Machine Learning
A Quantitative Approach

Henry H. Liu

PerfMath

ISBN-13: 978-1986487528
ISBN-10: 1986487520

10 9 8 7 6 5 4 3 2 1
03232019

1 Introduction to Machine Learning

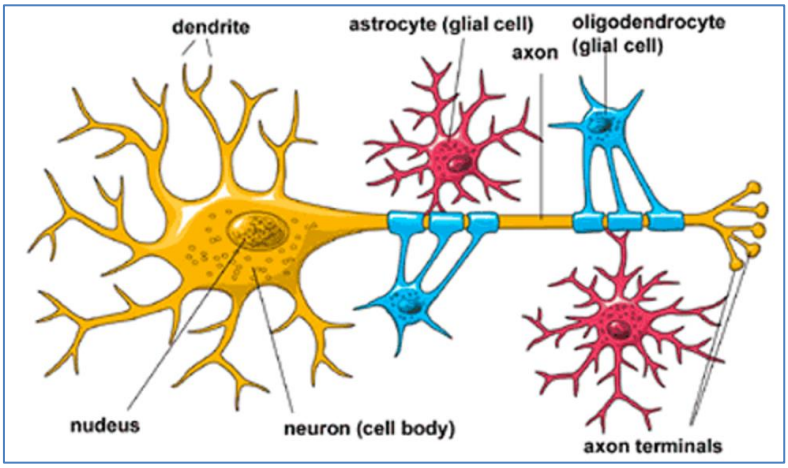


Figure 1.1 The structure of a neuron. (Source: Courtesy of Dr. Bear.)

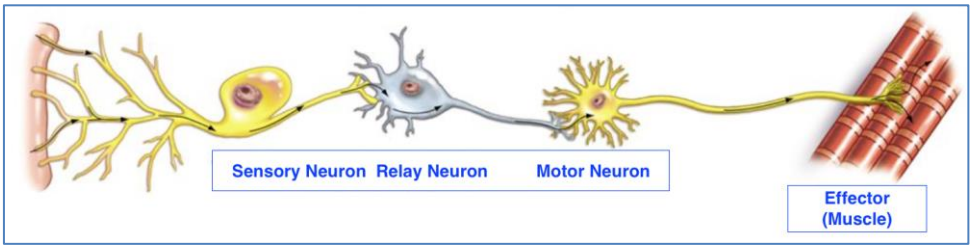


Figure 1.2 Three different types of neurons. (Source: Courtesy of Dr. Bear.)

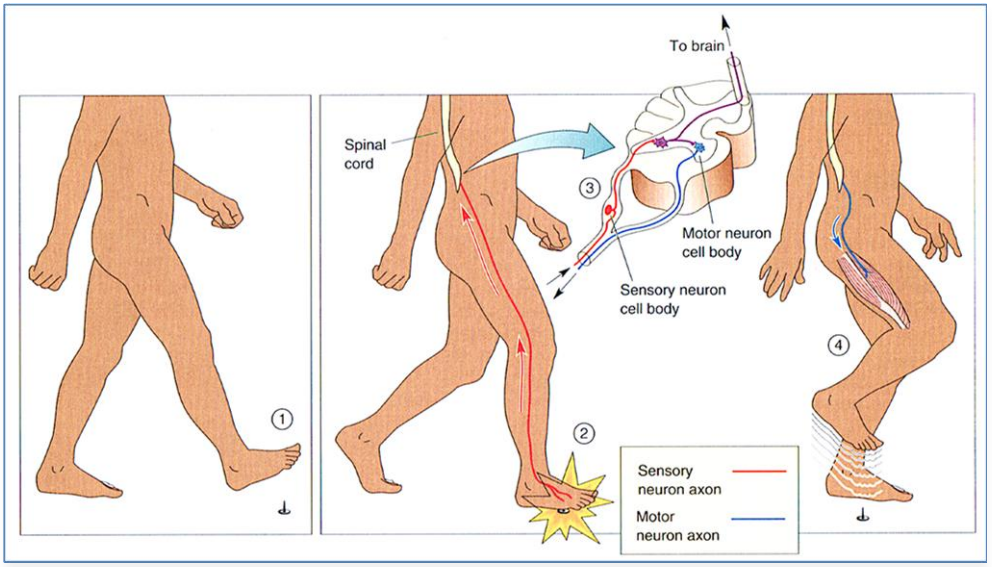


Figure 1.3 A simple reflex. (Source: Courtesy of Dr. Bear.)

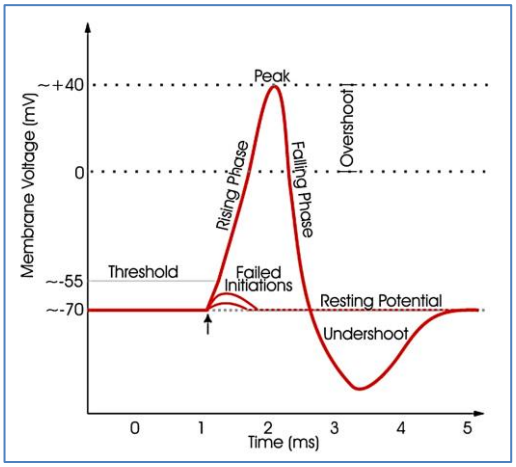


Figure 1.4 Action potentials. (Source: Courtesy of Dr. Bear.)

2 Machine Learning Fundamentals Illustrated with Regression

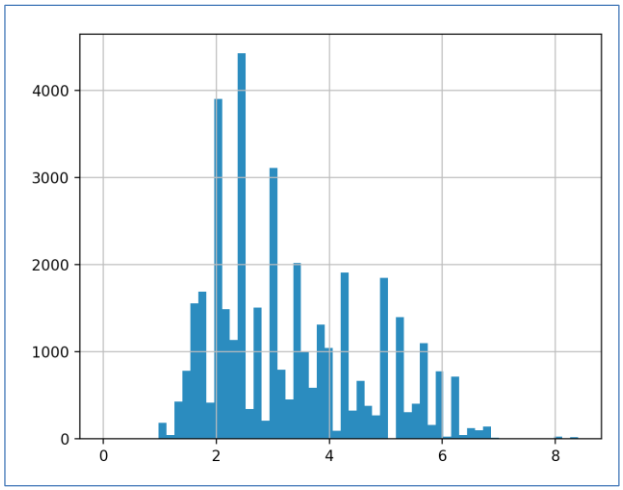


Figure 2.1 Histogram of the pandas vehicles_displ Series object.

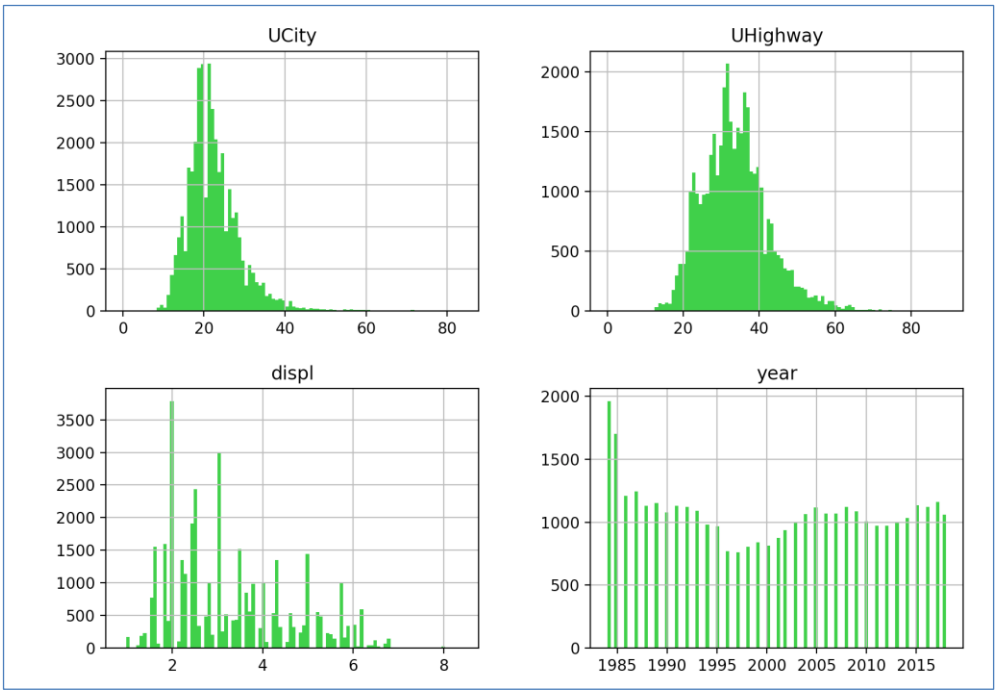


Figure 2.2 Histograms for each of the four numerical attributes of a pandas DataFrame object.

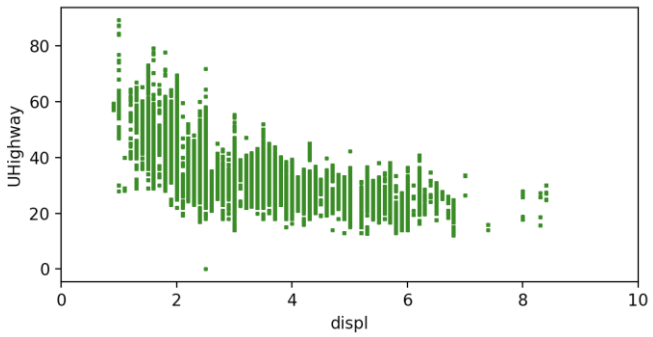


Figure 2.3 Scatter plot between unadjusted highway MPG (UHighway) and engine displacement (displ).

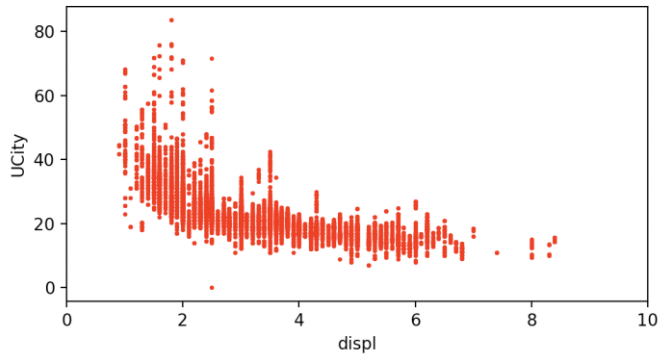


Figure 2.4 Scatter plot between unadjusted city MPG (UCity) and engine displacement (displ).

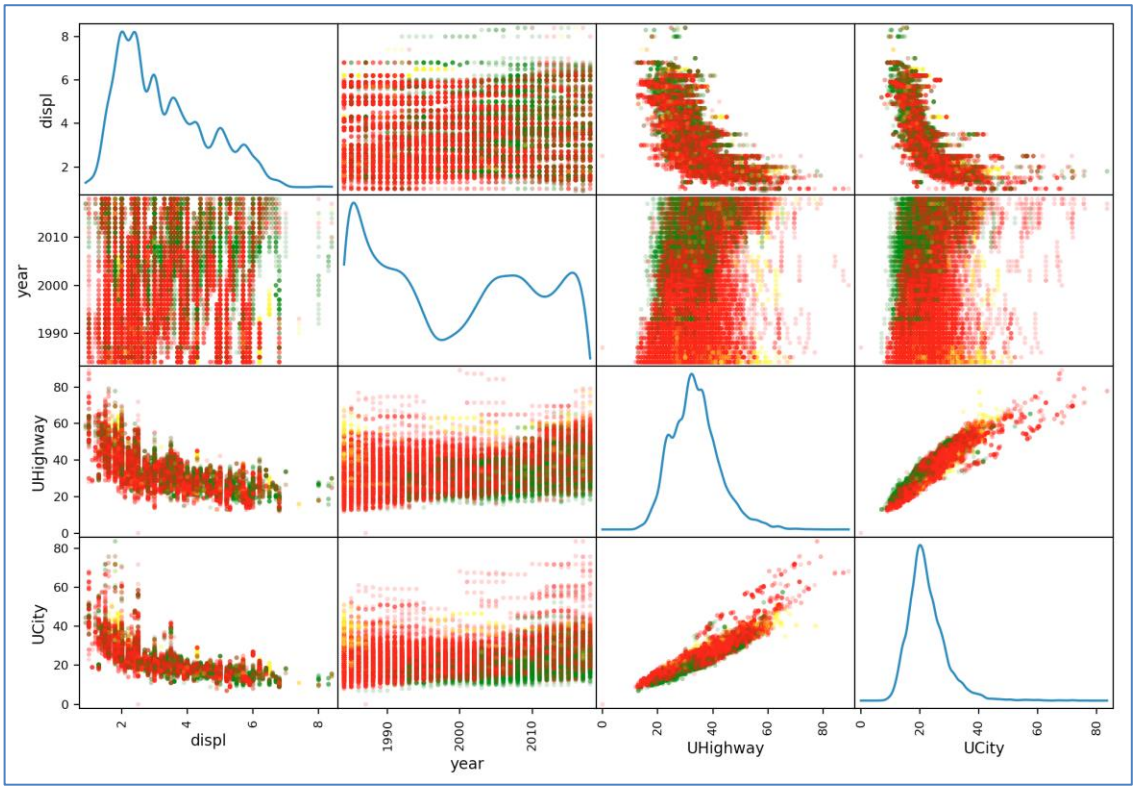


Figure 2.5 Scatter matrix plots among the four numerical attributes of displ, year, UHighway and UCity.

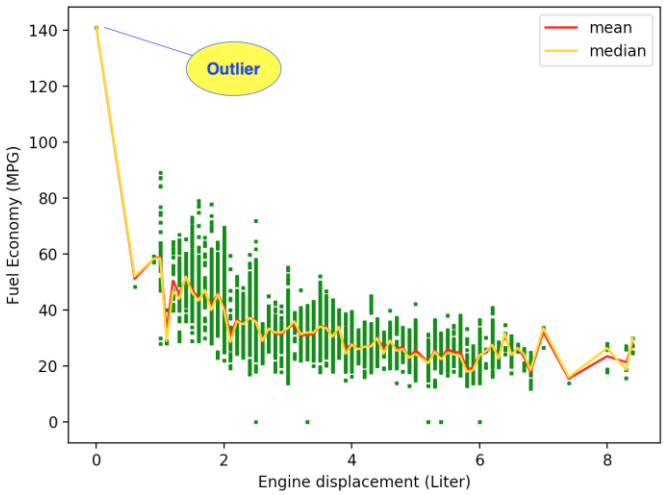


Figure 2.6 A case of an outlier for the fuel economy data set showing an MPG value of 140 at zero liter engine displacement.

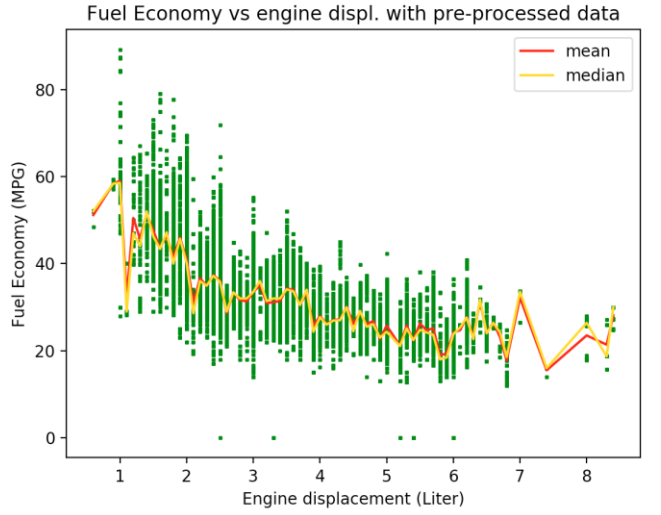


Figure 2.7 Mean and average fuel economy (MPG) data versus engine displacement.

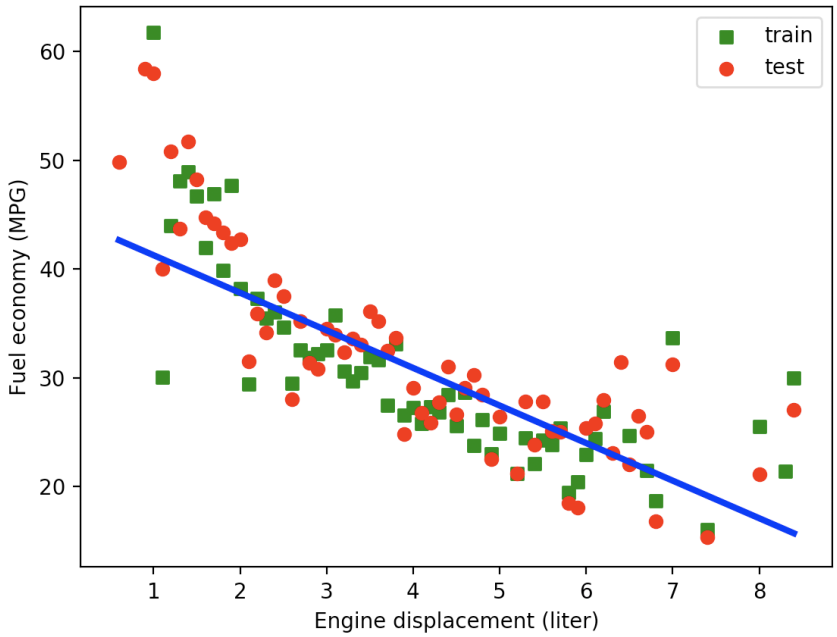


Figure 2.8 Result of fuel economy vs engine displacement with the linear regression model.

Figure 2.9 The author's 2016 2.5 liter Toyota Camry LE's measured average MPG for comparison with the EPA's fuel economy measurements modeled with the linear regression machine learning model.

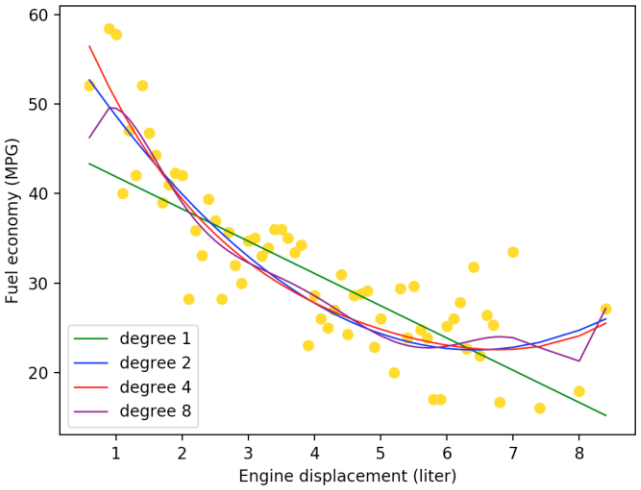


Figure 2.10 Fuel economy modeled with linear regression polynomial features.

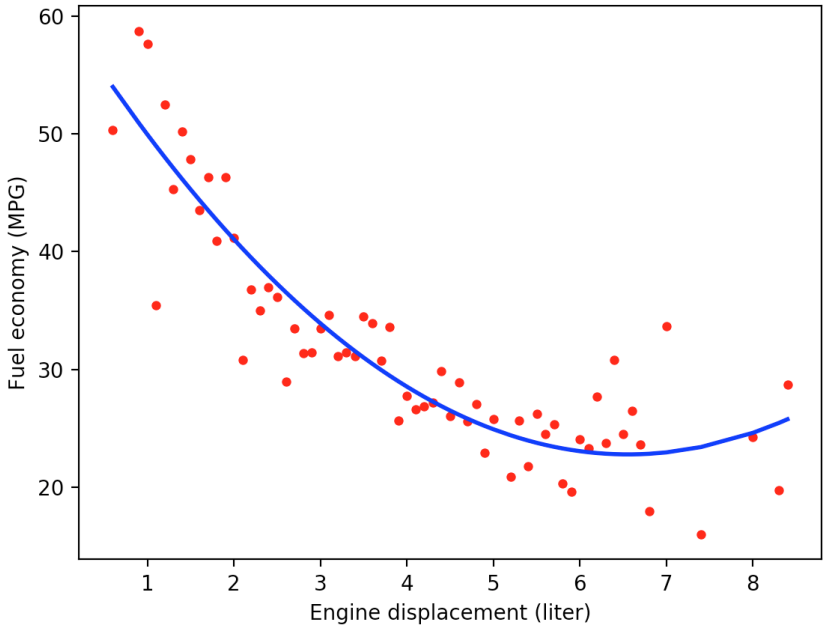


Figure 2.11 Baseline of the linear quadratic regression model for cross validation.

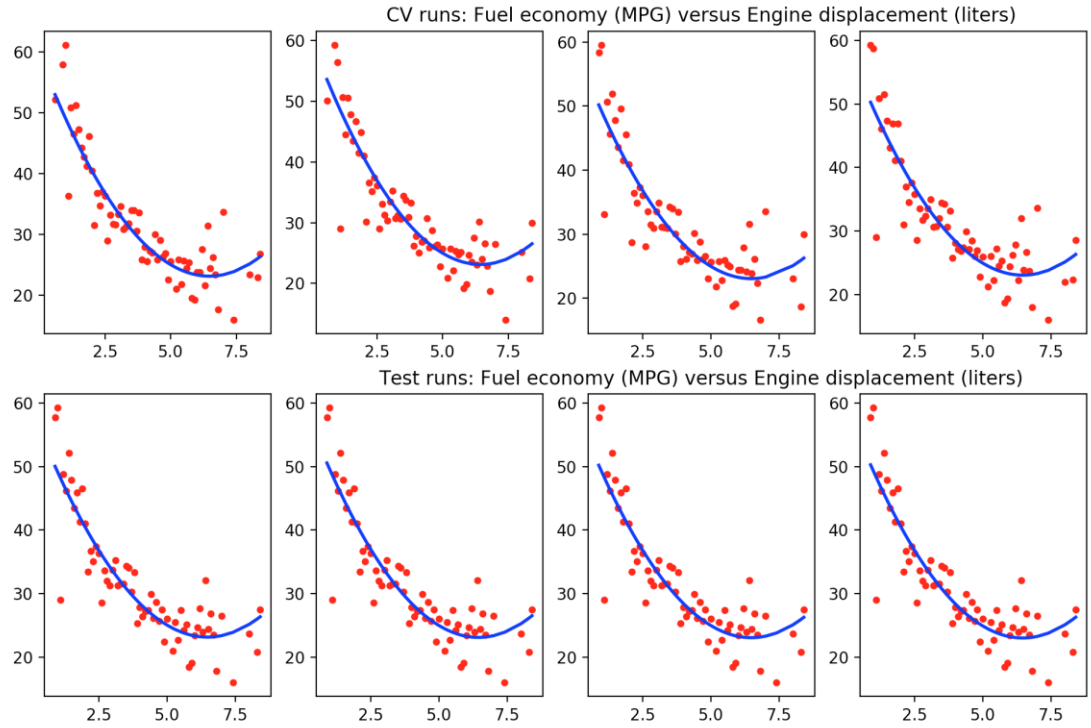


Figure 2.12 Plots showing each of the CV and test runs for the linear quadratic regression model.

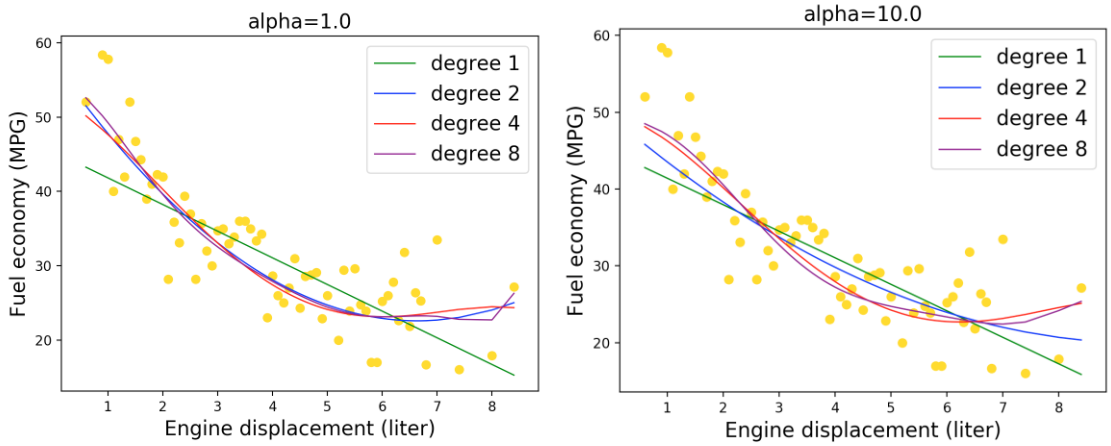


Figure 2.14 Ridge regularization applied to the fuel economy project with α = 1.0 & 10.0, respectively.

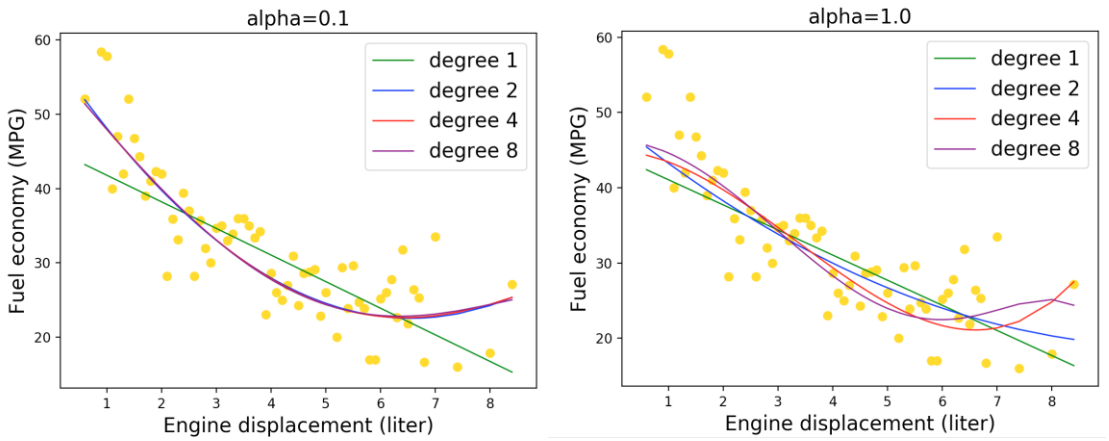


Figure 2.15 LASSO regularization applied to the fuel economy project with α = 0.1 & 1.0, respectively.

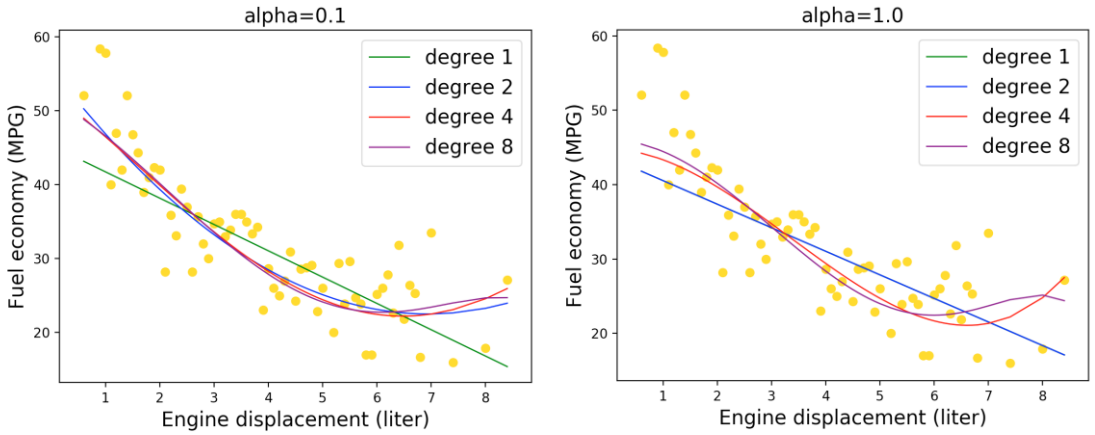


Figure 2.16 Elastic Net regularization applied to the fuel economy project with α = 0.1 and 1.0, respectively, while having l_1_ratio fixed to 0.7.

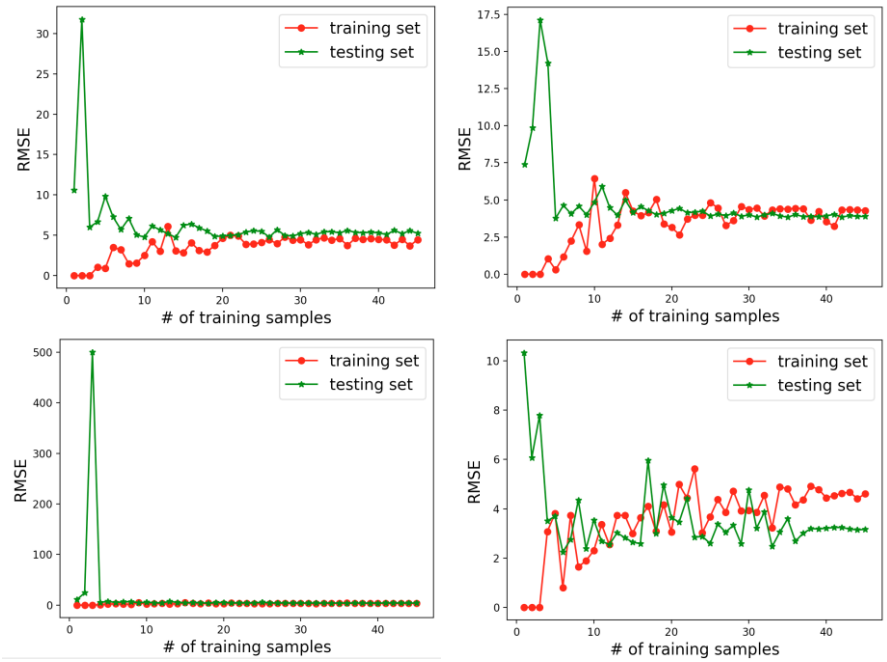
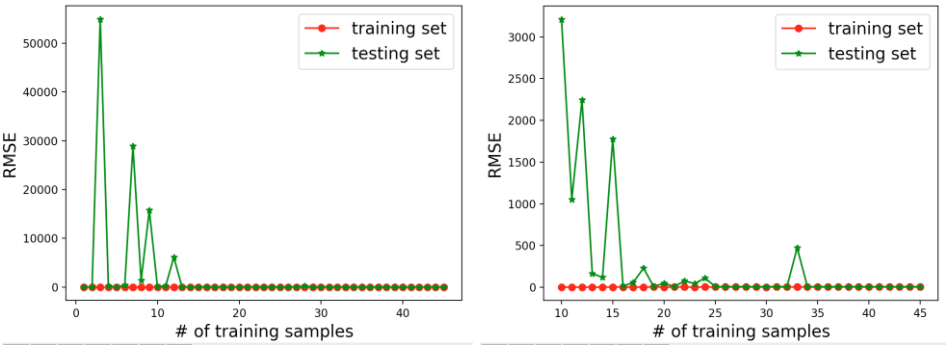


Figure 2.17 Learning curves from four consecutive runs for the linear regression model with `degree=2`.



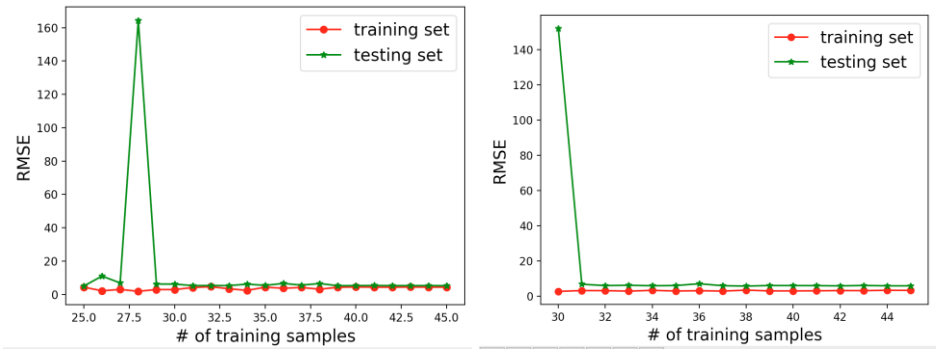


Figure 2.18 Learning curves from four consecutive runs for the linear regression model with `degree=8`.

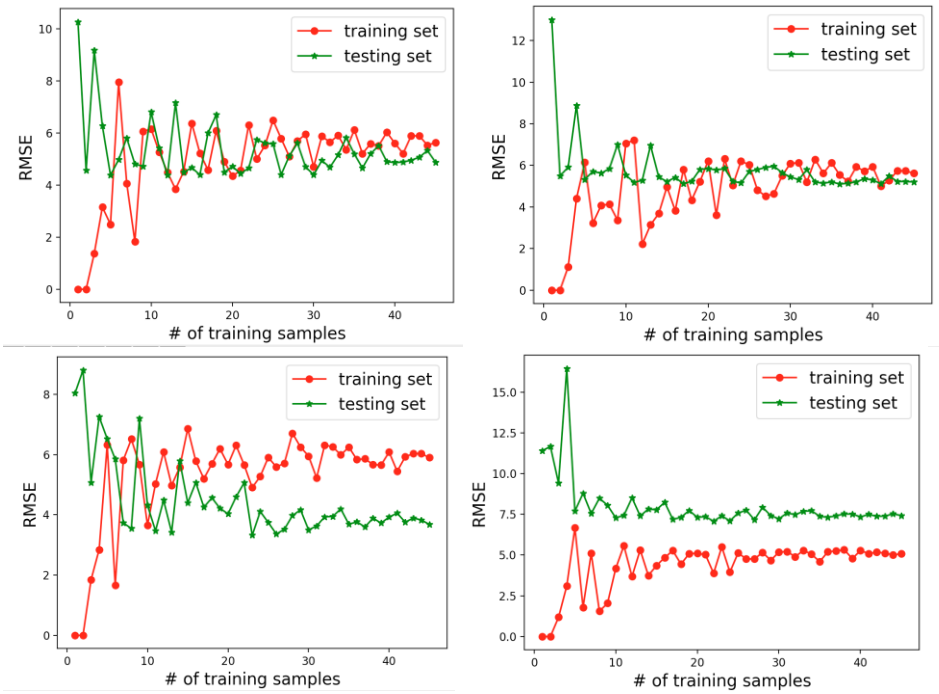


Figure 2.19 Learning curves from four consecutive runs for the linear regression model with `degree=1`.

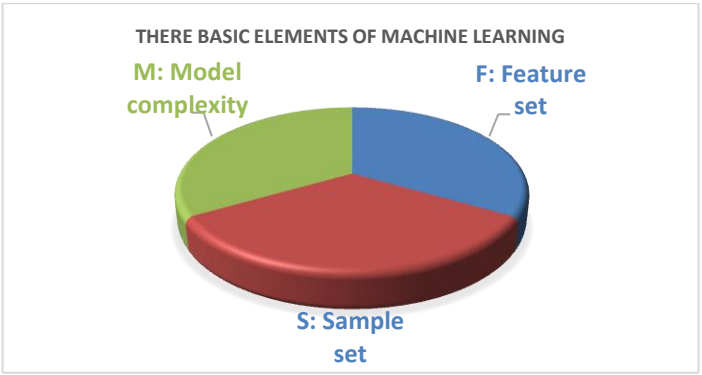


Figure 2.20 Three basic elements of machine learning.

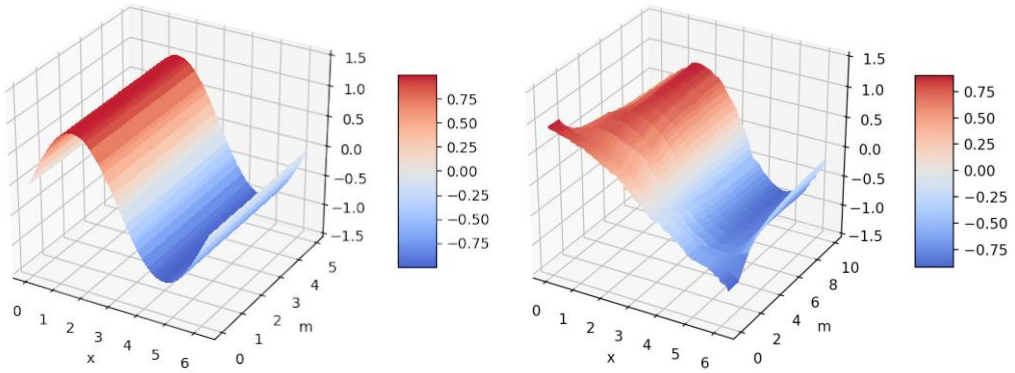


Figure 2.21 Left: Ideal case with a true sinusoidal distribution; Right: fitted/predicted distributions.

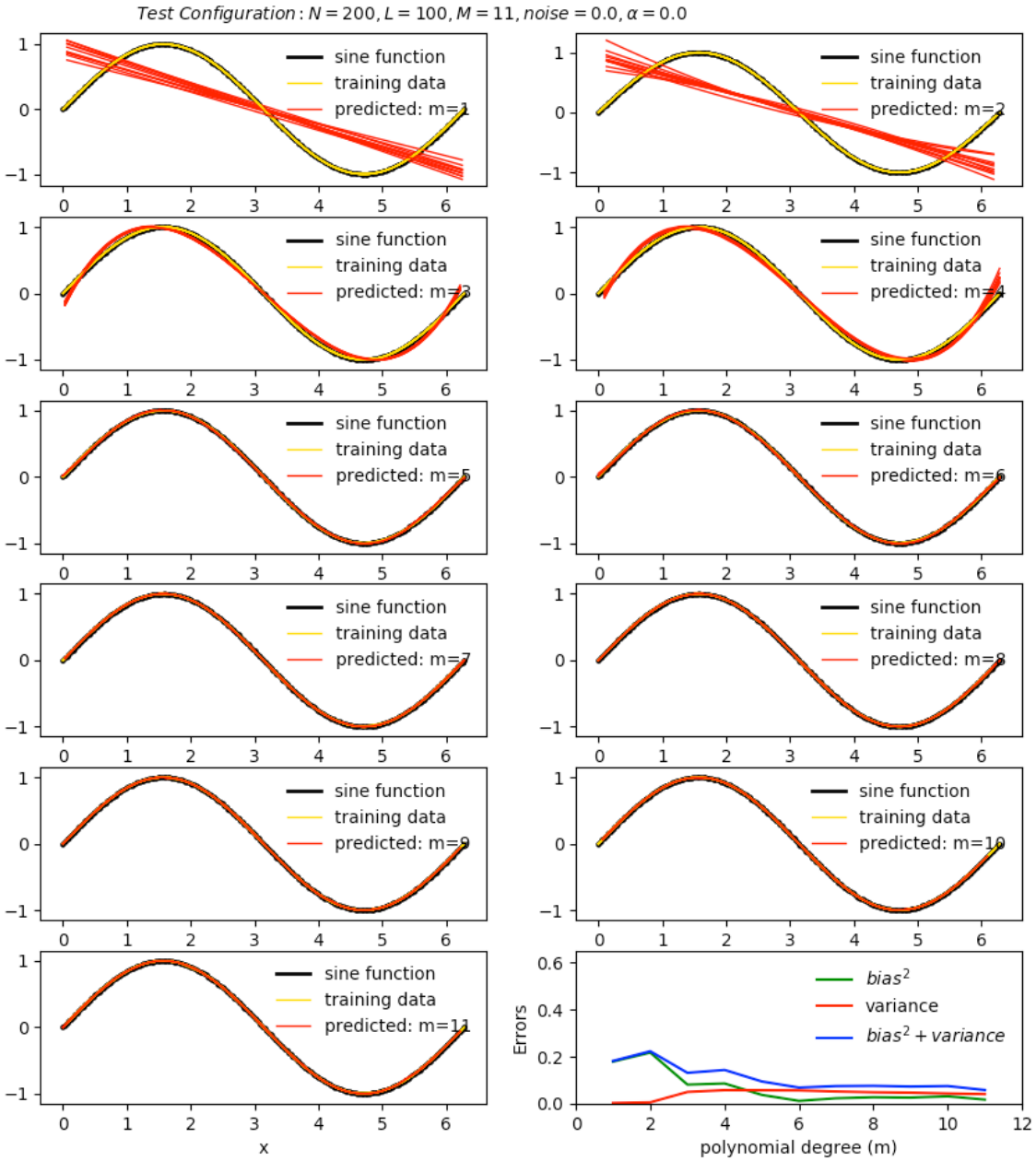


Figure 2.22 Bias-variance trade-off with linear polynomial fitting with a true sinusoidal distribution with (N=100/100, L=100, M =11, noise level = 0, and a fixed $\alpha = 0.0$ for Ridge regularization).

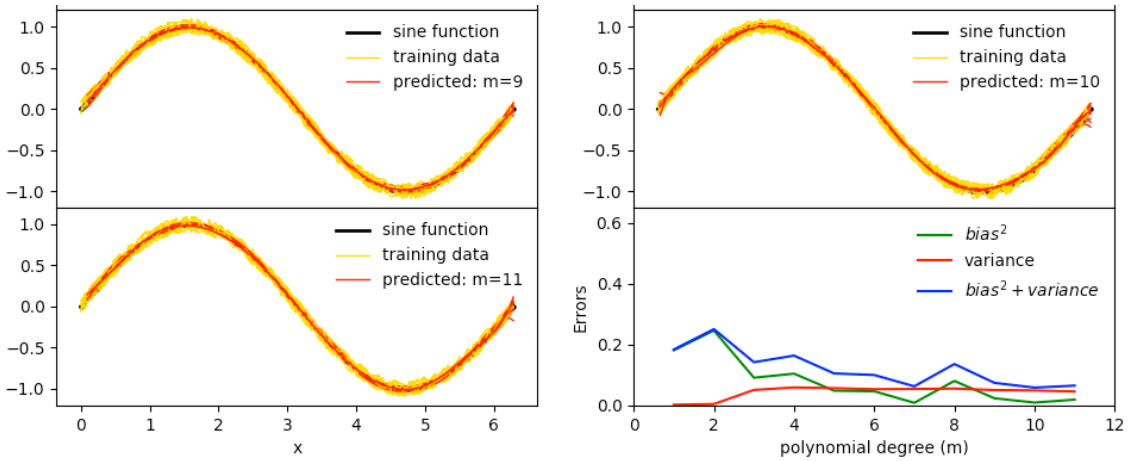


Figure 2.23 Linear polynomial fitting with a true sinusoidal distribution with (N=100/100, L=100, M =11, noise level = 0.1 or ±10%, and a fixed α = 0.0 for Ridge regularization).

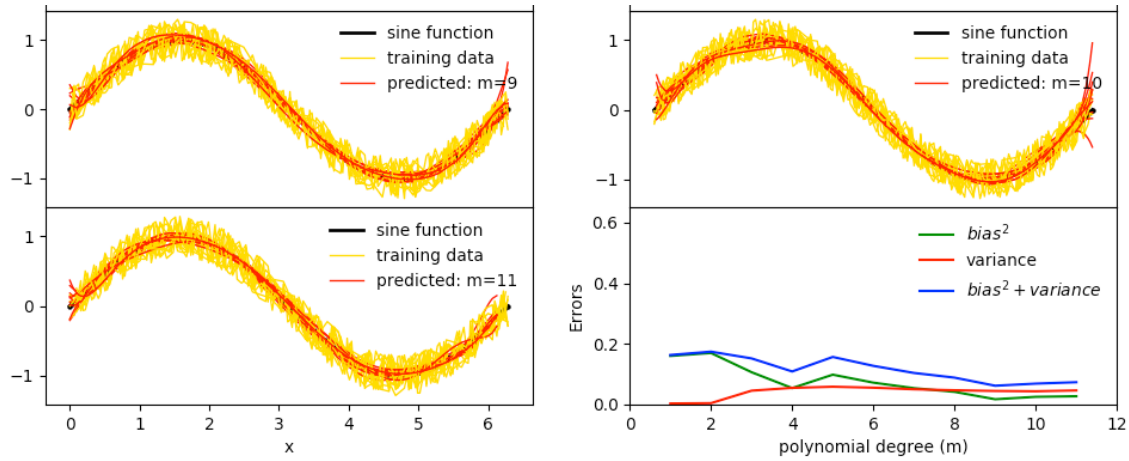


Figure 2.24 Linear polynomial fitting with a true sinusoidal distribution with (N=100/100, L=100, M =11, noise level = 0.3 or ±30%, and a fixed α = 0.0 for Ridge regularization).

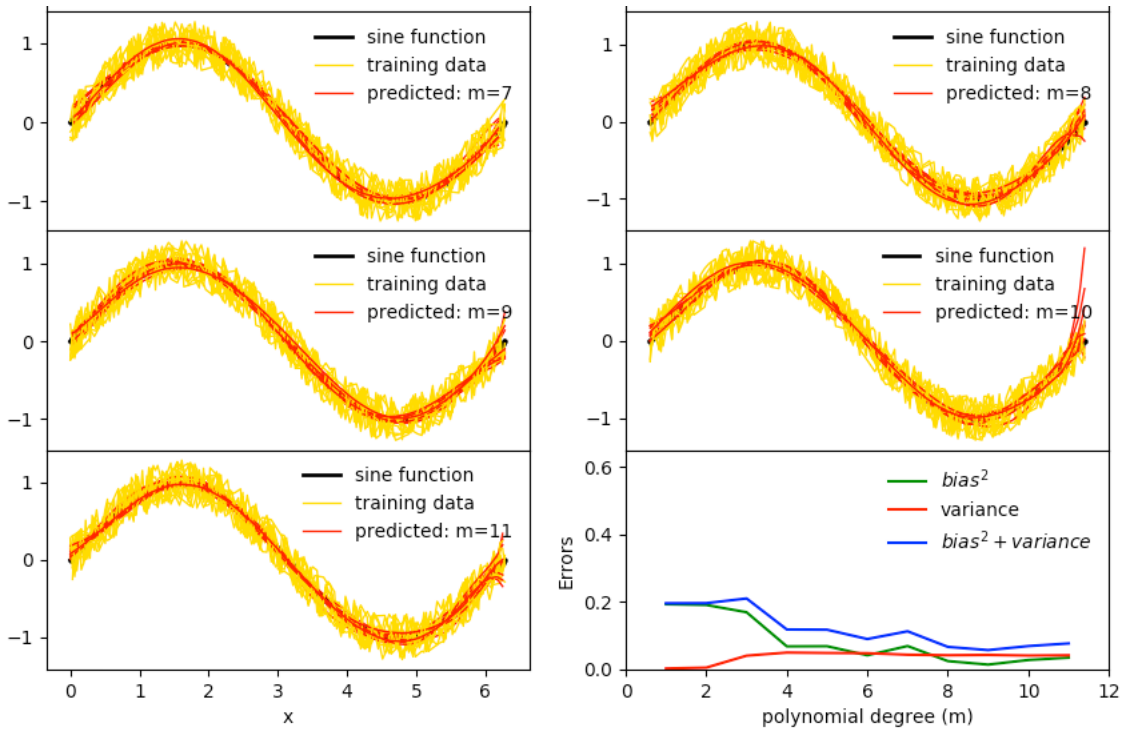


Figure 2.25 Linear polynomial fitting with a true sinusoidal distribution with (N=100/100, L=100, M =10, noise level = 0.3 or ±30%, and a fixed $\alpha = 0.1$ for Ridge regularization).

3 Pattern Recognition with Classification

Figure 3.1 100 randomly selected MNIST samples with target classes above each row.

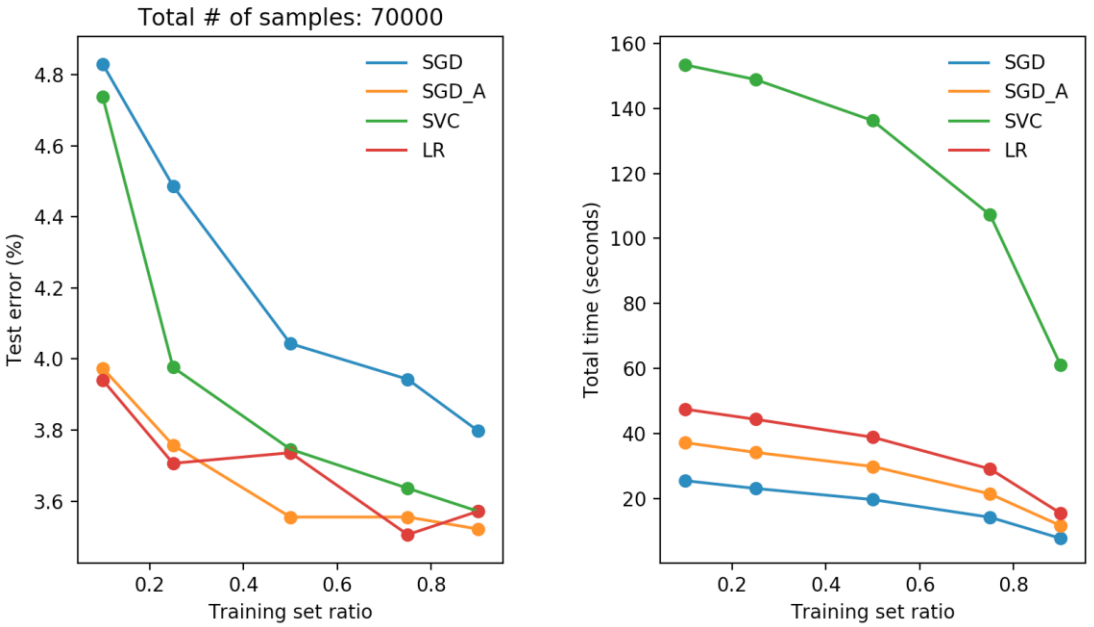
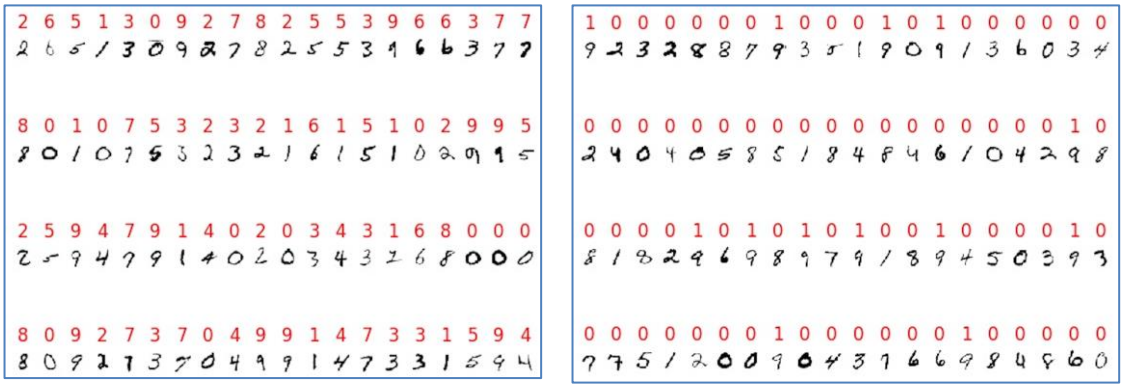


Figure 3.2 MNIST test results with linear binary classifiers.

3.1.1 Confusion Matrix



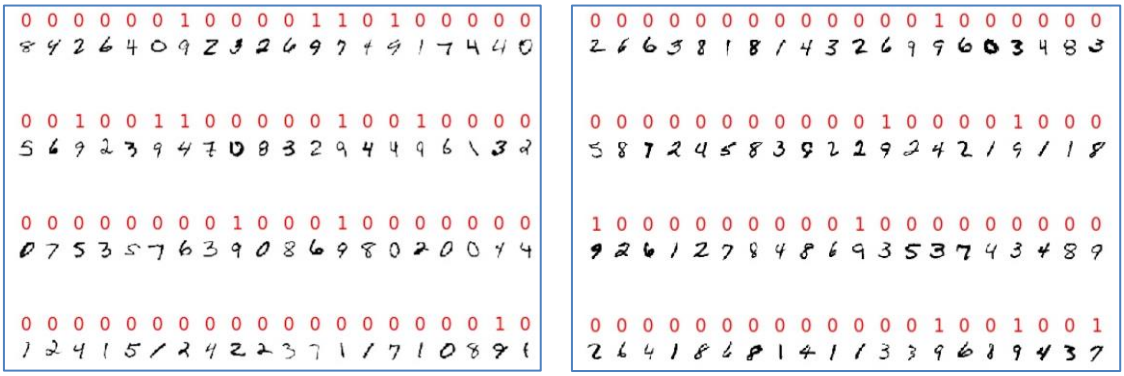


Figure 3.3 MNIST images classified by binary classifiers: Upper left: original testing data samples; upper right: binarized testing data samples; lower left and right: two examples of predictions.

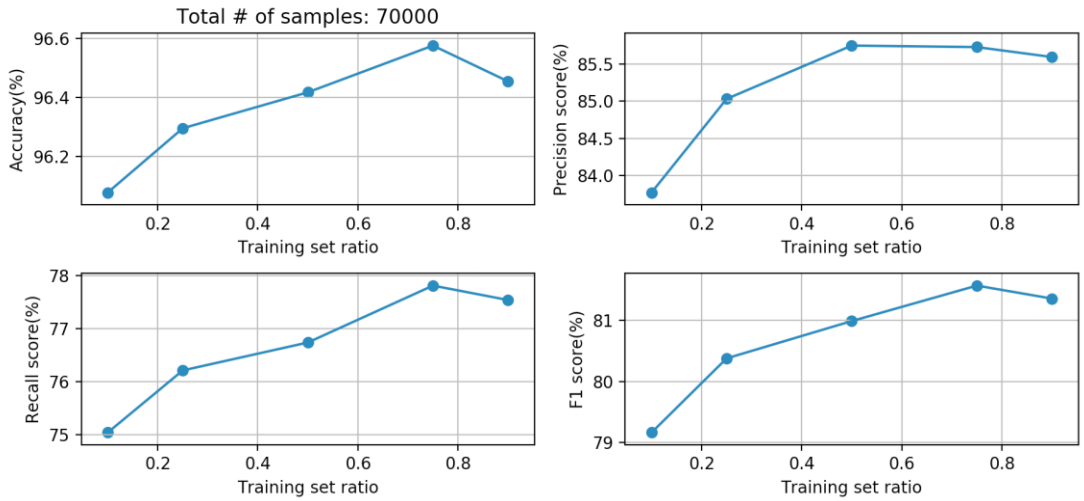


Figure 3.4 Various performance scores for the MNIST classification example of recognizing the digit of '9' obtained with using the classifier's `predict` function as a baseline.

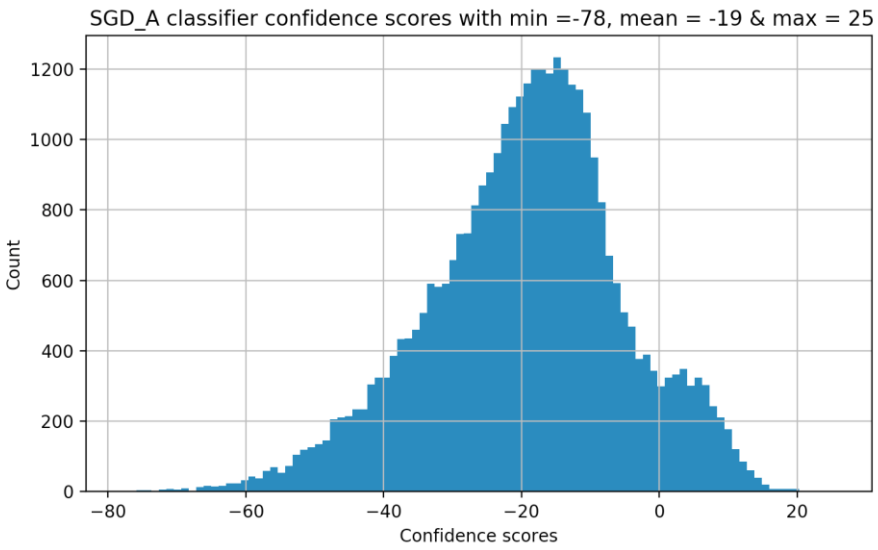


Figure 3.5 An example of confidence scores associated with the MNIST classification example of recognizing the digit of '9'.

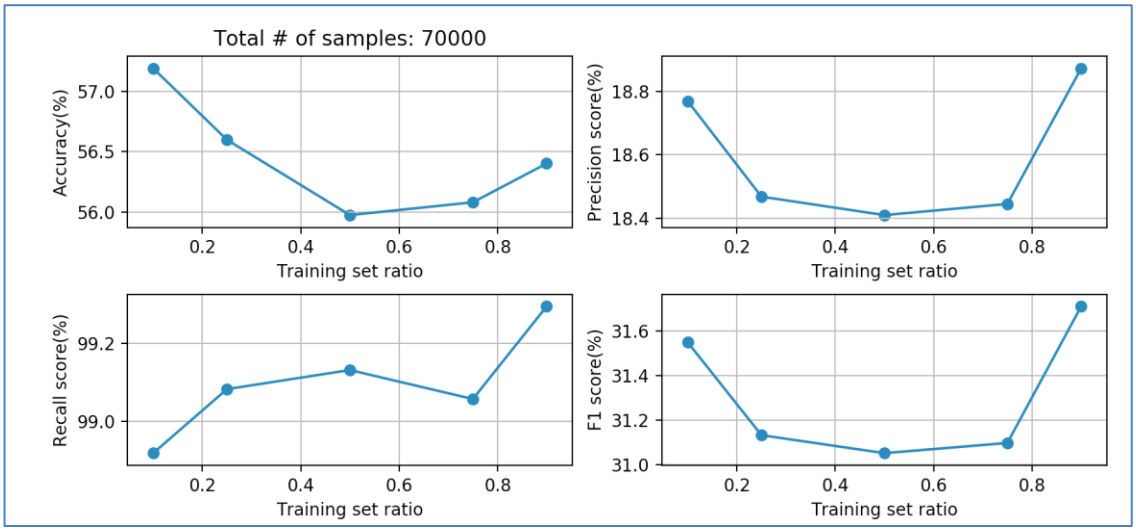


Figure 3.6 Various performance scores for the MNIST classification example of recognizing the digit of '9' obtained with using the pre-defined threshold set to the mean value of -19.

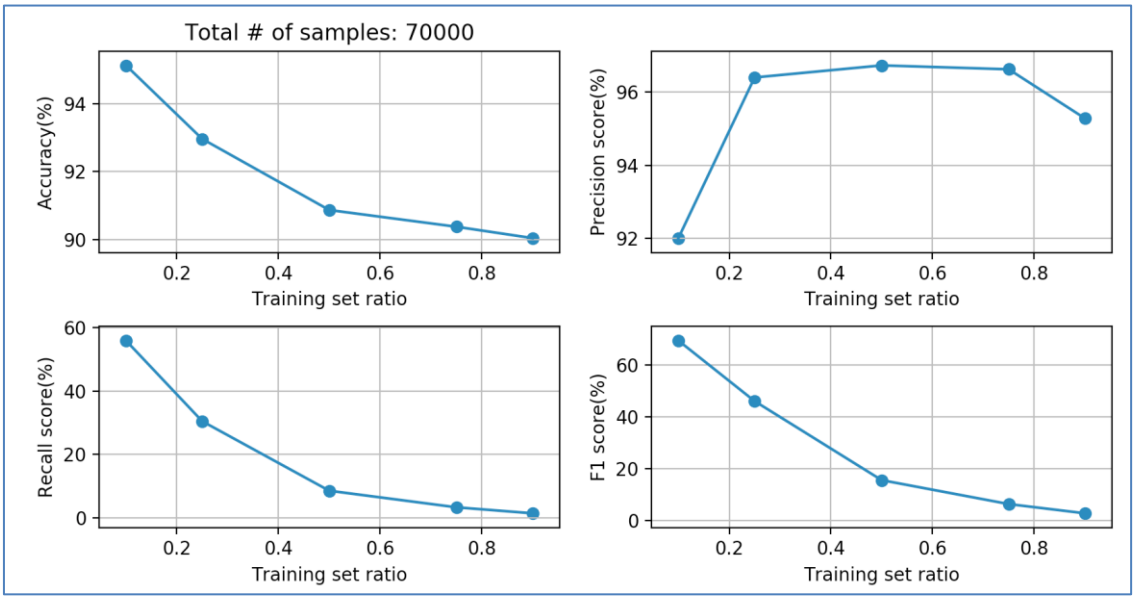


Figure 3.7 Various performance scores for the MNIST classification example of recognizing the digit of '9' obtained with using the pre-defined threshold set to the mean value of 10.

3.1.2 Precision and recall trade-off

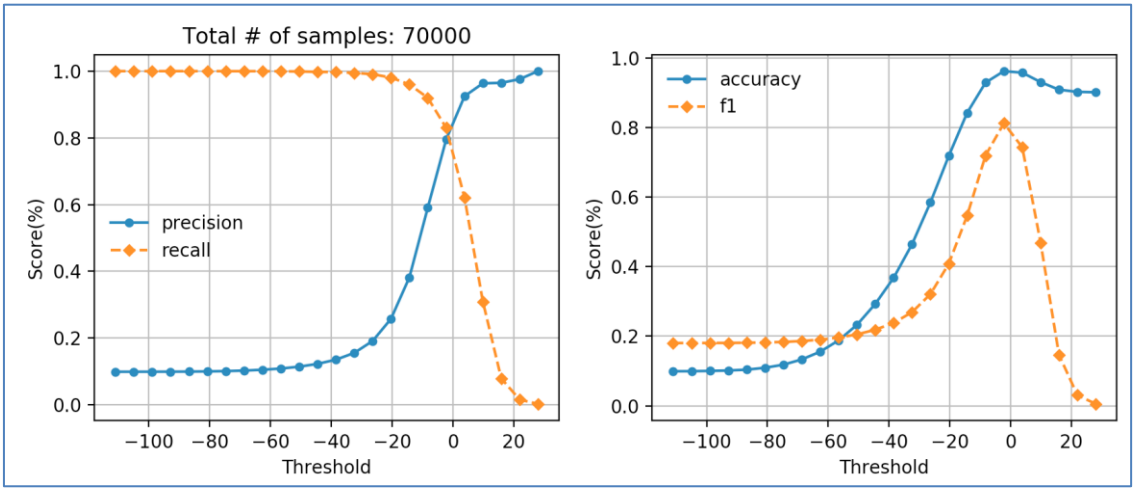


Figure 3.8 Precision-recall trade-off by sliding the threshold value from left to right.

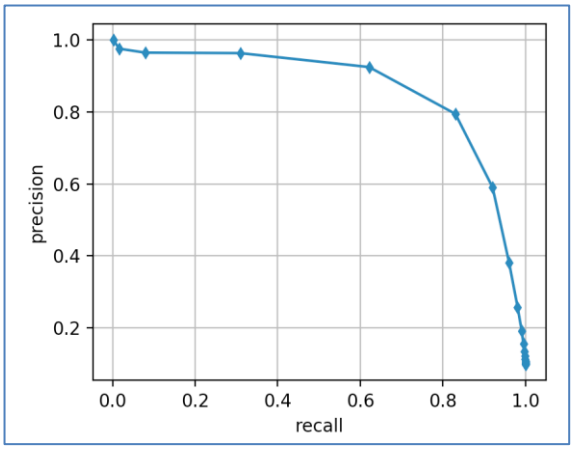


Figure 3.9 Variations of the precision with the recall as the threshold value varied from low to high.

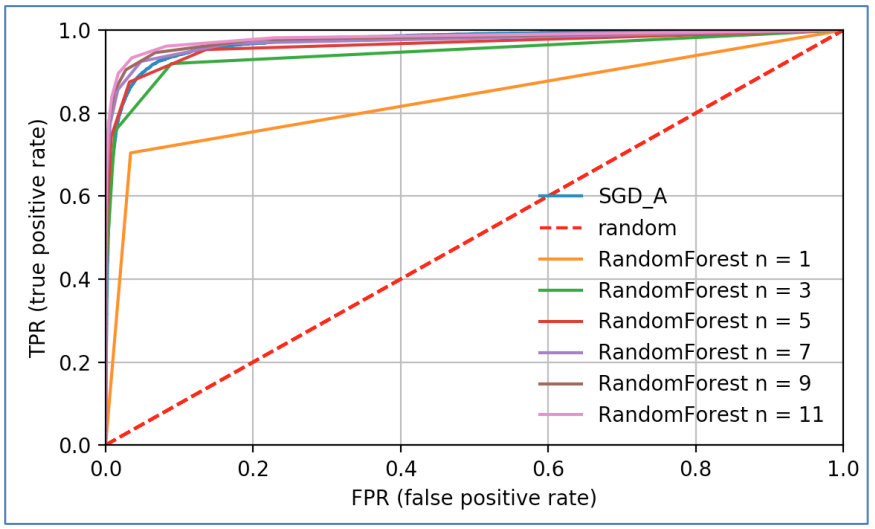


Figure 3.10 The ROC curves for the `SGDClassifier` and `RandomForestClassifier`.

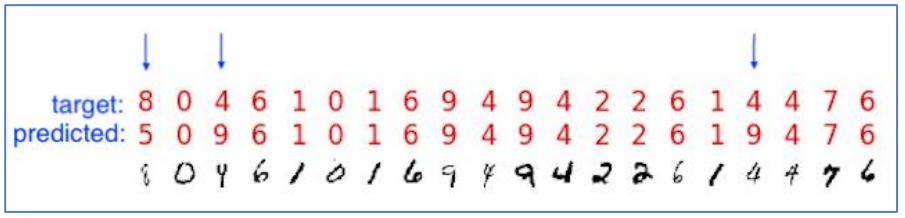


Figure 3.11 Multiclass classification with the `RandomForestClassifier`.

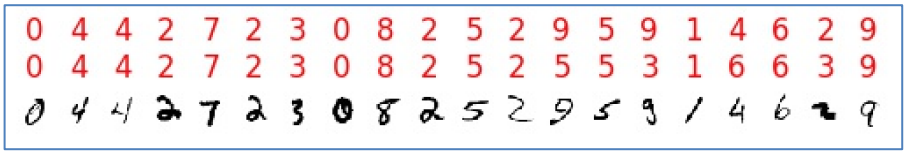


Figure 3.12 Another run of multiclass classification with the `RandomForestClassifier`.

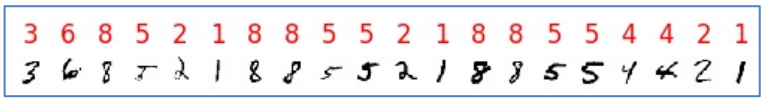


Figure 3.13 Outcome of running the `RandomForest` multiclass classifier, with the class probabilities shown in Listing 3.10.

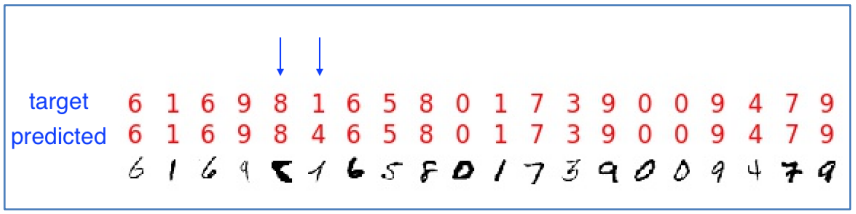


Figure 3.14 Outcome of a multiclass classification run with 20 samples.

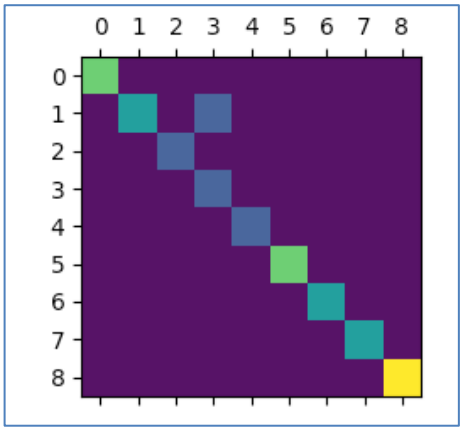


Figure 3.15 The confusion matrix map drawn with the `matshow` function.

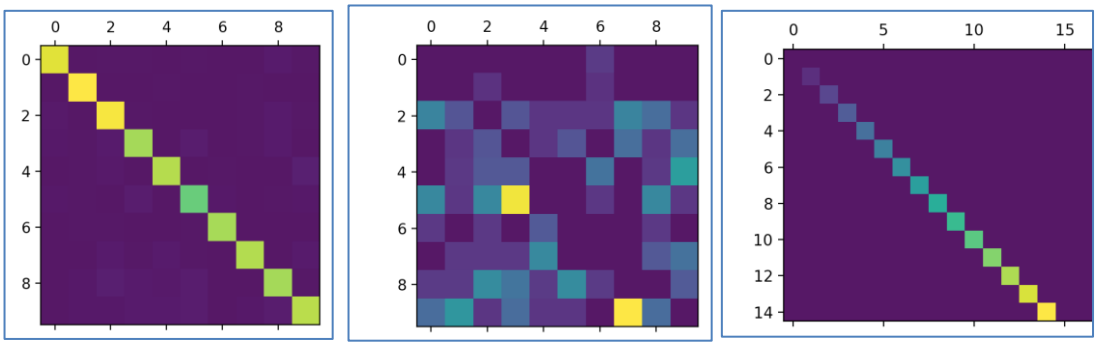


Figure 3.16 Original confusion matrix (left) and normalized confusion matrix with diagonal elements zeroed out (middle), with the right-most plot as a reference of color in increasing intensity.

```
[[ 0.     0.     0.     0.     0.     0.     0.005  0.     0.     0.   ]
 [ 0.     0.     0.004  0.     0.     0.     0.004  0.     0.     0.   ]
 [ 0.019  0.01   0.     0.01   0.005  0.005  0.005  0.019  0.015  0.005]
 [ 0.     0.005  0.01   0.     0.005  0.01   0.     0.015  0.005  0.015]
 [ 0.     0.005  0.01   0.01   0.     0.     0.016  0.     0.005  0.026]
 [ 0.021  0.005  0.021  0.051  0.     0.     0.005  0.     0.021  0.005]
 [ 0.005  0.     0.005  0.     0.011  0.     0.     0.     0.005  0.   ]
 [ 0.     0.005  0.005  0.005  0.021  0.     0.     0.     0.01   0.016]
 [ 0.005  0.005  0.022  0.016  0.005  0.022  0.005  0.     0.     0.011]
 [ 0.014  0.024  0.005  0.014  0.005  0.005  0.     0.053  0.014  0.   ]]
```

Figure 3.17 Normalized confusion matrix with diagonal elements zeroed out.

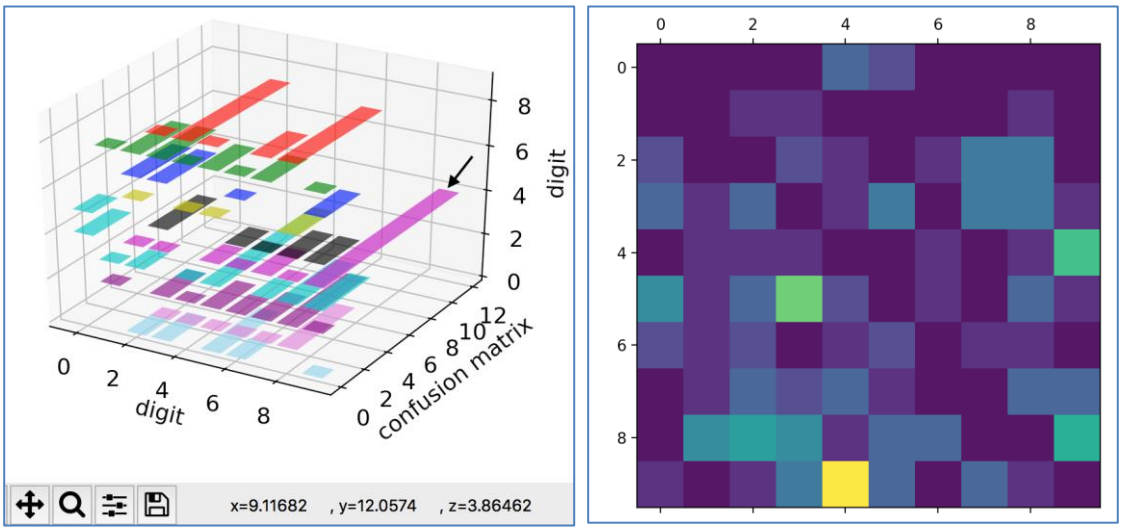


Figure 3.18 3D (left) representation of the same 2D (right) confusion matrix.

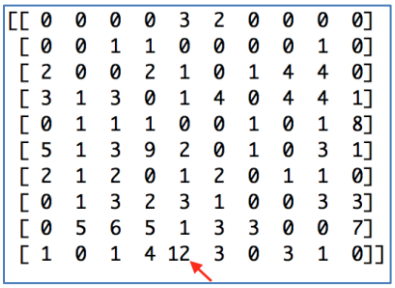


Figure 3.19 Confusion matrix in numerical format

4 Optimization and Search Illustrated with Logistic Regression

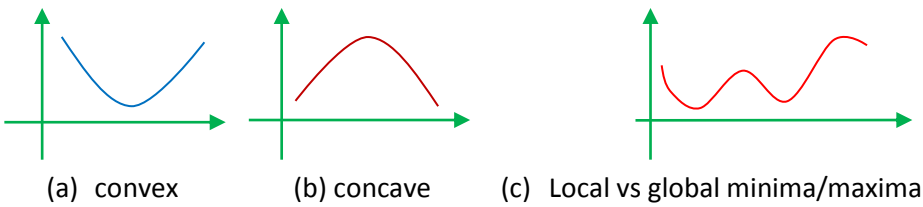


Figure 4.1 (a) Convex, (b) concave, and (c) non-convex, non-concave functions.

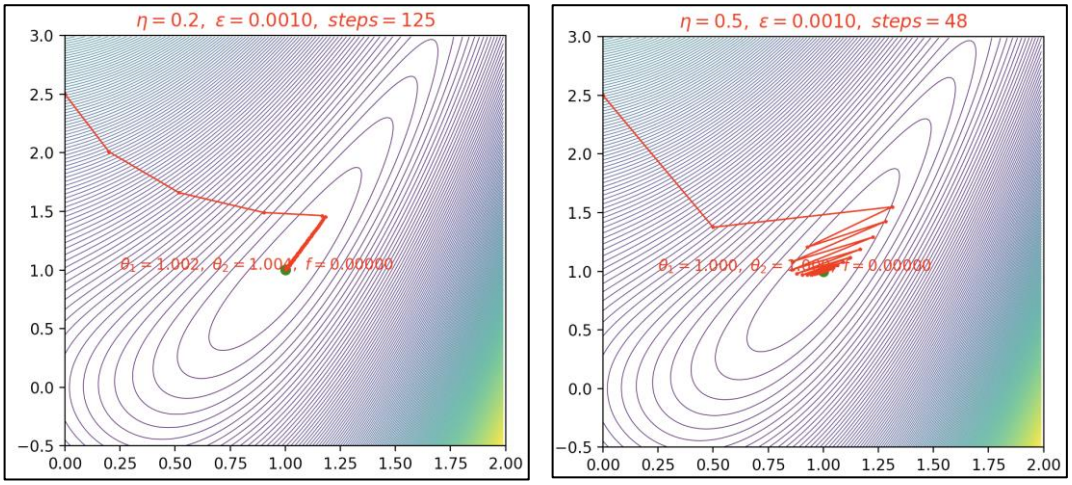


Figure 4.2 Gradient descent optimization on the known function expressed in Eq. (4.1): with two different learning rates of 0.2 (left) and 0.5 (right), respectively.

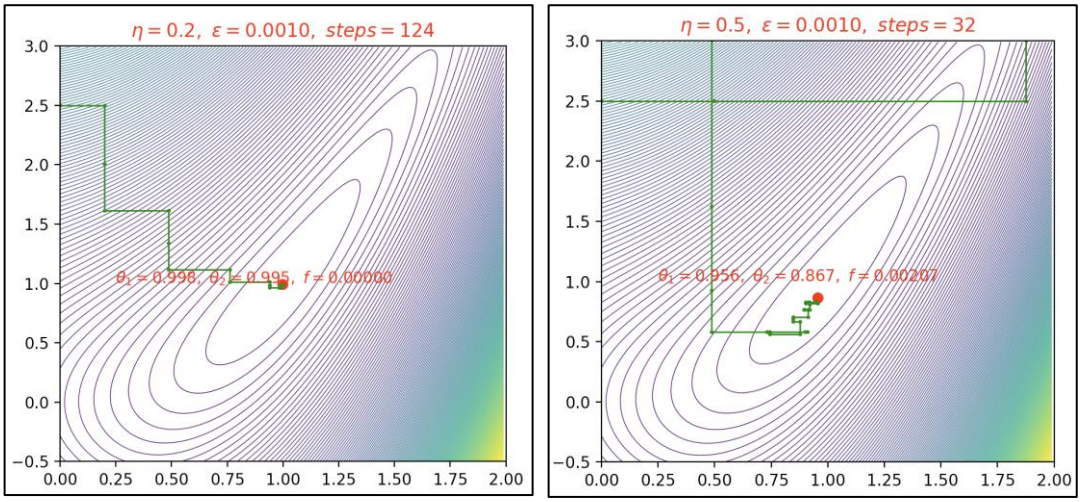


Figure 4.3 Stochastic gradient descent (SGD) optimization on the known function expressed in Eq. (4.1): with two different learning rates of 0.2 (left) and 0.5 (right), respectively.

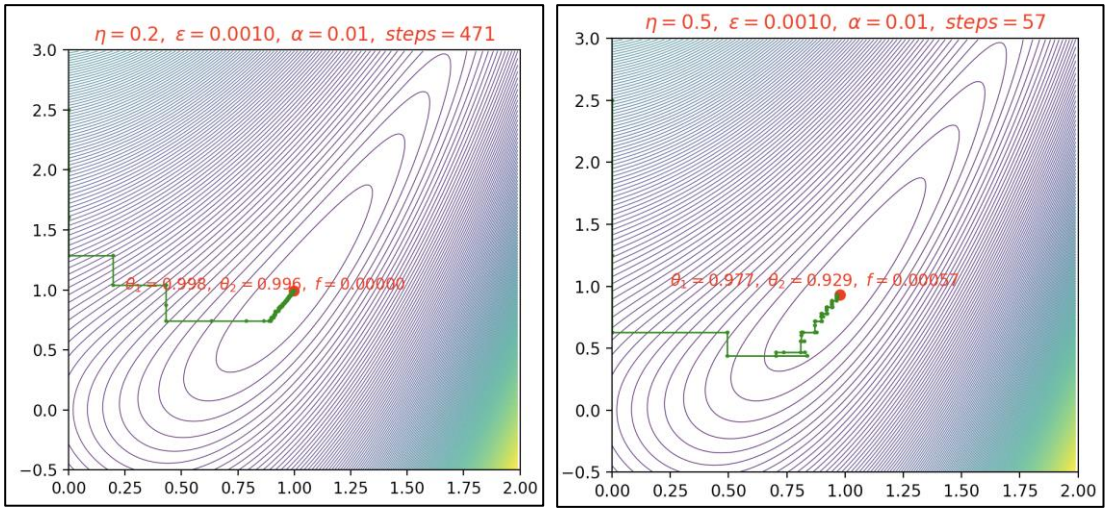


Figure 4.4 Stochastic gradient descent with a learning scheduler.

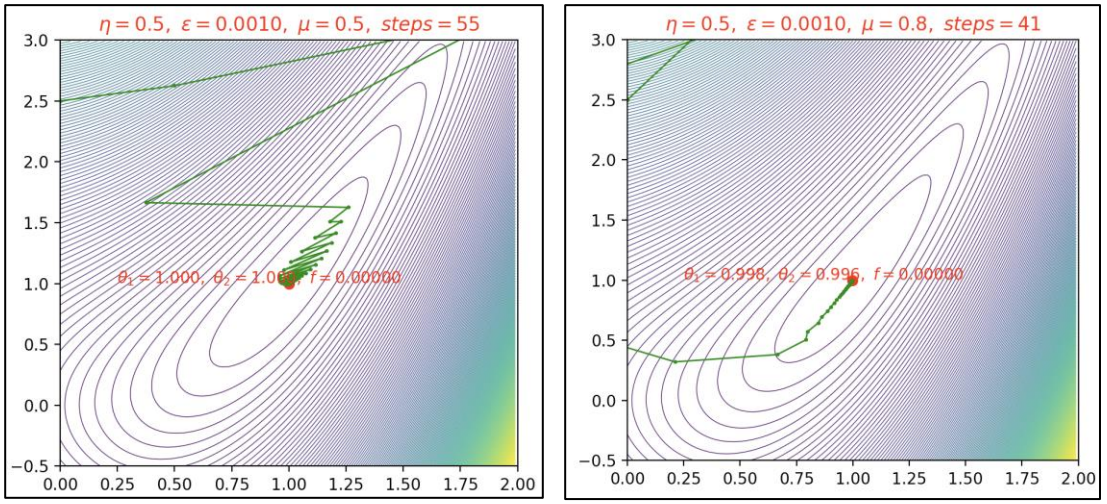


Figure 4.5 Efficacy with the heavy ball method for controlling the zig-zag behavior of gradient descent optimization.

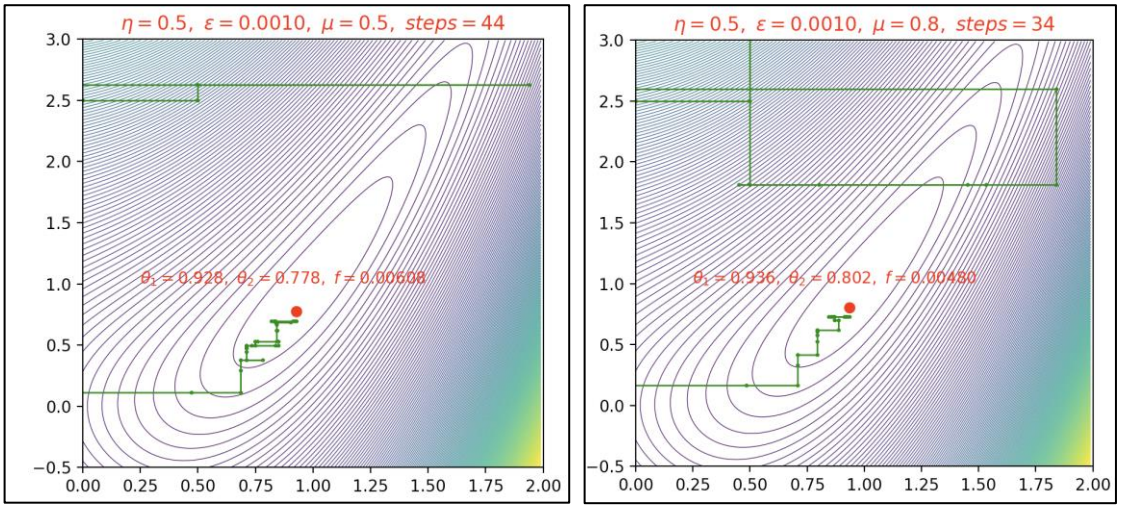


Figure 4.6 Efficacy with the heavy ball method for controlling the zig-zag behavior of the *stochastic* gradient descent optimization.

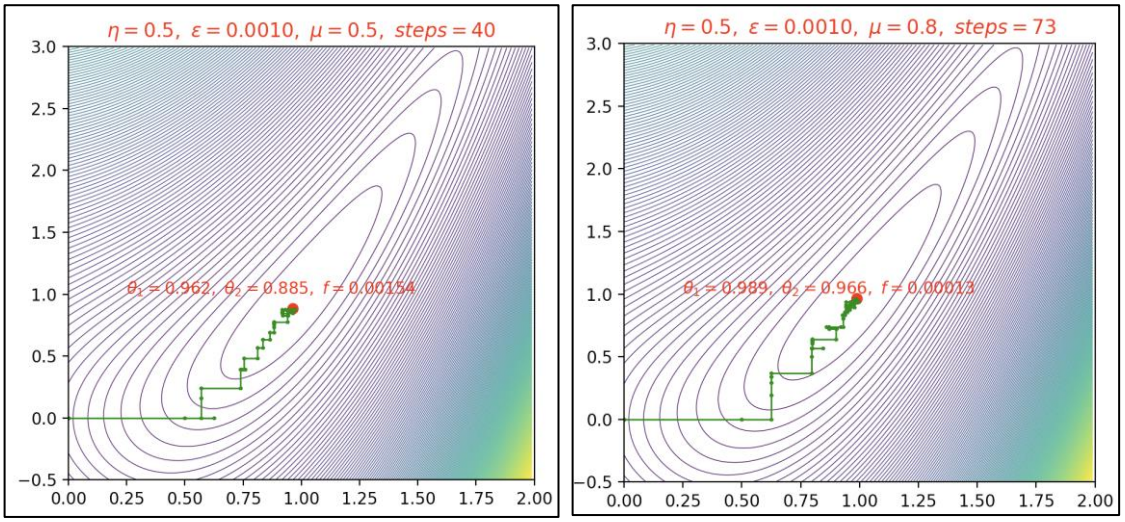


Figure 4.7 Efficacy with the heavy ball method for controlling the zig-zag behavior of the *stochastic* gradient descent optimization with both θ_1 and θ_2 started at (0, 0).

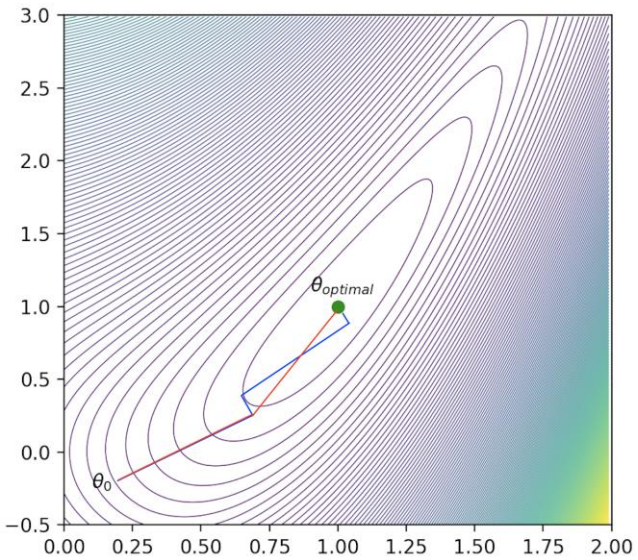


Figure 4.8 Conjugate gradient search: the blue line represents the gradient descent with perpendicular moves at each step, while the red line represents the conjugate gradient moves that are not perpendicular but take at most *n* steps where *n* is the dimension of the parameter space.

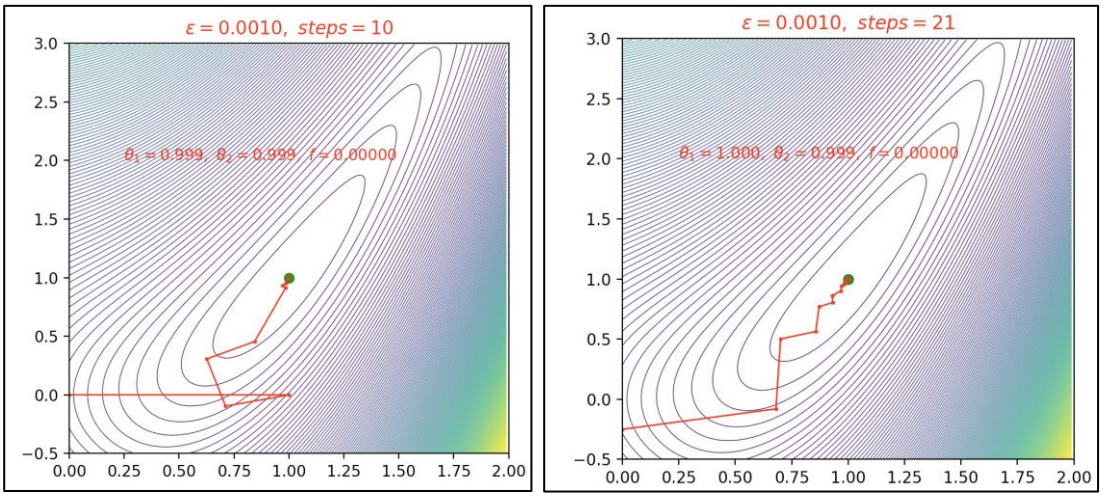


Figure 4.9 Two example runs with the conjugate gradient optimization with two different start points of (0, 0) (left) and (0, -0.25) (right).

4.1 SEARCH

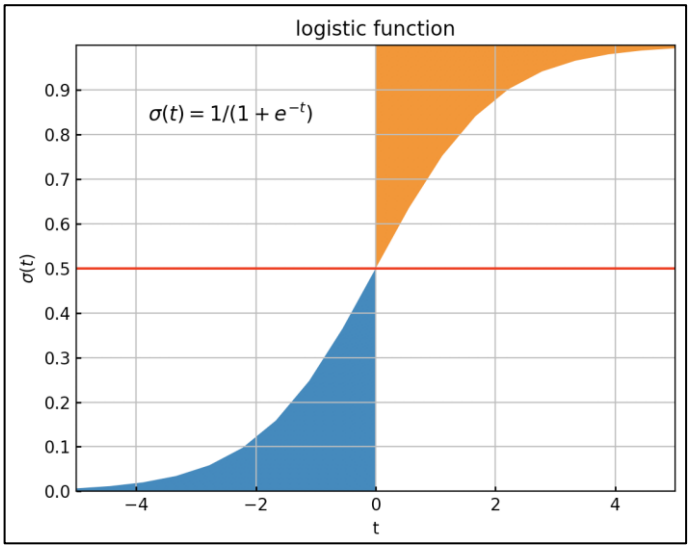


Figure 4.10 Logistic function.

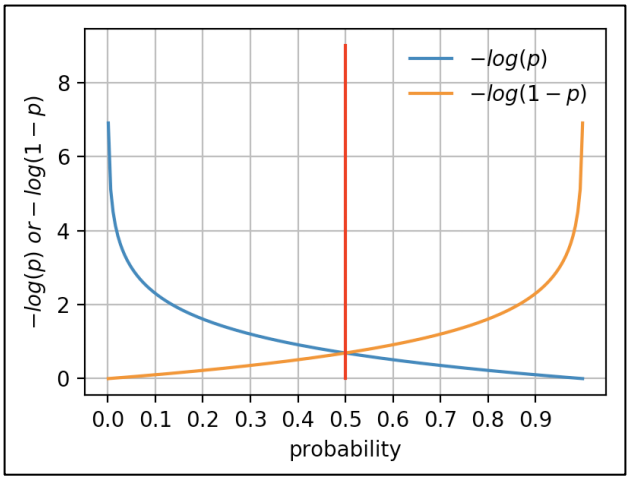


Figure 4.11 Cost (loss) functions for the logistic function based binary classifier.

(a) *iris versicolor* (b) *iris virginica* (c) *iris setosa*

Figure 4.12 Iris flowers.

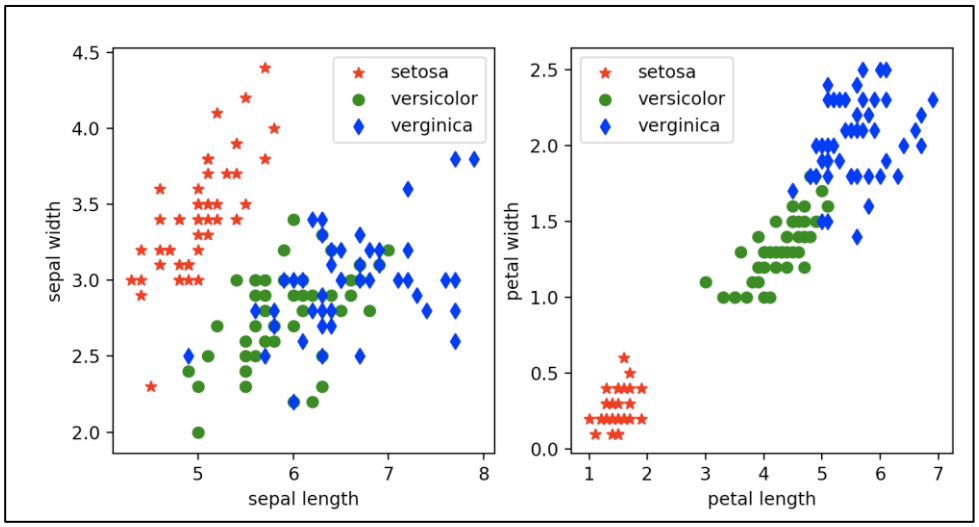


Figure 4.13 Iris flower dataset sepal length versus sepal width (left) and petal length versus petal width (right).

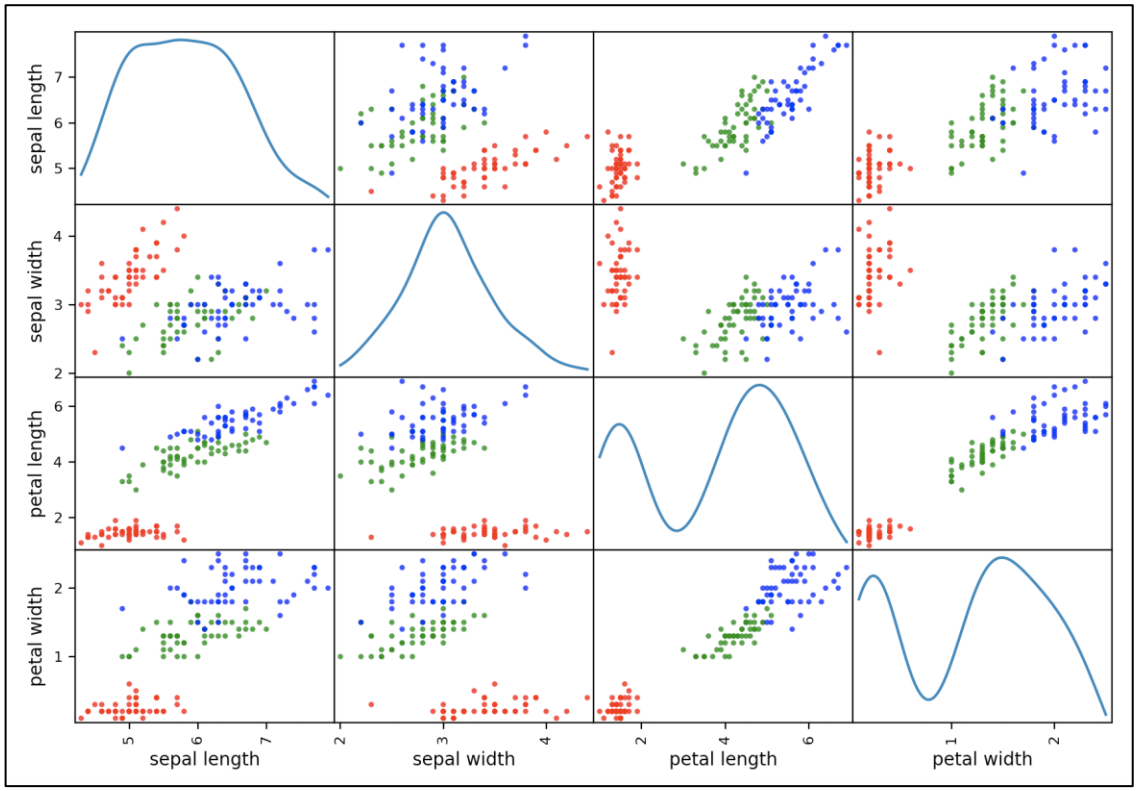


Figure 4.14 Scatter matrix of the iris flower dataset.

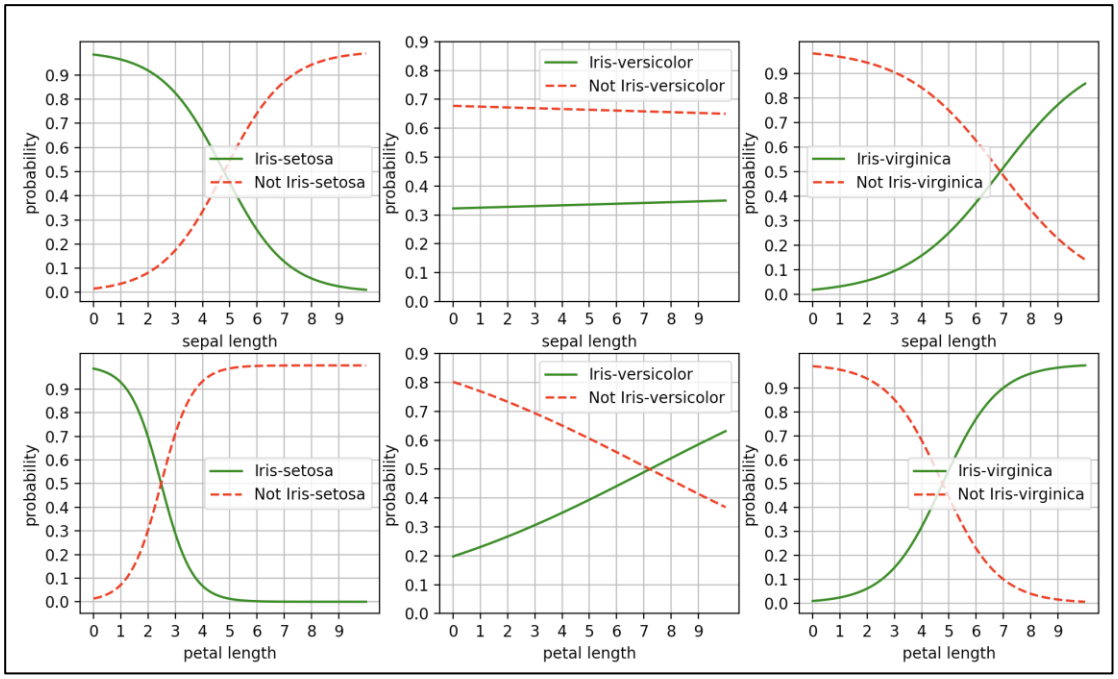


Figure 4.15 Logistic probabilities with the three iris classes as classified by sepal length and petal length.

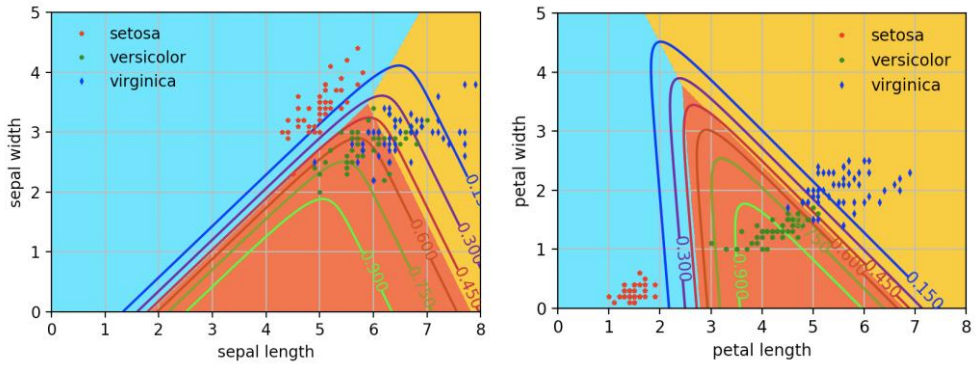


Figure 4.16 Softmax regression applied to the iris dataset multiclass classification: sepal length versus sepal width (left) and petal length versus petal width (right).

5 Rule-Based Learning: Decision Trees

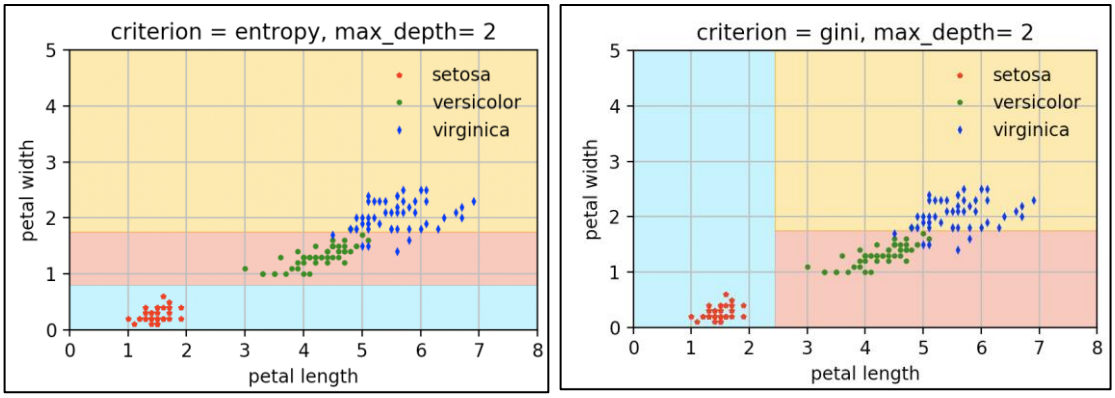


Figure 5.1 Iris dataset classification using Quinlan's tree model and CART with `max_depth` = 2.

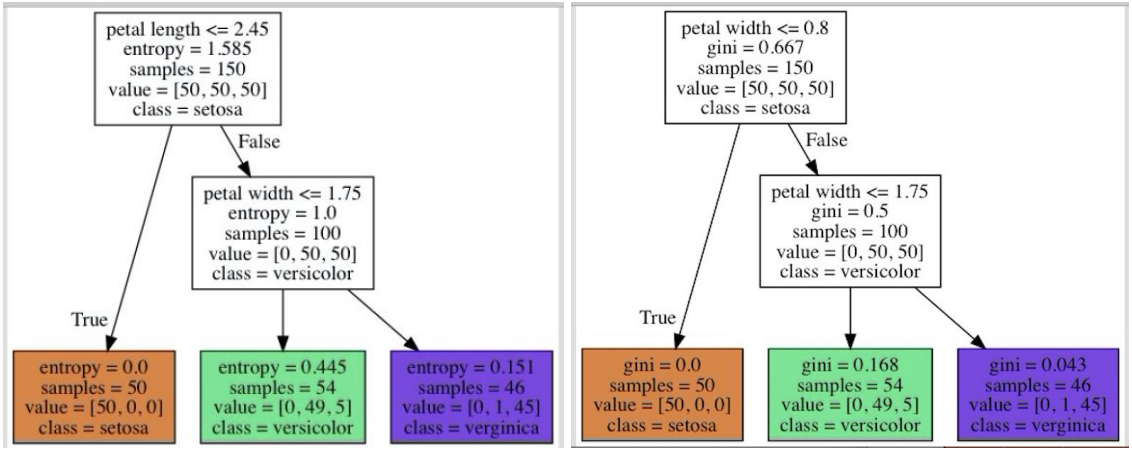
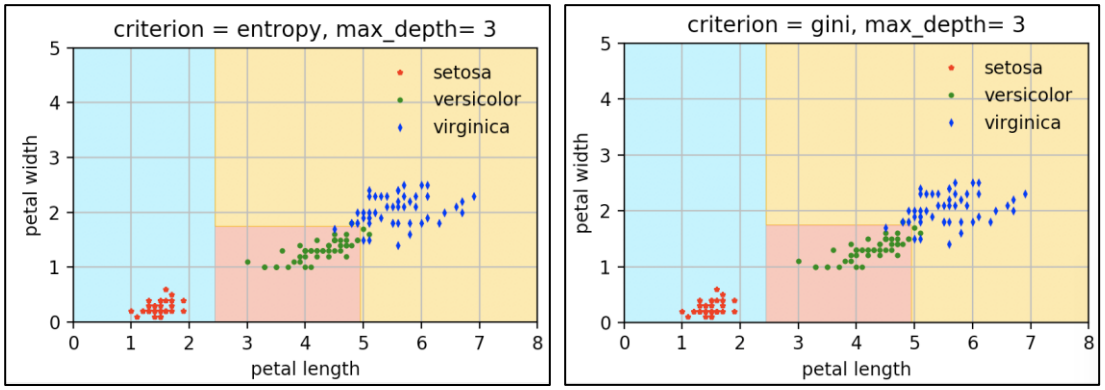


Figure 5.2 Graphical representations of the tree models corresponding to the classification results shown in Figure 5.1, based on entropy and Gini impurity criterions, respectively.



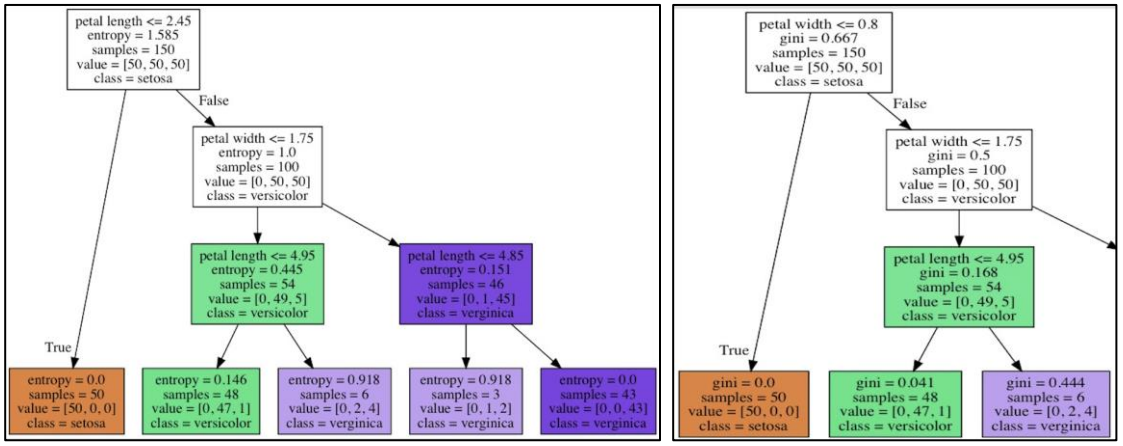


Figure 5.3 Iris dataset classification using Quinlan's tree model and CART with `max_depth` = 3 (gini model shown partially due to space limitation).

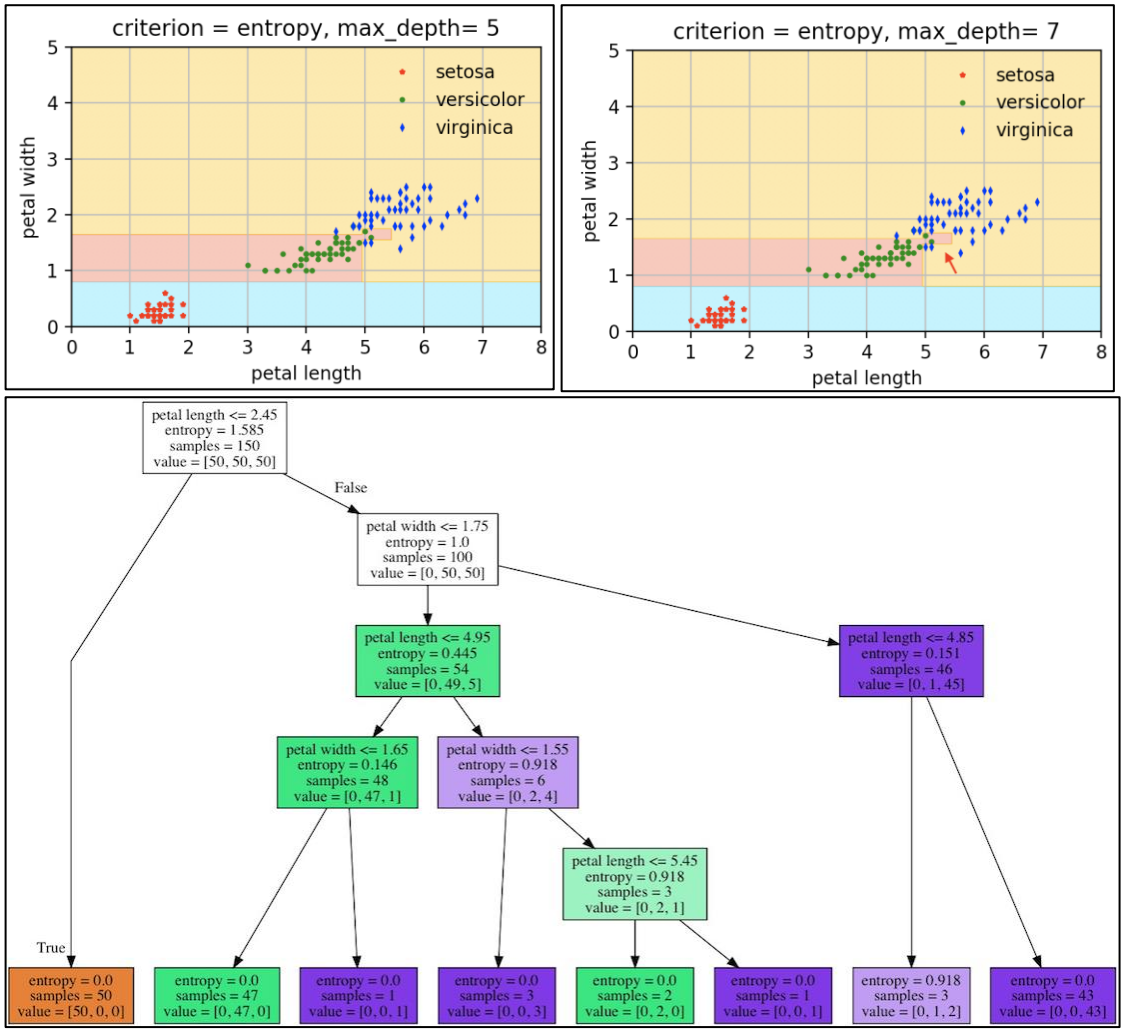


Figure 5.4 Iris dataset classification using Quinlan's tree model with `max_depth` = 5 and 7, respectively.

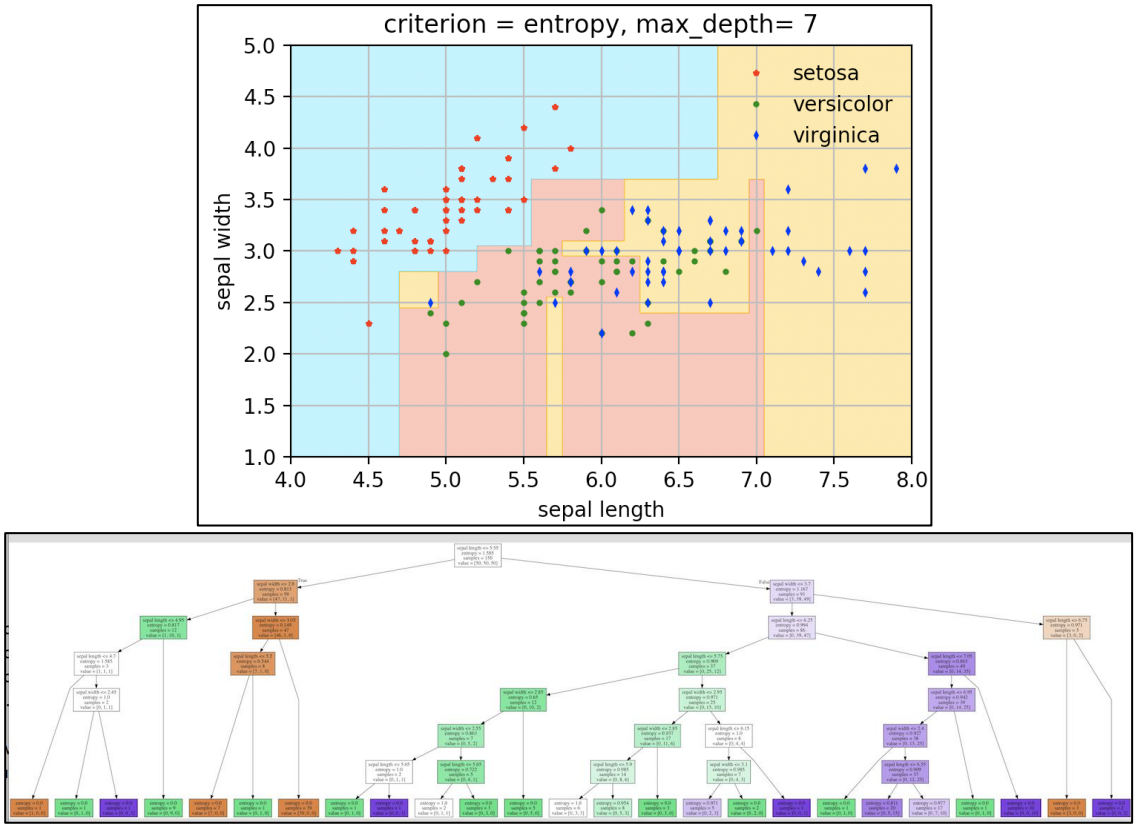


Figure 5.5 Iris dataset classification using a decision tree model with the feature pair of (sepal length, sepal width).

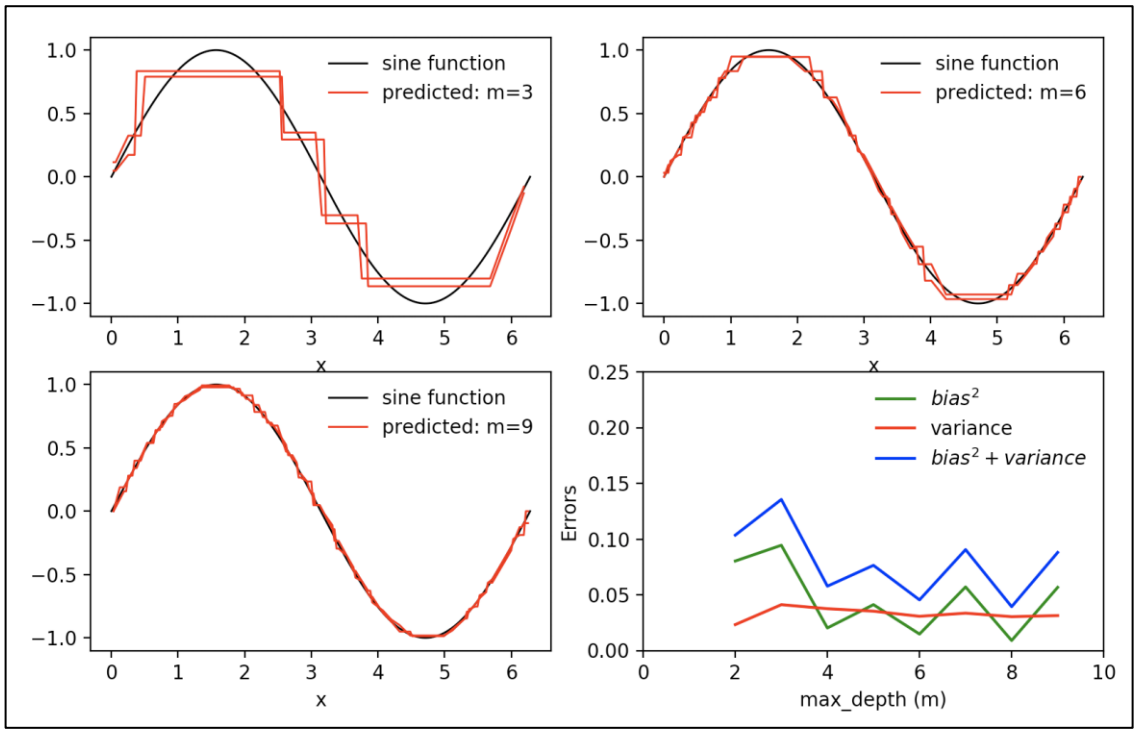


Figure 5.6 An ideal sine-function fitted with the `sklearn.tree.DecisionTreeRegressor`, with the `max_depth` hyper-parameter (`m`) varied from 2 to 9.

6 Instance-Based Learning: Support Vector Machines

6.1 SVM FUNDAMENTALS

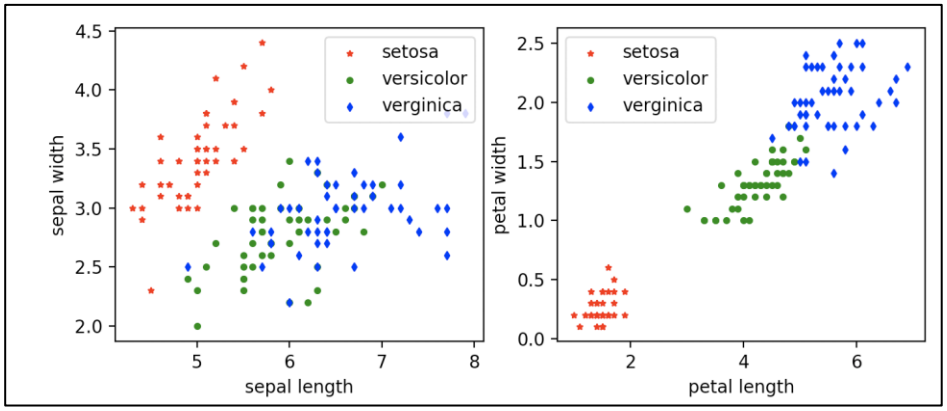


Figure 6.1 Iris dataset: sepal width versus sepal length (left) and petal width versus petal length (right).

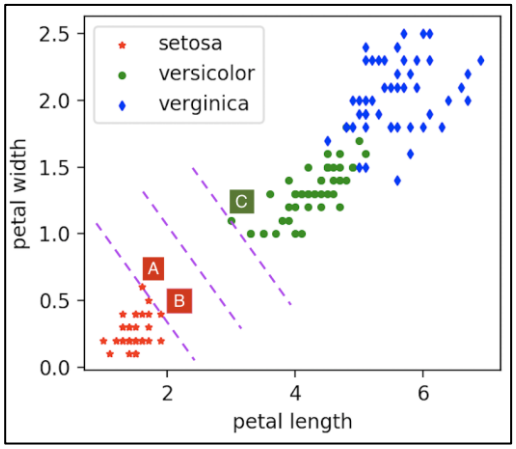


Figure 6.2 Concept of support vectors.

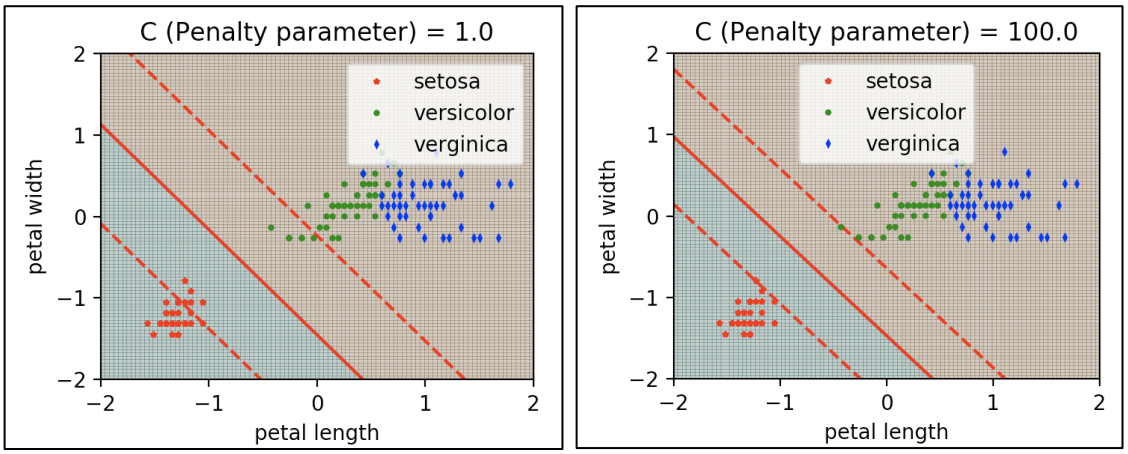


Figure 6.3 LinearSVC applied to the iris dataset petal features with two different penalty values of 1.0 (default) and 100.

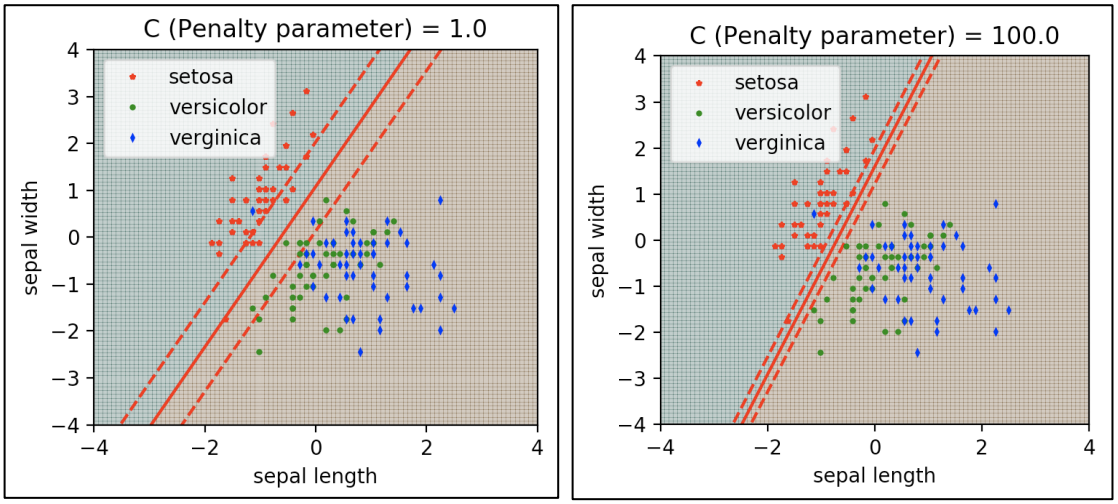


Figure 6.4 LinearSVC applied to the iris dataset sepal features with two different penalty values of 1.0 (default) and 100.

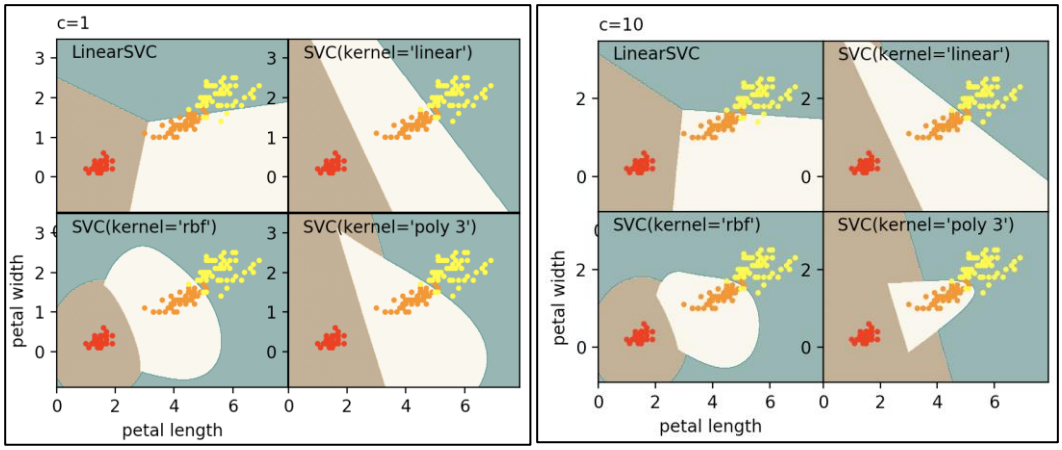


Figure 6.5 Comparison of various SVM models using the iris dataset with two different C parameters.

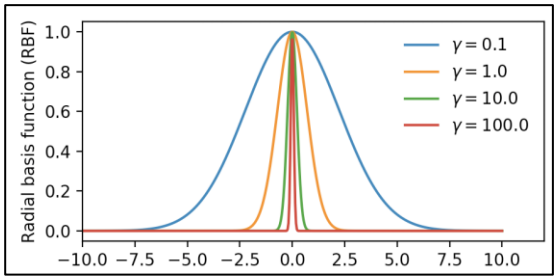


Figure 6.6 Gaussian radial basis function.

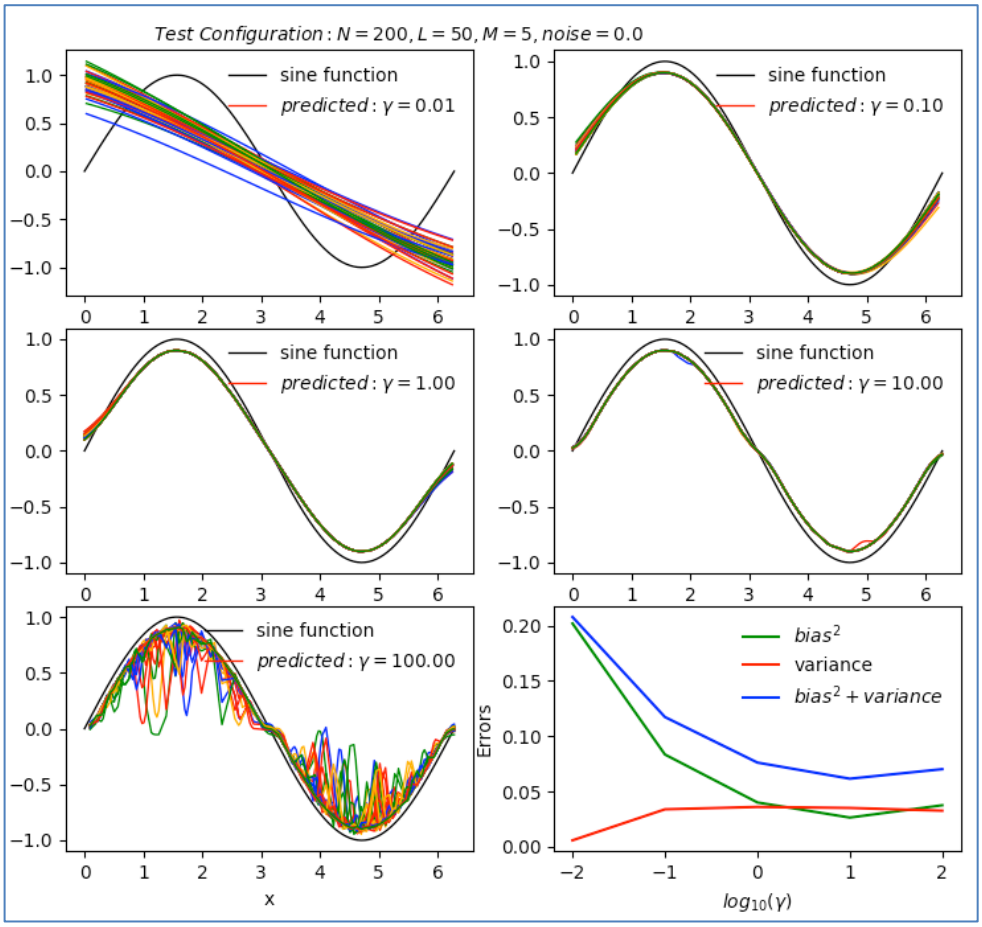


Figure 6.7 The ideal sine wave modeled with non-linear SVM models with N = 200.

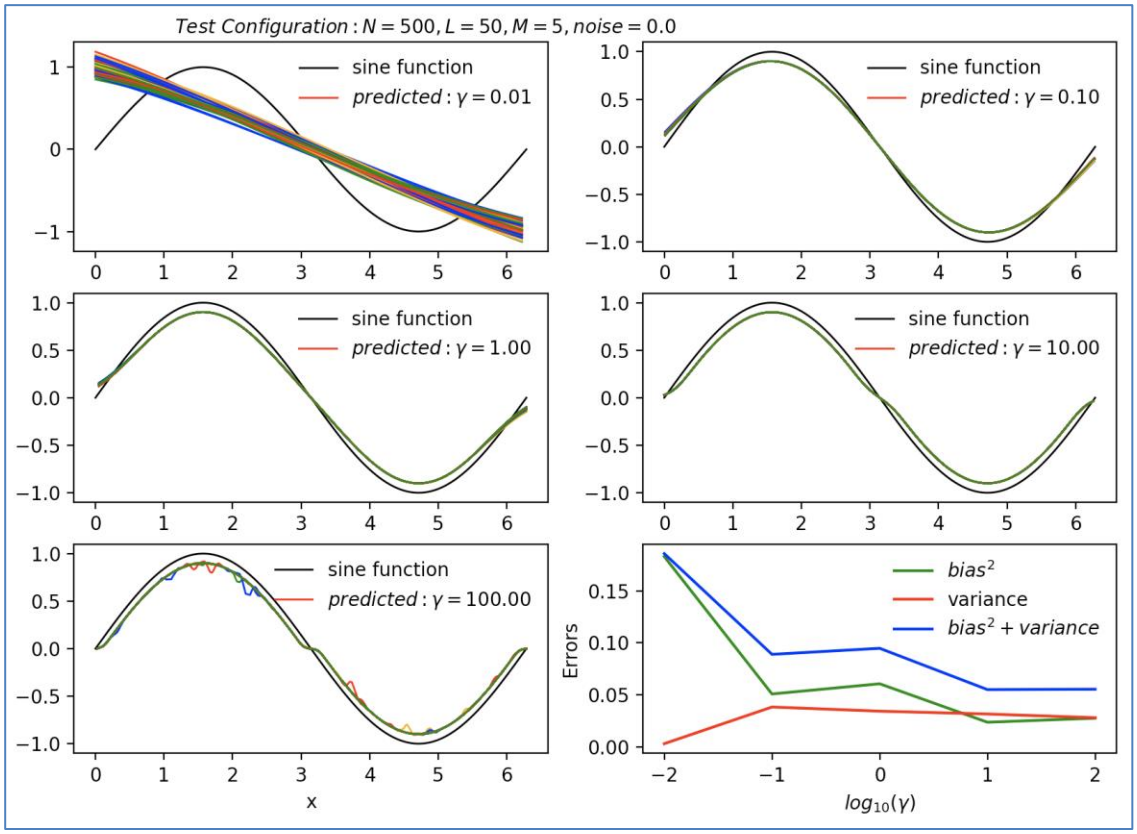


Figure 6.8 The ideal sine wave modeled with non-linear SVM models with N = 500.

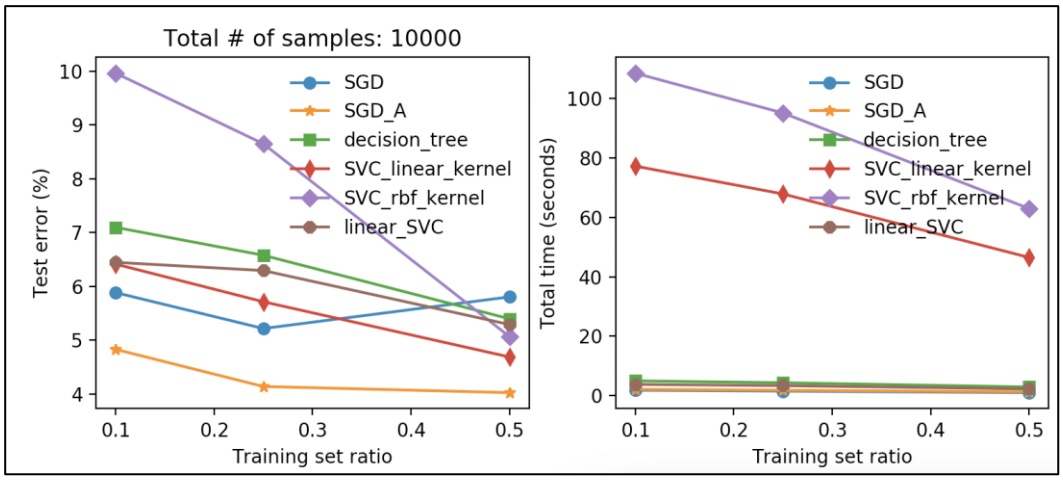


Figure 6.9 Classification benchmarking with multiple models applied to the MNIST dataset.

7 Random Forests and Ensemble Learning

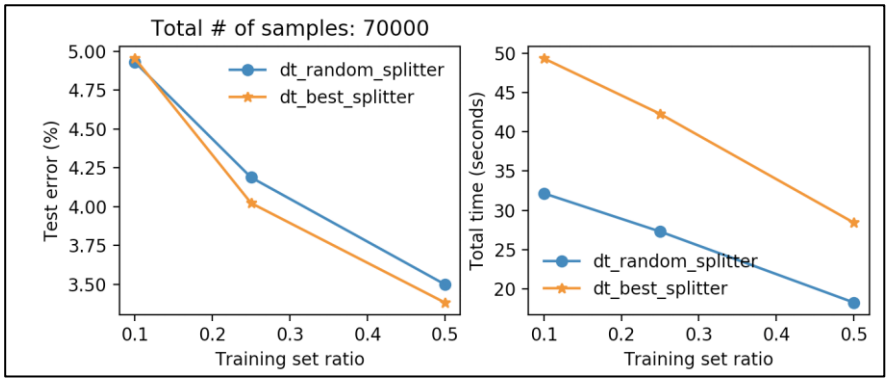


Figure 7.1 Baseline run with Decision Trees applied to the MNIST dataset.

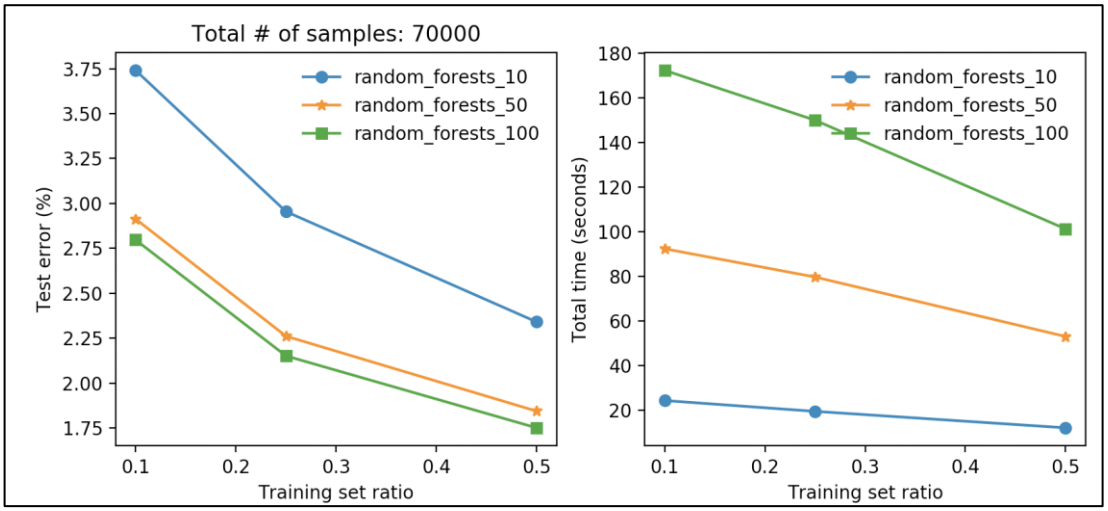


Figure 7.2 The MNIST dataset run with Random Forests of different number of trees (estimators).

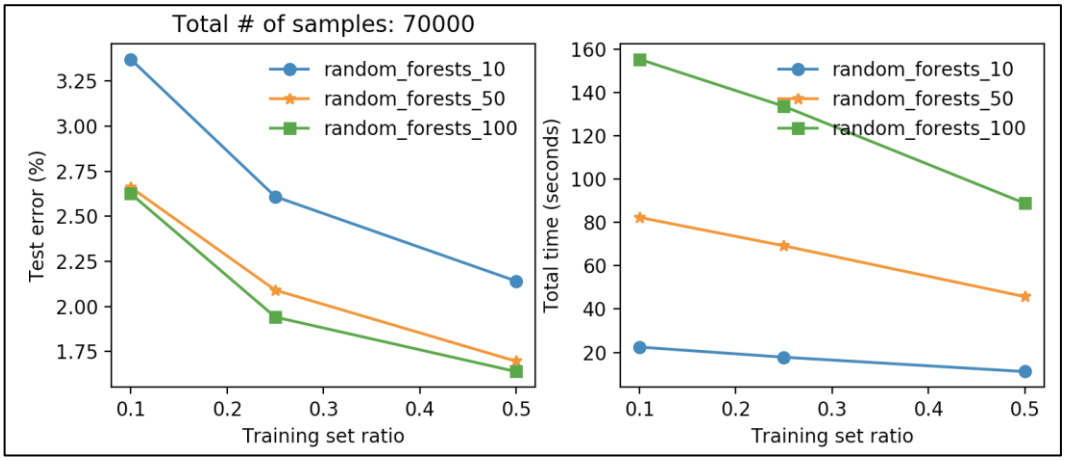


Figure 7.3 The MNIST dataset run with Random Forests of different number of trees (estimators) with `criterion` set to '`entropy`'.

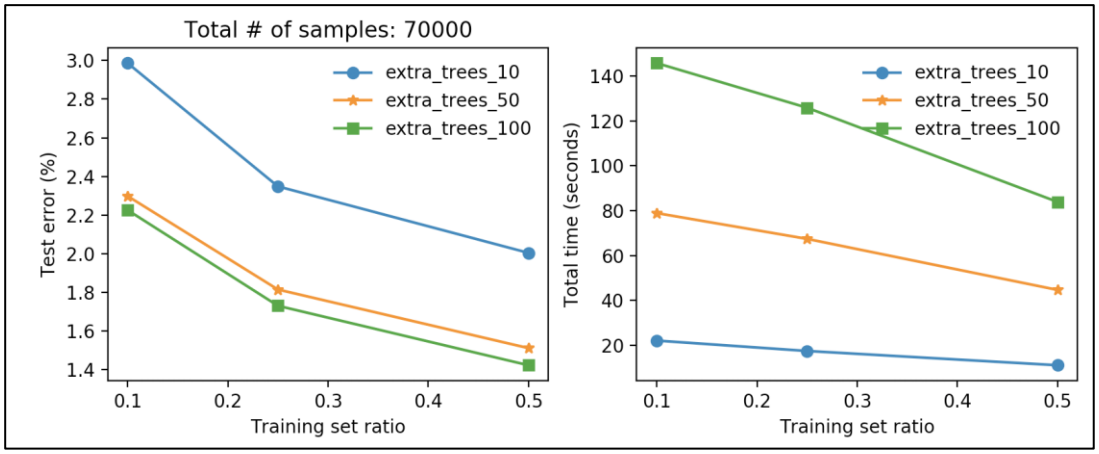


Figure 7.4 The MNIST dataset run with Extra Trees of different number of trees (estimators) with `criterion` set to '`entropy`'.

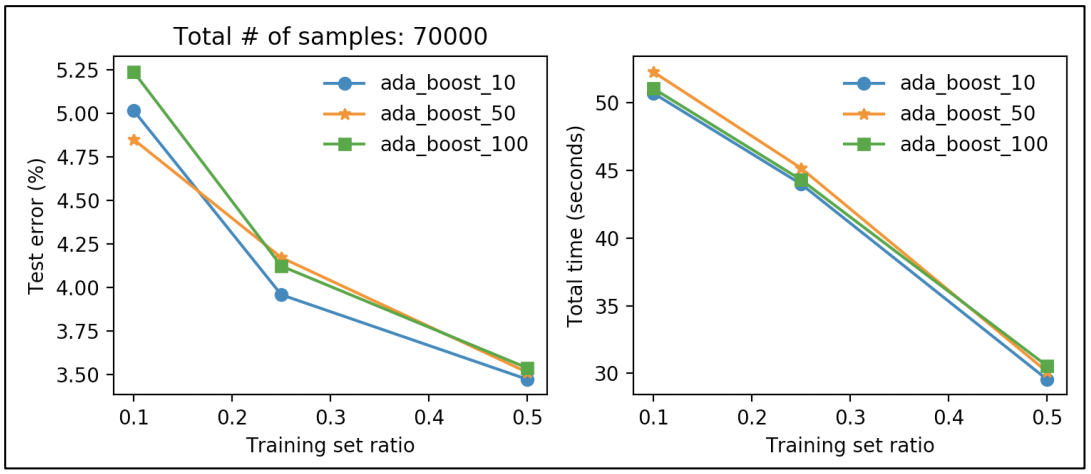


Figure 7.5 The MNIST dataset run with *AdaBoost* of different number of trees (estimators) with `criterion` set to '`entropy`'.

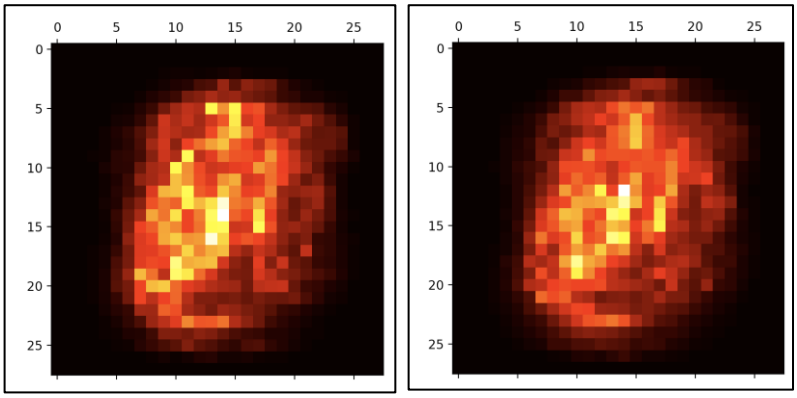


Figure 7.6 The MNIST dataset feature importance matrix images from the Random Forest (left) and Extra Trees (right).

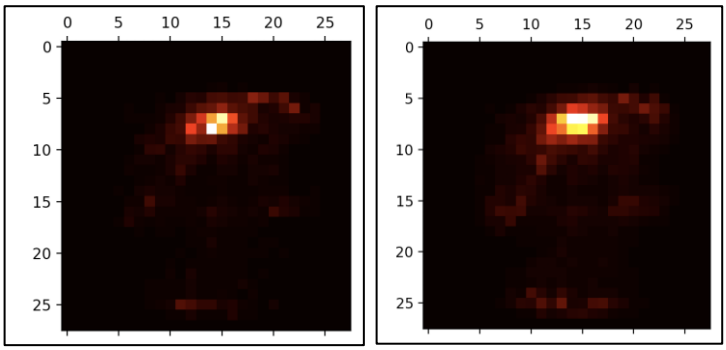


Figure 7.7 The MNIST dataset feature importance matrix images with 4's and 9's from the Random Forest (left) and Extra Trees (right).

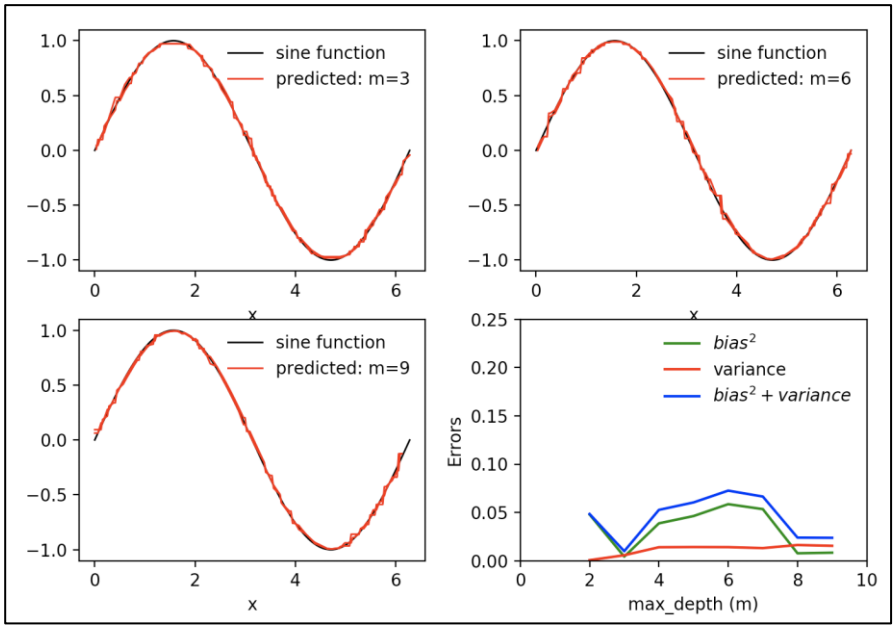


Figure 7.8 An ideal sine-function fitted with the `sklearn.tree.GradientBoostingRegressor`, with `n_estimators` = 50 and the `max_depth` hyper-parameter (`m`) varied from 2 to 9.

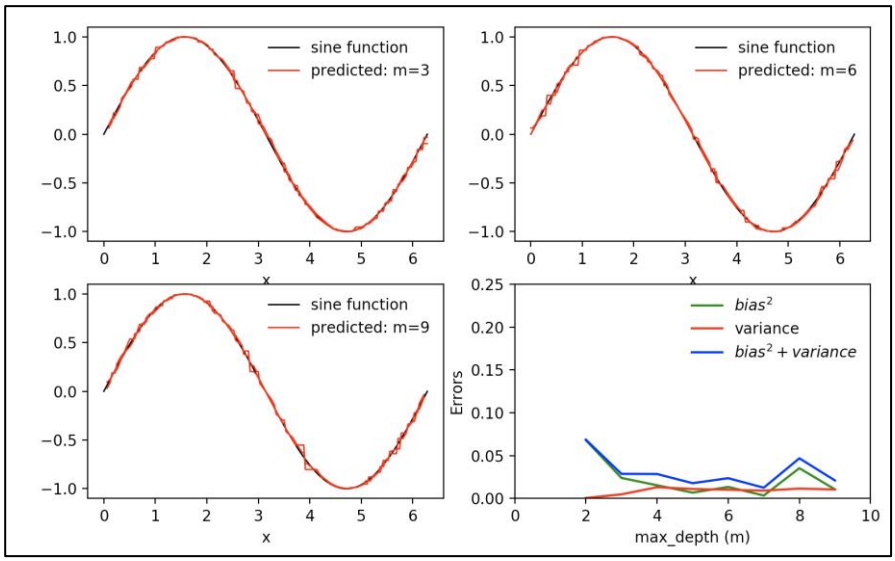


Figure 7.9 An ideal sine-function fitted with the `sklearn.tree.GradientBoostingRegressor`, with `n_estimators` = 100 and the `max_depth` hyper-parameter (`m`) varied from 2 to 9.

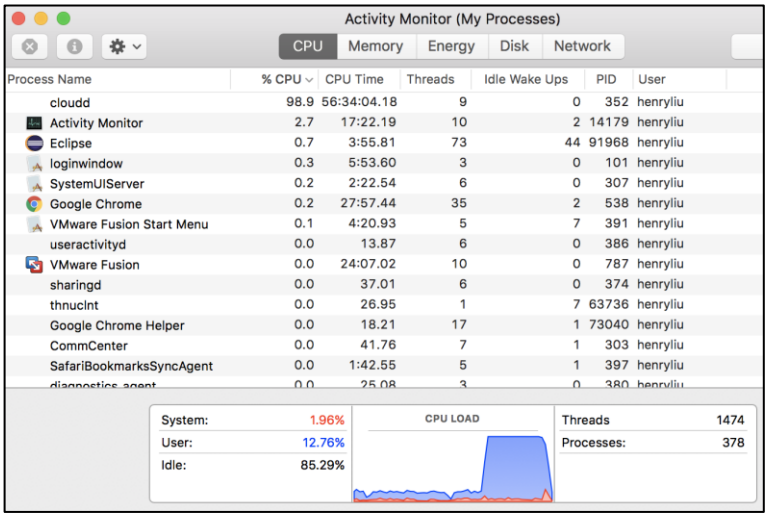


Figure 7.10 Missing `the max_samples = 1000` parameter in the classifier caused CPUs running at ~100% level.

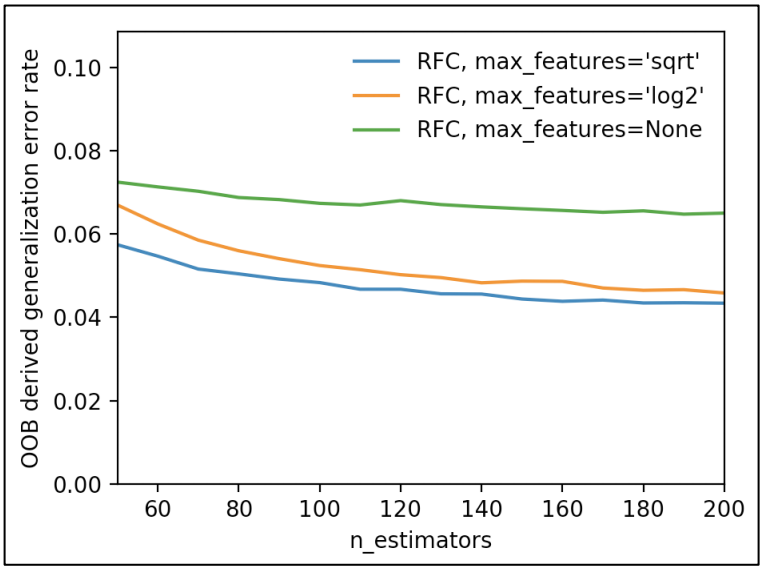
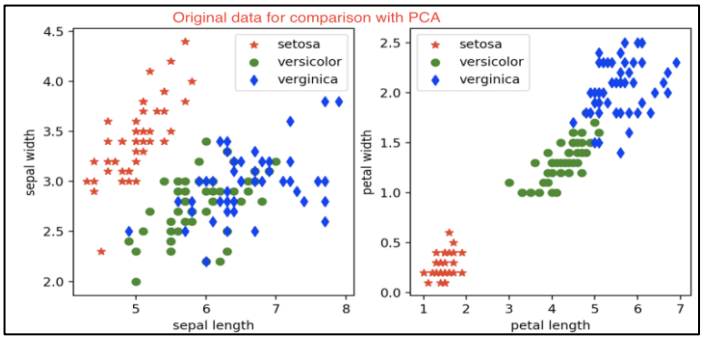
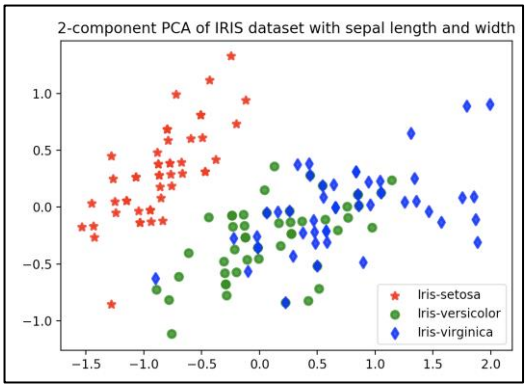
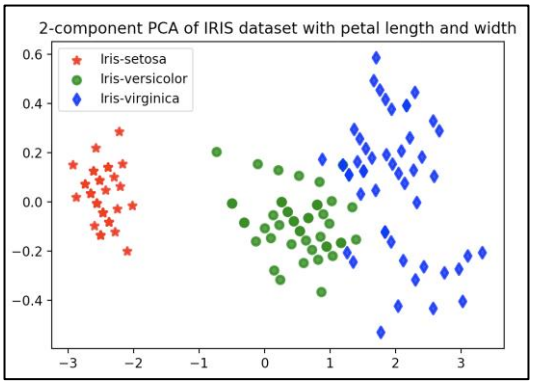


Figure 7.11 Extra 2% improvement gained with the `max_features` = '`sqrt`' or '`log2`' versus the default of `None` with the `RandomForestClassifier` ensemble learning (total time: 630s).

8 Dimensionality Reduction







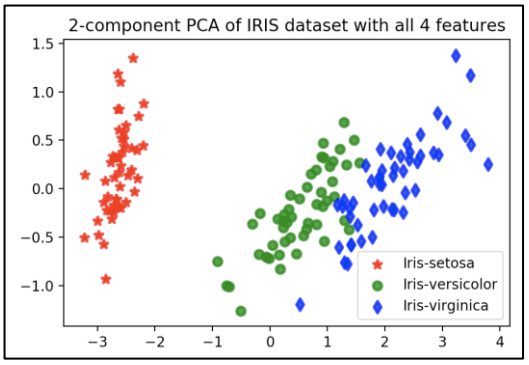


Figure 8.1 PCA applied to the Iris dataset: (a) Top row: original data for each pair of the features; (b) second row: 2 principal components for the two sepal features (left) and the two petal features (right), respectively; (3) third row: 2 principal components when all 4 features were included in the input.

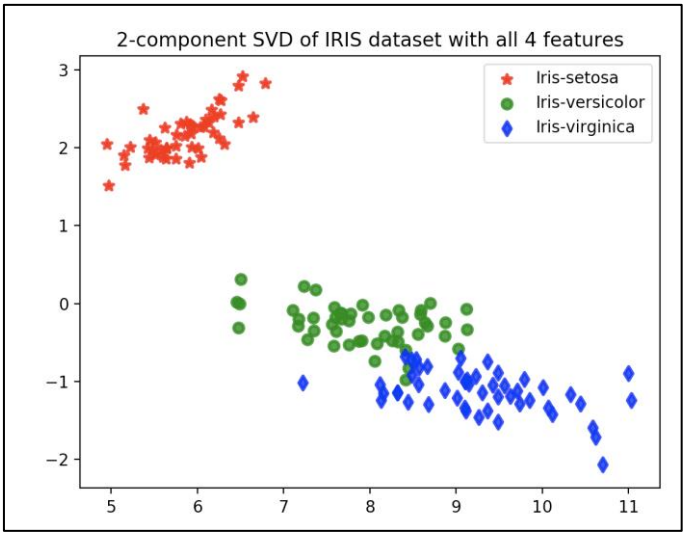


Figure 8.2 SVD applied to the iris dataset with all four input features included.

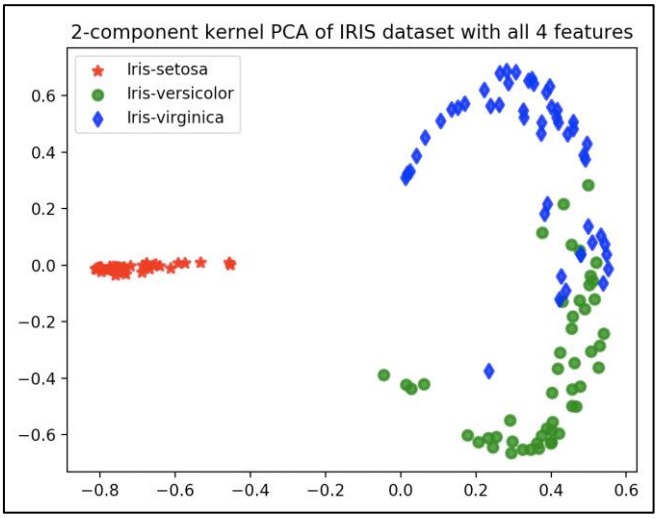


Figure 8.3 Kernel PCA applied to the iris dataset.

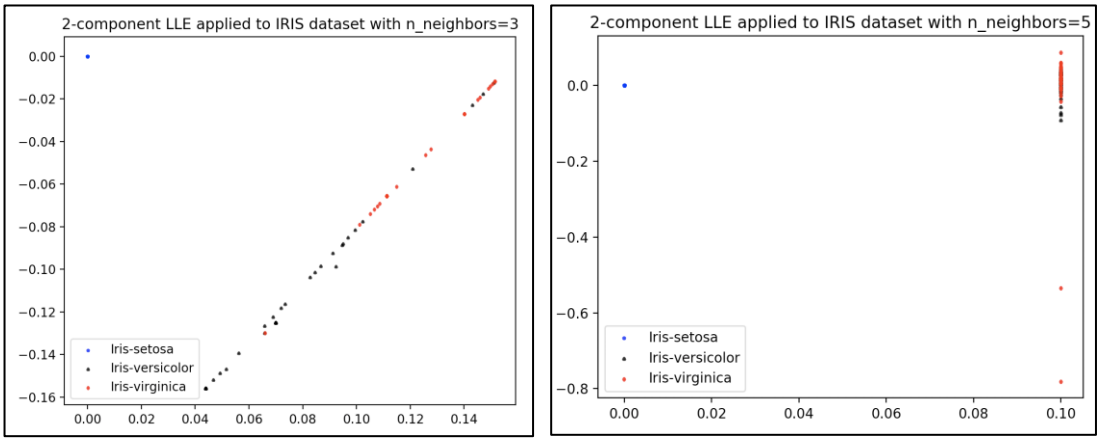


Figure 8.4 LLE applied to the iris dataset with `n_neighbors = 3` (left) and `n_neighbors = 5` (right).

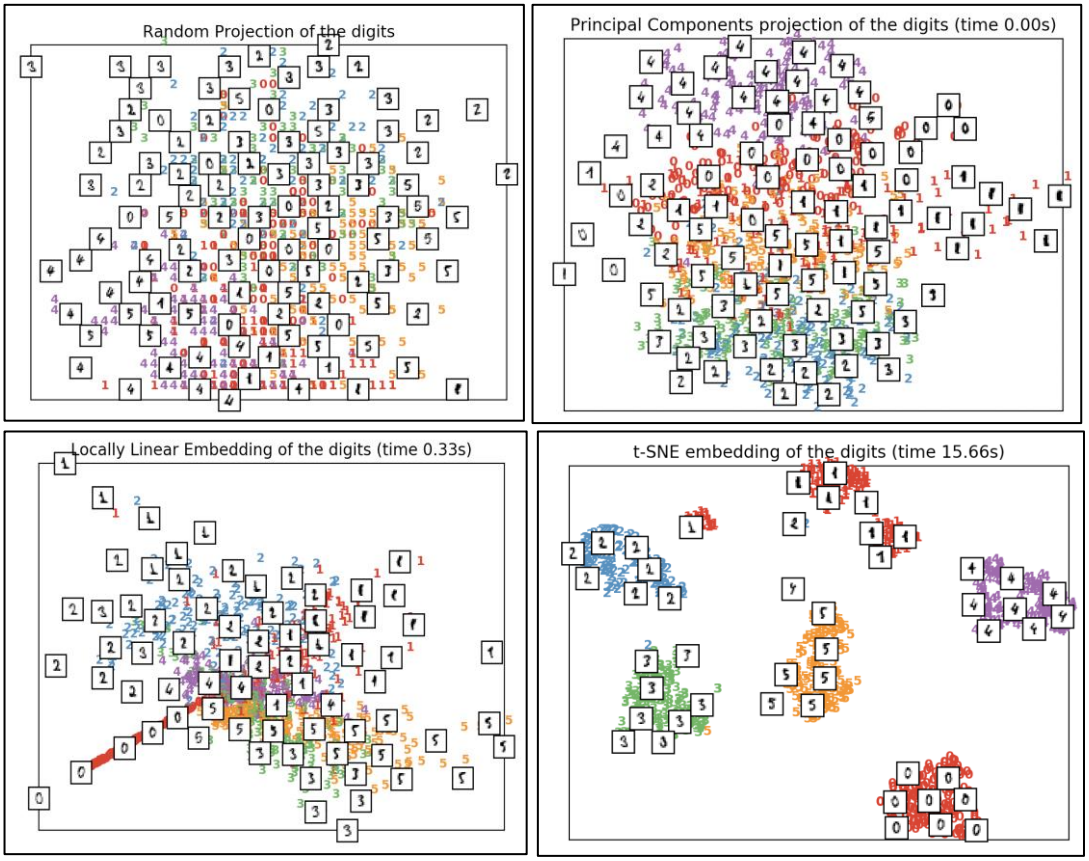


Figure 8.5 LLE applied to the MNIST dataset with (1) upper left: random projection, (2) upper right: PCA projection, (3) lower left: LLE, and (4) lower right: t-SNE (*t-distributed stochastic neighbor embedding*).

9 Introduction to Artificial Neural Networks

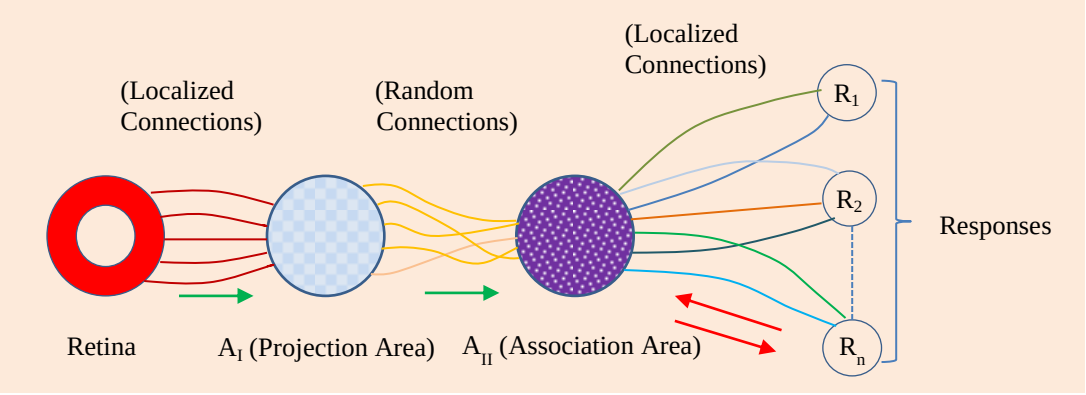


Figure 9.2 Organization of a perceptron proposed by Rosenblatt.

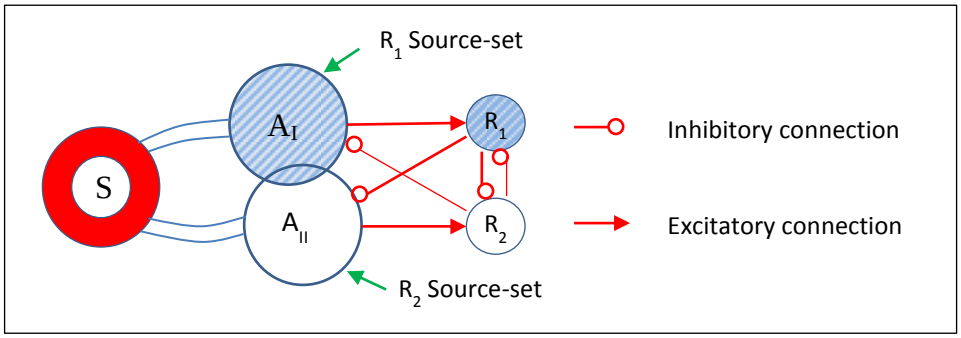


Figure 9.3 A simplified model of the perceptron for establishing the theory of statistical separability.

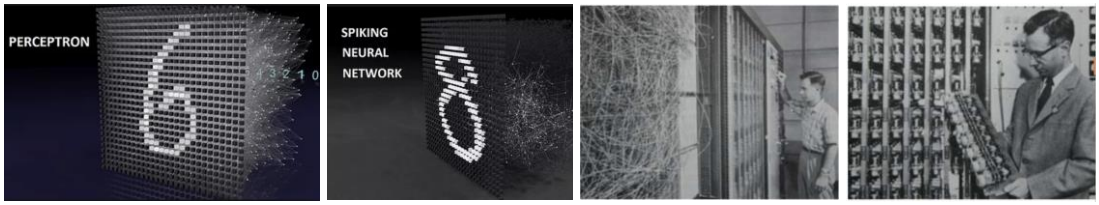


Figure 9.4 Rosenblatt and his Mark I Perceptron.

Table 9.1 An XOR operator

INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

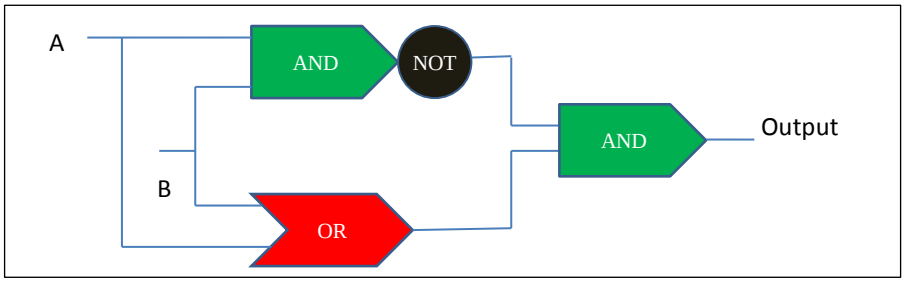


Figure 9.5 An XOR gate circuit composed of three mixed gates for implementing $(A + B) \cdot \overline{(A + B)}$.

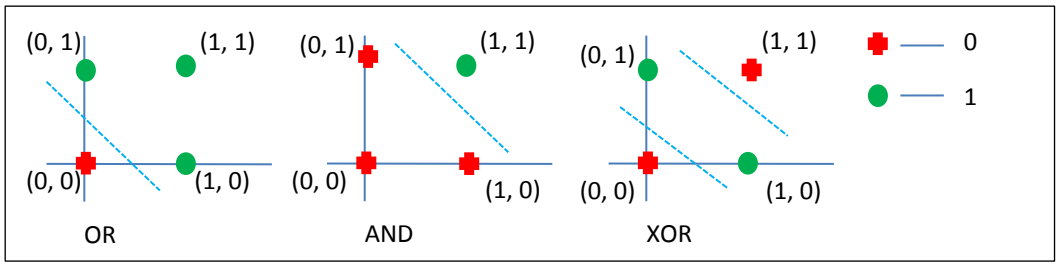


Figure 9.6 OR and AND are linearly separable while XOR is not.

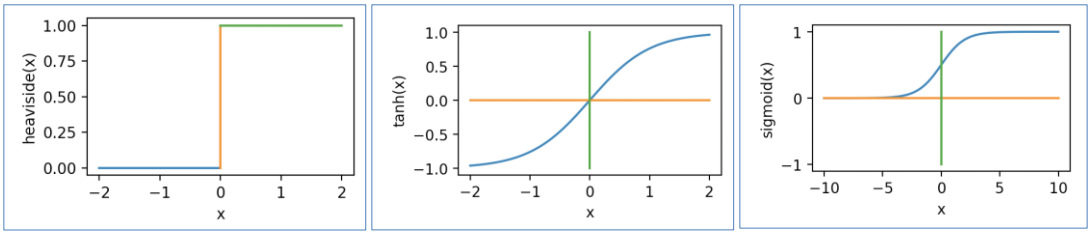


Figure 9.7 Three typical activation functions of *Heaviside*, *tanh* and *sigmoid*.

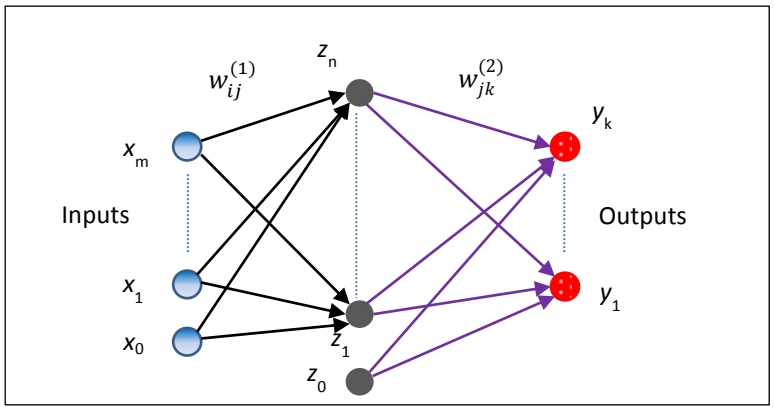


Figure 9.8 Feed-forward neural network architecture.

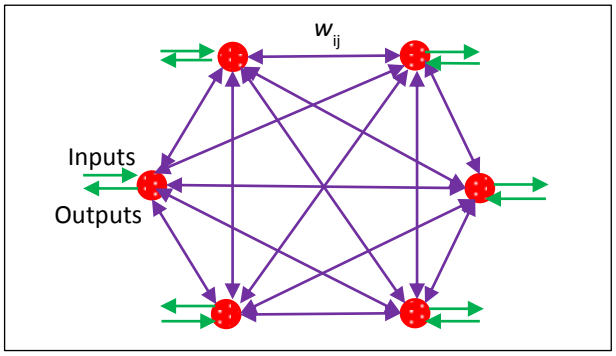


Figure 9.9 A Hopfield net with six units.

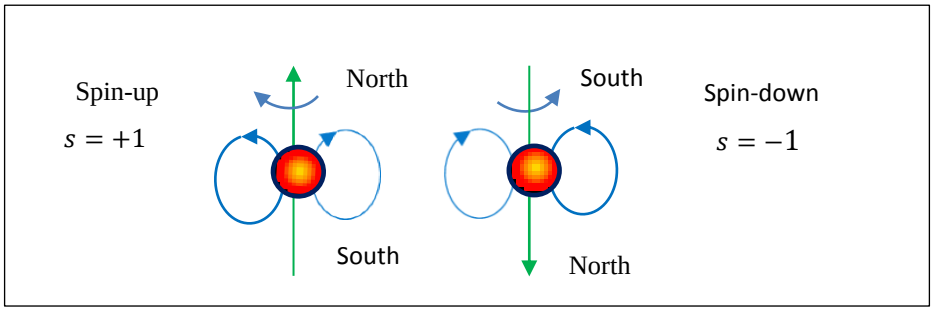


Figure 9.10 The two possible spin polarizations of the electron.

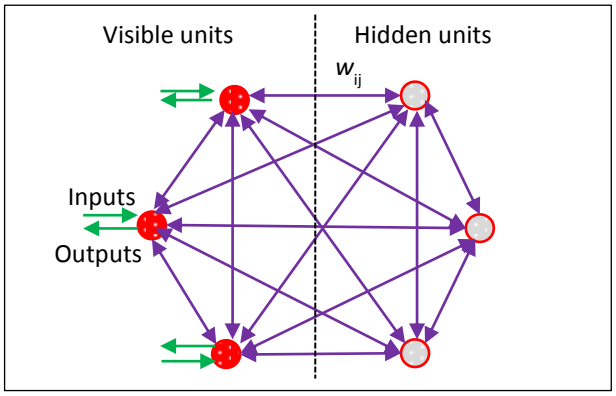


Figure 9.11 Boltzmann machines with visible and hidden units.

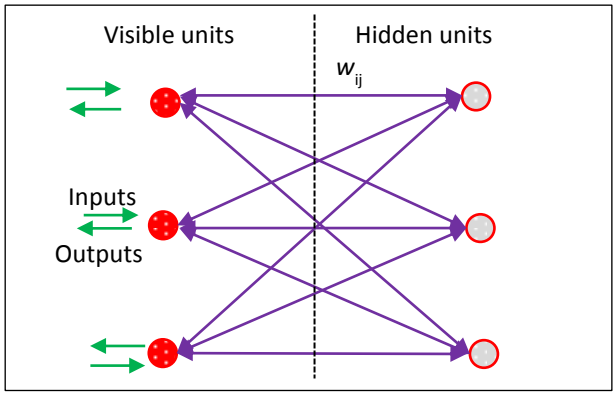


Figure 9.12 A restricted Boltzmann machine (RBM).

10 Convolutional Neural Networks

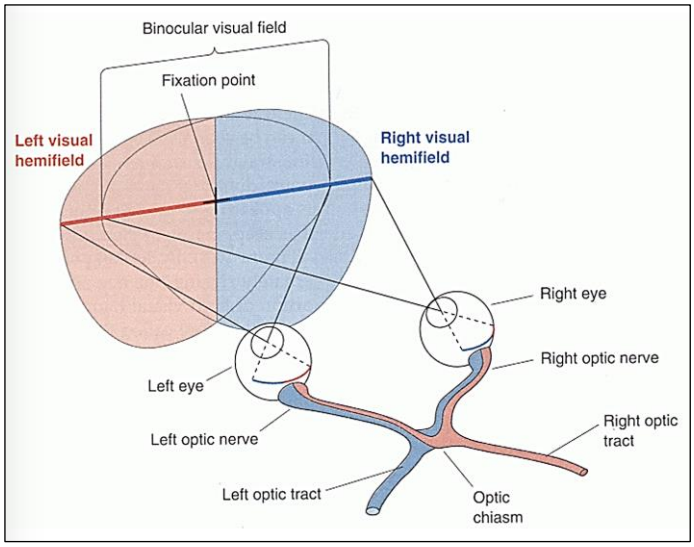


Figure 10.1 The visual field consisting of the right and left visual hemifields. (Source: Courtesy of Bear et al.)

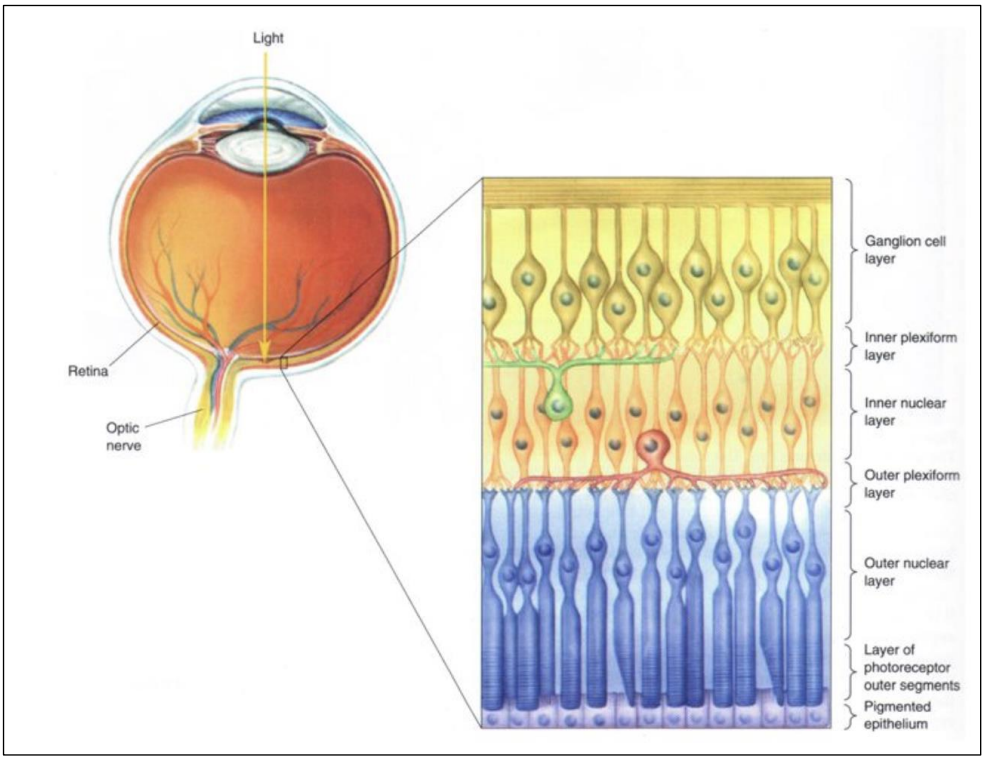


Figure 10.2 The laminar organization of the retina. (Source: Courtesy of Bear et al.)

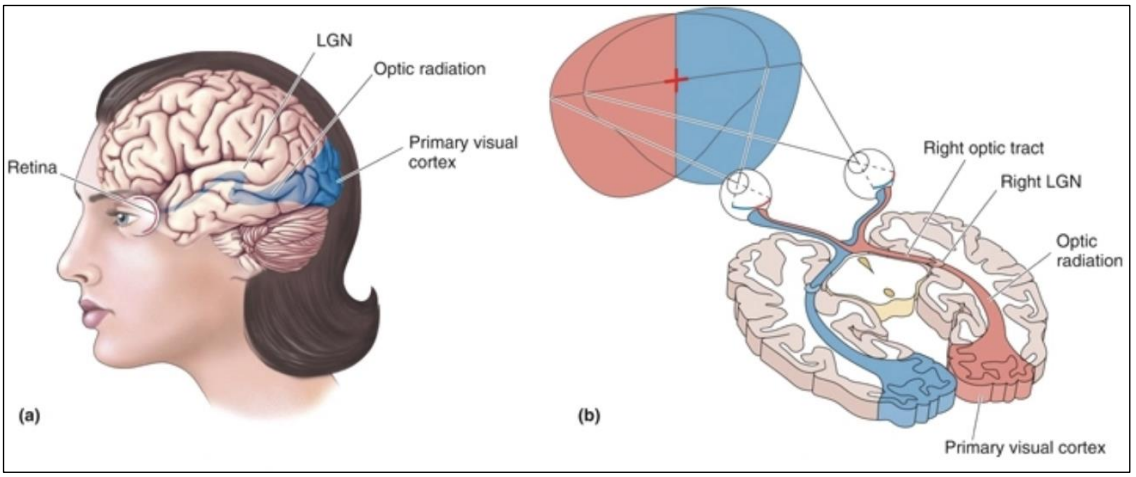


Figure 10.3 The visual pathway that mediates conscious visual perception. (a) A side view of the brain with the retinogeniculocortical pathway shown in blue. (b) A horizontal section through the brain exposing the same pathway. (Source: Courtesy of Bear et al.)

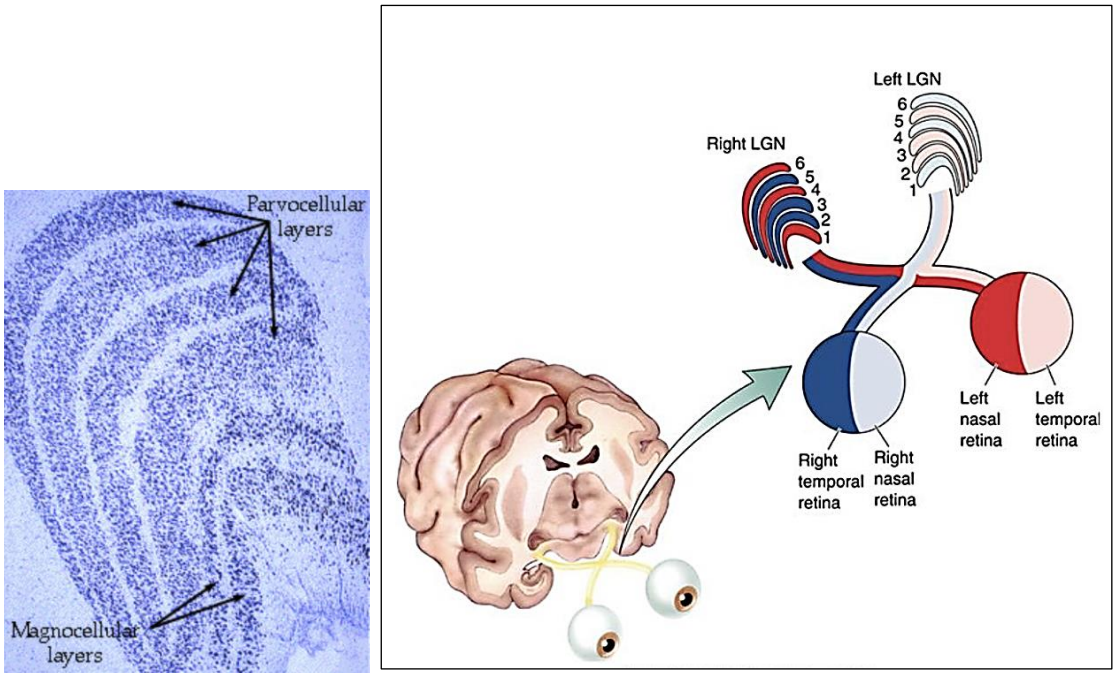


Figure 10.4 Retina inputs to the LGN layers. Left: A real LGN in six layers. Right: How right LGN and left LGN are connected to the left retina and right retina. (Source: Courtesy of Bear et al.)

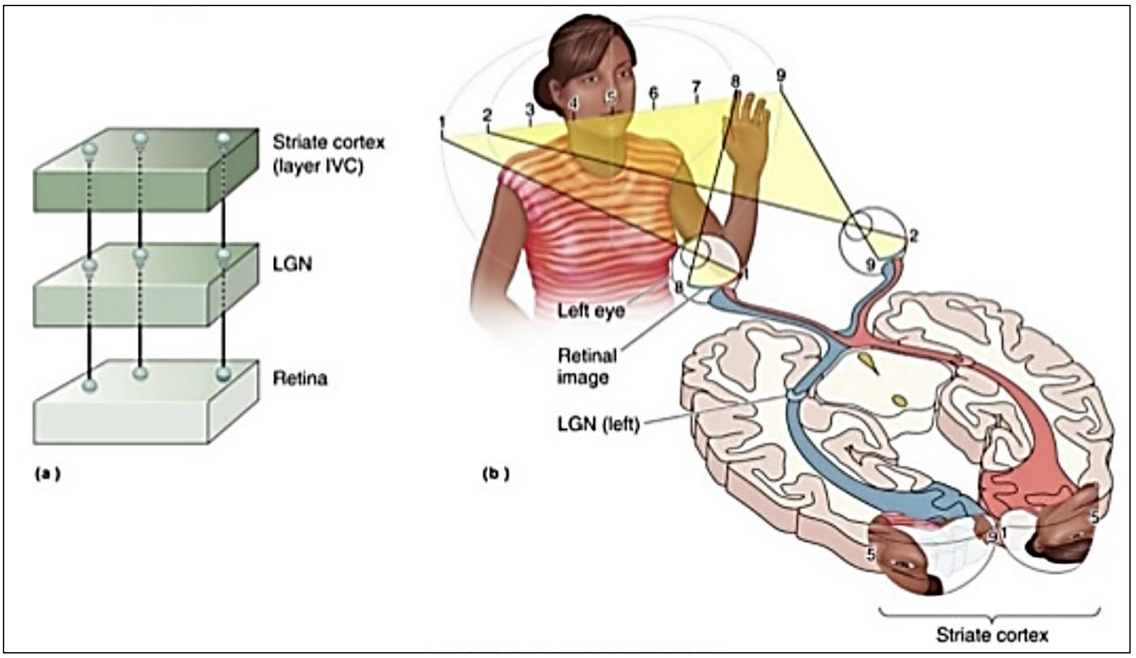


Figure 10.5 The retinotopic map in the striate cortex. (a) Signals originating from retina and ending up on the striate cortex. (b) How an image is mapped to the striate vertex from the left retina and right retina and through both LGN gateways. (Source: Courtesy of Bear et al.)

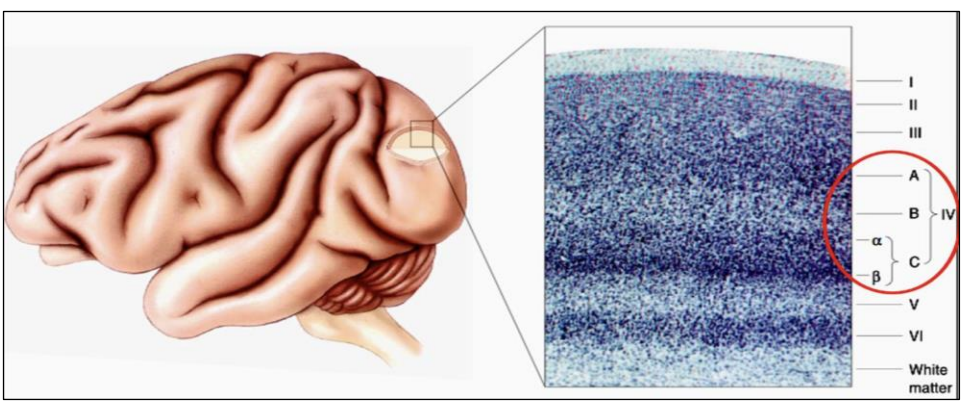


Figure 10.6 The cytoarchitecture of the striate cortex. Layer IV is further divided into layers A, B, Cα and Cβ. (Source: Courtesy of Dr. Bear.)

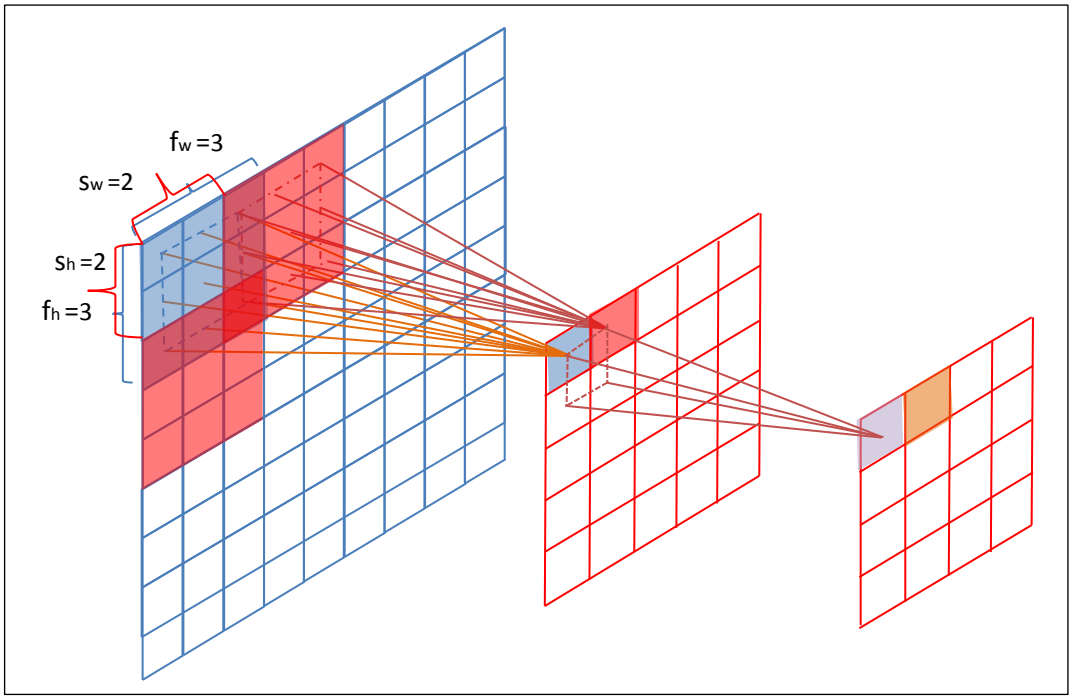


Figure 10.15 A generic configuration of a locally connected network with a 3×3 receptive field viewed from the second layer and a 2×2 receptive field viewed from the third layer. The stride is 2 for both layers.

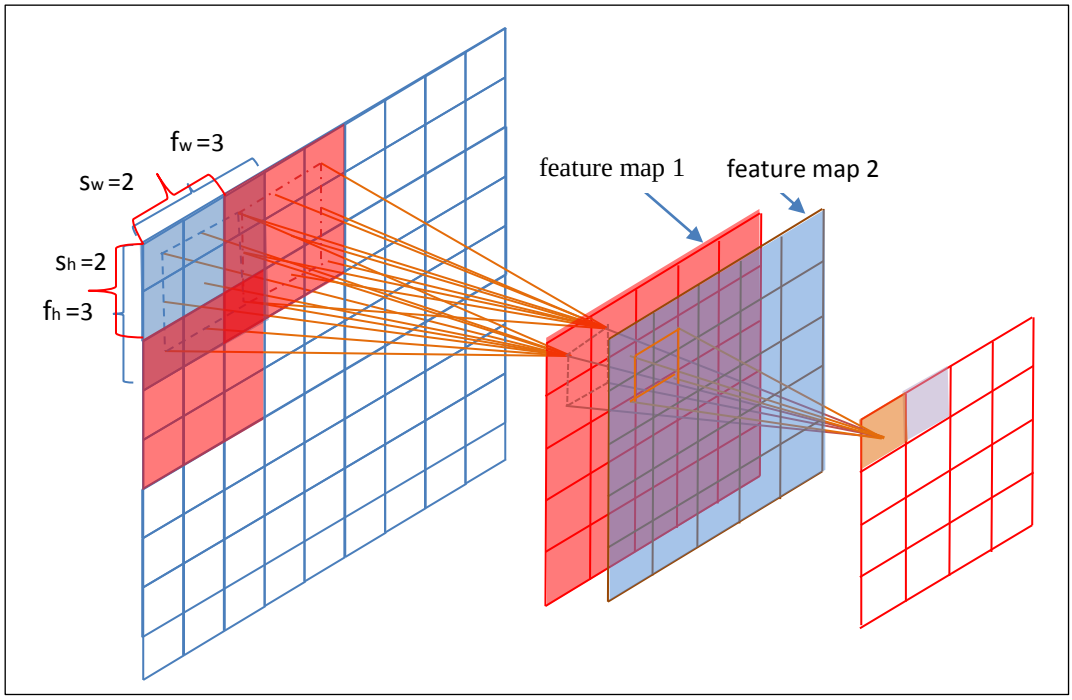


Figure 10.16 A generic configuration of a locally connected network with a 3×3 receptive field viewed from the second layer and a 2×2 receptive field viewed from the third layer. The stride is 2 for both layers. The second layer consists of two feature maps.

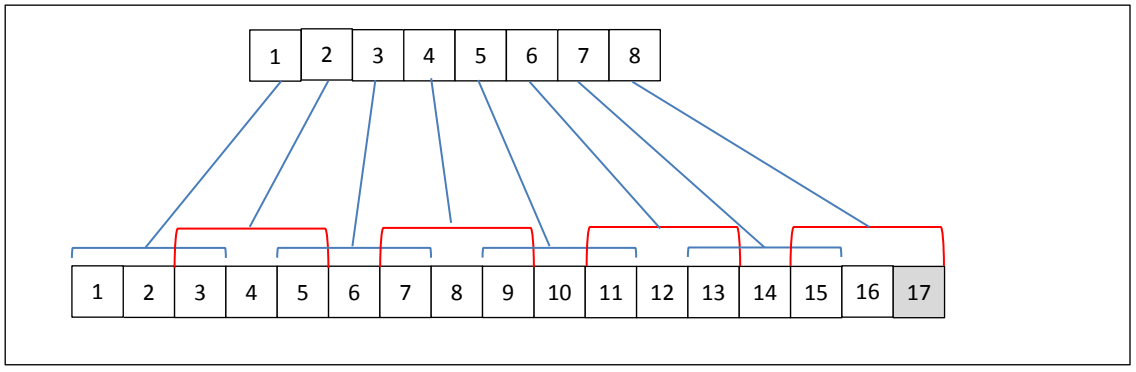


Figure 10.17 Zero padding for the first hidden layer of the Net-3 configuration shown in Figure 10.13 against the input layer.

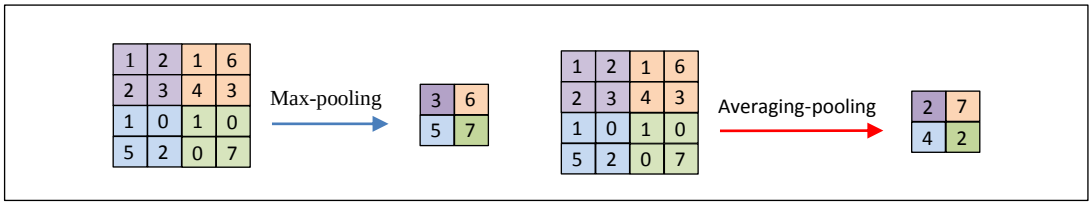


Figure 10.18 Max pooling versus averaging-pooling

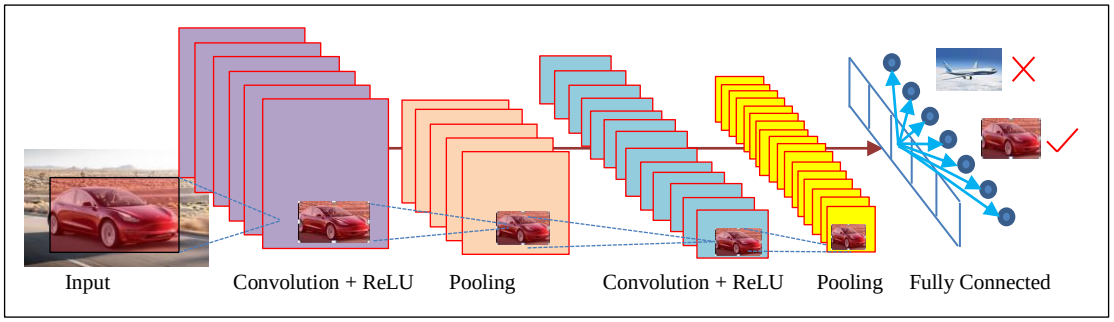


Figure 10.19 A generic CNN architecture

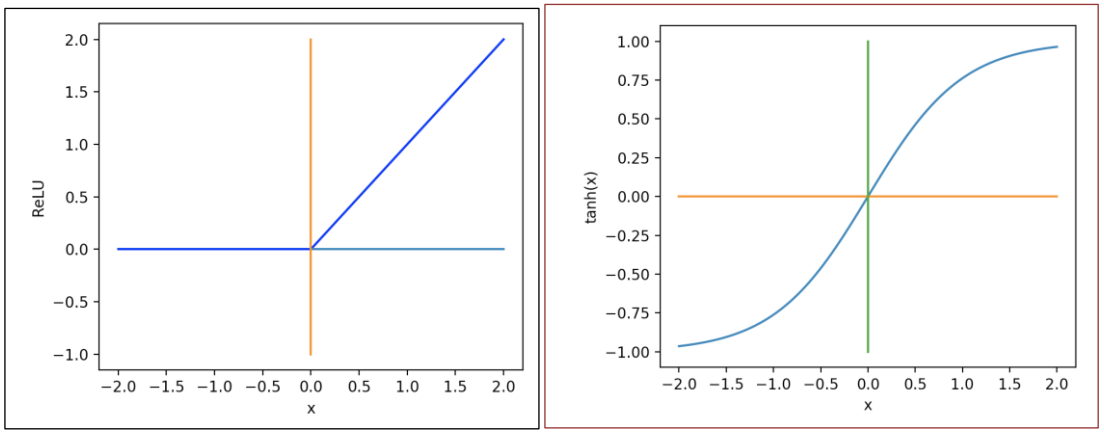


Figure 10.20 The ReLU function versus the hyperbolic tangent function (tanh).

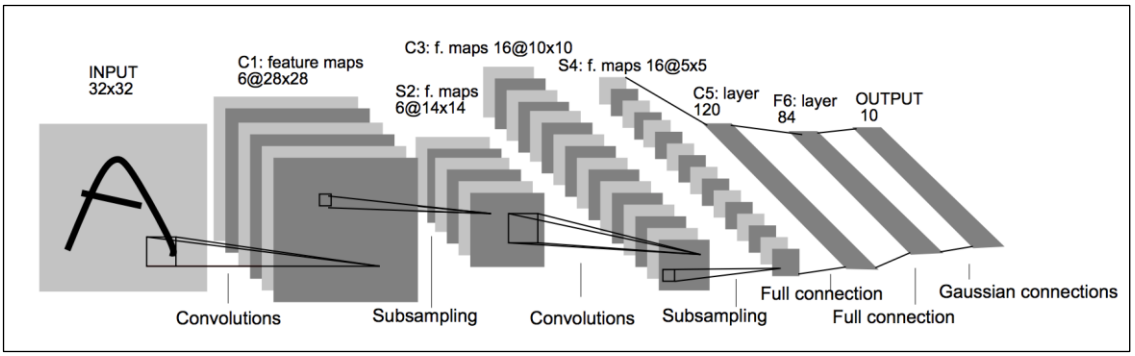


Figure 10.21 The LeNet-5 architecture for digits recognition. (Source: Courtesy of LeCun et al.)

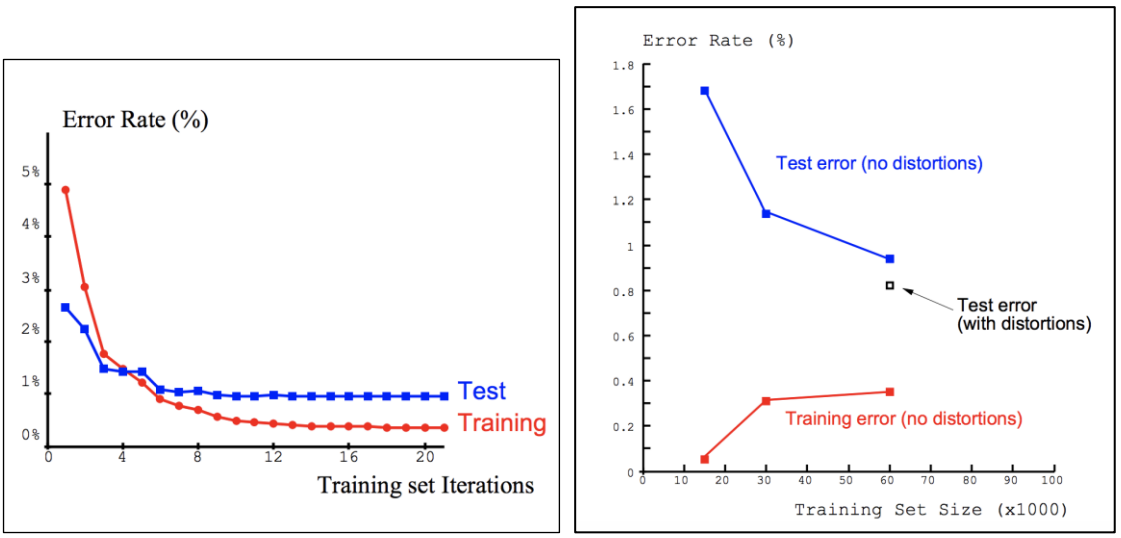


Figure 10.23 Error rates of LeNet-5 as a function of the numbers of passes through the 60,000 pattern training set (left) and training set size (right). The hollow square on the right shows the test error when more training patterns were artificially generated using random distortions. (Source: Courtesy of LeCun et al.)

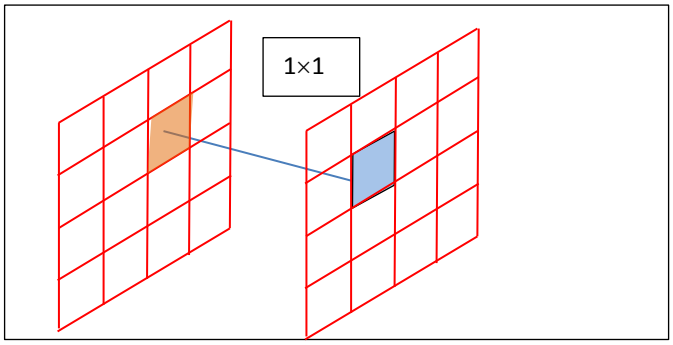


Figure 10.28 A 1×1 kernel in a 2D dimension cannot reduce.

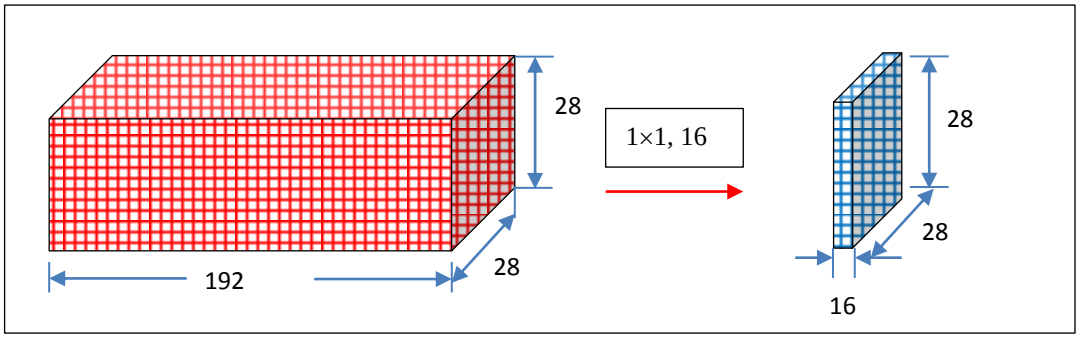


Figure 10.29 A 1×1 convolution gives a reduction ratio of 192/16 = 12x.

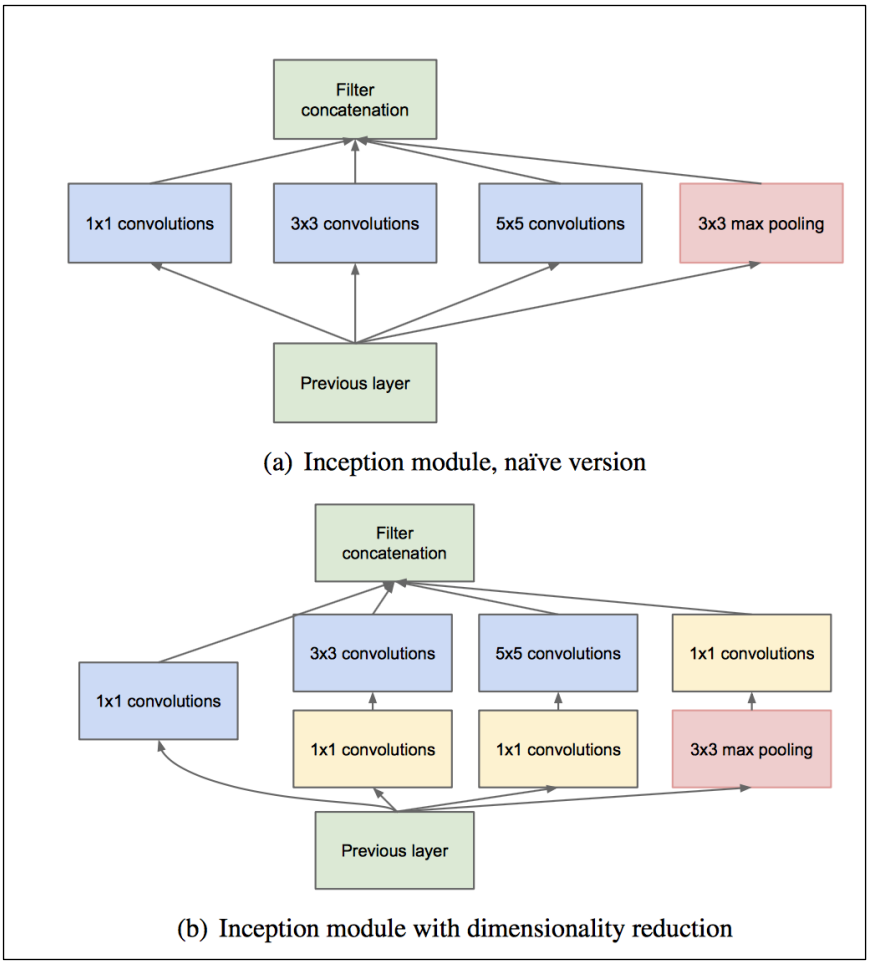


Figure 10.30 Two versions of the inception module. (a): naïve (non-working version). (b): working version. (Source: Courtesy of Szegedy et al.)

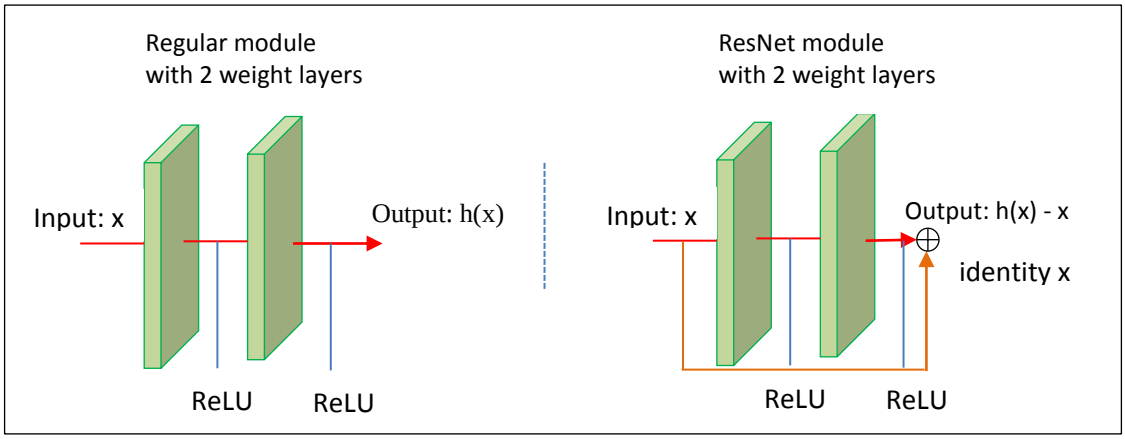


Figure 10.32 A building block for residual learning.

10.1.1 Batch normalization

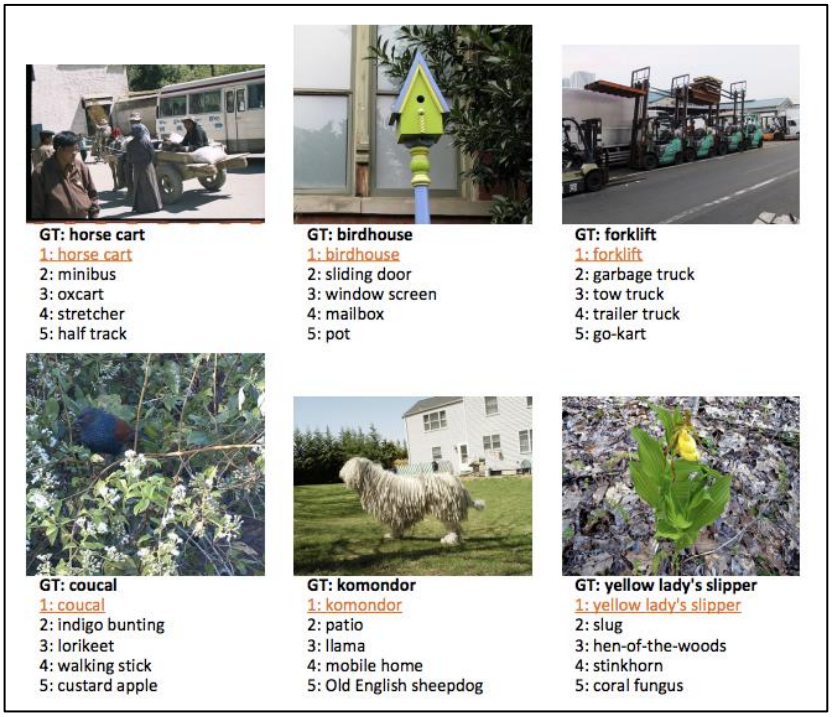


Figure 10.34 Example validation images successfully classified by using the He initialization. For each image, the ground-truth label and the top-5 labels predicted are listed. (Source: Courtesy of He et al.)

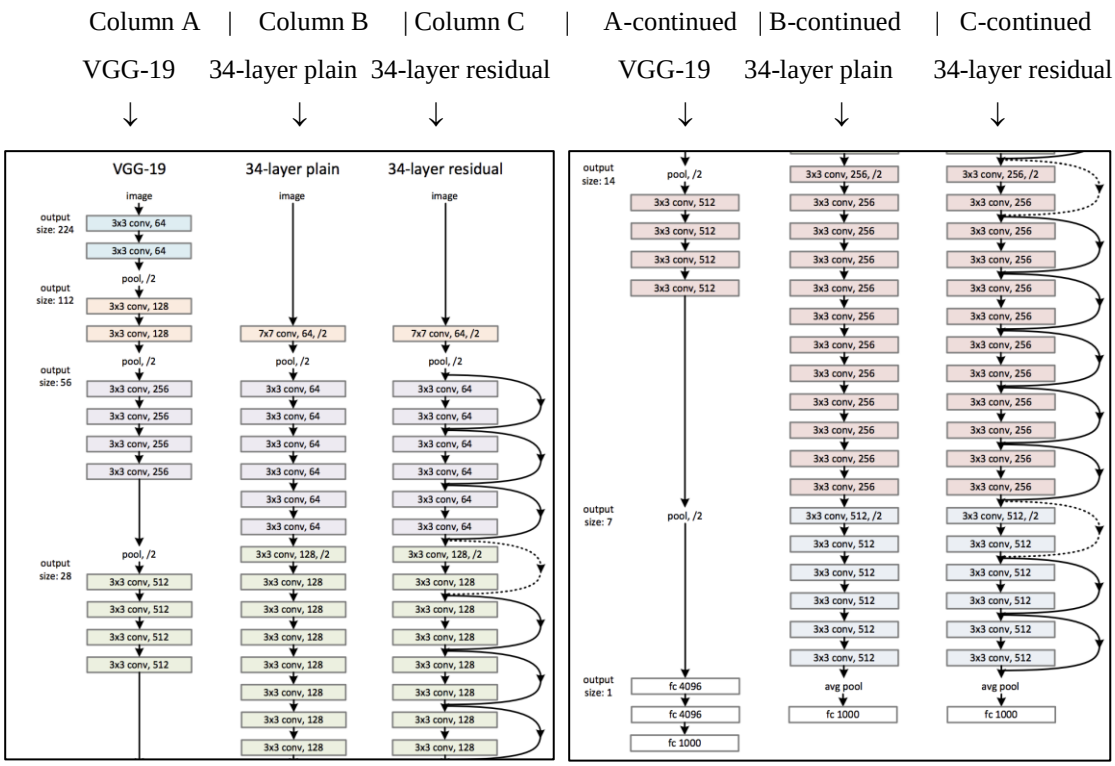


Figure 10.35 Example ResNet architectures, with the left three columns continued on the right. (Source: Courtesy of He et al.)

11 Recurrent Neural Networks

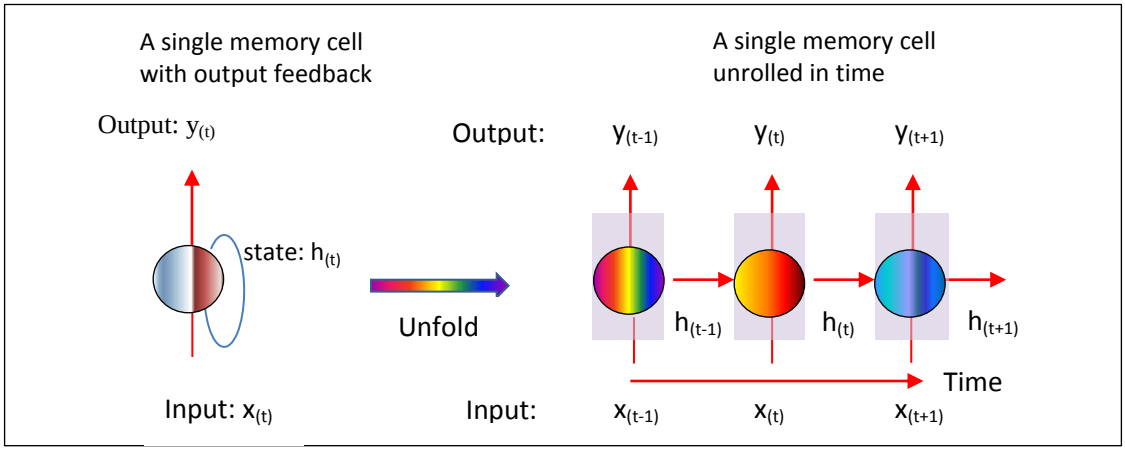


Figure 11.1 Simple RNN cells unrolled in time with a lag of 2.

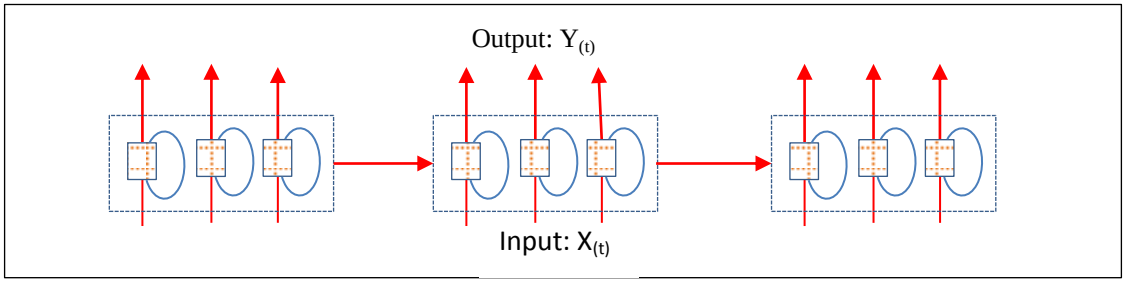


Figure 11.2 Multiple RNN layers connected together.

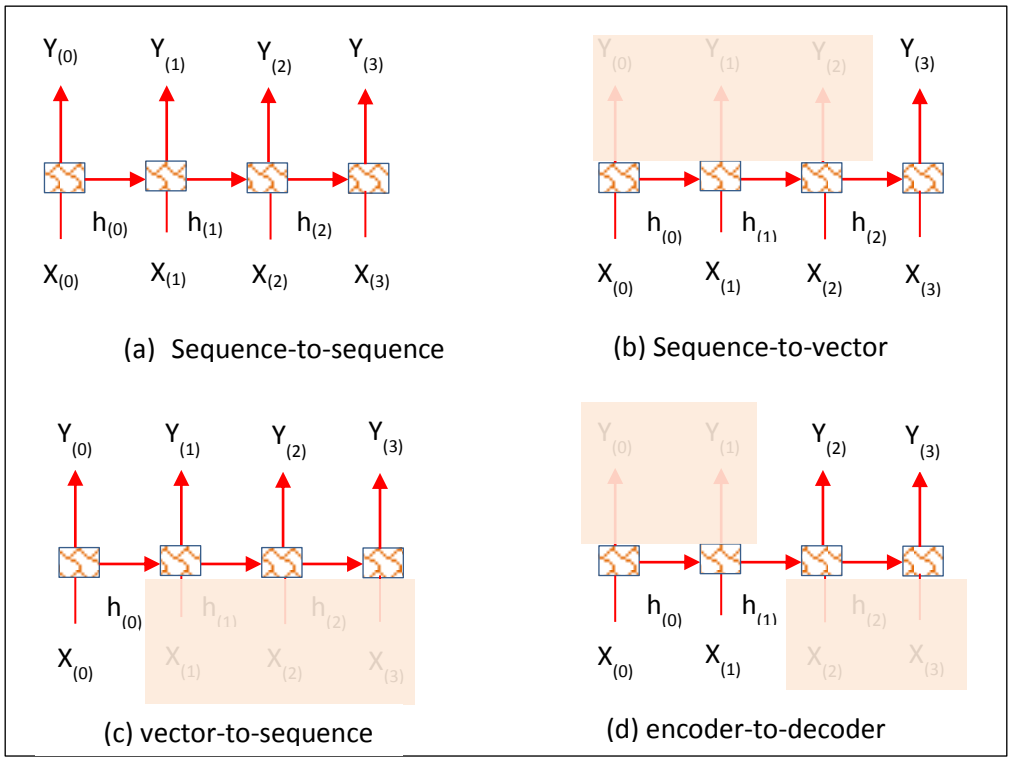


Figure 11.3 Classification of RNNs based on input-output relationships.

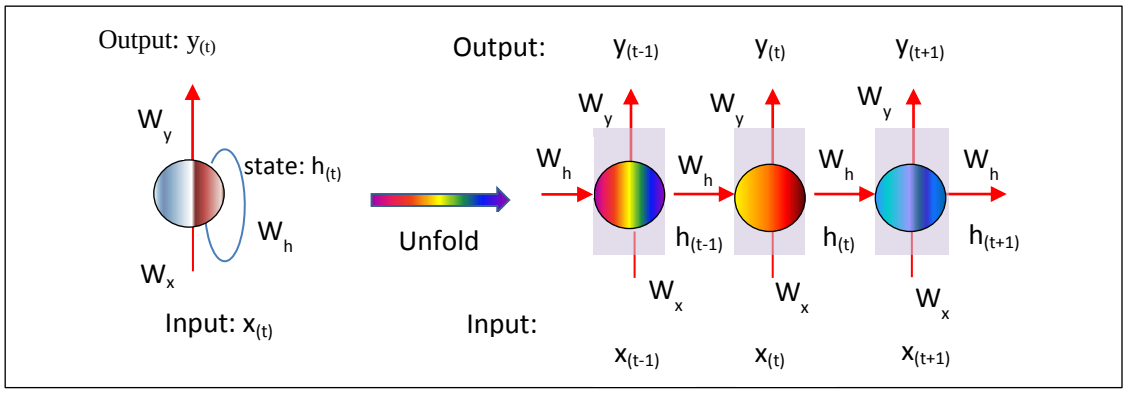


Figure 11.4 Back-propagation through time (BPTT).

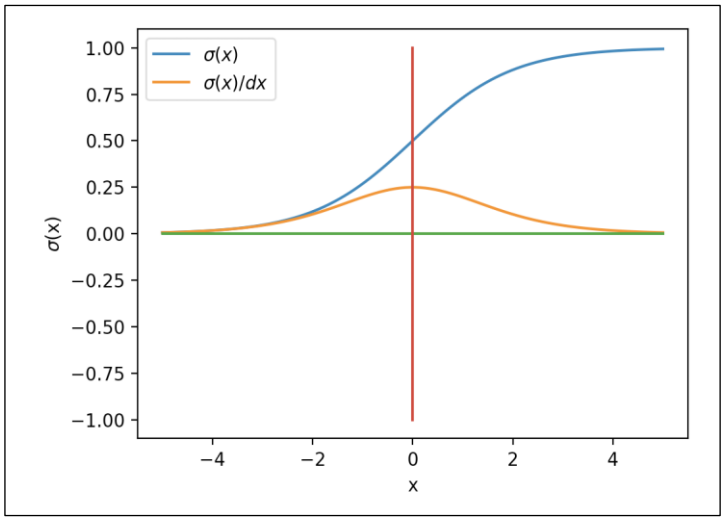


Figure 11.5 Gradient approaches zero for large absolute input values with the logistic activation function.

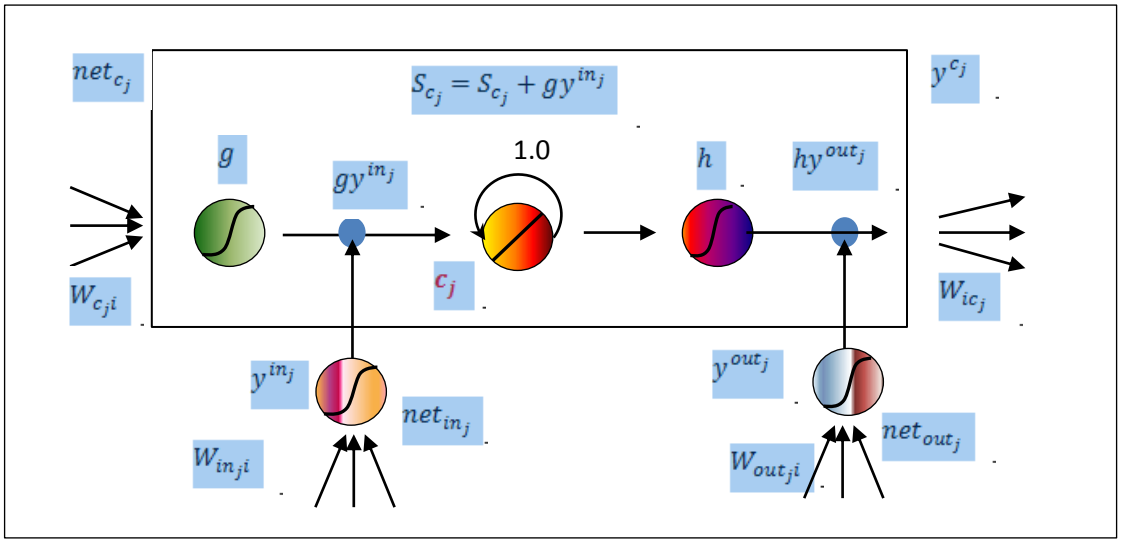


Figure 11.6 Architecture of a memory cell c_j guarded by an input gate unit (net_{in_j}) and an output gate unit (net_{out_j}). The boxed part represents a *memory cell*.

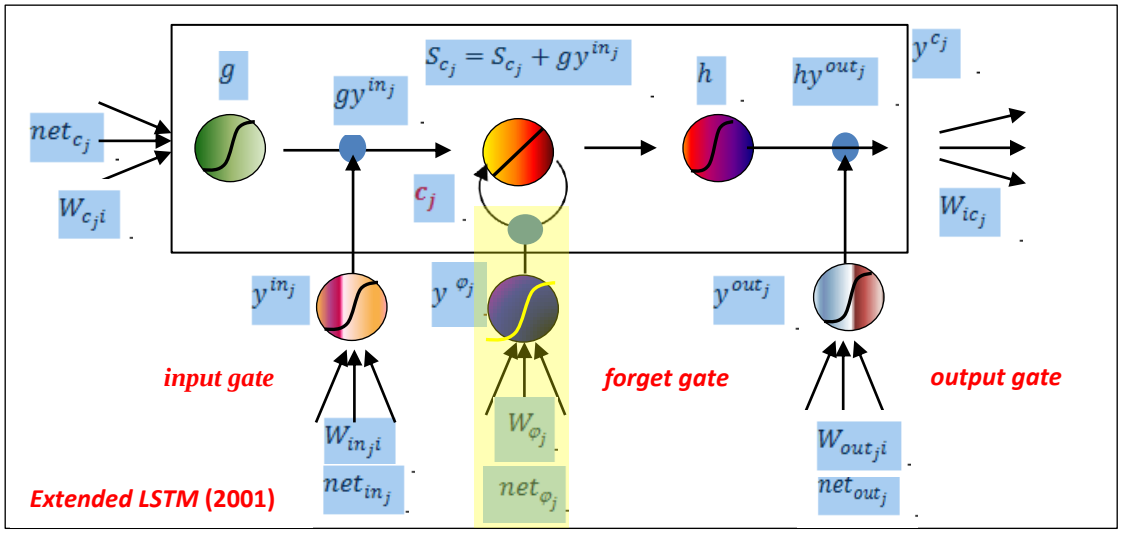


Figure 11.7 Memory block with only one cell for the extended LSTM with a *forget-gate* shaded yellow.

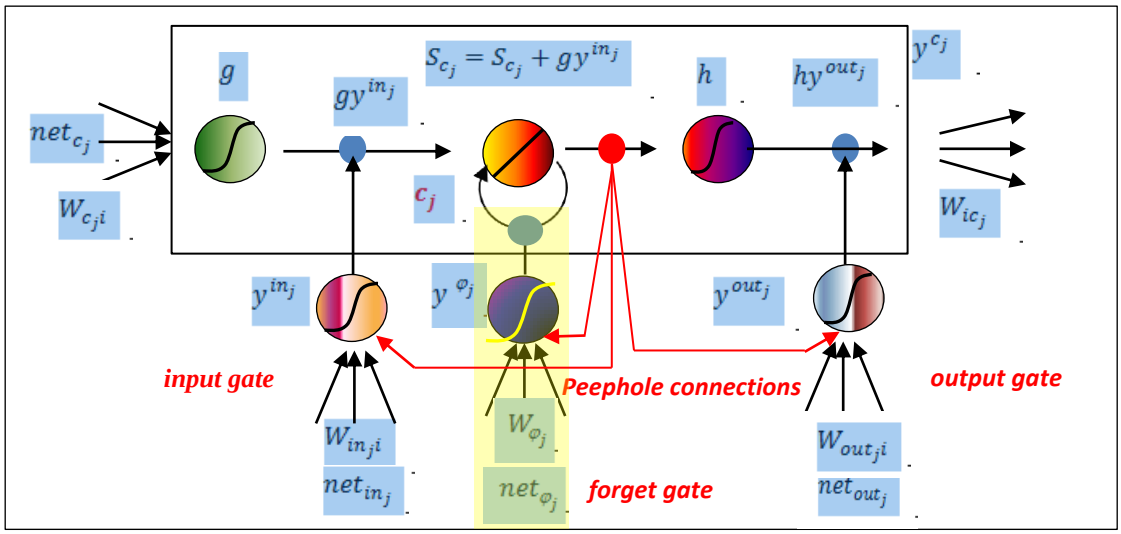


Figure 11.8 Memory block with only one cell for the extended LSTM with a *forget-gate* shaded yellow.

11.1 GATED RECURRENT UNIT (GRU)

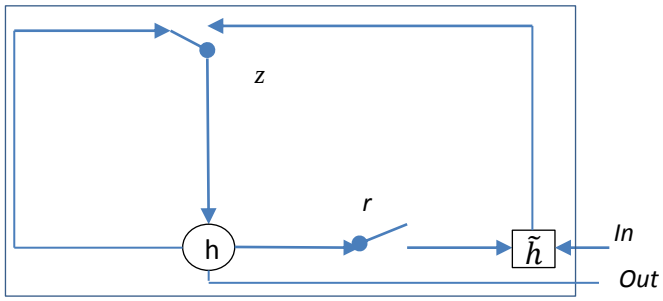


Figure 11.9 Gated Recurrent Unit (GRU) as a simpler, novel hidden unit. The update gate z selects whether the hidden state is to be updated with a new hidden state $\tilde{h}$. The reset gate r decides whether the previous hidden state is ignored.

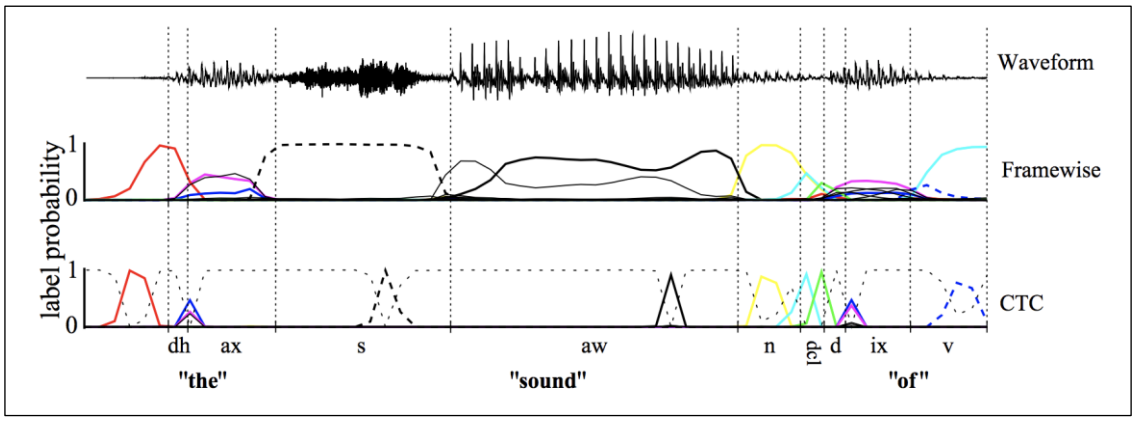


Figure 11.10 Classifying a speech signal with framewise and CTC methods. (Source: Courtesy of Graves et al.)

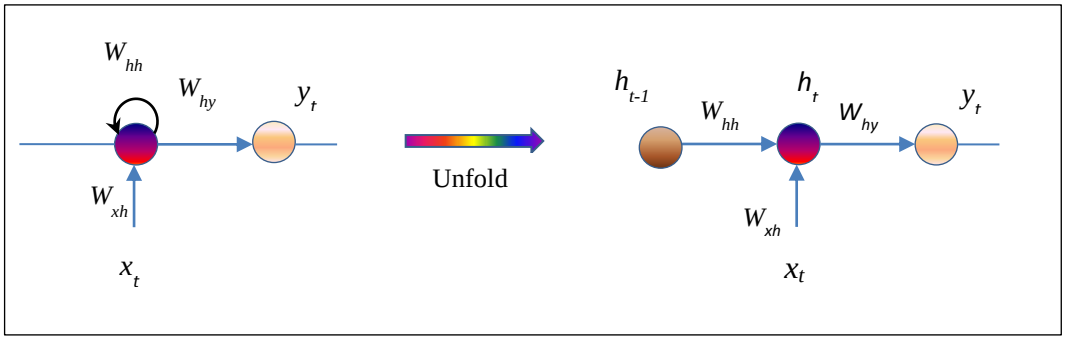


Figure 11.12 A regular RNN hidden unit.

12 Autoencoders

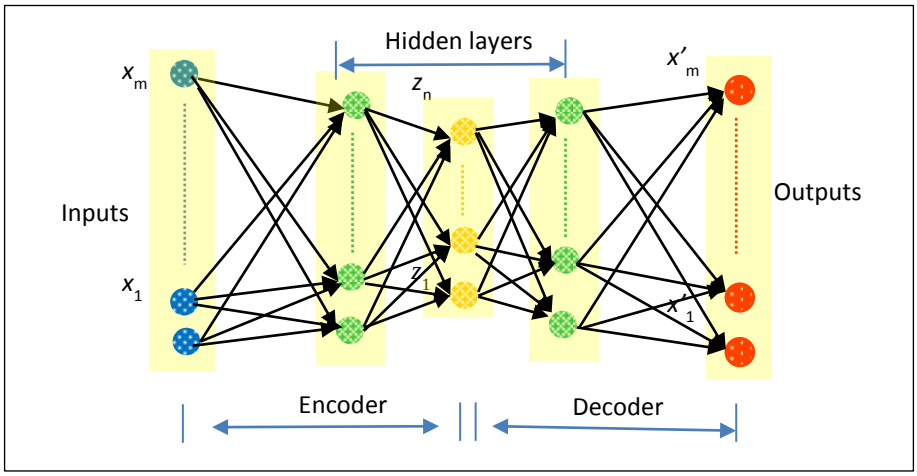


Figure 12.1 A generic autoencoder architecture (connections are illustrative).

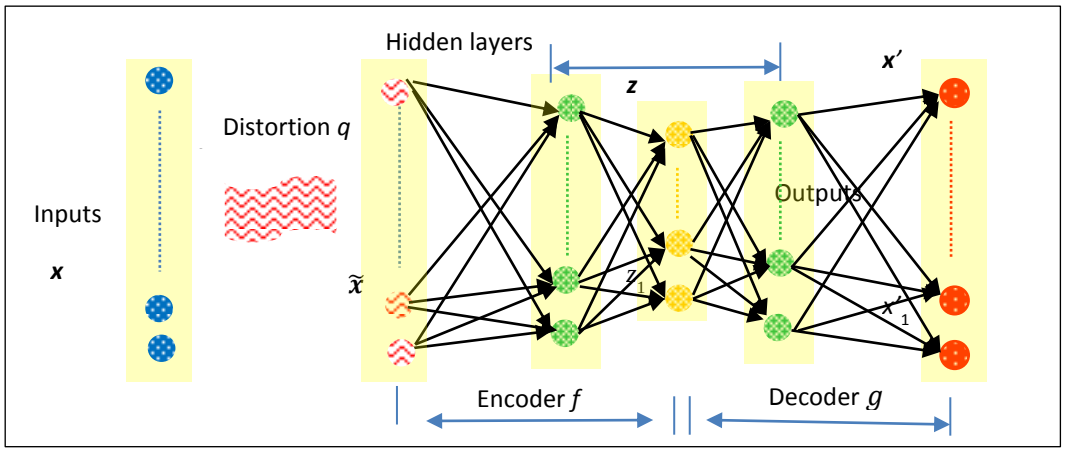


Figure 12.2 A denoising autoencoder architecture.

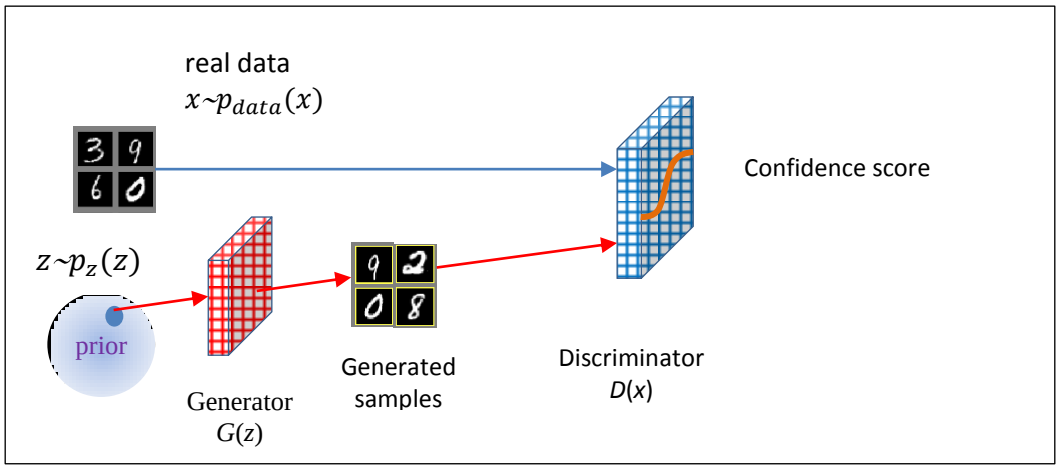


Figure 12.5 A generative adversarial network (GAN).

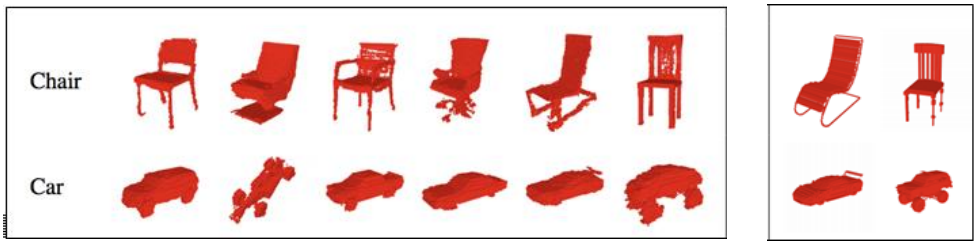


Figure 12.6 Objects generated by 3D-GAN models from vectors without referencing training set samples. Left: generated from the 3D GAN models. Right: nearest neighbors retrieved from the training set. (Source: Courtesy of Wu et al.)

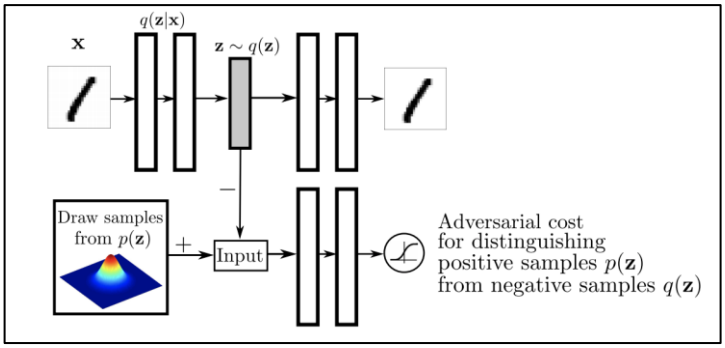


Figure 12.7 Adversarial autoencoders (AAEs). (Source: Courtesy of Makhzani et al.)

Appendix B Multi-Layer Perceptron Applied to the Fuel Economy Use Case

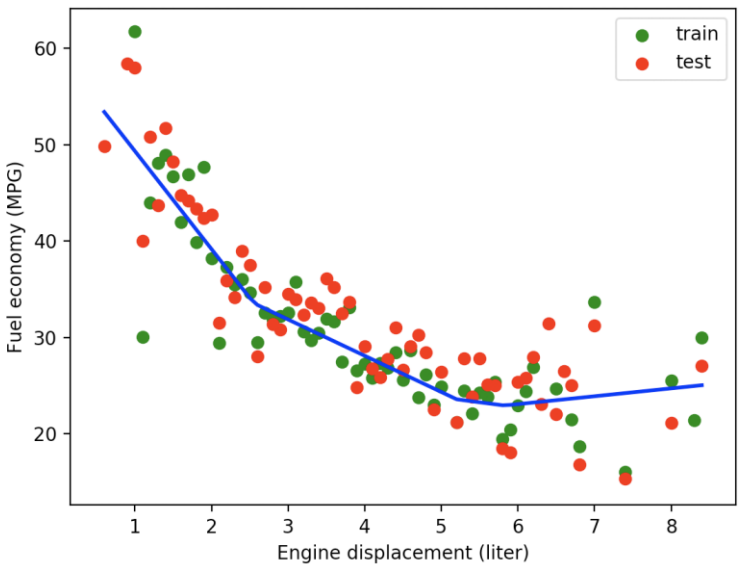


Figure B.1 The multi-layer perceptron model applied to the fuel economy machine learning use case.

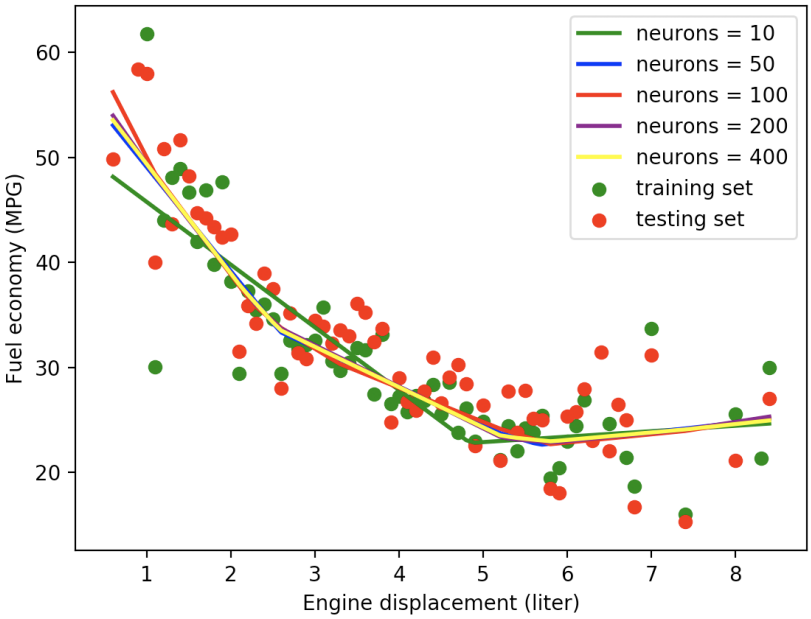


Figure B.2 Effects of varying number of neurons with a single layer perceptron.

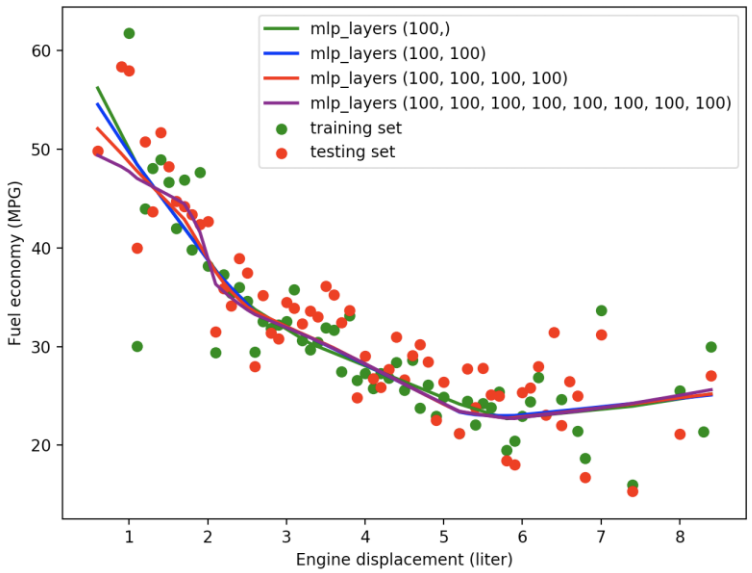


Figure B.3 Effects of varying number of layers with a multi-layer perceptron.

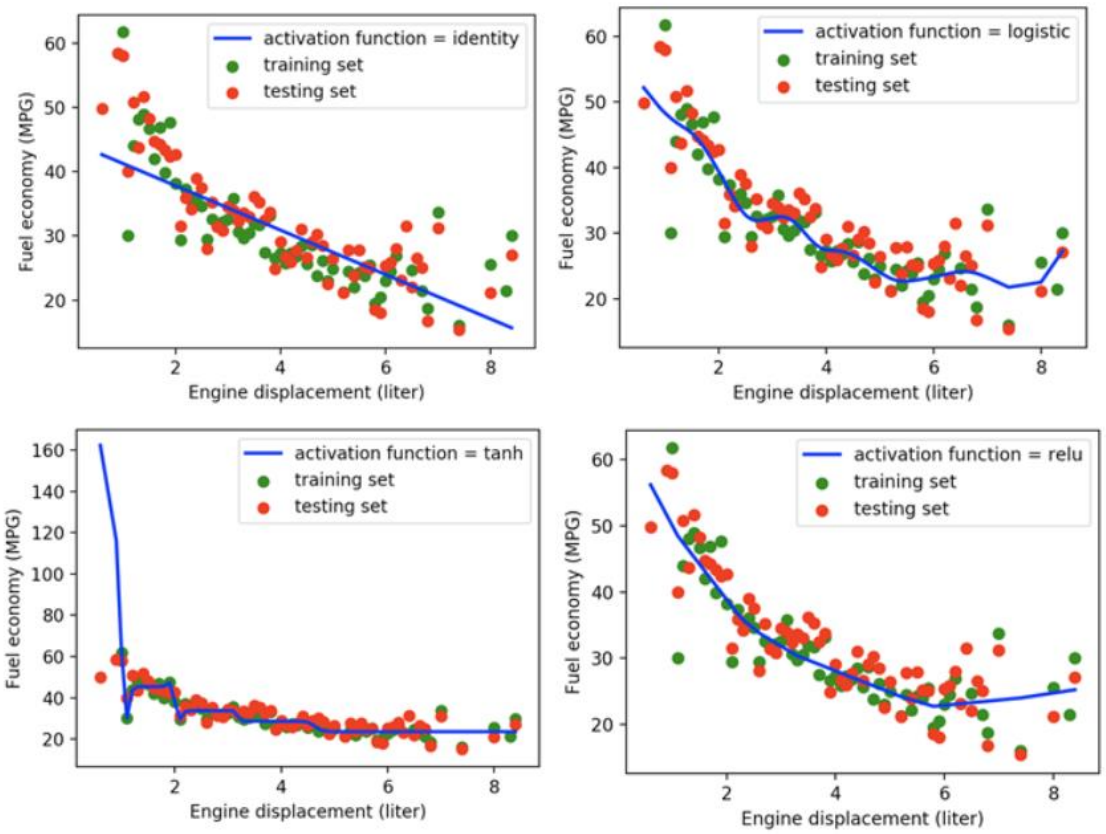


Figure B.4 Single-layer perceptron with different activation functions.

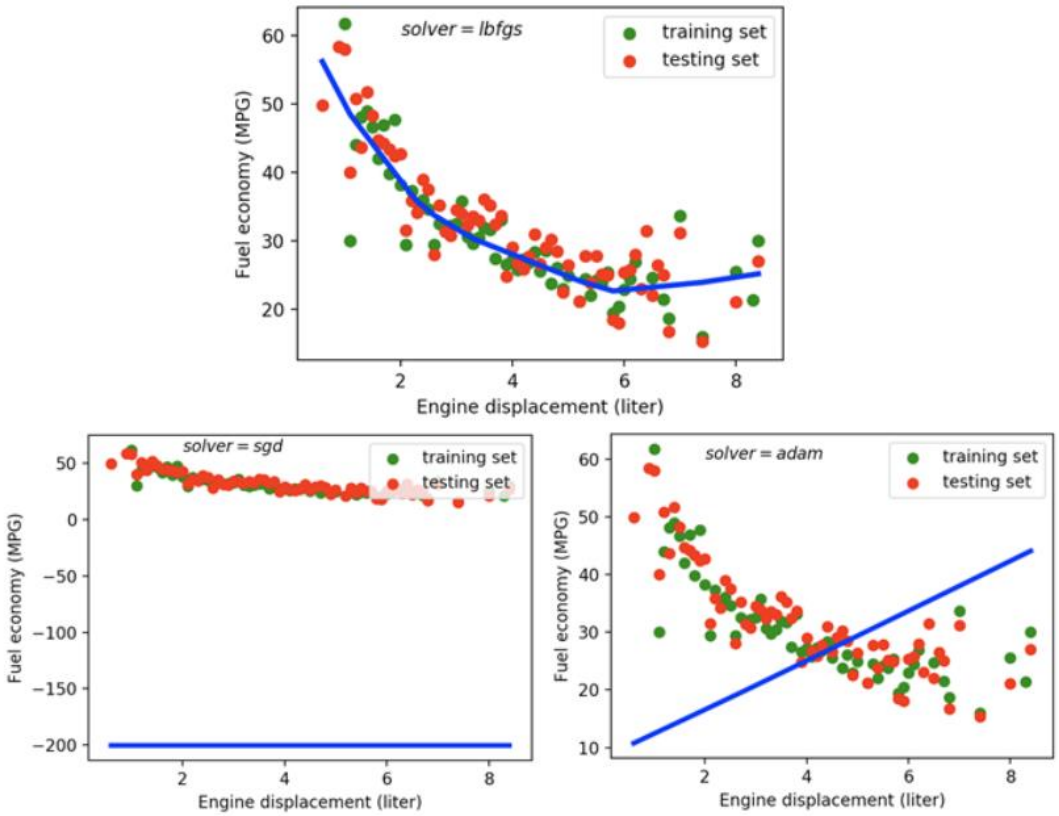


Figure B.5 Single-layer perceptron with different solvers.

Appendix C CNN Examples with Caffe, YOLOv3 and PyTorch

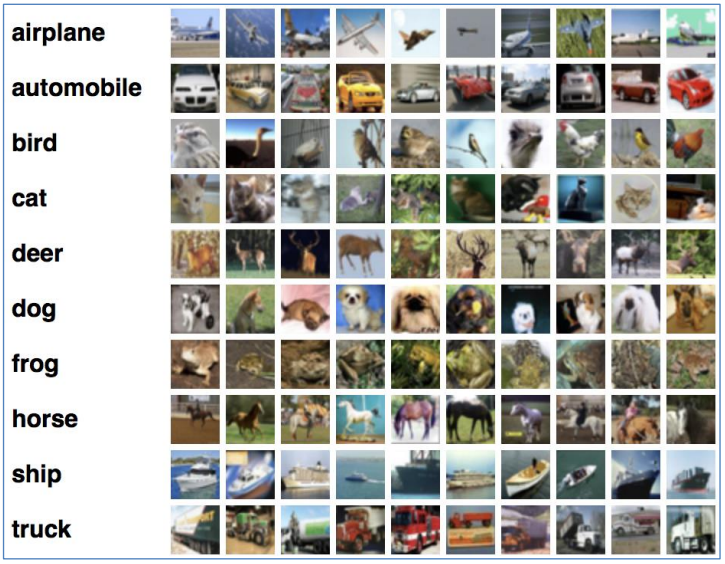


Figure C.2 Samples for Alex's CIFAR-10 dataset.

Figure C.3 Testing YOLOv3 recompiled with OpenCV3.4.0.

Figure C.4 A picture containing multiple objects.

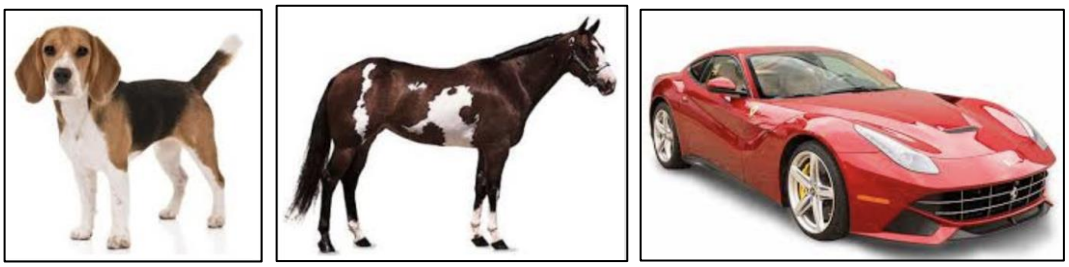


Figure C.5 Three separate images with one object per picture.

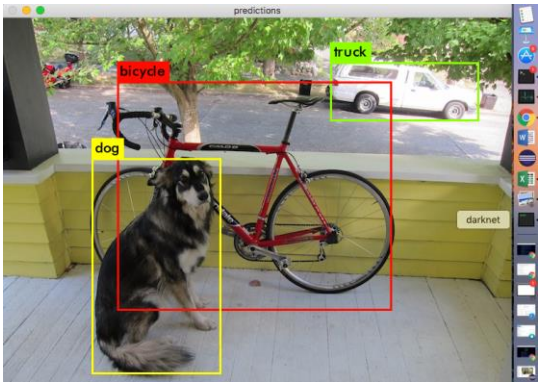


Figure C.6 Bounding boxes predicted by YOLOv3 with its own pre-trained weights.

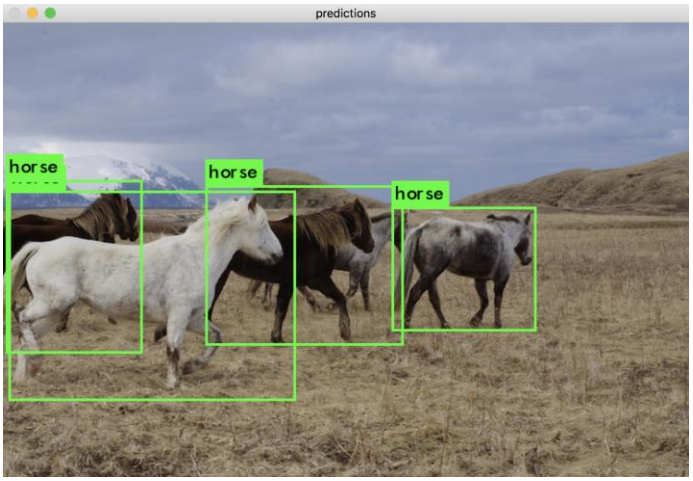


Figure C.7 Four horses detected by YOLO with its own pre-trained weights.

Figure C.9 IOU: Intersection over Union (Source: Courtesy of https://www.pyimagesearch.com/).

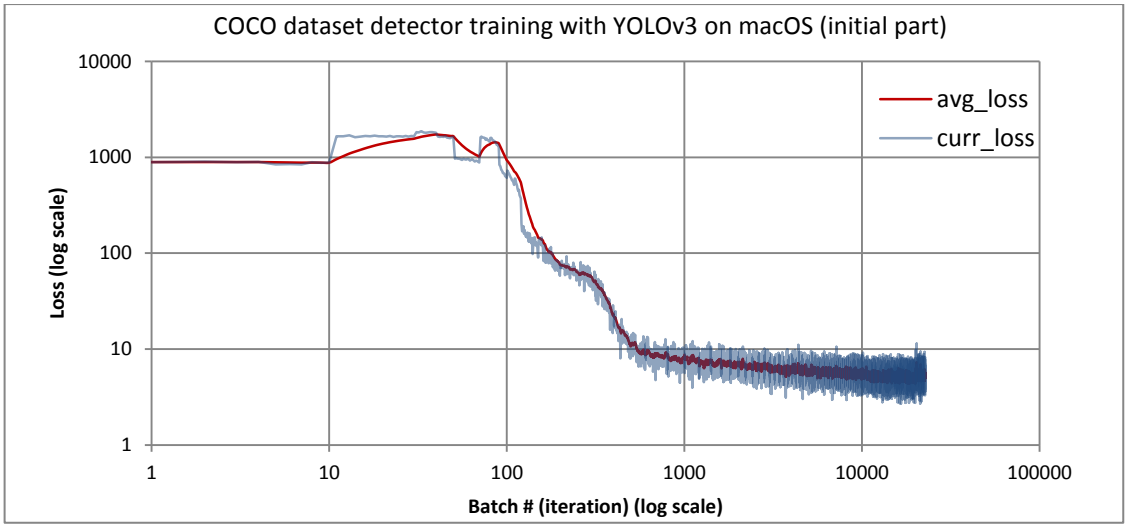


Figure C.18 COCO detector training with YOLOv3 optimized on macOS: current loss and average loss evolved with iterations from 1 up to 22756 with a batch size of 64 and subdivision of 16. The average loss started from over 1000 during the first 100 iterations and settled quickly down to around 8.0 at iteration 1000, which slowly progressed to 5.0 – 6.0 at the end of 22785 iterations.

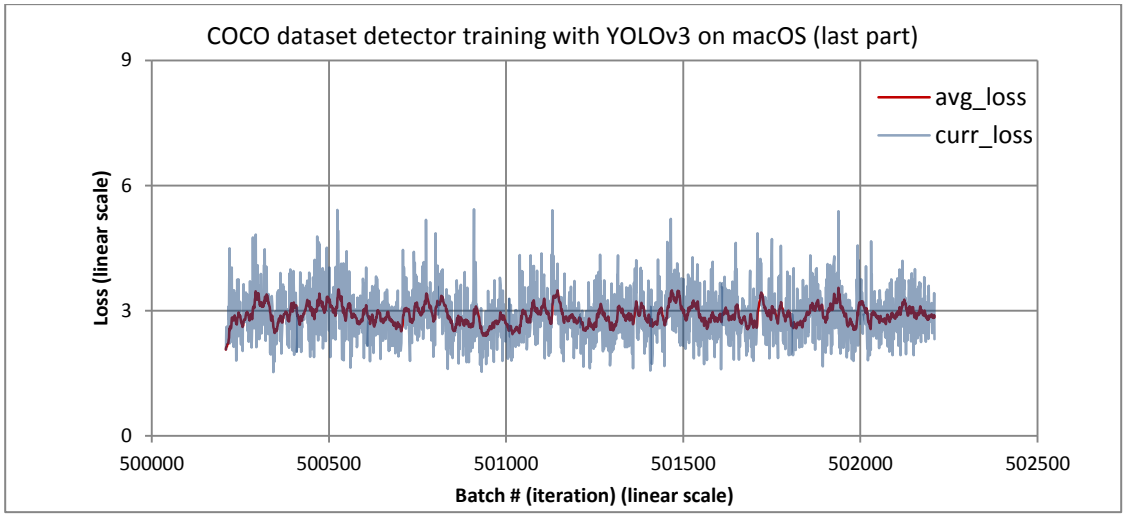


Figure C.19 COCO detector training with YOLOv3 optimized on macOS: current loss and average loss evolved with iterations from 500209 up to 502209 with a batch size of 64 and subdivision of 16. The losses settled down around 3.0.

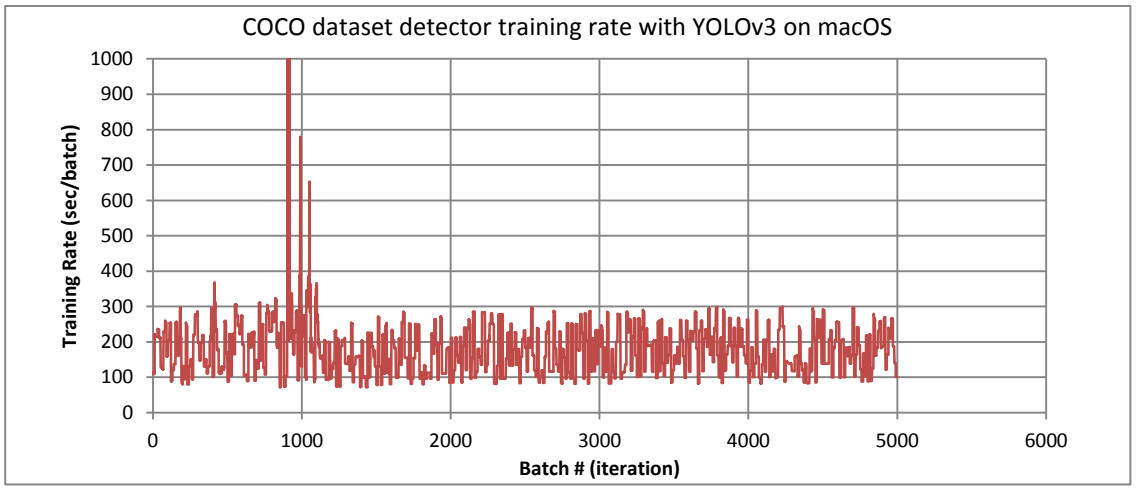


Figure C.20 COCO detector training with YOLOv3 optimized on macOS: training performance in terms of seconds per batch, with a batch size of 64 and subdivision of 16.

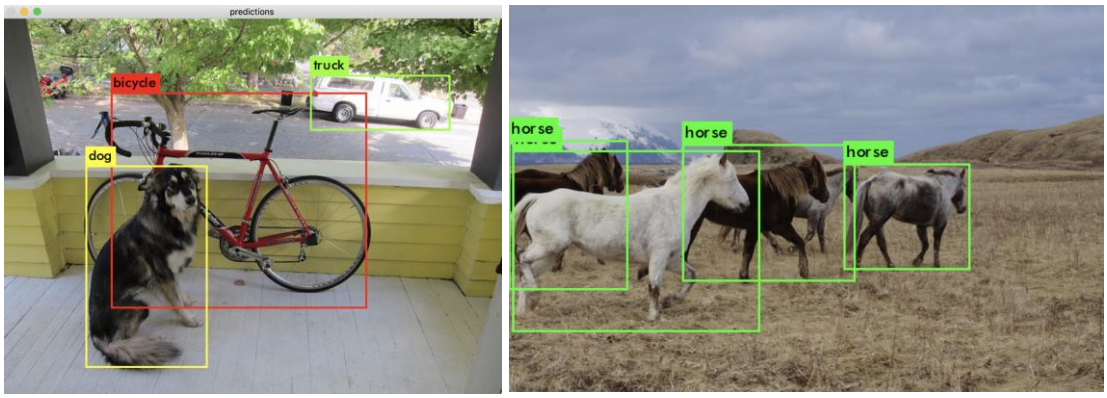


Figure C.21 Object detection test with YOLOv3 after fully trained with over 500k iterations, using the *dog.jpg* and *horses.jpg* image. All objects are detected correctly.

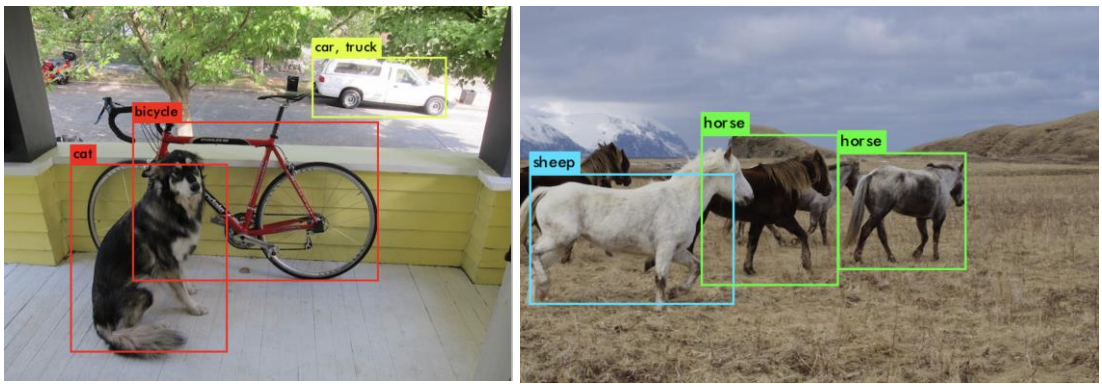


Figure C.22 Object detection test with YOLOv3 after trained for 10k iterations only, using: (a) the *dog.jpg* image, with the dog being mis-detected as a cat, and (b) using the *horses.jpg* image, with a horse mis-detected as a sheep.

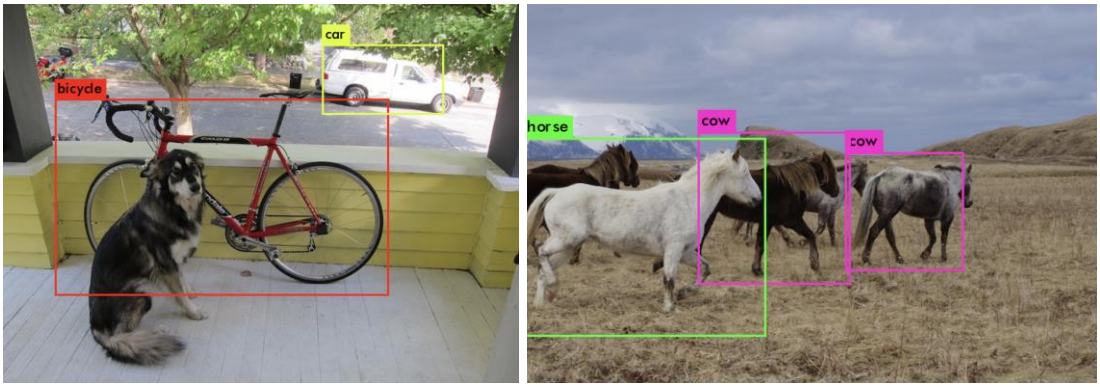


Figure C.23 Object detection test with YOLOv3 after trained for 20k iterations, using: (a) the *dog.jpg* image, with the dog not detected, and (b) the *horses.jpg* image, with two horses mis-detected as cows.

Figure C.24 Object detection test with YOLOv3 after trained for 1k iterations only, using: (a) the *dog.jpg* image, with no bounding boxes detected at all, and (b) the *horses.jpg* image, strangely enough that it actually mis-detected a "person" out of a horse.

Figure C.25 An image read and displayed with OpenCV in C in its original and inverted forms.

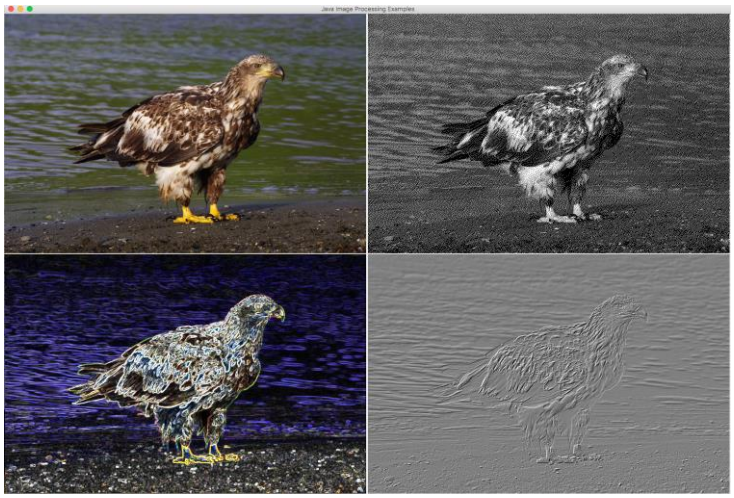


Figure C.27 An image read and processed with Marvin in Java in three different forms.

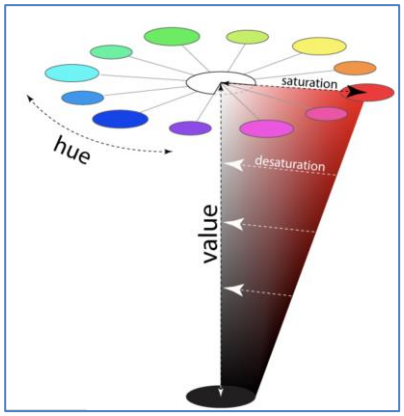


Figure C.29 The HSV color model.

Figure C.30 Adjust color from Preview – Tools menu on macOS.

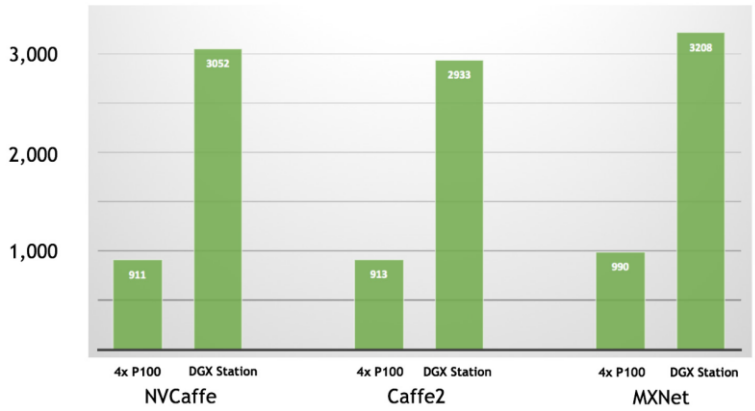


Figure C.31 NVidia 4x Tesla P100 (Pascal) versus DGX Station with 4x Tesla V100 (Volta). The y-axis represents the score in terms of images per second trained with the ResNet-50 training configuration with mixed precision FP16 and FP32.

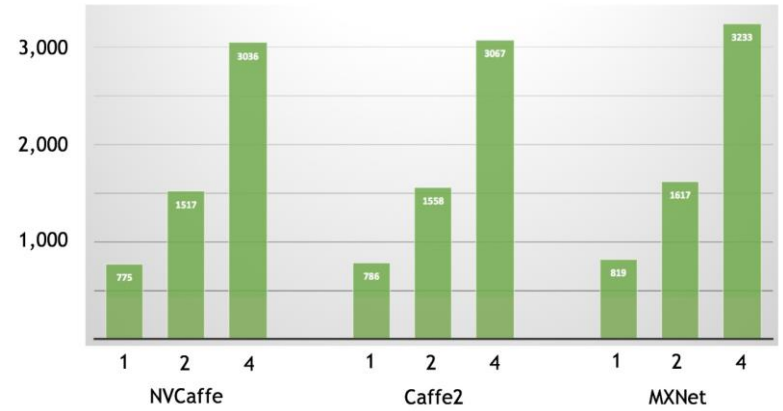


Figure C.32 NVidia 4x Tesla V100 (Pascal) scalability with 1, 2 and 4 GPUs. The y-axis represents the score in terms of images per second trained with the ResNet-50 training configuration with mixed precision FP16 and FP32.

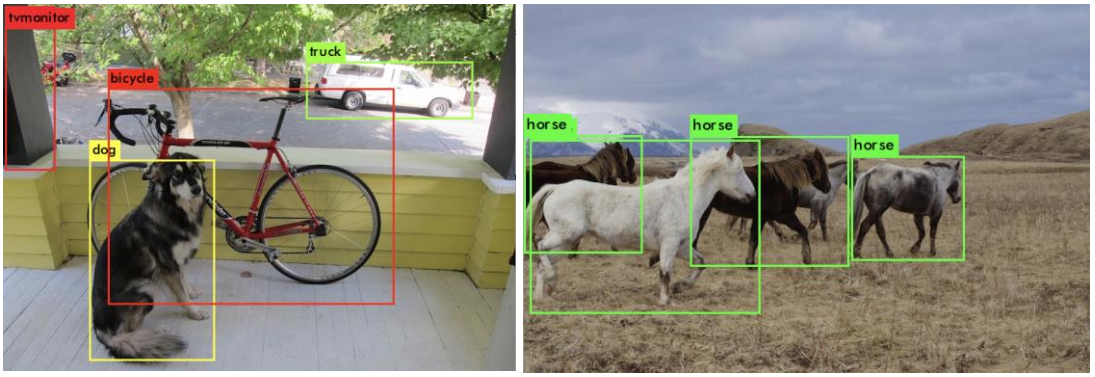


Figure C.33 Verification of the 4-GPU COCO training at the end of 520200 batches. The dog image had object:confidence scores of tvmonitor:67%, dog:98%, truck:83%, bicycle:99%, while the horses image had 99%, 97%, 95% and 85% for those four horses, respectively.

Figure C.35 A Ubuntu-18.04 desktop VM created with VMware Fusion on macOS.

Appendix D RNN/LSTM Example Implementations with Keras/TensorFlow

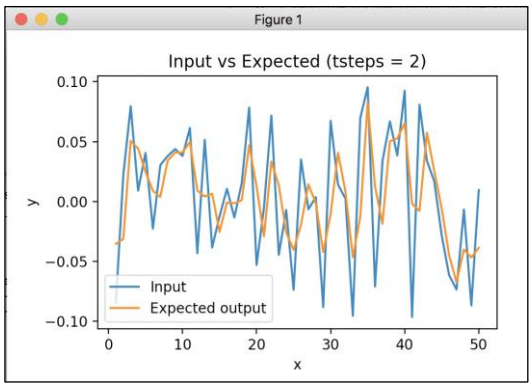


Figure D.2 Time series sequence input versus expected with a rolling window length of `tsteps = 2`.

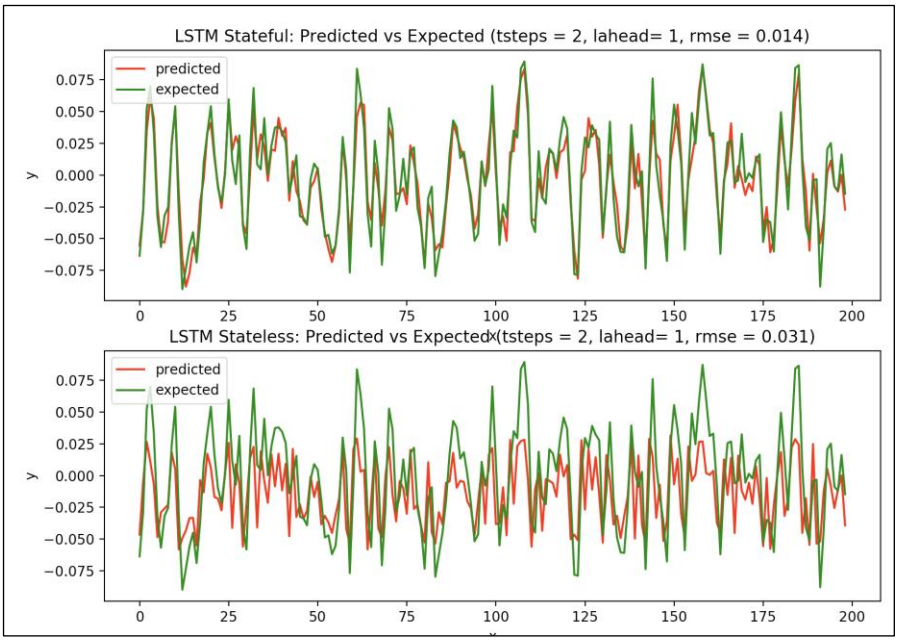


Figure D.3 Time series sequence modeled with stateful and stateless LSTM models, respectively.

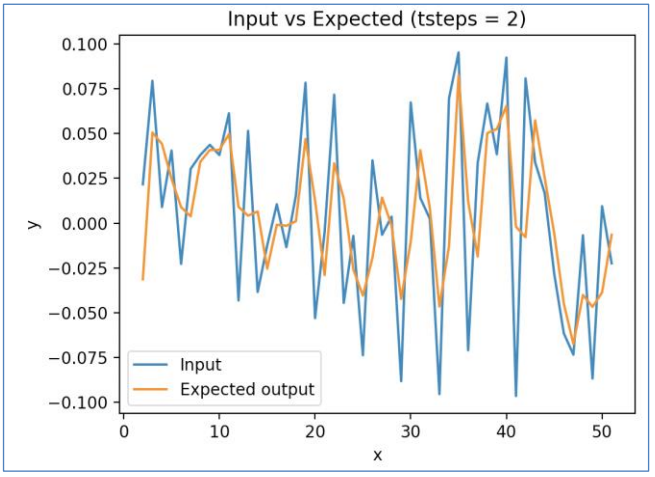


Figure D.4 Moving averaging of a random sequence with `tsteps` = 2.

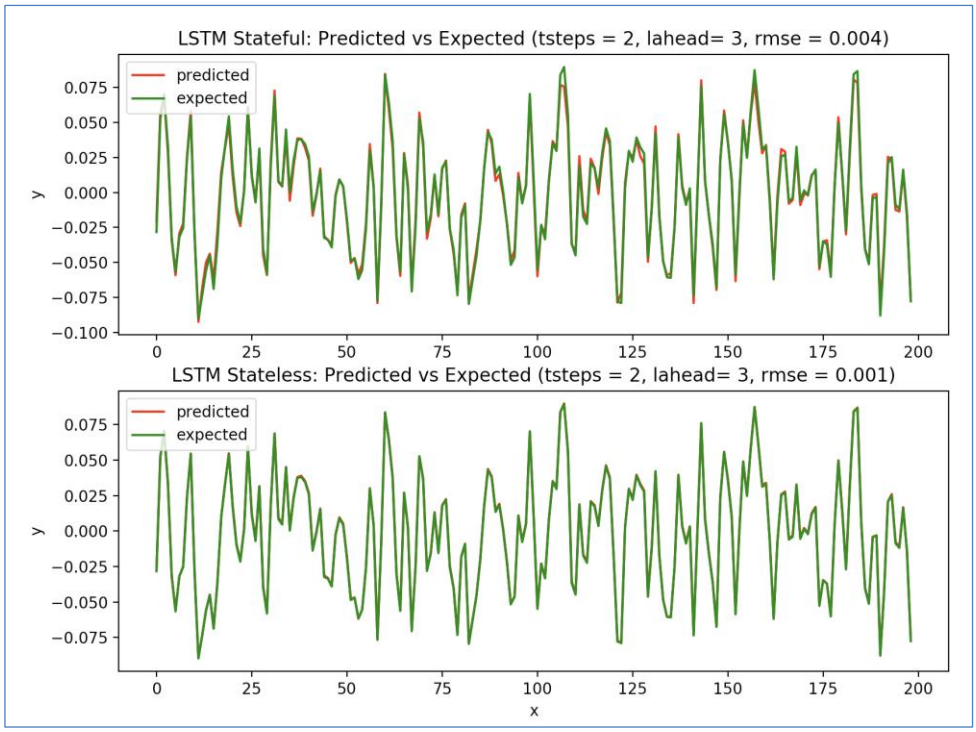


Figure D.5 Stateful versus stateless LSTM models for a random sequence with `tsteps` = 2 and `lahead` = 3.

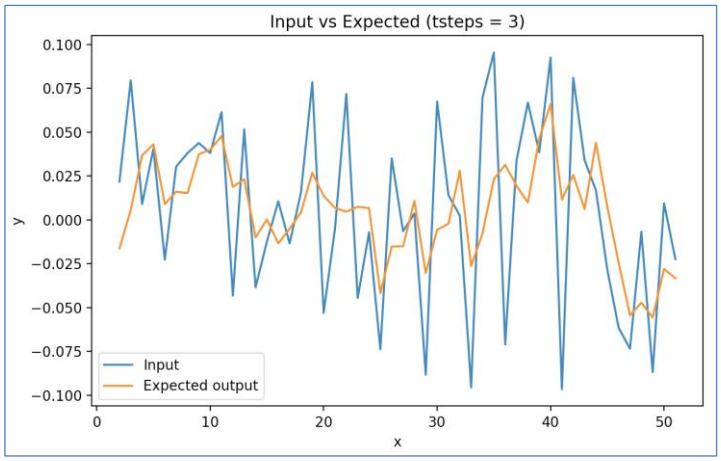


Figure D.6 Moving averaging of a random sequence with `tsteps` = 3.

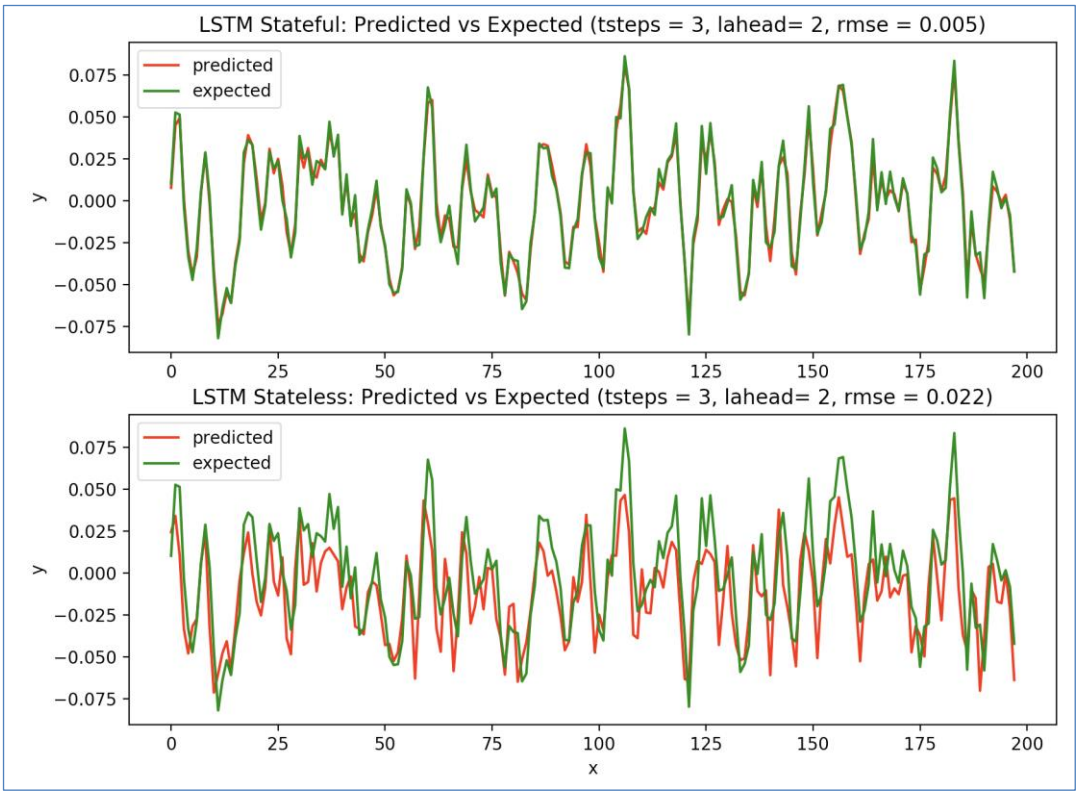


Figure D.7 Stateful versus stateless LSTM models for a random sequence with `tsteps` = 3 and `lahead` = 2.

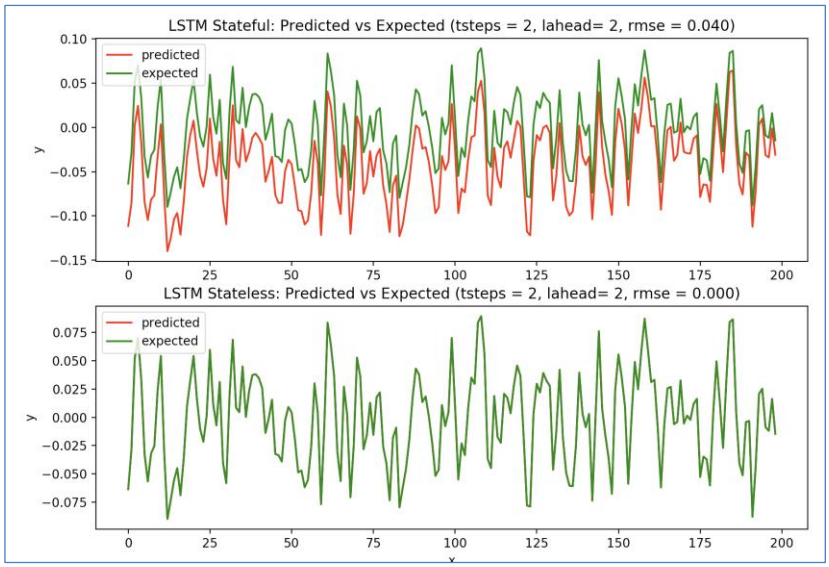


Figure D.8 Stateful versus stateless LSTM models for a random sequence with `tsteps` = 2 and `lahead` = 2.

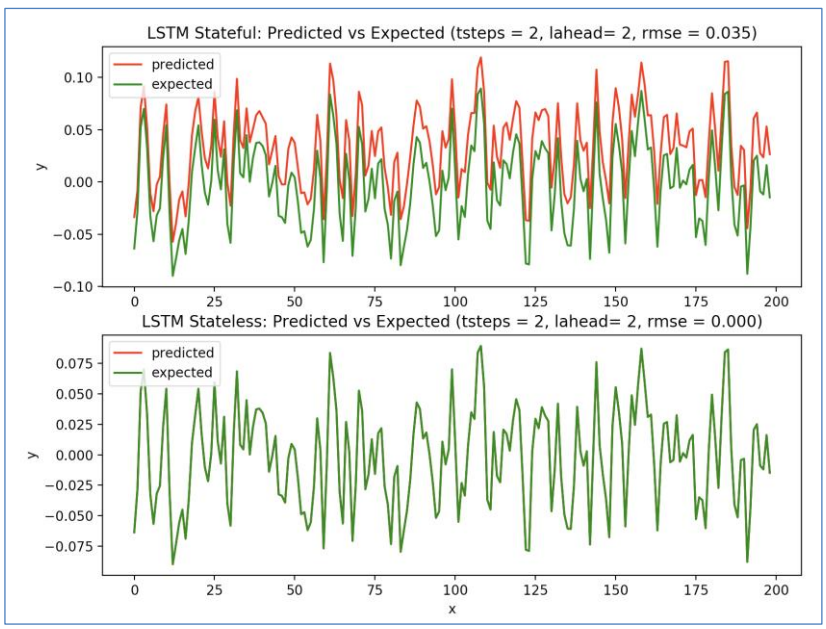


Figure D.9 Stateful versus stateless LSTM models for a random sequence with `tsteps` = 2 and `lahead` = 2 (second run)