Introduction to machine learning and neural networks

Toby Dylan Hocking

toby.hocking@nau.edu

July 2, 2021

Abstract

In this chapter we introduce basic concepts and algorithms from machine learning. We explain how neural networks can be trained for regression and classification problems, and how cross-validation can be used for training and testing machine learning algorithms. A supervised machine learning algorithm can be used on a training data set in order to infer a prediction function. The goal is to learn a function which yields accurate predictions in a held-out set of test data (which are not available to the learning algorithm at training time). Both overfitting and underfitting can yield sub-optimal predictions on held-out test data, and can be avoided by tuning hyper-parameters using splits of train data into subtrain and validation sets. Different learning algorithms can be compared by computing prediction accuracy with respect to held-out test sets. We conclude with a discussion of applications to image classification and earth system model parameter prediction.

36.1 Introduction and applications of machine learning

Machine learning is the domain of computer science which is concerned with efficient algorithms for making predictions in all kinds of big data sets. A defining characteristic of supervised machine learning algorithms is that they require a data set for training. The machine learning algorithm then memorizes the patterns present in those training data, with the goal of accurately predicting similar patterns in new test data. Many machine learning algorithms are domain-agnostic, which means they have been shown to provide highly accurate predictions in a wide variety of application domains (computer vision, speech recognition, automatic translation, biology, medicine, climate science, chemistry, geology, etc).

For example, consider the problem of image classification from the application domain of computer vision. In this problem, we would like a function that can input an image, and output an integer which indicates class membership. More precisely, let us consider the MNIST and Fashion-MNIST data sets (Figure 36.1), in which each input is grayscale image with height and width of 28 pixels, represented as a matrix of real numbers $\mathbf{x} \in \mathbb{R}^{28\times 28}$ [LeCun et al., 1998, Xiao et al., 2017]. In both the MNIST and Fashion-MNIST data sets each image has a corresponding label which is an integer $y \in \{0, 1, \dots, 9\}$. In the MNIST data set each image/label represents a digit, whereas in Fashion-MNIST each image/label represents a category of clothing (0 for T-shirt/top, 1 for Trouser, 2 for Pullover, etc). In both data sets the goal is to learn a function $f : \mathbb{R}^{28\times 28} \rightarrow \{0, 1, \dots, 9\}$ which inputs an image $\mathbf{x}$ and outputs a predicted class $f(\mathbf{x})$ which should ideally be the same as the corresponding label y.

As mentioned above, a big advantage of supervised learning algorithms is that they are typically domain-agnostic, meaning that they can learn accurate prediction functions f using data sets with different kinds of patterns. That means we can use a single learning algorithm LEARN on both the MNIST or Fashion-MNIST data sets (Figure 36.1, left). For the MNIST data set the learning algorithm will output a function for predicting the class of digit images, and for Fashion-

MNIST the learning algorithm will output a function for predicting the class of a clothing image (Figure 36.1, right). The advantage of this supervised machine learning approach to image classification is that the programmer does not need any domain-specific knowledge about the expected pattern (e.g., shape of each digit, appearance of each clothing type). Instead, we assume there is a data set with enough labels for the learning algorithm to accurately infer the domain-specific pattern and prediction function. This means that the machine learning approach is only appropriate when it is possible/inexpensive to create a large labeled data set that accurately represents the pattern/function to be learned.

How do we know if the learning algorithm is working properly? The goal of supervised learning is **generalization**, which means the learned prediction function f should accurately predict $f(\mathbf{x}) = y$ for any inputs/outputs $(\mathbf{x}, y)$ that will be seen in a desired application (including new data that were not seen during learning). To formalize this idea, and to compute quantitative evaluation metrics (accuracy/error rates), we need a test data set, as explained in the next section.

36.1.1 K-fold cross-validation for evaluating prediction accuracy

Each input $\mathbf{x}$ in a data set is typically represented as one of N rows in a "design matrix" with D columns (one for each dimension or feature). Each output y is represented as an element of a label vector of size N, which can be visualized as another column alongside the design matrix (Figure 36.2, left). For example, in the image data sets discussed above we have $N = 60,000$ labeled images/rows, each with $D = 784$ dimensions/features (one for each of the 28×28 pixels in the image).

The goal of supervised learning is to find a prediction function f such that $f(\mathbf{x}) = y$ for all inputs/outputs $(\mathbf{x}, y)$ in a test data set (which is not available for learning f). So how do we learn f for accurate prediction on a test data set, if that test set is not available? We must assume that we have access to a train data set with the same statistical distribution as the test data. The train data set is used to learn f, and the test data can only be used for evaluating the prediction accuracy/error of f.

Some benchmark data sets which are used for machine learning research, like MNIST and Fashion-MNIST, have designated train/test sets. However, in most applications of machine learning to real data sets, train/test sets must be created. One approach is to create a single train/test split by randomly assigning a set to each of the N rows/observations, say 50% train rows and 50% test rows. The advantage of that approach is simplicity, but the drawback is that we can only report accuracy/error metrics with respect to one test set (e.g., the algorithm learned a function which accurately predicted 91.3% of observations/labels in the test set, meaning 8.7% error rate).

In addition to estimating the accuracy/error rate, it is important to have some estimate of variance in order to make statements about whether the prediction accuracy/error of the learned function f is significantly larger/smaller than other prediction functions. The other functions to compare against may be from other supervised learning algorithms, or some other method that does not use machine learning (e.g., a domain-specific physical/mechanistic model). A common baseline is the constant function $f(\mathbf{x}) = y_0$ where y_0 is the average or most frequent label in the train data. This baseline ignores all of the inputs/features $\mathbf{x}$, and can be used to show that the algorithm is learning some non-trivial predictive relationship between inputs and outputs (for an example see Figure 36.4).

The K-fold cross-validation procedure generates K splits, and can therefore be used to estimate both mean and variance of prediction accuracy/error. The number of folds/splits K is a user-defined integer parameter which must be at least 2, and at most N. Typical choices range from $K = 3$ to 10, and usually the value of K does not have a large effect on the final estimated mean/variance of prediction accuracy/error. The algorithm begins by randomly assigning a fold ID number (integer from 1 to K) to each observation (Figure 36.2, left). Then for each unique fold value from 1 to K, we hold out the corresponding observations/rows as a test set, and use data from all other folds as a train set (Figure 36.2, right). Each train set is used to learn a corresponding prediction function, which is then used to predict on the held out test data. Finally, accuracy/error metrics are computed in order to quantify how well the predictions fit the labels for the test data. Overall for each data set and learning algorithm the K-fold cross-validation procedure results in K splits, K learned functions, and K test accuracy/error metrics, which are typically combined by taking the mean and standard deviation (or median and quartiles). Other algorithms may be used with the same fold assignments, in order to compare algorithms in terms of accuracy/error rates in particular data sets.

For example, Figure 36.4 uses $K = 4$-fold cross-validation to compare four learned functions on an image classification problem. The accuracy rates of the "dense" and "linear" functions, $97.4 \pm 1.6\%$ and $96.3 \pm 1.9\%$ (mean ± standard deviation) are not significantly different. Both rates are significantly larger than the accuracy of the "baseline" constant function, $16.4 \pm 1.4\%$, and smaller than the accuracy of the "conv" function, $99.3 \pm 1.1\%$. We can therefore conclude that the most accurate learning algorithm for this problem, among these four candidates, is the "conv" method (which uses a convolutional neural network, explained later). It is important to note that statements about what algorithm is most accurate can only be made for a particular data set, after having performed K-fold cross-validation to estimate prediction accuracy/error rates.

36.1.2 Other applications

So far we have only discussed machine learning algorithms in the context of a single prediction problem, image classification. In this section we briefly discuss other applications of machine learning. In each application the set of possible inputs $\mathbf{x}$ and outputs y are different, but machine learning algorithms can always be used to learn a prediction function $f(x) \approx y$.

[Jones et al., 2009] proposed to use interactive machine learning for cell image classification in the CellProfiler Analyst system. This application is similar to the previously discussed digit/fashion classification problem, but with only two classes (binary classification). In this context the input is a multi-color image of cell $\mathbf{x} \in \mathbb{R}^{h \times w \times c}$ where h, w are the height and width of the image in pixels, and $c = 3$ is the number of channels used to represent a color image (red, green, blue). The output $y \in \{0, 1\}$ is a binary label which indicates whether or not the image contains the cell phenotype of interest.

Some email programs use machine learning for spam filtering, which is another example of a binary classification problem. When you click the "spam" button in the email program you are labeling that email as spam ($y = 1$), and when you respond to an email you are labeling that email as not spam ($y = 0$). The input $\mathbf{x}$ is an email message, which can be represented using a "bag-of-words" vector (each element is the number of times a specific word occurs in that email message).

Russell et al. [2008] proposed the LabelMe tool for creating data sets for image segmentation, which is more complex than the previously discussed image classification problems. In this context the input $\mathbf{x} \in \mathbb{R}^{h \times w \times c}$ is typically a multi-color image, and the output $\mathbf{y} \in \{0, 1\}^{h \times w}$ is a binary mask (one element for every pixel in the image) indicating whether or not that pixel contains an object of interest.

Machine learning can be used for automatic translation between languages. In this context the input is a text in one language (e.g., French) and the output is the text translated to another language (e.g., English). The desired prediction function f inputs a French text and outputs the English translation.

Machine learning can be used for medical diagnosis. For example Poplin et al. [2017] showed that retinal photographs can be used to predict blood pressure or risk of heart attack. Since the output y is a real number (e.g., blood pressure of 120 mm mercury), we refer to this as a regression problem.

36.2 Avoiding under- and over-fitting in a neural network

In this section we begin by explaining the prediction function and learning algorithm for a simple neural network. We then demonstrate how the number of iterations of the learning algorithm can be selected using a validation set, in order to avoid underfitting and overfitting.

We assume that there is a train set of data to learn from. For example, consider a simple regression problem for which each input $x \in \mathbb{R}$ is a single real number ($D = 1$ feature/column in the design matrix), and each output $y \in \mathbb{R}$ is as well. Using a neural network with a single hidden layer of U units, there are two unknown **parameter** vectors which need to be learned, $\mathbf{w} \in \mathbb{R}^U$ and $\mathbf{v} \in \mathbb{R}^U$. The prediction function f is then defined as

$$f(x) = \mathbf{w}^\intercal \sigma(x\mathbf{v}) = \mathbf{w}^\intercal \mathbf{z}, \tag{36.1}$$

where $\sigma : \mathbb{R}^U \rightarrow \mathbb{R}^U$ is a non-linear activation function, and $\mathbf{z} \in \mathbb{R}^U$ is the vector of hidden units. Typical activation functions include the logistic sigmoid $\sigma(t) = 1/(1 + \exp(-t))$ and the rectifier (or rectified linear units, ReLU) $\sigma(t) = \max(0, t)$.

Assuming there are N data $(x_1, y_1), \dots, (x_N, y_N)$, we can learn the prediction function using gradient descent, which is an algorithm that attempts to find parameters $\mathbf{w}, \mathbf{v}$ that minimize the mean squared error between the predictions and the corresponding labels,

$$\mathcal{L}(\mathbf{w}, \mathbf{v}) = \frac{1}{N} \sum_{i=1}^{N} [\mathbf{w}^\intercal \sigma(x_i \mathbf{v}) - y_i]^2. \tag{36.2}$$

Note that the mean squared error in the equation above is referred to as the "loss function" and should only be used when the desired output is a real number (for other kinds of outputs, a different loss function should be used). Gradient descent begins using un-informative parameters $\mathbf{w}_0, \mathbf{v}_0$ (typically random numbers close to zero), then at each iteration $t \in \{1, \dots, T\}$ the parameters are improved by taking a step of size $\alpha > 0$ in the negative gradient direction,

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}_{t-1}, \mathbf{v}_{t-1}), \tag{36.3}$$

$$\mathbf{v}_t = \mathbf{v}_{t-1} - \alpha \nabla_{\mathbf{v}} \mathcal{L}(\mathbf{w}_{t-1}, \mathbf{v}_{t-1}). \tag{36.4}$$

These gradient steps have the effect of updating the parameters so that the resulting predictions achieve lower loss function values. The algorithm described above is referred to as "full gradient" because the gradient descent direction is defined using the full set of N samples. Other common variants include "stochastic gradient" (gradient uses one sample) and "minibatch" (gradient uses several samples). When doing gradient descent on a neural network model, one "epoch" includes computing gradients once for each sample (e.g., 1 epoch = 1 iteration of full gradient, 1 epoch = N iterations of stochastic gradient).

In the algorithm above, the number of hidden units U, the number of iterations T, and the step size α must be fixed before running the learning algorithm. These **hyper-parameters** affect the learning capacity of the neural network. An important consideration when using any machine learning algorithm is that you most likely need to tune the hyper-parameters of the algorithm in order to avoid underfitting and overfitting. **Underfitting** occurs when the learned function f neither provides accurate predictions for the train data, nor the test data. **Overfitting** occurs when the learned function f only provides accurate predictions for the train data (and not for the test data). Both underfitting and overfitting are bad, and need to be avoided, because the goal of any learning algorithm is to find a prediction function f which provides accurate predictions in test data.

How can we select hyper-parameters which avoid overfitting? Note that the choice of hyper-parameters such as number of hidden units U and iterations T affect the learned function f, so we can not use the test data to learn these hyper-parameters (by assumption that the test data are not available at train time). Then how do we know which hyper-parameters will result in learned functions which best generalize to new data?

A general method which can be used with any learning algorithm is splitting the train set into subtrain and validation sets, then using grid search over hyper-parameter values. The subtrain set is used for parameter learning, and the validation set is used for hyper-parameter selection. In detail, we first fix a set of hyper-parameters, say $U = 50$ hidden units and $T = 100$ iterations. Then the subtrain set is used with these hyper-parameters as input to the learning algorithm, which outputs the learned parameter vectors $\mathbf{w}, \mathbf{v}$. Finally the learned parameters are used to compute predictions $f(x)$ for all inputs x in the validation set, and the corresponding labels y are used to evaluate the accuracy/error of those predictions. The procedure is then repeated for another hyper-parameter set, say $U = 10$ hidden units with $T = 500$ iterations. In the end we select the hyper-parameter set with minimal validation error, and then retrain using the learning algorithm on the full train set with those hyper-parameters. A variant of this method is to use K-fold cross-validation to generate K subtrain/validation splits, then compute mean validation error over the K splits, which typically yields hyper-parameters that result in more accurate/generalizable predictions (when compared to hyper-parameters selected using a single subtrain/validation split). Note that this K-fold cross-validation for hyper-parameter learning is essentially the same procedure as shown in Figure 36.2, but we split the train set into subtrain/validation sets (instead of splitting all data into train/test sets as shown in the figure).

For example we simulated some data with a sine wave pattern (Figure 36.3), and used the R package `nnet` to fit a neural network with one hidden layer of $U = 50$ units [Venables and Ripley, 2002]. We demonstrate the effects of under/overfitting by varying the number of iterations/epochs from $T = 1$ to 1000. In this example $K = 4$-fold cross-validation was used, so each data point was randomly assigned a fold ID integer from 1 to 4. The result for only the first split is shown, so observations assigned fold ID=1 are considered the validation set, and other observations (folds 2–4) are considered the subtrain set (which is used at input to the `nnet` R function which implements the gradient descent learning algorithm). We then used the `predict` function in R to compute predictions for subtrain and validation data, and analyzed how the prediction error changes as a function of the number of iterations/epochs T of gradient descent. The data exhibit a non-linear sine wave pattern, but the learned function for $T = 4$ iterations/epochs is mostly linear (underfitting, large error on both subtrain/validation sets). For $T = 512$ iterations/epochs the learned function is highly non-linear (overfitting, small error for the subtrain set but large error for the validation set). When the error rates are plotted as a function of a model complexity hyper-parameter such as T (Figure 36.3, right), we see the characteristic U shape for the validation error, and the monotonic decreasing train error. The hyper-parameter with minimal validation error is $T = 64$ iterations/epochs; smaller T values underfit or are overly regularized, and larger T values overfit or are under-regularized.

Overall in this section we have seen how a neural network for regression can be trained using gradient descent (for learning parameter vectors, given fixed hyper parameters) and sub-train/validation splits (for learning hyper-parameter values to avoid under/overfitting).

36.3 Comparing prediction accuracy of neural networks

In this section we provide a comparison of several other neural networks for image classification. In general in a neural network with $L - 1$ hidden layers we can represent the prediction function as the composition of L intermediate f_l functions, for all layers $l \in \{1, \dots, L\}$:

$$f(\mathbf{x}) = f_L[\cdots f_1[\mathbf{x}]]. \tag{36.5}$$

Each of the intermediate functions has the same form,

$$f_l(t) = A_l(\mathbf{W}_l^\intercal t), \tag{36.6}$$

where A_l is an activation function and $\mathbf{W}_l \in \mathbb{R}^{u_l \times u_{l-1}}$ is a weight matrix with elements that must be learned based on the data. This model includes several hyper-parameters which must be fixed prior to learning the neural network weights:

- The number of layers L.
- The activation functions A_l.

- The number of units per layer u_l.

- The sparsity pattern in the weight matrices $\mathbf{W}_l$.

The number of units in the input layer is fixed, $u_0 = D$, based on the dimension of the inputs $\mathbf{x} \in \mathbb{R}^D$. The number of units in the output layer u_L is also fixed based on the outputs/labels y. The numbers of units in the hidden layers $(u_1, \dots, u_{L-1})$ are hyper-parameters which control under/overfitting. Increasing the numbers of hidden units u_l results in larger weight matrices $\mathbf{W}_l$, which in general means more parameters to learn, and larger capacity for fitting complex patterns in the data. The sparsity pattern of $\mathbf{W}_l$ means which entries are forced to be zero; this technique is used in "convolutional" neural networks for avoiding overfitting and reducing training/prediction time. When the matrix is not sparse (all entries non-zero), we refer to the layer as dense or fully connected.

For example in the previous section we used a neural network for regression with one hidden layer, which in this more general notation means using $L = 2$ intermediate functions; the input dimension is $u_0 = D = 1$, the number of hidden units is $u_1 = U = 50$, and there is a single output $u_2 = 1$ to predict. The weight matrices are dense/fully connected (no convolution/sparsity), of dimension $\mathbf{W}_1 \in \mathbb{R}^{50\times 1}, \mathbf{W}_2 \in \mathbb{R}^{1\times 50}$. The hidden layer activation function A_1 used by the R `nnet` package is the logistic sigmoid, $\sigma(t) = 1/(1 + \exp(-t))$, and the output activation for regression (real-valued outputs) is the identity, $A_2(t) = t$.

In this section we implement three other neural networks for image classification. Using the "zip.train" data set of $N = 7291$ handwritten digits [Hastie et al., 2009], each input is a greyscale image of 16×16 pixels which means that number of input units is $u_0 = 256$. As in Figure 36.1 (top) there are ten output classes, one for each digit. For the activation function A_L in the output layer we use the "softmax" function which results in a score/probability for each of the ten possible output classes, so the number of output units is $u_L = 10$.

The three neural networks that we consider are

linear $L = 1$ intermediate function with 2,570 parameters to learn (linear model, inputs fully connected to outputs, no hidden units/layers).

dense $L = 9$ intermediate functions with 97,410 parameters to learn (nonlinear model, each hidden layer dense/fully connected with 100 units).

sparse $L = 3$ intermediate functions with 99,310 parameters to learn (nonlinear model, one convolutional/sparse layer followed by two dense/fully connected layers).

We defined and trained each neural network using the `keras` R package [Allaire and Chollet, 2020]. We used the `fit` function with argument `validation_split=0.2`, which creates a single split (80% subtrain, 20% validation). We selected the number of epochs hyper-parameter by minimizing the validation loss, and we used the selected number of epochs to re-train the neural network on the entire train set (no subtrain/validation split).

We did this entire procedure $K = 4$ times, once for each fold/split in K-fold cross-validation. Note that even though these data have a pre-defined split into “zip.train” and “zip.test” files, we used K-fold cross-validation on the “zip.train” file, yielding K train/test splits that we used to estimate mean and variance of prediction accuracy for these models (the “zip.test” file was ignored). In each split we used the test set to quantify the prediction accuracy of the learned models. It is clear that the test accuracy of all three neural networks is significantly larger than the baseline model which always predicts the most frequent class in the train set (Figure 36.4, left); they are clearly learning some non-trivial predictive relationship between inputs and outputs. Furthermore it is clear from Figure 36.4 (right) that the dense neural network is slightly more accurate than the linear model ($p = 0.032$ in paired one-sided t_3-test), and the sparse/convoluational neural network is significantly more accurate than the dense model ($p = 0.009$).

Overall from this comparison it is clear that, among these three neural networks, the sparse model should be preferred for most accurate predictions in this particular “zip” data set. However, we must be careful to not generalize these conclusions to other data sets — even for some other image classification data sets such as MNIST (Figure 36.1), the most accurate algorithm may be different. For very difficult data sets, it may even be the case that these three neural networks are no more accurate than the baseline model which always predicts the most frequent class in the train set. In general we always need to use computational cross-validation experiments to determine which machine learning algorithm is most accurate in any given data set. To learn a predictive model with maximum prediction accuracy, machine learning algorithms other than neural networks should be additionally considered (e.g., regularized linear models, decision trees, random forests, boosting, support vector machines).

36.4 Cross-validation for evaluating predictions of earth system model parameters

As a final example application, we consider using cross-validation to evaluate a neural network that predicts carbon cycle model parameters [Tao et al., 2020]. In this context there is a data set with $N = 26,158$ observations, each one a soil sample with $D = 60$ input features. There are 25 real-valued output variables to predict; each is the value of an earth system model parameter at the location of the soil sample. We want a neural network that will be able to predict the values of these earth system parameters at new locations. Tao et al. [2020] proposed using a neural network with $L = 4$ fully connected layers and dropout regularization for this task (see paper for details). In this section the “multi-task” model uses the same number of layers/units as described in that paper; the term multi-task means that the neural network outputs a prediction for all 25 outputs/tasks. For comparison, we additionally consider “single-task” models with the same number of hidden layers/units, but only one output unit. We expect the multi-task model to sometimes be more accurate, because of the expected correlation between outputs (earth system model parameters). To see whether or not these neural networks learn any non-trivial predictive relationship between inputs and output, we consider a baseline model which always predicts the mean of the train set label/output values (and does not use the inputs at all).

Here we show how $K = 5$ fold cross-validation can be used to evaluate how well these neural networks predict each of the outputs at new locations. We first assign a fold ID from 1 to 5 to each observation/row, either systematically using the longitude coordinate, or randomly (Figure 36.5, top). We can define a cross-validation procedure using both sets of fold IDs, in order to answer the question, "is it more difficult to predict at new longitudes, or new random locations?" We expect that predicting at new longitudes should be more difficult, because that involves more extrapolation (predicting outside the range of observed data values). In detail, for each fold ID from 1 to 5, we define the test set as the data points which have been assigned that fold ID using both methods (longitude and random). For these data with $N = 26,158$ observations total, each fold has approximately 5000 observations, so each resulting test set has approximately 1000 observations.

As described in the last section on image classification, we used the R `keras` package to compute the neural network parameters and predictions (using a maximum of 100 epochs, and a single 80% subtrain 20% validation split to choose the optimal number of epochs for re-training on the entire train set). For each fold/model/output we computed mean squared error with respect to the test set, and we plot these values for four of the 25 outputs (Figure 36.5, bottom). It is clear that some outputs are more difficult to predict than others; for cryo and maxpsi outputs the neural networks show little or no improvement over baselines, whereas for tau4s3 and fs2s3 outputs we observed substantial improvements over baselines. As expected, there is a difference in test error between fold assignment methods (random has lower error rates than Lon for several outputs), indicating that it is indeed easier to predict at new random locations, and harder to predict at new longitudes. Finally, the multi-task models are slightly more accurate than the single task models, indicating that the neural network is learning to exploit the correlations between outputs. Overall this comparison has shown how cross-validation can be used to quantitatively evaluate and compare machine learning algorithms for predicting earth system model parameters.

In comparison to the neural network practice in unit 10, the main difference is that here we discussed how held-out test sets can be used to estimate prediction accuracy/error rates of learning algorithms. Unit 10 discusses how a validation set can be used to avoid overfitting, as we have done in this chapter as well. We have additionally discussed how $K = 5$ fold cross-validation can be used to generate several train/test splits, which can be used to estimate prediction error rates for each fold/data/algorithm combination (e.g., Figure 36.5, bottom). This technique is useful since it allows us to see which algorithms are significantly more/less accurate than others on given data sets.

36.5 Quiz questions

1. When using a design matrix to represent machine learning inputs, what does each row and column represent? What other data/options does a supervised learning algorithm such as gradient descent need as input, and what does it yield as output?

2. 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? What is the advantage of using K-fold cross-validation, relative to a single split?

3. In order to determine if any non-trivial predictive relationship between inputs and output has been learned, a comparison with a baseline that ignores the inputs must be used. How do you compute the baseline predictions, for regression and classification problems?

4. How can you tell if machine learning model predictions are underfitting or overfitting?

5. When using the `nnet` function in R to learn a neural network with a single hidden layer, do large or small values of the number of iterations hyper-parameter result in overfitting? Why?

6. When using the `nnet` function in R learn a neural network with a single hidden layer, and you do not yet know how many iterations to use, what data set should you use as input to

`nnet`? How should you learn the number of iterations to avoid underfitting and overfitting? After having computed the number of iterations to use, what data set should you then use as input to `nnet` to learn your final model? Hint: possible choices for set to use are all, train, test, subtrain, validation.

36.6 Additional reading

Machine learning is a large field of research with many algorithms, and there are several useful textbooks that provide overviews from various perspectives [Bishop, 2006, Hastie et al., 2009, Wasserman, 2010, Murphy, 2013, Goodfellow et al., 2016].

Reproducibility statement. Code for figures in this chapter can be freely downloaded from `https://github.com/tdhock/2020-yiqi-summer-school`

Bibliography

J. Allaire and F. Chollet. *keras: R Interface to 'Keras'*, 2020. URL https://CRAN.R-project.org/package=keras. R package version 2.3.0.0.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

I. J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Springer Series in Statistics. Springer, Springer Science+Business Media, LLC, 233 Spring Street, New York NY 10013, USA, second edition, 2009.

T. R. Jones, A. E. Carpenter, M. R. Lamprecht, J. Moffat, S. J. Silver, J. K. Grenier, A. B. Castoreno, U. S. Eggert, D. E. Root, P. Golland, and D. M. Sabatini. Scoring diverse cellular morphologies in image-based screens with iterative feedback and machine learning. *Proceedings of the National Academy of Sciences*, 106(6):1826–1831, 2009.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, November 1998.

K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT Press, Cambridge, MA, 2013.

R. Poplin, A. Varadarajan, K. Blumer, Y. Liu, M. McConnell, G. Corrado, L. Peng, and D. Webster. Predicting cardiovascular risk factors from retinal fundus photographs using deep learning. arXiv:1708.09843, 2017.

B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. LabelMe: a database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1–3):157–173, May 2008.

F. Tao, Z. Zhou, Y. Huang, Q. Li, X. Lu, S. Ma, X. Huang, Y. Liang, G. Hugelius, L. Jiang, R. Doughty, Z. Ren, and Y. Luo. Deep learning optimizes data-driven representation of soil organic carbon in earth system model over the conterminous united states. *Frontiers in Big Data*, 3:17, 2020. ISSN 2624-909X. doi: 10.3389/fdata.2020.00017. URL `https://www.frontiersin.org/article/10.3389/fdata.2020.00017`.

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL `http://www.stats.ox.ac.uk/pub/MASS4`. ISBN 0-387-95457-0.

L. Wasserman. *All of statistics: a concise course in statistical inference*. Springer, New York, 2010.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv:1708.07747, 2017.

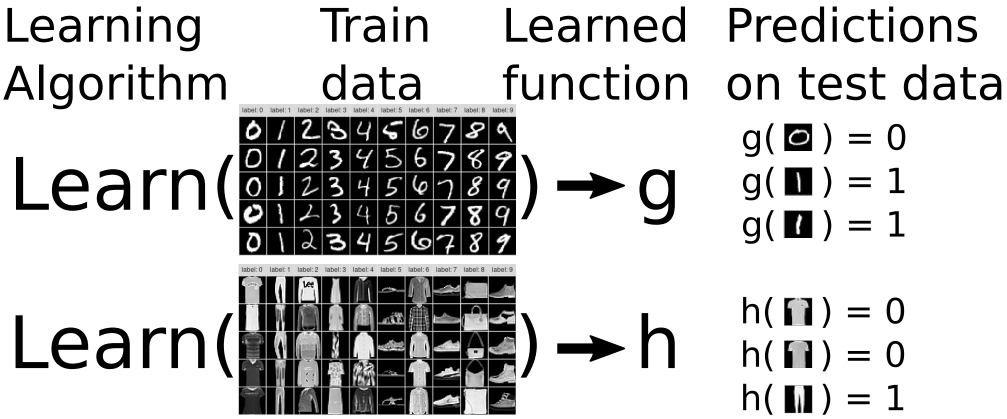

Figure 36.1: A learning algorithm inputs a train data set, and outputs a prediction function, g or h. Both g and h input a grayscale image and output a class (integer from 0 to 9), but g is for digits and h is for fashion.

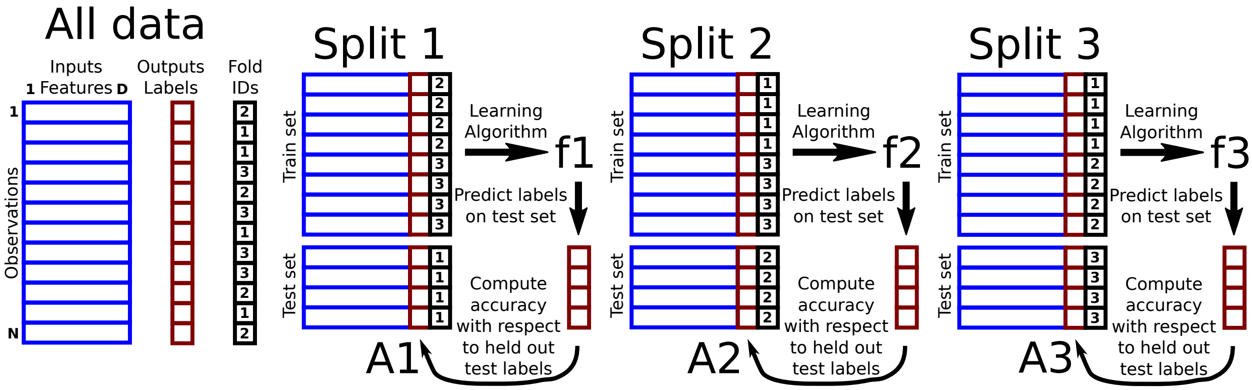

Figure 36.2: $K = 3$ fold cross-validation. **Left:** the first step is to randomly assign a fold ID from 1 to K to each of the observations/rows. **Right:** in each of the $k \in \{1, \dots, K\}$ splits, the observations with fold ID k are set aside as a test set, and the other observations are used as a train set to learn a prediction function (f1–f3), which is used to predict for the test set, and to compute accuracy metrics (A1–A3).

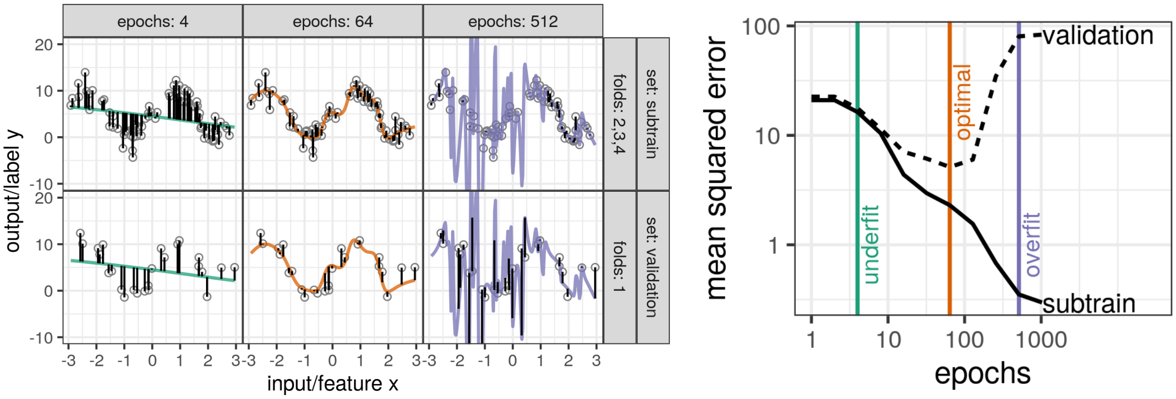

Figure 36.3: Illustration of underfitting and overfitting in a neural network regression model (single hidden layer, 50 hidden units). **Left:** noisy data with a nonlinear sine wave pattern (grey circles), learned functions (colored curves), and residuals/errors (black line segments) are shown for three values of epochs (panels from left to right) and two data subsets (panels from top to bottom). **Right:** in each epoch the model parameters are updated using gradient descent with respect to the subtrain loss, which decreases with more epochs. The optimal/minimum loss with respect to the validation set occurs at 64 epochs, indicating underfitting for smaller epochs (green function, too regular/linear for both subtrain/validation sets) and overfitting for larger epochs (purple function, very irregular/nonlinear so good fit for subtrain but not validation set).

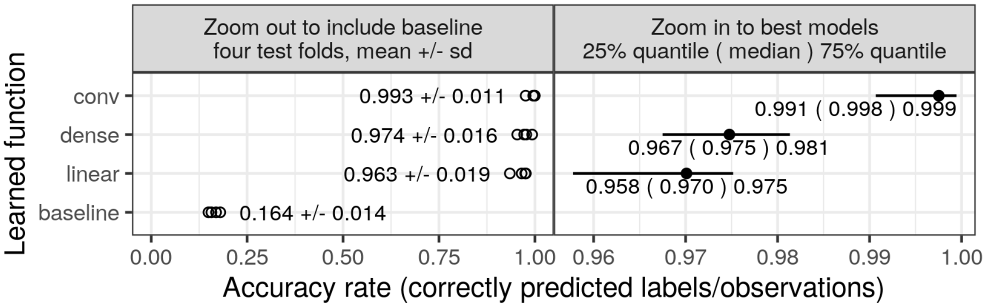

Figure 36.4: Prediction accuracy of functions learned for image classification of handwritten digits. The baseline function always predicts the most frequent class in the train set; other three learned functions are neural networks with different numbers of hidden layers (linear=0, conv=2, dense=8).

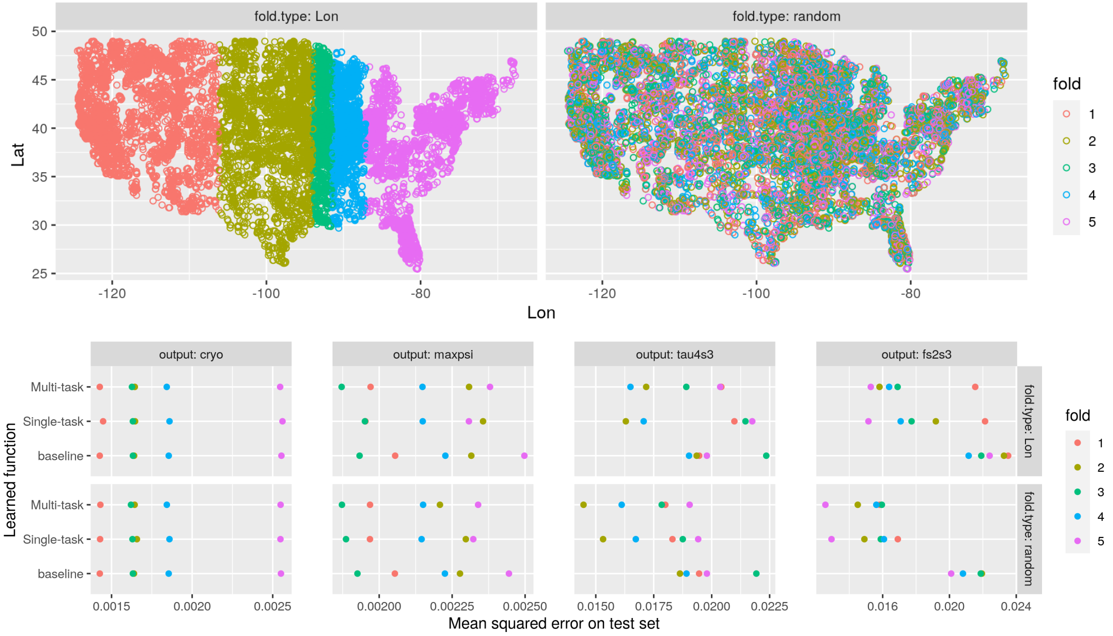

Figure 36.5: Cross-validation for estimating error rates of machine learning algorithms that predict earth system model parameters. **Top:** fold IDs were assigned to each observation using longitude (left) or randomly (right). **Bottom:** prediction error for four of the 25 outputs. Please see [Tao et al., 2020] for meanings of abbreviations (cryo, maxpsi, tau4s3, fs2s3).