NATURAL LANGUAGE PROCESSING

LECTURE 5 : RNN, LM, AWD techniques

Contents

- RNN reviews
- Language model and Perplexity
- AWD-LSTM/RNNs techniques (Merity et al., ICLR 2017)
    - **A**veraged stochastic gradient (AvSGD) **W**eight-**D**ropped LSTM

RNN

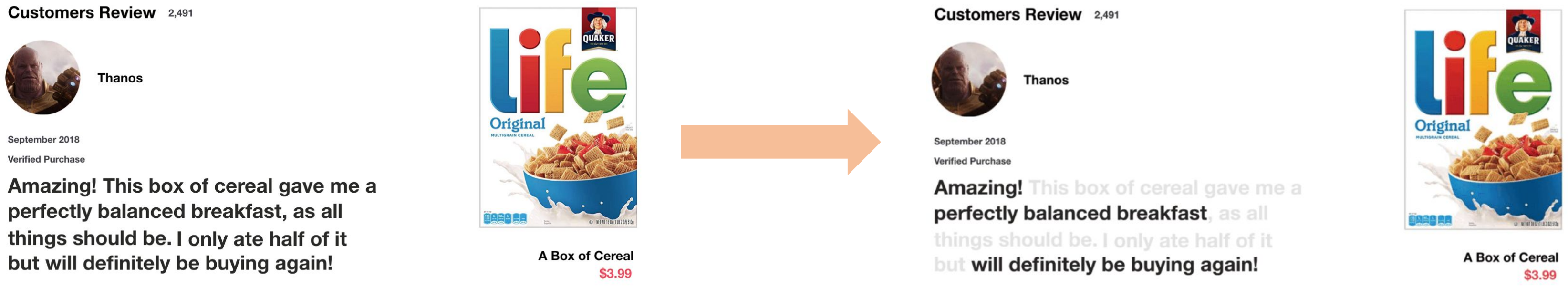
new weight = weight - learning rate*gradient

2.0999 = 2.1 - 0.001

Not much of a difference

update value

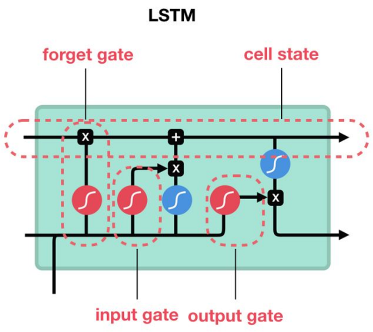
- RNN
  - RNN’s are good for processing sequence data for predictions but suffers from short-term memory (vanishing gradient problem).
  - LSTM’s and GRU’s were created as a method to mitigate short-term memory using mechanisms called gates. **Gates** are just neural networks that regulate the flow of information flowing through the sequence chain.

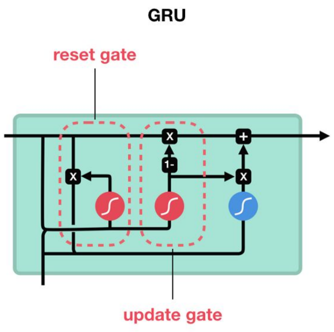


https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Reduce the effects of short-term memory in RNN

- LSTM
  - Cell state: a transport highway that transfers information all way down the sequence chain
  - Four different gates: forgot, input, output, gate
- GRU
  - Two gates: reset gate (r), update gate (z)
  - GRU’s has fewer tensor operations; therefore, they are a little speedier to train then LSTM’s. There isn’t a clear winner which one is better.





Language Model

- A **language model** is a statistical model or a probability distribution that assigns probabilities to words and sentences. Typically, we might be trying to guess the ***next word*** w in a sentence given all previous words.

- Examples
  - P(Today is Wednesday)=0.001
  - P(Today Wednesday is)=0.0000000001
  - P(pizza | For dinner I’m making) > P(cement | For dinner I’m making)

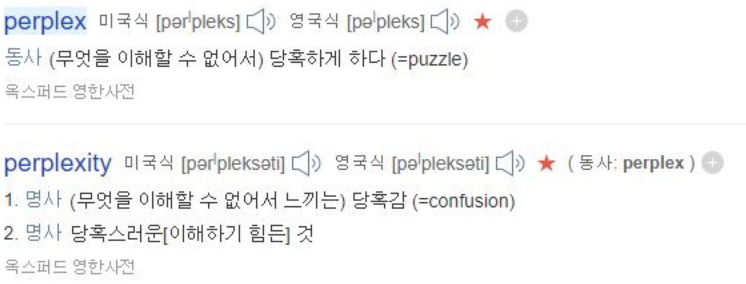
- https://lena-voita.github.io/nlp_course/language_modeling.html

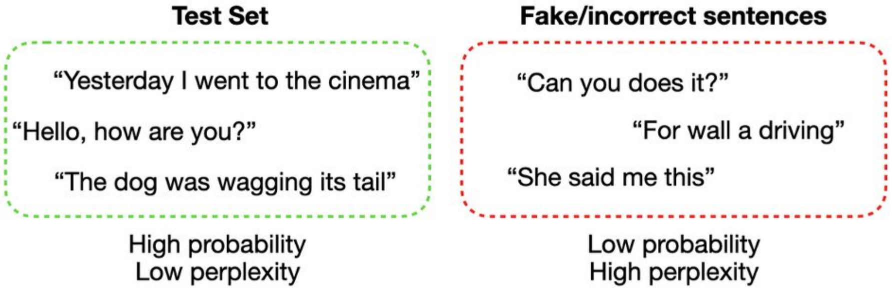
How to evaluate language model?

- Extrinsic evaluation
    - involves evaluating the models by employing them in an actual **task** (such as machine translation) and looking at their final loss/accuracy.
    - However, it can be computationally expensive and slow as it requires training a full system.
- Intrinsic evaluation
    - finding some metric to evaluate the language model itself, not taking into account the specific tasks it’s going to be used for.
    - a useful way of **quickly** comparing models
    - E.g., perplexity

Perplexity

- **Perplexity ↓** is a metric used to judge how good a language model is.
- Good models
  - assign high probabilities to sentences that are real and syntactically correct, and low probabilities to fake, incorrect, or highly infrequent sentences.
  - If a model assigns a high probability to the test set, it means that it is **not surprised** to see it (it's not *perplexed* by it), which means that it has a good understanding of how the language works.



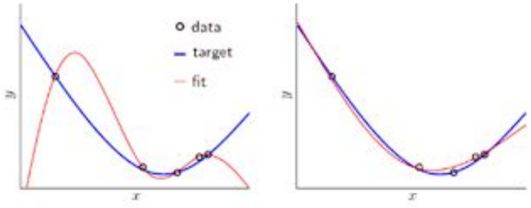


Problem of LSTM/RNN

- Recurrent Neural Networks and their variations are very likely to overfit the training data. This is due to the large network formed by unfolding each cell of the RNN, and relatively small number of parameters (since they are shared over each time step) and training data. Thus, the perplexities obtained on the test data are often quite larger than expected.

▸ How to minimize this overfitting problem on RNNs?

▸ A set of regularization strategies!



(a) without regularization (b) with regularization

Regularization strategies

- NT-ASGD
- DropConnect
- Variational dropout
- Embedding dropout
- Weight tying
- AR (Activation Regularization) and TAR (Temporal Activation Regularization)
- Variable backpropagation steps
- Independent sizes of word embeddings and hidden layer

Regularizing and Optimizing LSTM Language Models, Merity et al., ICLR 2017

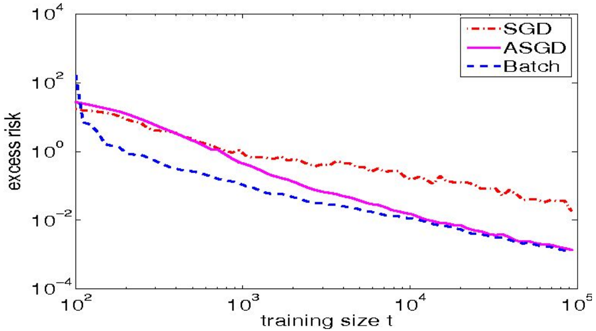
Averaged Stochastic Gradient Decent (ASGD)

- Take mean as the final solution
- Why?
  - The averaging is used to reduce the effect of noise. Namely, in practice, your gradient descent may get close to the optimal, but not really converge to it, instead oscillating around the optimal. In this case, averaging the results of stochastic gradient descent will give you a solution more likely to be closer to the optimum.

Ofir Nachum, I work on ML at Google Brain

Answered Dec 1, 2015 · Upvoted by Alexander Moreno, Machine Learning PhD, Georgia Tech and Alberto Bietti, PhD student in machine learning. Former ML engineer

The basic idea is to do regular stochastic gradient descent $w_{t+1} = w_t - \eta_t \nabla Q(w_t)$, but then take the mean $\overline{w} = \frac{1}{N} \sum_{t=1}^{N} w_t$ as the final solution.



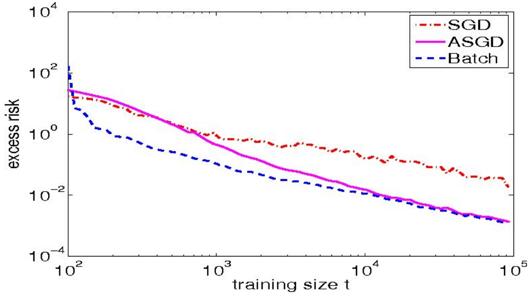
NT-ASGD

- Non-monotonically Triggered Averaged Stochastic Gradient Descent
    - Triggers the averaging **when the validation metric fails to improve for multiple cycles**
    - This conservatism ensures that the randomness of training does not play a major role in the decision

monotonic 미국식 영국식

1. 명사 단조로운, 변화 없는.
2. 명사 수학 단조로운. (1) 증가 또는 감소를 계속하는 함수 또는 함수의 값의 집합에 대하여 말한다. (2) 각 집...
3. 명사 음악 단[일]음의, 단[일]음으로 영창하는.

YBM 올인올 영한사전



Model	PTB Validation	PTB Test	WT2 Validation	WT2 Test
AWD-LSTM (tied)	60.0	57.3	68.6	65.8
– fine-tuning	60.7	58.8	69.1	66.0
– NT-AvSGD	66.3	63.7	73.3	69.7
– variable sequence lengths	61.3	58.9	69.3	66.2
– embedding dropout	65.1	62.7	71.1	68.1
– weight decay	63.7	61.0	71.9	68.7
– AR/TAR	62.7	60.3	73.2	70.1
– full sized embedding	68.0	65.6	73.7	70.7
– weight-dropping	71.1	68.9	78.4	74.9

Table 5: Model ablations for our best LSTM models reporting results over the validation and test set on Penn Treebank and WikiText-2. Ablations are split into optimization and regularization variants, sorted according to the achieved validation perplexity on WikiText-2.

Regularizing and Optimizing LSTM Language Models, Merity et al., ICLR 2017

Dropout

- Dropout is applied to each time step, however, mask for each time step is different
- RNN, LSTM, GRU in Pytorch includes dropout as parameter, following Bernoulli random variable

RNN

CLASS `torch.nn.RNN(*args, **kwargs)` [SOURCE]

Applies a multi-layer Elman RNN with $\tanh$ or ReLU non-linearity to an input sequence.

For each element in the input sequence, each layer computes the following function:

$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh})$$

where h_t is the hidden state at time *t*, x_t is the input at time *t*, and $h_{(t-1)}$ is the hidden state of the previous layer at time *t-1* or the initial hidden state at time *0*. If `nonlinearity` is `'relu'`, then ReLU is used instead of $\tanh$.

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two RNNs together to form a *stacked RNN*, with the second RNN taking in outputs of the first RNN and computing the final results. Default: 1
- **nonlinearity** – The non-linearity to use. Can be either `'tanh'` or `'relu'`. Default: `'tanh'`
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (*batch, seq, feature*) instead of (*seq, batch, feature*). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each RNN layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional RNN. Default: `False`

LSTM

CLASS `torch.nn.LSTM(*args, **kwargs)` [SOURCE]

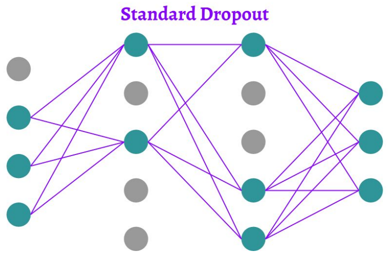
Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.

For each element in the input sequence, each layer computes the following function:

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two LSTMs together to form a *stacked LSTM*, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (*batch, seq, feature*) instead of (*seq, batch, feature*). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional LSTM. Default: `False`
- **proj_size** – If `> 0`, will use LSTM with projections of corresponding size. Default: 0

Dropout methods on LSTM

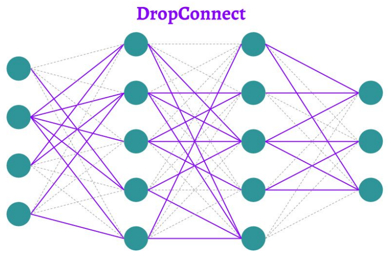


Training Phase :

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}) \circ \mathbf{m}, \quad m_i \sim Bernoulli(p)$$

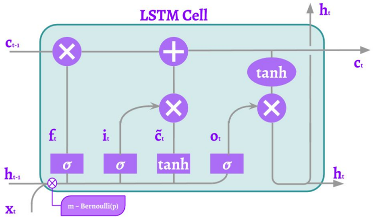
Testing Phase :

$$\mathbf{y} = (1-p)f(\mathbf{W}\mathbf{x})$$



Training Phase :

$$\mathbf{y} = f((\mathbf{W} \circ \mathbf{M})\mathbf{x}), \quad M_{i,j} \sim Bernoulli(p)$$



Training Phase :

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i\left(\begin{bmatrix}\mathbf{x}_t \\ \mathbf{h}_{t-1}\end{bmatrix} \circ \mathbf{m}\right)\right) \qquad \mathbf{o}_t = \sigma\left(\mathbf{W}_o\left(\begin{bmatrix}\mathbf{x}_t \\ \mathbf{h}_{t-1}\end{bmatrix} \circ \mathbf{m}\right)\right)$$

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f\left(\begin{bmatrix}\mathbf{x}_t \\ \mathbf{h}_{t-1}\end{bmatrix} \circ \mathbf{m}\right)\right) \qquad \tilde{\mathbf{c}}_t = tanh\left(\mathbf{W}_{\tilde{c}}\left(\begin{bmatrix}\mathbf{x}_t \\ \mathbf{h}_{t-1}\end{bmatrix} \circ \mathbf{m}\right)\right)$$

$$m_i \sim Bernoulli(p)$$

- DropConnect
  - The dropout mask for each **weight** is preserved and the same mask is used across all time steps, thereby adding negligible computation overhead.
- Variational dropout
  - Each **example** within the minibatch uses a unique dropout mask, rather than a single dropout mask being used over all examples.
- *Embedding dropout*
  - Dropout on the embedding matrix at a **word** level, which results in new word vectors which are identically zero for the dropped words. The remaining word vectors are scaled by as compensation.

Regularizing and Optimizing LSTM Language Models, Merity et al., ICLR 2017

Reference

- Language modeling and tricks: https://lena-voita.github.io/nlp_course/language_modeling.html
- Perplexity: https://towardsdatascience.com/perplexity-in-language-models-87a196019a94
- Dropout methods: https://towardsdatascience.com/12-main-dropout-methods-mathematical-and-visual-explanation-58cdc2112293
- LSTM, GRU: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
- AWD-LSTM
  - https://openreview.net/forum?id=SyyGPP0TZ
  - https://ys1998.github.io/blog/2018/01/merity