

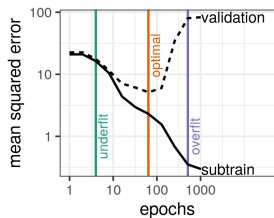
# Introduction to deep learning in R

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

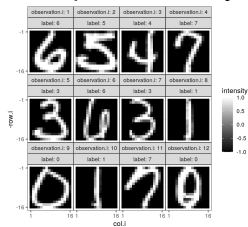
April 15, 2023

# Introduction and overview

## Example 1: avoiding overfitting in regression, overview of concepts



## Example 2: classifying images of digits, coding demos




## Summary and quiz questions

# Machine learning intro: image classification example

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

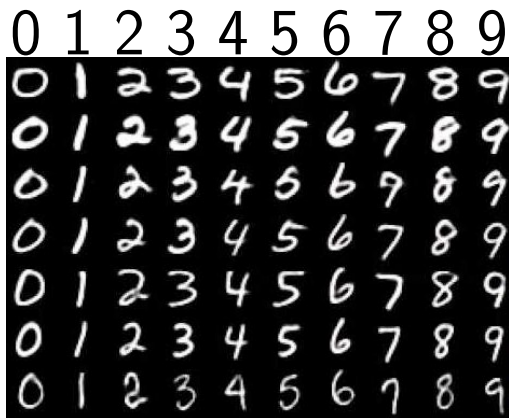
- ▶ Inputs/features  $x$  can be easily computed using traditional algorithms. For example, matrix of pixel intensities in an image.
- ▶ Outputs/labels  $y$  are what we want to predict, typically more difficult/costly to measure than inputs. For example, to get an image class label, you may have to ask a human.
- ▶ Input  $x$  = image of digit, output  $y \in \{0, 1, \dots, 9\}$ ,
  - this is a classification problem with 10 classes.

  $f(\text{image of } 0) = 0$ ,  $f(\text{image of } 1) = 1$

- ▶ Traditional/unsupervised algorithm: I give you a pixel intensity matrix  $x \in \mathbb{R}^{28 \times 28}$ , 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, `LEARN`, that infers a function  $f$  from the training data. (you don't code  $f$ )

Source: [github.com/cazala/mnist](https://github.com/cazala/mnist)

# Advantages of supervised machine learning

Learning Algorithm	Train data	Learned function	Predictions on test data
--------------------	------------	------------------	--------------------------

Learn(



)  $\rightarrow g$

$g(\text{0}) = 0$

$g(\text{1}) = 1$

$g(\text{1}) = 1$

Learn(



)  $\rightarrow h$

$h(\text{0}) = 0$

$h(\text{0}) = 0$

$h(\text{1}) = 1$

- ▶ Input  $x \in \mathbb{R}^{28 \times 28}$ , output  $y \in \{0, 1, \dots, 9\}$  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](https://github.com/cazala/mnist), [github.com/zalando-research/fashion-mnist](https://github.com/zalando-research/fashion-mnist)

# Overview of tutorial

In this tutorial we will discuss two kinds of problems, which differ by the type of the output/label/ $y$  variable we want to predict.

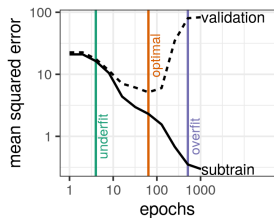
- ▶ Regression,  $y$  is a real number.
- ▶ Classification,  $y$  is an integer representing a category.

The rest of the tutorial will focus on three examples:

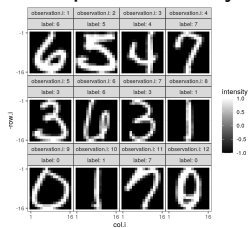
1. Regression with a single input, to demonstrate how to avoid overfitting.
2. Classification of digit images, to demonstrate how to compare machine learning algorithms in terms of test/prediction accuracy.

# Introduction and overview

## Example 1: avoiding overfitting in regression, overview of concepts



## Example 2: classifying images of digits, coding demos



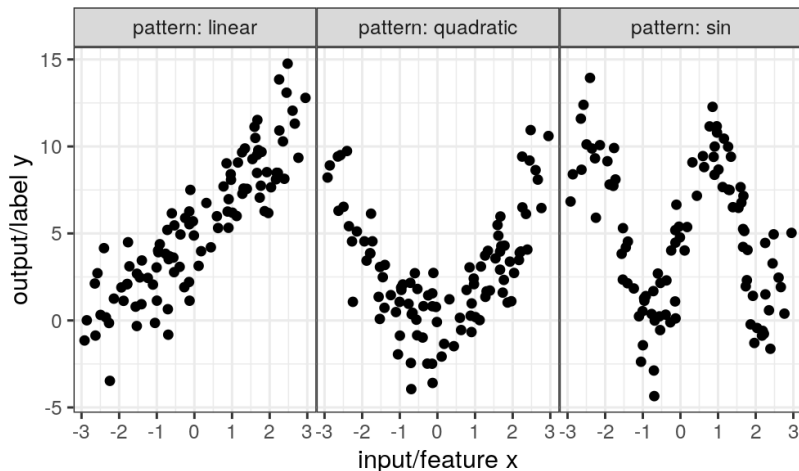
## Summary and quiz questions

## Goal of this section: demonstrate how to avoid overfitting

- ▶ The goal of supervised machine learning is to get accurate predictions on new/unseen/held-out test data.
- ▶ The data used during learning are called the train set.
- ▶ Any machine learning algorithm is prone to overfit, which means providing better predictions on the train set than on a held-out validation/test set. (BAD)
- ▶ To learn a model which does NOT overfit (GOOD), you need to divide your train set into subtrain/validation sets (subtrain used as input to gradient descent algorithm, validation set used to control number of iterations of gradient descent).
- ▶ Code for figures in this section:  
<https://github.com/tdhock/2023-res-baz-az/blob/main/figure-overfitting.R>



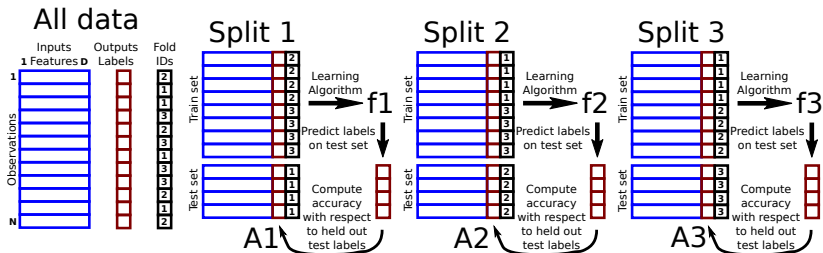
# Three different data sets/patterns



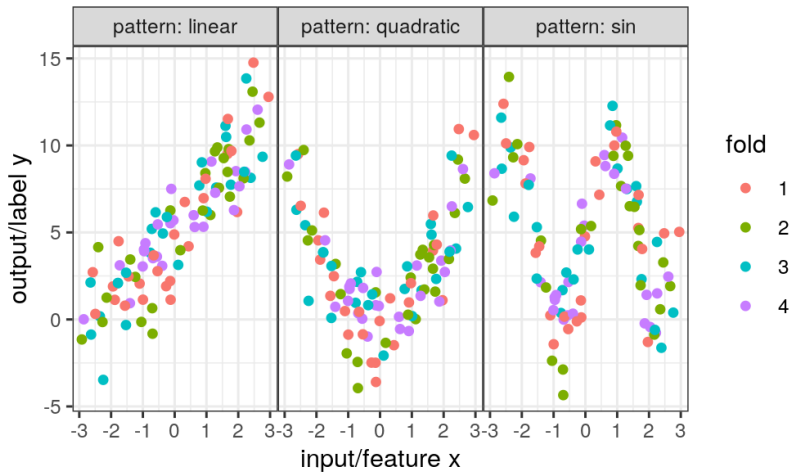
- ▶ We illustrate this using a single input/feature  $x \in \mathbb{R}$ .
- ▶ We use a regression problem with outputs  $y \in \mathbb{R}$ .
- ▶ Goal is to learn a function  $f(x) \in \mathbb{R}$ .

# K-fold cross-validation for splitting data

- ▶ One way to split is via K-fold cross-validation.
- ▶ Each row is assigned a fold ID number from 1 to K.
- ▶ For each for ID, those data are held out, and other data are kept.
- ▶ Popular relative to other splitting methods because of simplicity and fairness (each row is held out one time).



# Illustration of 4-fold cross-validation

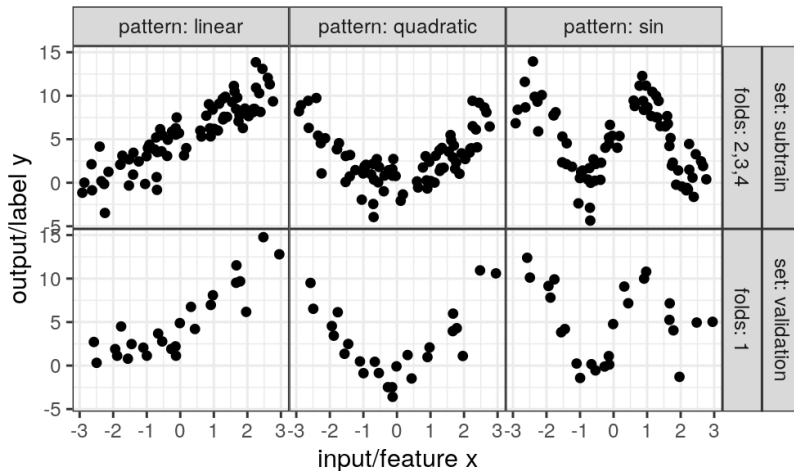


Randomly assign each observation a fold ID from 1 to 4.

# Neural network learning algorithm

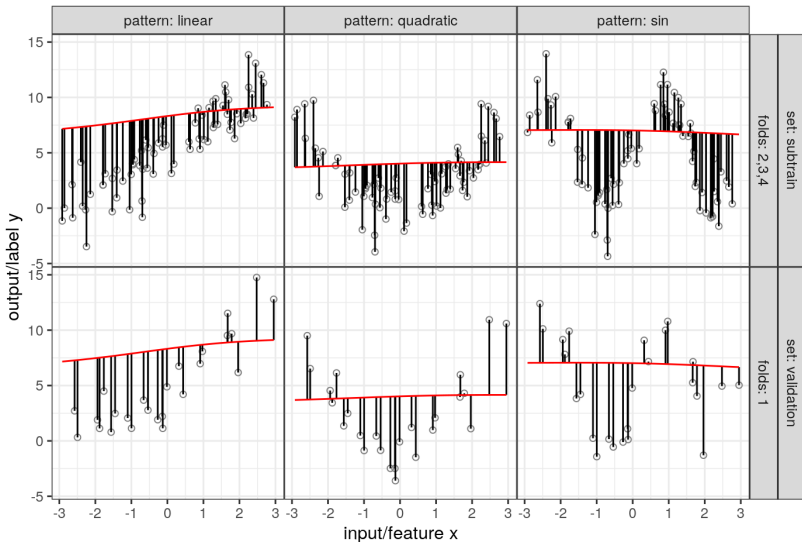
- ▶ We will fit a neural network to these data.
- ▶ The neural network learns how to predict the outputs from the inputs.
- ▶ The learning algorithm is gradient descent, which iteratively minimizes the loss of the predictions with respect to the labels in the subtrain set.
- ▶ We also compute the loss on the validation set, so we can select the number of gradient descent iterations that gives the best predictions on new data (avoiding overfitting).

# Illustration of subtrain/validation split



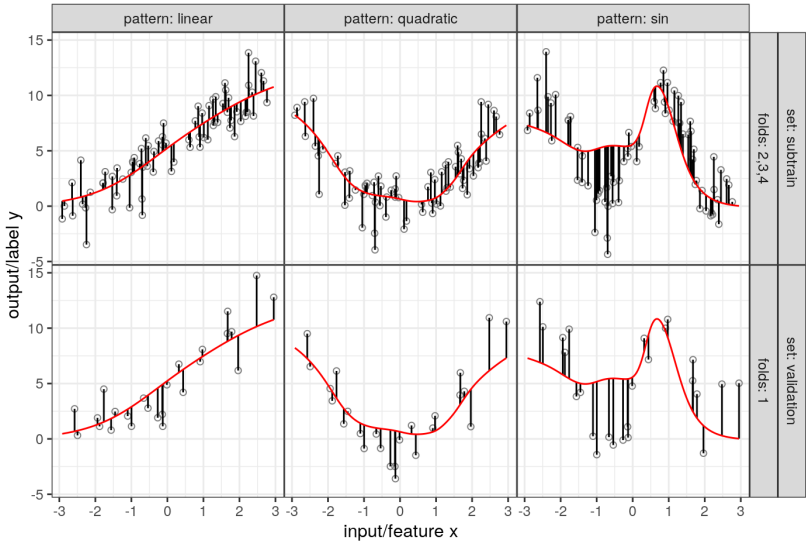
- ▶ For validation fold 1, all observations with that fold ID are considered the validation set.
- ▶ All other observations are considered the subtrain set.

## Neural network, 20 hidden units, 1 gradient descent iterations



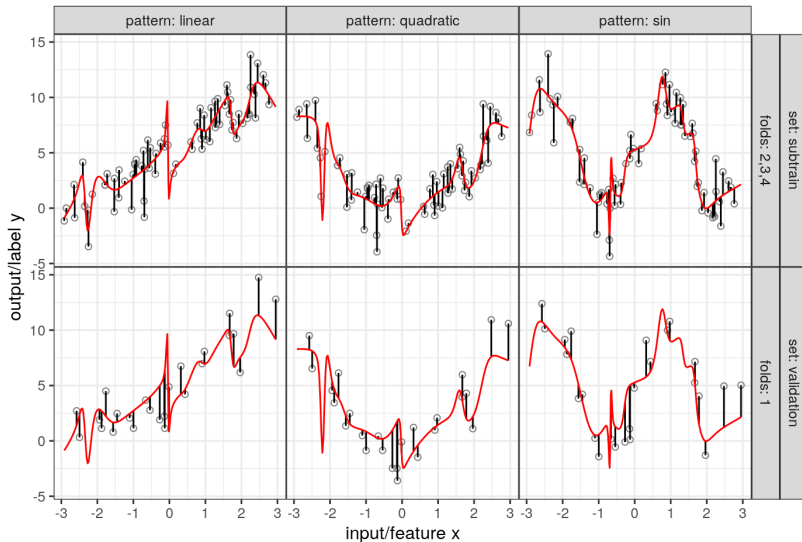
Data=grey dots, predictions=red curve, loss=black line segments.

## Neural network, 20 hidden units, 10 gradient descent iterations



Data=grey dots, predictions=red curve, loss=black line segments.

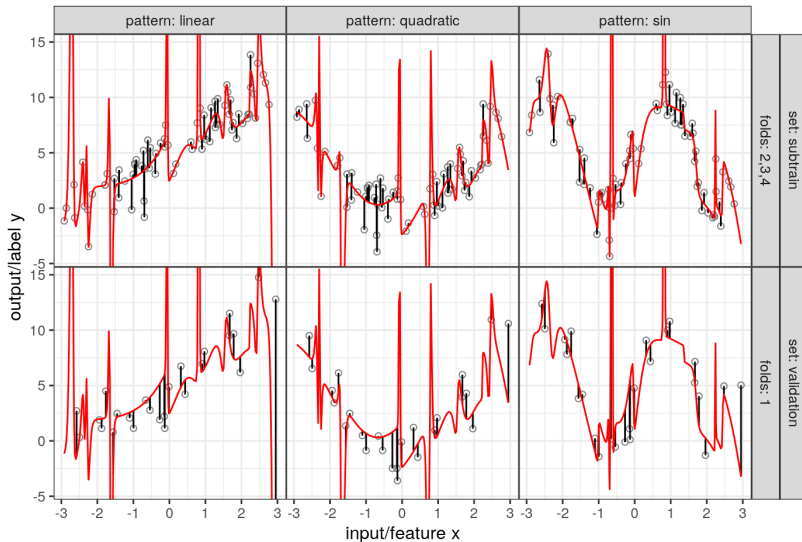
Neural network, 20 hidden units, 100 gradient descent iterations



Data=grey dots, predictions=red curve, loss=black line segments.

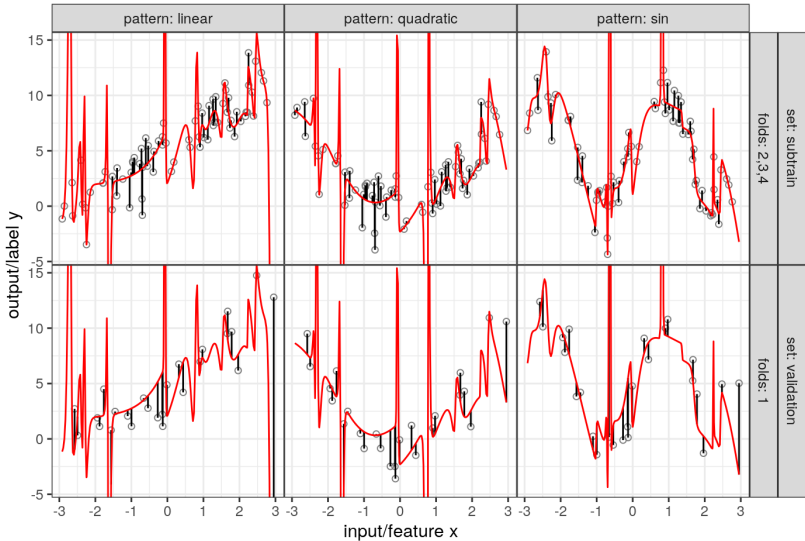


# Neural network, 20 hidden units, 1000 gradient descent iterations



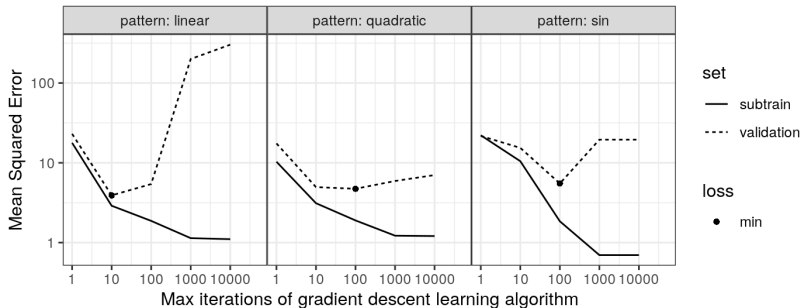
Data=grey dots, predictions=red curve, loss=black line segments.

Neural network, 20 hidden units, 10000 gradient descent iterations



Data=grey dots, predictions=red curve, loss=black line segments.

## Neural network, 20 hidden units



Different number of iterations best for different data.

# Neural network prediction function

For an input feature vector  $\mathbf{x} \in \mathbb{R}^{u_1}$ , the prediction function for a neural network with  $L$  layers (functions to learn) is:

$$f(\mathbf{x}) = f_L[\cdots f_1[\mathbf{x}]]. \quad (1)$$

We have for all  $l \in \{1, \dots, L\}$ :

$$f_l(t) = A_l(\mathbf{W}_l^T t), \quad (2)$$

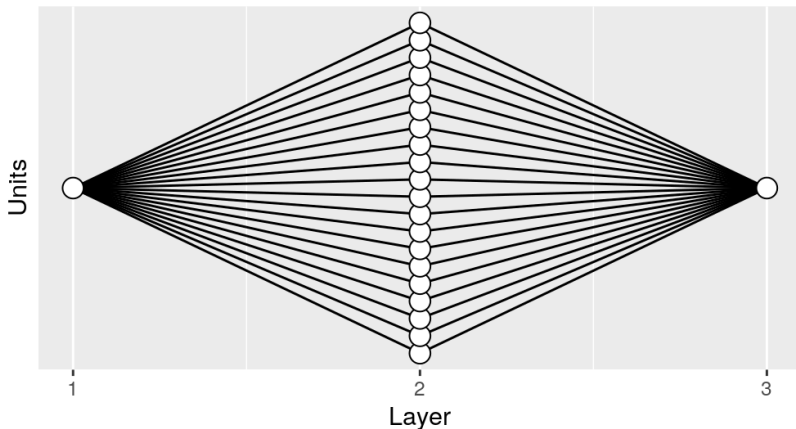
The hyper-parameters which must be fixed prior to learning:

- ▶ Number of functions to learn  $L$ .
- ▶ Activation functions  $A_l$  (classically sigmoid, typically ReLU).
- ▶ Number of hidden units per layer  $(u_1, \dots, u_{L-1})$ .
- ▶ Sparsity pattern in the weight matrices  $\mathbf{W}_l \in \mathbb{R}^{u_l \times u_{l-1}}$ .

## Network for 1 input, 1 output, 1 hidden layer

Neural network diagrams show how each unit (node) is computed by applying the weights (edges) to the values of the units at the previous layer.

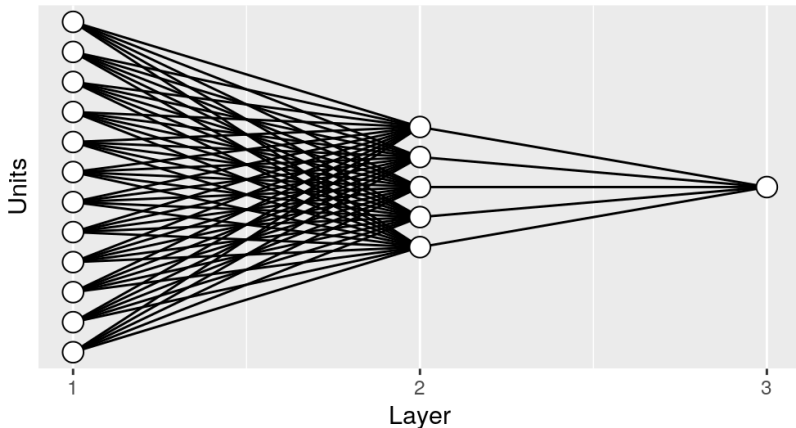
Number of units: 1,20,1



## Network for 12 inputs, 1 output, 1 hidden layer

Neural network diagrams show how each unit (node) is computed by applying the weights (edges) to the values of the units at the previous layer.

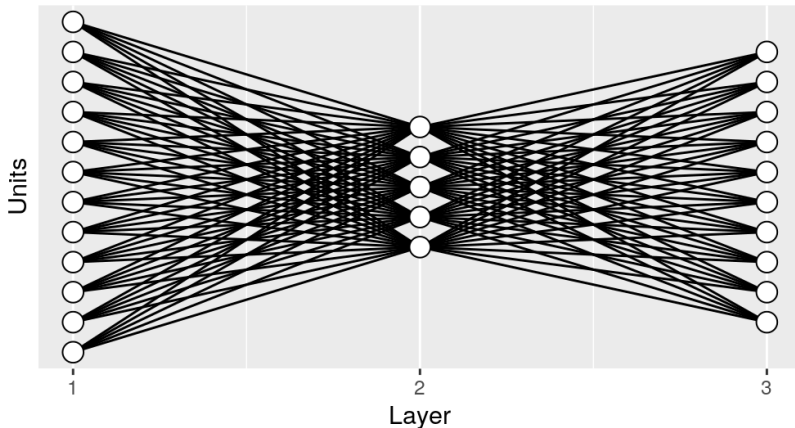
Number of units: 12,5,1



## Network for 12 inputs, 10 outputs, 1 hidden layer

Neural network diagrams show how each unit (node) is computed by applying the weights (edges) to the values of the units at the previous layer.

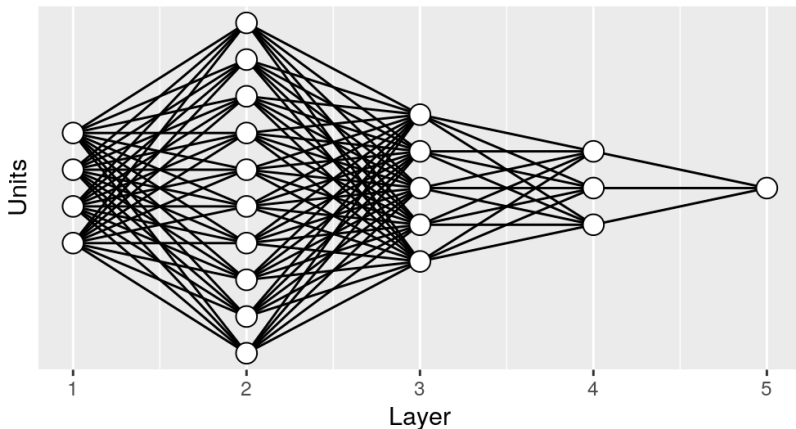
Number of units: 12,5,10



## Network for 4 inputs, 1 output, 3 hidden layers

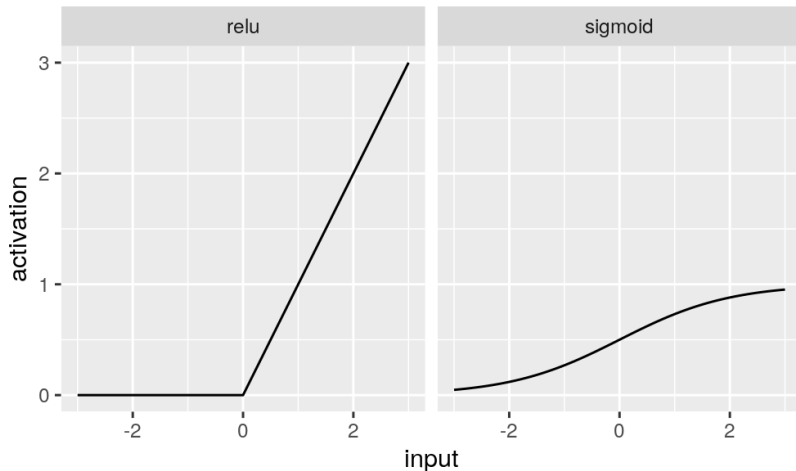
Neural network diagrams show how each unit (node) is computed by applying the weights (edges) to the values of the units at the previous layer.

Number of units: 4,10,5,3,1





## Non-linear activation functions $A_l$



Each layer except the last should have a activation function  $A_l$  which is not linear (last layer activation should be identity/linear).

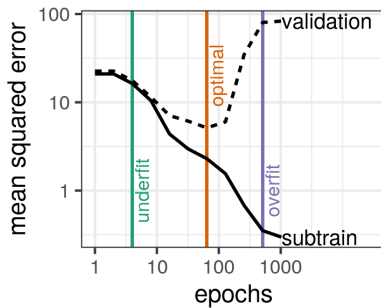
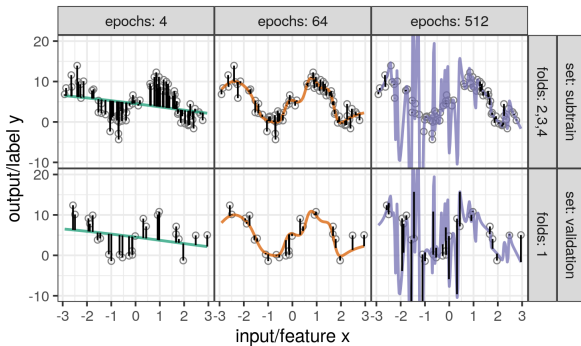
# Gradient Descent Learning

The neural network prediction function  $f(\mathbf{x}) = f_L[\cdots f_1[\mathbf{x}]]$  has  $l \in \{1, \dots, L\}$  component functions to learn:

$$f_l(t) = A_l(\mathbf{W}_l^\top t), \quad (3)$$

The weight matrices  $\mathbf{W}_l \in \mathbb{R}^{u_l \times u_{l-1}}$  are learned using gradient descent.

- ▶ A loss function  $\mathcal{L}[f(\mathbf{x}), y]$  computes how bad are predictions with respect to labels  $y$  (ex: mean squared error for regression, cross entropy loss for classification).
- ▶ In each **iteration** of gradient descent, the weights are updated in order to get better predictions on subtrain data.
- ▶ An **epoch** computes gradients on all subtrain data; there can be from 1 to  $N(\text{subtrain size})$  iterations per epoch.

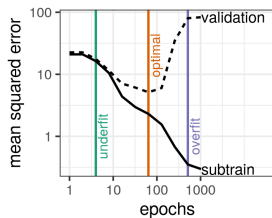


# Summary of how to avoid overfitting

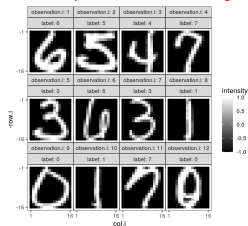
- ▶ Happens when subtrain error/loss decreases but validation error increases (as a function of some hyper-parameter)
- ▶ Here the hyper-parameter is the number of iterations of gradient descent, and overfitting starts after a certain number of iterations.
- ▶ To maximize prediction accuracy you need to choose a hyper-parameter with minimal validation error/loss.
- ▶ This optimal hyper-parameter will depend on the data set.
- ▶ To get optimal prediction accuracy in any machine learning analysis, you always need to do this, because you never know the best hyper-parameters in advance.

# Introduction and overview

## Example 1: avoiding overfitting in regression, overview of concepts



## Example 2: classifying images of digits, coding demos



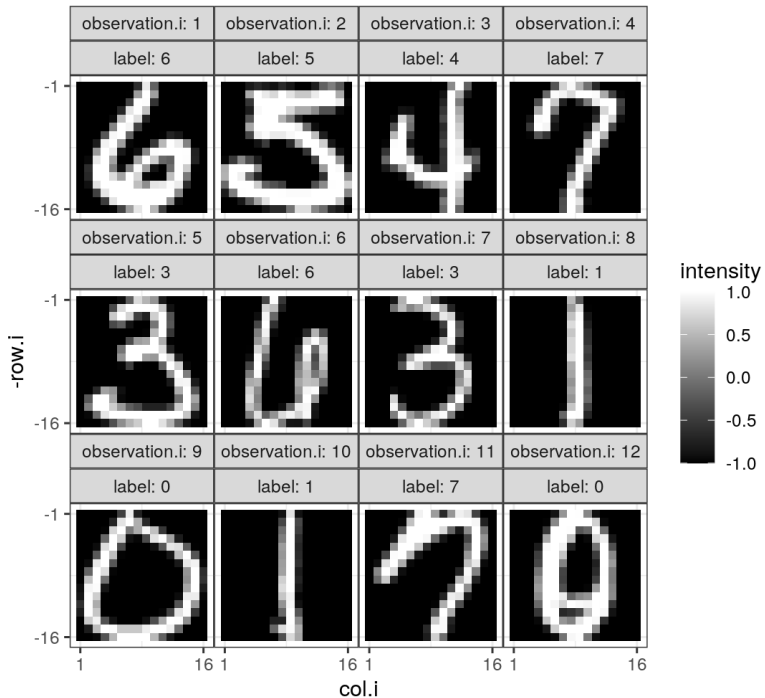
## Summary and quiz questions

# Image classification

- ▶ A problem in computer vision, one of the most popular/successful application domains 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  $y \in \{0, 1, \dots, 9\}$ , one for each digit.



- ▶ Want to learn  $f$  such that  $f(\text{0}) = 0$ ,  $f(\text{1}) = 1$ , etc.
- ▶ Code for figures in this section:  
<https://github.com/tdhock/2023-res-baz-az/blob/main/figure-validation-loss.R>



# 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).

Data from

<https://web.stanford.edu/~hastie/ElemStatLearn/datasets/zip.train.gz>

```
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
...
```

Demo: reading CSV, plotting digits,

<https://github.com/tdhock/2023-res-baz-az/blob/main/2023-04-19-deep-learning.Rmd>



# Converting R data to torch tensors

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/>

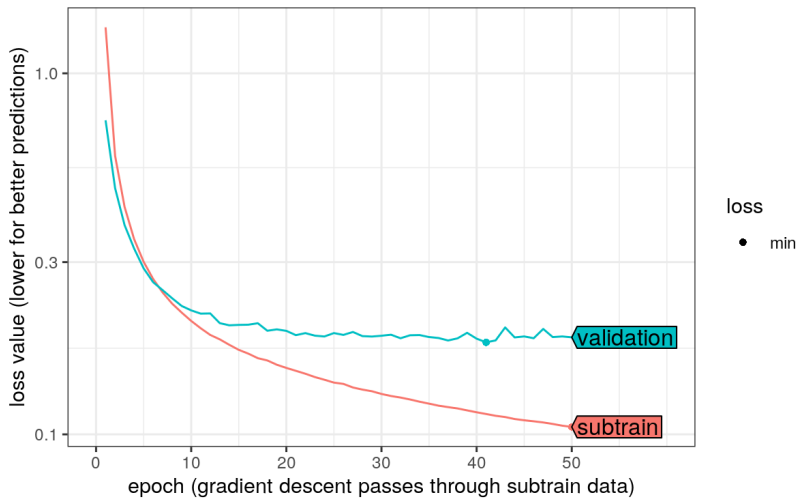
# Keras model fitting

```
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

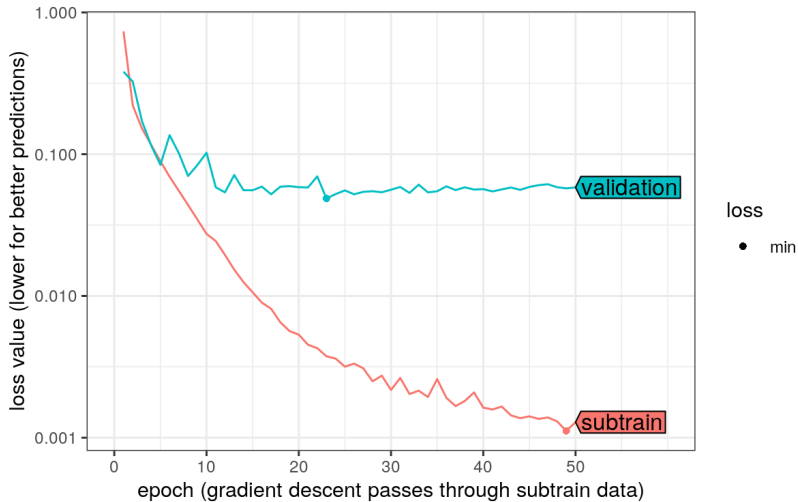


## Sparse (convolutional) model R code

```
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

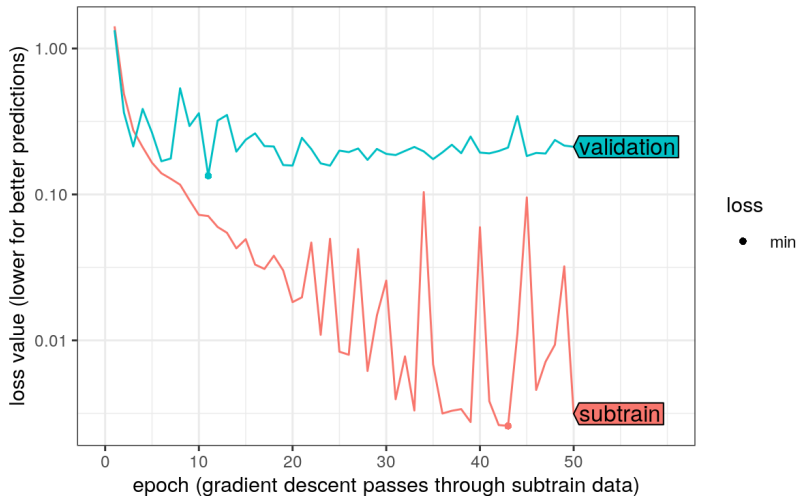


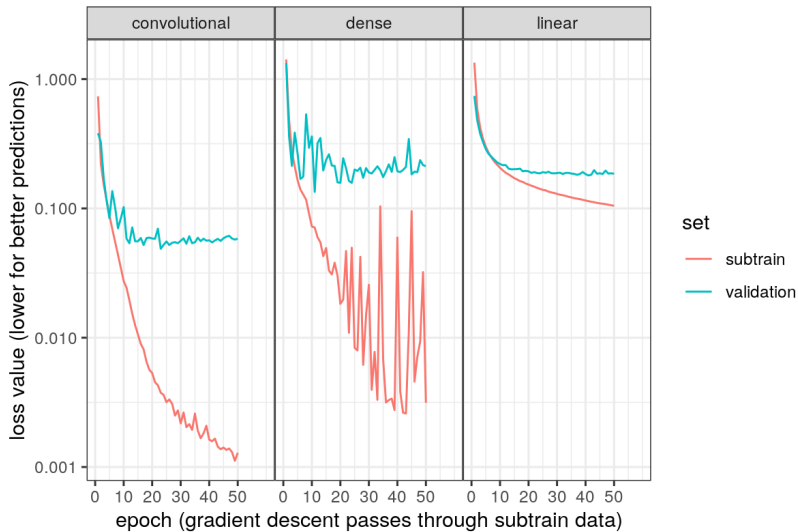
## Dense (fully connected) neural network R code

```
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 = ncol(zip.y.mat),
    activation = 'softmax')
```



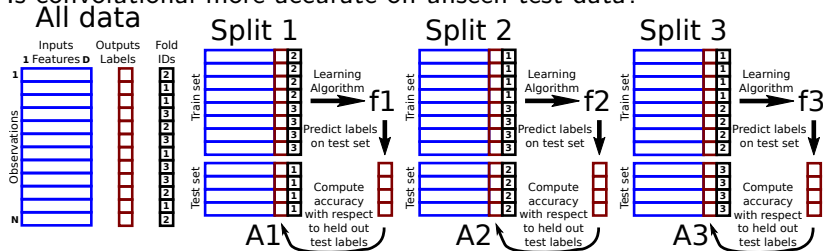
## Dense (fully connected) neural network with 8 hidden layers





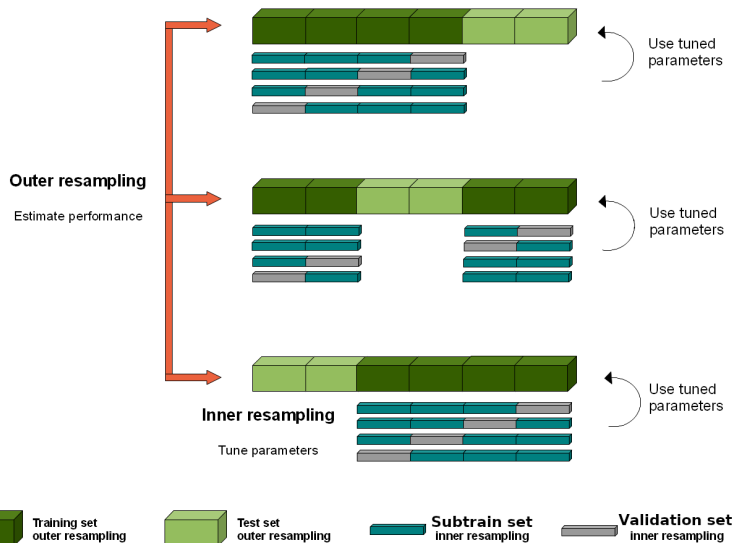
# K-fold cross-validation for model evaluation

Is convolutional more accurate on unseen test data?



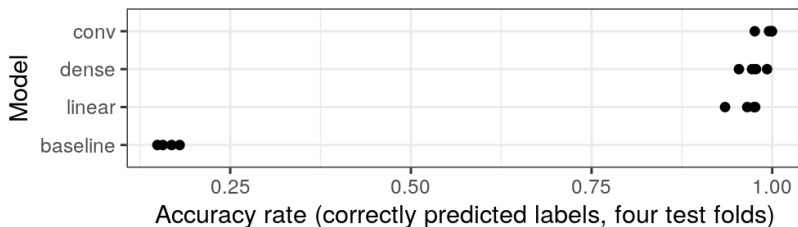
- ▶ Randomly assign a fold ID from 1 to K to each observation.
- ▶ Hold out the observations with the Split ID as test set.
- ▶ Use the other observations as the train set.
- ▶ Run learning algorithm on train set (including hyper-parameter selection), outputs learned function (f1-f3).
- ▶ Finally compute and plot the prediction accuracy (A1-A3) with respect to the held-out test set.

# Two kinds of cross-validation must be used



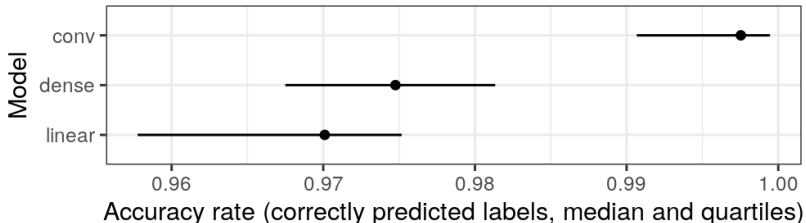
Source: [https://mlr.mlr-org.com/articles/tutorial/nested\\_resampling.html](https://mlr.mlr-org.com/articles/tutorial/nested_resampling.html)

# Accuracy rates for each test fold



- ▶ Always a good idea to compare with the trivial/featureless baseline model which always predicts the most frequent class in the train set. (ignoring all inputs/features)
- ▶ Here we see that the featureless baseline is much less accurate than the three learned models, which are clearly learning something non-trivial.
- ▶ Code for test accuracy figures:  
<https://github.com/tdhock/2023-res-baz-az/blob/main/figure-test-accuracy.R>

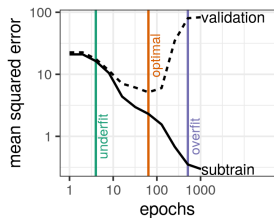
## Zoom to learned models



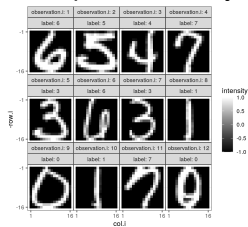
- ▶ Dense neural network slightly more accurate than linear model, convolutional significantly more accurate than others.
- ▶ Conclusion: convolutional neural network should be preferred for most accurate predictions in these data.
- ▶ Maybe not the same conclusion in other data sets, with the same models. (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)

# Introduction and overview

## Example 1: avoiding overfitting in regression, overview of concepts



## Example 2: classifying images of digits, coding demos



## Summary and quiz questions

# Summary

Thanks for participating! We have studied

- ▶ Two kinds of machine learning problems, regression  $y$ =real number, classification  $y$ =integer category.
- ▶ Splitting a data set into train/test/subtrain/validation sets for learning hyper-parameters and evaluating prediction accuracy.
- ▶ Overfitting and how to avoid it by choosing hyper-parameters based on a validation set.
- ▶ Comparing prediction accuracy of learning algorithms with each other and to a featureless baseline.



## Quiz questions

- ▶ When using a design matrix to represent machine learning inputs, what does each row and column represent?
- ▶ When splitting data into train/test sets, what is the purpose of each set? When splitting a train set into subtrain/validation sets, what is the purpose of each set?
- ▶ In order to determine if any non-trivial predictive relationship between inputs and output has been learned, a comparison with a featureless baseline that ignores the inputs must be used. How do you compute the baseline predictions, for regression and classification problems?
- ▶ How can you tell if machine learning model predictions are underfitting or overfitting?
- ▶ Many learning algorithms require input of the number of iterations or epochs. For example in R the `nnnet` function has the `maxit` argument and the `keras::fit` function has the `epochs` argument. How should this parameter be chosen?