Practical Machine Learning Dr. Ashish Tendulkar Department of Computer Science and Engineering Indian Institute of Technology, Madras

Lecture - 31 Text Generation with RNNs

(Refer Slide Time: 00:22)



Now, We will use RNNs to build models for time series data. We will build two models one for prediction of the univariate time series and second for predictions of multivariate time series. So, let us begin by importing necessary packages.

(Refer Slide Time: 00:51)



Here we are going to use a weather time series dataset which is recorded by Max Planck Institute for Biogeochemistry. The dataset contains 14 different features such as air temperature, atmospheric pressure and humidity. These were collected every ten minutes beginning in 2003, for efficiency we will use only the data collected between 2009 and 2016.

This data set is prepared by Francois Chollet, we will download the dataset. Dataset is in CSV file.

(Refer Slide Time: 01:39)



Let us look at top five entries of the dataset. See you can see that the observations are recorded every 10 minutes for a single hour we will have 6 observations. And in a day we will have 144 observations. Given a specific time, let us say if you want to predict temperature for 6 hours in the future; in order to make this prediction we choose to use 5 days of observation. Thus, you need to create a window containing last 720 observations to train the model.

Many such configurations are possible making this dataset a good dataset for experimentation. So, if we take last 5 days of data; in a day there are 144 observations. So, we take 720 observations in the history and from this 720 observations; we will try to make next 36 observations because we want to predict temperature for 6 hours in the future and in every hour there are 6 observations. So, there are in all 36 observations that we want to predict.

(Refer Slide Time: 03:03)



The univariate_data() function returns the windows of time for which the model is to be trained. It takes the parameter history_size which is the size of the past window of information. Target_size is how far in the future does the model need to learn to predict; the target size is the label that needs to be predicted.

So, we define start and end index of the data and we perform some kind of a selection in the history to get the data and the corresponding labels. For this exercise we will use first 300000 rows of the data for training and remaining ones for validation. So, this amounts to about 2100 days worth of training data.

(Refer Slide Time: 04:21)



So, let us define the training split. We will set a seed to ensure that the results are reproducible.

(Refer Slide Time: 04:30)

				-
Text 🗅 Copy to Drive	✓ RAM E Disk E	• /	Editing	
seed to ensure reproducibility.				
tf.random.set_seed(13)				
1. Forward a university data data and a				
T: Forecast a univariate time series				
ou will train a model using only a single feature (temperature), and use it to make predictions for that value in the future.				
st extract only the temperature from the dataset.				
		↑ ↓ ∞ \$	18.1	1
ni_data of['II (degC)]] ni_data.index (of['Data Time'] ni_data.head()				
ate Time				
1.01.2009 00:10:00 -8.02				
1.01.2009 00:30:00 -8.51				
1.01.2009 00:40:00 -8.31				
1.01.2009 00:50:00 *8.27				
	Text & Copy to Drive seed to ensure reproducibility. f(random.set_seed(13)) f(random.set_seed(13))	Text & Copy to Drive & Disk & Door of the Desk & De	Text ▲ Copy to Drive MAL → ✓ Ball → ✓ Ball → ✓ Ball → ✓ Ball → ✓ Øast → ✓ ✓ Øast → ✓ ✓ Øast → ✓ Øast → ✓ Øast ✓ ✓ Øast Øast ✓ ✓ Øast ✓ ✓ Øast Øast ✓ ✓ Øast Øast ✓ ✓ Øast Øast	Text ▲ Copy to Drive

Let us begin with the first part that deals with forecasting a univariate time series. Here we will use a single variable called temperature and we will use it to make predictions for the temperature values in the future. We first extract the temperature information from the dataset.

(Refer Slide Time: 05:09)

🍞 Text dassification with movie re- X 😀 time series spyth - Coluboratory X 🕇		- 0	×
🗧 👌 😋 🔹 colab.research.google.com/github/tensorflow/docs/blob//2.0rc/site/en/r2/tutorials/tent/time_series/pynb#scrolTfo=ht0EnwAqM2SU	1	k 0	÷
+ Code + Text d Copy to Drive	V RAM Disk	ting	×
> [1] 91.01.2009 00:10:00 - 0.51 01.01.2009 00:10:00 - 0.31 0 01.01.2009 00:10:00 - 0.27 Name: T (degC), dtype: float64			
Let's observe how this data looks across time.	↑ ↓ ∞ ¢	• •	
every((categotetik.ses_useplots.sesSobplot object at 0:07H96473556)), dyperoject)			
		VG 030	6 PM

(Refer Slide Time: 05:13)

ode + Text 🙆 Copy to Drive			
	Disk	/ Editing	1
(1) 			
[10] uni_data = uni_fata.values			
It is important to normalize features before training a neural network. A common way to do so is by subtracting the mean and divic each feature.	ling by the standard deviation of		
Note: The mean and standard deviation should only be computed using the training data.			
uni_train_mean + uni_drain[TRAIN_SPIT].exec() uni_train_std - uni_drain[TRAIN_SPIT].exe()	<u>↑</u> 4	, ∞ ‡ ∎	L
Let's normalize the data.			_

Let us look at the temperature data. This is temperature data as recorded on different dates. We will normalize the features before training a neural network, we will use a z-score normalization technique for normalization, for that we require to compute mean under standard deviation.

(Refer Slide Time: 05:39)



And here we normalize the data by subtracting mean from every data point and dividing the resultant by the standard deviation. Now that we have normalized the data, we will create the data for univariate model. The model will be given last 20 recorded temperatures and it needs to predict the temperature at the next time. So, we use univariate_data() function that we define for extracting the training and the target data.

So, we apply the same function to obtain training and validation datasets.

(Refer Slide Time: 06:34)



So, let us look at what this particular function returns.

(Refer Slide Time: 06:39)



(Refer Slide Time: 06:42)

C ecolab.research.google.com/github/tensorflow/docs/blob/r2.0rc/site/en/r2/tutorials/text/time_series.ipyn	b#scrollTo=feDd95XFdzSH 🛠 🔕 🗄
Code + Text & Copy to Drive	Cisk E Calting V
(-1.95334967) (-1.9533467) (-2.9533467) (-2.9533462) (-2.9533462) (-2.973778) (-2.9737535) (-2.9735453) (-2.9735453) (-2.9735453) (-2.9735453) (-2.9735453) (-2.9735453) (-2.9735453) (-2.9736453) (-2.97664545454545454545454545454545454545454	in blue, and it must predict the value at the
<pre>[] def create_time_step(length): time_steps = [] def is regret_ingth 0, 1): time_steps(i) return time_steps</pre>	

So, this is a single window of past history having 20 values and you also get the target temperature to predict.

(Refer Slide Time: 06:51)



(Refer Slide Time: 07:01)



Now, that the data has been created; let us look at a single example. The information given to the network is in blue; so this is the information that is provided to the network and you want to predict the point marked by the red cross and this particular cross is the actual value.

(Refer Slide Time: 07:19)

RAM C
Disk Eating
cts the next point to be

So, before proceeding to training a model; it is always a good idea to establish some kind of baseline. Here, the simple baseline could be looking at a historical data and predicting next point as the average of last 20 observations.

So, let us look at how this baseline performs; we simply np dot mean on the history.

(Refer Slide Time: 07:50)



So, you can see that the model prediction on the base model is quite off from the two feature. Now, we will try to see if we can do anything better than the base line.

(Refer Slide Time: 08:09)



So, here we will train a recurrent neural network model which is suitable for handling sequential data or a time series data which is also an example of a sequence data. RNN process a time series step by step maintaining an internal state summarizing the information they have seen so far. Here we will use a LSTM models for modeling the time series data. Before we begin we prepare dataset objects for training and validation. We will first shuffle the object then batch it and we cache the dataset.

(Refer Slide Time: 09:04)



For every time step we have number of features and this is a batch up example and here in the third axis you have features.

(Refer Slide Time: 09:31)

0000 T	Text & Copy to Drive	V RAM III	•	/ Editing	2
	Time steps				
You will	see the LSTM requires the input shape of the data it is being given. This shape can be inferred from dataset created.				
[21] 51] 51	<pre>imple_lstm_model = tf.keres.models.Sequential([tf.keres.lsyers.STM(6, imput_shapeer_train_uni.shape(-2:)), tf.keres.lsyers.Dense(1) imple_lstm_model.compile(optimizer*ader', loss**###)</pre>				
Let's ma	ke a sample prediction, to check the output of the model.				
0 1	rrx,y.in.val.univariate.take(): print(simple_iter_medai.predict(s).shape)		↑ ↓ <	• • •	1
• (2	56, 1)				

Let us build a simple LSTM model which gives out 8 outputs and output of LSTM is passed through a dense layer with a single unit. Note that we are not using any activation here because we are trying to solve a regression type of problem. We use *Adam* as an optimizer and we are going to optimize the minimum absolute error as a loss.

(Refer Slide Time: 10:19)



Let us make a sample prediction and check the output of the model.

(Refer Slide Time: 10:35)

Code ·	+ Text & Copy to Drive	۰	·	/ Ed	ting	
0						
Let's data	train the model now. Due to the large size of the dataset, in the interest of saving time, each epoch will only run for 200 steps, instead of the complete training as normally done.					
		1	4 00	\$	8.1	
0	EVALUATION_INTERVAL = 200					
*	(POOS = 10					
	simple_lstm_model.fit(train_univariate, epochs=EPOCHS,					
	steps_per_epocnetvaluation_intexval, validation_datavval_univariate_validation_steps=50)					
	Train for 200 steps, validate for 50 steps					
	Train for 200 steps, validate for 50 steps fpoch 1/10					
	Train for 200 stops, validate for 50 stops fach 1/10 WMRNDB-tessorFlaw/Fmilly (function Function_initialize uninitialized variables.clocals).initialize variables at 0/7/7/7/7/864 WMRNDB-tessorFlaw/Fmilly (function Function_initialized variables.clocals).initialize variables at 0/7/7/7/7/864 WMRNDB-tessorFlaw/Fmilly (function Function_initialized variables.clocals).initialize variables at 0/7/7/7/7/864 WMRNDB-tessorFlaw/Fmilly (function Function_initialized variables.clocals).initialized variables at 0/7/7/7/7/864 WMRNDB-tessorFlaw/Fmilly (function Function_initialized variables.clocals).initialized variables at 0/7/7/7/7/864 MARNDB-tessorFlaw/Fmilly (function Function_initialized variables.clocals).initialized variables at 0/7/7/7/7/7/864 MARNDB-tessorFlaw/Fmilly (function Function_initialized variables.clocals).initialized variables at 0/7/7/7/7/7/7/7/7/7/7/7/7/7/7/7/7/7/7/7	a8> co	uld :	ot be	tran	nsf
	Train for 200 stops, validate for 50 stops foch 1/10 MANDEStensorFlow:Entity (function Function_initialize_uninitialize_variables.clocals>.initialize_variables at 0x7977073564 MANDEStensorFlow:Entity (function Function_initialize_uninitialize_variables.clocals>.initialize_variables at 0x7977073564 MANDEStensorFlow:Entity (function_initialize_uninitialize_variables.clocals>.initialize_variables at 0x7977073564 MANDEStensorFlow:Entity (function_initialize_variables.clocals>.initialize_variables at 0x7977073564 MANDEStensorFlow:Entity (function_initialize_variables.clocals>.initialize_variables at 0x7977073564 MANDEStensorFlow:Entity (function_initialize_variables.clocals>.initialize_variables>.initialize_variabl	a8> co not be	uld n	ot be isform	tran wed an	nsf
	Train for 200 steps, validate for 50 steps Epoch 1/10 WANDED: Entry function Function_initialize_uninitialized_variables.locals>.initialize_variables at 0x7777e73664 WANDED: Entry function Function_initialize_uninitialized_variables.clocals>.initialize_variables at 0x7777e73664ab> could n 200/200 [reservencesses] - 3g l4ms/step - loss: 0.6978 - val_oss: 0.6722 Epoch 2/10	a8> co not be	uld (tran	iot be isform	tran wed an	nsf nd
-	Train for 200 stops, validate for 50 stops foch 1/10 WARDID: EnsorFlow:Entty (function Function_initialize_uninitialized_variables.clocals>.initialize_variables at 0/7/77/0756680 could r 200/200 [===================================	a8> cc not be	uld (tran	not be isform	e tran Wed an	nsf nd
	Train for 100 steps, validate for 50 steps fpsch 1/10 MADDIDE introflow.Initialize_variables.clocals>.initialize_variables at 00/777e73664 MADDIDE intrity function /unitialize_uninitialize_variables.clocals>.initialize_variables at 00/77e73664ab could r 200/200 [spch 1/10 200/200 [secont_risk] 200/200 [secont_risk	a8> co not be	uld (tran	not be isform	tran wed an	nsf nd
	Train for 200 steps, validate for 50 steps foch 1/10 WANTOB:FrestorFlow:Entity (function Function, initialize_uninitialize_variables.clocals).initialize_variables at 0x77777975658 WANTOB: frestorFunction Function_initialize_uninitialized_variables.clocals).initialize_variables at 0x77779756680 could n 200/200 [researcessessessessesses] - 3gl4em/step - loss: 0.8098 - val_loss: 0.8732 fpoch 1/10 200/200 [researcessessessessesses] - 15 Sem/step - loss: 0.8075 - val_loss: 0.8035 fpoch 3/10	a8> co not be	uld n tran	not be isform	tran ed an	nsf
	Train for 200 stops, validate for 50 stops foch 1/20 WARDED: EnsorFlow:Enstry (function Function_initialize_uninitialize_uninitialize_uniables.clocals).initialize_uniables at 0/7/77/0756688 could # 200/200 [###################################	a8> cc not be	uld (trai	iot be isform	tran ed an	nsf
	Train for 200 steps, validate for 50 steps Eych //10 MANDND: Entity (function Function_initialize_uninitialized_variables.clocals>.initialize_variables at 00/77/073664 MANDND: function function_initialize_uninitialized_variables.clocals>.initialize_variables at 00/77/07366480 could r 200/20 [soch //10 200/20 [s	a8> co not be	uld i tra	not be isform	e tran ed an	nsf
	Train for 200 steps, validate for 50 steps foch 1/10 WANDDD: fratty (function Function, initialize_uninitialized_variables.clocals).initialize_variables at 0x77777075658 WANDDD: fratty (function Function_initialize_uninitialized_variables.clocals).initialize_variables at 0x777770756680 could r foch 2/10 200/200 [researcher] - 3g 14em/step - loss: 0.8095 + val_loss: 0.8075 fpoch 3/10 200/200 [researcher] - 15 5em/step - loss: 0.8075 - val_loss: 0.8075 fpoch 3/10 200/200 [researcher] - 15 5em/step - loss: 0.8445 - val_loss: 0.8279 [spoch 4/10 200/200 [researcher] - 15 5em/step - loss: 0.8417 - val_loss: 0.8255 [spoch 5/10	a8> co not be	uld i i tra	not be	e tran ed an	nsf
	Train for 100 stops, validate for 50 stops Spech 1710 MARDING Entry (Auntion Function_initialize_uninitialize_uninitialize_unriables.clocals>.initialize_unriables at 0x777473664 MARDING Entry (Auntion Function_initialize_uninitialized_unriables.clocals>.initialize_unriables at 0x777473664ab could r 200/200 [entry (Auntion Function_initialized_unriables.clocals>.initialize_unriables at 0x777477673664ab could r 200/200 [entry (Auntion Function_initialized_unriables.clocals>.initialize_unriables at 0x7774 50c/1710 200/200 [entry (Auntion Function]] - 15 5ms/stop - loss: 0.4047 - val_loss: 0.4235 50c/1710 200/200 [entry (Auntion Function]] - 15 5ms/stop - loss: 0.4239 - val_loss: 0.4243 50c/1710	a8> co not be	uld i tra	not be isform	i tran wed an	nsf
	Train for 200 steps, validate for 50 steps Spoch /10 MANDED: Entity (function function_initialize_uninitialized_wrimbles.clocals.initialize_urlables at 0x7777073664 MANDED: Entity (function function_initialize_uninitialized_wrimbles.clocals.initialize_urlables at 0x777707366480) could n 200/200 [reservencessed] - 3d 14ms/step - loss: 0.5098 - val_loss: 0.6722 Epoch /210 200/200 [reservencessed] - 1s 5ms/step - loss: 0.6075 - val_loss: 0.6279 [poch /210 200/200 [reservencessed] - 1s 5ms/step - loss: 0.6445 - val_loss: 0.6255 Spoch 5/10 200/200 [reservencessed] - 1s 5ms/step - loss: 0.6229 - val_loss: 0.6243 Spoch 6/10 200/200 [reservencessed] - 1s 5ms/step - loss: 0.6225 Spoch 5/10 200/200 [reservencessed] - 1s 5ms/step - loss: 0.6225 - val_loss: 0.6233	a8> co not be	uld : tran	not be	i tran wed an	nsf
	Train for 100 steps, validate for 50 steps Epsch 1/10 MANDING: Entity (function function_initialize_uninitialized_unriables.clocals.initialize_unriables at 00797707364aD could n MANDING: Entity (function_functialize_uninitialized_unriables.clocals.initialize_unriables at 00797707364aD could n 200/20 [rememension] 200/20 [re	a8> co not be	uld : tra	not be	i tran wed an	nsf
	Train for 200 steps, validate for 50 steps Spech //D MANDED: Entity (function Function_initialize_uninitialized_variables.clocals).initialize_variables at 00/77/073664 MANDED: Entity (function Function_initialize_uninitialized_variables.clocals).initialize_variables at 00/77/07366480 could r 200/20 [stresserBound for 200 steps] 200/20 [stresserBound for 200 steps] 2	a8> co not be	uld : tra	not be	i tran med an	nsf
	Train for 200 stops, validate for 50 stops Spech /120 WANTDD: Entity (function function_initialize_uninitialized_wariables.clocals.initialize_wariables at 0x7777075656 WANTDD: function function function_initialized_wariables.clocals.initialize_wariables at 0x7777075656 WANTDD: Entity (function function_initialized_wariables.clocals.initialize_wariables at 0x7777075656 200/200 [===================================	a8> co not be	uld i i tra	not be isfore	: tran med an	nst

So, let us send the model; now we will run the model only for 200 steps and we are going to have 10 epochs; in each epoch you only train a model for 200 steps.

(Refer Slide Time: 10:44)



Let us predict using simple LSTM model; you can see that in the first case the prediction is very close.

(Refer Slide Time: 10:53)



(Refer Slide Time: 10:55)



Second case it is rightly off. Third case it is quite close. So this looks better than the baseline. Now, that you see in the basics of how to train a RNN model for time series forecasting, we will move on to the next part where we will do time series forecasting for multivariate time series.

(Refer Slide Time: 11:20)

🔿 C 🔹 colab.research.google.com/github/tensorflow/docs/blob/r2.0rc/site/en/r2/tutorials/test/time_series.jpynb#scrolITo=S2rRLv8MttGL	i i	☆	0
Code + Text & Copy to Drive	V RAM III	/ Editing	~
This looks better than the baseline. Now that you have seen the basics, let's move on to part two, where you will work with a multivariate time se	ries.		
 Part 2: Forecast a multivariate time series 			
The original dataset contains fourteen features. For simplicity, this section considers only three of the original fourteen. The features used are at atmospheric pressure, and air density.	ir temperature,		
To use more features, add their names to this list.			
[] features_considered = ['p (mbar)', 'T (degC)', 'rho (g/m**3)']			
<pre>[] features = df[features_considered] features.index = df['Date Time'] features.ined()</pre>			
Let's have a look at how each of these features vary across time.			
[] features.plot(subplots=True)			
As mentioned, the first step will be to normalize the dataset using the mean and standard deviation of the training data.			_
[] detaset = festures.values deta_mean = detaset.mem(auisme) deta_std = detaset.std(auisme)			
[] dataset = (dataset-data mean)/data std			
		🖓 🄙 ENG (03.14

So, for the sake of this exercise; we will just select 3 of the 14 features this features are air temperature, atmospheric pressure and the air density.

(Refer Slide Time: 11:33)

Code +	+ Text	rive			Disk E
0	Date Time				
θ	01.01.2009 00:10:00	996.52	-8.02	1307.75	
	01.01.2009 00:20:00	996.57	-8.41	1309.80	
	01.01.2009 00:30:00	996.53	-8.51	1310.24	
	01.01.2009 00:40:00	998.51	-8.31	1309.19	
Let's h	01.01.2009 00:50:00 have a look at how each o features.plot(subplot	998.51 of these featu s=True)	-8.27 res vary acros	1309.00 ss time.	
Let's h	01.01.2009 00:50:00 have a look at how each o features.plot(subplot indiced, the first step will	998.51 of these featu s=True) Il be to norma	-8.27 res vary acro fize the datas	1309.00 ss time. set using the m	ided deviation of the training data.
Let's h	01.01.2009 00:50:00 have a look at how each o features.plot(subplot antioned, the first step will dataset = features.va deta_nean + dataset.m deta_ttd = dataset.st	996.51 of these features (s=True) If be to normatives two statis=0) d(axis=0)	-8.27 res vary acros	1309.00 es time.	iderd deviation of the training data.
Let's h	01.01.2009 00:50:00 have a look at how each o features.plot(subplot miloid, the first step wil dataset = features.va deta_mean = dataset.m dataset = (dataset.	996.51 of these featu is=True) Il be to norma lues ean(axis=0) d(axis=0) ta_mean)/dat	-8.27 res vary acro fize the datast ta_std	1309.00 es time.	dard deviation of the training data.
Let's h	01.01.2009 00:50:00 have a look at how each o features.plot(subplot entioled, the first step will dataset = features.va data_ntm = dataset.st dataset = (dataset.dat	996.51 of these featu is=True) Il be to norma lues ean(axis=0) d(axis=0) ta_mean)/dat	-8.27 res vary acro slize the datas	1309.00 es time.	ideal deviation of the training data.

Let us look at how each of these features vary across time.

(Refer Slide Time: 11:39)



So, you can see that this features are on different scale and they have different variability across time.

(Refer Slide Time: 11:50)

C colab.research.google.com/github/tensorflow/docs/blob/r2.0rc/site/en/r2/tutorials/text/time_series.jpynb#scrollTo=e3UeWDqplob	i		贲	6
ode + Text & Copy to Drive	V RAM III. Disk II.	. /	Editing	
C Data Time				
As mentioned, the first step will be to normalize the dataset using the mean and standard deviation of the training data.				
[30] dataset = features.values data_mean = dataset.mann(axise) data_std = dataset.std(axise)				
Ø dataset = (dataset-data_mean)/data_std	1	ψ co t	-	-
Single step model In a single step setup, the model learns to predict a single point in the future based on some history provided.				
Single step model In a single step setup, the model learns to predict a single point in the future based on some history provided. The below function performs the same windowing task as below, however, here it samples the past observation based on the step size given.				
Single step model In a single step setup, the model learns to predict a single point in the future based on some history provided. The below function performs the same windowing task as below, however, here it samples the past observation based on the step size given. [] def multivariate_data(dataset, target, start_index, end_index, history_size,				
<pre>Single step model In a single step setup, the model learns to predict a single point in the future based on some history provided. The below function performs the same windowing task as below, however, here it samples the past observation based on the step size given. [] def multivariate_data(detaset, target, start_index, ed_index, history_size,</pre>				
<pre>Single step model In a single step setup, the model learns to predict a single point in the future based on some history provided. The below function performs the same windowing task as below, however; there it samples the past observation based on the step size given. () def militvariste_data(dataset, target, start_index, end_index, history_size,</pre>				

Let us normalize the data using mean and standard deviation. Now, we will use a single step model, this model lost to predict a single point in the feature based on some history provided.

(Refer Slide Time: 12:06)



So, let us define the function for performing the window in task. Here we will sample the past observations based on the step size that is given to us.

(Refer Slide Time: 12:18)



So, in this exercise we will show last 5 days of data to the network; last 5 days of data will constitute 720 observations that are sampled every hour. The sampling is done every 1 hour; since drastic change is not expected within 60 minutes.

Thus, 120 observation represent history of the last 5 days. For the single step prediction model the label for a data point is the temperature 12 hours into the future. In order to create a label for this the temperature after 72 observations is used. Note that we are having 6 observations per hour; so in 12 hours we will have total 72 observations; that is why we use temperature which is after 72 observations in the future.

So, we use past history of 720 points and we want to predict the target in the future which is 72 observations away from the current point in the future.

(Refer Slide Time: 13:40)

Code	+ Text A Copy to Drive	RAM III		/ Editio	0
[33]	pett_history = 720 future_target = 22 STFP = 6 x_train_single, y_train_single = multivariate_data(dataset, dataset[;, 1], 0,	UNK E			
	<pre>TABLU_SPLT, part, history,</pre>				
Let's	s look at a single data-point.				
[34]	<pre>print ('Single window of past history : {}'.format(x_train_single[0].shape))</pre>				
θ	Single window of past history : (120, $\overline{\sharp})$				
0	<pre>train_dsta_single = tf.dsta_Dataset.from_tensor_slices((v_train_single, y_train_single)) train_dsta_single = train_dsta_single.cscb().shuffle(UMPER_SIII).shuf()[AIO(_SIII).repart() val_dsta_single = tf.dsta_Dataset.from_tensor_slices((v_val_single, y_val_single)) ul_dsta_single = vd_sta_single_statio(ENDOCSEIT).repart()</pre>		r 4 0	0.17 8	1
-	0 COOE 0 TEXT		_		-
[]	<pre>single_ste_model = ff.wess.model.Sequential() single_ste_model.asd(ff.wess.lspv:s.SIM(32,</pre>				
	<pre>single_step_model.compile(optimizerwtf.keras.optimizers.WSprop(), losse'mae')</pre>				

Let us create training and validation data sets; the shape of the train data of a single point. So, we have 120 points and each point has got 3 features. So, we construct data set object with from_tensor_slice and then we shuffle it, batch it and then cache the dataset; in case of validation we only batch the dataset.

(Refer Slide Time: 14:12)

-> C	colab.research.google.com/github/tersorflow/docs/blob/r2.0rc/site/en/r2/tutorials/text/time_senes.ipynb#scrollTo=eCWG4xgQ306E		Ŷ 🕓
Code -	+ Text do Copy to Drive	V RAM III V	/ Editing
0	val_data_single = tf.data.Dataset.from_tensor_slices((x_val_single, y_val_single)) val_data_single = val_data_single.batch(BATOr_SIII).repart()		
[]	<pre>single_tteg_model = tf.keres.models.Sequential() single_tteg_model_add(tf.keres.layer.STM(1),</pre>		
Let's	theck out a sample prediction.		
[]	<pre>for x, y in val_dsta_single.take(): print(single_stap_model.predict(x).shape)</pre>		
[]	<pre>single_step_history = single_step_model.fit(Train_deta_single, esconselPACOM,</pre>		
[]	<pre>def plct_train_bistory(history, tills): loss = history.history("loss") w_lloss = history.history("ul_loss")</pre>		
	<pre>epochs = range(len(loss))</pre>		
	plt.figure()		
	plr_plr(spork, loss, 'b', labels'Twising lass') plr.plr(spork, willows, 'b', labels'Valiation loss') plr.tlet(titla) plr.tleges()		
	elt.show()		

We build the same sequential model with LSTM layer and a denser layer; use RMS prop as an optimizer and mean absolute error as the loss() function.

(Refer Slide Time: 14:44)

validati validati	on_data=val_data_single, on_steps=50)			
a bee stars will done for the stars				
or zee steps, validate for se steps				
/10	al all and a second state of the second s			
: Entity (function Function, initialize un	initialize_uninitialized_va	riables. (locals), initially	<pre>J_variables at ex/T/Tee452378 </pre>	is could not be trans
[antity conclose resctioninitiatite_or	ns/sten - loss: 0.3207 -	val loss: 0.2689	3 85 6X/1/1464323/0/ 50020 10	it be transformed and
/10				
[] - 2s 11s	ns/step - loss: 0.2691 -	val_loss: 0.2552		
/10				
[] - 2s 10r	ns/step - loss: 0.2683 -	val_loss: 0.2477		
/10				
[] - 25 119	ns/step - loss: 0.2632 -	val_loss: 0.2476		
/10		and have a party		
(10	ns/step = 1055: 0.2343 =	val_1055: 0.2424		
[] . 2: 10	st/stan + loss: 0.2476 +	val loss: 0.2698		
/18	arresp - ressi erreste			
[******] - 2s 9ms	s/step - loss: 0.2475 - v	al_loss: 0.2654		
/10		-		
[] - 2s 9ms	s/step = loss: 0.2471 = v	al_loss: 0.2469		
/10	*			
[=====] - 25 9h	s/step - loss: 0.2527 - v	#1_loss: 0.2671		
3/10	/	1 1		
[] - 23 340	s/step * 1055: 0.2404 * v	#1_1055: 0.2400		
	10 10 10 10 10 10 10 10 10 10 10 10 10 1	10 10 10 10 10 10 10 10 10 10	14 15 16 17 18 18 19 19 19 19 19 19 19 19 19 19	10 10 10 10 10 10 10 10 10 10

Let us check out the sample predictions; let us train the model.

(Refer Slide Time: 15:18)



Let us look at how training progressed. So, the training loss is coming down; validation loss came down went up it is slightly a zigzag kind of pattern in the validation loss, not so smooth.

(Refer Slide Time: 15:39)



Now that the model is trained let us make a few sample predictions. The model is given the history of three features for the past 5 days sampled every hour which is 120 data points. Since the goal is to predict the temperature, we only display the past temperature in the plot and the prediction is made one day in the future.

(Refer Slide Time: 16:12)

Anda I Trud	A Consta Datus				, RAM III.			10	dalar.	
Code + Text	Copy to Drive	V			V Disk E			10	ating	
(**) L	ó ż	4	é	i						
Θ										
	- la companya da serie da s									
Predict a singl	e step future									
Now that the n	nodel is trained, let's mai	ke a few sampi	le predictions.	The model is g	given the history of three features over the past five days sampled every hour (120					
data-points), s	ince the goal is to predic	t the temperat	ure, the plot on	nly displays the	e past temperature. The prediction is made one day into the future (hence the gap					
between the h	story and prediction).									
						Υ `	↓ 00	¢		1
O for X,	y in val_data_single.	<pre>take(3): 11 numnu() 1</pre>	()ummun [0]v							
b106	<pre>snow_piot([x[0]];; single s</pre>	teo model.or	y[0].humpy(),							
			eares(v)[a]])	14,						
plot.	'Single S	itep Predictio	on')	, 14,						
plot.	'Single S show()	itep Predicti	on')	, 14,						
plot.	'Single's	itep Prediction	ction	, 14,						
plot.	'Single's show() Si	itep Prediction	ction	History	٦					1
plot.	'Single's	itep Prediction	ction	History True Future]					Ì
plot.	'Single's	itep Prediction	ction	History True Future Model Prediction						
plot.	Single S	itep Prediction	ction	History True Future Model Prediction						
plot.	show() 'Single's	Itep Prediction	ction	History True Future Model Prediction						
plot. 16 - 14 - 12 -	show() 'Single's	itep Prediction	on') ction	History True Future Model Prediction	2					
plot.	show() 'Single's	ngle Step Predicti	ction	History True Future Model Prediction						
plot.	show() 'Single's	ngle Step Predicti	ction	History True Future Model Prediction						
plot.	shov() 'Single S	ngle Step Predicti	ction	History The Future Model Prediction						
plot.	shov() single s	itep Predicti	ction	History The Future Model Hediction X						

So, you can see for first two examples; for first example they are very close.

(Refer Slide Time: 16:21)



Second example also are very close.

(Refer Slide Time: 16:22)



Third example they are very close. So, you can see that using more features; you are able to predict the temperature quite accurately.

(Refer Slide Time: 16:33)



In multi step model, instead of predicting just one value we will try to predict multiple values in the future.

So, you can see that for the multi step model the training data consists of recording over the past five days sampled every hour which will be 120 points. However, the model needs to learn to predict the temperature for the next 12 hours; since an observation is taken every 10 minutes, will have 72 predictions over next 12 hours.

So, our target has now 72 values. So to predict the 72 values from the last 120 values.



(Refer Slide Time: 17:50)

Let us create a data set, you can see that we have 120 points in the training set; each has 3 features and we want predict 72 different values in the future.

(Refer Slide Time: 18:06)



We create the data set. So, the smooth line here is a history and the dots here are kind of predicted values.

(Refer Slide Time: 18:17)

ode	+ Text	Copy	to Drive							✓ Disk I	•	/ Edi	ing
[47]	-1.4			V									
Θ		-120	-100	-60	-60	-40	-20	ò					
0	multi	other madel										 ~ 1	
•	multi multi multi multi	_step_model _step_model. _step_model. _step_model. _step_model.	<pre>= tf.keras.mc add(tf.keras. add(tf.keras. add(tf.keras. compile(optim</pre>	odels.Sequent layers.LSTN(layers.LSTN(layers.Dense mizer=tf.kera	ial() 32, return_sequer input_shapexx 16, activatic (72)) s.optimizers.	ces=True, _train_multi. n='relu')) RMSprop(clipy	shape[-2:])) alue=1.0), lp	;='mae')			1		
Lefs	multi multi multi s see how	_step_model _step_model. _step_model. _step_model. _step_model. w the model p	<pre>= tf.keras.mc add(tf.keras. add(tf.keras. add(tf.keras. compile(optin redicts before i</pre>	odels.Sequent layers.LSTM(layers.CSTM(layers.Oense mizer=tf.kera it trains.	ial() 32, return_sequer input_shape=> 16, activatic (72)) s.optimizers.	ces=True, '_train_multi. n='relu')) RMSprop(clipy	shape[-2:]))	5='m#")			1		
Lefs	multi multi multi s see how for x	_step_model _step_model. _step_model. _step_model. _step_model. w the model p I, y in val_d int (multi_st	<pre>= tf.keras.mc add(tf.keras. add(tf.keras. add(tf.keras. compile(optin redicts before i ata_multi.tak ep_model.pred</pre>	<pre>>>dels.Sequent Llayers.LSTM(.layers.LSTM(.layers.LSTM(.layers.Dense sizer=tf.kera it trains. ke(1): dict(x).shape</pre>	<pre>ial() 32, 12, input_shapexs 16, activatic (72)) s.optimizers.</pre>	ces=True, _train_multi. m='relu')) RMSprop(clip)	shape[-2:])) alue=1.0), lp	se'mae')					
Lefti []	multi multi multi s see how for x pri multi	_rtep_model _step_model. _step_model. _step_model. _step_model. w the model p l, y in val_d nt (multi_st _step_histor	<pre>= tf.keras.ms add(tf.keras. add(tf.keras. add(tf.keras. add(tf.keras. compile(optin redicts before i ata_multi.tak ep_model.pred y = multi_ste</pre>	dels.Sequent layers.LSTM(.layers.LSTM(layers.Dense mizerwf.kera mizerwf.kera mizerwf.kera mizerwf.kera mizerwf.kera mizerwf.kera mizerwf.kera mizerwf.kera	ial() 32, input_shapess input_shapess (5, activatic (72)) s.optimiters.)) train_data_ms stapp_per_epo validation_da validation_st	ces=True, _train_multi. ms'relu')) RMSprop(clipy lti, epochas cheEvaLuATIOn tawwal_dats_ eps=50)	shape[-2:])) alues1.0), lp POCHS, _DITEROAL, _DITEROAL, ulti,	t*'mae")					

Since this task is a bit more complicated than the previous task; we will use two LSTM layers. Finally, the 72 predictions are made using a dense layer that outputs 72 predictions.

We are using *RMSProp* as an optimizer and we are using *clipvalue* as argument for clipping the gradient. We use mean absolute error as a loss() function here; let us look at how the model predicts before it trains.

(Refer Slide Time: 18:57)



Let us train the model; we will train the model for 200 steps in every epoch.

(Refer Slide Time: 09:10)



Now, that the model is trained; let us look at how the model training preceded.

(Refer Slide Time: 19:18)

	noino-okiqosjiqsiro 🕱 🔮 i
iode + Text 💩 Copy to Drive	✓ RAM Disk III ✓ ✓ Editing ✓
coo - coo -	
4000 -	
2000 -	
Predict a multi-step future	
Let's now have a look at how well your network has learnt to predict the future.	
for x, y in val_dsta_multi_take()): 	
cell has not been executed in this session	
(oil has not been executed in this session Next steps	
(of has not been executed in this session) Next steps This tutorial was a quick introduction to time series forecasting using an RNN. You may now try to predict the stock marke	and become a billionaire.

Let us predict how well the module has learn to predict the future.

(Refer Slide Time: 19:26)



So, you can see that the future; the true values in the future are in blue whereas, the model prediction is in red. You can see that in the first case it is able to predict the next values quite better.

(Refer Slide Time: 19:49)



Also in the second case and third case there are some deviations, but model seems to be doing a decent job of predictions. You can see that the model is quite smooth it is not able to get into some of the nitty-gritty details like that there are some abrupt turns; model is not able to learn that perfectly, but overall it is doing a decent job. So, that was it from modeling time series with RNNs.