Tips for Training Recurrent Neural Networks

Jun 30, 2017

Some practical tricks for training recurrent neural networks:

Optimization Setup

- **Adaptive learning rate.** We usually use adaptive optimizers such as Adam (Kingma14) because they can better handle the complex training dynamics of recurrent networks that plain gradient descent.
- **Gradient clipping.** Print or plot the gradient norm to see its usual range, then scale down gradients that exceeds this range. This prevents spikes in the gradients to mess up the parameters during training.
- **Normalizing the loss.** To get losses of similar magnitude across datasets, you can sum the loss terms along the sequence and divide them by the maximum sequence length. This makes it easier to reuse hyper parameters between experiments. The loss should be averaged across the batch.
- **Truncated backpropagation.** Recurrent networks can have a hard time learning long sequences because of vanishing and noisy gradients. Train on overlapping chunks of about 200 steps instead. You can also gradually increase the chunk length during training. Preserve the hidden state between chunk boundaries.

- **Long training time.** Especially in language modeling, small improvements in loss can make a big difference in the preserved quality of the model. Stop training when the training loss does not improve for multiple epochs or the evaluation loss starts increasing.
- **Multi-step loss.** When training generative sequence models, there is a trade-off between 1-step losses (teacher forcing) and training longer imagined sequences towards matching the target (Chiappa17). Professor forcing (Goyal17) combines the two but is more involved.

Network Structure

- **Gated Recurrent Unit.** GRU (Cho14) alternative memory cell design to LSTM. I found it to often reach the equal or better performance while using fewer parameters and being faster to compute.
- **Layer normalization.** Adding layer normalization (Ba16) to all linear mappings of the recurrent network speeds up learning and often improves final performance. Multiple inputs to the same mapping should be normalized separately as done in the paper.
- **Feed-forward layers first.** Preprocessing the input with feed-forward layers allows your model to project the data into a space with easier temporal dynamics. This can improve performance on the task.
- **Stacked recurrent networks.** Recurrent networks need a quadratic number of weights in their layer size. It can be more efficient to stack two or three smaller layers instead of one big one. Sum the outputs of all layers instead of using only the last one, similar to a ResNet or DenseNet.

Model Parameters

- **Learned initial state.** Initializing the hidden state as zeros can cause large loss terms for the first few time steps, so that the model focuses less on the actual sequence. Training the initial state as a variable can improve performance as described in this post.
- **Forget gate bias.** It can take a while for a recurrent network to learn to remember information form the last time step. Initialize biases for LSTM's forget gate to 1 to remember more by default. Similarly, initialize biases for GRU's reset gate to -1.
- **Regularization.** If your model is overfitting, use specific regularization methods for recurrent networks. For example recurrent dropout (Semeniuta16) or Zoneout (Krueger17).

I hope this collection of tips is helpful. Please feel free to post further suggestions for the list and ask questions.

9 Comments Danijar Hafner 1 Login

Recommend Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Kory Mathewson • a month ago
Nice post... also might note skip-layer connections and early stopping.
• Reply • Share ›

Jeffrey • 2 months ago
Summing (or concatenating) RNN layers seems cumbersome in TensorFlow since tf.nn.dynamic_rnn returns the output sequence of the last RNN layer only. Do you know of a straightforward solution to this? I find myself writing custom versions of TF RNN helper functions all too often...

Edit: I take it back. This isn't all that cumbersome since you can just call dynamic_rnn multiple times.
• Reply • Share ›

Danijar Mod ➦ Jeffrey • 2 months ago

Yes, just use `tf.nn.dynamic_rnn()` multiple times.

If you want a single cell, you can use `tf.contrib.rnn.ResidualWrapper` to create an additive skip connection around a cell. Then combine your wrapped layers with `tf.contrib.rnn.MultiRNNCell`.

It's also quite easy to implement your own cell wrappers, for example to implement DenseNet-style any-to-any connections. You can take a look at rnn_cell_impl.py for inspiration.

Reply • Share ›

lawrence mcdonell • 2 months ago

You are well brainy style :) thanks

Reply • Share ›

Niger1738 • 2 months ago

> Sum the outputs of all layers instead of using only the last one, similar to a ResNet or DenseNet.

By sum, you mean concat?

Reply • Share ›

Danijar Mod ➦ Niger1738 • 2 months ago

Skip connections are indeed usually summed to the output, not concatenated. Concatenation requires many more parameters, even though it might seem more intuitive. The recent work on Dual Path Networks suggests that both summing and concatenating are important and it would be interesting to see if that helps for multi-layer RNNs as well: https://arxiv.org/pdf/1707.....

Reply • Share ›

Gopal Sharma ➦ Niger1738 • 2 months ago

I have the same doubt.

Reply • Share ›

Sergey Serebryakov • 2 months ago

> 1. Train on overlapping chunks of about 200 steps instead.
> 2. Preserve the hidden state between chunk boundaries.

This is interesting. What if I have one sequence 123456789. Let's say, I use chunks of length 3. If they do not overlap (123, 456, 789) preserving hidden state makes sense. If they overlap, let's say with step size 2 (123, 345, 567, 789), what's the motivation behind preserving hidden state? Does it result in better convergence and final accuracy?
I am also curios if there is a significant difference in how initial state is initialized? All zeros or trainable vector?

Reply • Share ›

Danijar Mod ➦ Sergey Serebryakov • 2 months ago

The idea is to preserve the correct hidden states, so that the RNN sees one long sequence during the forward pass. For example, for chunk 123 you'd save the hidden state after 2. Then use that to initialize the RNN for the next chunk 345.

I did not find a significant difference between zeros, noise, or a trained variable as initial state. Some other people saw noticeable improvements though, so I think it depends on the task. The shorter the sequences, the more important the initial state might be.

• Reply • Share ›

ALSO ON **DANIJAR HAFNER**

Introduction to Recurrent Networks in TensorFlow

47 comments • a year ago•

Danijar — You can get a zero state via `cell.zero_state(batch_size, dtype=tf.float32)`. If you print this, you'll see

Structuring Your TensorFlow Models

42 comments • a year ago•

kaufmanuel — Great article! I'm always looking for how to improve the structure of my tensorflow code. Specifically, I'd be ...

What is a TensorFlow Session?

12 comments • a year ago•

h4k1m — Indeed, it should be replaced with tf.global_variables_initializer().

Variable Sequence Lengths in TensorFlow

65 comments • a year ago•

Danijar — Yes, the sequences should be aligned to the left and have additional zeros at the end.

Danijar Hafner

mail@danijar.com
danijar
danijarh

I'm a researcher and engineer aiming to build intelligent machines based on concepts of the human brain.