Debugging Neural Networks for NLP

Graham Neubig

Carnegie Mellon University
Language Technologies Institute

Site
https://phontron.com/class/nn4nlp2017/
In Neural Networks, Tuning is Paramount!

- **Everything** is a hyperparameter
  - Network size/depth
  - Small model variations
  - Minibatch creation strategy
  - Optimizer/learning rate
- Models are complicated and opaque, debugging can be difficult!
Understanding Your Problem
A Typical Situation

- You’ve implemented a nice model
- You’ve looked at the code, and it looks OK
- Your accuracy on the test set is bad

What do I do?
Possible Causes

- **Training time problems**
  - Lack of model capacity
  - Inability to train model properly
  - Training time bug
- **Decoding time bugs**
  - Disconnect between test and decoding
  - Failure of search algorithm
- **Overfitting**
- **Mismatch between optimized function and eval**
Debugging at Training Time
Identifying Training Time Problems

• Look at the loss function calculated on the training set
  • Is the loss function going down?
  • Is it going down basically to zero if you run training long enough (e.g. 20-30 epochs)?
• If not, you have a training problem
Is My Model Too Weak?

• Your model needs to be big enough to learn

• Model size depends on task
  • For language modeling, at least 512 nodes
  • For natural language analysis, 128 or so may do

• Multiple layers are often better

• For long sequences (e.g. characters) may need larger layers
Be Careful of Deep Models

- Extra layers can help, but can also hurt if you’re not careful due to vanishing gradients

- Solutions:

  Residual Connections (He et al. 2015)  Highway Networks (Srivastava et al. 2015)

\[
y = H(x, W_H) \cdot T(x, W_T) + x \cdot (1 - T(x, W_T))
\]
Trouble w/ Optimization

• If increasing model size doesn’t help, you may have an optimization problem

• **Possible causes:**
  
  • Bad optimizer
  
  • Bad learning rate
  
  • Bad initialization
  
  • Bad minibatching strategy
Reminder: Optimizers

- **SGD**: take a step in the direction of the gradient
- **SGD with Momentum**: Remember gradients from past time steps to prevent sudden changes
- **Adagrad**: Adapt the learning rate to reduce learning rate for frequently updated parameters (as measured by the variance of the gradient)
- **Adam**: Like Adagrad, but keeps a running average of momentum and gradient variance
- **Many others**: RMSProp, Adadelta, etc.
  (See Ruder 2016 reference for more details)
Learning Rate

• Learning rate is an important parameter
  
  • Too low: will not learn or learn very slowly
  
  • Too high: will learn for a while, then fluctuate and diverge

• **Common strategy:** start from an initial learning rate then gradually decrease

• **Note:** need a different learning rate for each optimizer! (SGD default is 0.1, Adam 0.001)
Initialization

• Neural nets are sensitive to initialization, which results in different sized gradients

• Standard initialization methods:
  • **Gaussian initialization**: initialize with a zero-mean Gaussian distribution
  • **Uniform range initialization**: simply initialize uniformly within a range
  • **Glorot initialization, He initialization**: initialize in a uniform manner, where the range is specified according to net size
  • Latter is common/default, but read prior work carefully
Reminder:
Mini-batching in RNNs

Loss Calculation

- Take Sum

Padding

Mask

Example:
this is an example
this is another
Bucketing/Sorting

- If we use sentences of different lengths, too much padding and sorting can **result in slow training**
- To remedy this: **sort sentences** so similarly-lengthed sentences are in the same batch
- But this can affect performance! (Morishita et al. 2017)
Debugging at Decoding Time
Training/Decoding Disconnects

• Usually your loss calculation and decoding will be implemented in different functions

• e.g. `enc_dec.py` example from this class has `calc_loss()` and `generate()` functions

• Like all software engineering: duplicated code is a source of bugs!

• Also, usually loss calculation is minibatched, generation not.
Debugging Minibatching

• Debugging mini-batched loss calculation

  • Calculate loss with large batch size (e.g. 32)
  • Calculate loss for each sentence individually and sum them
  • The values should be the same (modulo numerical precision)

• Create a unit test that tests this!
Debugging Decoding

• Your decoding code should get the same score as loss calculation

• Test this:
  • Calculate loss of reference
  • Perform **forced decoding**, where you decode, but tell your model the reference word at each time step
  • The score of these two should be the same

• Create a unit test doing this!
Beam Search

- Instead of picking one high-probability word, maintain several paths

- Some in reading materials, more in a later class
Debugging Search

- As you make search better, the model score should get better (almost all the time)
- Run search with varying beam sizes and make sure you get a better overall model score with larger sizes
- Create a unit test testing this!
Battling Overfitting
Symptoms of Overfitting

• Training loss converges well, but test loss diverges

• No need to look at accuracy, only loss! Accuracy is a symptom of a different problem.
Your Neural Net can Memorize your Training Data
(Zhang et al. 2017)

• Your neural network has more parameters than training examples

• If you randomly shuffle the training labels (there is no correlation b/t input and labels), it can still learn
Optimizers: Adaptive Gradient Methods Tend to Overfit More
(Wilson et al. 2017)

- Adaptive gradient methods are fast, but have a stronger tendency to overfit on small data
Reminder: Early Stopping, Learning Rate Decay

• Neural nets have tons of parameters: we want to prevent them from over-fitting

• We can do this by monitoring our performance on held-out development data and stopping training when it starts to get worse

• It also sometimes helps to reduce the learning rate and continue training
Reminder: Dev-driven Learning Rate Decay

• Start w/ a high learning rate, then degrade learning rate when start overfitting the development set (the “newbob” learning rate schedule)

• Adam w/ Learning rate decay does relatively well for MT (Denkowski and Neubig 2017)
Reminder: Dropout
(Srivastava et al. 2014)

• Neural nets have lots of parameters, and are prone to overfitting

• Dropout: randomly zero-out nodes in the hidden layer with probability $p$ at **training time only**

• Because the number of nodes at training/test is different, scaling is necessary:
  
  • Standard dropout: scale by $p$ at test time
  
  • Inverted dropout: scale by $1/(1-p)$ at training time
Recurrent Dropout  
(Gal and Ghahramani 2015)

• Dropout can be applied to RNNs through recurrent/variational dropout

• Zero out particular nodes in the NN for the entire sentence
Mismatch b/t Optimized Function and Evaluation Metric
Loss Function, Evaluation Metric

• It is very common to optimize for maximum likelihood for training

• But even though likelihood is getting better, accuracy can get worse

• Remember: teacher forcing
A Stark Example  
(Koehn and Knowles 2017)

• Better search (=better model score) can result in worse BLEU score!

• Why? Shorter sentences have higher likelihood, better search finds them, but BLEU likes correct-length sentences.
Managing Loss Function/Eval Metric Differences

• Most principled way: use structured prediction techniques discussed previously
  • Structured max-margin training
  • Minimum risk training
  • Reinforcement learning
  • Reward augmented maximum likelihood
A Simple Method: Early Stopping w/ Eval Metric

• Remember this graph: difference between number of iterations for best loss vs. best eval

• Why?: Over-confident predictions hurt loss.
• Solution: perform early stopping based on accuracy
Reproducing Previous Work
Reproducing Previous Work

- Reproducing previous work is hard because everything is a hyper-parameter
- If code is released, find and reduce the differences one by one
- If code is not released, try your best
- Feel free to contact authors about details, they will usually respond!
Questions?