Training a Subsampling Mechanism in Expectation

Colin Raffel & Dieterich Lawson[*]
Google Brain
craffel@gmail.com, dieterichl@google.com

Abstract

We describe a mechanism for subsampling sequences and show how to compute its expected output so that it can be trained with standard backpropagation. We test this approach on a simple toy problem and discuss its shortcomings.

1 Subsampling Sequences

Consider a mechanism which, given a sequence of vectors $\mathbf{s} = \{s_0, s_1, \dots, s_{T-1}\}, s_t \in \mathbb{R}^d$, produces a sequence of "sampling probabilities" $\mathbf{e} = \{e_0, e_1, \dots, e_{T-1}\}, e_t \in [0, 1]$ which denote the probability of including s_t in the output sequence $\mathbf{y} = \{y_0, y_1, \dots, y_{U-1}\}$. Producing $\mathbf{y}$ from $\mathbf{s}$ and $\mathbf{e}$ is encapsulated by the following pseudo-code and visualized in fig. 2 (appendix):

```
# Initialize y as an empty sequence
y = []
for t in {0, 1, ..., T - 1}:
    # Draw a random number in [0, 1] and compare to e[t]
    if rand() < e[t]:
        # Add s[t] to y with probability e[t]
        y.append(s[t])
```

We call this a "subsampling mechanism", because by construction, $U \leq T$, and each element of $\mathbf{y}$ is drawn directly from $\mathbf{s}$. The ability to subsample a sequence has various applications:

- When the input sequence $\mathbf{s}$ is oversampled (i.e. each element s_t contains much the same information as s_{t-1}), subsampling can be an effective way of shortening the sequence without discarding useful information. Using a shorter sequence can facilitate the use of recurrent network models, which have difficulties with long-term dependencies (Bengio et al., 1994; Hochreiter & Schmidhuber, 1997). Simple subsampling schemes such as choosing every other element of $\mathbf{s}$ have proven effective in tasks such as speech recognition (Chan et al., 2015).
- The mechanism can be used in sequence transduction tasks where the output sequence is shorter than the input. We contrast this approach with the commonly used Connectionist Temporal Classification loss (Graves et al., 2006) because subsampling actually shortens the sequence (instead of inserting blanks) and can be inserted arbitrarily into a neural network model (instead of specifically being a loss function). It also implicitly produces a monotonic alignment between elements in $\mathbf{s}$ and $\mathbf{y}$; such alignments have proven to be useful (Bahdanau et al., 2014).
- Applying this subsampling operation multiple times could build a hierarchy of shorter and shorter sequences which capture structure at different scales. A similar approach was recently shown to be effective in langauge modeling tasks (Chung et al., 2016).

Motivated by these applications, in this extended abstract we present a method for training this subsampling mechanism in expectation, i.e. without sampling. We then test this approach on a simple toy problem and study the resulting model's behavior. Finally, we discuss shortcomings of our approach and possibilities for future work.

[*] Work done as members of the Google Brain Residency program

2 Training in Expectation

We are interested in including the mechanism defined in the previous section in the midst of a neural network model. However, the sampling process used to construct $\mathbf{y}$ precludes the use of standard backpropagation. A common approach to this issue is to optimize the model according to the expected (or mean-field) output (Graves, 2016; Bahdanau et al., 2014). The following analysis shows how to employ this approach to our proposed subsampling mechanism using a dynamic program which analytically computes $p(y_m = s_n)$.

First, observe that $p(y_0 = s_0) = e_0$, i.e. the probability that the first output is the first entry in the sequence is just the probability of sampling at time 0. Next, in order for $y_0 = s_1$, we need $y_0 \neq s_0$ so $p(y_0 = s_1)$ is the probability that s_0 was not sampled at time 0 and that s_1 was, giving $p(y_0 = s_1) = e_1(1 - e_0)$. Continuing on in this way, we see that $p(y_0 = s_n) = e_n \prod_{i=0}^{n-1}(1 - e_i)$ or, in words, the probability that the first output element y_0 is a given element in the sequence s_n is the probability that none of $s_0, \dots, s_{n-1}$ were sampled multiplied by the probability of sampling s_n. Second, observe that $p(y_m = s_n) = 0$ when $n < m$ because in order for the output sequence to be of length m, at least $m - 1$ symbols must already have been sampled. If $n < m$, this relation is violated. Finally, in order for $y_m = s_n$ in general, we must have that $y_{m-1} = s_j \in s_0, \dots, s_{n-1}$ (i.e. the previous output must be one of the states before s_n), none of $s_{j+1}, \dots, s_{n-1}$ may be sampled at time m, and s_n is sampled at time m. To compute $p(y_m = s_n)$, we need to sum over all of the the possible cases $y_{m-1} \in \{s_0, \dots, s_{n-1}\}$. The probability of a single case is the combined probability that s_n is sampled, that $y_{m-1} = s_j$, and that none of $s_{j+1}, \dots, s_{n-1}$ are sampled at time m. We visualize these possibilities in fig. 3 (appendix). Summing over the possible j yields

$$p(y_m = s_n) = e_n \sum_{j=0}^{n-1} \left(p(y_{m-1} = s_j) \prod_{i=j+1}^{n-1} (1 - e_i) \right) \tag{1}$$

where for convenience we define the special case $\prod_{i=n}^{m} \bullet = 1$ when $n > m$. Once we compute $p(y_m = s_n)$, it is straightforward to find the expected value of y_m simply by computing $\sum_n s_n p(y_m = s_n)$. Note $p(y_m = s_n) = e_n((1-e_{n-1})p(y_m = s_{n-1})/e_{n-1} + p(y_{m-1} = s_{n-1}))$; it follows that each term $p(y_m = s_n)$ can be computed in $\mathcal{O}(1)$ time by reusing the already-computed terms $p(y_m = s_{n-1})$ and $p(y_{m-1} = s_{n-1})$. The resulting dynamic program allows all the terms $p(y_m = s_n)$ to be computed in $\mathcal{O}(T^2)$ time.

Note that $\sum_n p(y_m = s_n) \leq 1$ depending on the values of $\mathbf{e}$, so these probabilities may not form a valid probability distribution. Computing the expectation as-is without further normalization effectively associates any additional probability to an implicit zero vector in $\mathbb{R}^d$, which is the convention we will use for the remainder of this extended abstract.

3 Toy Problem Experiment

To evaluate the feasibility of this approach, we tested it on the following toy problem: Consider a length-T sequence $\mathbf{x}$ of symbols $[0, 1, 2]$ which occur with equal probability. The output is produced as follows for $t \in \{0, \dots, T - 1\}$, beginning with an empty memory:

1. If x_t is 0, don't output anything and maintain the current memory state.
2. If x_t is 1 or 2 and our memory is empty, place x_t in memory and don't output anything.
3. If x_t is 1 and we have 1 in our memory, output a 0 and empty the memory.
4. If x_t is 2 and we have 2 in our memory, output a 0 and empty the memory.
5. If x_t is 1 and we have 2 in our memory, output a 2 and empty the memory.
6. If x_t is 2 and we have 1 in our memory, output a 1 and empty the memory.

We also define special cases where if $T = 1$, the output is x_0; if $x_t = 0 \; \forall \; t \in \{0, \dots, T - 1\}$, the output is 0; and if all entries of x_t are 0 except one, the output is the single nonzero entry. An example input-output pair for this toy problem is shown in fig. 4 (appendix).

We utilized the following model:

$$s_t = \mathrm{LSTM}(x_t, s_{t-1}) \tag{2}$$

$$e_t = \sigma(W_{he}^\top s_{t-1} + W_{xe}^\top x_{t-1} + b_e) \tag{3}$$

$$y_t = \mathrm{softmax}\left(W_y^\top \sum_{n=0}^{T-1} p(y_t = s_n) s_n + b_y\right) \tag{4}$$

where $x_t \in \mathbb{R}^3$ is the one-hot encoding of the input sequence, LSTM is a long short-term memory RNN (Hochreiter & Schmidhuber, 1997) with state dimensionality 100, $W_{he} \in \mathbb{R}^{100\times 1}, W_{xe} \in \mathbb{R}^{3\times 1}, b_e \in \mathbb{R}$ are the weight matrices and bias scalar for computing emission probabilities, $\sigma(\cdot)$ is the logistic sigmoid function, and $W_y \in \mathbb{R}^{100\times 3}, b_y \in \mathbb{R}^3$ are the weight matrix and bias vector of the output softmax function. The $p(y_t = s_n)$ terms are computed as described in section 2.

We fed minibatches of 100 sequences of randomly chosen $[0, 1, 2]$ values, encoded as one-hot vectors, to the network. The network was trained with categorical cross-entropy against analytically computed targets using Adam with the learning hyperparameters suggested in (Kingma & Ba, 2015). We computed the network's accuracy on a separately generated test set that it was not trained on. As proposed in (Zaremba & Sutskever, 2014), we found it beneficial to use a simple curriculum learning (Bengio et al., 2009) strategy where the loss was only computed for the first T' elements of the output sequence, where T' was uniformly sampled from the values $\{1, 2, \ldots, T\}$ for each minibatch.

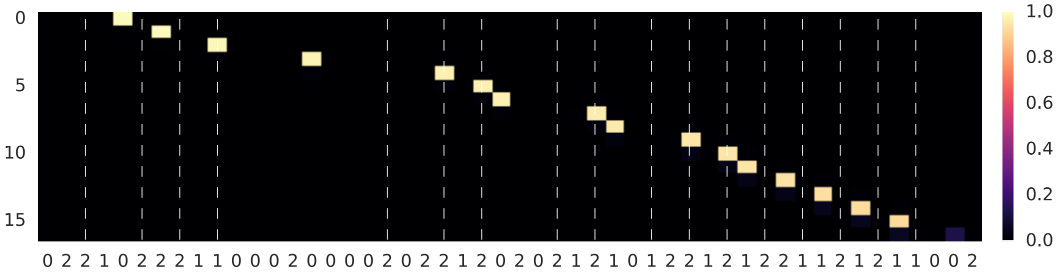


Figure 1: $p(y_t = s_n)$ for an example test sequence of length 50. The sequence is shown on the x-axis, with dashed vertical lines denoting where we might expect the model to emit symbols. The y-axis shows the output sequence index. For reference, the correct output for this sequence is $\{0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 2, 1, 1, 0\}$.

For all values of T we tried (up to $T = 500$), the network was able to achieve $> 98\%$ accuracy on the held-out test set after training for a modest number of minibatches (around 10,000). To get a picture of the qualitative behavior of the model, we plot the matrix $p(y_t = s_n)$ for an example test sequence with $T = 50$ in fig. 1. Note that emissions do not occur exactly when the model has seen sufficient input to produce them, i.e. once it sees a second nonzero input. In this particular case, this caused the model to emit one too few symbols. To facilitate further research, we provide a TensorFlow implementation of our approach.[1]

While we have shown that our model can quickly learn the desired behavior on a toy problem, we had issues applying this approach to real-world problems, which we attribute primarily to two factors: First, while a stated goal of the subsampling mechanism is to produce shorter sequences, the $\mathcal{O}(T^2)$ complexity of computing the terms $p(y_m = s_n)$ precludes its practical use on problems with large T. Second, the use of a sigmoid in eq. (4) and the cumulative product in eq. (1) can result in vanishing gradients in practice. The first issue could be mitigated by greedy approximations to the procedure outlined in section 2, for example by selecting which items in s_t are chosen using discrete latent variables and training with reinforcement learning methods as has been done in recent work (Luo et al., 2016). We hope the encouraging results and analysis presented here inspires future work on utilizing learnable subsampling mechanisms in neural networks.

[1] `https://github.com/craffel/subsampling_in_expectation`

REFERENCES

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning*, pp. 41–48, 2009.

William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. Listen, attend and spell. *arXiv preprint arXiv:1508.01211*, 2015.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine learning*, pp. 369–376, 2006.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

Yuping Luo, Chung-Cheng Chiu, Navdeep Jaitly, and Ilya Sutskever. Learning online alignments with continuous rewards policy gradient. *arXiv preprint arXiv:1608.01281*, 2016.

Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

A FIGURES

In this appendix we provide additional figures to help illustrate some of the concepts presented in this extended abstract.

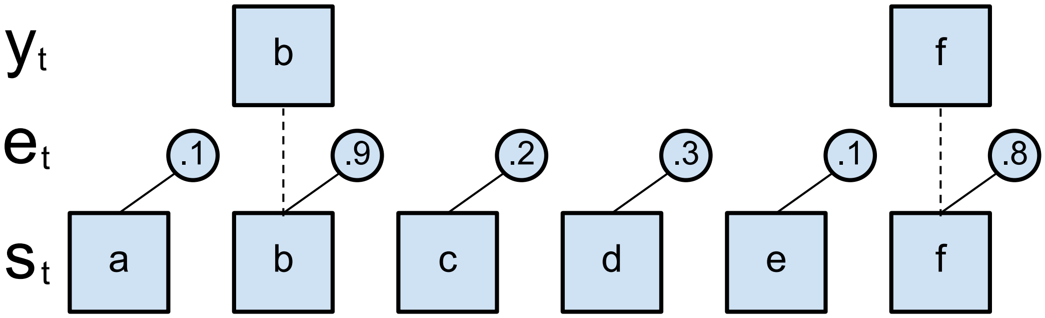


Figure 2: Illustration of the subsampling process described in section 1. Each element s_t of $\mathbf{s}$ is included in the output sequence $\mathbf{y}$ with probability e_t. In this case, the second and final elements of $\mathbf{s}$ were sampled.

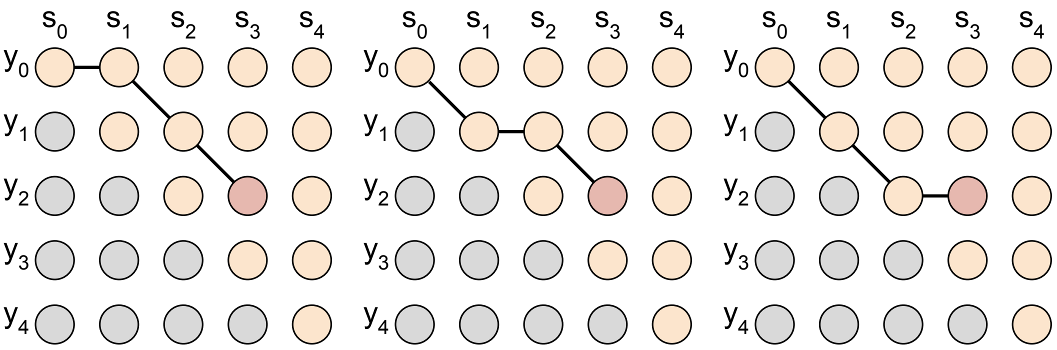
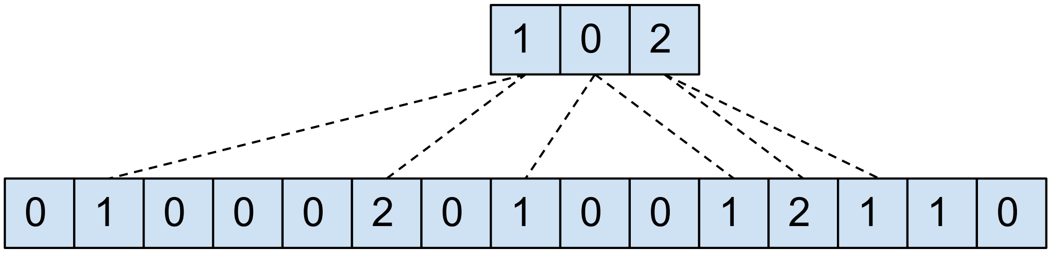


Figure 3: Possible ways that $y_2 = s_3$: either $y_0 = s_1, y_1 = s_2, y_2 = s_3$ or $y_0 = s_0, y_1 = s_1, y_2 = s_3$ or $y_0 = s_0, y_1 = s_1, y_2 = s_3$. Equation (1) sums over these possibilities to compute $p(y_2 = s_3)$. Gray and yellow nodes nodes indicate invalid and valid input-output pairings respectively.



1 0 2

0 1 0 0 0 2 0 1 0 0 1 2 1 1 0

Figure 4: Example input (bottom) and output (top) sequence for the toy problem described in section 3. Dashed lines indicate which values in the input sequence cause each output element.