A STATE-SPACE METHOD FOR LANGUAGE MODELING

Vesa Siivola and Antti Honkela

Helsinki University of Technology
Neural Networks Research Centre
Vesa.Siivola@hut.fi, Antti.Honkela@hut.fi

ABSTRACT

In this paper, a new state-space method for language modeling is presented. The complexity of the model is controlled by choosing the dimension of the state instead of the smoothing and back-off methods common in n-gram modeling. The model complexity also controls the generalization ability of the model, allowing the model to handle similar words in similar manner. We compare the state-space model to a traditional n-gram model in a task of letter prediction. In this proof-of-concept experiment, the state-space model gives similar results as the n-gram model with sparse training data, but performs clearly worse with dense training data. While the initial results are encouraging, the training algorithm should be made more effective, so that it can fully exploit the model structure and scale up to larger token sets, such as words.

1. INTRODUCTION

A language model is an important component of a modern speech recognition system. The language model ranks the hypotheses generated by the acoustic models. Usually, a hypothesis is expanded one word at a time, so the language model gives the probability of the new word given the known history. The overwhelmingly most common approach is n-gram modeling. The n-gram model assigns the probabilities based on the relative frequencies of the words with same truncated histories in the training set. With heuristics like smoothing and back-off, the n-gram model provides a robust model [1]. Corresponding models based on Bayesian probability theory give similar results [2].

The main drawback of n-gram models is that the model cannot generalize from semantically similar words. If the training data has a sentence "Monday morning was clear", the n-gram model cannot use any of that information to model the sentence "Tuesday evening is cloudy". If similar words are clustered and the n-gram estimates are based on these clusters, this kind of generalization can be achieved [3, 4]. Combining cluster n-grams and traditional n-grams improves the model.

In Neural Probabilistic Language Model (NPLM) [5], this generalization is achieved differently. A word is mapped to a low dimensional feature vector by a neural net. The feature vectors for fixed number of previous words are fed into the second layer of the network, which maps these vectors to probabilities through the softmax function. Since the network has too few parameters to learn the probability distribution separately for all feature vectors, the first layer of the network ends up mapping similar words close to each other. As the mapping of the second layer is smooth, this leads to good generalization ability. The method is computationally demanding and the authors reduced the size of the vocabulary to less than 20 000. The method achieved approximately 15% lower perplexity than a corresponding n-gram model.

The model presented here is based on state-space methods. A traditional state-space model is a model for continuous valued time series data. It consists of a linear dynamical system describing the evolution of the state. The state is not observed directly but through a separate linear observation mapping. State-space models are very popular in many applications due to their general nature and also because of simple processing algorithms such as Kalman filtering [6].

In this paper, we present a novel state-space method for modeling a discrete token source, such as words of a language. The probability distribution of a token is governed by the softmax function of a linear transformation of the corresponding state. The new state depends on a fixed number of previous tokens in addition to the previous state. Each of the previous tokens is projected to a low dimensional feature vector and the features form a part of the state vector. Because of this dimension reduction, increasing the number of tokens affecting the current state increases the model complexity only moderately. Since the model does not have enough parameters to learn the probability distribution separately for each possible feature vector, similar tokens are mapped close to each other, just like in NPLM. The lower the dimensionality of the state, the more the model generalizes. Too low state dimension will, however, make the model inflexible and unable to model the source well. Even though our model is mostly linear, the softmax nonlinearity

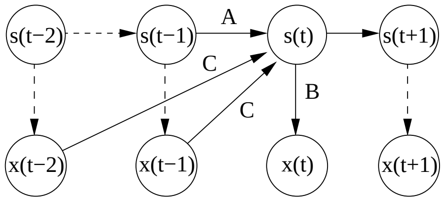

Fig. 1. The dependencies for state $\mathbf{s}(t)$.

makes its training computationally demanding.

2. THE STATE-SPACE MODEL

In this paper, boldface capital letters denote matrices and boldface lower case letters vectors. $\mathbf{s}_i$ denotes the i:th element of the vector $\mathbf{s}$, $\mathbf{A}_{j,k}$ the corresponding element of the matrix $\mathbf{A}$ and $(\mathbf{As})_l$ the l:th element of the vector resulting from the multiplication of $\mathbf{A}$ and $\mathbf{s}$.

2.1. Traditional state-space models

A basic state-space model for a time series $\mathbf{x}(t)$ is defined by two equations

$$\mathbf{s}(t+1) = \mathbf{As}(t) + \mathbf{m}(t) \tag{1}$$

$$\mathbf{x}(t) = \mathbf{Bs}(t) + \mathbf{n}(t), \tag{2}$$

where $\mathbf{s}(t)$ are the states of the system, $\mathbf{A}$ is the state transformation matrix and $\mathbf{B}$ is the observation matrix. Vector $\mathbf{n}(t)$ is Gaussian observation noise and $\mathbf{m}(t)$ is Gaussian process noise forming the innovation process.

The continuous state model is also related to discrete state hidden Markov model (HMM) that can be obtained by adding a simple discretizing nonlinearity to Eq. (1) as shown in [7]. In our case we perform the transformation to discrete domain in a different manner by adding a softmax nonlinearity to Eq. (2). Additionally our model contains explicit mappings from a few previous tokens to the new state as illustrated in Figure 1.

2.2. Structure of our model

Let us assume that we have a source which produces a stream of tokens $y(t)$, for example words or letters. The source is a random process, in which the distribution of the next token depends on the previously generated tokens. The goal is to model this source. For simplicity, let us assume that we can enumerate the set of tokens that the source can produce and the size of the set is W. A token w_i can be mapped to continuous space by using indicator vector $\mathbf{x}$, where the i:th element is set to 1 and others to zero. When the observation is missing, the elements of the estimate $\hat{\mathbf{x}}$ give the probability of each token. Later it will be shown how tokens that were not present in the training set can be cleanly handled.

Our model has a state vector $\mathbf{s}(t)$, which represents the state of the source. From this state, the probability distribution for the current token can be generated by linear mapping $\mathbf{B}$ and a softmax function. Thus, the estimated probability of drawing the i:th token is

$$\begin{aligned} P\big(y(t) = w_i \mid \mathbf{s}(t)\big) &= P\big(\mathbf{x}_i(t) = 1 \mid \mathbf{s}(t)\big) \\ &= \hat{\mathbf{x}}_i(t) = \frac{e^{(\mathbf{Bs}(t))_i}}{\sum_{i'=1}^{W} e^{(\mathbf{Bs}(t))_{i'}}} \end{aligned} \tag{3}$$

The state vector is actually a concatenation of smaller vectors. The previous state is mapped to the *internal state vector* $\mathbf{q}(t)$ by matrix $\mathbf{A}$. The dimension of the internal state is N_q. The probability vectors of the tokens in the history can be mapped by matrix $\mathbf{C}$ to lower dimensional *feature vectors* $\mathbf{l}(t)$ that are concatenated to the original internal state vector. The dimensionality of vector $\mathbf{l}(t)$ is denoted by N_l.

To estimate the new state vector $\hat{\mathbf{s}}(t)$, the prediction from the previous state $\mathbf{s}(t-1)$ is concatenated with the mappings from previous tokens, thus forming $N_s = N_q + n \cdot N_l$ dimensional vector, where n is the number of tokens that are connected to current state. Curly braces show the dimensions of the components:

$$\hat{\mathbf{s}}(t) = \left.\begin{bmatrix} \mathbf{q}(t) \\ \mathbf{l}(t-1) \\ \mathbf{l}(t-2) \\ \vdots \\ \mathbf{l}(t-n) \end{bmatrix}\right\} N_s \times 1 \tag{4}$$

$$= \begin{bmatrix} \overbrace{\mathbf{A}}^{N_q \times N_s} \overbrace{\mathbf{s}(t-1)}^{N_s \times 1} \\ \overbrace{\mathbf{C}}^{N_l \times W} \overbrace{\mathbf{x}(t-1)}^{W \times 1} \\ \mathbf{Cx}(t-2) \\ \vdots \\ \mathbf{Cx}(t-n) \end{bmatrix} \tag{5}$$

Assuming a Gaussian innovation process with covariance $\mathbf{\Lambda}$, the probability of a new state $\mathbf{s}(t)$ given the previous state $\mathbf{s}(t-1)$, the past history $\mathcal{X}_t = \{\mathbf{x}(t-1), \ldots, \mathbf{x}(t-n)\}$ and the model parameters $\mathcal{M} = \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$ is

$$\begin{aligned} &P(\mathbf{s}(t) \mid \mathbf{s}(t-1), \mathcal{X}_t, \mathcal{M}) \\ &\qquad = ce^{-\frac{1}{2}(\mathbf{s}(t)-\hat{\mathbf{s}}(t))^T \mathbf{\Lambda}^{-1}(\mathbf{s}(t)-\hat{\mathbf{s}}(t))}, \end{aligned} \tag{6}$$

where c is a normalization constant. Figure 1 illustrates the dependencies of the model.

2.3. Training the model

The model is trained by maximizing the posterior probability density of the state and the model parameters $\mathcal{M}$ for the training data:

$$\begin{aligned}&P(\mathbf{s}(t), \mathcal{M} \mid \mathbf{s}(t-1), \mathbf{x}(t), \mathcal{X}_t)\\ &\quad = \frac{P\left(\mathbf{x}(t) \mid \mathbf{s}(t), \mathcal{M}\right) \cdot P\left(\mathbf{s}(t), \mathcal{M} \mid \mathbf{s}(t-1), \mathcal{X}_t\right)}{P(\mathbf{x}(t) \mid \mathbf{s}(t-1), \mathcal{X}_t)} \qquad (7)\end{aligned}$$

Assuming that the parameters $\mathcal{M}$ have non-informative uniform priors and they are independent of the state $\mathbf{s}(t-1)$ we get joint posterior probability density

$$\frac{P(\mathbf{x}(t) \mid \mathbf{s}(t), \mathcal{M}) \cdot P(\mathbf{s}(t) \mid \mathbf{s}(t-1), \mathcal{M}, \mathcal{X}_t)}{P(\mathbf{x}(t) \mid \mathbf{s}(t-1), \mathcal{X}_t)}. \qquad (8)$$

For simplicity, we will also assume diagonal covariance $\mathbf{\Lambda}$.

Maximization of this function is performed with an EM-like algorithm one parameter at a time. As the denominator is constant with respect to parameters to be maximized, it can be ignored. First, the best state $\mathbf{s}(t)$ is found by maximizing the logarithm of the Eq. (8) with respect to $\mathbf{s}(t)$ while keeping parameters $\mathcal{M}$ constant:

$$\begin{aligned}\underset{\mathbf{s}(t)}{\operatorname{argmax}} \Big(&\log P(\mathbf{x}(t) \mid \mathbf{s}(t), \mathcal{M})\\ &+ \log P(\mathbf{s}(t) \mid \mathbf{s}(t-1), \mathcal{M}, \mathcal{X}_t)\Big)\end{aligned} \qquad (9)$$

The function can not easily be analytically maximized, but a numerical solution is feasible. A good starting point for searching the exact solution is the predicted new state $\hat{\mathbf{s}}(\mathbf{t})$. In this work, Fletcher-Reeves conjugate gradient algorithm [8] as implemented in GNU scientific library[1] is used.

After fixing $\mathbf{s}(t)$, we update the parameters $\mathcal{M}$. Gradient of logarithm of Eq. (8) is calculated with respect to each of parameters in $\mathcal{M}$ and the parameters are updated toward the maximum. This yields the following update rules.

$$\mathbf{A}_{i,j}^{new} = \mathbf{A}_{i,j} + \alpha \mathbf{\Lambda}_{i,i}^{-1} \mathbf{s}_j(t{-}1)\big(\mathbf{s}_i(t) - (\mathbf{A}\mathbf{s}(t-1))_i\big) \qquad (10)$$

$$\mathbf{B}_{i,j}^{new} = \mathbf{B}_{i,j} + \beta\, \mathbf{s}_j(t)\left(\mathbf{x}_i(t) - \hat{\mathbf{x}}_i(t)\right) \qquad (11)$$

$$\begin{aligned}\mathbf{C}_{i,j}^{new} = \mathbf{C}_{i,j} + \gamma \sum_{k=0}^{n-1} \mathbf{\Lambda}_{q,q}^{-1}\big(\mathbf{s}_q(t) - \mathbf{C}_{i,j}\big)\mathbf{x}_j(t-n),\\ \text{where } q = N_q + k \cdot N_l + i\end{aligned} \qquad (12)$$

Here α, β and γ are the corresponding learning rate parameters. Note that $\mathbf{x}(t)$ has only one nonzero element. Above, the procedure for updating the model for one learning sample was outlined. This procedure is iterated until convergence.

These update rules are valid for on-line learning. A corresponding batch algorithm can trivially be computed from Eqs. (10), (11) and (12) by summing the steps along the gradient over the batch window before updating the actual parameters. Note, that this is not the exact solution: During batch learning, we know also the future tokens. Based on these future tokens, we can approximate the future state, which should directly affect our current state.

2.4. Computational considerations

Since the scale of $\mathcal{M}$ is not fixed, we can choose arbitrary $\mathbf{\Lambda}$ and the scale of $\mathcal{M}$ should adapt accordingly. For numerical reasons, $\mathcal{M}$ should be kept small enough, so that exponential functions involved can be calculated without fear of overflow. Here, we choose that $\mathbf{\Lambda}$ is diagonal with values 0.1 on the diagonal.

If the model would be able to learn the data almost perfectly, it would still try to make the absolute values of parameters $\mathcal{M}$ bigger, since the softmax function would still give slightly higher likelihoods for bigger values. To prevent $\mathcal{M}$ from tending to infinity, we have restricted the sum of squared elements of each matrix to maximum of 10.

Training the model is computationally intensive. Finding the most probable $\mathbf{s}(t)$ requires iterations. During each of the iterations, we need to calculate the softmax function, which takes up most of the computation time of the whole learning (up to 70% in these experiments). Some clever approximations could make the algorithm computationally less demanding but such optimizations are beyond the scope of this paper.

2.5. Using the model for prediction

When predicting new tokens, the probability estimates are drawn from the estimated state $\hat{\mathbf{s}}(t+1)$. When the current token is fixed, the state estimate is corrected to the most likely state (Eq. (9)). As in training, this maximization can only be solved numerically.

In this model, no probability mass has been left to tokens that were not present in the training data. If such out-of-vocabulary (OOV) token is encountered in the history, the indicator vector $\mathbf{x}(t)$ is set to zero. This turns off the contribution of the OOV token for the next prediction.

3. EXPERIMENTS

In this work, we examine a task of letter prediction. We use letters as tokens for our language model and the goal of our language model is to predict the next letter. To measure the quality of our model, we calculate for the training material the reciprocal of the geometric mean of the probabilities

[1] http://www.gnu.org/software/gsl/

given to each token, also known as *perplexity*:

$$\mathrm{Perp}\big(y(1),\ldots,y(N)\big) = \left(\prod_{t=1}^{N} P(y(t) \mid y(t-1),\ldots,y(1))\right)^{-\frac{1}{N}}, \quad (13)$$

where N is the size of the test set. The handling of the out-of-vocabulary tokens is done in the same way as is traditional for the n-gram models: If the current token $y(t)$ is not in the vocabulary, the prediction is discarded from the mean and N is decreased accordingly. If a token in history $(y(t'), t' < t)$ is an OOV token, the prediction is taken into account normally. For our state-space model this means that for the OOV token, a zero vector is used instead of a token indicator vector.

For comparison, perplexity results obtained by a n-gram model with Good–Turing smoothing and back-off as implemented in the CMU–Cambridge toolkit [9] are also given.

3.1. Data

As data, we use excerpts from a book in Finnish. We test our model in two different situations:

- A sparse training data set (1 016 letters)
- A dense training data set (100 080 letters)

We use a different excerpt of the book (development set, 5 006 letters) to test, how different choices for parameters affect our model. Based on these tests, we choose the best state-space model and the best n-gram model and compare these in yet another data set (test set, 5 037 letters).

The sparse training data included 24 different tokens. The development set included 8 instances of OOV tokens and the test set 2. The dense training data included 25 different tokens. The development set included 6 instances of OOV tokens and the test set 2. These OOV tokens were letters that don't normally appear in Finnish text, but can be found in foreign names.

3.2. Results

When referring to state-space models, we try to conform with the traditional n-gram model naming: order 1 (unigram) refers to model with no direct connections to previous tokens, order 2 (bigram) refers to a model with one previous token directly connected to the current state etc.

In training, the initial state $\mathbf{s}(0)$ was set to zero and model parameters $\mathcal{M}$ were randomly initialized. The learning parameters (α, β, γ) were set at first to fast learning and then decreased toward the end. The learning was done in batches, with small batch sizes at the beginning of the training and increasingly bigger batches toward the end. The tests were run for orders $\{1, 2, 3, 5\}$ with internal dimensions $\{0, 1, 3, 5, 10, 20, 40\}$ and tokens were projected to dimensions $\{1, 3, 5, 10, 15, 25\}$. When the development set perplexity started to rise, the teaching was stopped to prevent overfitting.

To be fair, n-gram model's discount ranges and cutoffs were tuned by hand. This improved the n-gram performance significantly with the sparse training data.

The perplexity results for development set are shown in Tables 1 and 2. The models are grouped according to the number of previous directly connected tokens. For our state-space model, the tables show the best obtained perplexity for each order and the corresponding parameters. To show that the model is not overly sensitive to choice of parameters, the perplexity as function of model dimensions is plotted in Figure 2.

For the test set, the state-space model and the n-gram model which gave the lowest perplexities for the development set were chosen. The results are shown in Table 3.

order	N_q^{best}	N_l^{best}	perplexity	n-gram perp.
1	5	-	22.9	17.3
2	10	15	**12.5**	13.7
3	20	15	12.5	13.4
5	10	15	14.4	12.7

Table 1. Development set results, sparse training data. Order 1 state-space model does not seem to learn the data very well and order 5 state-space model also has some problems. Both order 2 and 3 state-space models get slightly better results than the best n-gram model (of order 5).

order	N_q^{best}	N_l^{best}	perplexity	n-gram perp.
2	3	15	11.1	10.8
3	5	25	10.0	8.0
5	0	15	9.7	**5.7**

Table 2. Development set results, dense training data. N-gram models are clearly better than the state-space models.

type	training set	best order	perplexity
state-space	sparse	2	12.3
n-gram	sparse	5	11.8
state-space	dense	5	9.5
n-gram	dense	5	5.7

Table 3. Test set results. The n-gram model gets slightly better perplexity with the sparse training data and clearly better perplexity with dense training data.

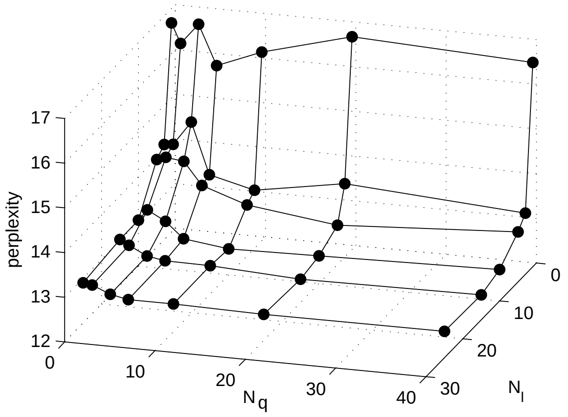

Fig. 2. Perplexity as a function of internal state dimension N_q and feature space dimension N_l for order 2 model trained from sparse data. Other state-space models behave similarly. The models are not very sensitive to the choice of parameters.

4. DISCUSSION

In the tests, the proposed state-space models seem to be about as good as the n-gram model with sparse training data. In the dense case, the n-gram models were clearly the better. Having examined training set perplexity, we conclude that the bad performance of state-space model is not because of model overfitting. This case illustrates clearly, that although our model should be able to give accurate representation of the dense case (there are enough free parameters), the training algorithm does not find the correct parameter values.

It is also clear that the training algorithm is unable to make full use of the internal state. Take for example

1) order 5 model with $N_q = 0$ and $N_l = 5$ (training set perplexity for dense data 10.0)

2) order 2 model with $N_q = 20$ and $N_l = 5$ (training set perplexity for dense data 11.1)

We can transform the first model into equivalent model of order 2 with $N_q = 20$ and $N_l = 5$: Instead of explicitly connecting the previous tokens to the state, we can have the internal state to remember these previous tokens. This equivalent model should get the same training set perplexity. The best solution that our training algorithm finds is clearly worse.

Looking at the training set perplexity, we noticed that toward the end of the training, the perplexity occasionally got worse. There are two possible explanations for this: We have limited the sum of each matrix's squared elements to 10 and this causes problems or the learning speed parameters were poorly chosen. This phenomenon should be studied more closely.

It is possible that single letters cannot be mapped all that well to a low dimensional feature space, since they do not have an independent semantic meaning. For words, the mapping should be more effective, because words have clear semantic meanings, which allows semantically similar words to be handled similarly.

4.1. Future work

In the future, the training algorithm should be studied and improved. The efficiency should be increased so that the model can be used for predicting words. Ultimately, the model should be fast enough to be used in real speech recognition tasks. Finding better ways to initialize the model could be a part of the solution.

Instead of random model initialization, the initialization could be based on a priori knowledge. For our letter prediction task, we could make use of the knowledge, that Finnish has a strict rule of vowel harmony. Encoding this kind of information into initialization is not trivial, however.

Different model structures should be studied. This paper shows one possibility of connecting states and tokens, different variations are possible. Maybe some of these variations are easier to train effectively. Ultimately, one connection from previous token should be sufficient and the training algorithm should store all other information it needs to the internal state.

The linear mapping from previous state to next state could be too restrictive. Nonlinear mappings for state dynamics, for example MLP networks, could be explored. Interpolating n-gram and state-space model estimates could lead to improved performance.

5. CONCLUSIONS

In this paper it was shown that a token source (here letters) can be modeled with a simple state-space model. With sparse training data, the model yields similar results as the baseline n-gram model. With dense training data, the n-gram model is clearly better than our state-space model. Here, the training algorithm seems to have difficulties in finding the optimal parameter values.

This kind of model does not need a separate smoothing and back-off steps to prevent overlearning, since the model complexity is directly controlled by the state dimension. It was shown that with sparse data, the state dimension can be set to a fairly small value and good results are still obtained. The internal state should be able to store also longer term dependencies, but it seems that the training algorithm is not capable of fully exploiting the internal state.

The results obtained in the experiments are encouraging. Despite some problems with the training algorithm, the state-space model performs on par with the baseline n-gram model on sparse training data. This is important, since when modeling word sequences, the data is usually also sparse. For words, there are probably stronger semantical relationships than with letters. This means that the mapping of the history to lower dimension in the state is probably more effective with words as tokens.

6. ACKNOWLEDGMENTS

Without discussions with Harri Valpola and Krista Lagus, we would never have tried this kind of model. We thank the both of you. This work was partly funded by the Finnish National Technology Agency TEKES, of which the authors are thankful.

7. REFERENCES

[1] Joshua Goodman, “A bit of progress in language modeling,” *Computer Speech and Language*, pp. 403–434, October 2001.

[2] David J. C. MacKay and Linda C. Bauman Peto, “A hierarchical Dirichlet language model,” *Natural Language Engineering*, vol. 1, no. 3, pp. 1–19, 1994.

[3] Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jennifer C. Lai, and Robert L. Mercer, “Class-based n-gram models of natural language,” *Computational Linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[4] Thomas Niesler, *Category-based statistical language models*, Ph.D. thesis, University of Cambridge, 1997.

[5] Yoshua Bengio, Rejean Ducharme, and Pascal Vincent, “A neural probabilistic language model,” *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, February 2003.

[6] Rudolf E. Kalman, “A new approach to linear filtering and prediction problems,” *Transactions of the ASME, Journal of Basic Engineering*, vol. 82, pp. 35–45, 1960.

[7] Sam Roweis and Zoubin Ghahramani, “A unifying review of linear gaussian models,” *Neural Computation*, vol. 11, no. 2, pp. 305–345, 1999.

[8] Roger Fletcher, *Practical Methods of Optimization*, John Wiley & Sons, New York, second edition, 1987.

[9] Philip Clarkson and Ronald Rosenfeld, “Statistical language modeling using the CMU-Cambridge toolkit,” in *Proceedings of Eurospeech 1997*, 1997, pp. 2707–2710.