Neural network projects with python pdf

This is a code repository for neural network projects with Python, published by Packt. The ultimate guide to using Python to explore the true power of neural networks through six projects What is this book about? Neural networks are at the heart of AI's latest advances, providing some of the best solutions to many real-world problems, including image recognition, medical diagnosis, text analysis and more. This book runs through some basic neural network and deep learning concepts as well as some popular libraries in Python to implement them. This book covers the following interesting features: Learn the different architectures of neural networks and its achievements in AI Master deep learning at Python by creating and learning the neural network of Master Neural Networks for Regression and Classification Discover a entitent neural network for image recognition Learn to analyze the moods on text data using long-term short-term memory If you feel this book is for you, get a copy today! Instructions and navigation All code is organized in folders. For example, Chapter 02. The code will look like this: def detect_faces (img, draw_box grayscale_img'True): COLOR_BGR2GRAY) Below is what you need for this book: This book is perfect for data scientists, machine learning engineers, and deep learning enthusiasts who want to create practical neural network projects at Python. Readers should already have some basic knowledge of machine learning and neural networks. With the following software and hardware list, you can run all the code files present in the book (Chapter 1-7). Software and Hardware List Chapter Software requires os required 1-7 Python, Jupyter Laptop Windows, Mac OS X, and Linux (Anyone) We also provide a PDF file that has color image screenshots/diagrams used in this book. Click here to download it. Related Products Get To Know author James Loy has more than five years of expertise in data science in finance and health. He worked with the largest bank in Singapore to drive innovation and increase customer loyalty through predictive analytics. He also has experience in the health sector, where he has applied data analytics to improve decision-making in hospitals. He holds a master's degree in computer science from Georgia Tech, specializing in machine learning. His research interest includes deep learning and applied machine learning, as well as the development of AI computer agents for automation in the industry. He writes on K Data Science, a popular machine learning site with more than 3 million views per month. Suggestions and reviews Click here if you have any reviews or suggestions. Page 2 Watch 18 Star 130 Fork 113 You Can't Perform This Action in It's time. You've signed up with another tab or window. Reboot to update the session. You subscribe to another tab or window. Reboot to update the session. We use the use of third-party analytical cookies to understand how you use GitHub.com so we can create the best products. Learn more. We use additional third-party analytical cookies to understand how you use GitHub.com so we can create the best products. You can always update your choices by clicking on Cookie Preferences at the bottom of the page. For more information, see us that we use important cookies to perform the main functions of a website, such as logging in. Find out more Always Active We use analytical cookies to understand how you use our websites so we can make them better, for example, they are used to gather information about the pages you visit and how many clicks you need to accomplish the task. Read more last update September 15, 2020, Keras is a powerful and easy-to-use free open source Python library to develop and evaluate deep learning models. It wraps up the effective theano and TensorFlow numerical computing libraries and allows you to identify and train neural network models in just a few lines of code. In this tutorial, you'll learn how to create your first model of a deep neural network in Python with Keras. Start your project with my new book Deep Learning with Python, including step-by-step tutorials and Python source code files for all examples. Start. February/2017 Update: An updated example of forecasting so rounding works in Python 2 and 3. Mar/2017 Update: An updated example for the latest versions of Keras and TensorFlow. Mar/2018 Update: An alternative link has been added to download the dataset. July/2019 Update: More useful resources have been expanded and added. Sep/2019 Update: Updated for Keras v2.2.5 API. October/2019 update: Updated for Keras v2.3.0 API and TensorFlow v2.0.0. Update Aug/2020: Updated for Keras v2.4.3 and TensorFlow v2.3. Develop your first neural network in Python with Phil Whitehouse's Keras Step-By-StepPhoto, some rights reserved. Keras Tutorial Review There is not much code required, but we are going to step over it slowly, so you will know how to create your own models in the future. The steps you are going to cover in this tutorial are: Download the data. Identify the Keras model. Composite Keras Model. Fit Keras Model. Check out the Keras model. Tie it all together. Make predictions This Keras tutorial has several requirements: You have a Python 2 or 3 installed and configured. You have SciPy (including NumPy) installed and configured. You have Keras and the backend (Theano or TensorFlow) installed and configured. If you need help with the environment, see tutorial: How to customize the Python environment for deep learning Create a new file called keras_first_network.py and enter or copy and paste code into the file, you're coming. Take my free 2-week email course and discover MLPs, CNNs and LSTMs (with the code). Click to sign up now and get a free version of the PDF Ebook course. Start a free mini-course right now! 1. Download data The first step is to identify the features and classes that we intend to use in this We will use the NumPy library to download our dataset, and we will use two classes from the Keras library to determine our model. The import required is below. The first neural network with keras tutorial of numpy import loadtxt from keras.models import Consistent from keras.layers import Dense ... In this Keras tutorial, we're going to use Pima Indians to start a diabetes dataset. This is a standard machine learning dataset from the UCI machine learning repository. It describes patient records for Pima Indians and whether they had developed diabetes within five years. Thus, it is a problem of binary classification (the onset of diabetes as 1 or not as 0). All input variables described by each patient are numerical. This simplifies the use directly with neural networks that expect numerical values of input and output, and is ideal for our first neural network in Keras. The dataset is available here: Dataset CSV File (pima-indians-diabetes.csv) Details of the dataset Download the dataset and place it in your local work catalog, in the same place as your python file. Save it with the name of the file: pima-Indians-diabetes.csv pima-Indians-diabetes.csv Look inside the file, you should see the strings of data, as follows: 6,148,72,35,0,33,6,627,50,1 1,85,66,29,0,26,6,0.351,31,0 8,183,64,0 23,0 3.0.672.32.1 1,89,66,23,94,28.1,167,21,0,137,40,35,168,43.1,2,288,33,1 ... 6 148 72.35,0,33.6,0.627,50,11,85,66,29,0 26,6,0.351,31,08,183,64,0,23,3,0,67,67 32,11,89,66,23,94,28.1,0.167,21,00,137,40,35,168,43.1,288,33,1 We can now download the file as a number matrix using the NPmPy download function. There are eight input variables and one output variable (last column). We'll study the model for the X series map to the output variable (y), which we often summarize as y q f(X). Variables can be summarized as follows: Input variables (X): The number of times pregnant plasma glucose concentration 2 hours in the oral glucose tolerance test Diastolic blood pressure (mm Hg) Triceps skin thickness (mm) 2-hour insulin serum (mu U/ml) Body mass index (weight per kg/ (height in m) we can divide columns of data into variables. The data will be stored in a 2D array, where the first dimension is the strings, and the second dimension is columns, such as rows, columns. We can then select the output column (9th variable) through index 8. download the dataset into the dataset - loadtxt ('pima-indians-diabetes.csv', delimiter',') and divide into input (X) and weekends X variables - dataset:0:8 u dataset:,8 ... dataset - loadtxt ('pima-indians-diabetes.csv', delimiter',') is divided into input (X) and weekend (y) variables We are now ready to define our neural network model. Note that the dataset has 9 columns, and the 0:8 range will select columns from 0 to 7, stopping in front of index 8. If this is new to you, then you can learn more about the array of slicing and ranges in this post: How to index, slice and modify NumPy Arrays for machine learning in Python 2. Identify Keras Model models in Keras defined as a sequence of layers. We create successive models and add layers one at a time until we are happy with our network architecture. The first thing to do right is to make sure that the input layer has the right amount of input. This can be stated when you create the first layer with the input_dim argument and set it up to 8 for 8 input variables. How do I know the number of layers and their types? This is a very difficult question. There are guristics that we can use, and often the best network structure is through the process of trial and error experiments (I'll explain more about it here). Typically, you need a network big enough to capture the structure of the problem. In this example, we'll use a fully connected network structure with three layers. Fully connected layers are defined by the Dense class. We can specify the number of neurons or nodes in the layer as the first argument, and specify the activation function using the activation argument. We'll use the corrected linear block activation function, called ReLU on the first two layers, and the Sigmoid function in the output layer. It used to be that Sigmoid and Tanh activation features were preferable to all layers. These days, the best performance is achieved with the ReLU activation feature. We use a sigmoid on the output layer to ensure our network output between 0 and 1 and easily compare either the probability of a Class 1 or a snap to a tight classification of any class with a default threshold of 0.5. We can put it all together by adding each layer: The model expects a string of data with 8 variables (argument input_dim'8) The first hidden layer has 12 nodes and uses the relu activation function. The second hidden layer has 8 knots and uses the relu activation function. The output layer has one node and uses the function of sigmoid activation. ... - to define the model model keras - successive () model.add (Dense (12, input_dim'8, activation'relu')) model.add (Dense (8, activation))) model.add (Dense (1, activation'sigmoid') ... model.add (Dense (12, input_dim'8, activation'relu'))model.add (Dense(8, activation))model.add (Dense (1, activation'sigmoid')) Note, the most confusing thing here is that the form of model input is defined as an argument on the first hidden layer. This means that the line of code that adds the first layer of Dense does two things, defining the input or visible layer and hidden layer. 3. Make a keras model Now that the model is defined, we can compose it. The model uses efficient numerical libraries under the lids (the so-called backend), such as Theano or TensorFlow. The backend automatically chooses the best way to present the network for training and forecasting to work on your hardware, such as a processor or GPU, or even distributed. When compiling, we must specify some of the additional properties required when learning the network. Keep in mind that learning the network means finding the best set of weights for the input map for the conclusions in our dataset. We must specify the loss function to use to evaluate the weight gain, the optimizer used to search through different weights for the network and any additional metrics we would like to collect and report during training. In this case, we will use cross entropy as a loss argument. This loss is for binary classification problems and is defined in Keras as binary_crossentropy. You can learn more about the choice of loss functions based on your problem here: How to choose loss functions when learning deep neural network learning We will define the optimizer as an effective algorithm of stochistic gradient descent adam. This is a popular version of the gradient descent because it automatically adjusts itself and gives good results in a wide range of problems. To learn more about Adam's version of the stochistic gradient descent, see the post: A gentle introduction to Adam's algorithm for deep learning Finally, because it's a classification problem, we'll collect and report on the accuracy of the classification determined by argument metrics. compilation of the keras model model.compile binary_crossentropy.binary_crossentropy ..... Fit Keras Model We defined our model and made it ready for efficient computing. Now it's time to run a model on some data. We can train or place our model on our downloaded data by calling the function fit () on the model. Training takes place in different eras, and each era is divided into parties. Epoch: One runs through all the lines in the training data set. Package: One or more samples reviewed by the model during the era before the weights were updated. One era consists of one or more parties based on the chosen size of the party and the model is suitable for many eras. For more information on the difference between eras and parties, see the post: What is the difference between a package and an era in a neural network? The learning process will take place for a fixed number of iterations through a data set called the epochs that we must specify using the argument of epochs. We must also set the number of dataset lines that are considered before updating the scale of the model during each era, called batch size and set using batch_size argument. To solve this problem will work for a small number of eras (150) and use Small batch size 10. These configurations can be selected experimentally by trial and error. We want to train the model enough to recognize a good (or good enough) display of input lines in the output classification. The model will always have some errors, but the number of errors will be leveled after some point for a given model configuration. This is called the convergence of models. - fits the keras model on the dataset model.fit (X, y, epochs-150, batch_size'10) batch_size ... You don't need a GPU for this example, but if you're interested in how to cheaply run large models on GPU hardware in the cloud, see this post: How to set up Amazon AWS EC2 graphics processors to train Keras Deep Learning Models 5. Evaluate the Keras Model We have trained our neural network throughout the data set and can evaluate network performance on one data set. This will only give us an idea of how well we have washed away the data set (such as train accuracy), but don't know how well the algorithm can work on the new data. We did this for simplicity, but ideally you could split your data into data sets to train and evaluate your model. You can evaluate your model in the training dataset using the evaluation function on the model and pass it on to the same input and output that are used to train the model. This will create a forecast for each pair of input and output data and collect points, including average losses and any customized metrics such as accuracy. The evaluation function returns a list with two values. First, the loss of the model in the dataset, and the second will be the accuracy of the model on the data set. We are only interested in reporting on accuracy, so we will ignore the loss of value. Keras model score, accuracy and model.assessment (X, y) printing ('Precision: .2f' % (accuracy 100)) - keras model_ score, accuracy and model.assessment (X, y)print ('Precision: .2f' % (accuracy 100)) 6. Tie It All Together You have just seen how you can easily create your first neural network model in Keras. Let's link all this to a complete example of code. The first neural network with keras-textbook from numpy import loadtxt from keras.models imports sequential from keras.layers import Dense and downloads a data set - loadtxt ('pima-indians-diabetes.csv', delimiter',') - divided into input (X) and weekend (y) variables X - dataset:,0:8 u dataset:,8 to define the model of keras - sequential () model.add (Dense (12, input_dim'8, activation'relu') model.add (Dense(8, activation'relu')) model.add (Dense(1, activation'sigmoid')) - a compilation of the model keras model model.compile.compile (loss'binary_crossentropy' ,adam', metrics 'accuracy') - fit models keras on the model of the dataset.fit (X, y, epochs'150, batch_size'10) - evaluate the model of keras, accuracy and model. The first neural network with keras tutorialfrom numpy import loadtxtfrom keras.models imports sequential from keras.layers import Densedataset - loadtxt ('pima-indians-diabetes.csv', delimiter',') input_dim'8, activation 'relu'))model.add (Dense (8, activation 'relu'))model.add (Dense (1, activation'sigmoid')))) compilation keras modelmodel.compile (loss 'binary_crossentropy', optimizer ''adam', metrics accuracy) fit models keras on datasetmodel.fit y, epochs'150, batch_size'10) estimate keras model_, accuracy and model.assess (X, y))Print ('Accuracy: .2f' % (accuracy 100)) you can copy the entire code into the Python file and save it as keras_first_network.py in the same catalog as the pima-indians-diabetes.csv. You can then run the Python file as a script from the command line (team query) as follows: python keras_first_network.py python keras_first_network.py Running this example, you should see the message for each of the 150 eras of print loss and accuracy, and then the final assessment of the trained model on the training data set. It takes about 10 seconds to perform on my workstation running on the CPU. Ideally, we would like the losses to go to zero and the accuracy to 1.0 (e.g. 100%). 768/768 [==============================] - 0s 63us/step - loss: 0.4817 - acc: 0.7708 Epoch 147/150 768/768 [==============================] - 0s 63us/step - loss: 0.4764 - acc: 0.7747 Epoch 148/150 768/768 [==============================] - 0s 63us/step - loss: 0.4737 - acc: 0.7682 Epoch 149/150 768/768 [==============================] - 0s 64us/step - loss: 0.4730 - acc: 0.7747 Epoch 150/150 768/768 [==============================] - 0s 63us/step - loss: 0.4754 - acc: 0.7799 768/768 [==============================] - 0s 38us/step Accuracy: 76.56 768/768 [==============================] - 0s 63us/step - loss: 0.4817 - acc: 0.7708768/768 [==============================] - 0s 63us/step - loss: 0.4764 - acc: 0.7747768/768 [==============================] - 0s 63us/step - loss: 0.4737 - acc: 0.7682768/768 [==============================] - 0s 64us/step - loss: 0.4730 - acc: 0.7747768/768 [==============================] - 0s 63us/step - loss: 0.4754 - acc: 0.7799768/768 [==============================] - 0s 38us/step Note, if you try running this example in an IPython or Jupyter notebook you may get an error. The reason is the output of the progress bars during training. You can easily turn them off by installing wordy features in a fit call () and evaluate (the) features, such as: ... verbose-0) - appreciate the model keras, precision and model.assess (X, y, verbose-0) model_ batch_size ... Consider running the example several times and compare the average result. What grade did you get? Post the results in the comments below. Neural networks are a stochastic algorithm, meaning that the same algorithm on the same data can train a different model with different skills every time the code is running. This is a function, not a bug. You can learn more about this in the post: Embracing randomness in machine learning the difference in model performance means that to get a reasonable approximation of how well your model performs, you may need to match it many times and calculate the average accuracy score. For more information on this approach to the evaluation of neural networks, See the post: How to evaluate Deep Learning Models For example, below are the accuracy of the score from the re-run example 5 times: Precision: 75.00 Accuracy: 77.73 Precision: 77.60 Precision: 78.12 Precision: 76.17 We see that all accuracy estimates are about 77% and average 76.924%. 7. Make predictions number one question I get asked: After I train my model, how can I use it to make predictions for new data? That's a good question. We can adapt the above example and use it to generate predictions on the training data set, pretending to be a new data set that we haven't seen before. Making predictions is as easy as calling the forecasting function on a model. We use the sigmoid activation function on the output layer, so the predictions will be likely in the range between 0 and 1. We can easily convert them into a clear binary forecast for this classification task by rounding them up. For example:... We can trigger the function of predict_classes () on the model to predict clear classes directly, model.predict_classes for example: model.predict_classes ... The first neural network with keras makes predictions from numpy import loadtxt from keras.models import sequential keras.layers import Dense - download dataset - loadtxt ('pima-indians-diabetes.csv', delimiter',') and weekend (y) variables X - dataset:0:8 u dataset:,8 to define the keras model - Serial() model.add (Dense(12, input_dim'8, activation'relu')) model.add (Dense (8, activation'relu')) model.add (Dense(1, activation 'sigmoid')) - a compilation of the model keras model.compile (loss'binary_crossentropy', optimizer','adam', metrics'"accuracy'y, epochs'150, batch_size'10, verbose-0) - make classy predictions with model predictions - model.predict_classes (X) - summarize the first 5 cases for i in the range (5) : print (%s zgt;%d (expected %d)' % (X'i'.tolist),), predictions, y'i)) - the first neural network with keras makes predictions from numpy import loadtxtfrom keras.models import sequentialfrom keras.layers import Densedataset - loadtxt ('pima-indians-diabetes.' delimiter','y) variablesmodel.add (Dense (12, input_dim'8, activation 'relu'))model.add (Dense (8, activation'relu')) model.add (Dense (1, activation 'sigmoid'))binary_crossentropy)) y, epochs'150, batch_size'10, verbatim0) make classy predictions with the model of presstitution - model.predict_classes (X) summarize the first 5 cases of printing ('%s' q'gt%d (expected % d)' % (X'i'.tolist ,), predictions,i'i'i')) Running the example does not show progress as before as we have a verbose argument to 0. Once the model is appropriate, predictions are made for all the examples in the dataset, and the input lines and predicted class value for the first 5 examples are printed and compared to the expected class value. We see that most lines are correctly predicted. In fact, we expect that about 76.9% of the lines will be correctly predicted based on our estimated model performance in the previous section. [6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 0 (expected 1) [1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] => 0 (expected 0) [8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (expected 1) [1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] => 0 (expected 0) [0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] => 1 (expected 1) [6.0 , 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 0 (expected 1)[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] => 0 (expected 0)[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (expected 1)[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] => 0 (expected 0)[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] => 1 (expected 1) If you would like to know more about how to make predictions with Keras models , see post: How to make predictions with Keras Keras Training Summary In this post, you discovered how to create your first model of the neural network using the powerful Keras Python library for deep learning. Specifically, you've learned six key steps in using Keras to create a neural network or deep learning model, step by step including: How to download data. to identify the neural network in Keras. How to build a Keras model with an effective numerical backend. How to train a model with data. How to evaluate the model by data. How to make predictions with the model. Do you have any questions about Keras or about this tutorial? Ask your question in the comments and I will do my best to answer. Keras Tutorial Extensions Well done, you have successfully developed your first neural network using Keras's deep learning library at Python. This section presents some extensions to this tutorial that you can explore. Set up a model. Change the configuration of the model or learning process and see if you can improve the performance of the model, for example, achieve more than 76% accuracy. Save the model. Update the tutorial to save the file model and then download it later and use it to make predictions (see this tutorial). Let's generalize the model. Update the tutorial to summarize the model and create a plot of layer models (see this tutorial). Separate sets of train and test data. Divide the downloaded dataset into a train and a test set (divided into strings) and use one set to train the model and another set to measure the performance of the model on the new data. The curves of the learning plot. The fit function returns a story object that summarizes loss and accuracy at the end of each era. Create linear plots of this data called learning curves (see this tutorial). Explore the new data set. Updating the textbook to use another tablicular dataset, possibly from the UCI machine learning repository. Use a functional API. Update the tutorial on using the functional Keras API to determine the model (see this tutorial). Further reading Are you looking for some more in-depth tutorials with Python and Keras? Take a look at some of them: Related Textbooks Books Deep Learning (textbook), 2016. Deep learning with Python (my book). API Keras Deep Learning Library HomePage Keras API Documentation How Did You Go? Do you have questions about deep learning? Post your questions in the comments below and I'll do my best to help. ... with just a few lines Of Python Discover, as in my new book Ebook: Deep Learning with Python It covers end-of-the-end projects on topics such as: Multilayer Perceptrons, Convolutional Networks and Periodic Neural Networks, and more... Finally bring deep learning into your own projects to miss scientists. Just the results. See what's inside Jason Brownlee, Ph.D. is a machine learning specialist who teaches developers how to get results with modern machine learning techniques through practical tutorials. Tutorials.

conversor_para_epub_online.pdf