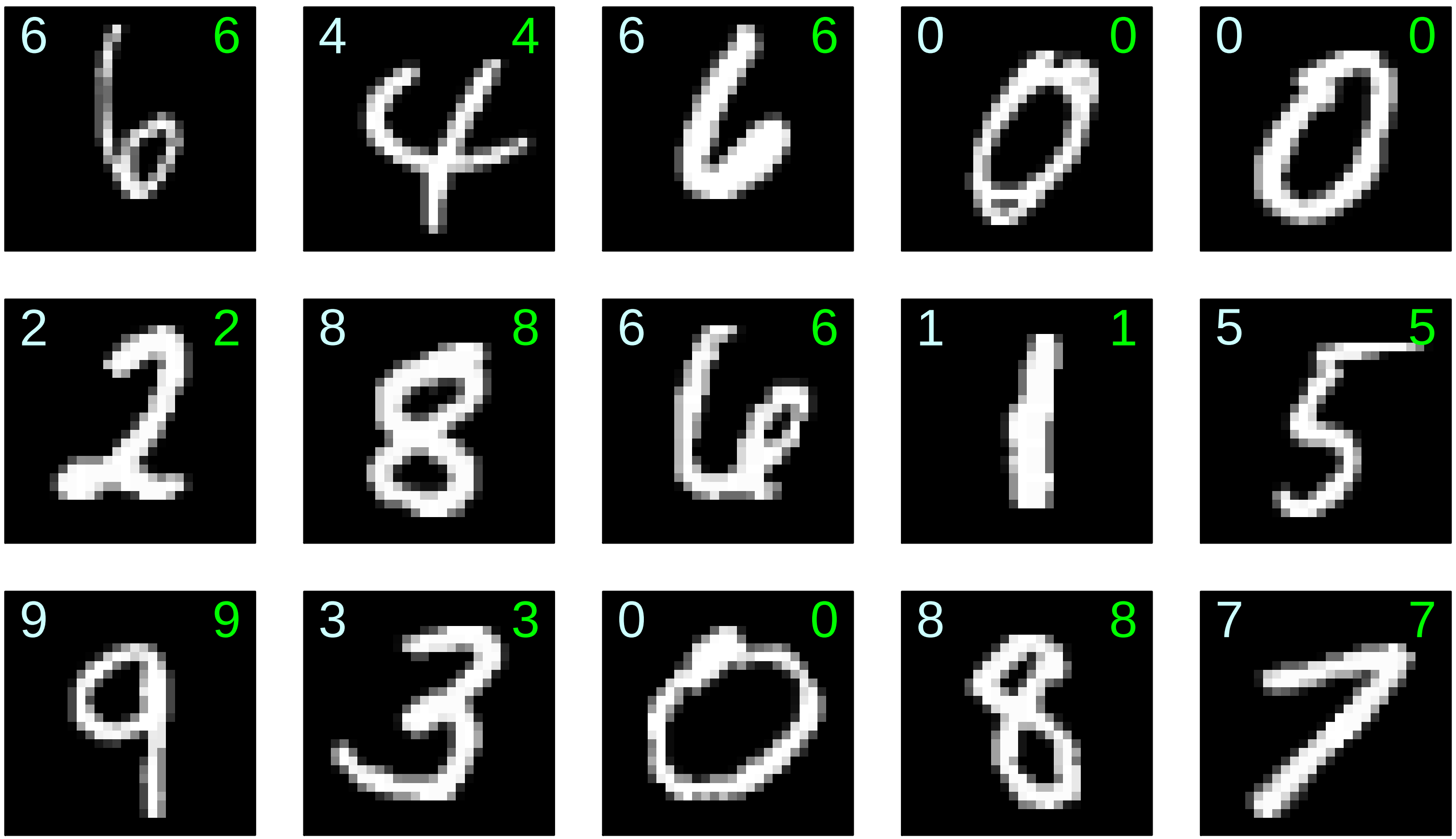
Deep learning

```
# Update drawing function
draw_labeled.fnc = function(num) {

  par(mar=c(1,1,1,1))
  pic = as.numeric(test[num,2:785])
  pic = matrix(pic,ncol=28,byrow=TRUE)
  pic = t(apply(pic,2,rev))
  image(pic,col=grey(level=seq(0,1,by=0.01)),xaxt="n",yaxt="n",
        useRaster=TRUE)
  text(0.1,0.9,test$label[num],col="#CCFFFF",cex=2.5)
  if(class_deeplearning[num]==test$label[num]) {col.lab="green"}
     else {col.lab="red"}
  text(0.9,0.9,class_deeplearning[num],col=col.lab,cex=2.5)
}
```

Deep learning

```
# Pick random set of images
set.seed(416)
draw = sample(1:nrow(test),15,replace=F)
#
# Plot using function
par(mfrow=c(3,5))
par(oma=c(1,1,1,1))
invisible(sapply(draw,draw_labeled.fnc))
```

Deep learning



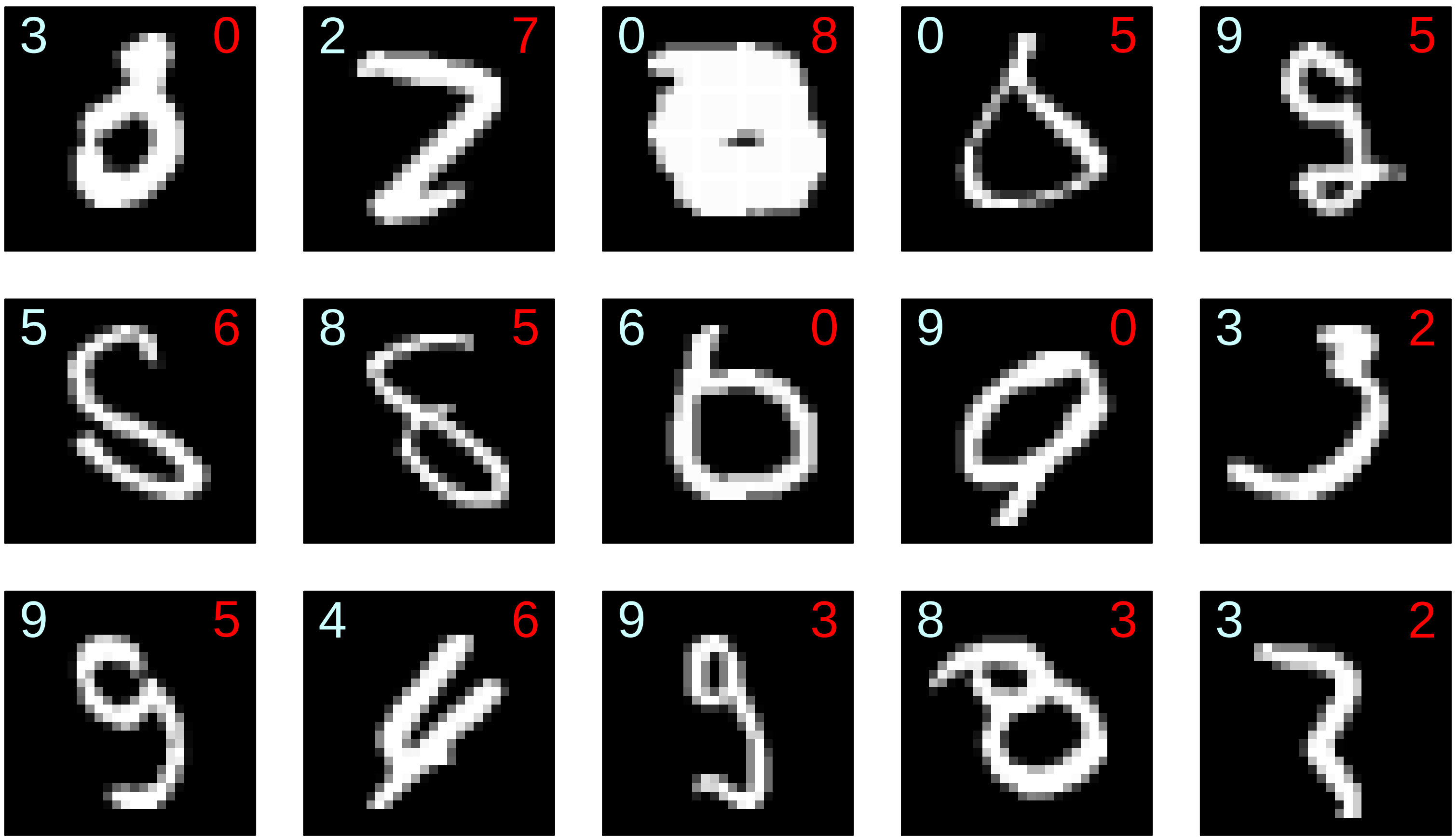
Deep learning

When are predictions incorrect?

Deep learning

```r
# Find images that are labelled wrong
wrong = which(class_deeplearning!=test$label)
# Pick random set of images
set.seed(2799)
draw = sample(wrong,15,replace=F)
#
# Plot using function
par(mfrow=c(3,5))
par(oma=c(1,1,1,1))
invisible(sapply(draw,draw_labeled.fnc))
```

Deep learning



Conclusions

- Many good techniques for statistical classification exist
- Pick an analysis technique depending on:
  - the data
  - the objective of the analysis
  - computational resources
  - …
- Do not be afraid to try a few different techniques

Conclusions

- Want better performance?
  - preprocessing
  - parameter tuning
  - data augmentation
  - ensembles
- Want to improve efficiency?
  - dimension reduction
  - graphics processing unit (GPU)

Conclusions

"The answer to 'Should I ever use learning algorithm (a) over learning algorithm (b)' will pretty much always be yes."

Jack Rae, Google DeepMind Research Engineer
Quora

Conclusions

There are no definite answers

Conclusions

"If a crab and a half weigh a pound and a half, but the half crab weighs half as much again as the whole crab... what do half the whole crab and the whole of the half crab weigh?"

Conclusions

```
# A crab and a half weigh a pound and a half
whole_crab + half_crab = 1.5
whole_crab = 1.5 - half_crab
# The half crab weighs half as much again as the whole crab
half_crab = 1.5*whole_crab
# Substitute
half_crab = 1.5*(1.5 - half_crab)
half_crab = 2.25 - 1.50*half_crab
2.50 * half_crab = 2.25
half_crab = 0.9
# Substitute to get weight of whole crab
whole_crab + half_crab = 1.5
whole_crab + 0.9 = 1.5
whole_crab = 0.6
```

Conclusions

"... what do half the whole crab and the whole of the half crab weigh?"

Conclusions

```
# Repeat values we calculated
whole_crab = 0.6
half_crab = 0.9
# Calculate half the whole crab plus the whole of the half crab
0.5*whole_crab + 2*half_crab
0.5 * 0.6 + 2 * 0.9
2.1
```

Conclusions

Thank you!