Slide-5-Managing-your-experiment

Created	@Apr 10, 2021 10:43 AM

World Embedding in Tensorflow

▼ One-hot representation?

- Each word is represented by one vector with a single 1 and the rest is 0
- But: Vocal can be large, can't `represent relation` btw word

▼ What is word embedding?

- It is Distributed representation of word
- It perform as continues values
- Low dimension and capture the semantic meaning btw words

▼ What is couting technique?

From corpus, we count the correlation matrix btw each word (cout each time if two words are in one sentence)

▼ The different between `CBOW` and `Skip-Gram`?

- CBOW predicts center words from context words
- Skip-gram does the inverse and predicts source context-words from the center words

But:

- the effect that CBOW smoothes over a lot of the distributional information (by treating an entire context as one observation). For the most part, this turns out to be a useful thing for smaller datasets.
- skip-gram treats each context-target pair as a new observation, and this tends to do better when we have larger datasets.

⇒ CBOW using for small dataset / Skip-Gram using for large dataset

▼ Useful techniques for Word Embedding?

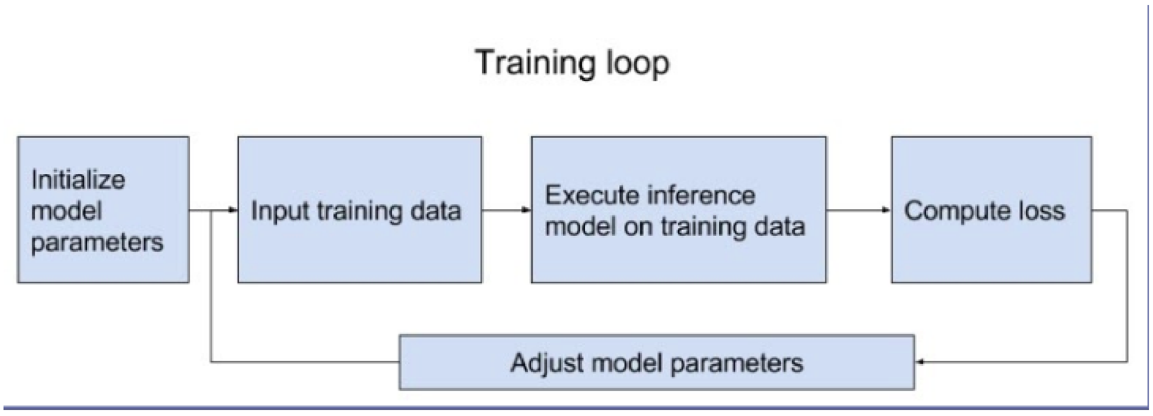
- Embedding Lookup
- NCE loss

Word2Vec in TensorFlow

▼ Phase 1: Assemble graph?

- Import data (with `tf.data` or `placeholder`
- Define the weights
- Define the inference model
- Define the loss function
- Define the optimization

▼ Phase 2: Compute?


Training loop
Initialize model parameters
Input training data
Execute inference model on training data
Compute loss
Adjust model parameters

▼ How to resuse model?

- Define the `class` (it see as OOP) for your model
- Set up model in a `collection`
- In case you really want to reuse the model without rebuilding it ⇒ save the `grap_def` in a file and then load it

Variable sharing

▼ What is name scope?

- TensorFlow doesn’t know what nodes should be grouped together, unless you tell it to

- Using `tf.name_scope(name)`
- `with  tf.name_scope("loss"):`

▼ What is the difference btw `name_scope` and `variable_scope`?

- Variable scope facilitates variable sharing
- Problem with sharing variable: We want all inputs us the same set of weights and bias

▼ The `tf.get_variable` has the "name" feature, why we just use this to reuse variable?

- It will raise ValueError, we need other method to reuse!
- `with tf.variable_scope('two_layers') as scope:`
  …
  `scope.reuse_variables()`
  …

Managing Experiments

▼ What the `tf.train.Saver` do?

- saves graph’s variables in binary files
- A good practice is to periodically save the model’s parameters after a certain number of steps so that we can restore/retrain our model from that step if need be. The tf.train.Saver() class allows us to do so by saving the graph’s variables in binary files.
- But! We actually need to save sess, and just params in maybe 10x epoch!
- The step you choose to save graph's variable is called checkpoint!
- We need to save `step` in `global_step` to easy keep track

▼ What the `tf.summary` do?

- To answer the question: Why matplotlib when you can summarize?
- Just need to create a scope_name and push all `scalar, histogram and image` to one scope
- Then run it like simple way! Using session (cause summary is just a ops)
- And you can save it in a file

- Finally, you can see all stuff on Tensorboard

Control Randomization

▼ Some informations about Control Randomization!

- Op level random seed, you can add `seed=x` in operations (each op keeps its own seed)
- The session can keep track of random state (Each new session restarts the random state)
- Graph level seed, using `tf.set_random_seed(2)`

Autodiff

▼ What are the gradients?

- So far, we build some models but do anything regard to gradients ⇒ the build-in of TF do it itselve
- TensorFlow builds the backward path for you ⇒ Path btw two tensors
- But: the TF have it own gradient functions to test and trying new things!

▼ Then we still need to learn gradients when everything have auto?

- To solve vanishing/explosing gradients ⇒ we need to cal gradients in effictively way!