DS-GA 1008: Deep Learning, Spring 2019
Homework Assignment 3

Lekha Iyengar

March 2019

1.1 - Dropout

a. List the torch.nn module corresponding to 2D dropout.

Dropout2d is the torch.nn module corresponding to dropout. Its parameters are the dropout probability 'p' and 'inplace'. p is the probability of an element to be zero-ed. If inplace is set to True, the dropout operation is done inplace.

b. Read on what dropout is and give a short explanation on what it does and why it is useful.

Dropout is a regularization technique which is used to prevent overfitting. The technique involves temporarily dropping nodes (hidden and input) along with all its incoming and outgoing connections. A node at training time is present with probability p (dropped with probability 1-p). For the input nodes, however, the optimal probability of retention is usually closer to 1 At test time, the node is always present and the weights are multiplied by p.
During training, dropout uses several reduced networks. This prevents the nodes from co-adapting too much. This model combination also improves the performance of the network. "Although dropout alone gives significant improvements, using dropout along with max-norm regularization, large decaying learning rates and high momentum provides a significant boost over just using dropout. A possible justification is that constraining weight vectors to lie inside a ball of fixed radius makes it possible to use a huge learning rate without the possibility of weights blowing up. As the learning rate decays, the optimization takes shorter steps, thereby doing less exploration and eventually settles into a minimum." [1]

1.2 - Batch Norm

a. What does mini-batch refer to in the context of deep learning?

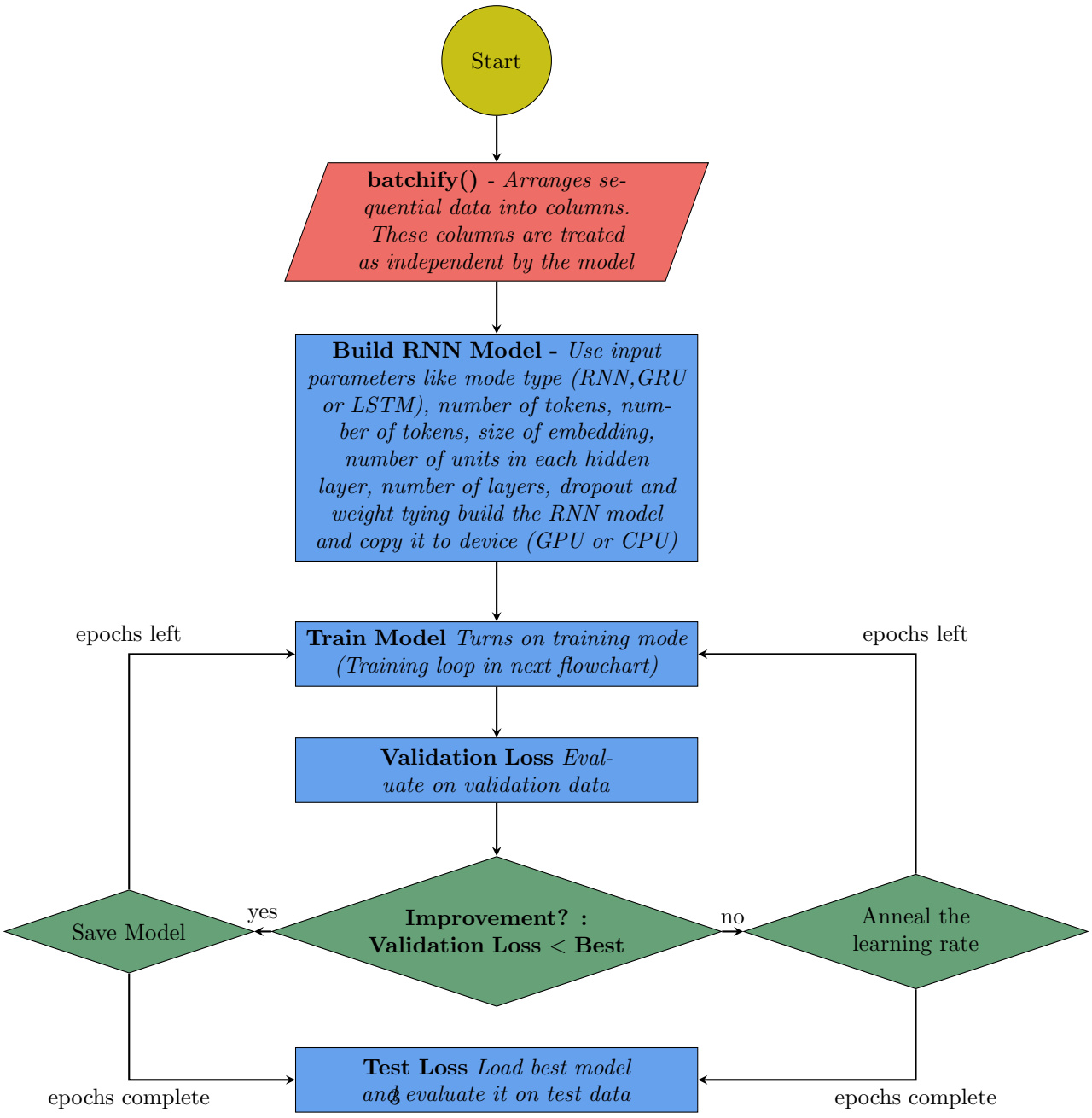
Usually the total amount of training data available is quite large. It is computationally expensive to use the entire batch at each step to train the algorithm. We prefer to use a small number of training instances which constitute a mini-batch in one iteration. Training over a mini-batch constitutes an iterator and training over all the mini batches constitutes an epoch.

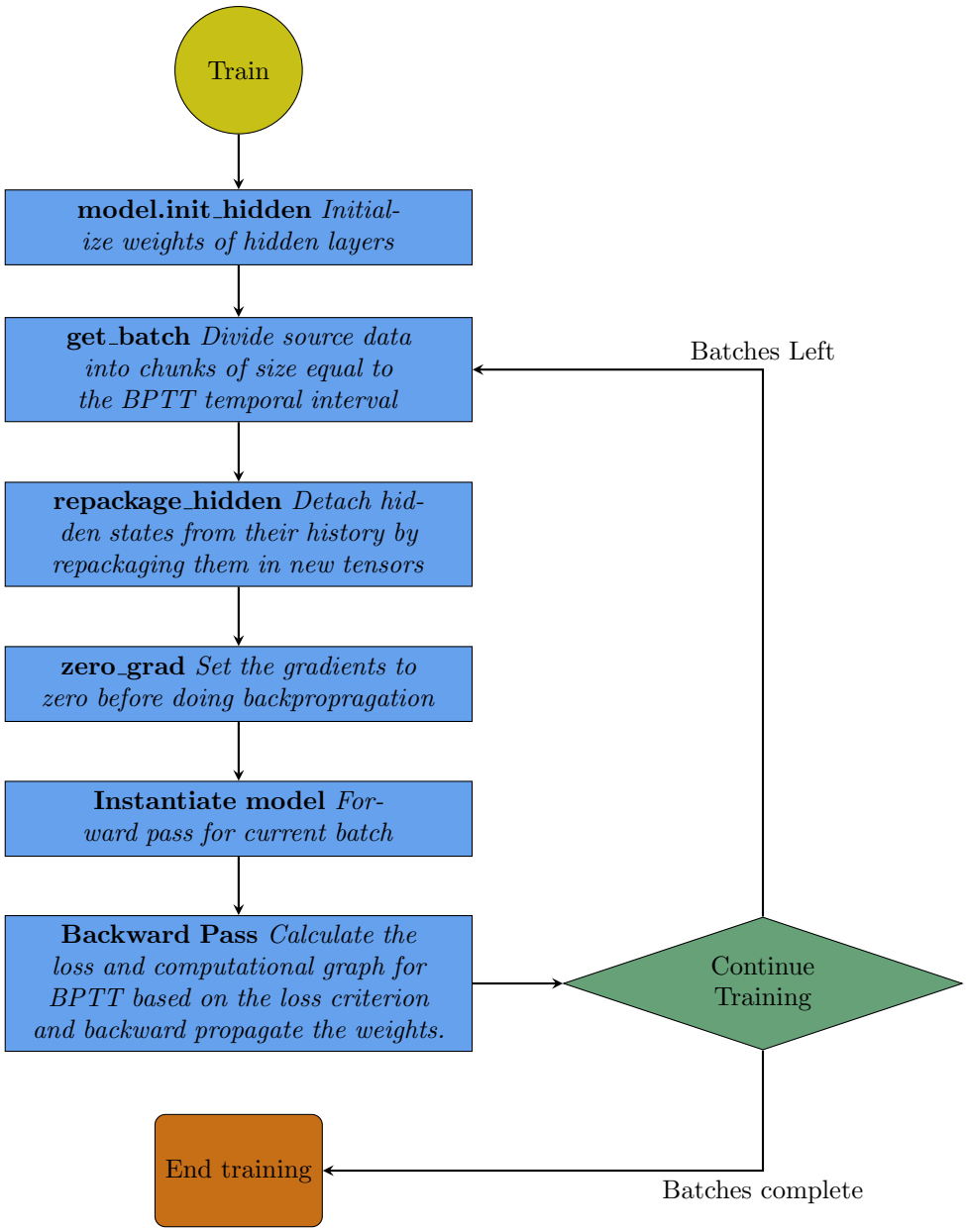
b. Read on what batch norm is and give a short explanation on what it does and why it is useful.

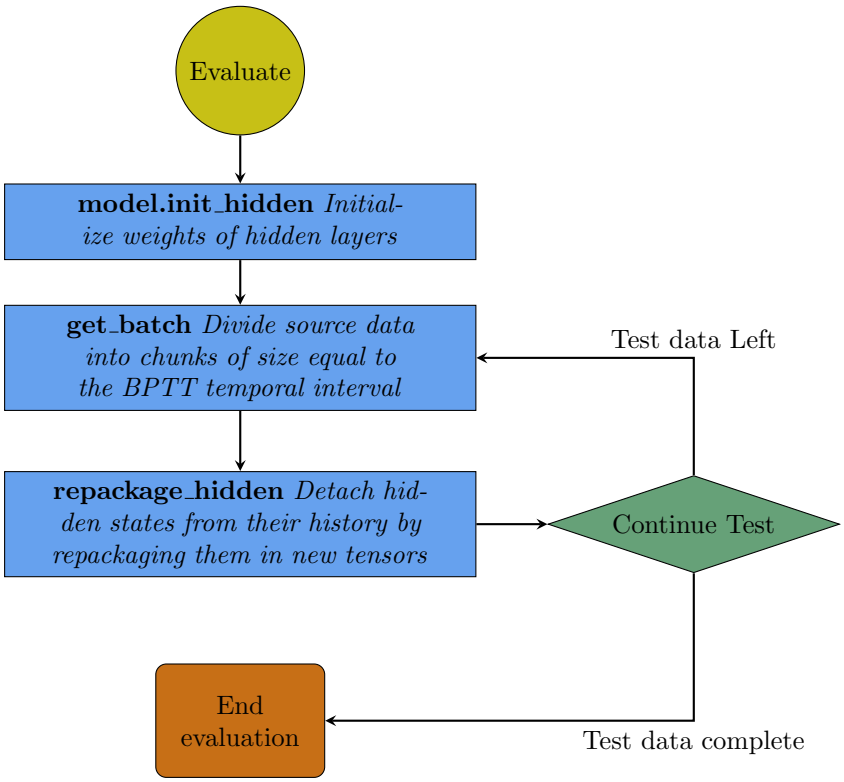
The distribution of each layer's inputs changes during training, as the parameters of the previous layers change. To avoid this problem we need to use smaller learning rates and carefully initialize parameters. This slows down the training. Networks converge faster if the inputs have been whitened (zero mean, unit variances) and are uncorrelated [2] [3]. Batch normalization reduces the dependence of gradients on the scale of the parameters or on their initial values (reduces internal covariate shift) by fixing the means and variances of layer inputs. This allows us to use much higher learning rates without the risk of divergence. It also allows each layer to learn independently of other layers.It also acts as a regularizer, in some cases eliminating the need for Dropout. It normalizes the output of previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. The gradient descent algorithm does the denormalization if required (to minimize loss function) by changing only these two weights for each activation, instead of losing the stability of the network by changing all the weights.

2 - Language Modelling

a. Go through the code and draw a block diagram / flow chart which highlights the main interacting components, illustrates their functionality, and provides an estimate of their computational time percentage




Train
model.init_hidden *Initialize weights of hidden layers*
get_batch *Divide source data into chunks of size equal to the BPTT temporal interval*
repackage_hidden *Detach hidden states from their history by repackaging them in new tensors*
zero_grad *Set the gradients to zero before doing backpropragation*
Instantiate model *Forward pass for current batch*
Backward Pass *Calculate the loss and computational graph for BPTT based on the loss criterion and backward propagate the weights.*
Continue Training
Batches Left
End training
Batches complete



Function	Time Percentage
batchify	1.2
get_batch	0.27
test set evaluation	0.68
validation time	4.76
training loop time	91.32

b. Find and explain where and how the back-propagation through time (BPTT) takes place

Backpropagation through time takes place in loss.backward()which is called in the training loop.

Backpropagation Through Time, or BPTT, is the training algorithm used to update weights in recurrent neural networks like LSTMs. BPTT works by unrolling all input time-steps. Each time-step has one input time-step, one copy of the network, and one output. Errors are then calculated and accumulated for each time-step. The network is rolled back up and the weights are updated. Spatially, each time-step of the unrolled recurrent neural network may be seen as an additional layer given the order dependence of the problem and the internal state from the previous time-step is taken as an input on the subsequent

time-step.(Introduction to BPTT)
BPTT comprises of following steps:
1.Present a sequence of timesteps of input and output pairs to the network.
2.Unroll the network then calculate and accumulate errors across each timestep.
3.Roll-up the network and update weights.
4.Repeat

c. Describe why we need the repackage_hidden function, and how it works.

The repackage_hidden() function wraps hidden states in new variables, to detach them from their history. If we don't detach the hidden state, the model would try backpropagating all the way to start of the dataset. "Every variable has a .creator attribute that is an entry point to a graph, that encodes the operation history. This allows autograd to replay it and differentiate each operation. Each hidden state will have a reference to some graph node that has created it, but in BPTT, we don't want to backprop to it after finishing the sequence. To get rid of the reference, we have to take out the tensor containing the hidden state and wrap it in a fresh Variable, that has no history. This allows the previous graph to go out of scope and free up the memory for next iteration." (https://discuss.pytorch.org/repackage-hidden-in-word-language-model)

d. Why is there a –tied (tie the word embedding and softmax weights) option?

Tying the word embedding and the softmax weights, greatly reducing the number of trainable variables."[4]"By tying the two embeddings together, the joint embedding evolves in a more similar way to the output embedding than to the input embedding of the untied model. Tying the input and output embeddings leads to an improvement in the perplexity of various language models.Weight tying in neural translation models can reduce their size (number of parameters) to less than half of their original size without harming their performance. Weight tying significantly reduces perplexity on both the validation set and the test set, but not on the training set. This indicates less overfitting, as expected due to the reduction in the number of parameters."[5]

e. Compare LSTM and GRU performance (validation perplexity and training time) for different values of the following parameters: number of epochs, number of layers, hidden units size, input embedding dimensionality, BPTT temporal interval, and non-linearities (pick just 3 of these parameters and experiment with 2-3 different values for each).

Experimenting with number of epochs:

Type	Epochs	Validation perplexity	Training time
LSTM	10	117.35	0:08:12
LSTM	20	109.59	0:16:07
LSTM	40	109.57	0:41:57
GRU	10	167.5	0:08:06
GRU	20	148.69	0:15:53
GRU	40	134.87	0:31:18

LSTM performs better than GRU in this experiment. The perplexity decreases as number of epochs increase.

Experimenting with number of layers:

Type	Nlayers	Validation perplexity	Training time
LSTM	10	910.63	1:33:51
LSTM	20	109.57	0:41:57
LSTM	30	915.88	4:25:14
LSTM	40	915.15	6:41:42
GRU	10	927.35	1:31:38
GRU	20	134.87	0:31:18
GRU	30	914.85	4:16:40
GRU	40	914.17	5:46:28

Perplexity and training time increases if number of layers is decreased to 10 or increased to 30 and 40. Best performance is observed for 20 layers for both LSTM and GRU. LSTM performs better than GRU but the training time is longer.

Experimenting with input embedding dimensionality:

Type	Embedding size	Validation perplexity	Training time
LSTM	100	171.09	0:30:39
LSTM	200	109.57	0:41:57
LSTM	400	108.06	0:34:53
LSTM	650	109.85	0:30:45
GRU	100	109.51	0:30:59
GRU	200	134.87	0:31:18
GRU	400	127.8	0:28:49
GRU	650	205.45	0:30:42

The training time does not vary much as the size of input embeddings is varied. The performance of GRU deteriorates as the embedding dimensionality is increased. The performance of LSTM improves

f.Why do we compute performance on a test set as well? What is this number good for?

Perplexity is the inverse probability of the test set, normalized by the number of words. Minimizing perplexity is the same as maximizing probability. Intrinsic evaluation metrics like perplexity are a bad approximation of performance of language model if the test set doesn't look like the training set. Computing the performance on test set helps us determine if the training and test set are similar and also validate if the model generalizes well. The validation set is used to estimate prediction error and the test set is used to assess the performance of a fully trained model.

Links

- Read only link to Overleaf Project

References

[1] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal oDropout: A Simple Way to Prevent Neural Networks from Overfittingf Machine Learning Research*, 15(1):1929–1958, 2014.

[2] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998. Springer-Verlag.

[3] Simon Wiesler and Hermann Ney. A convergence analysis of log-linear training. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 657–665. Curran Associates, Inc., 2011.

[4] Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *CoRR*, abs/1611.01462, 2016.

[5] Ofir Press and Lior Wolf. Using the output embedding to improve language models. *CoRR*, abs/1608.05859, 2016.