Re-implementation of “Attention-based End-to-End Models for Small-Footprint Keyword Spotting”

Hu Wenchao

2018/05/20

About the paper

KWS:
keyword spotting, also know as spoken term detection (STD), is a task to detect pre-defined keywords in a stream of audio.

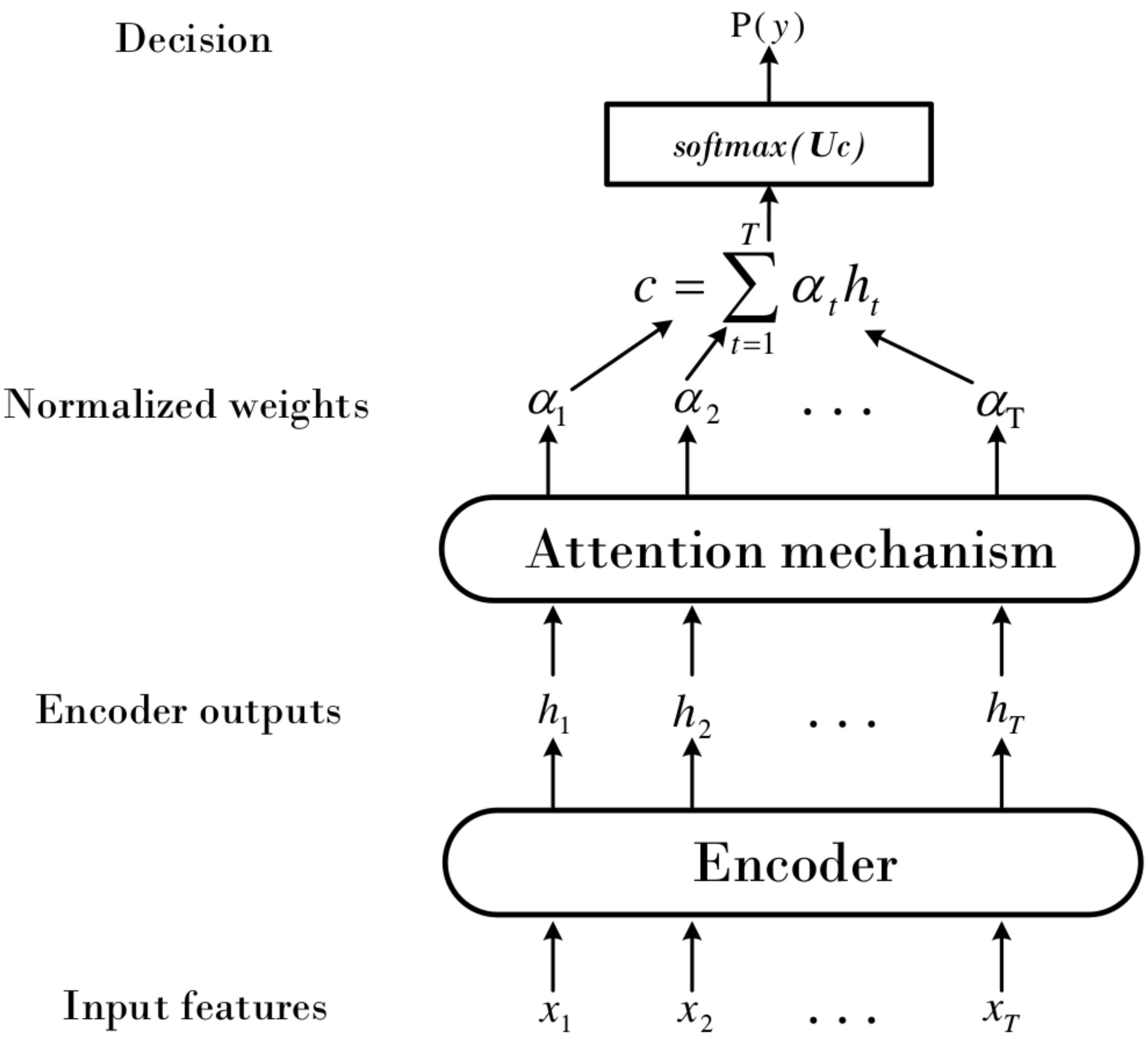
Challenges:
1. minimize the false rejection rate (FRR) at a low false alarm (FA) rate.
2. limit memory usage, enable instant reponse, lower computational cost.

Problems with existing systems:
1. LVCSR is flexible to change keyword, but computation cost is too high.
2. HMM remains strongly competitive until today, but outperformed by DNN models.
3. Deep KWS is quite simple, but it needs a well-trained acoustic model to obtain frame-level alignments.

Advantages of the model in this paper:
1. a simple model that directly outputs keyword detection.
2. no complicated searching involved.
3. no alignments needed beforehand to train the model.

Best performance of the model:
The CRNN-based attention model (~84K parameters) achieves 1.02% FRR at 1.0 FA per hour.

Model structure



Softmax classification

Attention: can be average attention or soft attention, soft attention is better. It weights the encoder features and generates a fixed-length vector.

Encoder: can be RNN, LSTM, GRU, and CRNN, transforms the input signal into a high level representation.

Datasets

In the paper, authors use real-world wake-up data collected from Mi AI Speaker, which I cannot acquire.

Data used by me:
https://github.com/castorini/honk, I used the data from this open project.

Keyword: olivia

Positive data: 194;   Negative data: 873; Duration of each data file: 1s.

Here the data is randomly split into train/test by 80:20.

Possible data augmentation methods (not implemented):

https://github.com/enggen/Deep-Learning-Coursera/blob/master/Sequence%20Models/Week3/Trigger%20word%20detection/Trigger%20word%20detection%20-%20v1.ipynb

By adding different background to the words, it is possible to obtain a much larger dataset. Moreover, it also enable us to create longer sound tracks by randomly insert the words into a longer background wave file.

Feature engineering

Mel-filterbank channel: 40
Frame window: 25ms
Frame shift: 10ms
Final feature: per-channel energy normalized (PCEN) Mel-spectrograms.

$$PCEN(t, f) = \left(\frac{E(t, f)}{(\epsilon + M(t, f))^{\alpha}} + \delta \right)^{r} - \delta^{r}$$

Where t and f denote time and frequency index and E(t, f) denote filterbank energy in each time-frequency (T-F) bin.

$$M(t, f) = (1 - s)M(t - 1, f) + sE(t, f)$$

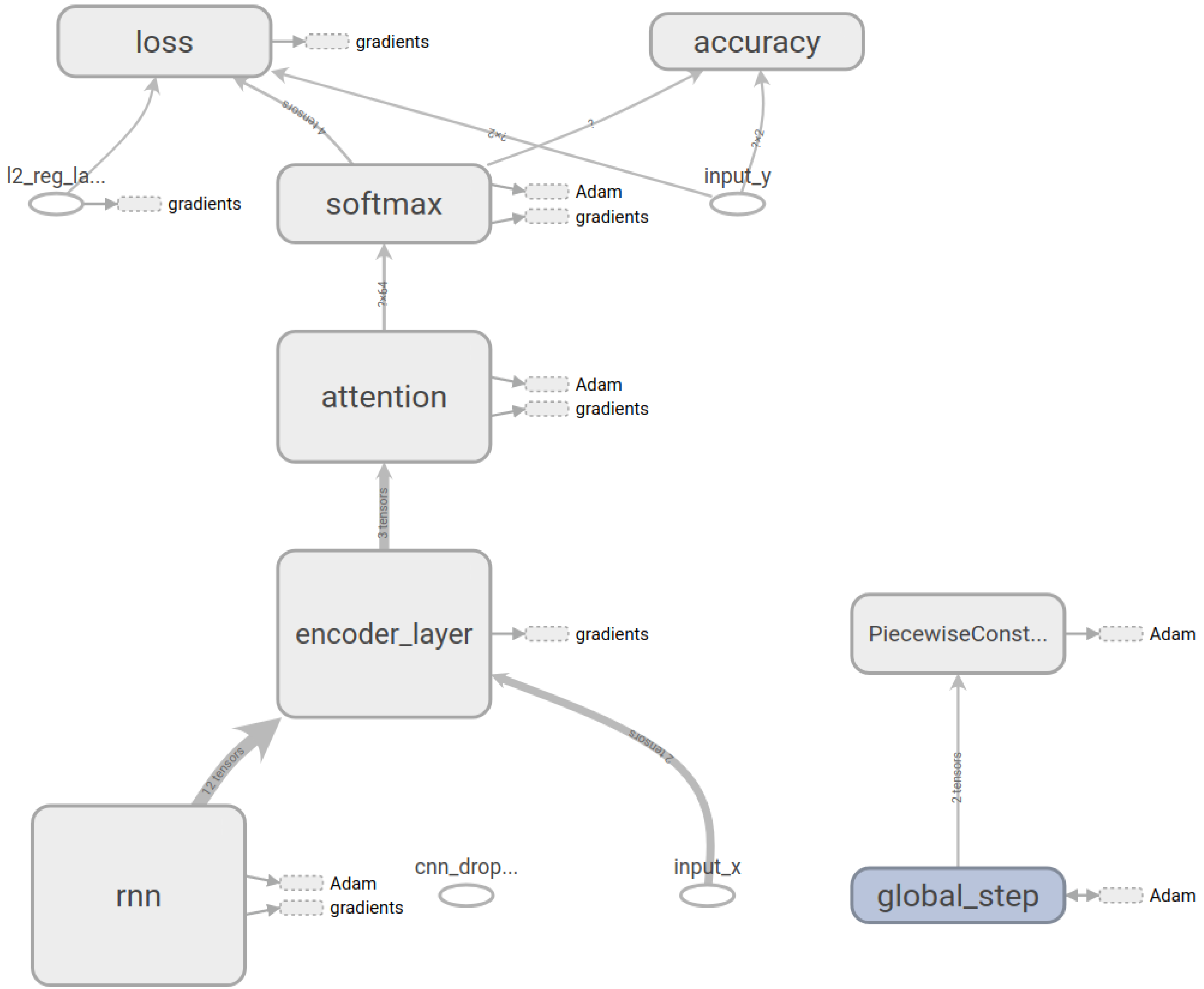
Where s is the smoothing coefficient, epsilon is a small constant to prevent division by zero.

This feature transform can be computed by (using default value from the paper):

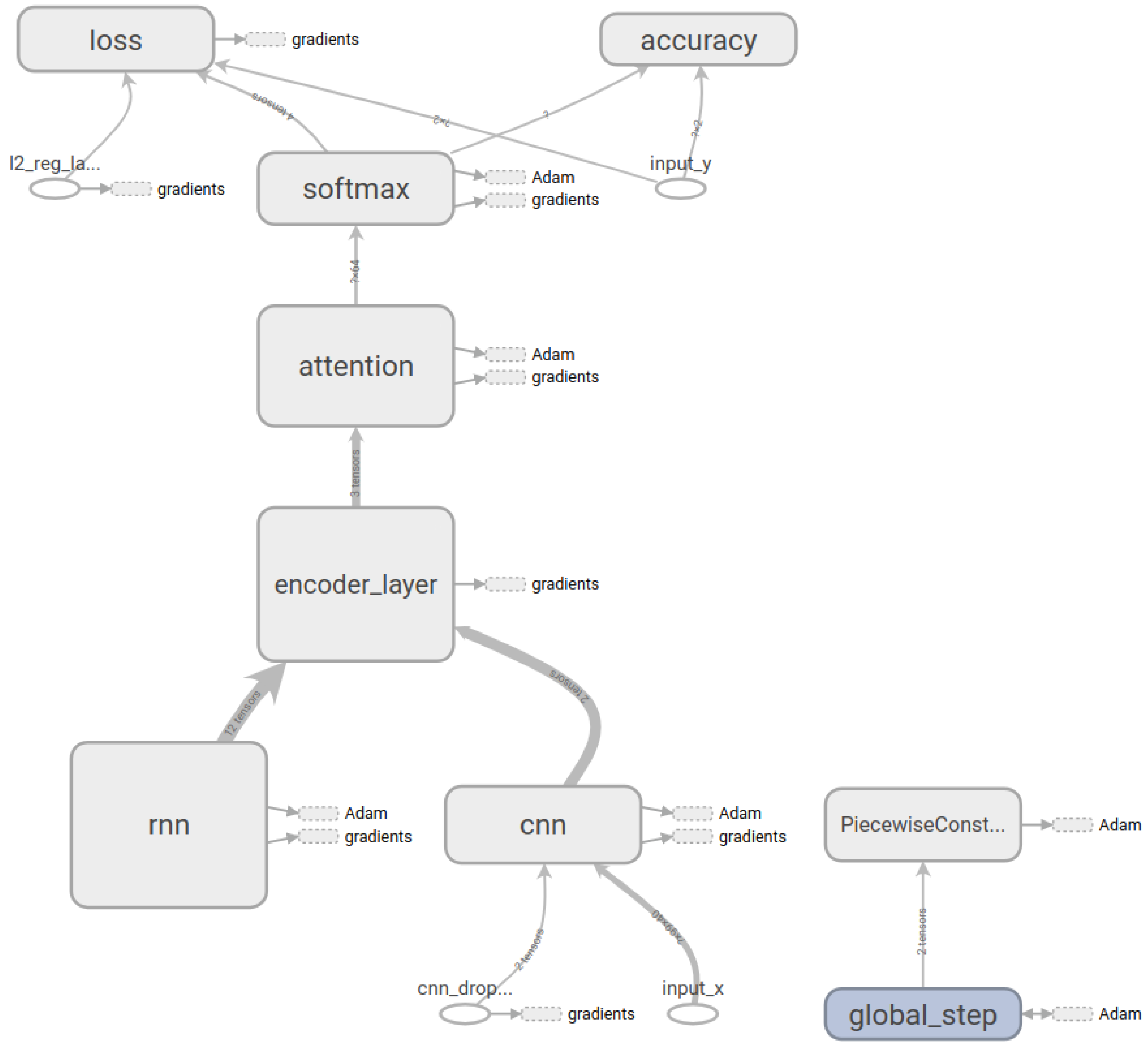
```
def gen_pcen(E, alpha=0.98, delta=2, r=0.5, s=0.025, eps=1e-6):
    M = scipy.signal.lfilter([s], [1, s - 1], E)
    smooth = (eps + M)**(-alpha)
    return (E * smooth + delta)**r – delta**r
```

After this, each 1s .wav sample is mapped to a [99, 40] array.

Model structure, GRU + attention



Model structure, RCNN + GRU + atention



Performance

Training steps: 4800, attention size: 100,
Test data: 39 positive, 175 negative
Result: RCNN > GRU, GRU with more nodes > GRU with more layers, this trend matches the result in the paper.

Model	Model detail	Precision	Recall
RNN + ATT	1-64	86.84%	84.62%
LSTM + ATT	1-64	89.47%	87.18%
GRU + ATT	1-64	94.74%	92.31%
GRU + ATT	1-128	97.37%	94.87%
GRU + ATT	2-64	92.31%	92.31%
GRU + ATT	3-64	94.74%	92.31%
RCNN + GRU + ATT	16-1-64	84.44%	97.44%
RCNN + GRU + ATT	16-3-64	92.68%	97.44%
RCNN + GRU + ATT	16-3-128	95.12%	100%

Remaining problem

1. In the paper, the author set all initial biases to 0, which will slow the convergence of the model, I am not sure whether it is a good choice.

2. In the paper, it use 189 frames for input during training, but use 100 frames during running, I am not very clear how to act like this, since there is a CNN layer ahead of RNN, in which a fix size is required by the CNN.

3. In this paper, the CRNN structure use a quite large filter (20 x 5), if we use several small CNN filters (3 x 3 + 1 x 1, like VGG structure) instead, is it possible to achieved better performance?

Thank You!