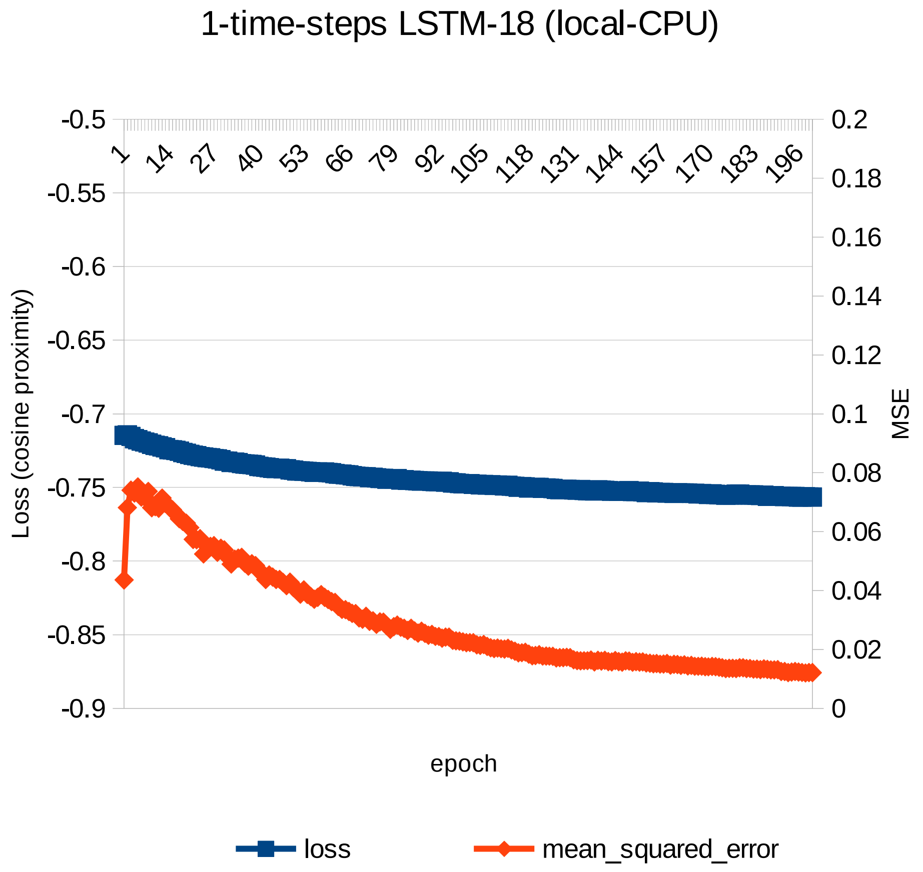
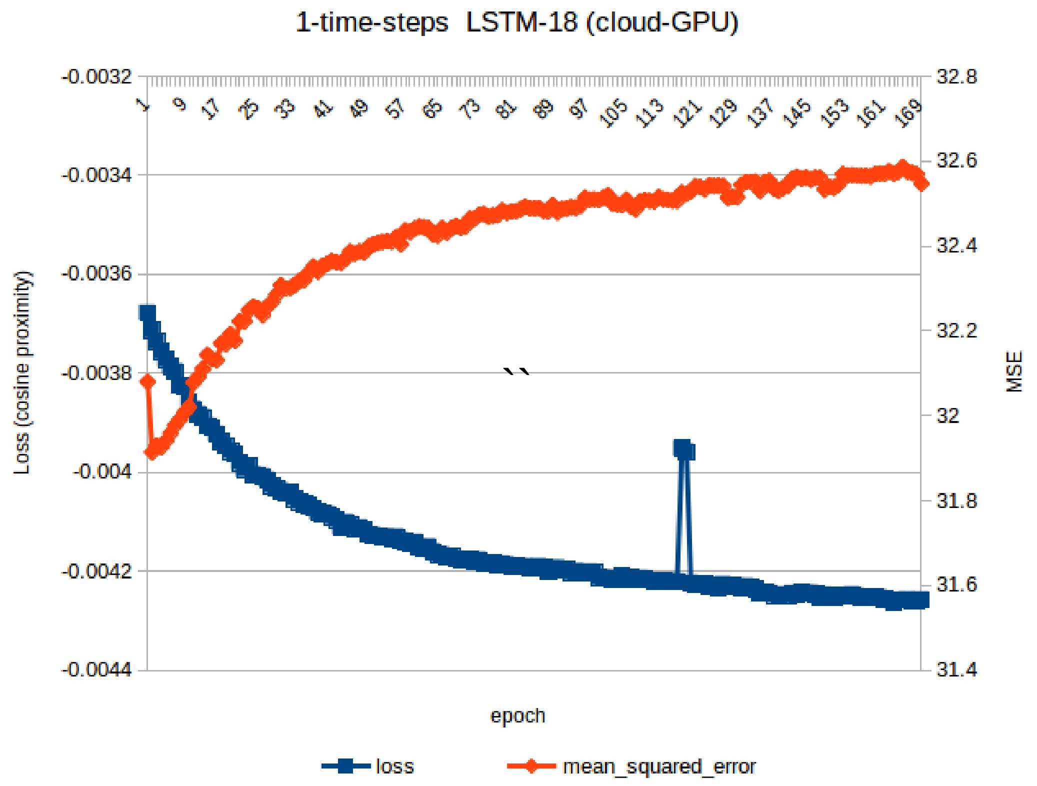
Job Description & Job Titles Matching Report - 2

Strayn Wang

2018.8.7

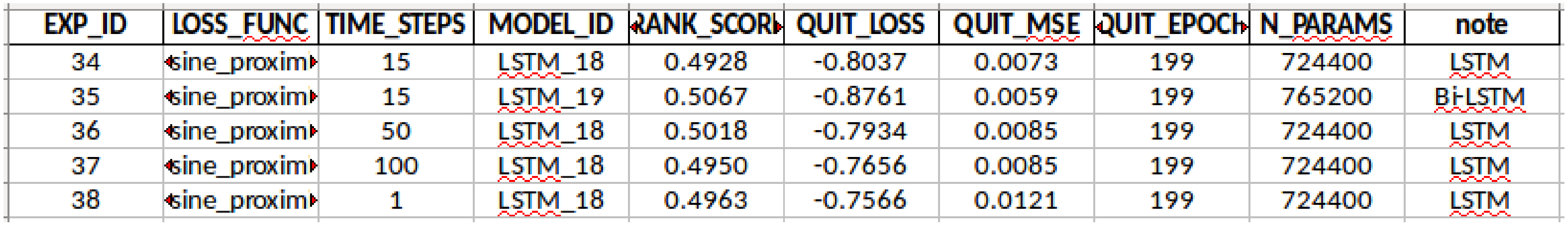
1. Explanation for the previous learning-nothing problem





After comparing the logs of cloud-GPU server and that of local-CPU,
I found the problem tiny-loss problem was caused by using cloud-GPU, all experiments, with no exclusion, from cloud has this problem whereas none for my local-CPU.
It may be a Keras-GPU bug, e.g. the loss wasn’t allocated from multiple GPU computing unit, or it could be my fault configuration of GPU computing. Whatever reason leads to this, looking at the MSE calculated along the learning (not used for gradient descent but only for monitoring), it did influence the learning and made it garbage. Given limited time, I re-run the afterwards experiments on my local machine with CPU, and that’s what took time.

2. Exploring Model Structure with local CPU

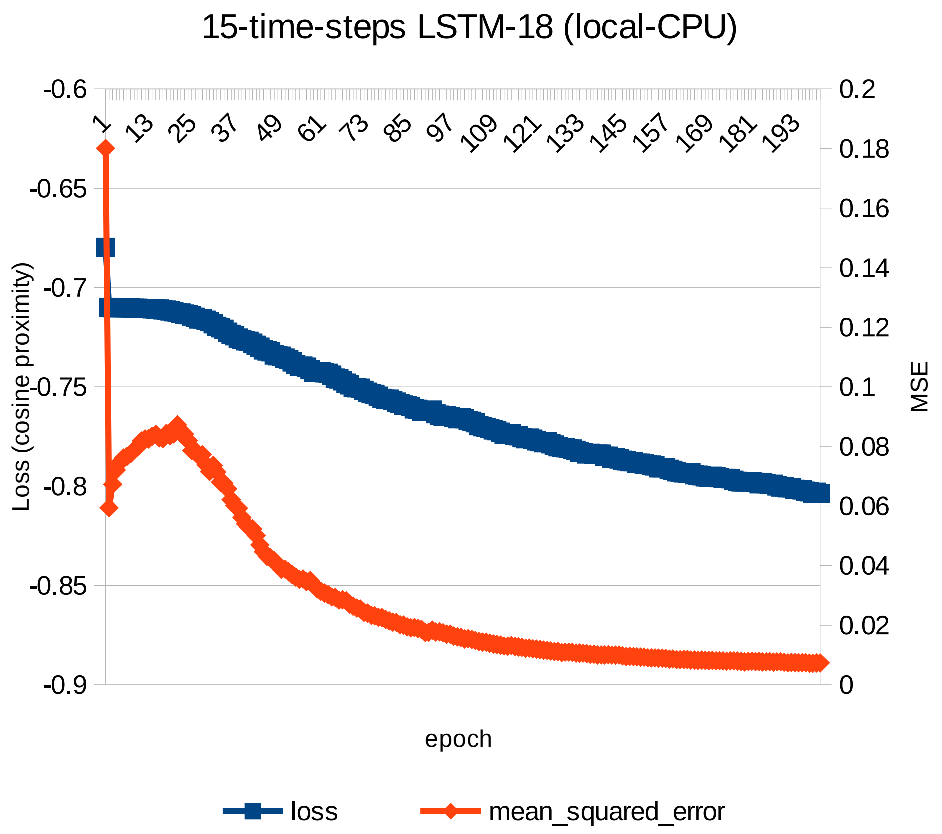


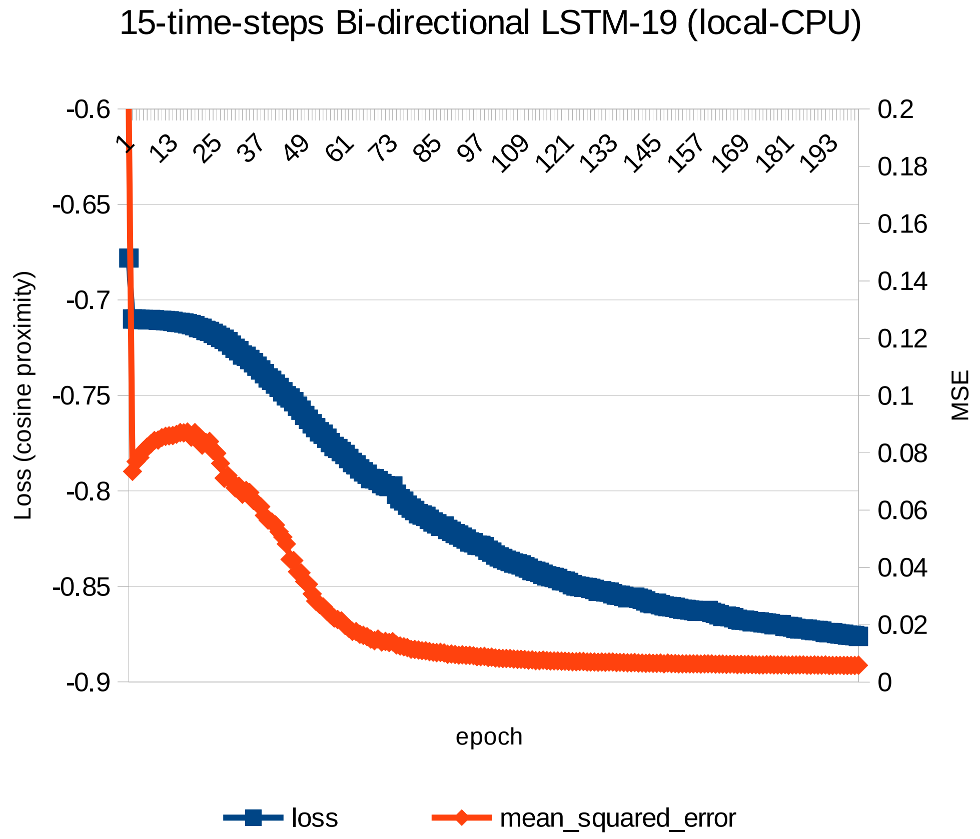
EXP_ID	LOSS_FUNC	TIME_STEPS	MODEL_ID	RANK_SCOR	QUIT_LOSS	QUIT_MSE	QUIT_EPOCH	N_PARAMS	note
34	sine_proxim	15	LSTM_18	0.4928	-0.8037	0.0073	199	724400	LSTM
35	sine_proxim	15	LSTM_19	0.5067	-0.8761	0.0059	199	765200	Bi-LSTM
36	sine_proxim	50	LSTM_18	0.5018	-0.7934	0.0085	199	724400	LSTM
37	sine_proxim	100	LSTM_18	0.4950	-0.7656	0.0085	199	724400	LSTM
38	sine_proxim	1	LSTM_18	0.4963	-0.7566	0.0121	199	724400	LSTM

With above 5 experiments, I tested 3 factors:
a. Cosine Proximity Loss v.s. Mean Squared Error Loss
b. Number of time-steps used for recurrent network (LSTM actually)
c. LSTM v.s. Bi-directional LSTM

Conclusion, considering the balance of variance & bias:
a. Cosine Proximity Loss or MSE? More experiments required.
b. Time-steps 15 should be used, for it's good and fast.
c. Bi-directional LSTM should be used, for it's powerful at fitting, but regularization/dropout/noises should added to avoid overfitting.

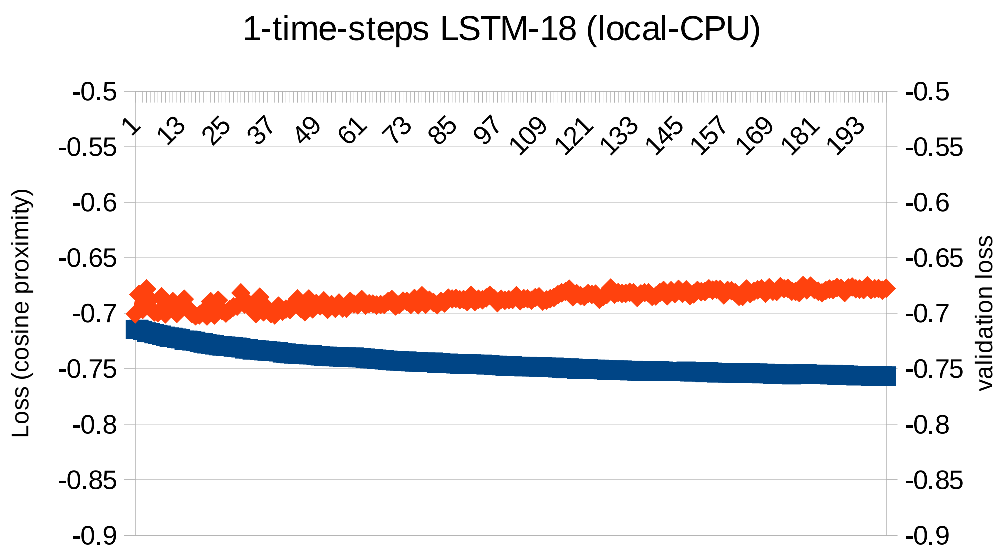
a. Cosine Proximity Loss v.s. Mean Squared Error Loss

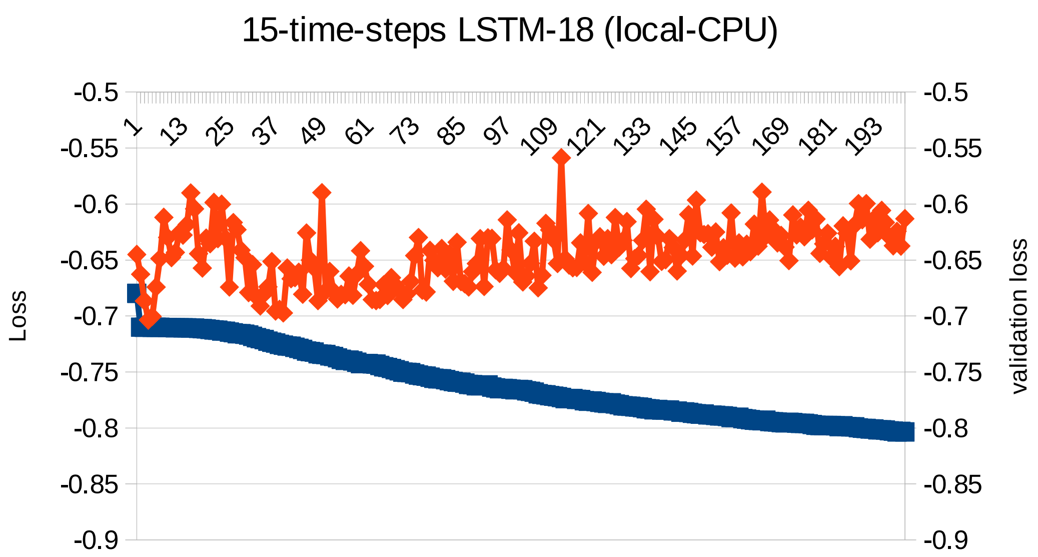


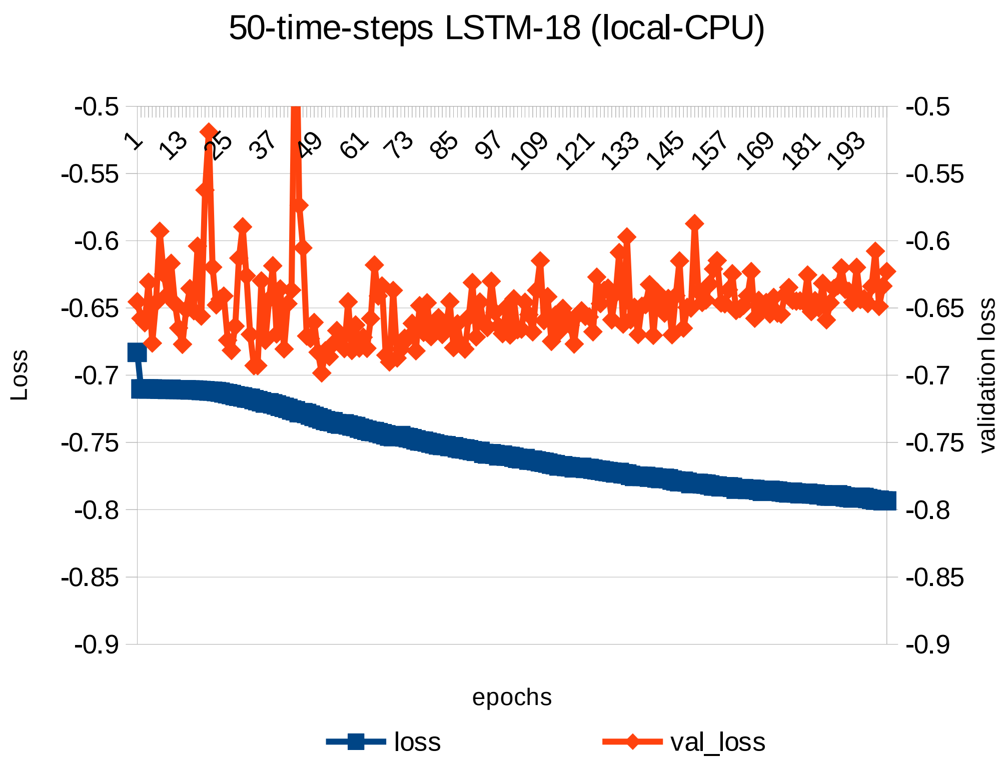


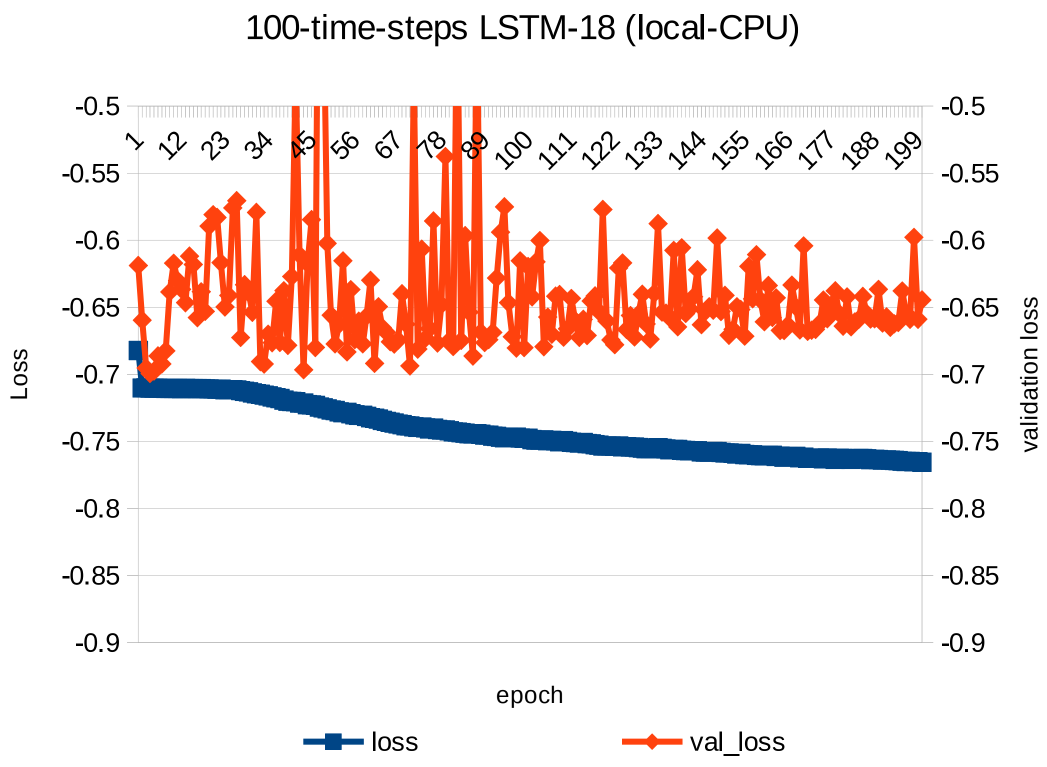
1. Cosine similarity is faster(steeper) than MSE at beginning of training
2. Cosine similarity converges slower than MSE.

b. Number of time-steps used for recurrent network

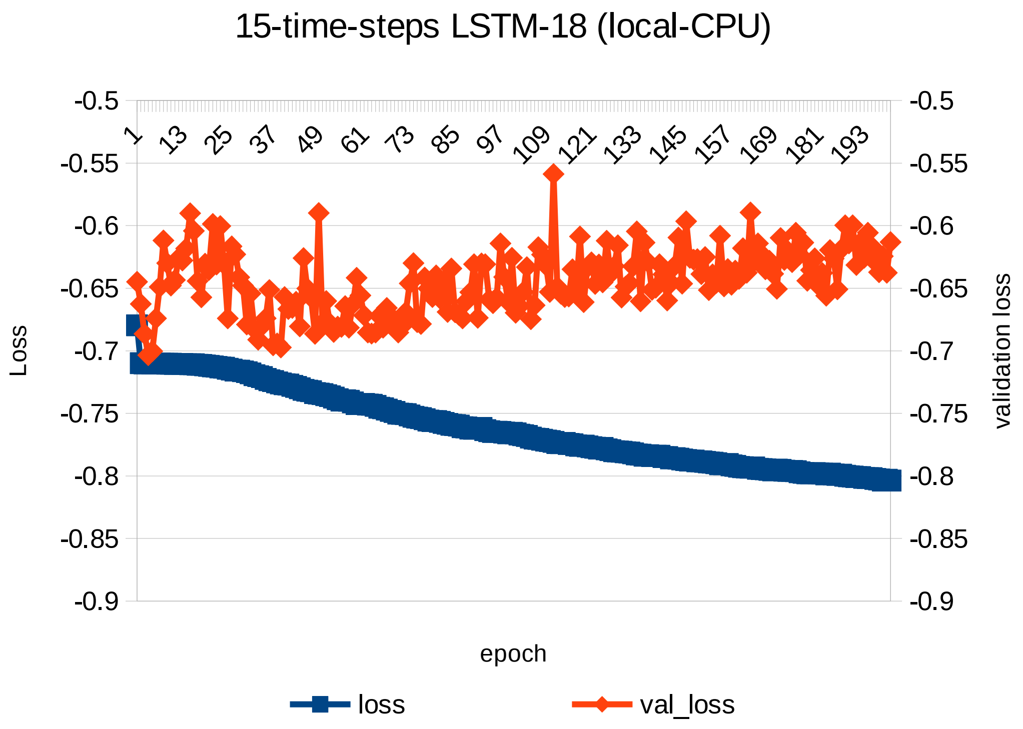


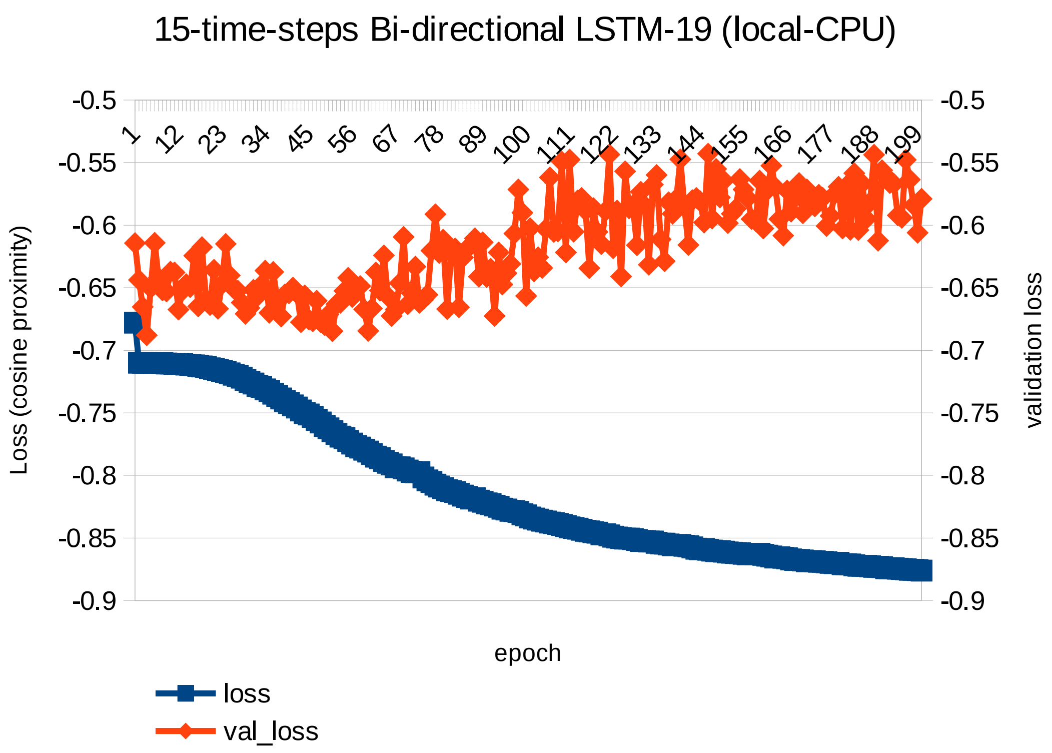






c. LSTM v.s. Bi-directional LSTM





Given similar capacity (number of parameters),
1. Bi-directional LSTM learns faster (steeper loss, bigger gradients) than LSTM.
2. Lower minimal loss suggests Bi-directional LSTM more powerful at fitting than LSTM.
3. The bigger validation loss of Bi-directional LSTM suggest that it tends to overfit the data regularization/noises/dropouts should be added.

3. Next - 1

A title can be split into a **descriptive component (DC)** and an **objective component (OC).** In English, the last noun in the phrase is normally the OC and what’s before it the DC.

For example, in “product manager”, the “product”, although a noun but it is a DC, and “manager” is the OC.

Therefore, it makes sense to split each job title fastText-embedding into two component, i.e. rather than embedding full title into a full embedding vector, we can embed DC into vector A and OC into vector B and concatenate them together to form the vector representation for a job title.

3. Next - 2

The descriptive components sometimes contain information of job level, which suggests whether it is a senior/junior/intern/entry-level position.

However, this is not always explicitly showed in the job titles, making them noises for model.

To de-noise job titles by this sense, we can get rid of "Jr., Sr., Intern ..." in training labels and add a numeric value to represent position level.

And we predict the level-free job titles and numeric job level seperately.

3. Next - 3

As stated before, a job description can be split into “qualification, responsibility, skills, experience level, education level ...”, given enough time, the separate handlings of these different sections should be added to the model input layers. And also, an embedding of input words should help.