Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Exercises Section 2.1

Problem 2.1.1 (Jupyter Notebook) Let

$$X = \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 7 \end{bmatrix}$$

Perform the following numpy operations:

(a) Define each of X, Y and Z as a 2d numpy arrays
(b) Concatenate X and Y in the row direction (X on top of Y)
(c) Reshape Z into a 2x3 matrix and concatenate X then reshapedZ in the column direction
(d) Concatenate X, Y and reshapedZ in the row direction
(e) Compute X +3*reshapedZ
(f) Compute X*reshapedZ (component-wise multiplication)
(g) Sum Z in the row direction and keep row dimension
(h) Sum Z in the column direction and keep column dimension

Hint: look at IntroML/Examples/Chapter2/NumpyDemo.ipynb

Solution: see IntroML/Exercises/Chapter2/Problem2.1.1.ipynb

Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Exercises Section 2.2

Problem 2.2.1 (Jupyter Notebook) Using numpy and matplotlib functionality plot the following functions for x between -5 and 5.

(a) $f(x) = \frac{1}{1+e^{-x}}$ - Hint: use numpy exp function
(b) $f(x) = \tanh(x)$ Hint: use numpy tanh function
(c) $f(x) = \log(1 + e^{x})$ Hint: use numpy log and exp functions
(d) $f(x) = \max(x, 0)$ Hint: use numpy maximum function

These are examples of functions that arise in this course.

Hint: look at IntroML/Examples/Chapter2/NumpyDemo.ipynb and MatplotlibBasicsDemo.ipynb

Solution: see IntroML/Exercises/Chapter2/Problem2.2.1

Problem 2.2.2 (Jupyter Notebook) The Heaviside step function is defined as:

$$H(x) = \begin{cases} 1 & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases}$$

(a) Using the numpy heaviside function, plot H(x) for x between -5 and 5.
(b) Consider the function:

$$h(x) = \begin{cases} \mathrm{x} & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases}$$

With the help of your result in (a) create a function using numpy functionality for h(x). Hint: multiply by Heaviside function with appropriate function. Plot h(x) for x between -5 and 5
(c) Create a function based on numpy heaviside for

$$G(x) = \begin{cases} 0 & if\ x \geq 0 \\ 1 & if\ x < 0 \end{cases}$$

Plot G(x) for x between -5 and 5.
(d) Using the same multiplication approach as in (b), use the results of (c) to create a function using numpy heaviside for

$$g(x) = \begin{cases} 0 & if\ x \geq 0 \\ e^{x} - 1 & if\ x < 0 \end{cases}$$

and plot for x between -5 and 5.
(e) Using the information in (a), (b), (c), (d), use the numpy functionality to create a function for

$$f(x) = \begin{cases} x & if\ x \geq 0 \\ e^{x} - 1 & if\ x < 0 \end{cases}$$

Plot f(x) for x between -5 and 5.

Hint: look at IntroML/Examples/Chapter2/NumpyDemo.ipynb and MatplotlibBasicsDemo.ipynb. You can get documentation by searching for numpy heaviside

Solution: see IntroML/Chapter2/Exercises/Problem2.2.2.ipynb

Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Exercises Section 2.3

Problem 2.3.1 (Jupyter Notebook) Repeat the calculations in IntroML/Examples/Chapter2/MatplotlibHeatmapDemo.ipynb for the case:

$$f(X) = \begin{cases} 1 & if\ X_1 - (X_0 - 0.5)^2 - 0.35 > 0 \\ 0 & if\ X_1 - (X_0 - 0.5)^2 - 0.35 \leq 0 \end{cases}$$

Solution: see IntroML/Exercises/Chapter2/Problem2.3.1.ipynb

Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Exercises Section 2.4

Problem 2.4.1 (Jupyter Notebook) Repeat the first part of IntroML/Examples/PandasDemo.ipynb. Read data from mydf1.csv.

(a) Output a Y numpy array that has the square of the entries in the "label" column of the file
(b) Output a X numpy array whose first column has the square root of the entries in the "feature 1" column and whose second column has the exponential of the entries in the "feature 2" column. Hint: process each column separately and concatenate.

Solution: see IntroML/Exercises/Chapter2/Problem2.4.1.ipynb

Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Exercises Section 2.5

Problem 2.5.1 (Jupyter Notebook) For the sigmoid function:

$(*) \quad f(x) = \frac{1}{1+e^{-x}}$

it can be shown that

$(**) \quad f'(x) = \frac{e^{-x}}{(1+e^{-x})^2}$

One can manipulate (*) to show that

$(***) \; f'(x) = f(x) - [f(x)]^2$

Create a unit test to confirm that (**) and (***) are equivalent. Hint: generate x values between -5 and 5 and assert that the maximum absolute difference between (**) and (***) for these x values is less than a tolerance that you specify. Hint: use numpy max to find the maximum of a numpy array.

Hint: see IntroML/Examples/Chapter2/unittestDemo.ipynb

Solution: see IntroML/Exercises/Chapter2/Problem2.5.1.ipynb

Exercises Section 2.6

Problem 2.6.1 (Jupyter Notebook) Consider the messages:
"A bird in the hand is worth 2 in the bush"
"The early bird gets the worm"
"A cat has nine lives"
"Honesty is the best policy"

Repeat the countvectorizer example in IntroML/Examples/Chapter2/sklearnDemo.ipynb using stop words ("the", "a", "is" "in"). Hint: use stop_words= list of stop words in call to CountVectorizer.

Solution: See IntroML/Exercises/Chapter2/Problem2.6.1.ipynb

Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Exercises Section 3.1

Problem 3.1.1 (Theory or Jupyter Notebook) Let

$$X = \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 7 \end{bmatrix}$$

Compute

(a) $X + Y$
(b) $Z + Y^T$
(c) XZ
(d) $X + Z^T$
(e) ZX
(f) $X^T Z^T$
(g) $ZX + Y$

Hint: look at IntroML/Examples/Chapter3/LinearAlgebra.ipynb

Solution: see IntroML/Exercises/Chapter3/Problem3.1.1.ipynb or see next page for written solutions

$$X = \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 7 \end{bmatrix}$$

Solution:

(a) $X + Y$

Use broadcasting:

$$\begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 1+1 & 2+2 & 4+3 \\ -2+1 & -5+2 & -8+3 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 7 \\ -1 & -3 & -5 \end{bmatrix}$$

(b) $Z + Y^T$

Use broadcasting:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 7 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1+1 & 2+1 \\ 3+2 & 4+2 \\ 5+3 & 7+3 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 5 & 6 \\ 8 & 10 \end{bmatrix}$$

(c) XZ

$$\begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 7 \end{bmatrix} = \begin{bmatrix} (1)(1)+(2)(3)+(4)(5) & (1)(2)+(2)(4)+(4)(7) \\ (-2)(1)+(-5)(3)+(-8)(5) & (-2)(2)+(-5)(4)+(-8)(7) \end{bmatrix} = \begin{bmatrix} 27 & 38 \\ -57 & -80 \end{bmatrix}$$

(d) $X + Z^T$

$$\begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} + \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 7 \end{bmatrix} = \begin{bmatrix} 1+1 & 2+3 & 4+5 \\ -2+2 & -5+4 & -8+7 \end{bmatrix} = \begin{bmatrix} 2 & 5 & 9 \\ 0 & -1 & -1 \end{bmatrix}$$

(e) ZX

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 7 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} = \begin{bmatrix} (1)(1)+(2)(-2) & (1)(2)+(2)(-5) & (1)(4)+(2)(-8) \\ (3)(1)+(4)(-2) & (3)(2)+(4)(-5) & (3)(4)+(4)(-8) \\ (5)(1)+(7)(-2) & (5)(2)+(7)(-5) & (5)(4)+(7)(-8) \end{bmatrix} = \begin{bmatrix} -3 & -8 & -12 \\ -5 & -14 & -20 \\ -9 & -25 & -36 \end{bmatrix}$$

(f) $X^T Z^T$

$$\begin{bmatrix} 1 & -2 \\ 2 & -5 \\ 4 & -8 \end{bmatrix} \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 7 \end{bmatrix} = \begin{bmatrix} (1)(1)+(-2)(2) & (1)(3)+(-2)(4) & (1)(5)+(-2)(7) \\ (2)(1)+(-5)(2) & (2)(3)+(-5)(4) & (2)(5)+(-5)(7) \\ (4)(1)+(-8)(2) & (4)(3)+(-8)(4) & (4)(5)+(-8)(7) \end{bmatrix} = \begin{bmatrix} -3 & -5 & -9 \\ -8 & -14 & -25 \\ -12 & -20 & -36 \end{bmatrix}$$

(g) $ZX + Y$

Use broadcasting for the addition

$$\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 7\end{bmatrix}\begin{bmatrix}1 & 2 & 4\\ -2 & -5 & -8\end{bmatrix}+[1 \quad 2 \quad 3]=\begin{bmatrix}-3 & -8 & -12\\ -5 & -14 & -20\\ -9 & -25 & -36\end{bmatrix}+[1 \quad 2 \quad 3]=$$
$$\begin{bmatrix}-3+1 & -8+2 & -12+3\\ -5+1 & -14+2 & -20+3\\ -9+1 & -25+2 & -36+3\end{bmatrix}=\begin{bmatrix}-2 & -6 & -9\\ -4 & -12 & -17\\ -8 & -23 & -33\end{bmatrix}$$

Exercises Section 3.2

Problem 3.2.1 (Theory) Compute the gradient of

$$L(W_0, W_1) = W_0^4 + 3W_1^4 + 2W_0^2 W_1^2$$

Problem 3.2.2 (Theory) Let $L = 2e^{Z_0} + Z_1$ and assume:

$$Z = WX + b \text{ where } W = [W_0 \quad W_1], X = \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix}$$

(a) Compute gradients $\nabla_W L,\ \nabla_X L,\ \nabla_b L$ directly by substituting for Z_0 and Z_1
(b) Compute gradients $\nabla_W L,\ \nabla_X L,\ \nabla_b L$ using the chain rule

See next page for solutions

Problem 3.2.1
Solution:

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial W_0} & \frac{\partial L}{\partial W_1} \end{bmatrix} = [4W_0^3 + 4W_0W_1^2 \quad 12W_1^3 + 4W_0^2W_1\ ]$$

Problem 3.2.2
Solution:

(a) Compute gradients $\nabla_W L,\ \nabla_X L,\ \nabla_b L$ directly by substituting for Z_0 and Z_1

First, we compute Z and then L

$$Z = [Z_0 \quad Z_1] = WX + b = [W_0 \quad W_1]\begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} + b =$$
$$[W_0X_{00} + W_1X_{10} + b \quad W_0X_{01} + W_1X_{11} + b]$$

$$L = 2e^{Z_0} + Z_1 = 2e^{W_0X_{00}+W_1X_{10}+b} + W_0X_{01} + W_1X_{11} + b$$

Compute gradients:

$$\nabla_W L = \begin{bmatrix} \frac{\partial L}{\partial W_0} & \frac{\partial L}{\partial W_1} \end{bmatrix} = [2X_{00}e^{W_0X_{00}+W_1X_{10}+b} + X_{01} \quad 2X_{10}e^{W_0X_{00}+W_1X_{10}+b} + X_{11}]$$

$$\nabla_X L = \begin{bmatrix} \frac{\partial L}{\partial X_{00}} & \frac{\partial L}{\partial X_{01}} \\ \frac{\partial L}{\partial X_{10}} & \frac{\partial L}{\partial X_{11}} \end{bmatrix} = \begin{bmatrix} 2W_0e^{W_0X_{00}+W_1X_{10}+b} & W_0 \\ 2W_1e^{W_0X_{00}+W_1X_{10}+b} & W_1 \end{bmatrix}$$

$$\nabla_b L = \begin{bmatrix} \frac{\partial L}{\partial b} \end{bmatrix} = [2e^{W_0X_{00}+W_1X_{10}+b} + 1]$$

(b) Compute gradients $\nabla_W L,\ \nabla_X L,\ \nabla_b L$ using the chain rule

First, compute

$$\nabla_Z L = \begin{bmatrix} \frac{\partial L}{\partial Z_0} & \frac{\partial L}{\partial Z_1} \end{bmatrix} = [2e^{Z_0} \quad 1]$$

Now compute gradients:

$$\nabla_W L = \nabla_Z L X^T = [2e^{Z_0} \quad 1]\begin{bmatrix} X_{00} & X_{10} \\ X_{01} & X_{11} \end{bmatrix} = [2X_{00}e^{Z_0} + X_{01} \quad 2X_{10}e^{Z_0} + X_{11}] =$$
$$[2X_{00}e^{W_0X_{00}+W_1X_{10}+b} + X_{01} \quad 2X_{10}e^{W_0X_{00}+W_1X_{10}+b} + X_{11}]$$

This is the same as $\nabla_W L$ in (a)

$$\nabla_X L = W^T \nabla_Z L = \begin{bmatrix} W_0 \\ W_1 \end{bmatrix}[2e^{Z_0} \quad 1] = \begin{bmatrix} 2W_0e^{Z_0} & W_0 \\ 2W_1e^{Z_0} & W_1 \end{bmatrix} = \begin{bmatrix} 2W_0e^{W_0X_{00}+W_1X_{10}+b} & W_0 \\ 2W_1e^{W_0X_{00}+W_1X_{10}+b} & W_1 \end{bmatrix}$$

This is the same as $\nabla_X L$ in (a)

$$\nabla_b L = \sum_j \nabla_Z L_j = \frac{\partial L}{\partial Z_0} + \frac{\partial L}{\partial Z_1} = 2e^{Z_0}+1 = 2e^{W_0X_{00}+W_1X_{10}+b} + 1$$

This is the same as in (a)

Exercises Section 3.3

Problem 3.3.1 (Jupyter Notebook) Use the Gradient Descent algorithm to find a minimum for

$$L(W_0, W_1) = W_0^4 + 3W_1^4 + 2W_0^2 W_1^2$$

You pick the initial guess, the number of epochs and learning rate. The minimum is at $W_0 = 0, W_1 = 0$. Hint: use IntroML/Examples/Chapter3/Optimization.ipynb as a starting point. You may see an overflow error, which means a blow up occurs. Change the learning rate appropriately.

Hint: see gradient result in Problem 3.2.1

Solution: see IntroML/Exercises/Chapter3/Problem3.3.1.ipynb

Exercises Section 4.2

Problem 4.2.1 (Theory) Consider the Log-Cosh Loss function defined as:

$$L = \frac{1}{m}\sum_{j=0}^{m-1} \log\left(\cosh\left(A_j - Y_j\right)\right)$$

(a) Show that $L \geq 0$.
(b) Compute the gradient $\nabla_A L$.

Solution: see next page

Problem 4.2.2 (Jupyter Notebook) Consider linear regression using the following training data

$X = \begin{bmatrix} 3 & 1 & -1 \\ 1 & -2 & 2 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$

Assume that

$W = \begin{bmatrix} W_0 & W_1 \end{bmatrix} = \begin{bmatrix} 2 & -1 \end{bmatrix} \quad b = -1$

Use the Log-Cosh loss function defined above for this problem.

(a) Perform forward propagation for the above W, b.
(b) Compute the Loss function after forward propagation.
(c) Perform back propagation for the above training data and parameter matrices to determine $\nabla_W L$ and $\nabla_b L$.
(d) Perform 1 epoch of training using Gradient Descent with learning rate of 0.2 and recompute the loss function with the updated W, b
(e) Compute the prediction based on input feature matrix X above and the updated W, b from (d)
(f) Compute the accuracy of the prediction in (e) when compared against the actual Y specified above.

Hint: look at IntroML/Examples/Chapter4/LinearRegression.ipynb

Solution: see IntroML/Exercises/Chapter4/Problem4.2.2.ipynb

Problem 4.2.1
Solution

(a) Show that $L \geq 0$.
Recall that cosh() is the hyperbolic cosine. It is defined as:

$$\cosh(x) = \frac{e^x + e^{-x}}{2}$$

You can plot $\cosh(x)$ or use calculus to find the minimum. It can be shown that the minimum is 1 so that $\cosh(x) \geq 1$ for $-\infty < x < \infty$. Hence, $\log(\cosh(x)) \geq 0$ for all $-\infty < x < \infty$.

(b) Compute the gradient $\nabla_A L$.
If

$$f(x) = \cosh(x) = \frac{e^x + e^{-x}}{2}$$

then

$$f'(x) = \frac{e^x - e^{-x}}{2} = \sinh(x)$$

Using this result and applying the chain rule and the rule for logs, we get the following for the gradient of L:

$$\frac{\partial L}{\partial A_j} = \frac{1}{m}\frac{\sinh(A_j - Y_j)}{\cosh(A_j - Y_j)} = \frac{1}{m}\tanh(A_j - Y_j)$$

sinh() is the hyperbolic sine function and tanh() is the hyperbolic tangent function.

Exercises Section 4.3

Problem 4.3.1 (Jupyter Notebook) For the Problem 4.2.2 compute the derivatives $\frac{\partial L}{\partial W_0}$, $\frac{\partial L}{\partial W_1}$, $\frac{\partial L}{\partial b}$ using the centred differences method with $\varepsilon = 0.1$ and compare with derivatives found in 4.2.2(c). The loss function is:

$$L = \frac{1}{m}\sum_{j=0}^{m-1} \log\left(\cosh\left(A_j - Y_j\right)\right)$$

Hint: see IntroML/Examples/Chapter4/DerivativeTesting.ipynb

Solution: see IntroML/Exercises/Chapter4/Problem4.3.1.ipynb

Exercises Section 4.5

Problem 4.5.1 (Programming) Add the Log-Cosh Loss function from problem 4.2.1 to the course framework (specifically add to the functions_loss.py file).

Hint: see Problem 4.2.1 and solution for the gradient of the Log-Cosh function.

Solution: See IntroML/Exercises/Chapter4/functions_loss.py

Problem 4.5.2 (Programming) Add a test case for linear regression using the Log-Cosh loss function to the unittest_forwardbackprop.py file and run to confirm test passes.

Solution: See IntroML/Exercises/Chapter4/unittest_forwardbackprop_logcash.py

Hint: run these solution files in a copy of folder Code/Version1.1. Overwrite the existing functions_loss.py with the version that you create.

Exercises Section 4.6

Problem 4.6.1 (Programming) Modify the prediction and accuracy methods in NeuralNetwork_Base to be able to handle the Log-Cosh Loss function. The formulas for prediction and accuracy are the same as those for “meansquarederror” loss.

Solution: See IntroML/Exercises/Chapter4/NeuralNetwork_Base.py

Problem 4.6.2 (Programming) Modify driver_linearregression.py to use the Log-Cosh loss function and re-run with the default parameters. How does final loss and accuracy compare with the mean squared error loss case? Adjust the learning rate or number of epochs if needed.

Solution: See IntroML/Exercises/Chapter4/driver_linearregression_logcash.py. See also discussion on next page after you have investigated yourself.

Hint: run these solution files in a copy of folder Code/Version1.2 (make sure that the Log-Cosh loss function is available in the functions_loss.py file)

Problem 4.6.2
Solution: Default settings have learning rate of 0.5 for Gradient Descent and 50 epochs for training. Here are some notes

(a) Let us not compare values of the loss function for mean squared error and log-cosh as they are different functions.
(b) We can compare accuracy for runs with mean squared error and log-cosh loss, as accuracy is mean absolute error in both cases.
(c) I find that with default settings, accuracy is worse (higher) for log-cosh case than for mean squared error case. If I increase learning rate to 1 for log-cosh case, then accuracy after 50 epochs is lower in general. Note: because of the random initial W and b, your results may vary.

Exercises Section 4.7

Problem 4.7.1 (Jupyter Notebook) Consider binary classification using Logistic Regression and the following training data

$$X = \begin{bmatrix} 3 & 1 & -1 \\ 1 & -2 & 2 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}$$

Assume that

$$W = \begin{bmatrix} W_0 & W_1 \end{bmatrix} = \begin{bmatrix} 2 & -1 \end{bmatrix} \quad b = -1$$

(a) Perform forward propagation using above training data and parameters.
(b) Compute the value of the loss function (binary cross entropy) after forward propagation.
(c) Perform back propagation for the above training data and parameter matrices to determine $\nabla_W L$ and $\nabla_b L$.
(d) Perform 1 epoch of training using Gradient Descent with learning rate of 0.1 and recompute the loss function with the updated W, b
(e) Compute the prediction based on input feature matrix X above after the 1 epoch
(f) Compute the accuracy of the prediction in (e) when compared against the actual Y specified above.
(g) Compute the derivatives $\frac{\partial L}{\partial W_0}$, $\frac{\partial L}{\partial W_1}$, $\frac{\partial L}{\partial b}$ using the centred differences method with $\varepsilon = 0.1$.
(h) Focusing on the derivative $\frac{\partial L}{\partial W_0}$, redo the calculation in (f) with $\varepsilon = 0.02$, 0.01, 0.05. Confirm that the error (absolute difference) in the approximate derivative when compared to the actual derivative computed in (c) decreases by a factor of 4 when ε is cut in half.

Hint: use IntroML/Examples/Chapter4/LogisticRegression.ipynb and DerivativeTesting.ipynb as starting points.

Solution: See IntroML/Exercises/Chapter4/Problem4.7.1.ipynb

Exercises Section 4.8

Problem 4.8.1 (Theory/Programming) Let us define a new activation function "mirrorsigmoid" defined as

$$f(z) = \frac{1}{1 + e^{z}}$$

This activation function is to be used for binary classification.

(a) Compute the derivative f'(z). If A = f(Z), then express derivative in terms of A.
(b) Add "mirrorsigmoid" to the activation and activation_der functions in the functions_activation in the framework.
(c) Add a test case for logistic regression using the "mirrorsigmoid" function in unittest_forwardbackprop
(d) Create a new driver for logistic regression model using the "mirrorsigmoid" activation function. How does this logistic regression model compare with the standard approach using "sigmoid" activation? Try a few cases and compare.

Solution: for (a) see next page.

See files IntroML/Exercises/Chapter4/functions_activation.py, unittest_forwardbackprop_mirrorsigmoid.py, driver_logisticregression_mirrrorsigmoid.py

Hint: run these solution files in a copy of folder Code/Version1.3. Over-write the existing functions_activation.py with the version that you create.

Solution

(a) Compute the derivative f'(z). If A = f(Z), then express derivative in terms of A.

Using the rules of calculus:

$$f'(z) = \frac{-e^z}{(1+e^z)^2}$$

If we manipulate the above expression, we get

$$f'(z) = \frac{-e^z}{(1+e^z)^2} = \frac{-1-e^z}{(1+e^z)^2} + \frac{1}{(1+e^z)^2} = -\frac{1}{1+e^z} + \frac{1}{(1+e^z)^2} = -A + A^2$$

Logistic regression with mirrorsigmoid activation should give similar results same as logistic regression with sigmoid activation. (Should get roughly same accuracy and loss with same learning rate and number of epochs for 2 activation functions.) If f(z) is sigmoid and g(z) is mirrorsigmoid, then g(z)=f(-z). It is not difficult to show that the optimal W and b for mirrorsigmoid are the negatives ((-1) times) of optimal W and b for sigmoid.

Exercises Section 5.1

Problem 5.1.1 (Theory) Suppose that a 4 layer neural network is set up. Assume 5 features and 4 units in layer 1, 3 units in layer 2, 2 units in layer 3 and 1 unit in layer 4. What are the dimensions of $W^{[k]}$ and $b^{[k]}$ for each of the layers for k=1,2,3,4. What is the total number of entries for all $W^{[k]}$ and $b^{[k]}$ in all layers.

Solution: see next page

Problem 5.1.2 (Jupyter Notebook) Consider binary classification with the case of 2 features and 3 data points (m=3):

$X = \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$

Assume that layer 1 has 2 units and that layer 2 has 1 unit with parameter matrices

$W^{[1]} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \quad b^{[1]} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \quad W^{[2]} = \begin{bmatrix} -1 & 1 \end{bmatrix} \quad b^{[2]} = [-0.1]$

Assume activation functions $f^{[1]}(z) = \log(1 + e^z)$ and $f^{[2]}(z) = \frac{1}{1+e^{-z}}$ and binary cross entropy loss function.

(a) Compute the value of the loss function for the above $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$.
(b) Perform 1 epoch of training using Gradient Descent with learning rate of 0.1 and recompute the loss function with the updated $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$.
(c) Compute the prediction based on input feature matrix X above after the 1 epoch
(d) Compute the accuracy of the prediction in (c) when compared against the actual Y specified above.

Hint: use IntroML/Examples/Chapter5/NeuralNetworkBinary.ipynb as a starting point

Solution: see IntroML/Exercises/Chapter5/Problem5.1.2.ipynb

Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Problem 5.1.1
Solution

Layer	Units In Previous Layer	Units in Current Layer	Dimensions of W	Dimensions of b	Total Entries in W and b
1	5	4	4x5	4x1	24
2	4	3	3x4	3x1	15
3	3	2	2x3	2x1	8
4	2	1	1x2	1x1	3

Total entries in all W and b is 50.

Exercises Section 5.2

Problem 5.2.1 (Theory) The following is the Exponential Linear Unit activation function:

$$f(z) = \begin{cases} z & if\ z \geq 0 \\ e^z - 1 & if\ z < 0 \end{cases}$$

Compute the derivate of f(z) and confirm that it is continuous at z=0. If A = f(Z), then express f'(Z) in terms of A.

Solution: see next page

Problem 5.2.1
Solution:

$$f'(z) = \begin{Bmatrix} 1 & if\ z \geq 0 \\ e^z & if\ z < 0 \end{Bmatrix}$$

Note that $f'(z) \to 1$ as $z \to 0$ from the positive side and that $f'(z) \to 1$ as $z \to 0$ from the negative side. Hence $f'(z)$ is continuous at z=0.

Note that $f(z) \geq 0$ if and only if $z \geq 0$ and that $f(z) < 0$ if and only if $z < 0$. (You can see this from the plots in Problem 2.2.2.) Hence, it follows

$$f'(z) = \begin{Bmatrix} 1 & if\ f(z) \geq 0 \\ e^z & if\ f(z) < 0 \end{Bmatrix}$$

If we define $A = f(z)$, then:

$$f'(z) = \begin{Bmatrix} 1 & if\ A \geq 0 \\ A+1 & if\ A < 0 \end{Bmatrix}$$

Exercises Section 5.3

Problem 5.3.1 (Programming) Add the Exponential Linear Unit (ELU) activation function to the course framework. Specifically, add ELU and derivative functions to the functions_activation file. Hint: have a look at Problem 2.2.2 for a hint at how to do this efficiently in numpy without looping. Make sure there is appropriate limit on Z so there is not an overflow for the exponential term.

Solution: See IntroML/Exercises/Chapter5/functions_activation.py

Problem 5.3.2 (Programming) Add a test case like that in test_NeuralNetwork_binary method in unittest_forwardbackprop so where one of the first 2 layers uses the ELU activation. Confirm that the unit test passes.

Solution: See IntroML/Exercises/Chapter5/unitest_forwardbackprop_elu.py

Problem 5.3.3 (Programming/Theory) Modify the test_NeuralNetwork_binary method in unittest_forwardbackprop so that one of the first 2 layers uses the RELU activation. Why doesn't the unit test pass in this case?

Solution: See next page for explanation

See IntroML/Exercises/Chapter5/unittest_forwardbackprop_relu.py

Problem 5.3.4 (Research) Investigate how well the driver_neuralnetwork_binary performs for other cases, such as "cubic", "disk", and "ring". Investigate how the number of units, number of layers, learning rate, and number of epochs affects the accuracy.

Solution: leave for student

Hint: run these solution files in a copy of folder Code/Version2.1. Overwrite the existing functions_activation.py with the one that you create.

Problem 5.3.3:

Solution: In general, test does not pass because RELU is not differentiable when its argument is 0. If values in Z are away from 0, then test may pass.

Exercises Section 5.4

Problem 5.4.1 (Jupyter Notebook) Compute softmax(Z) for:

$$Z = \begin{bmatrix} 1 & -1 & -1 \\ -2 & 2 & 1 \end{bmatrix}$$

Solution: see IntroML/Exercises/Chapter5/Problem5.4.1.ipynb

Problem 5.4.2 (Jupyter Notebook and Theory) The algorithm for computing softmax(Z) involves subtracting the maximum of Z in each column from the entries of that column before taking the exponential. Consider the alternative approach of subtracting the maximum of Z for the entire matrix from the entries of Z before taking the exponential? Hint: see what happens when this alternative approach is applied to the following matrix.

$$Z = \begin{bmatrix} 1000 & -1000 & 0 \\ -500 & 500 & 250 \end{bmatrix}$$

Solution: see next page

Problem 5.4.2
Solution: See IntroML/Exercises/Chapter5/Problem5.4.2.ipynb

Suppose that the maximum of Z for the entire matrix is subtracted from Z before taking the exponential. For the example matrix, $Z_{max} = 1000$ and

$$Z - Z_{max} = \begin{bmatrix} 0 & -2000 & -1000 \\ -1500 & -500 & -750 \end{bmatrix}$$

Taking the exponential of each entry of $Z - Z_{max}$ we get:

$$e^{Z-Z_{max}} = \begin{bmatrix} e^{0} & e^{-2000} & e^{-1000} \\ e^{-1500} & e^{-500} & e^{-750} \end{bmatrix}$$

When **implemented on a computer**, we will get (you can confirm this by printing $e^{Z-Z_{max}}$ in the Jupyter notebook):

$$e^{Z-Z_{max}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & e^{-500} & 0 \end{bmatrix}$$

When we sum each column and divide the column by that sum, there is division by 0 in the final column.

If the largest Z in each column is subtracted from that column, then there can never be division by 0, as each column of $Z - Z_{colmax}$ has an entry that is 0, and each column of $e^{Z-Z_{colmax}}$ has an entry that is 1. For the above example:

$$Z - Z_{colmax} = \begin{bmatrix} 0 & -1500 & -250 \\ -1500 & 0 & 0 \end{bmatrix}$$

$$e^{Z-Z_{colmax}} = \begin{bmatrix} e^{0} & e^{-1500} & e^{-250} \\ e^{-1500} & e^{0} & e^{0} \end{bmatrix} \approx \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The sum of each column is at least 1 when implemented on a computer, so there can never by division by zero issue.

Exercises Section 5.5

Problem 5.5.1 (Theory) Assuming 5 classes, compute the one-hot matrix version of [0 2 1 3 0 2].

Problem 5.5.2 (Theory) Compute the one-hot inverse of

$$Z = \begin{bmatrix} 0.1 & 0.15 & 0.40 \\ 0.2 & 0.25 & 0.50 \\ 0.4 & 0.35 & 0.05 \\ 0.3 & 0.25 & 0.05 \end{bmatrix}$$

Solutions: see next page

Problem 5.5.1 Solution:

$$Z = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Problem 5.5.2 Solution:

The one-hot inverse is a row vector, whose entries are the row indices of the largest entries of the corresponding column. One-hot inverse = [2 2 1]

Exercises Section 5.6

Problem 5.6.1 (Theory) Suppose that a 4-layer neural network is set up for multiclass classification with 4 classes. Assume 5 features and 1 unit in layer 1, 2 units in layer 2, 3 units in layer 3 and 4 unit in layer 4. What are the dimensions of $W^{[k]}$ and $b^{[k]}$ for each of the layers for k=1,2,3,4. What is the total number of entries for all $W^{[k]}$ and $b^{[k]}$ in all layers.

Solution: See next page

Problem 5.6.2 (Jupyter Notebook) Consider multiclass classification (3 classes) with the case of 2 features and 3 data points (m=3):

$$X = \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & 1 & 2 \end{bmatrix}$$

Assume that layer 1 has 2 units and that layer 2 has 1 unit with parameter matrices

$$W^{[1]} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \quad b^{[1]} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \quad W^{[2]} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \\ -2 & 1 \end{bmatrix} \quad b^{[2]} = \begin{bmatrix} -0.1 \\ -0.1 \\ -0.1 \end{bmatrix}$$

Assume activation functions $f^{[1]}(z) = \log(1 + e^z)$ and $f^{[2]}(z) = softmax(z)$ and cross entropy loss function.

(a) Compute the value of the loss function for the above $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$.
(b) Perform 1 epoch of training using Gradient Descent with learning rate of 0.1 and recompute the loss function with the updated $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$.
(c) Compute the prediction based on input feature matrix X above after the 1 epoch
(d) Compute the accuracy of the prediction in (c) when compared against the actual Y specified above.

Solution: see IntroML/Chapter5/Problem5.6.2.ipynb

Problem 5.6.1: Solution

Layer	Units In Previous Layer	Units in Current Layer	Dimensions of W	Dimensions of b	Total Entries in W and b
1	5	1	1x5	1x1	6
2	1	2	2x1	2x1	4
3	2	3	3x2	3x1	9
4	3	4	4x3	4x1	16

Total number of entries in all W and b is 35.

Exercises Section 5.7

Problem 5.7.1 (Research) Investigate how well the driver_neuralnetwork_multiclass.py performs for other cases, such as “cubic”, “disk”, and “band”. Investigate how the number of units, number of layers, learning rate, and number of epochs affects the accuracy.

Solution: leave for students.

Exercises Section 6.1

Problem 6.1.1: (Jupyter Notebook) The Jupyter notebook IntroML/Examples/Chapter6/StochasticGradientDescent.ipynb performs stochastic gradient descent using the following split of X and Y:

$$X = \begin{bmatrix} 1 & 2 \\ -2 & -5 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$X_{sample=0} = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \quad Y_{sample=0} = \begin{bmatrix} 0 \end{bmatrix} \quad X_{sample=1} = \begin{bmatrix} 2 \\ -5 \end{bmatrix} \quad Y_{sample=1} = \begin{bmatrix} 1 \end{bmatrix}$$

Redo the calculation by changing order of the data samples:

$$X_{sample=0} = \begin{bmatrix} 2 \\ -5 \end{bmatrix} \quad Y_{sample=0} = \begin{bmatrix} 1 \end{bmatrix} \quad X_{sample=1} = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \quad Y_{sample=1} = \begin{bmatrix} 0 \end{bmatrix}$$

Show that the order of the samples does indeed affect the final values of W and b after 1 epoch.

Solution: see IntroML/Exercises/Chapter6/Problem6.1.1.ipynb

Problem 6.1.2: (Jupyter Notebook) The Jupyter notebook IntroML/Examples/Chapter6/MiniBatchGradientDescent.ipynb performs mini-batch gradient descent using the following split of X and Y:

$$X = \begin{bmatrix} 1 & 2 & 4 & 1 & -1 \\ -2 & -5 & -8 & -2 & 2 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$$X_{minibatch=0} = \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} \quad Y_{minibatch=0} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$

$$X_{minibatch=1} = \begin{bmatrix} 1 & -1 \\ -2 & 2 \end{bmatrix} \quad Y_{minibatch=1} = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

Redo the 1 epoch of mini-batch gradient descent by using minibatch=1 first, then minibatch=0 second. Show that the order of the mini-batches does indeed affect the final values of W and b after 1 epoch.

Solution: see IntroML/Exercises/Chapter6/Problem6.1.2.ipynb

Problem 6.1.3: (Jupyter Notebook) Consider binary classification using Logistic Regression and the following training data

$$X = \begin{bmatrix} 3 & 1 & -1 \\ 1 & -2 & 2 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}$$

Assume that

$$W = \begin{bmatrix} W_0 & W_1 \end{bmatrix} = \begin{bmatrix} 2 & -1 \end{bmatrix} \quad b = -1$$

This problem was considered in Problem 4.7.1 where the gradients $\nabla_W L$ and $\nabla_b L$ were computed. Suppose that the data samples are re-ordered:

$$XX = \begin{bmatrix} 1 & -1 & 3 \\ -2 & 2 & 1 \end{bmatrix} \quad YY = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}$$

Perform back propagation to show that the gradients $\nabla_W L$ and $\nabla_b L$ are not changed by the order of the samples.

Solution: see IntroML/Exercises/Chapter6/Problem6.1.3.ipynb

The following 2 problems are for the mathematically inclined. Skip if you like.

Problem 6.1.4 (Theory) Consider problem the Logistic Regression problem in 6.1.3. Suppose X is the feature matrix, Y is the value vector and W and b are the parameter matrices. Prove that the order of the samples in X and Y does not have an impact on values $\nabla_W L$ and $\nabla_b L$ when back propagation is performed. Hint: use permutation matrices.

Solution: see next page

Problem 6.1.5 (Theory) Generalize the proof of Problem 6.1.4 to a neural network with an arbitrary number of layers.

Solution: see next page

Problem 6.1.4

Solution: Intuitively, we expect this to be true. Why should the order of the samples in back propagation used in batch gradient descent affect the values of the gradients $\nabla_W L$ and $\nabla_b L$? Consider

$$X = \begin{bmatrix} 3 & 1 & -1 \\ 1 & -2 & 2 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \quad XX = \begin{bmatrix} 1 & -1 & 3 \\ -2 & 2 & 1 \end{bmatrix} \quad YY = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}$$

Define the permutation matrix:

$$P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

The columns of P are permutations of the columns of the 3x3 identity matrix. (In general, columns of a nxn permutation matrix are permuted columns of nxn identity.) Multiplying PX, where X is a matrix, permutes the rows of X. Multiplying XP, where X is a matrix, permutes the columns of X. For example:

$$XP = \begin{bmatrix} 3 & 1 & -1 \\ 1 & -2 & 2 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 3 \\ -2 & 2 & 1 \end{bmatrix} = XX$$

Similarly, it can be shown that $YP = YY$

We are going to use the following properties of matrices and permutation matrices:

(a) The inverse of a permutation matrix is its transpose: $P^T P = I$ and $PP^T = I$, where I is the identity (Confirm this yourself for the above example.)
(b) $(AB)^T = B^T A^T$
(c) For the computation of $\nabla_b L = \sum_j \nabla_Z L_j = \nabla_Z LM$, where M is the column vector of ones of length j.
(d) If P is a permutation matrix and M is the column vector of ones as in (c), then PM is still a column vector of ones, so PM=M

The following table gives a "rough" proof. I have not specified every step in detail. The key point is that the original and permuted versions are related by the permutation matrix P in steps (1) – (6). When we finally calculate $\nabla_W L$ and $\nabla_b L$ in (7) and (8) below, we use properties (a) – (d) above, to show that the original and permuted cases yield the same gradients.

Phase	Original	Permuted	Notes
(1)Data	X, Y	XX = XP, YY = YP	We will use the notation XX, YY, ZZ, AA for the permuted versions of the original matrices
(2)Computation of Z	Z = WX+b	ZZ = WXX+b = WXP+b=(WX+b)P = ZP	The entries of ZZ are simply permuted entries of Z using same permutation matrix. Note same b is added for each

			entry of ZZ so permuting bP is the same as b
(3)Computation of A	A = f(Z)	AA = f(ZZ)=f(Z)P	Entries of AA are permuted entries of A using same permutation matrix
(4)Computation of $\nabla_A L$	$\nabla_A L$	$\nabla_{AA} L = \nabla_A L P$	Entries of AA and YY are permuted entries of A and Y, respectively. It can be shown, that $\nabla_{AA} L$ is permuted version of $\nabla_A L$ using the same permutation matrix.
(5)Computation of $\begin{bmatrix} \frac{\partial A_0}{\partial Z_0} & \cdots & \frac{\partial A_{m-1}}{\partial Z_{m-1}} \end{bmatrix}$	$\begin{bmatrix} \frac{\partial A_0}{\partial Z_0} & \cdots & \frac{\partial A_{m-1}}{\partial Z_{m-1}} \end{bmatrix}$	$\begin{bmatrix} \frac{\partial AA_0}{\partial ZZ_0} & \cdots & \frac{\partial AA_{m-1}}{\partial ZZ_{m-1}} \end{bmatrix} = \begin{bmatrix} \frac{\partial A_0}{\partial Z_0} & \cdots & \frac{\partial A_{m-1}}{\partial Z_{m-1}} \end{bmatrix} P$	Original and permuted are related by permutation matrix P
(6)Computation of $\nabla_Z L$	$\nabla_Z L$	$\nabla_{ZZ} L = \nabla_Z L P$	Entries of $\nabla_{ZZ} L$ are permuted entries of $\nabla_Z L$ using ideas in (4) and (5)
(7)Computation of $\nabla_W L$	$\nabla_W L = \nabla_Z L X^T$	$\nabla_W L = \nabla_{ZZ} L (XX)^T = \nabla_Z L P (XP)^T = \nabla_Z L P P^T X^T = \nabla_Z L I X^T = \nabla_Z L X^T$	Key point here is that we use property (a) for $PP^T = I$ and property (b) for $(XP)^T = P^T X^T$
(8)Computation of $\nabla_b L$	$\nabla_b L = \nabla_Z L M$	$\nabla_b L = \nabla_{ZZ} L M = \nabla_{ZZ} L P M = \nabla_Z L M$	Note that summing the entries of $\nabla_Z L$ and $\nabla_{ZZ} L$ in the column direction gives the same result. Summing permuted entries is same as summing original entries. See properties (c) and (d)

Problem 6.1.5

Solution: I am not going to go through all the details. The ideas from the previous proof for Logistic Regression can also be applied to a general neural network. Formally, one can use mathematical induction. The one step I would like to go over is the transition step from layer k to k-1. Recall that

$$\nabla_{A^{[k-1]}} L = W^{[k]T} \nabla L_{Z^{[k]}}$$

The following table compares original and permuted versions of this step.

Phase	Original	Permuted	Notes

Computation of $\nabla_{A^{[k-1]}} L$	$\nabla_{A^{[k-1]}} L = W^{[k]T} \nabla L_{Z^{[k]}}$	$\nabla_{AA^{[k-1]}} L = W^{[k]T} \nabla L_{ZZ^{[k]}} = W^{[k]T} \nabla L_{Z^{[k]}} P = \nabla_{A^{[k-1]}} LP$	Entries of $\nabla_{AA^{[k-1]}} L$ are permuted entries of $\nabla_{A^{[k-1]}} L$. Here we use the fact that $\nabla_{ZZ^{[k]}} L = \nabla_{Z^{[k]}} LP$

With the above step, one can go through the steps to show that $\nabla_{ZZ^{[k-1]}} L = \nabla_{Z^{[k-1]}} LP$ and ultimately that $\nabla_{W^{[k-1]}} L$ and $\nabla_{b^{[k-1]}} L$ do not depend on the order of data samples.

Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Exercises Section 6.2

Problem 6.2.1: (Jupyter Notebook) The following is Jupyter notebook showing how to use Gradient Descent with a for loop to minimize $2W_0^2 + W_1^2$. It is taken from the file: IntroML/Examples/Chapter3/Optimization.ipynb. Create analogous Jupyter notebook examples for the Momentum, RmsProp and Adam optimizers. Minimize the function with [2,2] as the initial guess point using 30 epochs.

In [1]:
```
# import numpy and matplotlib
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# define loss and gradient functions
def loss(W):
    return 2*W[0]**2 + W[1]**2

def grad(W):
    return np.array([4*W[0],2*W[1]])
```

In [3]:
```
# initialization
W = np.array([2,2])
alpha = 0.1
nepoch = 30

# iteration
loss_history = []
for epoch in range(nepoch):
    gradW = grad(W)
    W = W - alpha*gradW
    loss_history.append(loss(W))
print("After {} epochs".format(nepoch))
print("W: {}".format(W))
print("Loss: {}".format(loss_history[-1]))

plt.figure()
epoch_list = list(range(1,nepoch+1))
plt.plot(epoch_list,loss_history)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
```

Solution: see IntroML/Exercises/Chapter6/Problem6.2.1.ipynb

Problem 6.2.2 (Jupyter Notebook) The Adagrad method is defined as

$v_{epoch=i} = v_{epoch=i-1} + \nabla_W L^2_{epoch=i-1} \qquad v_{epoch=0} = 0$

$Update_{epoch=i} = -\alpha \frac{\nabla_W L_{epoch=i-1}}{\sqrt{v_{epoch=i} + \epsilon}}$

There are 2 parameters: α and ϵ. Create a Jupyter notebook example for Adagrad like those for Momentum, RmsProp, and Adam as discussed in Problem 6.2.1 to minimize the function $2W_0^2 + W_1^2$.

Solution: see IntroML/Exercises/Chapter6/Problem6.2.2.ipynb

Exercises Section 6.3

Problem 6.3.1 (Programming Exercise) Add the Adagrad optimizer class (see Problem 6.2.2) to the course framework. (Add to the Optimizers.py file.) Use for the multiclass classification problem. (Use the IntroML/Code/Version3.1/driver_neuralnetwork_multiclass.py driver.) Set $\epsilon = 10^{-8}$ and pick a learning rate of 0.1 and compare with default settings for the other optimization methods.

Adagrad defined by:

$$v_{epoch=i} = v_{epoch=i-1} + \nabla_W L^2_{epoch=i-1} \qquad v_{epoch=0} = 0$$
$$Update_{epoch=i} = -\alpha \frac{\nabla_W L_{epoch=i-1}}{\sqrt{v_{epoch=i} + \epsilon}}$$

Solution: see IntroML/Exercises/Chapter6/Optimizers.py, IntroML/Exercises/Chapter6/driver_neuralnetwork_multiclass_adagrad.py

Hint: create and run your solution programs in a copy of Code/Version3.1.

Exercises Section 6.5

Problem 6.5.1 (Theory) Suppose that training has been performed and the following confusion matrix is produced

	Actual 0	Actual 1
Predicted 0	107	13
Predicted 1	9	71

Compute the accuracy, precision, recall, and f1 score.

Problem 6.5.2 (Theory) True Negative Rate (TNR) and Negative Predictive Value (NPV) are defined as

$$TNR = \frac{\#TrueNegative}{\#ActualNegative}$$

$$NPV = \frac{\#TrueNegative}{\#PredictedNegative}$$

These measures are more meaningful than accuracy when there are few "negative" cases in a dataset. Compute TNR and NPV for the example from Problem 6.5.1

See solutions on next page

Problem 6.5.1
Solution:

	Actual 0	Actual 1
Predicted 0	107	13
Predicted 1	9	71

Let us define 0 as negative and 1 as positive. We have:
#TrueNegative = 107
#FalseNegative = 13
#ActualNegative = 116
#PredictedNegative = 120
#TruePositive = 71
#FalsePositive = 9
#ActualPositive = 84
#PredictedPositive=80
#Total = 200

$$Accuracy = \frac{\#TrueNegative+\#TruePositive}{\#Total} = \frac{178}{200} = 0.89$$

$$Precision = \frac{\#TruePositive}{\#PredictedPositive} = \frac{71}{80} = 0.8875$$

$$Recall = \frac{\#TruePositive}{\#ActualPositive} = \frac{71}{84} = 0.8452$$

$$F1score = 2\frac{Precision*Recall}{Precision+Recall} = 2\frac{0.8875*0.8452}{0.8875+0.8452} = 0.8658$$

Problem 6.5.2

Solution:

$$TNR = \frac{\#TrueNegative}{\#ActualNegative} = \frac{107}{116} = 0.9224$$

$$NPV = \frac{\#TrueNegative}{\#PredictedNegative} = \frac{107}{120} = 0.8917$$

Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Exercises Section 6.6

Problem 6.6.1 (Programming) Create a K-fold Cross Validation version of the driver_neuralnetwork_multiclass.py analogous to driver_neuralnetwork_binary_kfold.py.

Solution: See IntroML/Exercises/Chapter6/driver_neuralnetwork_muilticlass_kfold.py

Problem 6.6.2 (Programming) Add calculation of TNR and NPV (defined in Problem 6.5.2) to the file IntroML/Code/Version3.2/metrics.py and calculate these quantities for the binary classification problem in driver_neuralnetwork_binary.py. Here are the definitions:

$$TNR = \frac{\#TrueNegative}{\#ActualNegative}$$

$$NPV = \frac{\#TrueNegative}{\#PredictedNegative}$$

Solution: see IntroML/Exercises/Chapter6/metrics.py,
IntroML/Exercises/Chapter6/driver_neuralnetwork_binary_metrics.py

Hint: create and run your solution programs in a copy of Code/Version3.2

Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Exercises Section 6.9

Problem 6.9.1 (Programming) Create a hyperparameter search for multiclass classification for the 3 class "quadratic" classification case (analogous to what is done for binary classification in driver_neuralnetwork_binary_search.py). Use momentum optimization and search over the learning rate, batch size, and lambda.

Solution: see IntroML/Exercises/Chapter6/driver_neuralnetwork_multiclass_search.py

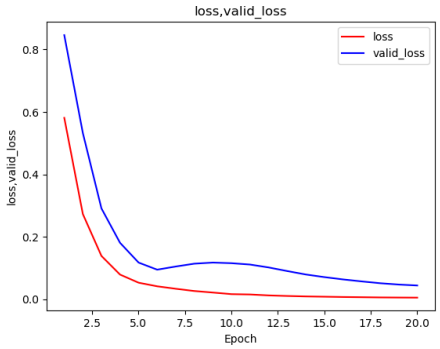
Hint: create and run your solution programs in a copy of Code/Version3.3

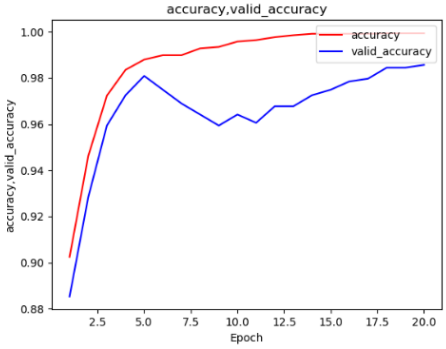
Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Exercises Section 6.10

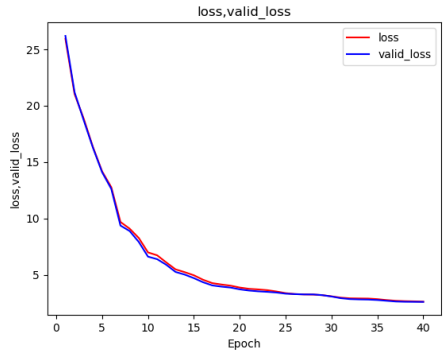
Problem 6.10.1 (Theory) Consider the following plots of loss and accuracy versus epoch for machine learning system. Here (loss = training loss, valid_loss = validation loss) and (accuracy = training accuracy, valid_accuracy = validation accuracy). In each case, is there underfitting or overfitting or both or neither? Provide a justification for your answer.

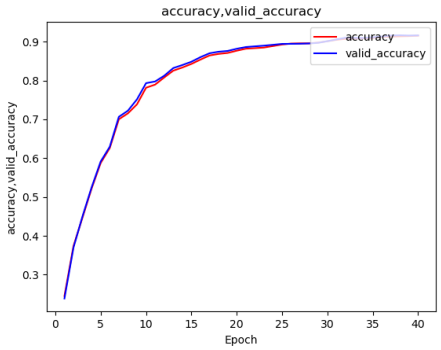
(a)



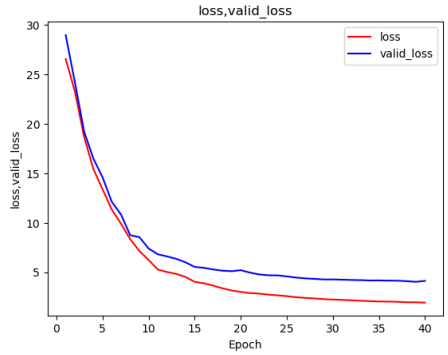


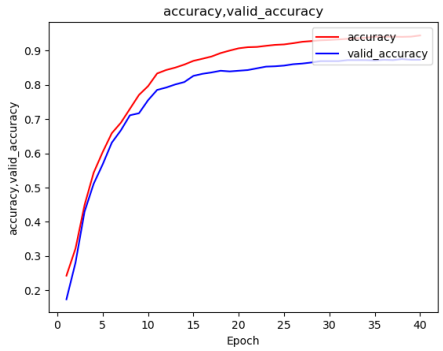
(b)





(c)





Solutions on next page

Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Solutions:
Problem 6.10.1(a)

Question 1: is there underfitting? Underfitting is not likely. If you look at the accuracy plot, the training accuracy is close to 1, so it is probably close to a target accuracy.

Question 2: Is there overfitting? Yes, there appears to be overfitting. As the validation loss is greater than the training loss and the validation accuracy is less than the training accuracy.

Answer: Overfitting

Problem 6.10.1(b)

Question 1: is there underfitting? This depends on the target accuracy. In the accuracy plot, the accuracy levels off at about 92%. If the target accuracy is higher, then there is an underfitting. If the target accuracy is around 92%, then there is not an underfitting.

Question 2: Is there overfitting? No, training loss and validation loss are roughly the same and training accuracy and validation accuracy are roughly the same.

Answer: Underfitting or Neither

Problem 6.10.1(c)

Question 1: is there underfitting? This depends on the target accuracy. In the accuracy plot, the accuracy levels off at about 93-94%. If the target accuracy is higher, then there is an underfitting. If the target accuracy is around 94%, then there is not an underfitting.

Question 2: Is there overfitting? Yes, there appears to be overfitting. As the validation loss is greater than the training loss and the validation accuracy is less than the training accuracy.

Answer: Underfitting and Overfitting or just Overfitting.

Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Exercises Section 7.1

Problem 7.1.1 (Programming Exercise) Nothing prevents us from using a neural network for a regression problem. Create a neural network to predict house prices. Hints: (A) use 2 layers (16 units and 1 unit) with “relu” activation for both.

Solution: see IntroML/Exercises/Chapter7/driver_casestudy_house_nn.py

Hint: create and run your solution programs in a copy of Code/Version4.1

Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Exercises Section 7.2

Problem 7.2.1 (Programming Exercise) In the lectures we saw that the approach used in IntroML/Code/Version4.1/driver_casestudy_spam.py led to an overfitting. A remedy for overfitting is to use a simpler neural network. Create a new driver for the spam case study using a Logistic Regression model. See if it does indeed provide remedy for overfitting.

Solution: see IntroML/Exercises/Chapter7/driver_casestudy_spam_logistic.py

Hint: create and run your solution programs in a copy of Code/Version4.1

Exercises Section 7.3

Problem 7.3.1 (Programming Exercise) Update driver_casestudy_mnist.py so that it performs a hyperparameter search. You choose the parameters to search over. Stay with 6000 training points and 1000 validation points so that the calculation time is reasonable.

Solution: see IntroML/Exercises/Chapter7/driver_casestudy_mnist_search.py

Hint: create and run your solution programs in a copy of Code/Version4.1

Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python
Solution files located at Github site: https://github.com/satishchandrareddy/IntroML

Exercises Section 8.1

Problem 8.1.1 (Programming Exercise) Create a Tensorflow version of the driver_casestudy_house.py driver. Hints: (A) use a single Dense layer with 1 unit and "linear" activation, (B) use the "mean_squared_error" loss function and for the metrics input in compile use "mean_absolute_error", (C) use the SGD optimizer with learning_rate=0.5.

Solution: see IntroML/Exercises/Chapter8/driver_casestudy_house_tensorflow.py

Problem 8.1.2 (Programming Exercise) Create a Tensorflow version of the driver_casestudy_spam.py driver. Hints: (A) use same neural network structure as in driver_casestudy_spam.py, (B) use the "binary_crossentropy" loss function and for the metrics input in compile use "accuracy", (C) use the same optimizer as in driver_casestudy_mnist.py.

Solution: see IntroML/Exercises/Chapter8/driver_casestudy_spam_tensorflow.py

Hint: create and run your solution programs in a copy of Code/Version5.1. Use driver_casestudy_mnist_tensorflow.py as a guide.