West Valley CIST-005B, Advanced Python

Personal Project

Basic Keras Neuronal Network to identify Palindromes

Theme: Data Science
Author: Bertrand B. Blanc

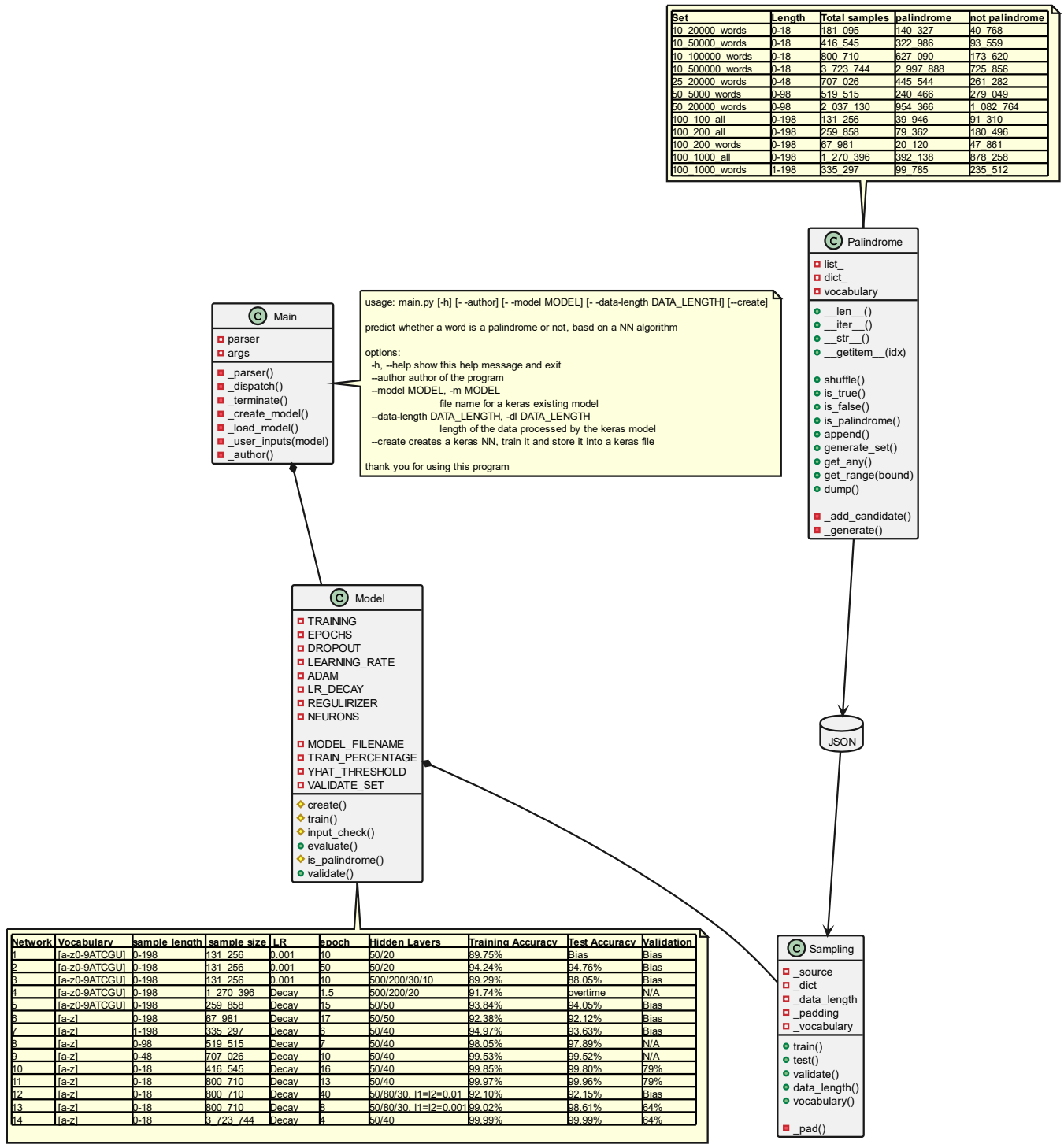
My investigative project aims at exploring the theme of data science with a neuronal network using supervised learning. The Python library I used is Keras from Tensorflow. I attempted to train a model to predict palindromes.

I generated a set of samples to train and test the model.

Then created a neuronal network, and trained it.

According to the results, I modified the sampled data, the network architecture or the meta-parameters to obtain better results. I operated iteratively that way until I obtained satisfactory results.

Set	Length	Total samples	palindrome	not palindrome
10_20000_words	0-18	181_095	140_327	40_768
10_50000_words	0-18	416_545	322_986	93_559
10_100000_words	0-18	800_710	627_090	173_620
10_500000_words	0-18	3_723_744	2_997_888	725_856
25_20000_words	0-48	707_026	445_544	261_282
50_5000_words	0-98	519_515	240_466	279_049
50_20000_words	0-98	2_037_130	954_366	1_082_764
100_100_all	0-198	131_256	39_946	91_310
100_200_all	0-198	259_858	79_362	180_496
100_200_words	0-198	67_981	20_120	47_861
100_1000_all	0-198	1_270_396	392_138	878_258
100_1000_words	1-198	335_297	99_785	235_512



Network	Vocabulary	sample length	sample size	LR	epoch	Hidden Layers	Training Accuracy	Test Accuracy	Validation
1	[a-z0-9ATCGU]	0-198	131_256	0.001	10	50/20	89.75%	Bias	Bias
2	[a-z0-9ATCGU]	0-198	131_256	0.001	50	50/20	94.24%	94.76%	Bias
3	[a-z0-9ATCGU]	0-198	131_256	0.001	10	500/200/30/10	89.29%	88.05%	Bias
4	[a-z0-9ATCGU]	0-198	1_270_396	Decay	1.5	500/200/20	91.74%	overtime	N/A
5	[a-z0-9ATCGU]	0-198	259_858	Decay	15	50/50	93.84%	94.05%	Bias
6	[a-z]	0-198	67_981	Decay	17	50/50	92.38%	92.12%	Bias
7	[a-z]	1-198	335_297	Decay	6	50/40	94.97%	93.63%	Bias
8	[a-z]	0-98	519_515	Decay	7	50/40	98.05%	97.89%	N/A
9	[a-z]	0-48	707_026	Decay	10	50/40	99.53%	99.52%	N/A
10	[a-z]	0-18	416_545	Decay	16	50/40	99.85%	99.80%	79%
11	[a-z]	0-18	800_710	Decay	13	50/40	99.97%	99.96%	79%
12	[a-z]	0-18	800_710	Decay	40	50/80/30, l1=l2=0.01	92.10%	92.15%	Bias
13	[a-z]	0-18	800_710	Decay	8	50/80/30, l1=l2=0.001	99.02%	98.61%	64%
14	[a-z]	0-18	3_723_744	Decay	4	50/40	99.99%	99.99%	64%

Palindrome samples for training

Different sets of samples in separate JSON files. The samples are composed of strings of different length, up to a maximum length. Strings represent both palindromes and not palindromes.

These different sets were created to experiment different hypothesis.

Set	Length	Total samples	palindrome	not palindrome
10_20000_words	0-18	181_095	140_327	40_768
10_50000_words	0-18	416_545	322_986	93_559
10_100000_words	0-18	800_710	627_090	173_620
10_500000_words	0-18	3_723_744	2_997_888	725_856
25_20000_words	0-48	707_026	445_544	261_282
50_5000_words	0-98	519_515	240_466	279_049
50_20000_words	0-98	2_037_130	954_366	1_082_764
100_100_all	0-198	131_256	39_946	91_310
100_200_all	0-198	259_858	79_362	180_496
100_200_words	0-198	67_981	20_120	47_861
100_1000_all	0-198	1_270_396	392_138	878_258
100_1000_words	1-198	335_297	99_785	235_512

The vocabulary for the *_words sets is composed of:

- the lowercase Latin alphabet letters [a-z] to represent strings of characters.

The vocabulary for the *_all sets is composed of

- the lowercase Latin alphabet letters [a-z] to represent strings
- the digits [0-9] to represent numbers
- the 4 letters A,T,C,G to represent DNA sequences
- the 4 letters A,U,C,G to represent RNA sequences

The samples are unique (i.e. no duplicates).

West Valley CIST-005B, Advanced Python

Strings Normalization

Strings are of different sizes. All samples in a network shall be of the same length. I chose the token 'X' to pad the strings up to the maximum length of the set. 'X' does not appear in any vocabulary, hence safe to use without interfering with the content of the strings.

Eg: In the set 10_20000_words, the string "abcdefg" is padded into 'abcdefgXXXXXXXXXXX' to reach a total length of 18 characters which is the maximum length for that set.

Furthermore, the inputted data are expected to have the shape (N,M,1) with

- N: size of the training batch
- M: length of the training samples
- 1: dimension of the training samples (1 since strings have a single dimension)

Eg: 10_20000_words has a shape of (181095, 18, 1)

Such shape is obtained by casting the strings converted into lists, using numpy.array. Indeed, the numpy arrays are generally the inputs for Keras data and the way Keras manipulates the data under the hood.

Finally, a percentage of the batch data may be spared to test the accuracy of the trained model to identify an eventual overfit.

In sum, the input data are normalized into:

- X_train with a shape of (p.N, M, 1) and Y_train with a shape of (p.N,1)
- X_test with a shape of ((1-p).N, M, 1) and Y_test with a shape of ((1-p).N, 1)

In the NN trials, p varies between 80% and 95%.

The Keras StringLookup layer helps vectorizing the strings of characters i.e. the padded human-readable strings are converted into a same-shaped array of integers which are machine-readable. Indeed, the gradient descent which is the core technology for any ML model is linear algebra computing weights and biases within the vectorial space R.

West Valley CIST-005B, Advanced Python

Neuronal Network – NN

The basic NN to predict palindromes is composed of a sequence of hidden layers. The general features of these hidden layers are:

- Xavier (Glorot) weights initialization, 0's for the biases
- reLu activation function
- initially no regularization, eventually experimented with an l1/l2 regulirizer

The output layer is a logistic regression hence composed of a single cell and a sigmoid activation function.

Generally, I consider a probability above 0.5 to be a palindrome and below 0.5 not to be a palindrome.

For the training, I used the defaults values of the Adam optimizer. Initially, no learning rate optimizer was used. Eventually used a learning rate decay optimizer to converge quicker to the loss function minimum.

The loss function I used was binary cross entropy well suited for such logistic regression. The accuracy metrics was binary accuracy.

West Valley CIST-005B, Advanced Python

Networks analysis

Network	Sample length	Sample size	LR	epoch	Hidden Layers	Training Accuracy	Test Accuracy	Validation
1 (all)	0-198	131_256	0.001	10	50/20	89.75%	Bias	Bias
2	0-198	131_256	0.001	50	50/20	94.24%	94.76%	Bias
3	0-198	131_256	0.001	10	500/200/30/10	89.29%	88.05%	Bias
4	0-198	1_270_396	Decay	1.5	500/200/20	91.74%	overtime	N/A
5	0-198	259_858	Decay	15	50/50	93.84%	94.05%	Bias
6 (words)	0-198	67_981	Decay	17	50/50	92.38%	92.12%	Bias
7	1-198	335_297	Decay	6	50/40	94.97%	93.63%	Bias
8	0-98	519_515	Decay	7	50/40	98.05%	97.89%	N/A
9	0-48	707_026	Decay	10	50/40	99.53%	99.52%	N/A
10	0-18	416_545	Decay	16	50/40	99.85%	99.80%	79%
11	0-18	800_710	Decay	13	50/40	99.97%	99.96%	79%
12	0-18	800_710	Decay	40	50/80/30, 0.01L1L2	92.10%	92.15%	Bias
13	0-18	800_710	Decay	8	50/80/30, 0.001L1L2	99.02%	98.61%	64%
14	0-18	3_723_744	Decay	4	50/40	99.99%	99.99%	64%

The table above is a summary of the different network configurations I experimented. More detailed information can be found below.

Networks 1-5 were based on long samples with different training batch sizes, and different number of hidden layers. All of these networks appeared to suffer from bias.

I identified that the sample set used for the training came from different distributions. Networks 6 and onwards focus only on string of letters. The number of hidden layers and their width don't appear to play a significant role. I experimented with different training set sizes up to about 3 million training samples.

None of these networks show any sign of variance or bias, except network 12 with a regularizer aimed at adding some noise.

Nonetheless, despite the good accuracy, many validation candidates I kept aside such as "Madam" or "Dad!" were poorly predicted to be palindromes.

The best network so far would be #11. Despite a 99.96% accuracy on both training and verification sets, only 79% of the validation candidates were properly identified as palindromes. This variance may highlight an overfit of the hyper-parameters.

I may need to test a different network algorithm such as Recurrent Neuronal Network (RNN) or a Long-Short Term Memory (LSTM) network, having both a memory component, to figuring out whether these validation candidates would get higher results.

Besides, it may be good to figure out whether the training/test and validation distributions are similar. If not, this may explain the bad validation results.

West Valley CIST-005B, Advanced Python

Network #1

Batch 100_100_all, 95% train, 5% test
Epochs: 10
Learning rate: 0.001

Two hidden layers: 50 neurons, 20 neurons.

After 10 epochs, the loss was 0.2292 and the accuracy 0.8975.

The accuracy showed too much bias.

Network #2

The bias in the network #1 needed to be lowered. I chose to increase the number of epochs to keep training the model longer in an attempt to get further improvements.

Batch 100_100_all, 95% train, 5% test
Epochs: 50
Learning rate: 0.001

Two hidden layers: 50 neurons, 20 neurons.

After 50 epochs, the loss was 0.1468 and the accuracy 0.9424.

The bias decreased to get a better accuracy compared to network #1. The test set provided an equivalent result with 0.9476.

Even though the results were better, 0.94 was not good enough yet to reliably predict palindromes.

Network #3

I attempted to increase the number of hidden layers to figure out whether more generalization could help. Since network #2 didn't show any sign of overfitting, I had low hopes that a wider and deeper NN would help.

Batch 100_100_all, 95% train, 5% test
Epochs: 10
Learning rate: 0.001

Four hidden layers: 500, 200, 30, 10

After 10 epochs, the binary accuracy was 0.8929 on the training set and 0.8805 on the test set.

The bias was still quite high, even worst than in network #2. At that point, I'd try to increase the training batch to expose the NN to more samples.

Network #4

Batch 100_1000_all, 95% train, 5% test
Epochs: 10
Learning rate: 0.001 with 0.9 decay rate

Three hidden layers: 50, 200, 20

While increasing the number of samples to process, I didn't want the model to spend too much time converging. Consequently, I enhanced the learning rate with a Decay optimizer with the default values to help converging quicker.

After 1.5 epochs in 1.5 days, I preferred aborted this training. The resources on my personal laptop are scarce, I have some constraints I cannot ignore. After 1.5 day was about 0.9174 for a loss of 0.1931. Compared to Network #1 which was quite quick to complete, I didn't see any advantage keeping this experiment way to expensive.

I needed to find a trade-off: increasing the training batch while having the training completed within reasonable time.

I also noticed that if for any reason the training aborted, I would have wasted a lot of time. Consequently, I improved my network in two ways using Keras callbacks:

- if the model doesn't learn anymore within 2 consecutive epochs, the training would stop
- if the training is interrupted, it could resume at the epoch were it stopped

Network #5

Batch 100_200_all, 95% train, 5% test
Epochs: 50
Learning rate: 0.001 with 0.9 decay rate

Two hidden layers: 50, 50

The training stopped after 15 epochs, meaning it had reached its learning plateau. The accuracy was 0.9384 for the training set and 0.9405 for the test set.

The bias was still important.

I noticed that the training set was composed of basically 3 different distributions:

- Words composed of letters
- Numbers composed of digits
- DNA/RNA sequences composed of nucleotides

Adding some "noise" in the training data may help avoiding overfitting to generalize better, however overfitting was not an issue. Maybe that training set was too much confusing for the model to learn anything meaningful.

Network #6

Batch 100_200_words, 95% train, 5% test
Epochs: 50
Learning rate: 0.001 with 0.9 decay rate

Two hidden layers: 50, 50

After 17 epochs the learning plateau'ed with a loss of 0.1883 and an accuracy of 0.9238. The test set showed an accuracy of 0.9212.

The bias was worst than the one for Network #5.

However 100_200_all had a total of about 260k samples whereas 100_200_words had about 68k samples. I had the same concern that I faced in Network #3: not enough training samples.

Network #7

Batch 100_1000_words, 90% train, 10% test
Epochs: 50
Learning rate: 0.001 with 0.9 decay rate

Two hidden layers: 50, 40

After 6 epochs no learning occurred anymore, reaching an accuracy of 0.9497 for the training and 0.9363 for the test set. The results were close enough to eliminate the variance. I still had a bias problem. Hence I needed to keep increasing the sample size.

Networks #8, #9, #10

These 3 networks are similar, except for the size of the training sets.

Epochs: 50
Learning rate: 0.001 with 0.9 decay rate

Two hidden layers: 50, 40

Batch	Final epoch	Loss	Training accuracy	Test accuracy
50_5000_words	7	0.0730	0.9805	0.9789
25_20000_words	10	0.0245	0.9953	0.9952
10_50000_words	16	0.0080	0.9985	0.9980

The more samples I had, the better training I got for the same model. The length of the samples decreased in order to increase the number of samples to accommodate my scarce resources while trying to prove that the more data to train decrease the bias.

Both training and test accuracy are equivalent, showing no variance overfit.

At that point, I was confident that the network #10, limited to 18-character long candidates, would reliably predict the result.

Then I tried a few examples such as “madam” or “dad”. The model predicted that ‘dad’ was likely a palindrome (76.80%) while ‘madam’ was not (5.10^{-23}). With an accuracy rate of almost 100%, it’s improbable that ‘madam’ would be a false negative.

I assumed the model was not trained with enough samples in the distribution of shorter strings.

Network #11

Batch 10_100000_words, 80% train, 20% test
Epochs: 50
Learning rate: 0.001 with 0.9 decay rate

Two hidden layers: 50, 40

After 13 epochs, the loss was 0.0034 for an accuracy of 99.97% on the training set and 99.96% on the test set.

I trained that network with higher number of samples on shorter strings to get almost 100% of accuracy.

For this model, ‘dad’ was identified to likely be a palindrome (68.49%) while ‘madam’ was still not identified to likely be a palindrome (5.10^{-13}).

Network #12

Due to disappointing results from Network #11, I chose to explore more generalizing architectures as if I had to compensate for overfitting (what the previous results didn’t show).

I added for each hidden layer a regularizer aiming at injecting some noise in the samples.

Batch 10_100000_words, 80% train, 20% test
Epochs: 40
Learning rate: 0.001 with 0.9 decay rate
Regularizer: l1 = l2 = 0.01

Three hidden layers: 50, 80, 30

The model was trained all way through the 40 epochs ending up with a loss of 0.23, a training accuracy of 92.10% and test accuracy of 92.15%. Again, there was no blatant overfit, however the bias issue was back: that was somehow expected.

Network #13

That network aimed at experimenting a larger network with a lower regularization rate.

Batch 10_100000_words, 80% train, 20% test
Epochs: 40
Learning rate: 0.001 with 0.9 decay rate
Regularizer: l1 = l2 = 0.001

Four larger hidden layers: 500, 200, 30, 100

After 8 epochs, the loss was 0.10 with an accuracy of 99.02% and test accuracy of 98.61%. The results are similar to Network #11 although it was smaller.

'Dad' was predicted not to be a palindrome (0.0045) and 'madam' as well with 0.0045 probability.

This model, despite having similar accuracy results than Network #11, was actually worst.

In sum, it's unlikely that there is any overfitting problem, hence keeping the network small with regularization seems to be good enough.

Network #14

Batch 10_500000_words, 80% train, 20% test
Epochs: 4
Learning rate: 0.001 with 0.9 decay rate
Two hidden layers: 50, 40

The training batch to process is about 3 million samples. Processing such amount of data using scarce personal laptop resources needs about 12 hours per epoch. Thus, I limited this trial to 4 epochs to have the experiment terminating within a "reasonable" time frame over a few days.

The loss ended up being 0.0040, training accuracy 99.99% and test training accuracy being 99.99% as well. Unfortunately, 'madam' and 'dad' are rated to be unlikely palindromes with respectively $5.76e^{-12}$ and 0.37 below the 0.50 threshold. These results are quite disappointing: the expectation was to see better results with a larger training set.

I start wondering whether I'm truly working with comparable distributions at this point.