guillaume-chevalier / **HAR-stacked-residual-bidir-LSTMs**

Using deep stacked residual bidirectional LSTM cells (RNN) with TensorFlow, we do Human Activity Recognition (HAR).
Classifying the type of movement amongst 6 categories or 18 categories on 2 different datasets.

tensorflow lstm rnn stacked-layers residual-layers residual-lstm-cells bidirectional-lstm-cells

guillaume-chevalier committed on **GitHub** Added reference to Google's NMT — Latest commit 69d5242 on 24 May

data	Reverted to no accelerometer filtering	7 months ago
.gitignore	Adapted project to receive the opportunity dataset, some cleaning and…	8 months ago
3x3_result_HAR_6.txt	Update 3x3_result_HAR_6.txt	6 months ago
3x3_result_opportunity_18.txt	Shortened time windows which yields similar results	6 months ago
LICENSE	Adapted project to receive the opportunity dataset, some cleaning and…	8 months ago
README.md	Added reference to Google's NMT	3 months ago
architecture_example_2x2.jpg	Added image of architecture	5 months ago
config_dataset_HAR_6_classes.py	add result of Deep LSTM for HAR_6_classes	6 months ago
config_dataset_opportunity_18_cla…	Shortened time windows which yields similar results	6 months ago
lstm_architecture.py	Reverted to no accelerometer filtering	7 months ago
sliding_window.py	Adapted project to receive the opportunity dataset, some cleaning and…	8 months ago

README.md

HAR-stacked-residual-bidir-LSTM

The project is based on this repository which is presented as a tutorial. It consists of Human Activity Recognition (HAR) using stacked residual bidirectional-LSTM cells (RNN) with TensorFlow.

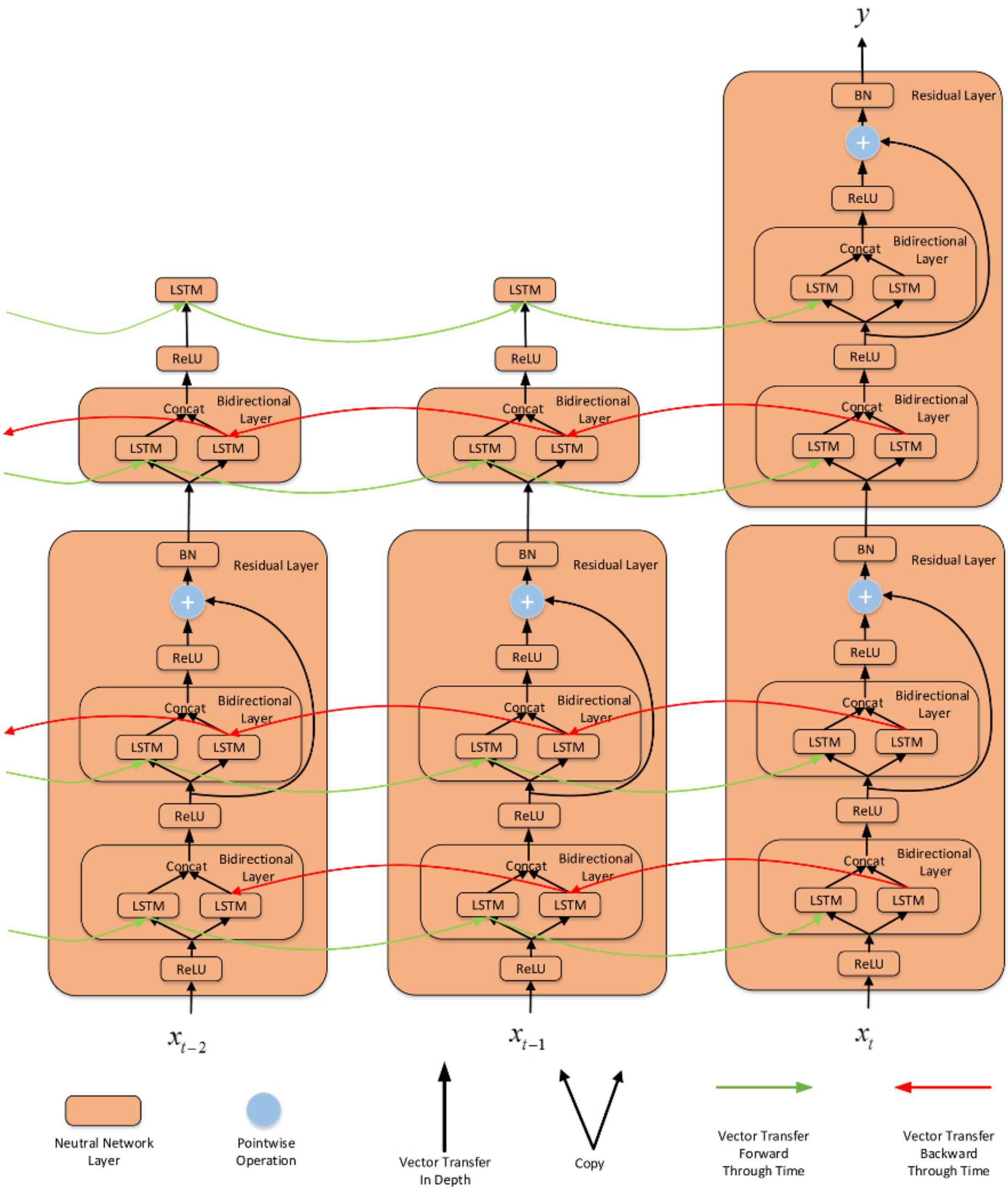
It resembles to the architecture used in "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation" without an attention mechanism and with just the encoder part. In fact, we started coding while thinking about applying residual connections to LSTMs - and it is only afterwards that we saw that such a deep LSTM architecture was already being used.

Here, we improve accuracy on the previously used dataset from 91% to 94% and we push the subject further by trying our architecture on another dataset.

Our neural network has been coded to be easy to adapt to new datasets (assuming it is given a fixed, non-dynamic, window of signal for every prediction) and to use different breadth, depth and length by using a new configuration file.

Here is a simplified overview of our architecture:

Simplified view of a "2x2" architecture. We obtain best results with a "3x3" architecture (details below figure).



Bear in mind that the time steps expands to the left for the whole sequence length and that this architecture example is what we call a "2x2" architecture: 2 residual cells as a block stacked 2 times for a total of 4 bidirectional cells, which is in reality 8 unidirectional LSTM cells. We obtain best results with a 3x3 architecture, consisting of 18 LSTM cells.

Neural network's architecture

Mainly, the number of stacked and residual layers can be parametrized easily as well as whether or not bidirectional LSTM cells are to be used. Input data needs to be windowed to an array with one more dimension: the training and testing is never done on full signal lengths and use shuffling with resets of the hidden cells' states.

We are using a deep neural network with stacked LSTM cells as well as residual (highway) LSTM cells for every stacked layer, a little bit like in ResNet, but for RNNs.

Our LSTM cells are also bidirectional in term of how they pass trough the time axis, but differ from classic bidirectional LSTMs by the fact we concatenate their output features rather than adding them in an element-wise fashion. A simple hidden ReLU layer then lowers the dimension of those concatenated features for sending them to the next stacked layer. Bidirectionality can be disabled easily.

Setup

We used `TensorFlow 0.11` and `Python 2`. `Sklearn` is also used.

The two datasets can be loaded by running `python download_datasets.py` in the `data/` folder.

To preprocess the second dataset (opportunity challenge dataset), the `signal` submodule of scipy is needed, as well as `pandas`.

Results using the previous public domain HAR dataset

This dataset named `A Public Domain Dataset for Human Activity Recognition Using Smartphones` is about classifying the type of movement amongst six categories: (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING).

The bests results for a **test accuracy of 94%** are achieved with the 3x3 bidirectional architecture with a learning rate of `0.001` and an L2 regularization multiplier (weight decay) of `0.005`, as seen in the `3x3_result_HAR_6.txt` file.

Training and testing can be launched by running the config: `python config_dataset_HAR_6_classes.py`.

Results from the Opportunity dataset

The neural network has also been tried on the Opportunity dataset to see if the architecture could be easily adapted to a similar task.

Don't miss out this nice video that offers a nice overview and understanding of the dataset.

We obtain a **test F1-score of 0.893**. Our results can be compared to the state of the art DeepConvLSTM that is used on the same dataset and achieving a test F1-score of 0.9157.

We only used a subset of the full dataset as done in other research in order to simulate the conditions of the competition, using 113 sensor channels and classifying on the 17 categories output (and with the NULL class for a total of 18 classes). The windowing of the series for feeding in our neural network is also the same 24 time steps per classification, on a 30 Hz signal. However, we observed that there was no significant difference between using 128 time steps or 24 time steps (0.891 vs 0.893 F1-score). Our LSTM cells' inner representation is always reset to 0 between series. We also used mean and standard deviation normalization rather than min to max rescaling to rescale features to a zero mean and a standard deviation of 0.5. More details about preprocessing are explained furthermore in their paper. Other details, such as the fact that the classification output is sampled only at the last timestep for the training of the neural network, can be found in their preprocessing script that we adapted in our repository.

The config file can be runned like this: `config_dataset_opportunity_18_classes.py`. For best results, it is possible to readjust the learning rate such as in the `3x3_result_opportunity_18.txt` file.