Introduction to machine learning and neural networks

Toby Dylan Hocking
toby.hocking@nau.edu
toby.hocking@r-project.org

February 24, 2021

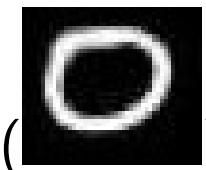
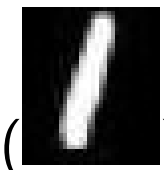
Introduction and applications

Demonstration of overfitting in regression / first steps with R

Classifying images of digits

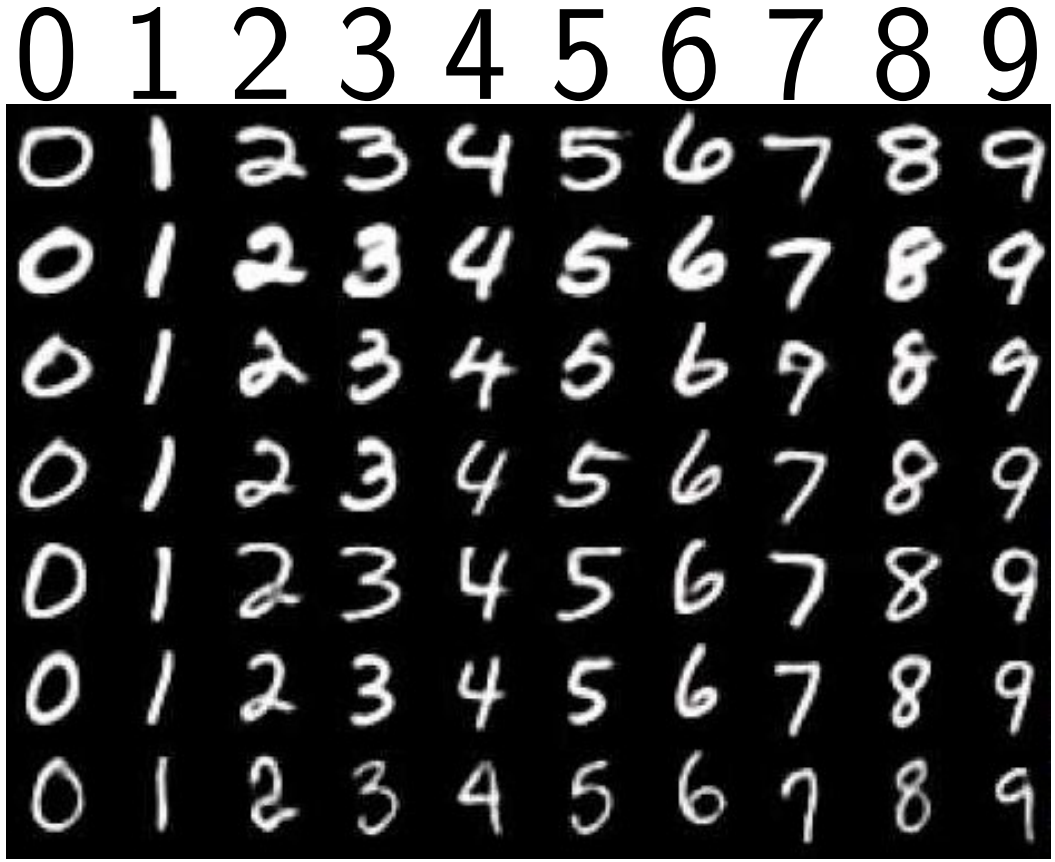
Machine learning intro: image classification example

ML is all about learning predictive functions $f(x) \approx y$, where

- Inputs/features x can be easily computed using traditional algorithms, e.g. matrix of pixel intensities in an image.
- Outputs/labels y are what we want to predict, easy to get by asking a human, but hard to compute using traditional algorithms, e.g. image class.
- Input x = image of digit, output $y \in \{0, 1, \ldots, 9\}$,
  – this is a classification problem with 10 classes.
  $f($  $) = 0$, $f($  $) = 1$
- Traditional/unsupervised algorithm: I give you a pixel intensity matrix $x \in \mathbb{R}^{16 \times 16}$, you code a function f that returns one of the 10 possible digits. Q: how to do that?

Supervised machine learning algorithms

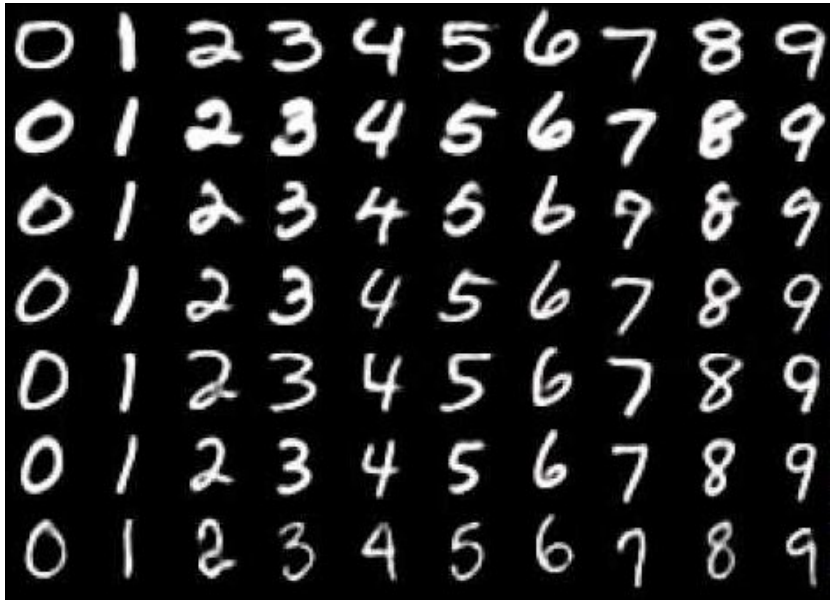
I give you a training data set with paired inputs/outputs, e.g.

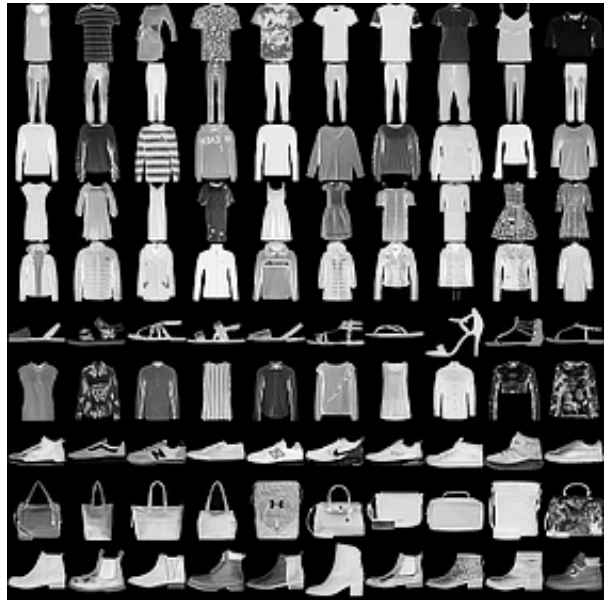


Your job is to code an algorithm that learns the function f from the training data. (you don't code f)

Source: github.com/cazala/mnist

Advantages of supervised machine learning



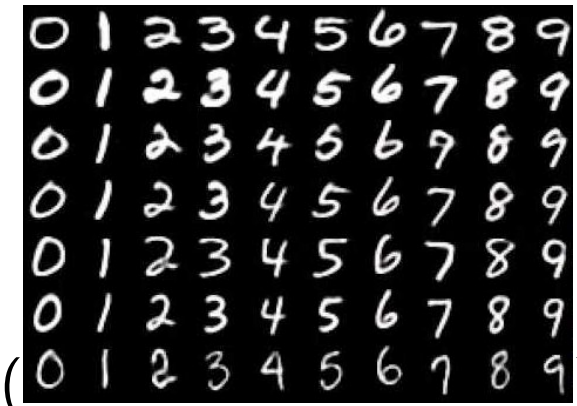
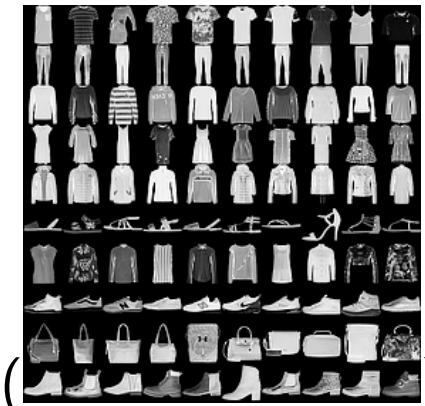


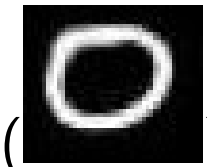
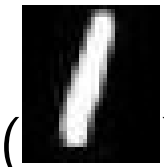
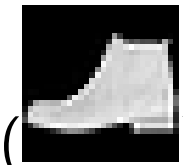
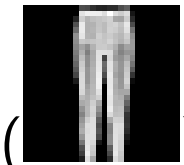
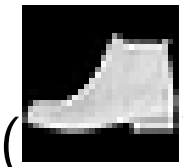
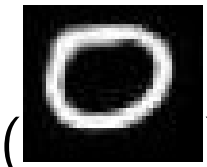
- Input $x \in \mathbb{R}^{16\times16}$, output $y \in \{0, 1, \ldots, 9\}$ types the same!
- Can use same learning algorithm regardless of pattern.
- Pattern encoded in the labels (not the algorithm).
- Useful if there are many un-labeled data, but few labeled data (or getting labels is long/costly).
- State-of-the-art accuracy (if there is enough training data).

Sources: github.com/cazala/mnist, github.com/zalandoresearch/fashion-mnist

Learning two different functions

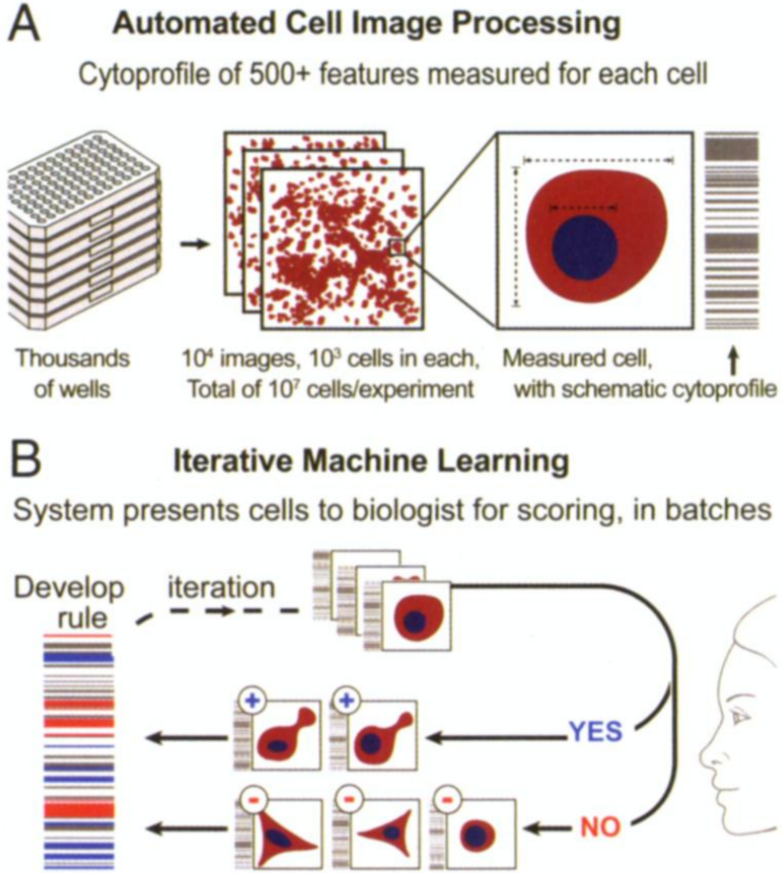
Say LEARN is a learning algorithm you have coded.

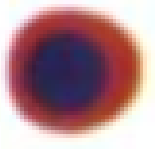
LEARN() $\rightarrow f$, LEARN() $\rightarrow g$

- Then we would expect $f($  $) = 0$, $f($  $) = 1$
- $g($  $) =$ shoe$/0$, $g($  $) =$ pants$/1$
- Q: what happens if you do $f($  $)$, or $g($  $)$?

Machine learning for cell image classification (CellProfiler)

Jones *et al.* PNAS 2009. Scoring diverse cellular morphologies in image-based screens with iterative feedback and machine learning.



- Input $x =$ image of cell,
- Output $y \in \{\text{yes}, \text{no}\}$ (binary classification),
- $f($  $) =$ yes,
- $f($  $) =$ no.

Machine learning for image segmentation (LabelMe)

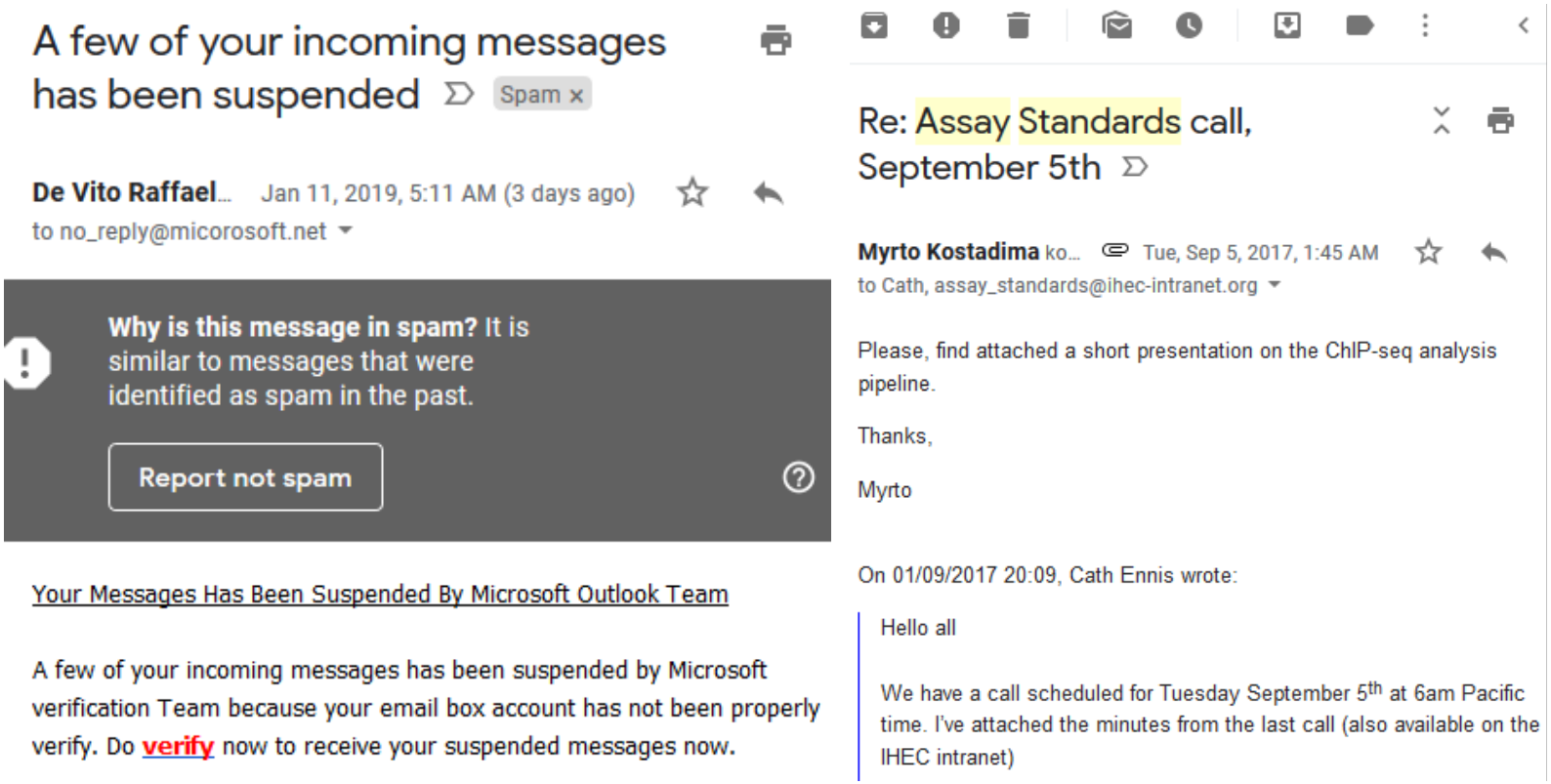
Figure 1. A screenshot of the labeling tool in use. The user is shown an image along with possibly one or more existing annotations, which are drawn on the image. The user has the option of annotating a new object by clicking along the boundary of the desired object and

Machine learning for image segmentation

Russell *et al.* 2007.

Q: What are the types/dimensions of x, y, f in this example?

Machine learning for spam filtering (Gmail)



Want: $f(\text{email message}) \in \{0, 1\}$ – binary classification, spam=1 or not=0.

Machine learning for translation (google books)

LE COMTE

DE

MONTE-CRISTO

PAR

ALEXANDRE DUMAS

ABRIDGED AND ANNOTATED

BY

EDGAR EWING BRANDON, A.M.

Professor of French in Miami University

NEW YORK

HENRY HOLT AND COMPANY

1900

THE

COUNT OF MONTE-CRISTO.

BY

ALEXANDRE DUMAS.

WITH

Twenty Illustrations,

DRAWN ON WOOD BY M. VALENTIN,

AND EXECUTED BY THE MOST EMINENT ENGLISH ENGRAVERS,
UNDER THE SUPERINTENDENCE OF MR. CHARLES HEATH.

IN TWO VOLUMES.

VOL. II.

LONDON:

CHAPMAN AND HALL, 186 STRAND.

1846.

Machine learning for translation (google translate)

— Fernand! lui cria-t-il, de mes noms, je n'aurais besoin de t'en dire qu'un seul pour te foudroyer; mais ce nom, tu le devines, n'est-ce pas ? ou plutôt tu te le rappelles ? car, malgré tous mes chagrins, toutes mes tortures, je te montre aujourd'hui un visage que le bonheur de la vengeance rajeunit, un visage que tu dois avoir vu bien souvent dans tes rêves depuis ton mariage . . . avec Mercédès, ma fiancée!

Le général, la tête renversée en arrière, les mains étendues, le regard fixe, dévora en silence ce terrible spectacle; puis, allant chercher la muraille comme point d'appui, il s'y glissa lentement jusqu'à la porte par laquelle il sortit à reculons, en laissant échapper ce seul cri lugubre, lamentable, déchirant:

— Edmond Dantès!

Want: $f($ $)=$

" Fernand," cried he, " of my hundred names I need only tell you one to overwhelm you ! But you guess it now; do you not ? — or, rather, you remember it ? For, notwithstanding all my sorrows and my tortures, I shew you to-day a face which the happiness of revenge makes young again — a face you must often have seen in your dreams since your marriage with Mercédès, my betrothed ! "

The general, with his head thrown back, hands extended, gaze fixed, looked silently at this dreadful apparition ; then seeking the wall to support him, he glided alon close to it until he reached the door, through which he went out backwards, uttering this single mournful, lamentable, distressing cry,—

" Edmond Dantès ! "

Machine learning for medical diagnosis

Original

f( ) =

Thanks to artificial intelligence and machine learning, diagnostic tools can be trained to read tissue samples and radiologic scans. Google researchers fed more than a quarter-million patients' retinal scans into algorithms that recognize patterns—and the technology "learned" to spot which patterns predict a patient has high blood pressure or is at increased risk for heart attack or stroke. In some comparisons, digital tools produced more accurate analyses than did human pathologists, dermatolo-

Introduction and applications

Demonstration of overfitting in regression / first steps with R

Classifying images of digits

Machine learning algorithms in practice

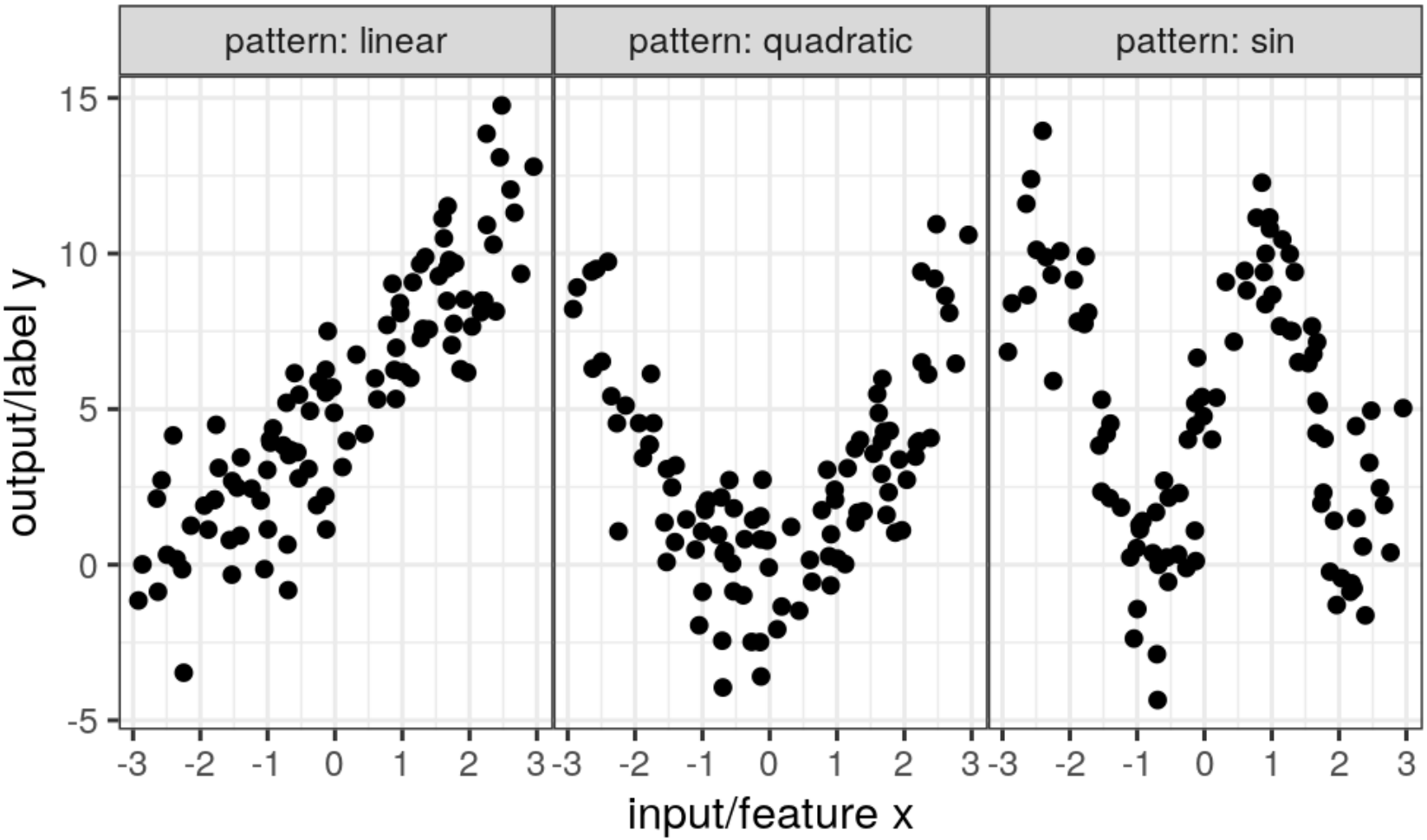
To learn a good prediction function for any of these problems (and any other problems you encounter in your work) you should

- First gather a data set that you will use to train/test the machine learning algorithms, typically a CSV file with rows for observations and columns for features.
- Use K-fold Cross-Validation to repeatedly divide the observations into train/test sets. Use the train set to learn machine learning model parameters, and use the test set to estimate prediction error.
- Try a variety of different learning algorithms (e.g. linear models, random forests, boosting, neural networks, support vector machines, ...), because you don't know which one will result in most accurate predictions for any given data set.
- Each learning algorithm has different hyper-parameters which control complexity/regularization/overfitting, and typically must be chosen by the user (you) by dividing the train data into subtrain/validation sets. Use the hyper-parameter which is best with respect to the held-out validation set.

Goal of this section: demonstrate overfitting

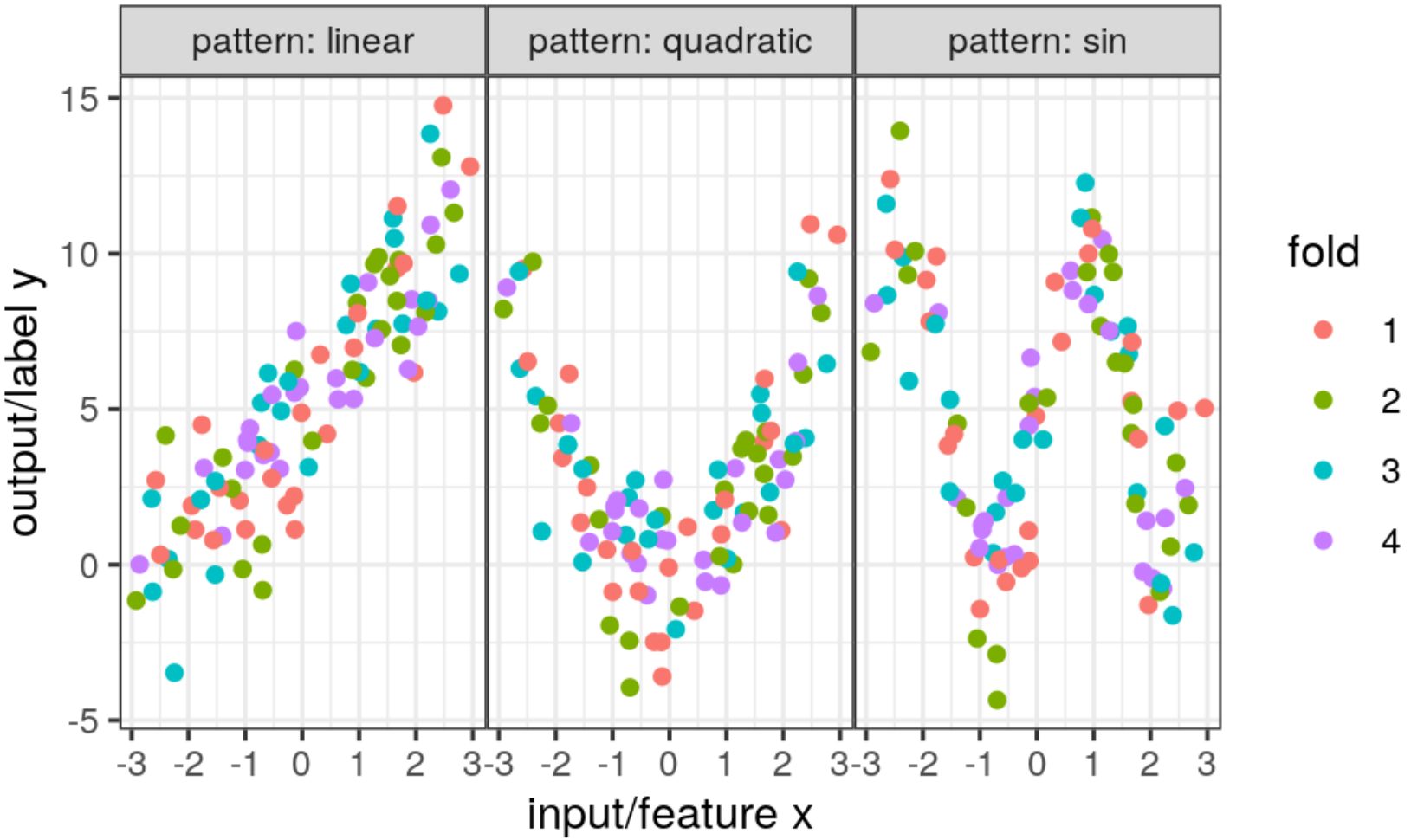
- The goal of supervised machine learning is to get accurate predictions on new/unseen/held-out test data.
- Any machine learning algorithm is prone to overfit, which means providing better predictions on the train/subtrain set than on a held-out validation/test set. (BAD)
- To learn a model which does NOT overfit (GOOD), you need to first divide your train set into subtrain/validation sets.
- Code for figures in this section: `https://github.com/tdhock/2020-yiqi-summer-school/blob/master/figure-overfitting.R`

Three different data sets/patterns



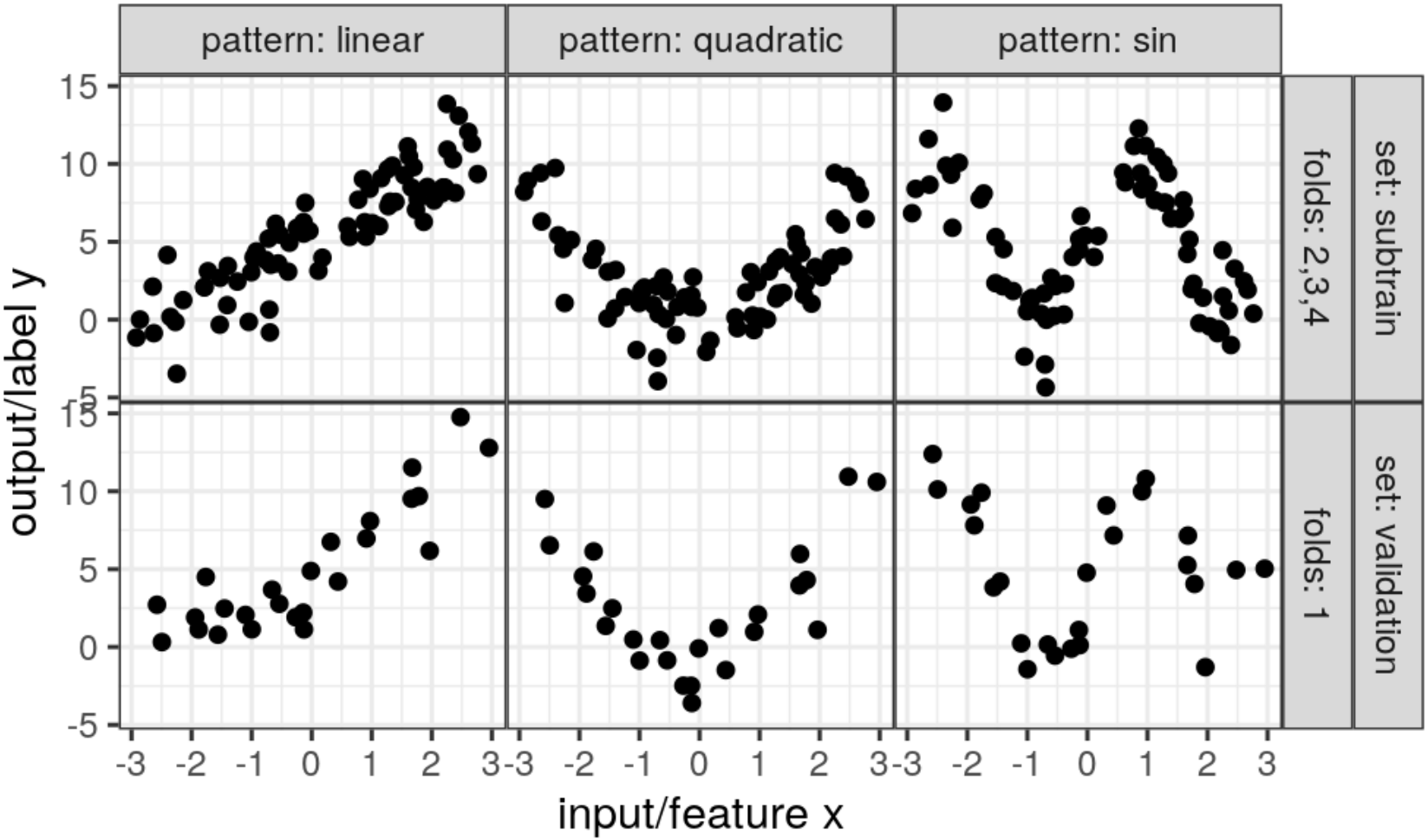
- We illustrate this using a single input/feature $x \in \mathbb{R}$.
- We use a regression problem with outputs $y \in \mathbb{R}$.
- Goal is to learn a function $f(x) \in \mathbb{R}$.

Illustration of 4-fold cross-validation



Randomly assign each observation a fold ID from 1 to 4.

Illustration of subtrain/validation split



- For validation fold 1, all observations with that fold ID are considered the validation set.
- All other observations are considered the subtrain set.

How to do this in practice? Use R with an IDE

- Machine learning model training/evaluation is easy to code in programming languages which support vectors/matrices.
- I recommend using R because it has so many useful packages (each one for a different machine learning model/algorithm).
- Download R at `https://cloud.r-project.org/`
- If you already use Emacs (classic text editor), I recommend using R inside Emacs Speaks Statistics, which is a great Interactive Development Environment (IDE) for R `http://ess.r-project.org/` Also see my screencasts about ESS, `https://www.youtube.com/playlist?list=PLwc48KSH3D1Onsed66FPLywMSIQmAhUYJ`
- Otherwise, I recommend using RStudio, which is another popular IDE for R `https://rstudio.com/products/rstudio/download`

CSV data tables for machine learning

- One row for each observation.
- One column for the output/label/y (in regression the label is a real number, in classification the label is a class/category).
- The other columns should be inputs/features/X that will be used to predict the corresponding output/label/y.

Example: `https://raw.githubusercontent.com/tdhock/2020-yiqi-summer-school/master/data_linear.csv`

```
                x             y
 1: -1.40694802  0.933196336
 2: -0.76725660  3.832773444
 3:  0.43712018  4.202983135
 4:  2.44924674 13.089055084
...
```

Download CSV data and get file names into R

Download a copy of the github repo to get these data sets
`https://github.com/tdhock/2020-yiqi-summer-school/archive/master.zip`
Set working directory to the downloaded repo:

```
> setwd("~/teaching/2020-yiqi-summer-school")
> getwd()
[1] "/home/tdhock/teaching/2020-yiqi-summer-school"
```

Get a character vector of CSV data file names:

```
> csv.vec <- Sys.glob("data_*.csv")
> csv.vec
[1] "data_linear.csv"    "data_quadratic.csv"
[3] "data_sin.csv"
```

Read each CSV data file into R

```
sim.data.list <- list()
for(csv in csv.vec){
  pattern <- gsub("data_|.csv", "", csv)
  sim.dt <-  data.table::fread(csv)
  set.seed(1) # for reproducibility.
  fold.vec <- sample(rep(1:4, l=nrow(sim.dt)))
  sim.data.list[[pattern]] <- data.table::data.table(
    pattern, sim.dt, fold=factor(fold.vec))
} # combine into a single data table:
sim.data <- do.call(rbind, sim.data.list)
```

- gsub removes prefix/suffix from CSV file name.
- data.table::fread gets CSV data from file into R.
- set.seed used to get consistent/reproducible results from the pseudo-random number generator (sample).
- data.table::fun means use fun from data.table package, install it via install.packages("data.table")

Result of reading CSV data into R

Result is:

```
> sim.data
     pattern           x           y fold
  1:  linear -1.4069480   0.9331963    4
  2:  linear -0.7672566   3.8327734    3
  3:  linear  0.4371202   4.2029831    1
  4:  linear  2.4492467  13.0890551    2
  5:  linear -1.7899084   2.0791987    3
 ---
296:     sin  1.7838530   4.0502991    1
297:     sin -0.2683533  -0.1097264    1
298:     sin -0.5394955  -0.5539398    1
299:     sin  1.8652215  -0.2262517    4
300:     sin  0.6295997   8.8124249    4
```

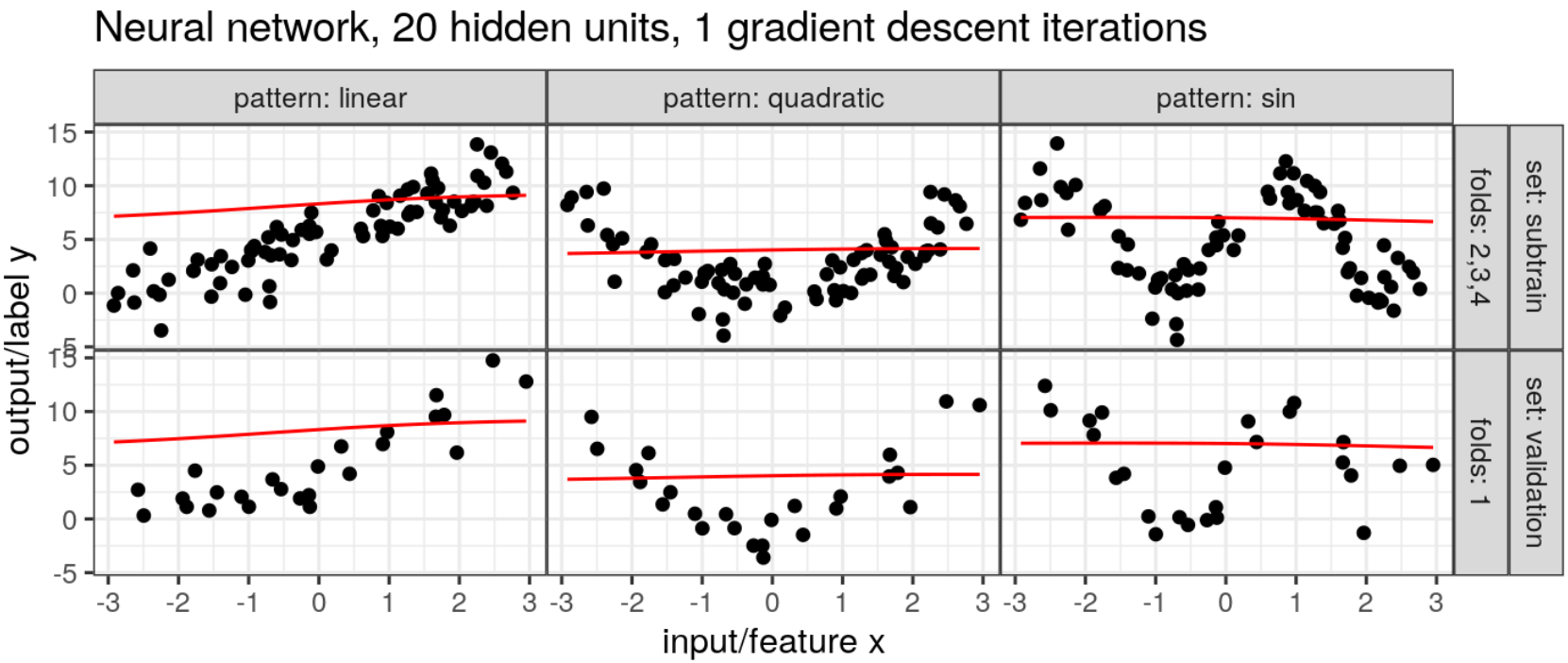
Assign each observation to subtrain/validation set

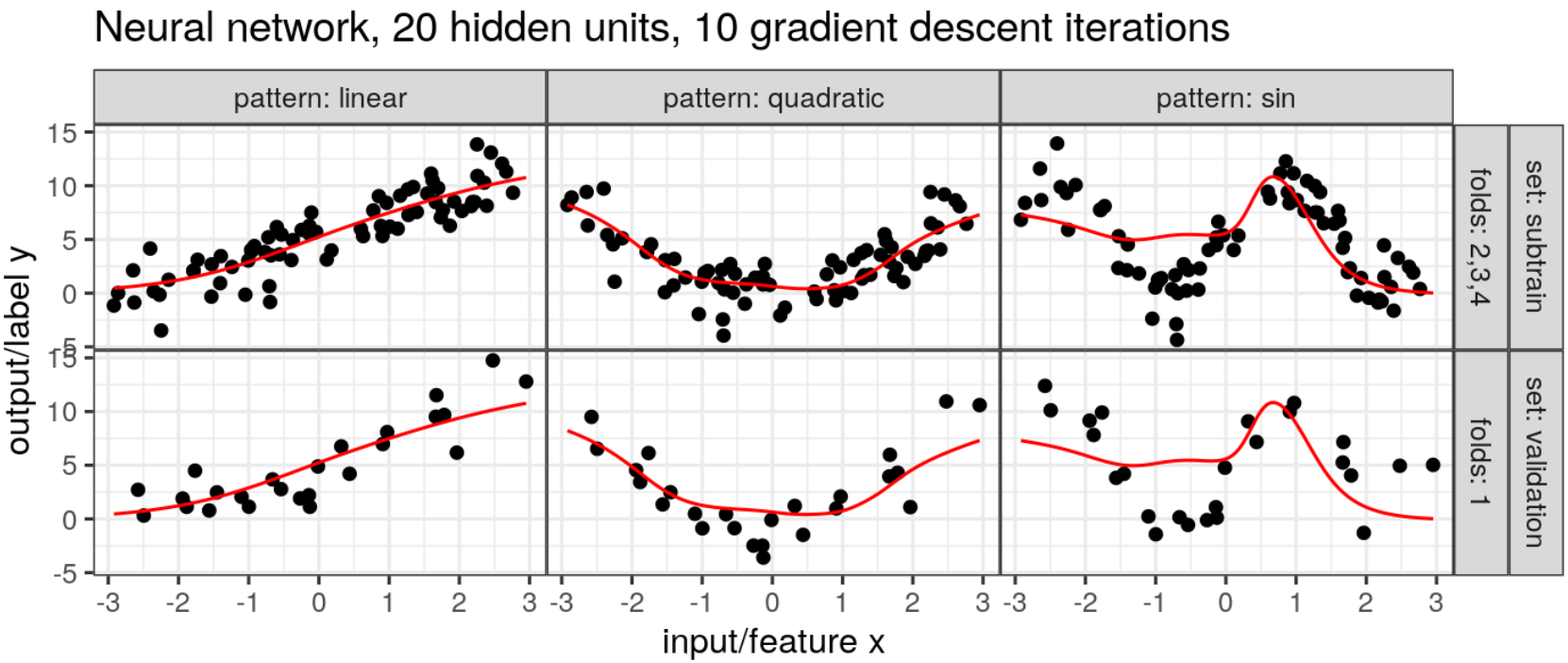
```
validation.fold <- 1
sim.data[, set := ifelse(
  fold==validation.fold, "validation", "subtrain")]
> sim.data
     pattern           x           y fold        set
  1:  linear -1.4069480   0.9331963    4   subtrain
  2:  linear -0.7672566   3.8327734    3   subtrain
  3:  linear  0.4371202   4.2029831    1 validation
  4:  linear  2.4492467  13.0890551    2   subtrain
  5:  linear -1.7899084   2.0791987    3   subtrain
 ---
296:     sin  1.7838530   4.0502991    1 validation
297:     sin -0.2683533  -0.1097264    1 validation
298:     sin -0.5394955  -0.5539398    1 validation
299:     sin  1.8652215  -0.2262517    4   subtrain
300:     sin  0.6295997   8.8124249    4   subtrain
```

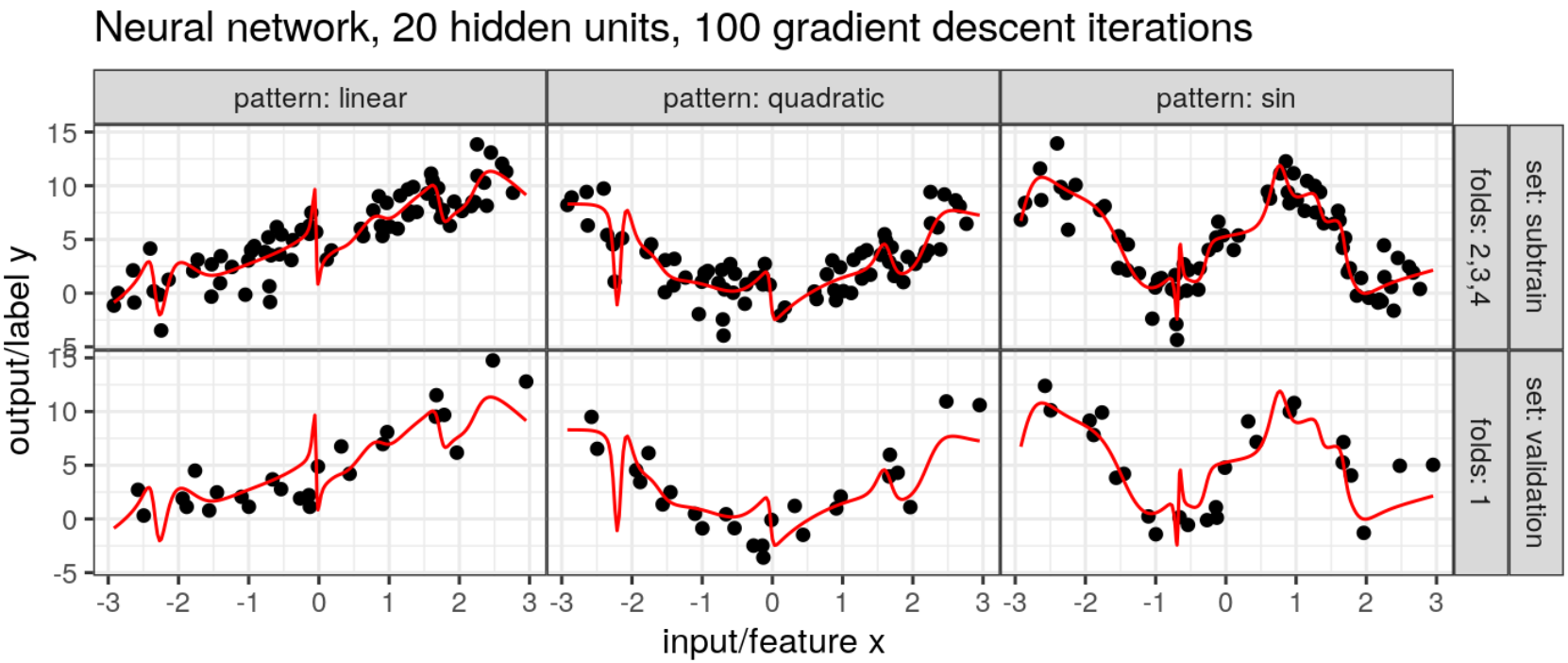
Neural network with one hidden layer, 20 hidden units

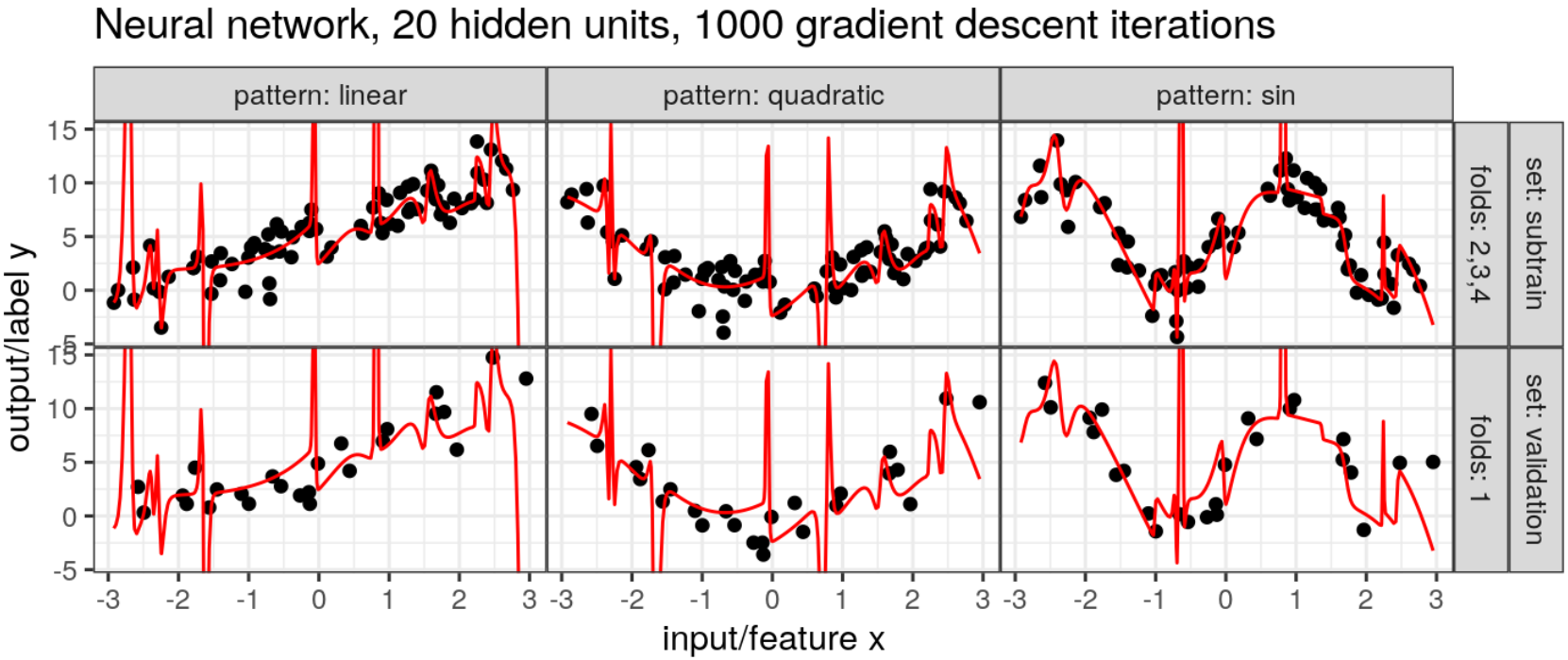
Use for loops to fit different neural network models for each data set and number of iterations.

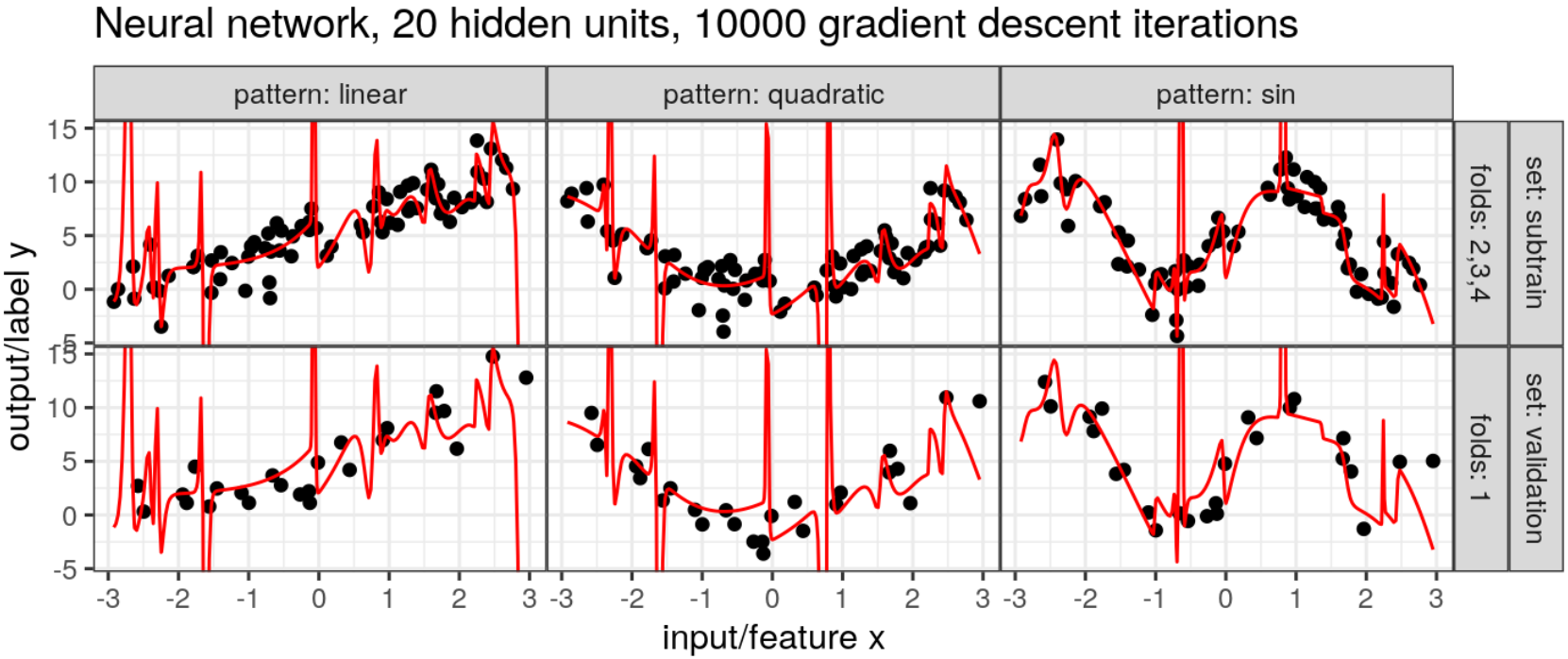
```
maxit.values <- 10^seq(0, 4)
pattern.values <- c("linear", "quadratic", "sin")
for(i in maxit.values)for(p in pattern.values){
  pattern.data <- sim.data[pattern==p]
  fit <- nnet::nnet(
    y ~ x,
    pattern.data[set=="subtrain"],
    size=20,      #hidden units
    linout=TRUE, #for regression
    maxit=i)      #max number of iterations
...
```

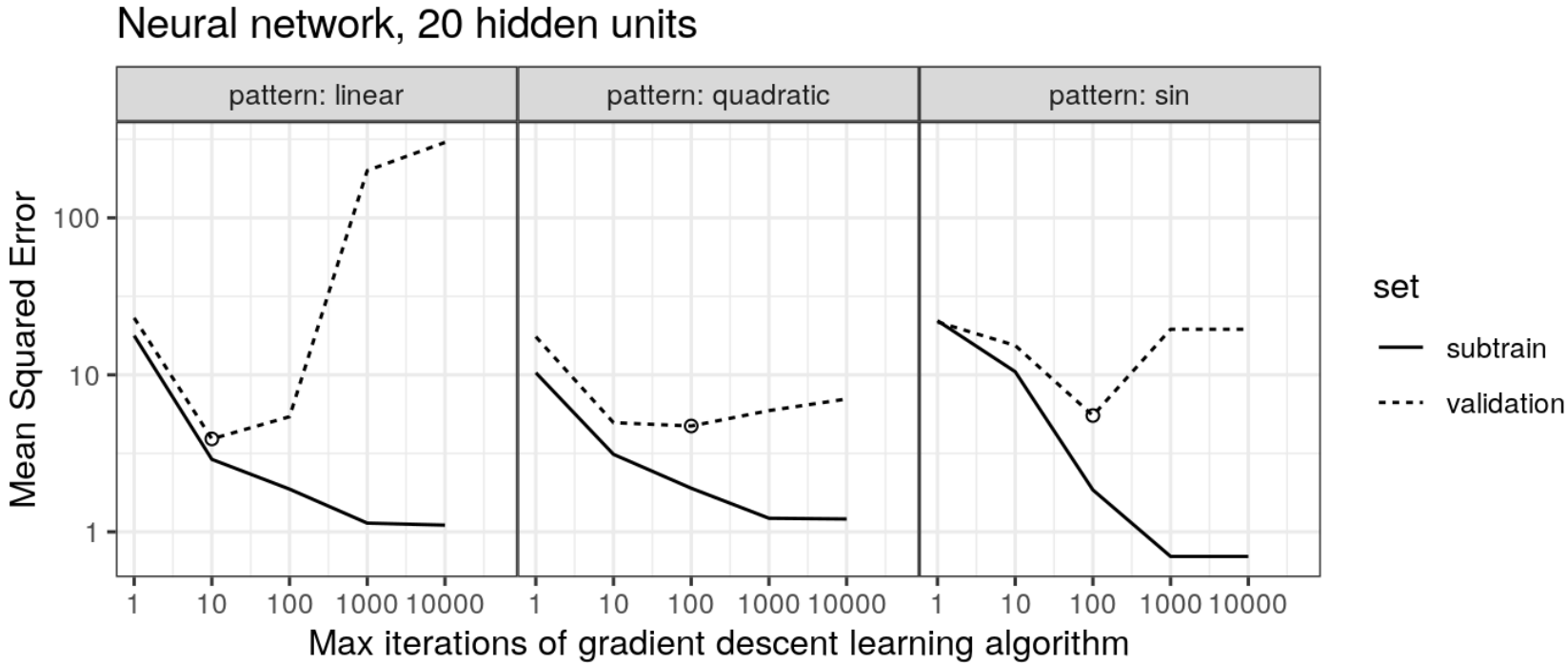

Neural network, 20 hidden units, 1 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
set: validation
folds: 2,3,4
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3


Neural network, 20 hidden units, 10 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
set: validation
folds: 2,3,4
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3


Neural network, 20 hidden units, 100 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
set: validation
folds: 2,3,4
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3


Neural network, 20 hidden units, 1000 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
folds: 2,3,4
set: validation
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3


Neural network, 20 hidden units, 10000 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
set: validation
folds: 2,3,4
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3


Neural network, 20 hidden units
pattern: linear
pattern: quadratic
pattern: sin
Mean Squared Error
100
10
1
1
10
100
1000
10000
Max iterations of gradient descent learning algorithm
set
subtrain
validation

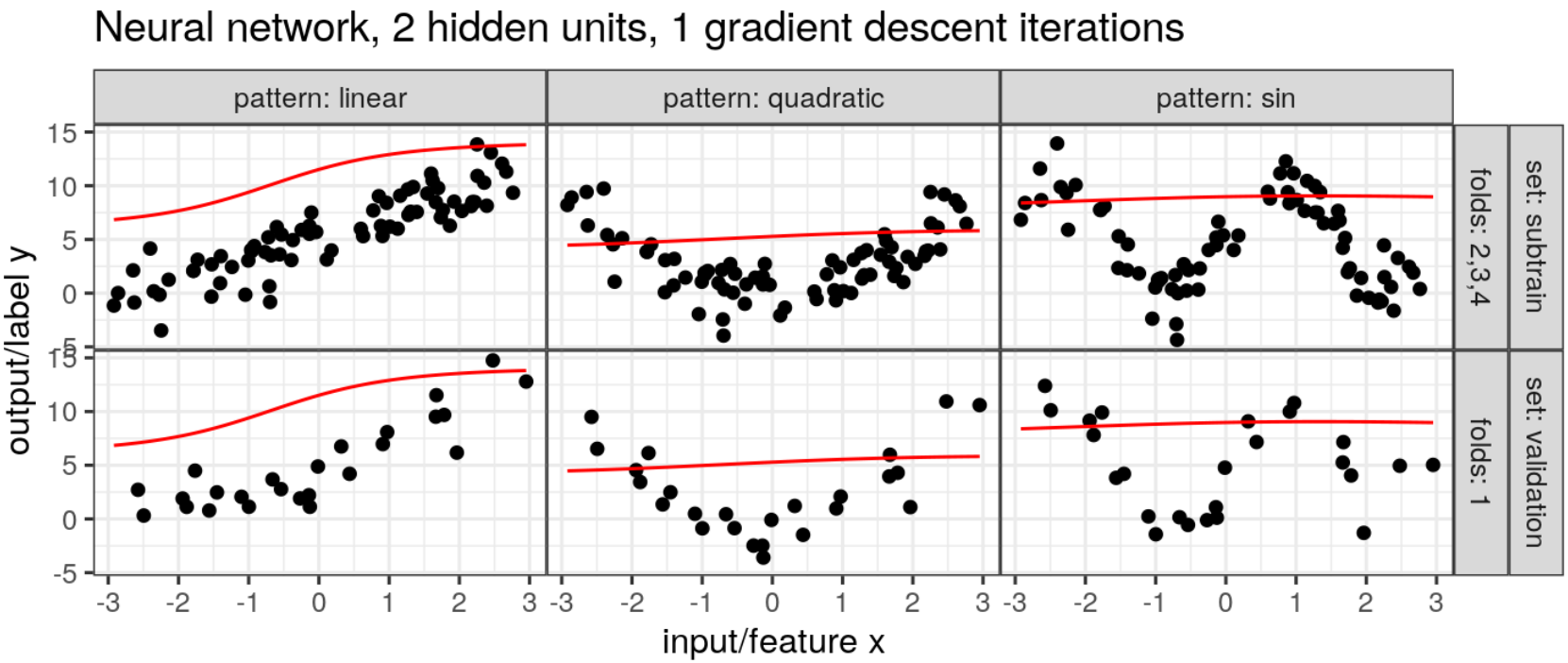
Overfitting

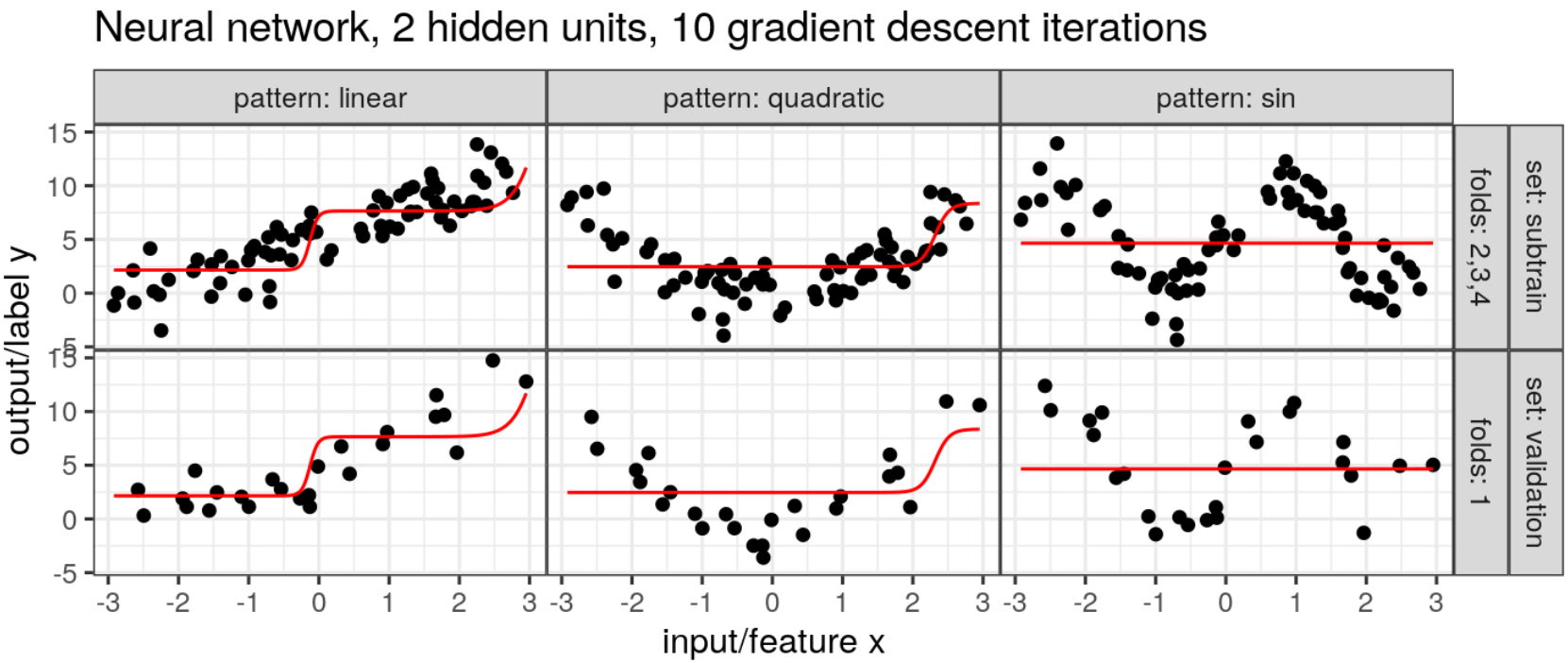
- Happens when subtrain error/loss decreases but validation error increases (as a function of some hyper-parameter)
- Here the hyper-parameter is the number of iterations of gradient descent, and overfitting starts after 10–100 iterations.
- To maximize prediction accuracy you need to choose a hyper-parameter with minimal validation error/loss.
- This optimal hyper-parameter will depend on the data set.
- In this case that means choosing 10 iterations for the linear data set, and 100 iterations for the quadratic/sin data sets.

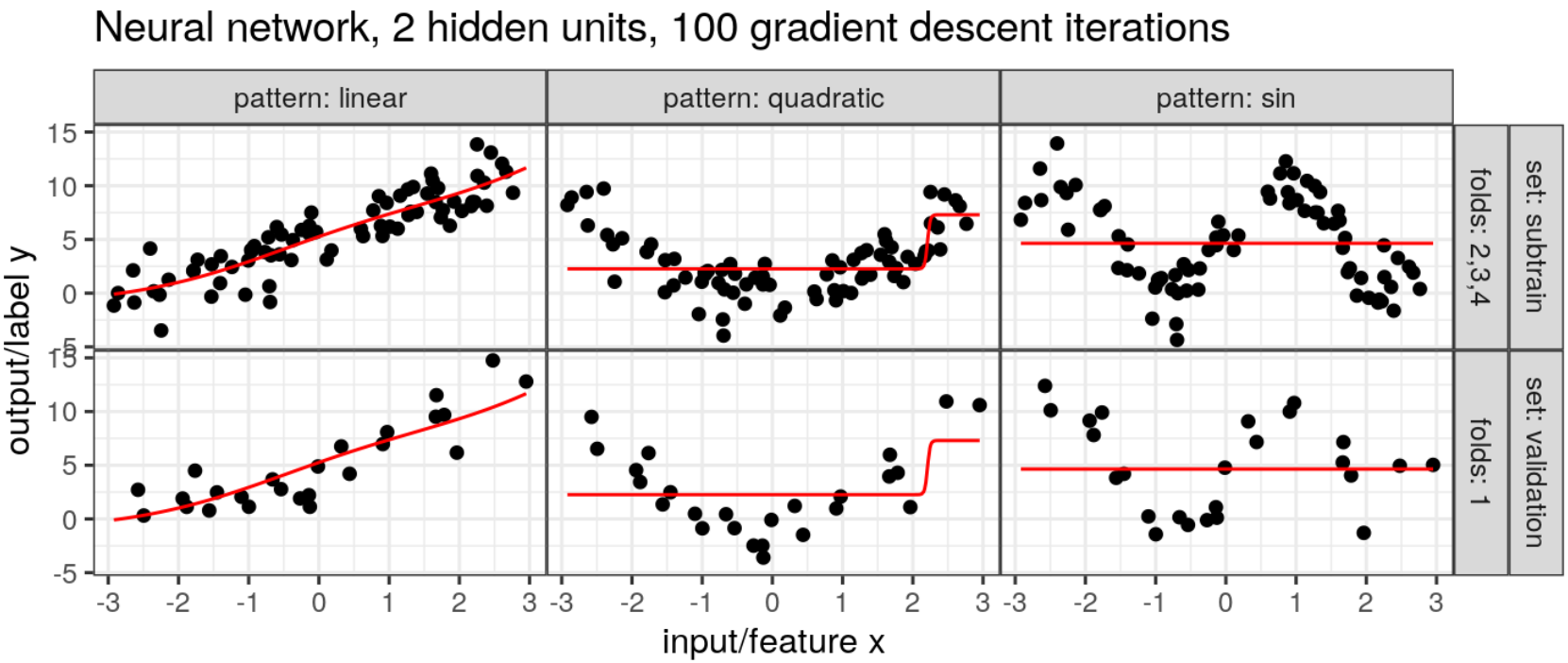
Computing subtrain/validation loss curves

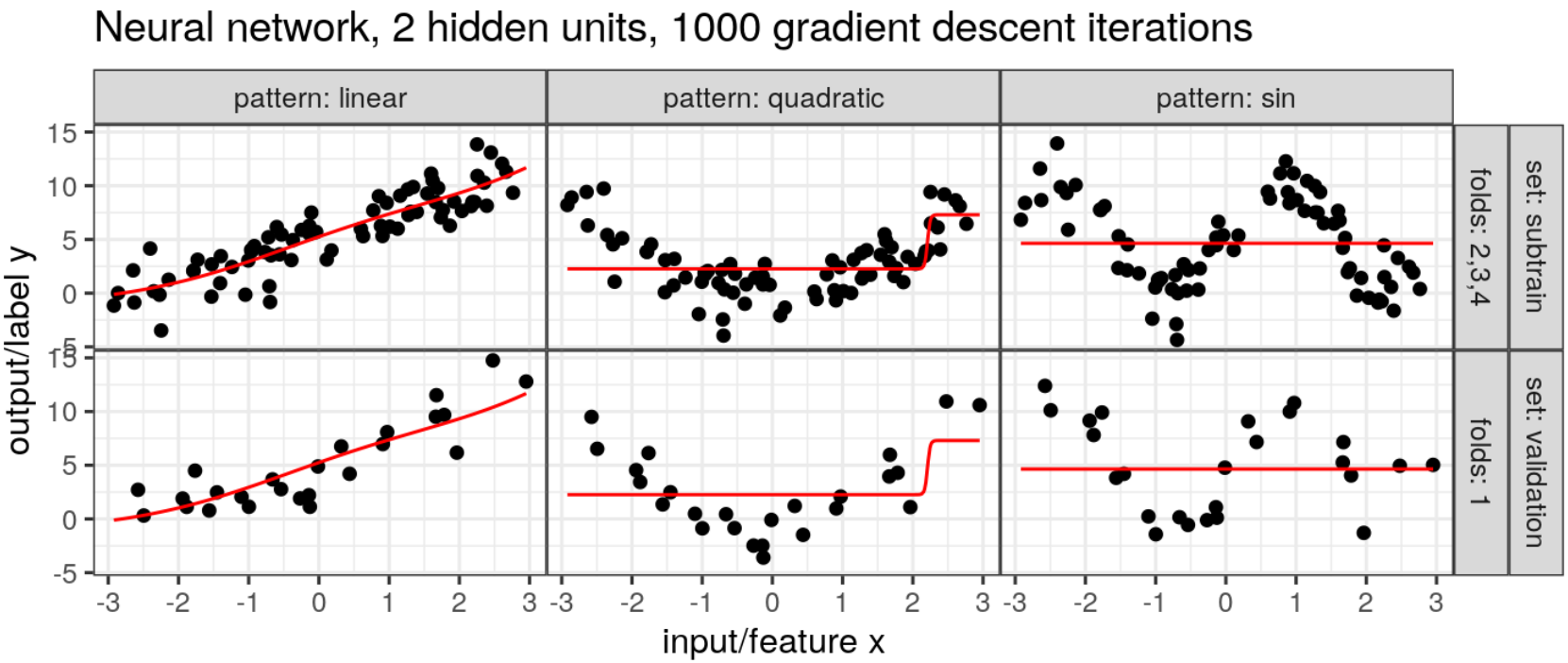
```
...
  pattern.data[, pred.y := as.numeric(predict(
    fit, pattern.data))]
  loss.dt.list[[paste(pattern, maxit)]] <- data.table(
    pattern=p, maxit=i, pattern.data[, .(
      mse=mean((pred.y-y)^2)), by=set])
}
loss.dt <- do.call(rbind, loss.dt.list)
```

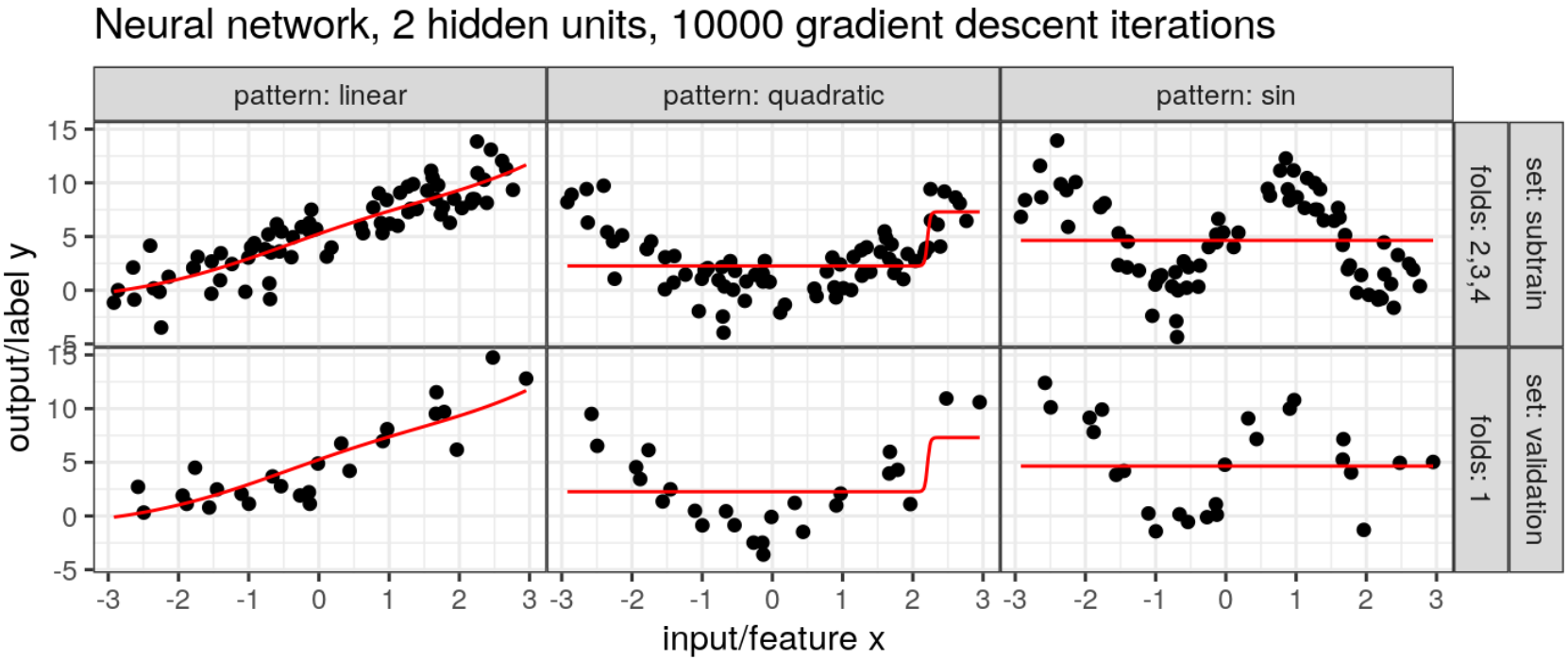
```
      pattern        set maxit         mse
 1:    linear   subtrain     1  17.7785881
 2:    linear validation     1  23.0575528
...
29:      sin   subtrain 10000   0.6970455
30:      sin validation 10000  19.5051785
```

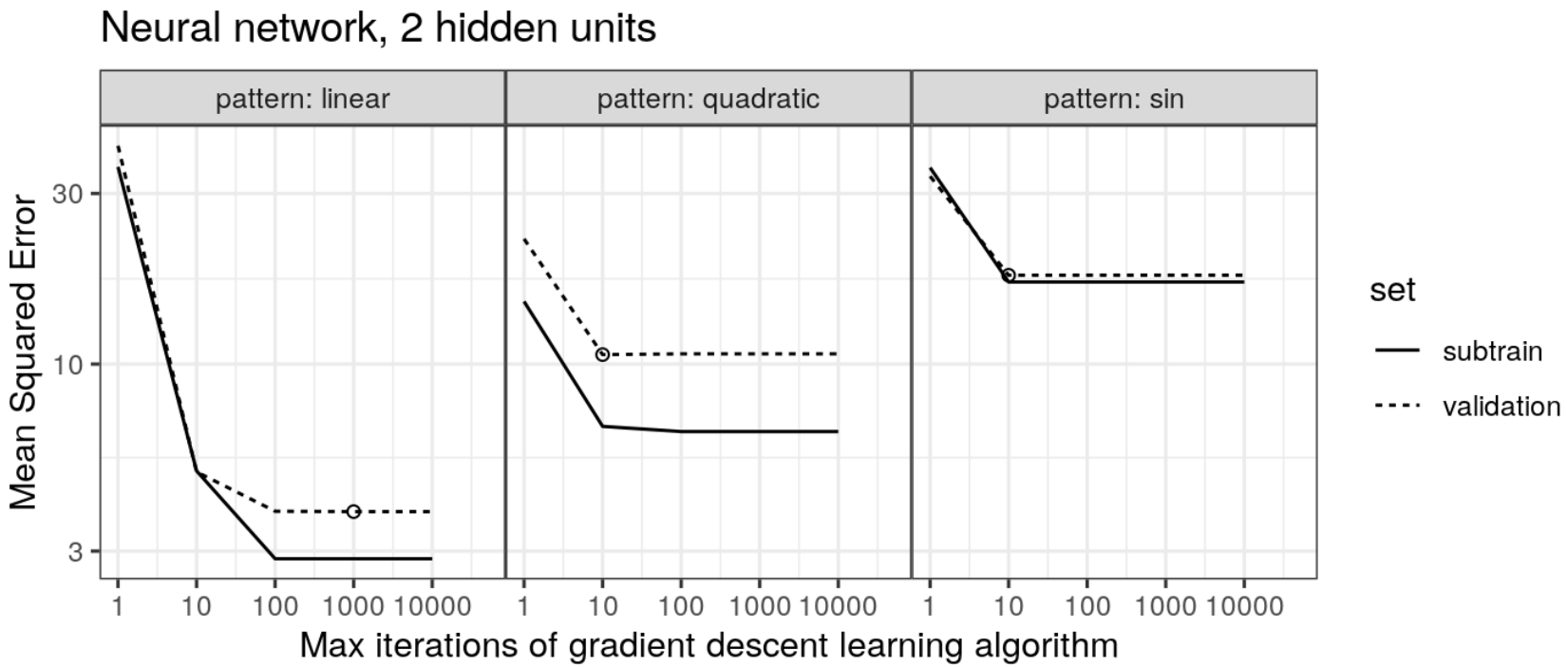

Neural network, 2 hidden units, 1 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
set: validation
folds: 2,3,4
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3


Neural network, 2 hidden units, 10 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
set: validation
folds: 2,3,4
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3


Neural network, 2 hidden units, 100 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
folds: 2,3,4
set: validation
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3


Neural network, 2 hidden units, 1000 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
folds: 2,3,4
set: validation
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3


Neural network, 2 hidden units, 10000 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
folds: 2,3,4
set: validation
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3


Neural network, 2 hidden units
pattern: linear
pattern: quadratic
pattern: sin
Mean Squared Error
30
10
3
1
10
100
1000
10000
Max iterations of gradient descent learning algorithm
set
subtrain
validation

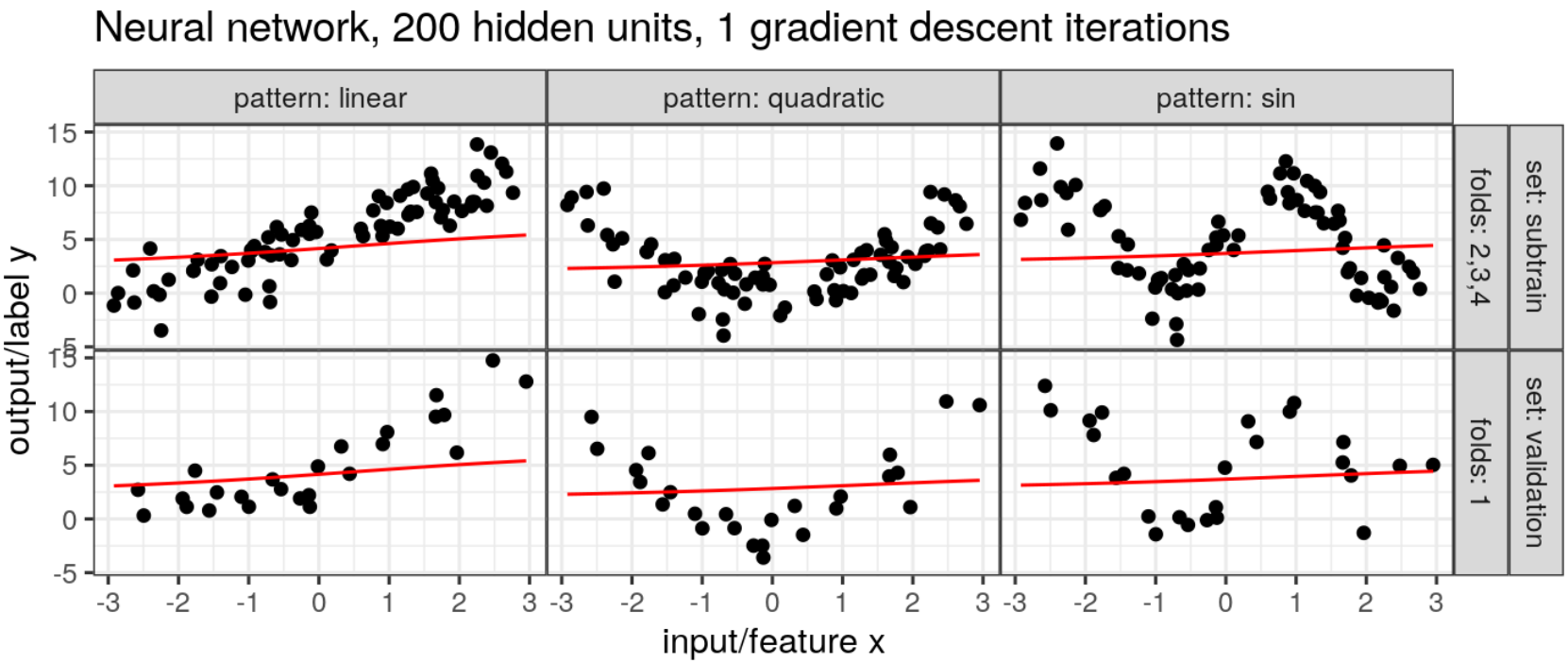
How does the neural network predict/learn?

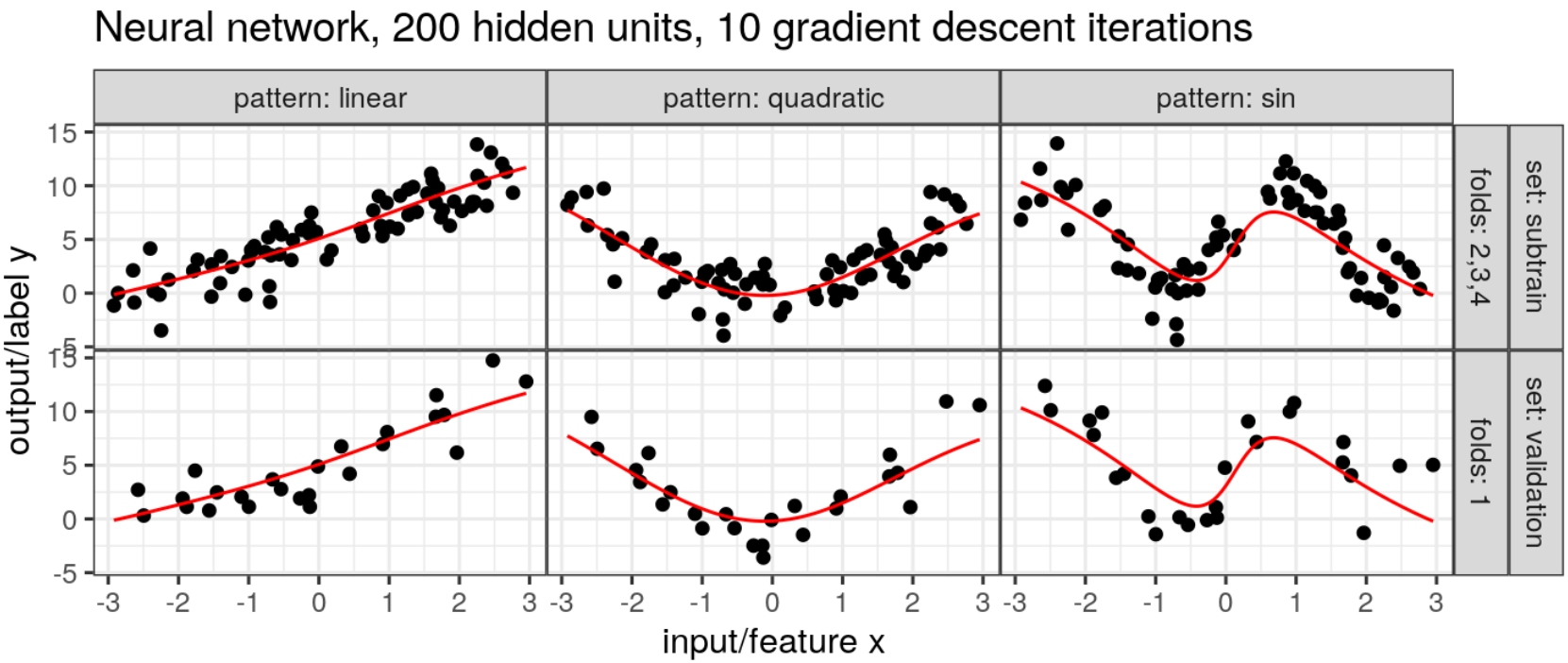
$$\text{hidden units:} \quad h_j = \sigma(w_j x + b_j)$$

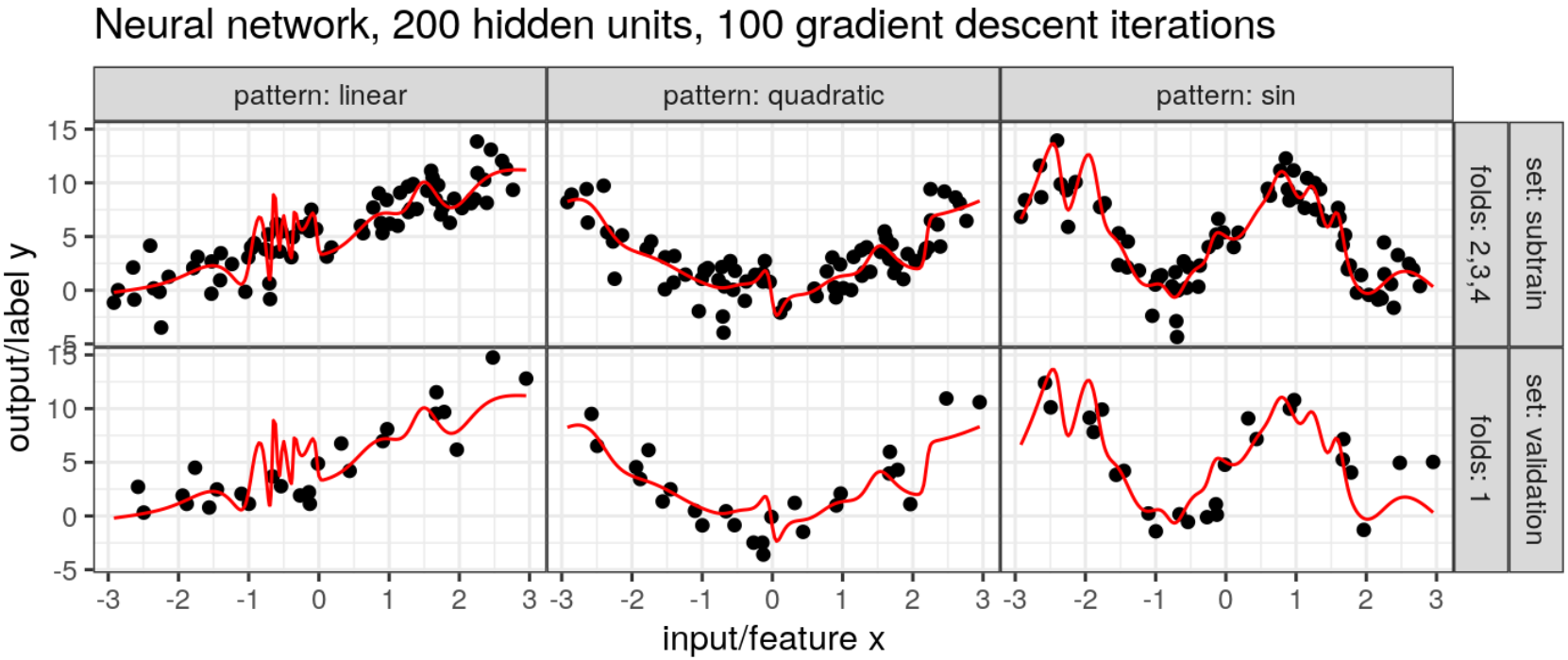
$$\text{predicted output:} \quad \text{o} = \hat{y} = \sum_{j=1}^{u} v_j h_j + \beta$$

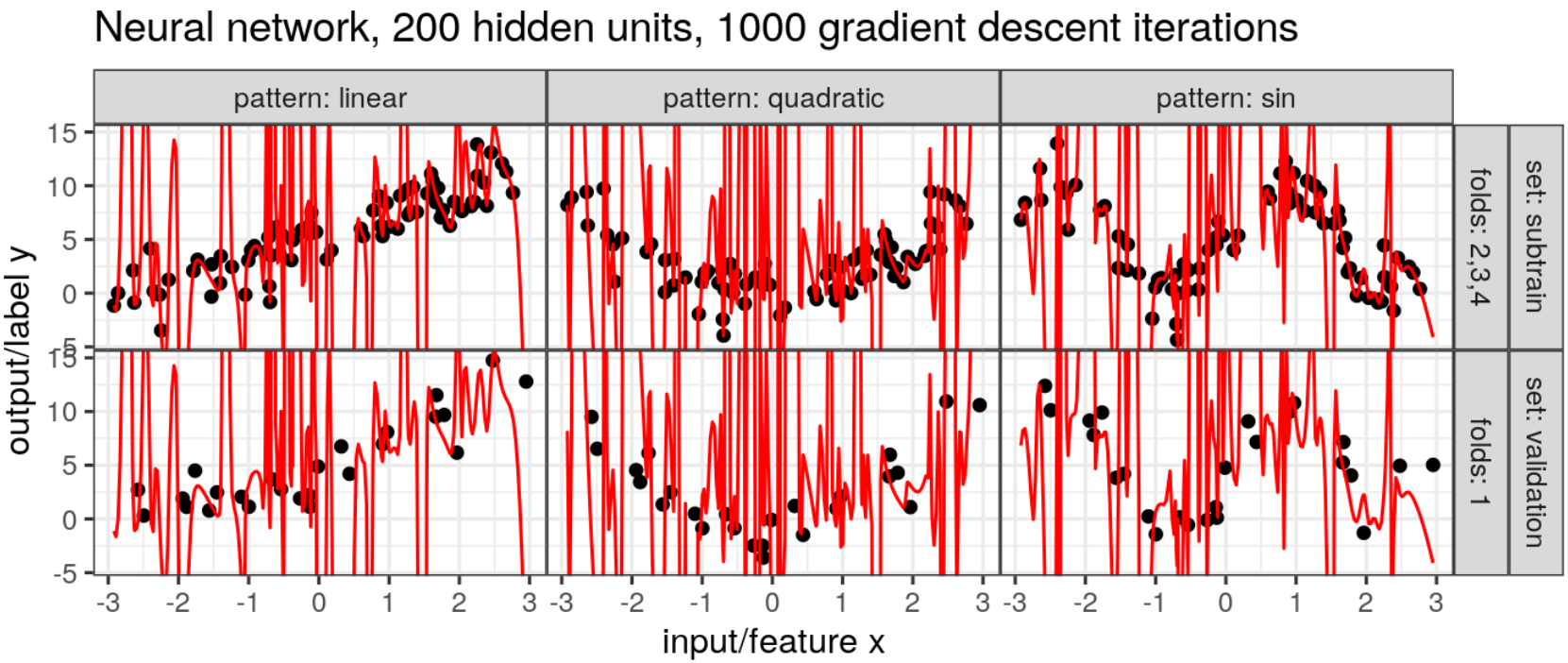
- $x \in \mathbb{R}$ is the input/feature (`i1` in R notation below).
- $\sigma : \mathbb{R} \to [0, 1]$ is the sigmoid activation function, $\sigma(z) = (1 + e^{-z})^{-1}$.
- $h_j \in [0, 1]$ are u hidden units, e.g. $u \in \{20, 2\}$ (`h1`,`h2`).
- $w_j, b_j, v_j, \beta \in \mathbb{R}$ are the weights which are learned using gradient descent to minimize the squared error, e.g. for $u = 2$:

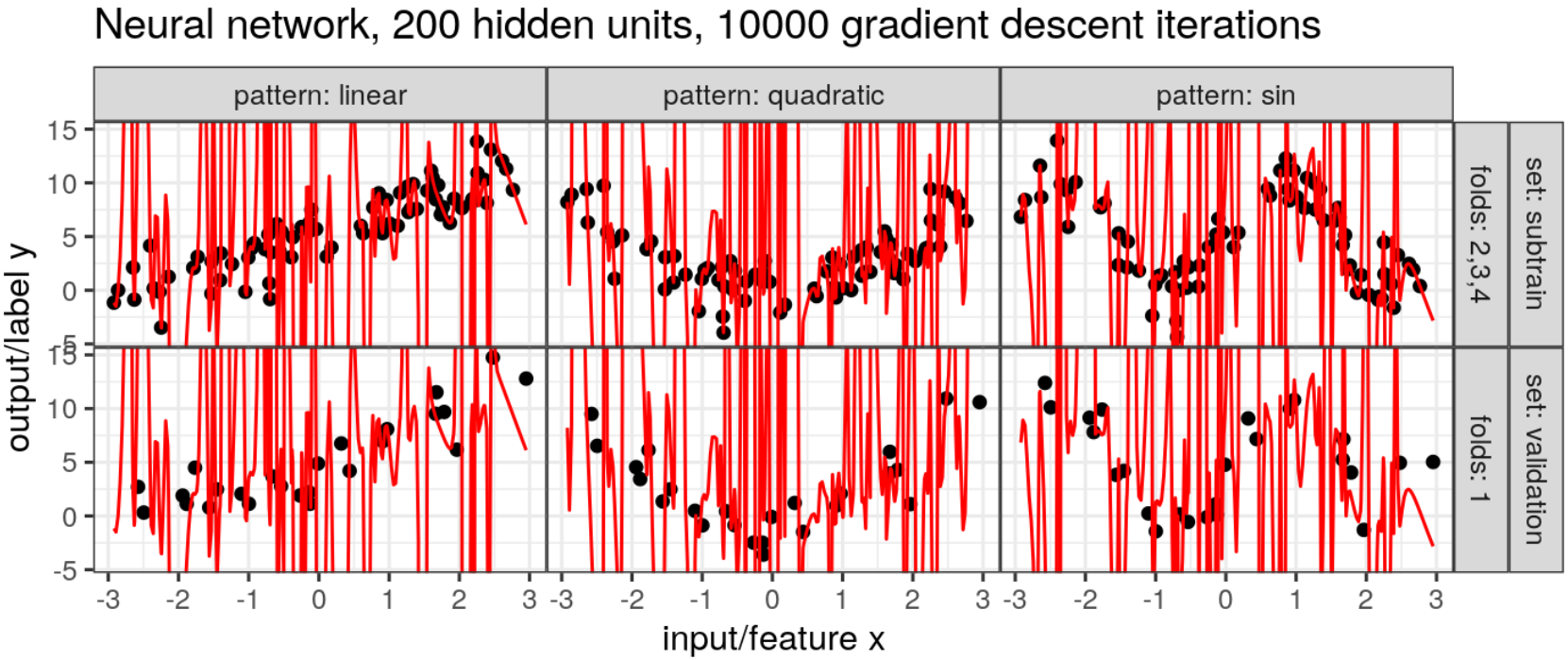
```
> coef(fit)
     b->h1      i1->h1      b->h2      i1->h2
-40.770433  -2.636863 -20.719333   2.629485
      b->o       h1->o       h2->o
 4.643852  18.973225  28.348342
```

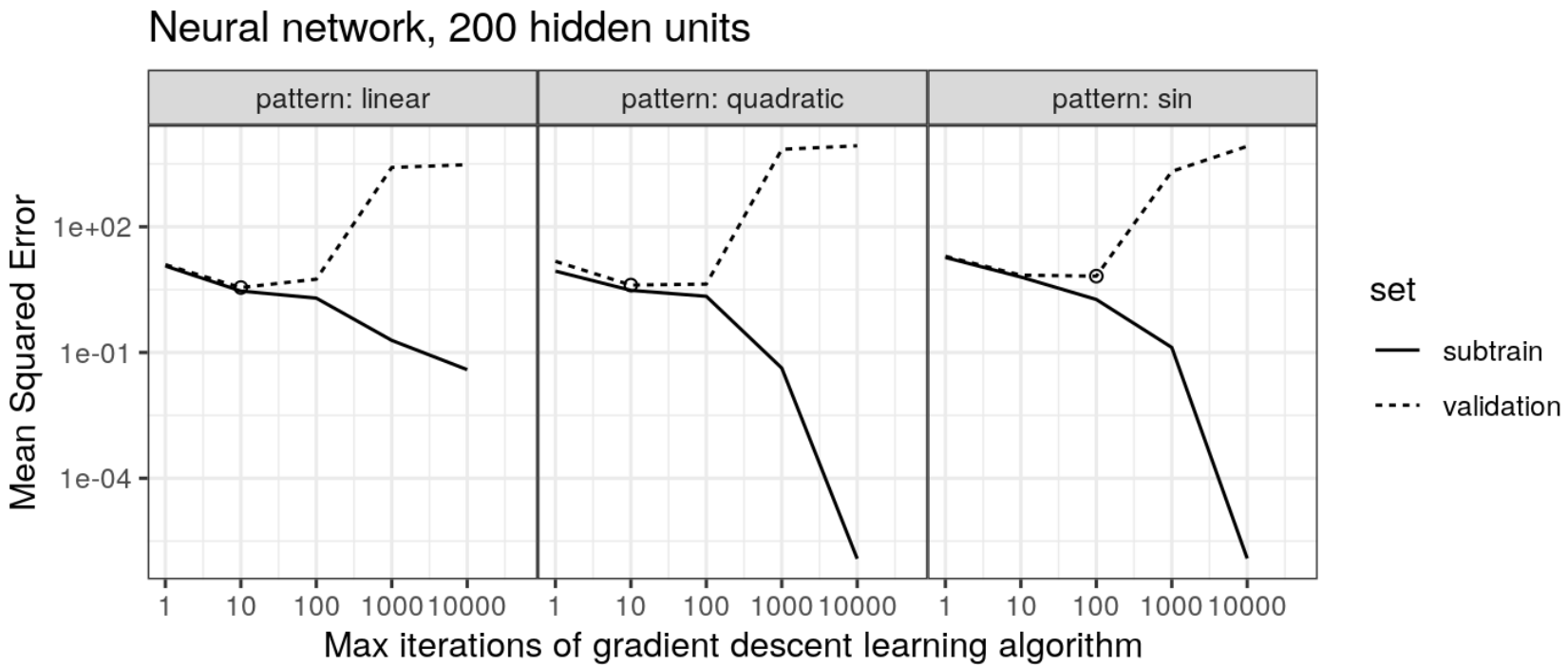

Neural network, 200 hidden units, 1 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
set: validation
folds: 2,3,4
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3


Neural network, 200 hidden units, 10 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
set: validation
folds: 2,3,4
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3


Neural network, 200 hidden units, 100 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
set: validation
folds: 2,3,4
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3


Neural network, 200 hidden units, 1000 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
set: validation
folds: 2,3,4
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3

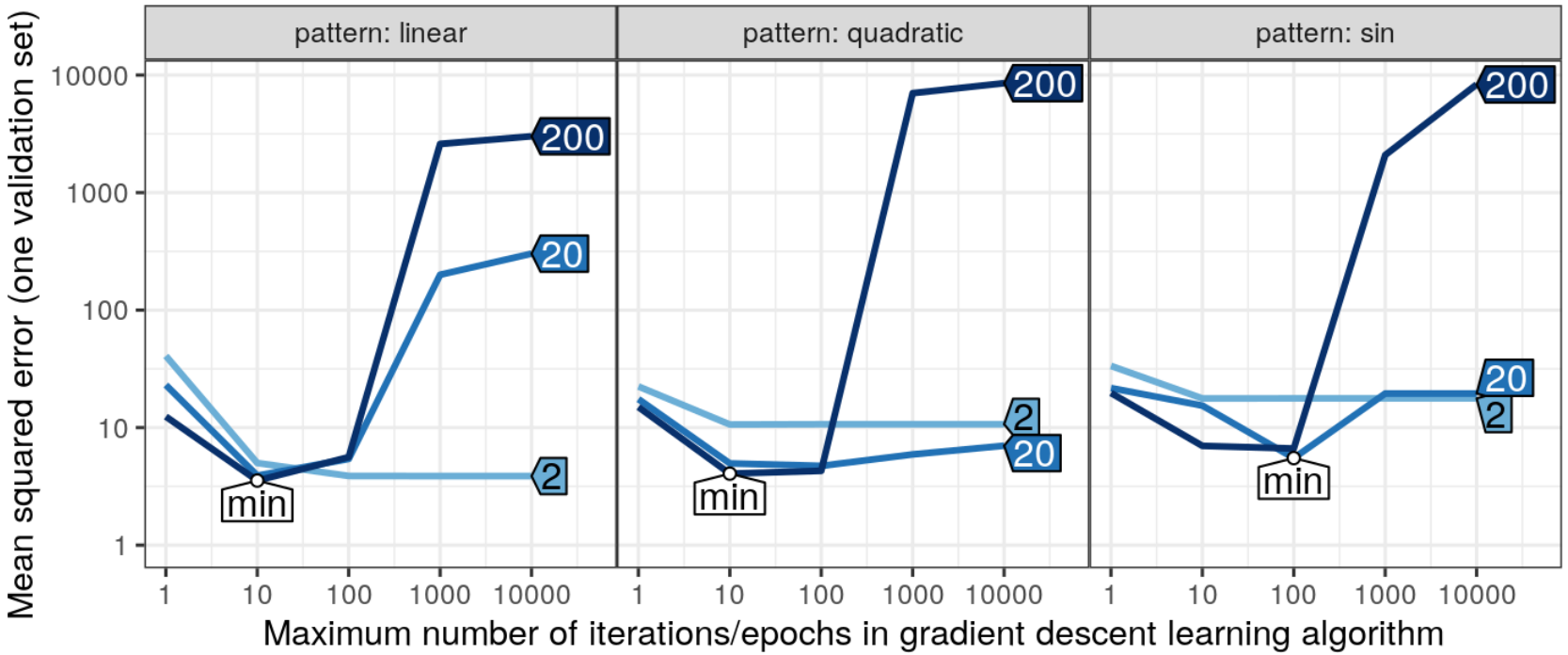

Neural network, 200 hidden units, 10000 gradient descent iterations
pattern: linear
pattern: quadratic
pattern: sin
set: subtrain
folds: 2,3,4
set: validation
folds: 1
output/label y
input/feature x
15
10
5
0
-5
-3
-2
-1
0
1
2
3


Neural network, 200 hidden units
pattern: linear
pattern: quadratic
pattern: sin
Mean Squared Error
1e+02
1e-01
1e-04
1
10
100
1000
10000
Max iterations of gradient descent learning algorithm
set
subtrain
validation

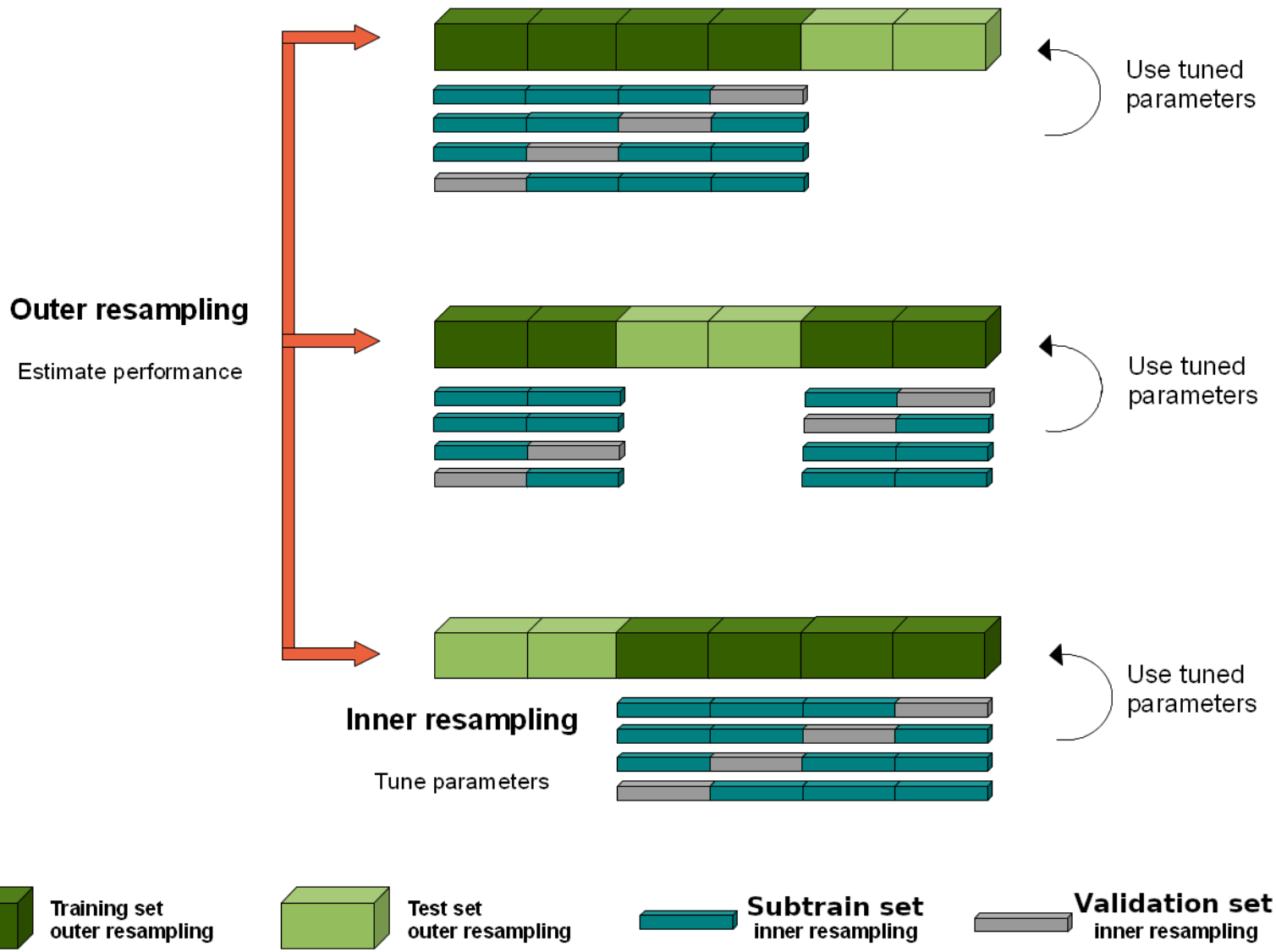
Comments on other model sizes

- Model with 2 hidden units is less flexible: does not fit the subtrain data as well, and does not overfit as much.
- Model with 200 hidden units is more flexible: fits subtrain data even better, and overfits even more.
- Therefore model size / number of hidden units is also a hyper-parameter that can be used to control overfitting.
- R code is the same, just use `size=2` or 200 (or even better, I wrote a for loop over size values).
- Conclusion: you need to use a held-out validation set to choose the best values of hyper-parameters (e.g. number of iterations, hidden units).

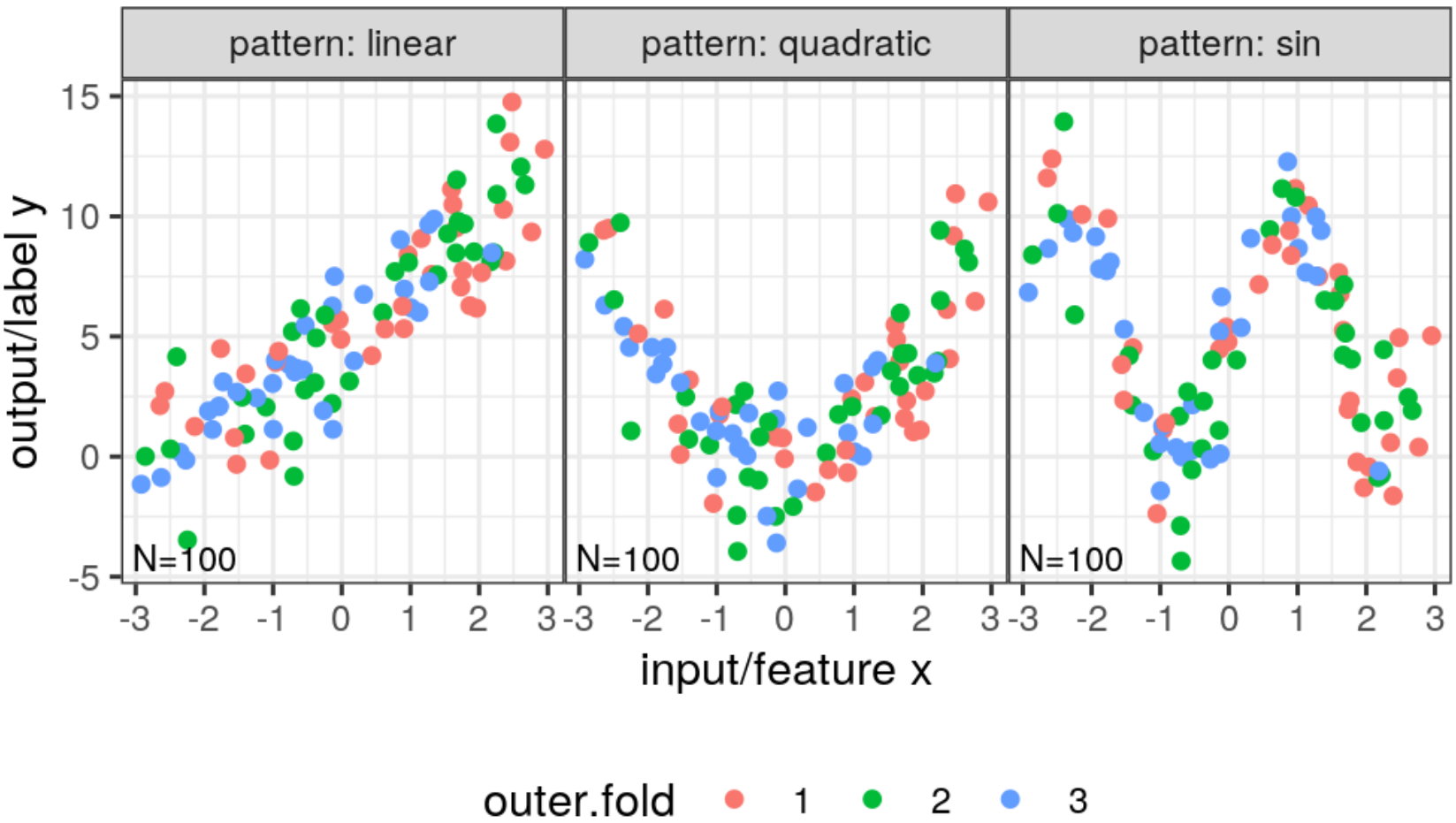
Minimizing over both hyper-parameters



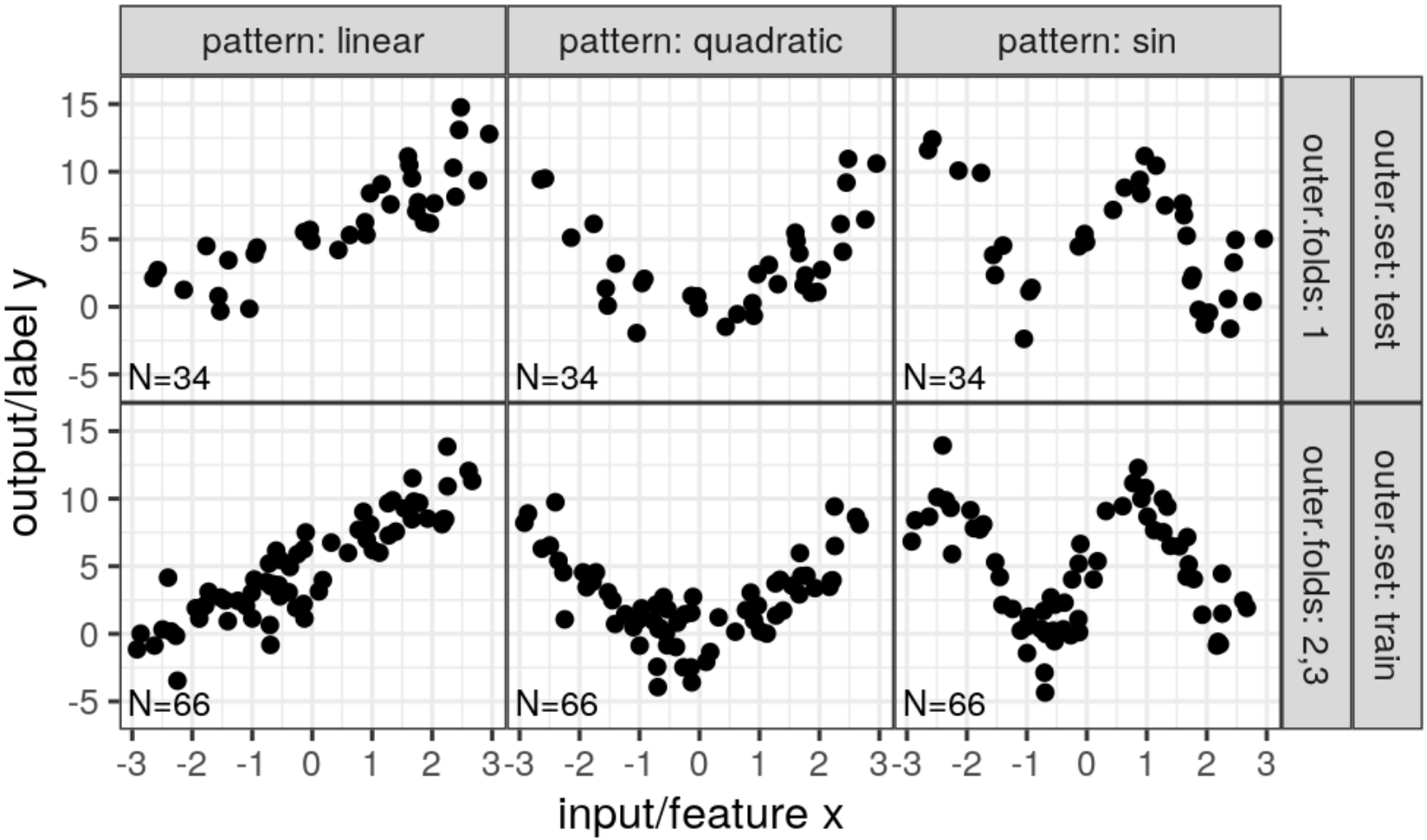
Inner and outer cross-validation



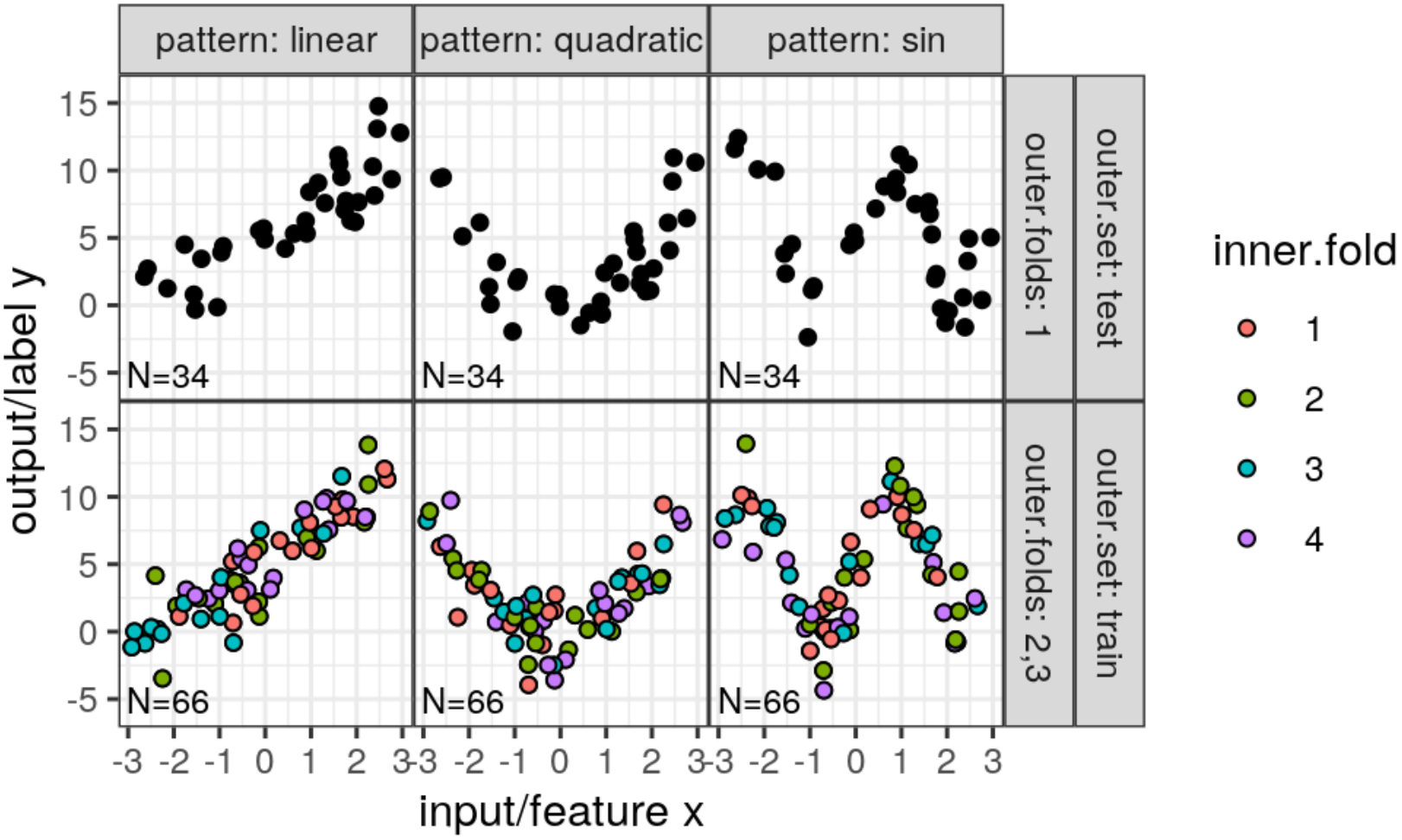
Outer cross-validation



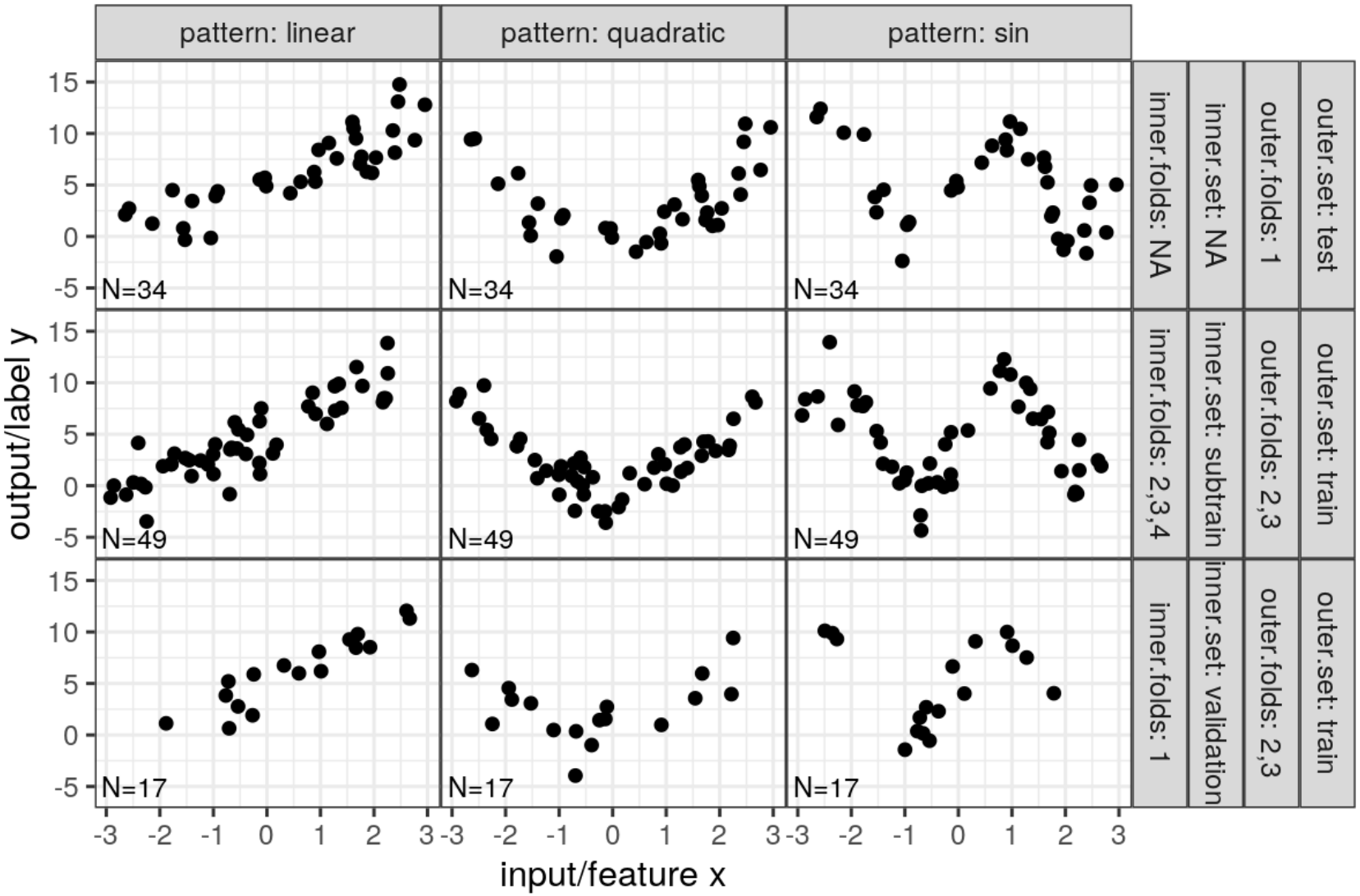
Fix one outer fold = one train/test split



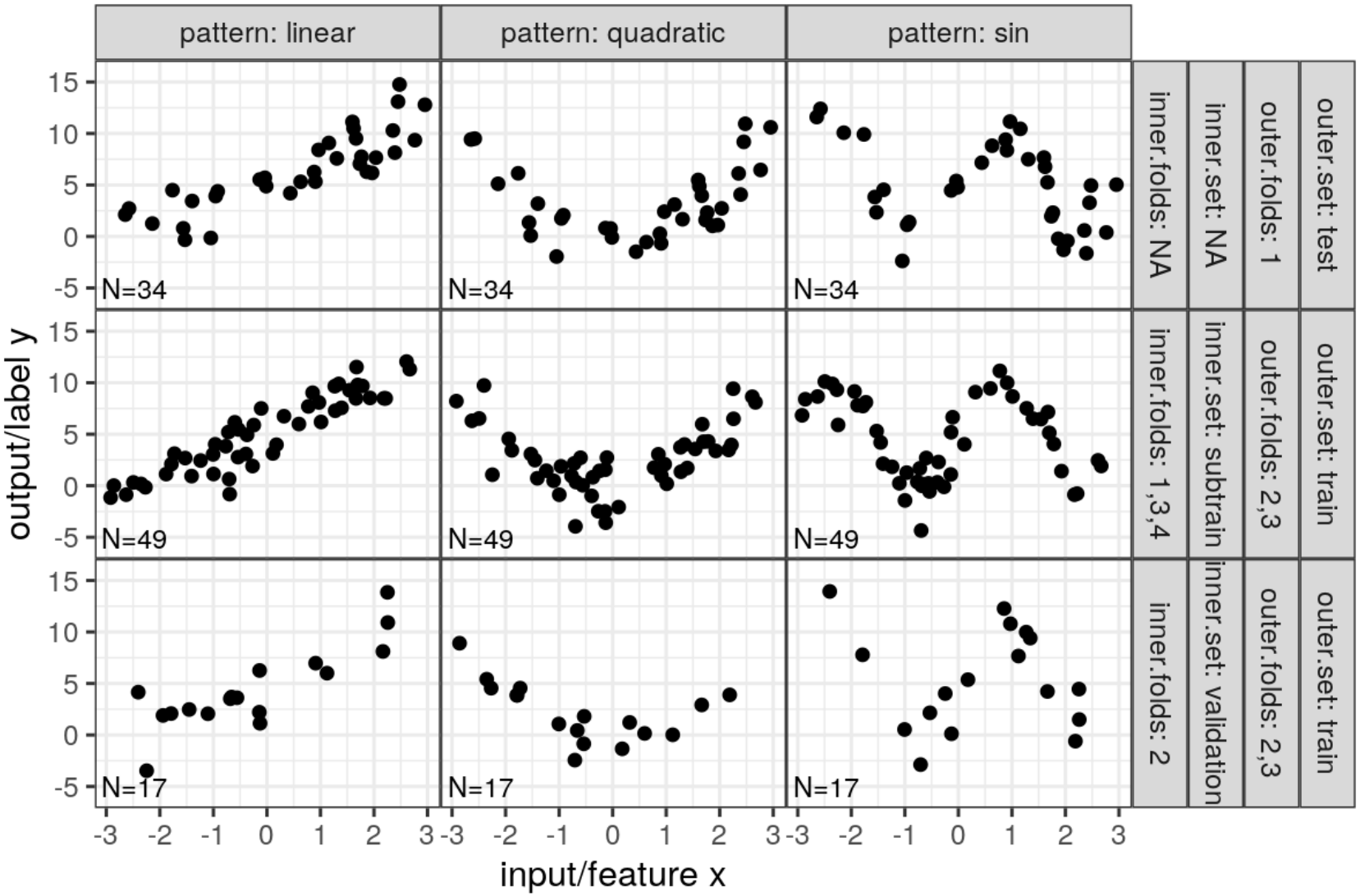
Assign inner folds for one train set



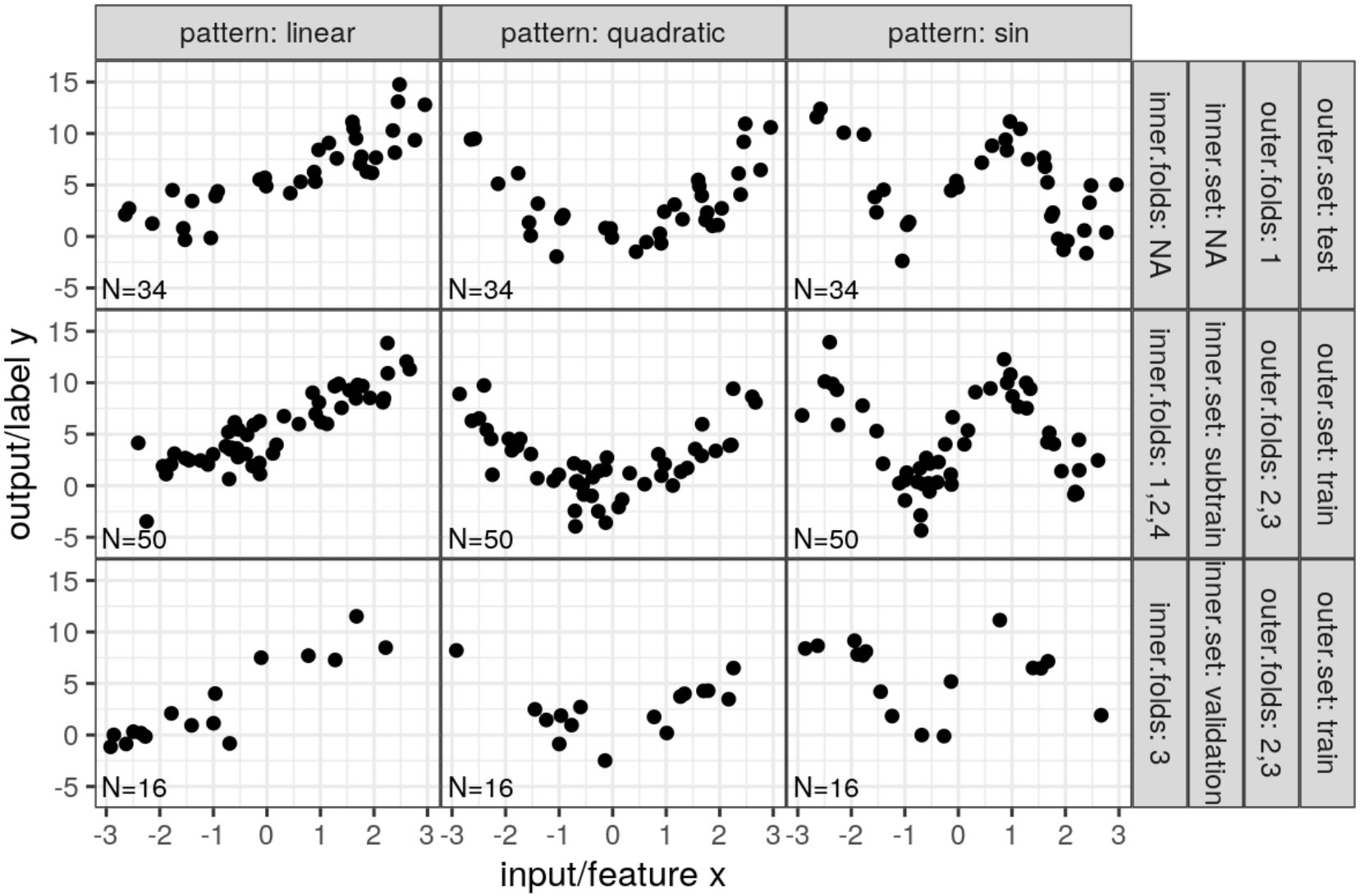
Fix one inner fold = one subtrain/validation split



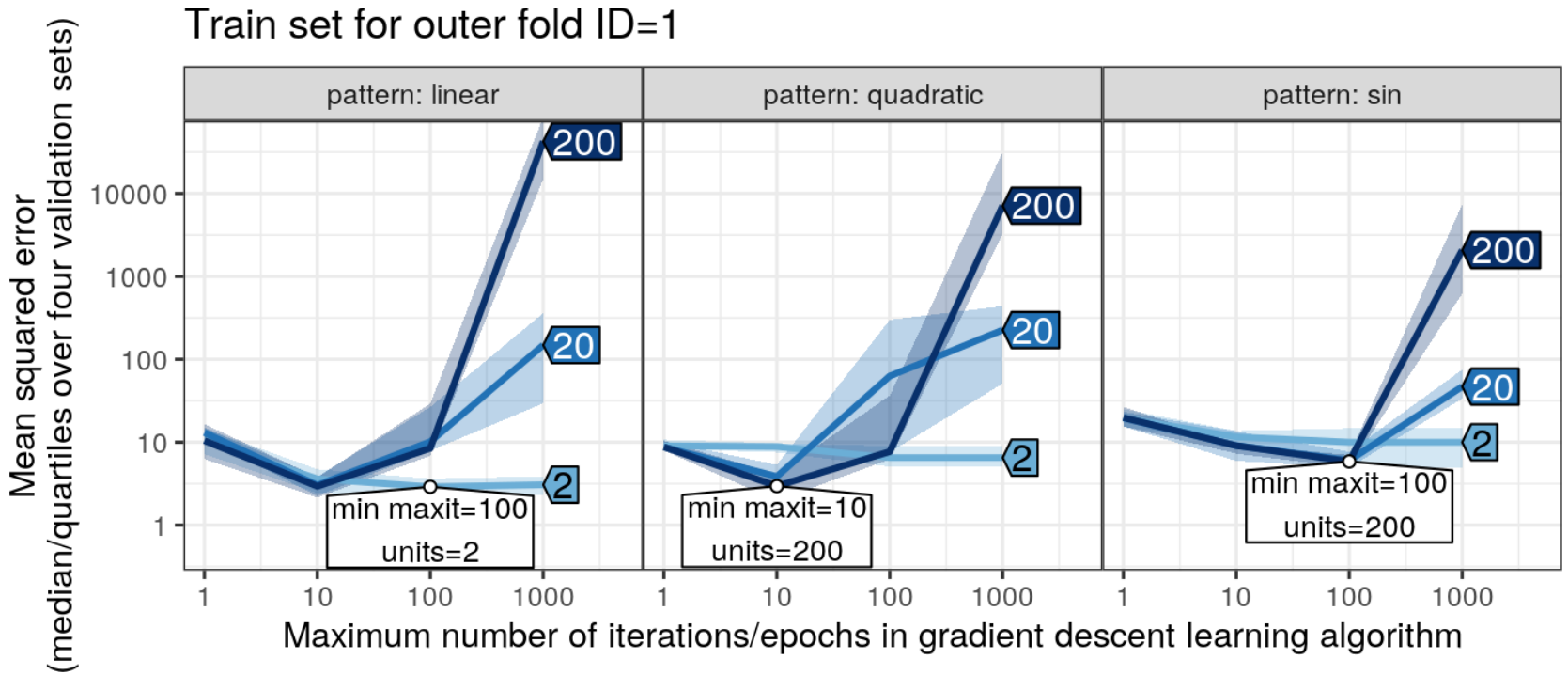
Another subtrain/validation split



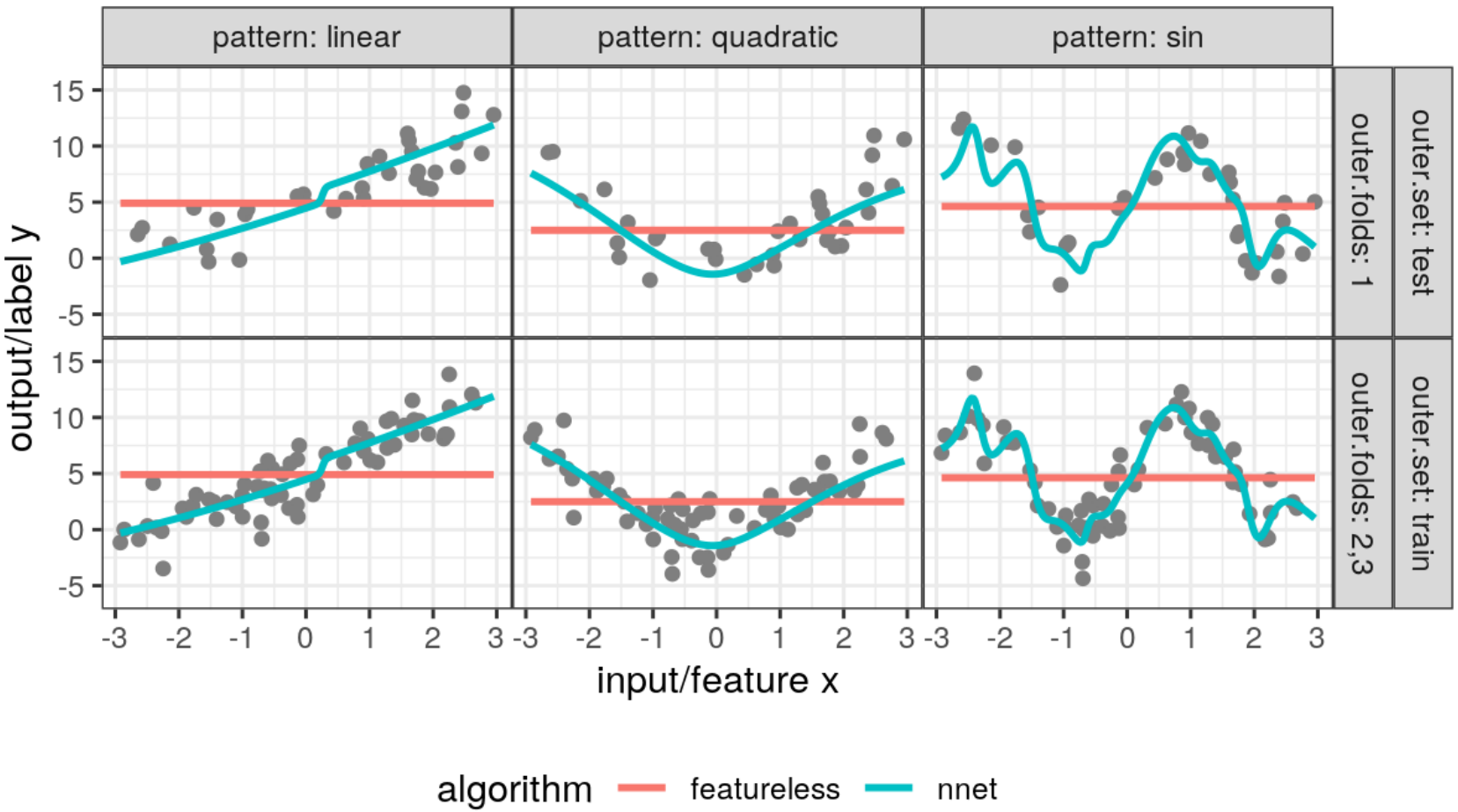
Another subtrain/validation split



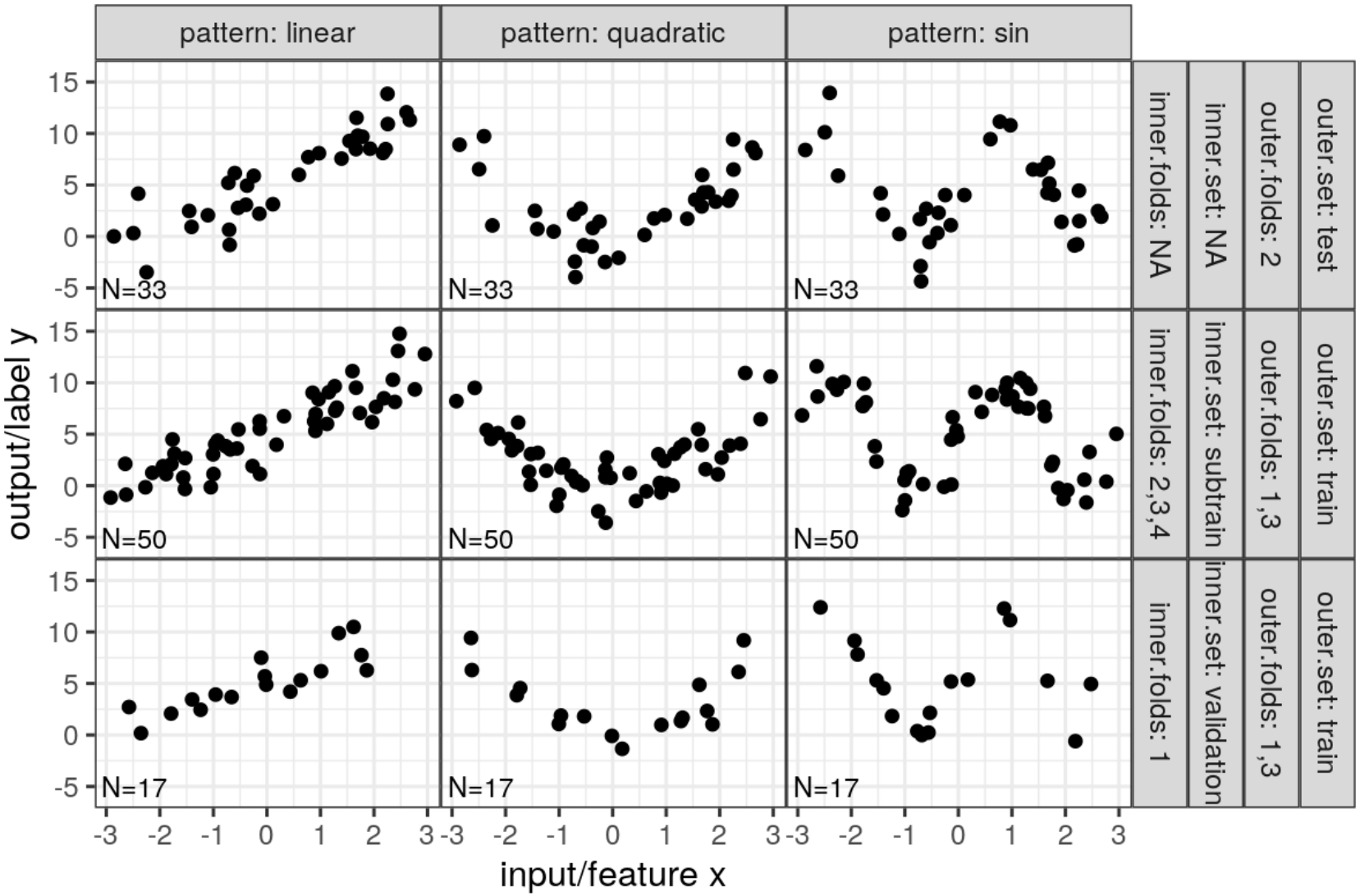
Average validation error over four splits



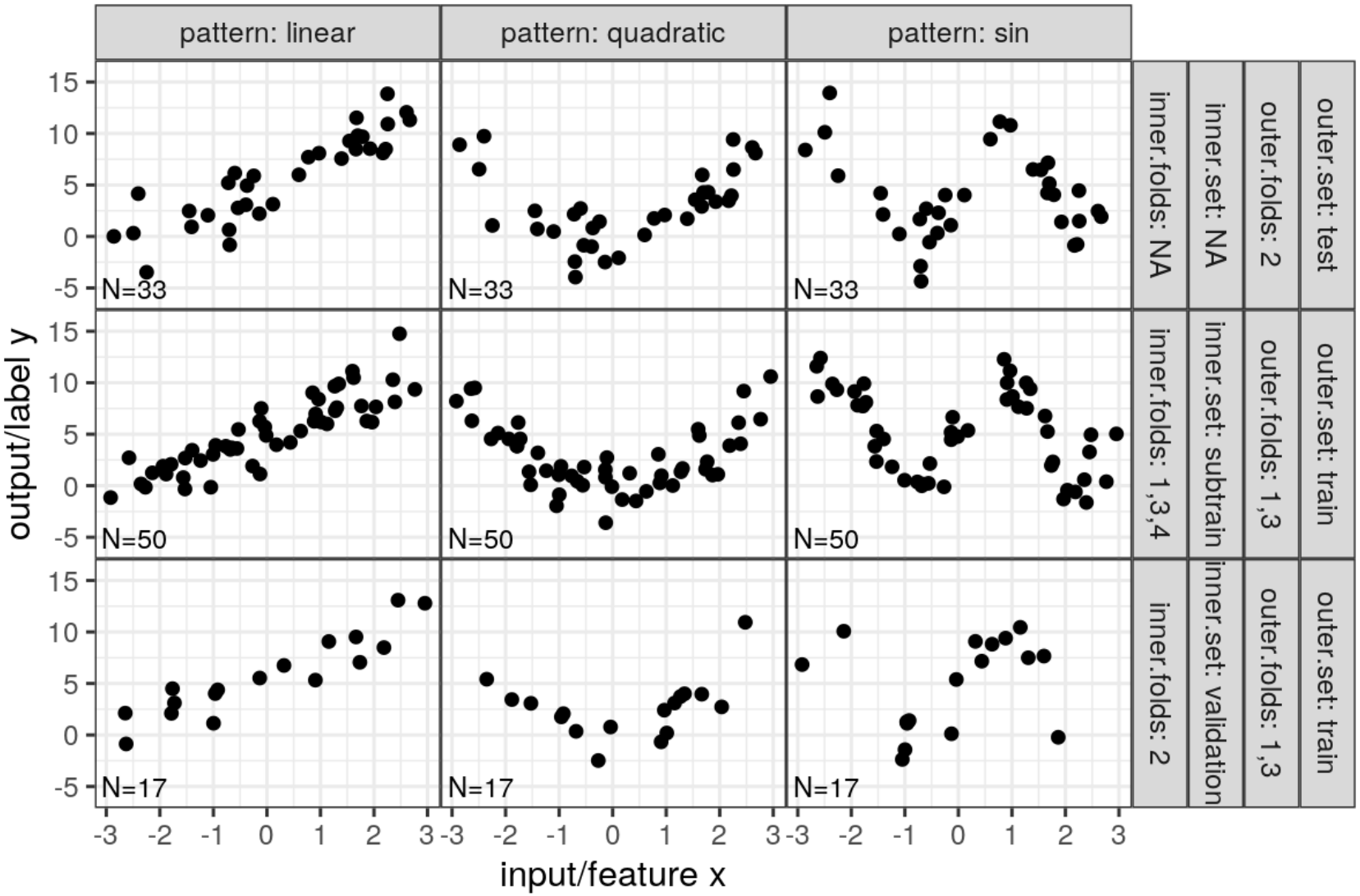
Final models fit to entire train set using best hyper-parameters



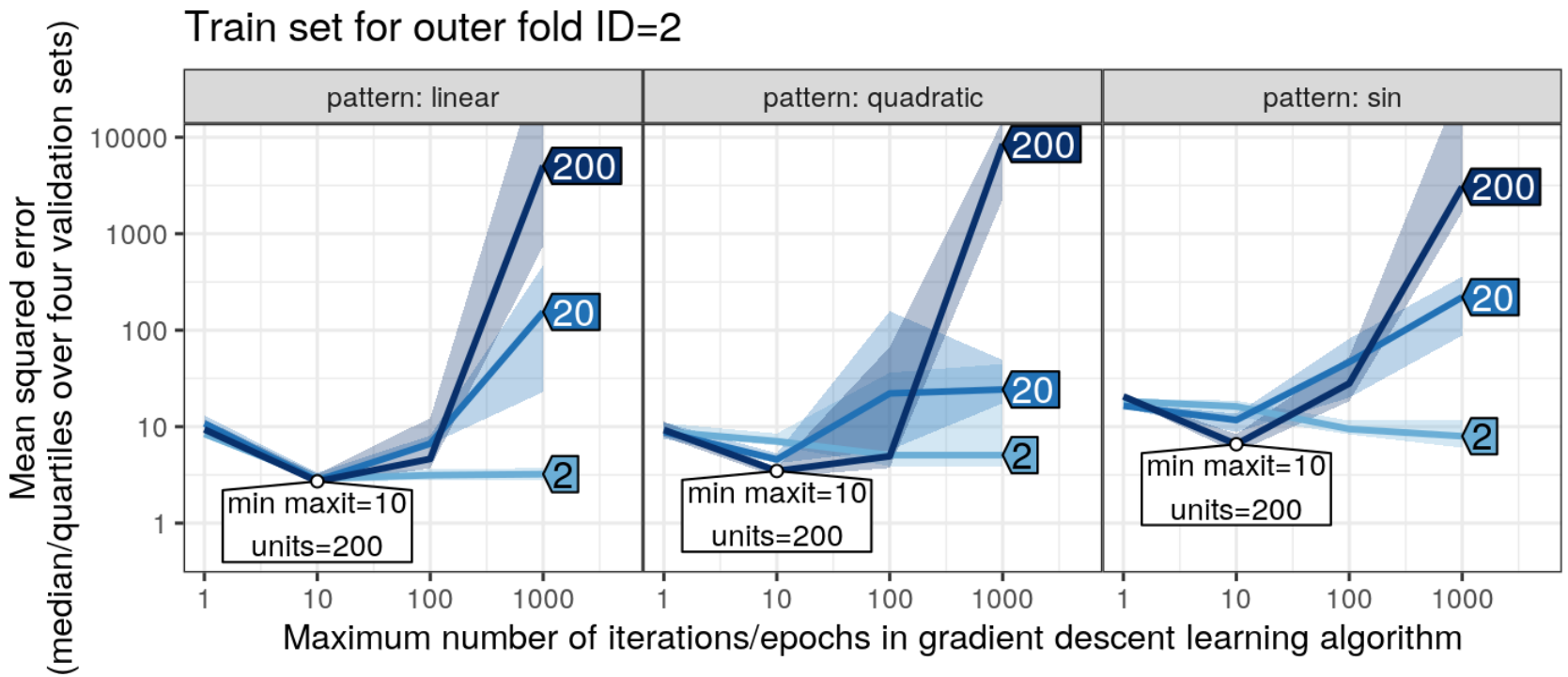
Fix one inner fold = one subtrain/validation split



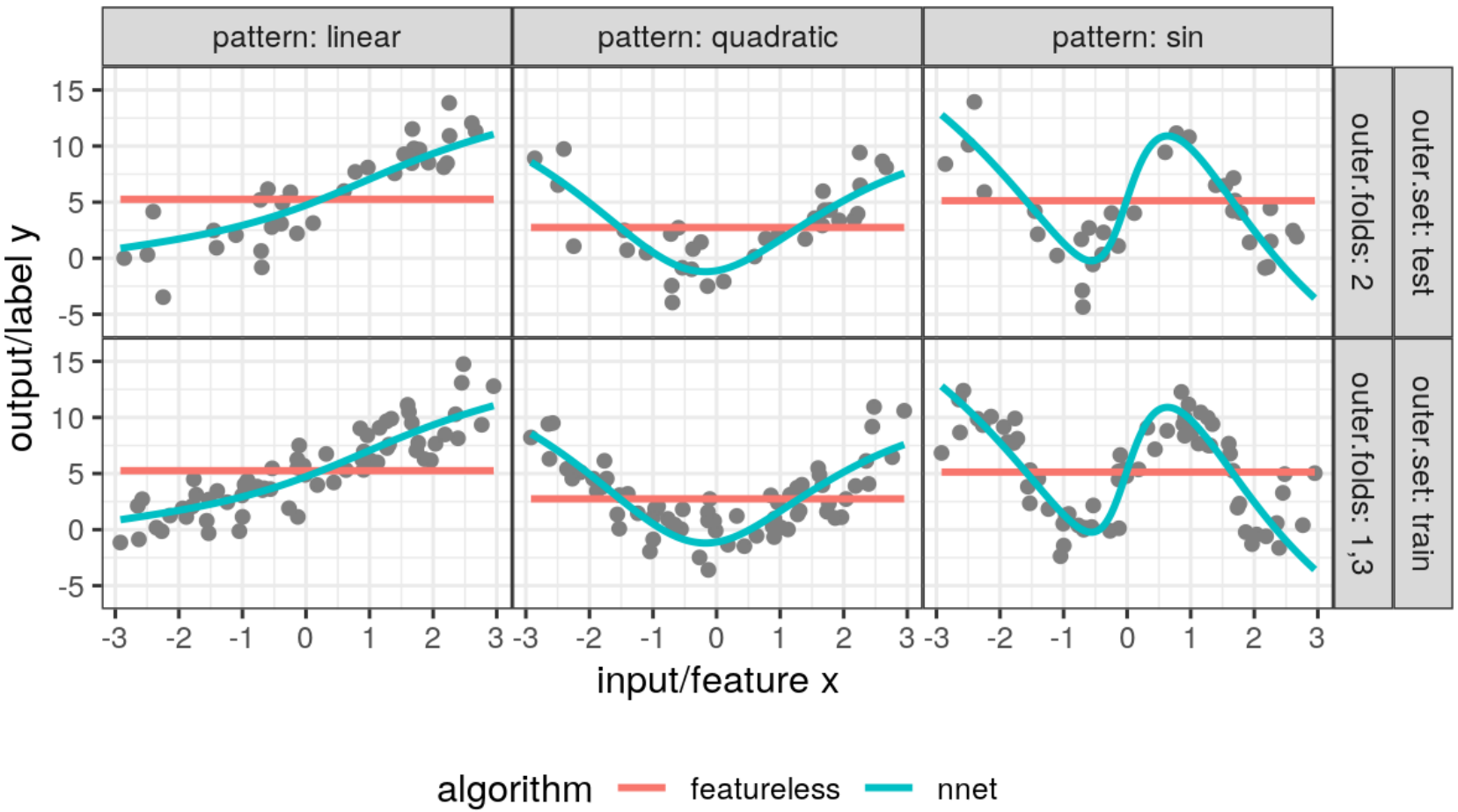
Another subtrain/validation split



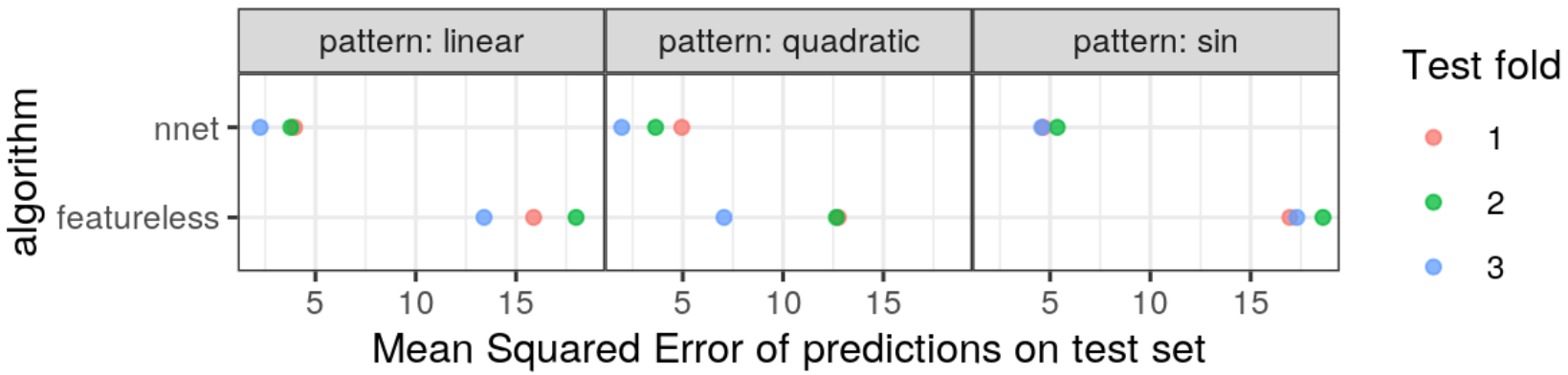
Average validation error over four splits



Final models fit to entire train set using best hyper-parameters



Finally compare test error across all train/test splits

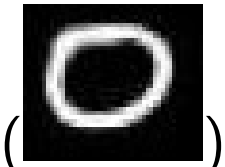
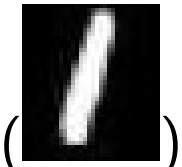


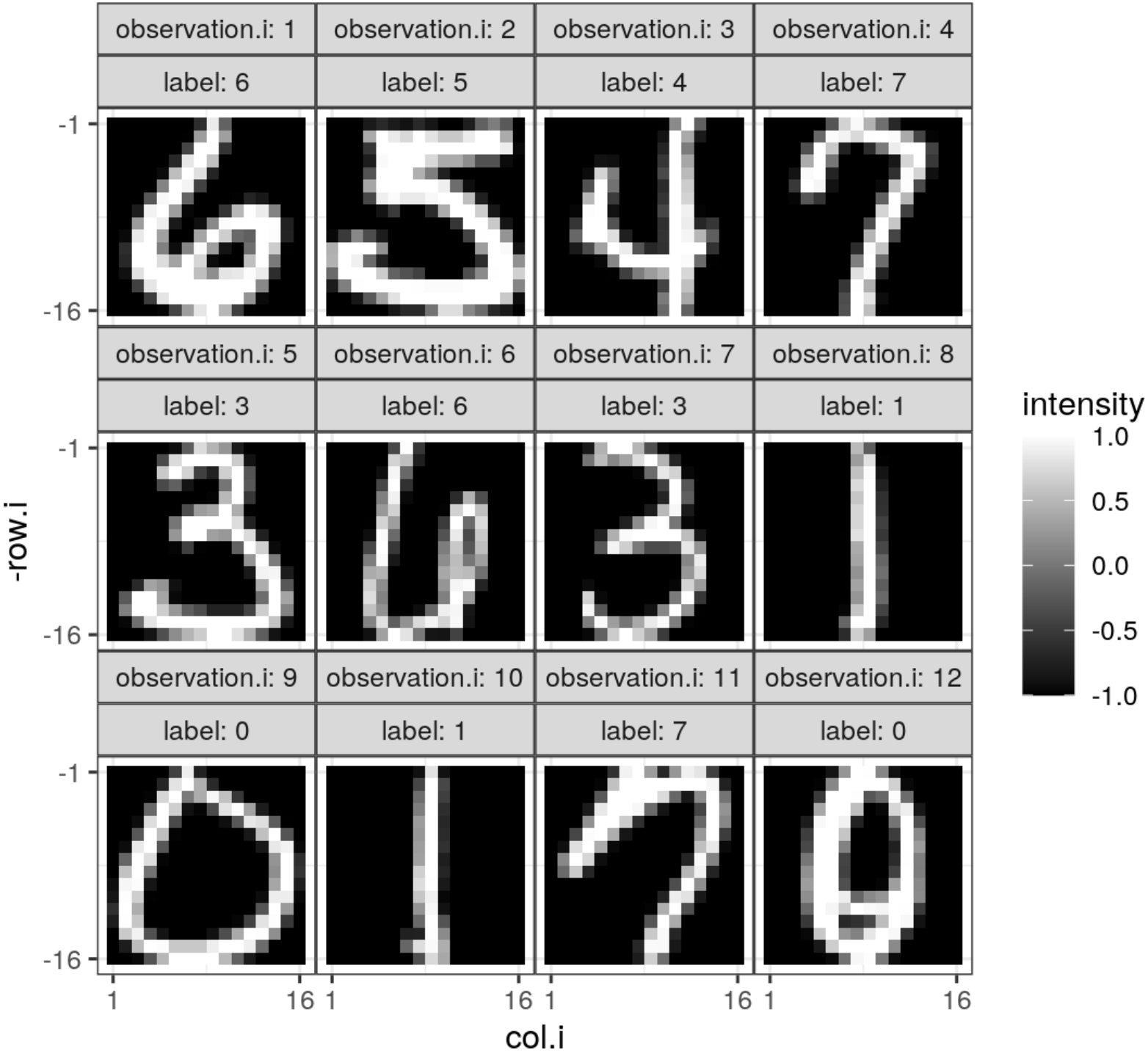
Introduction and applications

Demonstration of overfitting in regression / first steps with R

Classifying images of digits

Image classification

- One of the most popular/successful applications of machine learning.
- Input: image file $x \in \mathbb{R}^{h \times w \times c}$ where h is the height in pixels, w is the width, c is the number of channels, e.g. RGB image $c = 3$ channels.
- In this tutorial we use images with $h = w = 16$ pixels and $c = 1$ channel (grayscale, smaller values are darker).
- Output: class/category y (from a finite set).
- In this tutorial there are ten image classes $y \in \{0, 1, \ldots, 9\}$, one for each digit.
- Want to learn f such that $f($  $) = 0$, $f($  $) = 1$, etc.
- Code for figures in this section: `https://github.com/tdhock/2020-yiqi-summer-school/blob/master/figure-validation-loss.R`


observation.i: 1
label: 6
observation.i: 2
label: 5
observation.i: 3
label: 4
observation.i: 4
label: 7
observation.i: 5
label: 3
observation.i: 6
label: 6
observation.i: 7
label: 3
observation.i: 8
label: 1
observation.i: 9
label: 0
observation.i: 10
label: 1
observation.i: 11
label: 7
observation.i: 12
label: 0
-1
-16
1
16
-row.i
col.i
intensity
1.0
0.5
0.0
-0.5
-1.0

Representation of digits in CSV

- Each image/observation is one row.
- First column is output/label/class to predict.
- Other 256 columns are inputs/features (pixel intensity values).

```
 1:  6 -1 -1  ... -1.000 -1.000  -1
 2:  5 -1 -1  ... -0.671 -0.828  -1
 3:  4 -1 -1  ... -1.000 -1.000  -1
 4:  7 -1 -1  ... -1.000 -1.000  -1
 5:  3 -1 -1  ... -0.883 -1.000  -1
 6:  6 -1 -1  ... -1.000 -1.000  -1
 7:  3 -1 -1  ... -1.000 -1.000  -1
 8:  1 -1 -1  ... -1.000 -1.000  -1
 9:  0 -1 -1  ... -1.000 -1.000  -1
10:  1 -1 -1  ... -1.000 -1.000  -1
11:  7 -1 -1  ... -1.000 -1.000  -1
12:  0 -1 -1  ... -1.000 -1.000  -1
```

Converting label column to matrix for neural network

This is a "one hot" encoding of the class labels.

```
zip.dt <- data.table::fread("zip.gz")
zip.y.mat <- keras::to_categorical(zip.dt$V1)
```

```
      0 1 2 3 4 5 6 7 8 9
 [1,] 0 0 0 0 0 0 1 0 0 0
 [2,] 0 0 0 0 0 1 0 0 0 0
 [3,] 0 0 0 0 1 0 0 0 0 0
 [4,] 0 0 0 0 0 0 0 1 0 0
 [5,] 0 0 0 1 0 0 0 0 0 0
 [6,] 0 0 0 0 0 0 1 0 0 0
 [7,] 0 0 0 1 0 0 0 0 0 0
 [8,] 0 1 0 0 0 0 0 0 0 0
 [9,] 1 0 0 0 0 0 0 0 0 0
[10,] 0 1 0 0 0 0 0 0 0 0
[11,] 0 0 0 0 0 0 0 1 0 0
[12,] 1 0 0 0 0 0 0 0 0 0
```

Conversion to array for input to neural network

Use array function with all columns except first as data.

```
zip.size <- 16
zip.X.array <- array(
  data = unlist(zip.dt[1:nrow(zip.dt),-1]),
  dim = c(nrow(zip.dt), zip.size, zip.size, 1))
```

Need to specify dimensions of array:

- Observations: same as the number of rows in the CSV table.
- Pixels wide: 16.
- Pixels high: 16.
- Channels: 1 (greyscale image).

Linear model R code

```
library(keras)
linear.model <- keras::keras_model_sequential() %>%
  keras::layer_flatten(
    input_shape = c(16, 16, 1)) %>%
  keras::layer_dense(
    units = 10,
    activation = 'softmax')
```

- First layer must specify shape of inputs (here 16x16x1).
- `layer_flatten` converts any shape to a single dimension of units (here 256).
- `layer_dense` uses all units in the previous layer to predict each unit in the layer.
- `units=10` because there are ten possible classes for an output.
- `activation='softmax'` is required for the last/output layer in multi-class classification problems.

Keras model compilation

```
linear.model %>% keras::compile(
  loss = keras::loss_categorical_crossentropy,
  optimizer = keras::optimizer_adadelta(),
  metrics = c('accuracy')
)
```

In `compile` you can specify

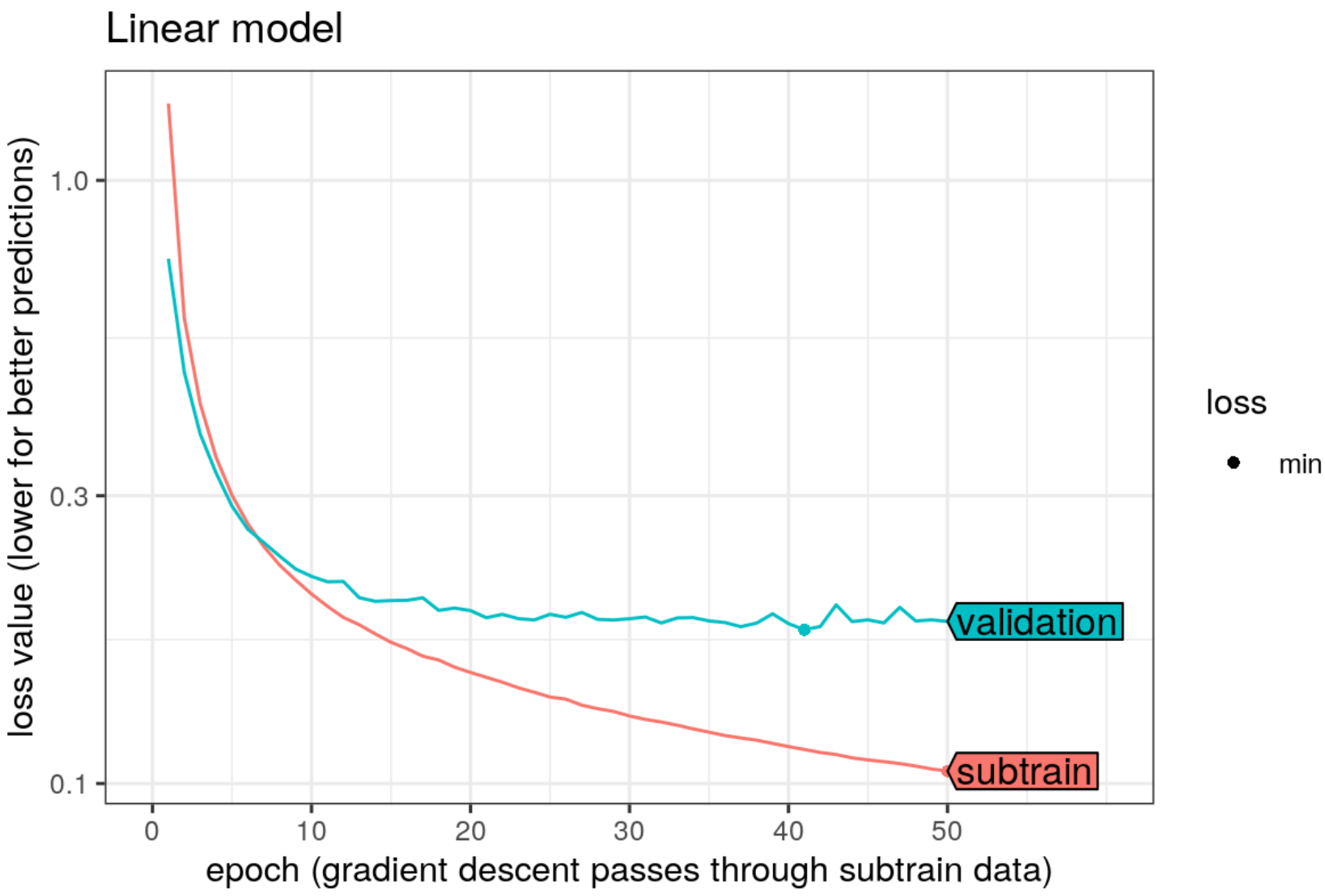
- a `loss` function, which is directly optimized/minimized in each iteration of the gradient descent learning algorithm. `https://keras.io/api/losses/`
- an `optimizer`, which is the version of gradient descent learning algorithm to use. `https://keras.io/api/optimizers/`
- an evaluation `metric` to monitor, not directly optimized via gradient descent, but usually more relevant/interpretable for the application (e.g. accuracy is the proportion of correctly predicted labels). `https://keras.io/api/metrics/`

Keras model fitting

```
linear.model %>% keras::fit(
  zip.X.array, zip.y.mat,
  epochs = 50,
  validation_split = 0.2
)
```

In `fit` you can specify

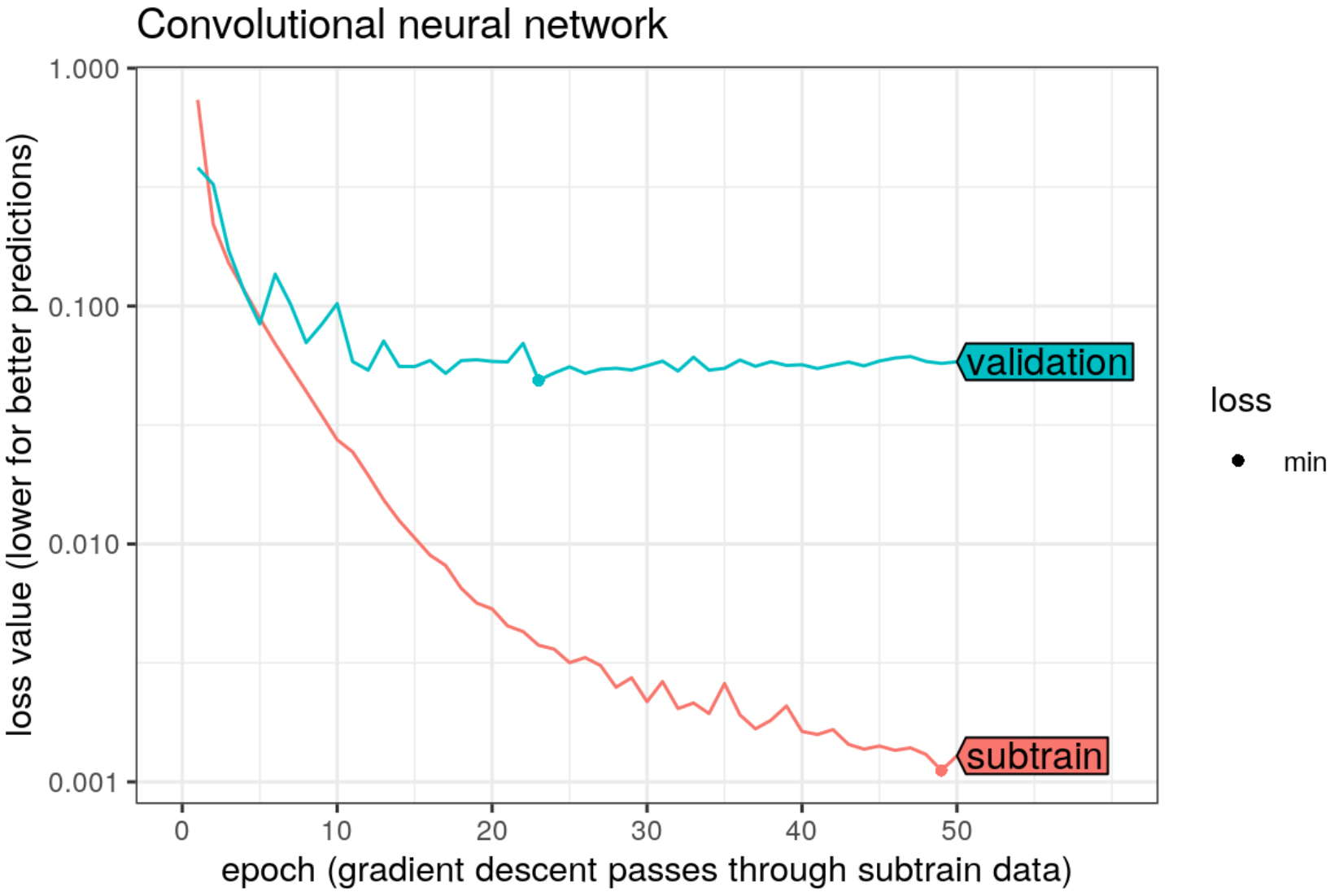
- Train data inputs `zip.X.array` and outputs `zip.y.mat` (required).
- Number of full passes of gradient descent through the subtrain data (`epochs`). In each epoch the gradient with respect to each subtrain observation is computed once.
- `validation_split=0.2` which means to use 80% subtrain (used for gradient descent parameter updates), 20% validation (used for hyper-parameter selection).


Linear model
loss value (lower for better predictions)
1.0
0.3
0.1
0
10
20
30
40
50
epoch (gradient descent passes through subtrain data)
validation
subtrain
loss
min

Sparse (convolutional) model R code

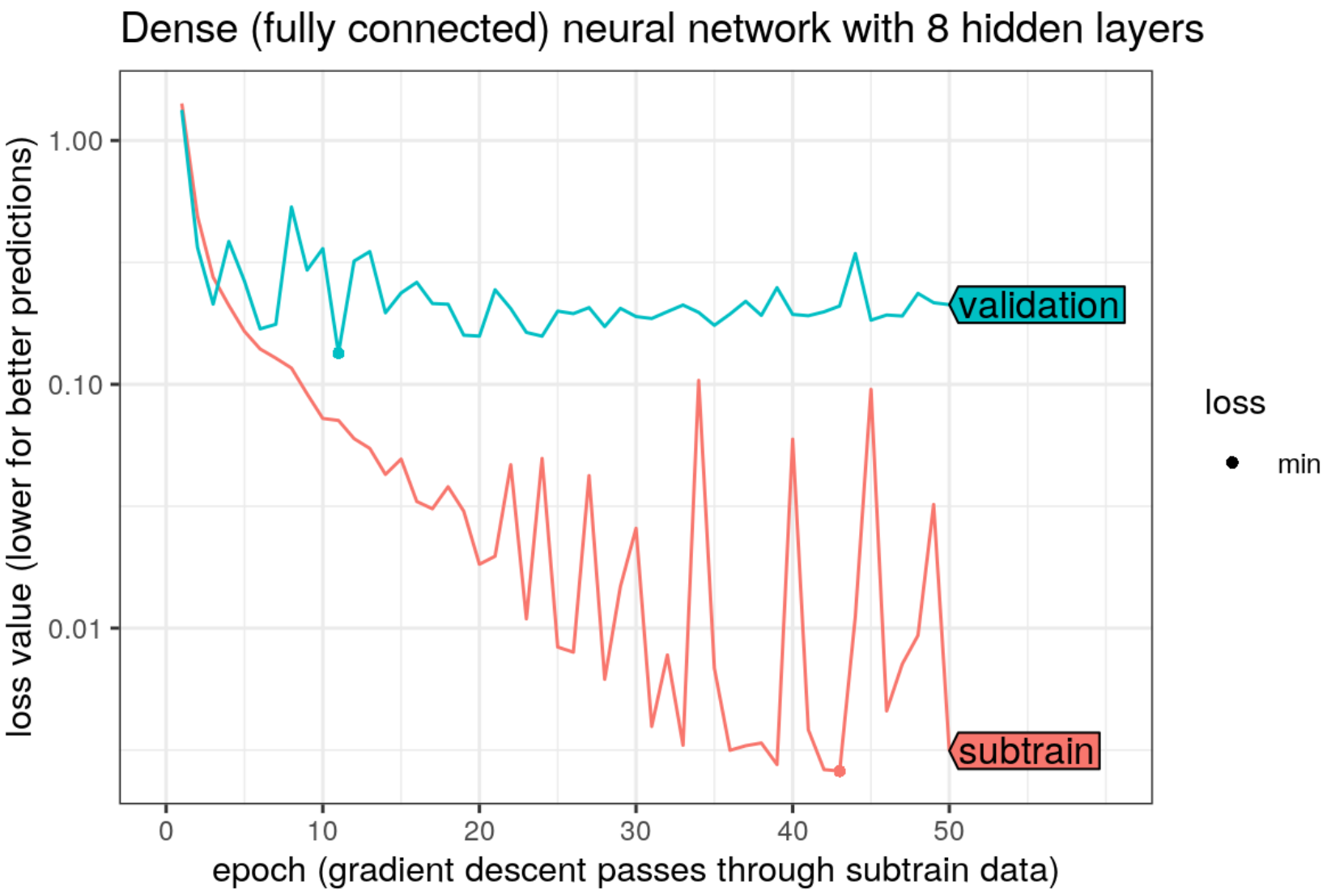
```
library(keras)
conv.model <- keras_model_sequential() %>%
  layer_conv_2d(
    input_shape = dim(zip.X.array)[-1],
    filters = 20,
    kernel_size = c(3,3),
    activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(
    units = ncol(zip.y.mat),
    activation = 'softmax')
```

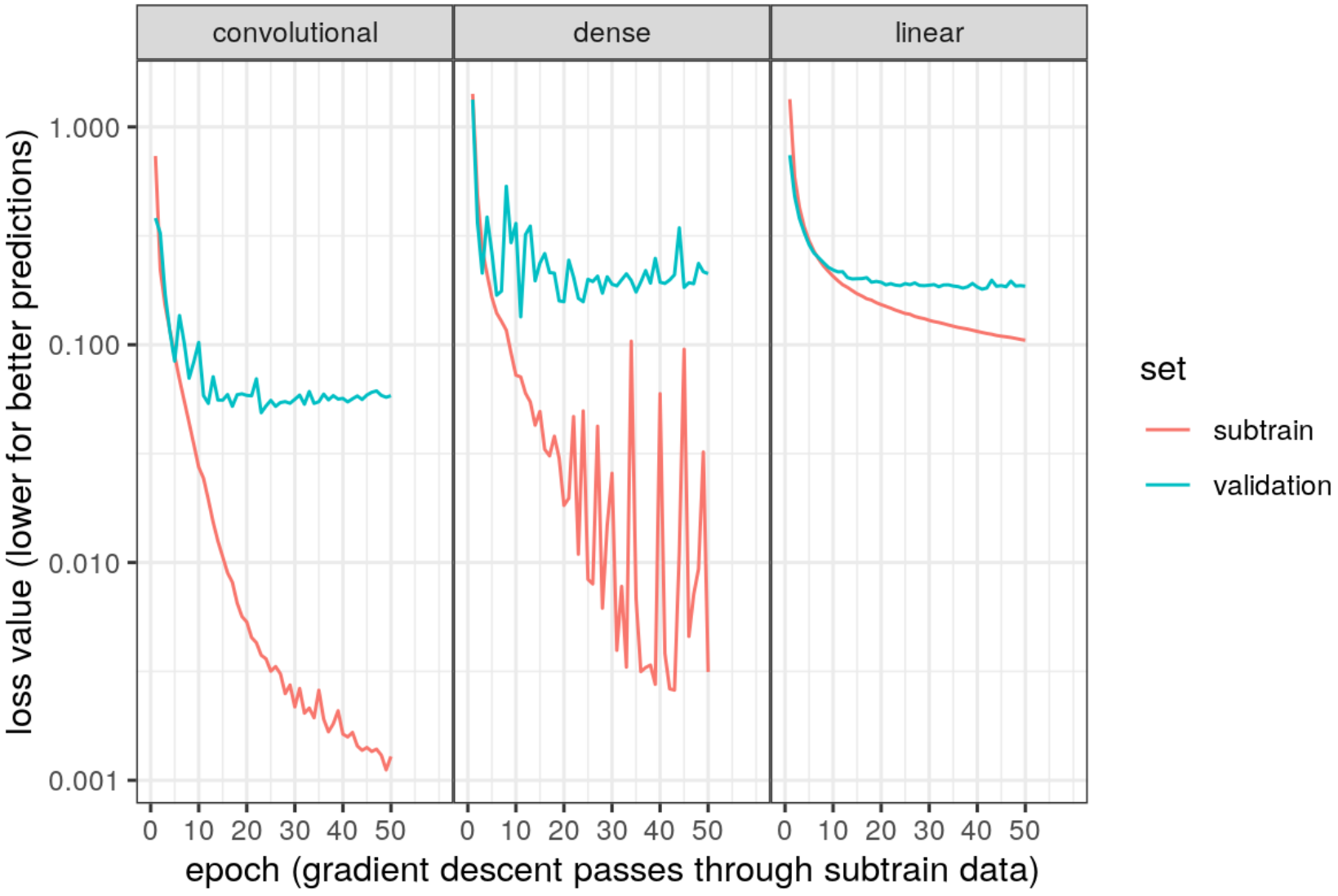
- Sparse: few inputs are used to predict each unit in `layer_conv_2d`.
- Exploits structure of image data to make learning easier/faster.


Convolutional neural network
1.000
0.100
0.010
0.001
loss value (lower for better predictions)
0
10
20
30
40
50
epoch (gradient descent passes through subtrain data)
validation
subtrain
loss
min

Dense (fully connected) neural network R code

```
library(keras)
dense.model <- keras_model_sequential() %>%
  layer_flatten(
    input_shape = dim(zip.X.array)[-1]) %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(
    units = ncol(zip.y.mat),
    activation = 'softmax')
```


Dense (fully connected) neural network with 8 hidden layers
loss value (lower for better predictions)
1.00
0.10
0.01
validation
subtrain
0
10
20
30
40
50
epoch (gradient descent passes through subtrain data)
loss
min


convolutional
dense
linear
loss value (lower for better predictions)
1.000
0.100
0.010
0.001
0 10 20 30 40 50
epoch (gradient descent passes through subtrain data)
set
subtrain
validation

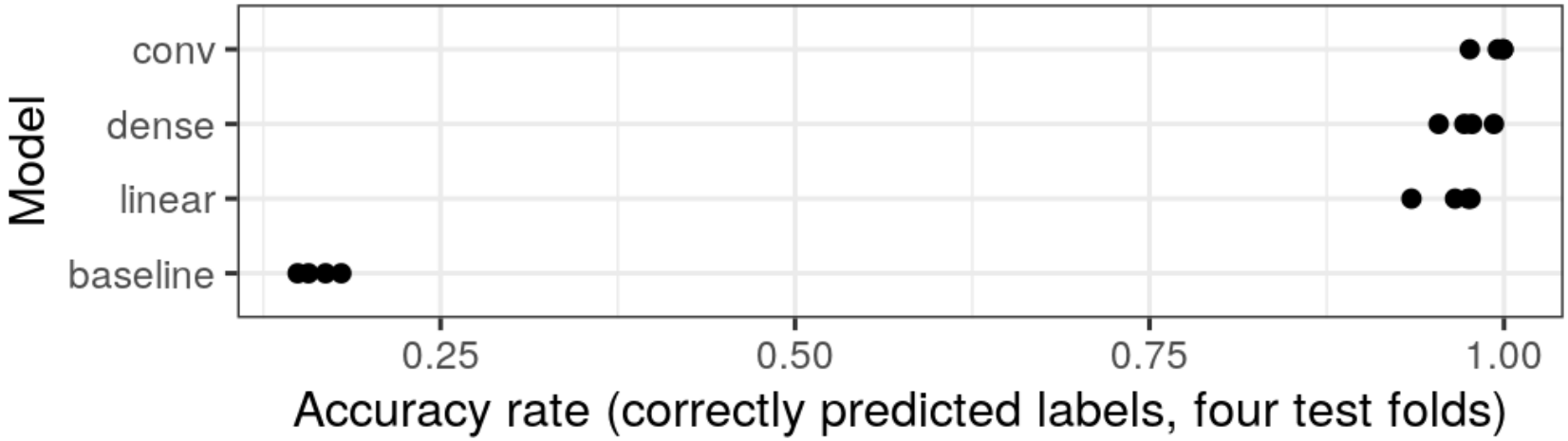
4-fold cross-validation for model evaluation

Does the convolutional model provide more accurate predictions on unseen test data? First randomly assign a fold ID to each image/observation, then for each test fold ID from 1 to 4:

- Hold out the images/observations with the test fold ID as a test set (not used at all for model training).
- Use the other images/observations to train all three models using `validation_split=0.2`.
- Plot the validation loss curve as a function of the number of epochs, and select the number of epochs which minimizes the validation loss (hyper-parameter learning).
- Re-train with `epochs`=the learned number of epochs and `validation_split=0` (all train data used for gradient descent parameter updates).
- Finally compute the prediction accuracy with respect to the held-out test set.

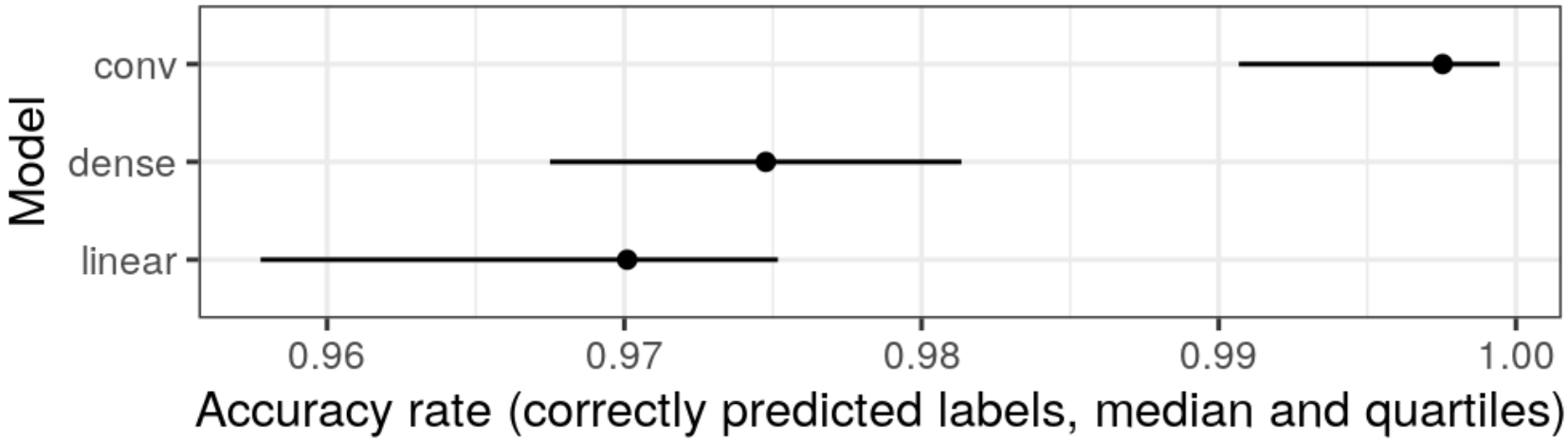
Plot accuracy values (or mean/sd) after learning/testing four times (one for each test fold).

Accuracy rates for each test fold



- Always a good idea to compare with the trivial baseline model which always predicts the most frequent class in the train set. (ignoring all inputs/features)
- Here we see that the baseline is much less accurate than the three learned models, so they are clearly learning something non-trivial.
- Code for test accuracy figures: `https://github.com/tdhock/2020-yiqi-summer-school/blob/master/figure-test-accuracy.R`

Zoom to learned models



- Dense neural network slightly more accurate than linear model, convoluational significantly more accurate than others.
- Conclusion: convolutional neural network should be preferred for most accurate predictions in these data.
- Maybe not the same conclusion in other data sets, with the same models. (always need to do cross-validation experiments to see which model is best in any given data set)
- Maybe other models/algorithms would be even more accurate in these data. (more/less layers, more/less units, completely different algorithm such as random forests, boosting, etc)