Introduction to machine learning and neural networks

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

May 26, 2021

Introduction and overview

Problem 1: avoiding overfitting in regression

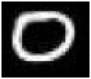
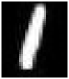
Problem 2: classifying images of digits

Problem 3: predicting earth system model parameters

Machine learning intro: image classification example

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

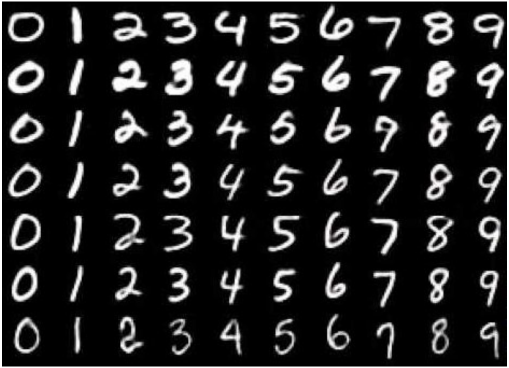
- Inputs/features x can be easily computed using traditional algorithms, e.g. matrix of pixel intensities in an image.
- Outputs/labels y are what we want to predict, easy to get by asking a human, but hard to compute using traditional algorithms, e.g. image class.
- Input $x =$ image of digit, output $y \in \{0, 1, \dots, 9\}$,
  – this is a classification problem with 10 classes.

  $f($$) = 0$, $f($$) = 1$
- Traditional/unsupervised algorithm: I give you a pixel intensity matrix $x \in \mathbb{R}^{28\times 28}$, 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.

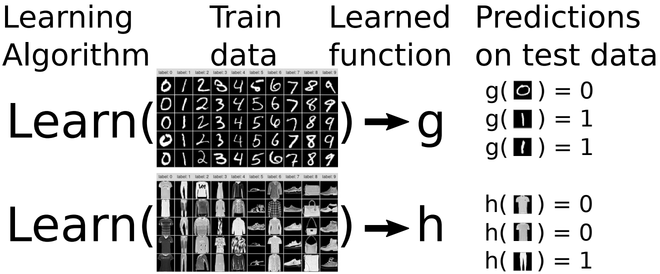
0 1 2 3 4 5 6 7 8 9



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

Advantages of supervised machine learning



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

Learn( ) → g

g( ) = 0
g( ) = 1
g( ) = 1

Learn( ) → h

h( ) = 0
h( ) = 0
h( ) = 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, github.com/zalandoresearch/fashion-mnist

Overview of tutorial

In this tutorial we will discuss two types of problems, which different 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 explain three learning problems:

- Regression with a single input, to demonstrate how to avoid overfitting.
- Classification of digit images, to demonstrate how to compare machine learning algorithms in terms of test/prediction accuracy.
- Regression for predicting earth system model parameters, as a relevant application.

Introduction and overview

Problem 1: avoiding overfitting in regression

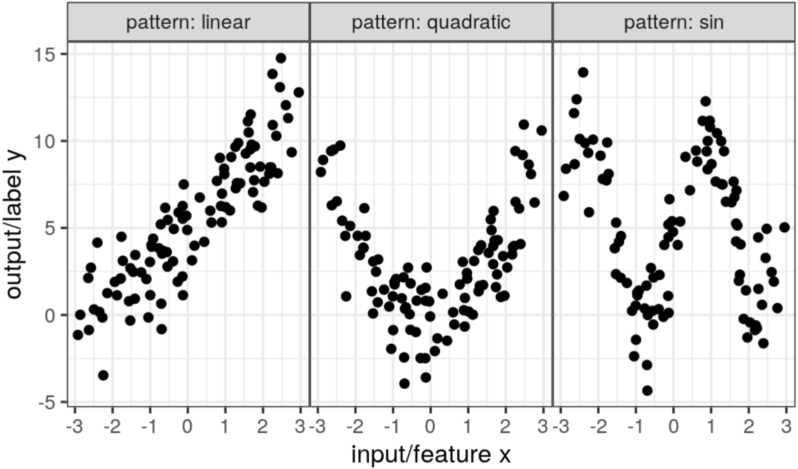
Problem 2: classifying images of digits

Problem 3: predicting earth system model parameters

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.
- 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 $f(x) \in \mathbb{R}$.

Neural network prediction function

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

With $L-1$ hidden layers, we have for all $l \in \{1, \dots, L\}$:

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

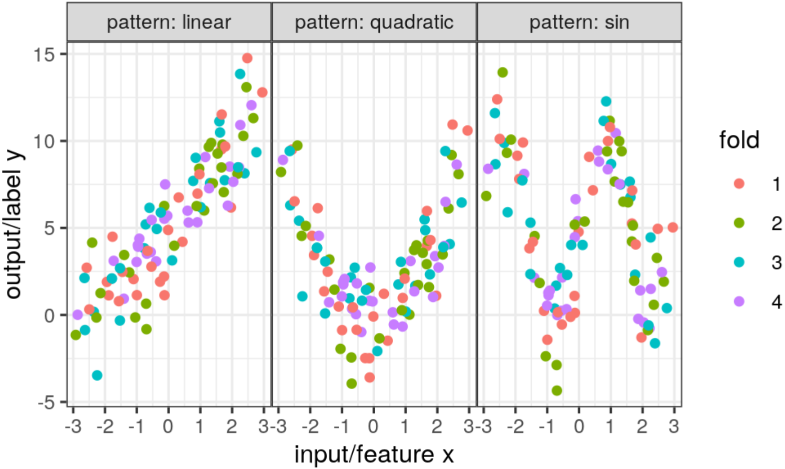
The hyper-parameters which must be fixed prior to learning:

- Number of layers 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 $\mathbf{W}_l \in \mathbb{R}^{u_l \times u_{l-1}}$.

The weight matrices $\mathbf{W}_l$ are learned using gradient descent.

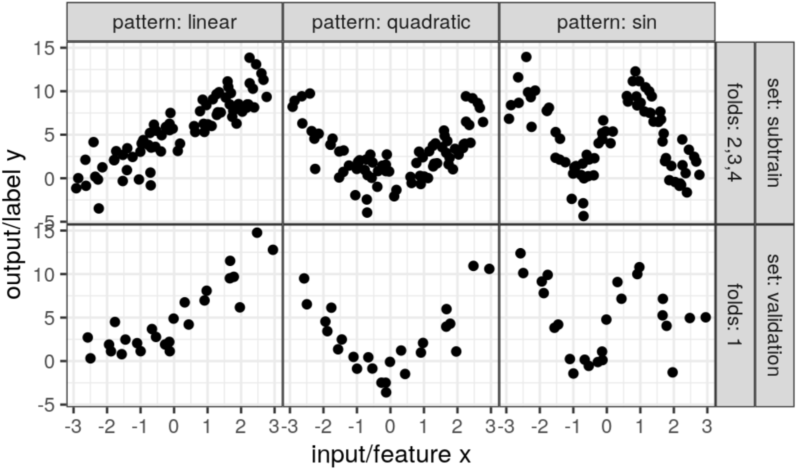
- In each **iteration** of gradient descent, the weights are updated in order to get better predictions on subtrain data.
- An **epoch** computes gradients on all subtrain data; there can be from 1 to N(subtrain size) iterations per epoch.

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.

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
...
```

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 prediction function

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

For the nnet code, we have:

$$\begin{aligned}
f_1(t) &= A_1(\mathbf{W}_1^\intercal t),\\
f_2(t) &= A_2(\mathbf{W}_2^\intercal t),\\
f(x) &= f_2[f_1(x)] = A_2[\mathbf{W}_2^\intercal A_1(\mathbf{W}_1^\intercal x)].
\end{aligned}$$

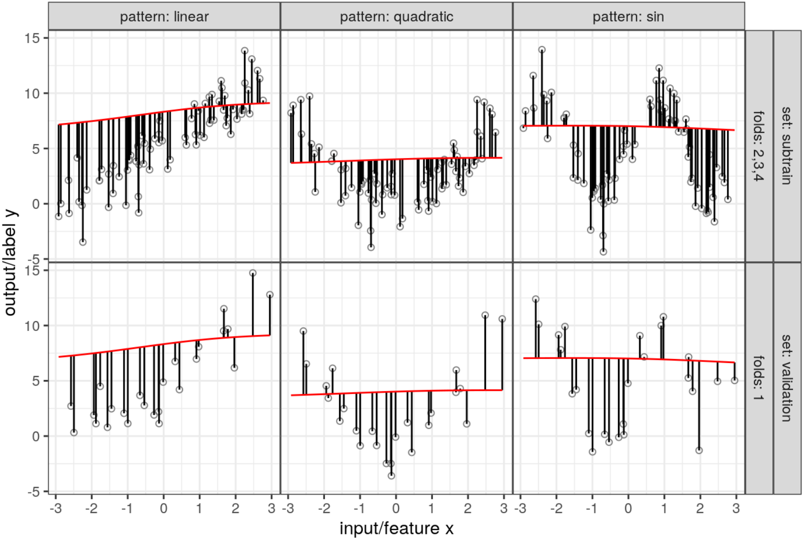
The hyper-parameters are fixed prior to learning:

- Number of layers $L = 2$.
- Activation functions A_1=sigmoid, A_2=identity.
- Number of units in the hidden layer $u_1 = 20$.
- No sparsity in the weight matrices (fully connected).

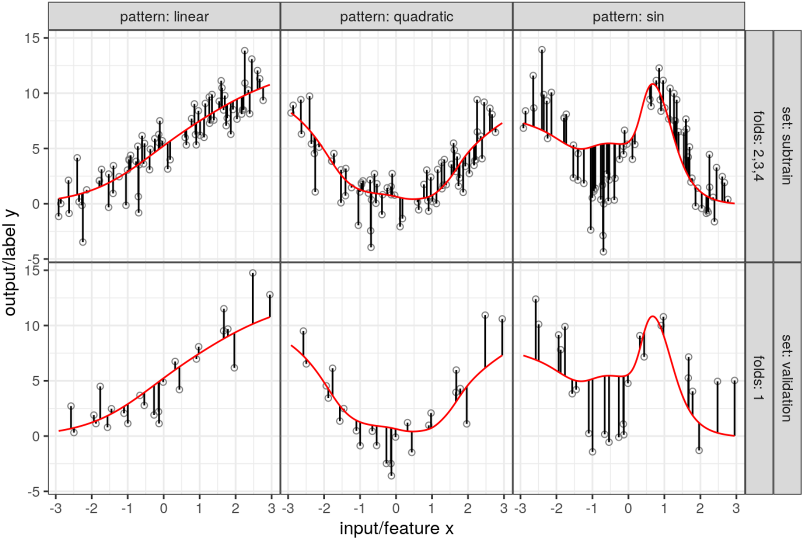
The weight matrices $\mathbf{W}_1, \mathbf{W}_2$ are learned using gradient descent.

- "Full gradient" method is used, so in each **epoch** there is 1 iteration/update to the weights that is based on the gradient summed over all subtrain data.

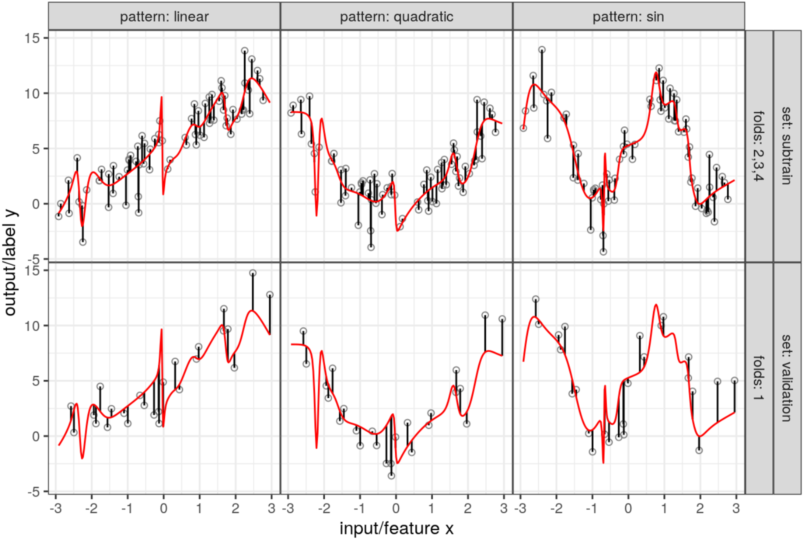
Neural network, 20 hidden units, 1 gradient descent iterations



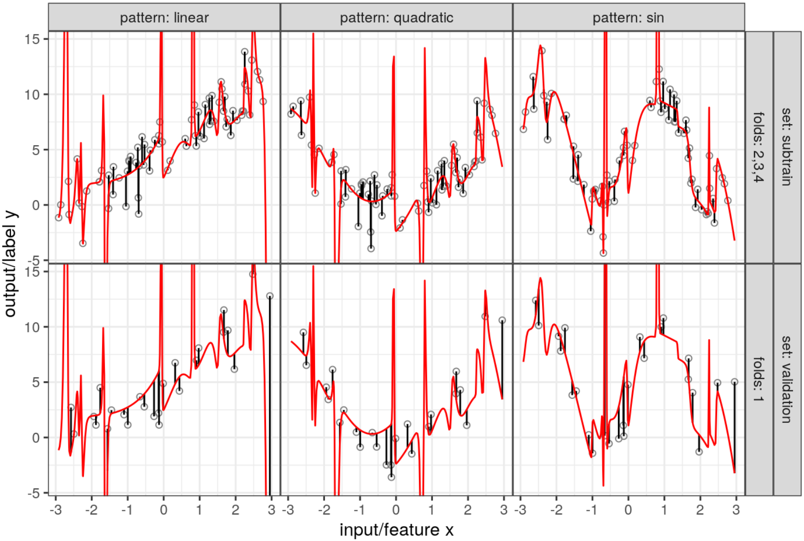
Neural network, 20 hidden units, 10 gradient descent iterations



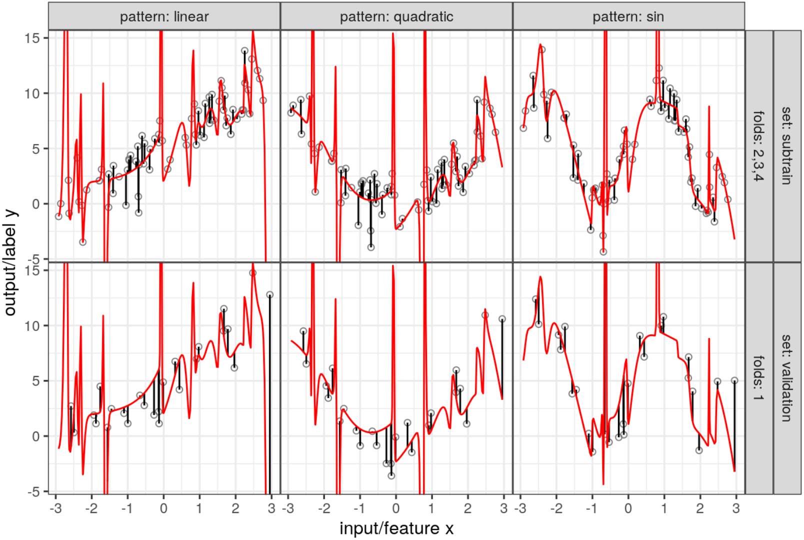
Neural network, 20 hidden units, 100 gradient descent iterations

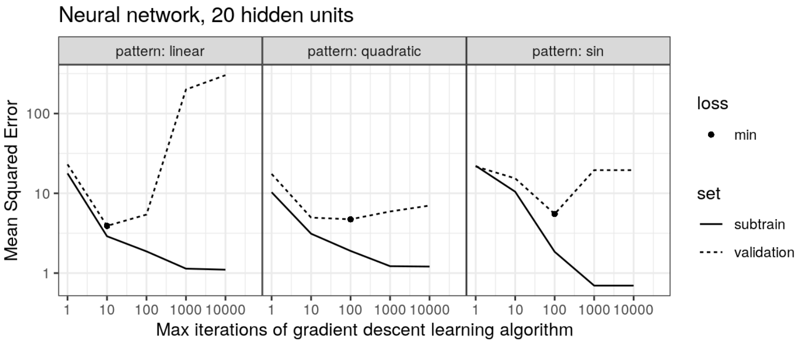


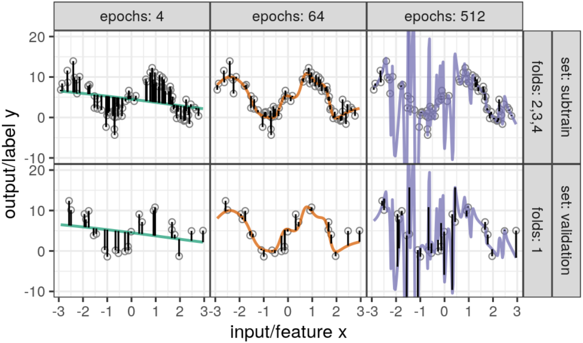
Neural network, 20 hidden units, 1000 gradient descent iterations

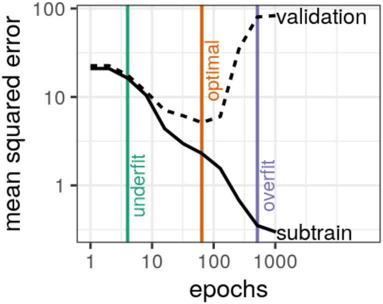


Neural network, 20 hidden units, 10000 gradient descent iterations




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
loss
min
set
subtrain
validation


epochs: 4
epochs: 64
epochs: 512
folds: 2,3,4
set: subtrain
folds: 1
set: validation
output/label y
20
10
0
-10
20
10
0
-10
input/feature x
-3
-2
-1
0
1
2
3


mean squared error
100
10
1
epochs
1
10
100
1000
validation
subtrain
underfit
optimal
overfit

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.

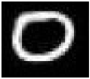
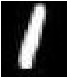
Introduction and overview

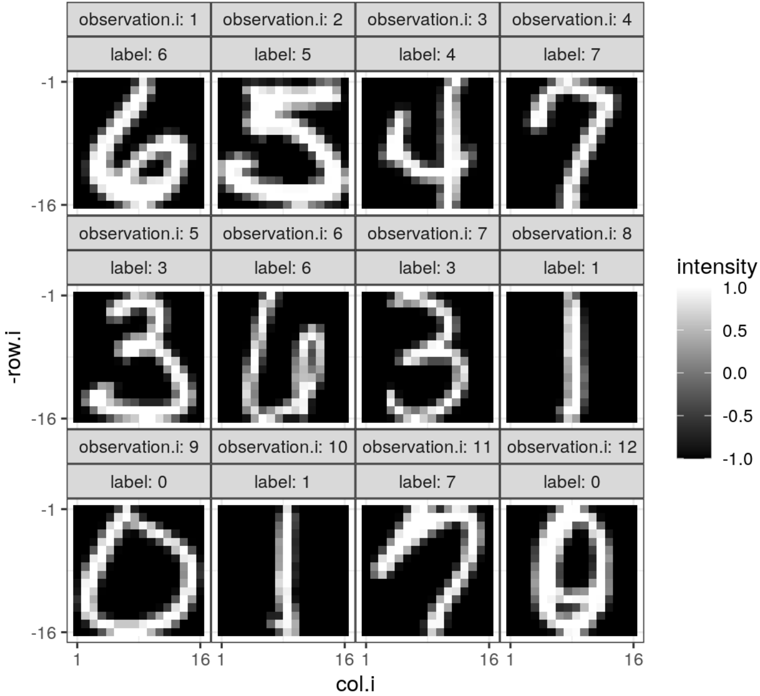
Problem 1: avoiding overfitting in regression

Problem 2: classifying images of digits

Problem 3: predicting earth system model parameters

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
-row.i
1
16
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).

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
...
```

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
...
```

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

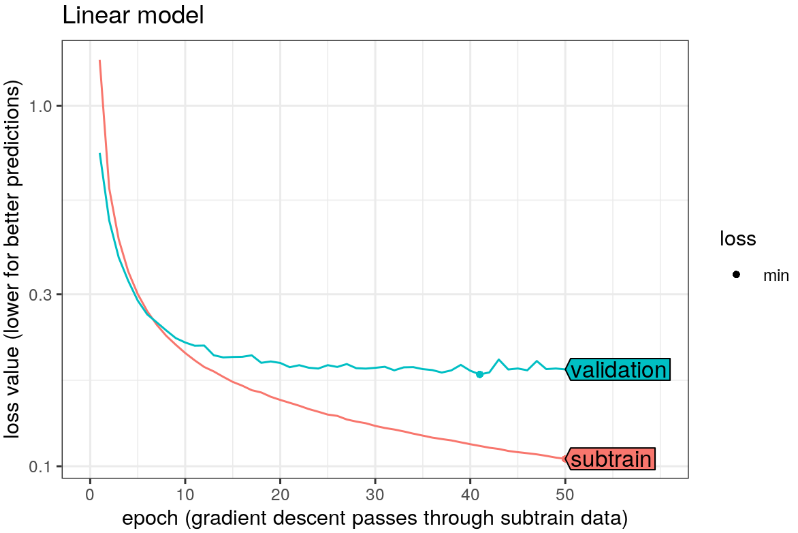
- a `loss` function, which is directly optimized/minimized in each iteration of the gradient descent learning algorithm. `https://keras.io/api/losses/`
- an `optimizer`, which is the version of gradient descent learning algorithm to use. `https://keras.io/api/optimizers/`
- an evaluation `metric` to 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

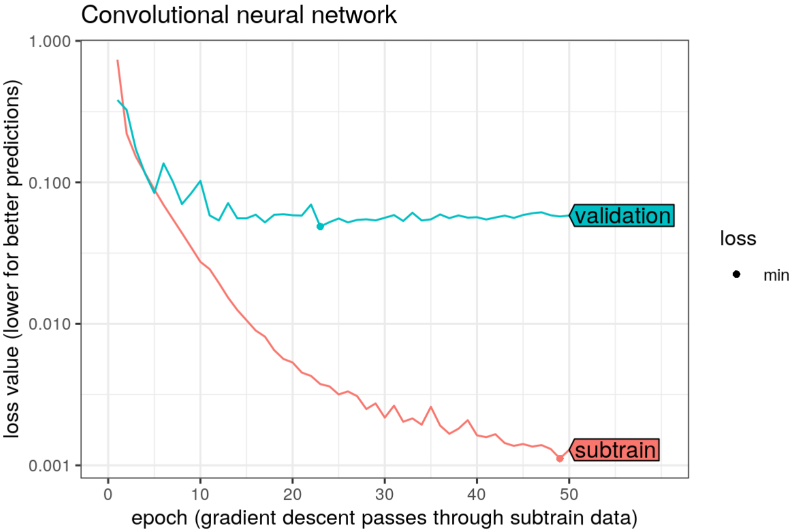
- Train data inputs `zip.X.array` and outputs `zip.y.mat` (required).
- Number of full passes of gradient descent through the subtrain data (`epochs`). In each epoch the gradient with respect to each subtrain observation is computed once.
- `validation_split=0.2` which means to use 80% subtrain (used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

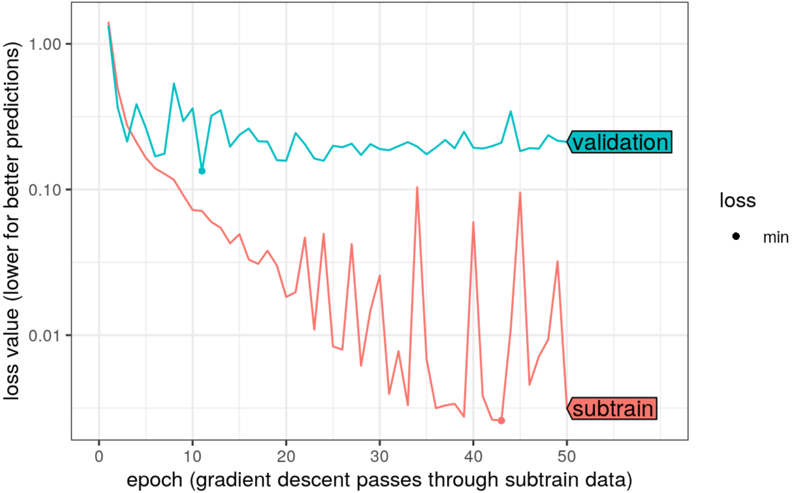
```
library(keras)
conv.model <- keras_model_sequential() %>%
  layer_conv_2d(
    input_shape = dim(zip.X.array)[-1],
    filters = 20,
    kernel_size = c(3,3),
    activation = ’relu’) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 100, activation =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

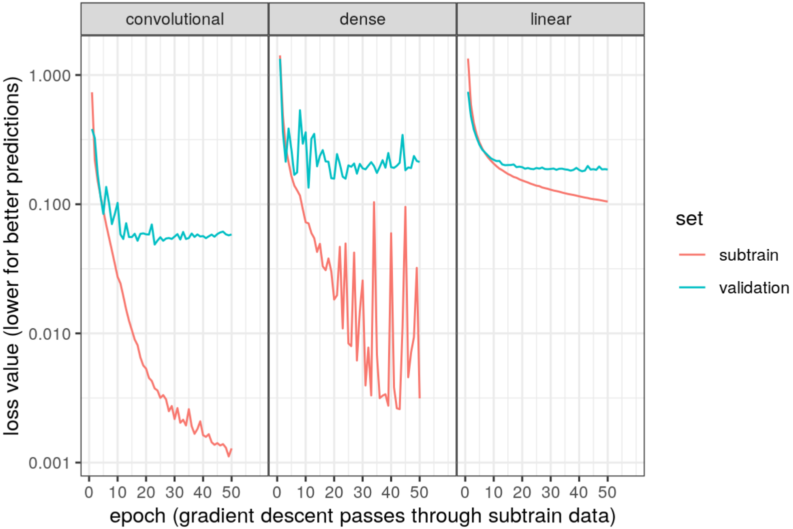

Convolutional neural network
1.000
0.100
0.010
0.001
loss value (lower for better predictions)
0
10
20
30
40
50
epoch (gradient descent passes through subtrain data)
validation
subtrain
loss
● min

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’)
```

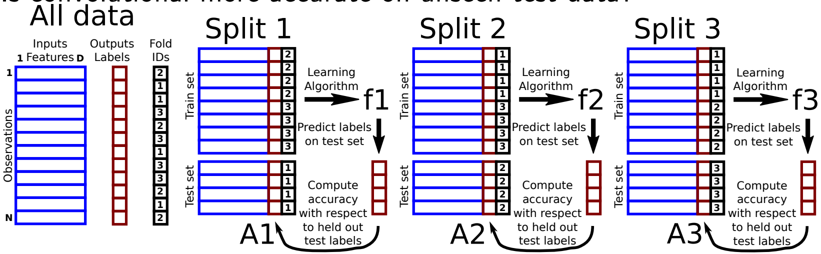


Dense (fully connected) neural network with 8 hidden layers


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

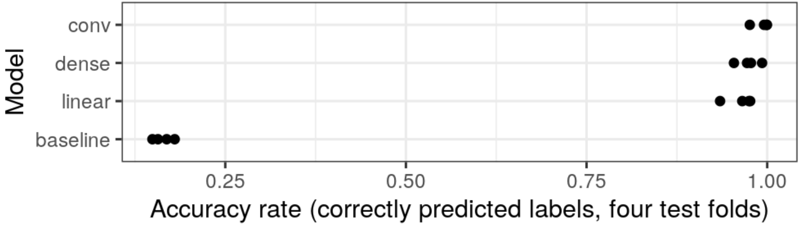
K-fold cross-validation for model evaluation

Is convolutional more accurate on unseen test data?



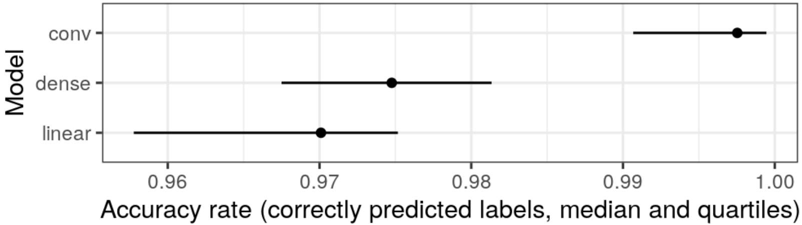
- Randomly assign a fold ID from 1 to K to each observation.
- Hold out the observations with the Split ID.
- Use the other observations as the train set.
- Run learning algorithm on train set (including hyper-parmeter selection), outputs learned function (f1-f3).
- Finally compute and plot the prediction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

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)

Introduction and overview

Problem 1: avoiding overfitting in regression

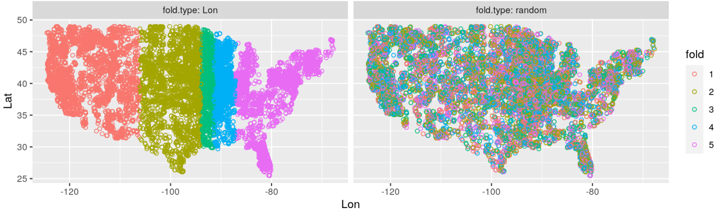
Problem 2: classifying images of digits

Problem 3: predicting earth system model parameters

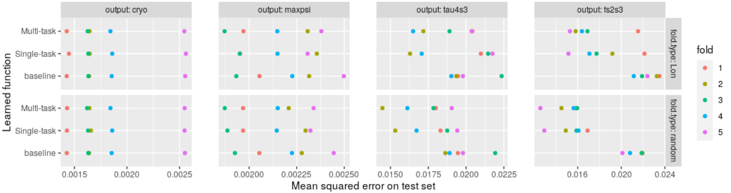
Problem setting

TODO

Results



fold.type: Lon
fold.type: random
50
45
40
35
30
25
Lat
-120
-100
-80
Lon
fold
1
2
3
4
5



output: cryo
output: maxpsi
output: tau4s3
output: fs2s3
Multi-task
Single-task
baseline
Learned function
fold.type: Lon
fold.type: random
0.0015
0.0020
0.0025
0.00200
0.00225
0.00250
0.0150
0.0175
0.0200
0.0225
0.016
0.020
0.024
Mean squared error on test set
fold
1
2
3
4
5