Probabilistic Neural Network Models for Sequential Data

Yoshua Bengio
Département d'informatique et recherche opérationnelle
Université de Montréal
C.P. 6128 Succ. Centre-Ville, Montréal, Québec, Canada, H3C 3J7
bengioy@iro.umontreal.ca

Abstract

It has already been shown how Artificial Neural Networks (ANNs) can be incorporated into probabilistic models. In this paper we review some of the approaches which have been proposed to incorporate them into probabilistic models of sequential data, such as Hidden Markov Models (HMMs). We also discuss new developments and new ideas in this area, in particular how ANNs can be used to model high-dimensional discrete and continuous data to deal with the curse of dimensionality, and how the ideas proposed in these models could be applied to statistical language modeling to represent longer-term context than allowed by trigram models, while keeping word-order information.

1 Introduction

Probabilistic models are commonly used to build machine learning applications, and Artificial Neural Networks (ANNs) could play a useful role in such models. This paper starts from an overview of some of the approaches that have been proposed to incorporate ANNs into probabilistic models, especially those that are used to model sequential data. After a discussion of how ANNs can be helpful in this context in order to deal with the curse of dimensionality, especially for modeling discrete data, the paper presents a proposal for new ANN-based probabilistic language model.

In the next section, we review how ANNs can be given a probabilistic interpretation and incorporated into probabilistic models such as HMMs (Hidden Markov Models). In section 3, we digress to a recently proposed general approach to representing joint distributions from the product of conditionals, using ANNs (Bengio and Bengio, 2000). This approach is interesting because it raises the issue of how ANNs can be useful to address the curse of dimensionality. Finally, in section 4, we propose a new language modeling approach based on these ideas. The proposed class of models allows to represent long-term context without suffering in principle from the curse of dimensionality that hurts n-gram models when n goes beyond 3.

2 ANNs in Probabilistic Models and IOHMMs

As already described in detail in (Bishop, 1995), it is straightforward to interpret multi-layer neural networks in a probabilistic setting. For example, the ordinary neural networks with input x trained to minimize the squared error between their output $\mu(x)$ and some desired outputs y can be interpreted as estimating the expected value $\mu(x) = E[y|x]$ (see for example (White, 1989) for a demonstration). Minimizing the average squared error of the ANN is equivalent to maximizing the average log-likelihood of a probabilistic model of $P(y|x)$ where y is conditionally Gaussian with expectation $\mu(x)$. When regularization terms such as weight decay are incorporated into the training criterion of ANNs, this corresponds to training the probabilistic model according to the MAP (Maximum A Posteriori) criterion, which is the sum over the training pairs (x_t, y_t) of the log-likelihoods $\log P(y_t|x_t)$ and a log-prior $\log P(\theta)$ (where θ are the parameters of the ANN). For example, weight decay given by a penalty $0.5\lambda||\theta||^2$ added to the cost function corresponds to a 0-mean Gaussian prior with variance $1/\lambda$ for the parameters.

To go further in the direction of building a probabilistic model, as in (Bishop, 1995) one can add extra free parameters to represent the log-variances. Log-variances are not constrained to be positive so they can be learned using the same gradient-based methods applied to train ANNs. Similarly, in the case of classification, the natural probabilistic model is a binomial or multinomial, and the ANN can compute conditional class probabilities. When the number of classes

is greater than 2, the *softmax* output transformation (eq. 1 below) can be used to ensure that the outputs of the ANN are positive and sum to 1. For example, the ANN can compute the estimated conditional probabilities

$$P(y = i|x) = e^{g_i(x)} / \sum_j e^{g_j(x)} \tag{1}$$

where

$$g_i(x) = b_i + \sum_j w_{ij} \tanh(c_j + \sum_k v_{jk} x_k) \tag{2}$$

with ANN parameters $(b_i, w_{ij}, c_j, v_{jk})$. Softmax outputs can also be used to build *mixtures of experts* (Jacobs et al., 1991), where $P(y|x)$ is decomposed as

$$P(y|x) = \sum_i P(E = i|x) P(y|x, E = i)$$

where E denotes an expert, $P(E = i|x)$ is the output of a "gating" ANN with a softmax output layer (as in 1), and $P(y|x, E = i)$ is the model associated to the i-th "expert". For example, each of these "expert" models could be conditionally Gaussian, like the simple ANNs initially described in this section. Another way to interpret and build this model is that $P(y|x)$ is a Gaussian mixture whose parameters (means, log-variances, and mixture weights) are the outputs of a large ANN. More generally, one can **use an ANN to compute the parameters $\theta(x)$ of a parametrized conditional distribution** $P(y|x) = P(y|\theta(x))$.
These ideas have been applied to HMMs in order to incorporate ANNs in these probabilistic models and transform these models into conditional probability models.

2.1 HMMs and IOHMMs

The joint probability distribution of a sequence of observations $y_1^T = \{y_1, y_2, \ldots, y_T\}$ can always be factored as

$$P(y_1^T) = P(y_1) \prod_{t=2}^{T} P(y_t|y_1^{t-1}).$$

It appears intractable in general to model sequential data in which the conditional distribution $P(y_t|y_1^{t-1})$ of an observed variable y_t at time t depends on all the details of the previous values y_1^{t-1}. However, most models have the property that they assume that the past sequence can be summarized concisely, often using an unobserved random variable called a **state variable**, which carries all the information from y_1^{t-1} that is useful to describe the distribution of the next observation y_t.
The most common of these models are the HMMs, which are best known for their contribution to advances in automatic speech recognition in the last two decades (Rabiner, 1989). See (Bengio, 1999) for a review of Markovian models such as HMMs and ANN/HMM hybrids. In basic HMMs, a discrete state variable q_t is postulated, with the following conditional independence assumptions:

$$P(y_t|q_1^t, y_1^{t-1}) = P(y_t|q_t) \tag{3}$$

$$P(q_{t+1}|q_1^t, y_1^t) = P(q_{t+1}|q_t) \tag{4}$$

In simple terms, the state variable $q_t \in \{1, \cdots, n\}$ summarizes all the relevant past values of the observed and hidden variables when one tries to predict the value of the observed variable y_t, or of the next state q_{t+1}. The marginalization of the joint distribution of states and observations gives the likelihood of a sequence,

$$P(y_1^T) = \sum_{q_1^T} P(y_1^T, q_1^T)$$

$$P(y_1^T, q_1^T) = P(q_1^T) P(y_1^T|q_1^T) = P(q_1) \prod_{t=1}^{T-1} P(q_{t+1}|q_t) \prod_{t=1}^{T} P(y_t|q_t) \tag{5}$$

The joint distribution is therefore completely specified in terms of (1) the *initial state probabilities* $P(q_1)$, (2) the *transition probabilities* $P(q_t|q_{t-1})$ and, (3) the *emission probabilities* $P(y_t|q_t)$.
Conditional HMMs or Input-Output HMMs (IOHMMs) are described in more detail in (Bengio, 1999) and in different variants in (Cacciatore and Nowlan, 1994; Bengio and Frasconi, 1995; Meila and Jordan, 1996; Bengio and Bengio, 1996). Whereas HMMs represent the distribution $P(y_1^T)$ of sequences y_1^T, IOHMMs represent the conditional distribution $P(y_1^T|x_1^L)$ of a sequence y_1^T given another sequence x_1^L (which may be of the same length or not as y_1^T, see (Bengio and Bengio, 1996)). The IOHMM distribution is parametrized with two types of ANNs: the *conditional emission models* represent $P(y_t|x_t, q_t = i)$ (they are like the experts in a mixture of experts, there is one such expert per state i of the model), and the *conditional transition models* represent $P(q_t|x_t, q_{t-1} = i)$ (they are like the gating ANNs of the mixture of experts, there is one such gater per state i, and they usually have a softmax output layer). The data likelihood for IOHMMs can be computed in a way that is very similar to the algorithm used for HMMs (with the so-called "forward" recurrence (Bengio, 1999)).

3 Dealing with the Curse of Dimensionality in Joint Distributions

Although the fact that ANNs can be incorporated into probabilistic models is interesting in itself, some recent work suggests that it may bring a notable benefit. In (Bengio and Bengio, 2000), a probabilistic ANN-based model of the joint distribution of many discrete or continuous variables is presented. The objective of this work is to use the ANN representation to deal with the *curse of dimensionality* that hits when trying to represent the joint distribution of many variables. For example, if we have 100 binary variables, one needs in principle $2^{100} - 1$ free parameters to represent their joint probability function: these parameters are the probabilities assigned to each of the possible combinations of values of the variables. In the 60's, approximations of the joint probability function of a large number of binary variables have been proposed, essentially based on *polynomial approximations* (Bahadur, 1961; Duda and Hart, 1973) of the probability function. In (Frey, 1998), an approximation based on the logistic distribution was proposed, called LARC (logistic auto-regressor classifier). This can be seen as an ANN without hidden units, for each of the random variables, giving its conditional probability $P(y_i|y_1^{i-1})$ given the previous variables in some arbitrary order.
In (Bengio and Bengio, 2000), this idea is extended to ANNs with hidden units, using an architecture that maintains the constraint that the functions computed by the network, $p_{i,j} = f_{i,j}(y_1^{i-1}) = P(y_i = j|y_1^{i-1})$ depend only on the first $i-1$ input values, with $j = 1$ to number of values of y_i, $i = 1$ to the number of variables. From the conditional probabilities $p_{i,z_i} = f_{i,z_i}(z_1^{i-1}) = P(y_i = z_i|y_1^{i-1})$ computed by the network, one can obtain the joint probability $P(y_1 = z_1, y_2 = z_2, \cdots, y_n = z_n)$ simply by multiplying the values of the selected output units $\prod_i p_{i,z_i}$. In the case where y_i is binary, a sigmoid output unit can be used to compute $p_{i,1}$ (with $p_{i,0} = 1 - p_{i,1}$). In the case where y_i is a discrete variable taking n_i values, a group of output units with a softmax function can be used to obtain probabilities $p_{i,j}$ for the n_i possible values of y_i. Both for the discrete inputs and outputs of the ANN, a *one-hot* representation is therefore used (a "high" value at the k-th position and "low" values at the others, to represent $y_i = k$).
It is interesting to compare this ANN model with other models of the joint distribution. In the polynomial approximation of the probability function of n random variables, all the k-th order dependencies (between groups of k variables) will be represented, but at the cost of having to estimate $O(n^k)$ parameters, which the amount of data may not allow, so one is typically restricted to $k = 2$. In that case, the joint probability is expressed as a constant plus a linear combination of the variable values, plus a linear combination of all the pairs of variables products. In the ANN of (Bengio and Bengio, 2000), the number of free parameters is $O(Hn^2)$, where H scales the number of hidden units, but *dependencies of potentially any order can be represented*. Although an ANN with H small does not allow to represent all of them, those that are most important to explain the data will be "chosen" through learning. In addition to this advantage in terms of the order of the dependencies that can be represented, ANNs bring another feature that is very important for generalization. It is the fact that they allow to *transfer* something learned on some combination of the input variables to other combinations. In contrast, when "learning" the joint probability function of a set of variables by counting co-occurences, it is not clear how to assign probabilities to new combinations of values not seen in the training set (since most out-of-sample combinations are likely to be new combinations of the values of the variables). The maximum likelihood model would assign 0 probability to unseen cases (this is disastrous for generalization!). A multinomial model trained with the MAP criterion would typically give a constant small probability to unseen combinations. Instead the ANN will give a non-constant probability to

new combinations, depending on how "far" they are or how they relate (in some sense learned by the parameters of the ANN) to previously seen combinations.
Interestingly, in experiments described in (Bengio and Bengio, 2000), the above approach was found to generalize very well in terms of out-of-sample log-likelihood, on four publically available data sets, against different models including polynomial models, simple belief networks, the naive Bayes model, and the LARC model. As proposed in (Bengio and Bengio, 2000), this approach can be extended to modeling both continuous and discrete random variables as well as conditional probabilities (following the principles illustrated in the previous section and in (Bishop, 1995)).

4 A New Language Model

Can the type of ANN described in the previous section be applied to modeling the joint probability of word sequences, i.e. to build a language model? State-of-the-art language models are based on a very crude estimation of the word sequence joint probability through so-called n-grams:

$$P(y_1^T) = \prod_t P(y_t|y_{t-n}^{t-1})$$

where $y_t \in V$ is the symbol associated to the word observed at position t in a word sequence, in the vocabulary V. The parameters are obtained easily from *co-occurence counts*. Because of the curse of dimensionality, the best models are based on n=2 (*trigrams*) and because most triplets are infrequent, the probability is smoothed (Jelinek and Mercer, 1980) using bigrams ($P(y_t|y_{t-1})$) and unigrams ($P(y_t)$) (for example with a mixture between the predictions of the trigram, bigram and unigram, with more weight given to bigram and unigram when the triples are rare).
Let us see how one could learn such conditional probabilities with an ANN. One problem is that each word is a discrete variable that can take around $|V| = 10,000$ to several $100,000$'s different values, depending on the size of the vocabulary V. With a one-hot encoding of the input words, that would make a very large number of network weights and therefore a very large number of computations. For statistical models based on co-occurence counts the number of required parameters may grow as $|V|^{n+1}$. Using *Variable-Length Markov Models* (Ron, Singer and Tishby, 1996), one would only learn the probabilities associated to contexts that occur sufficiently in the data, i.e. the number of parameters would grow like $O(nT|V|)$ where T is the length of the corpus in words (in the millions, or even hundreds of millions!). However, in both cases, the only "generalization" that would occur from contexts seen in the training corpus to new contexts y_{t-n}^{t-1} (which will be seen out-of-sample) is obtained by considering a shorter context $y_{t-n'}^{t-1}$ with $n' < n$ (a context short enough to have been seen in the training data, and to have been seen enough to make more reliable predictions on the next word). In the case of a straightforward ANN taking a one-hot representation of the previous n words in inputs, with H hidden units, the number of parameters would grow as $(n+1)|V|H$ or $n|V|^2 + (n+1)|V|H$ (depending on whether or not there are direct input to output connections).
To make the computation with the ANN more manageable and to improve its chances of generalizing, we propose to use a different representation for the words, one which has been proposed for a long time in the connectionist language modeling litterature (Miikkulainen and Dyer, 1991). The idea is to use a *distributed representation* in which each word is represented by a real-valued code that is a point in a low-dimensional vector space (e.g. in $\mathbb{R}^{50}$) rather than using the "one-hot" representation. In input to the ANN, there will be the low-dimensional code vector for each of the n input words y_{t-1} to y_{t-n}. With this distributed representation, each element of the code vector associated to a word represents a "direction" in a sort of semantic space. Words i and j that are close in this space should be easily replaced for each other in the same context, i.e. $P(y_t = i|y_1^{t-1})$ is "close" to $P(y_t = j|y_1^{t-1})$ for most contexts y_1^{t-1}. But how do we choose this particular representation (i.e. the code vector, e.g. in $\mathbb{R}^{50}$, for each word in the vocabulary)? The idea of mapping words to low-dimensional real vectors has already been used successfully for information retrieval, e.g. see the "Latent Semantic Indexing" (Deerwester et al., 1990) and "Word Space" ideas (Schütze, 1992), both using a *singular value decomposition* of a high-dimensional sparse representation of words based on co-occurence counts in different contexts. Ideally, we would like to *learn this low-dimensional representation* such that it helps to build a good language model (one that gives high probabilities to plausible word sequences in the language of interest).
This can be achieved as follows. Let $v_i \in \mathbb{R}^m$ be the low-dimensional real code vector associated to word i. We want to learn the matrix v whose columns are v_i along with the parameters of the ANN that computes conditional

word probabilities. We will construct a parametrized function (e.g. an ANN) with parameters w computing the vector-valued function $f(v_{y_{t-1}}, v_{y_{t-2}}, \cdots, v_{y_{t-n}}; w)$ whose value is in $(0,1)^{|V|}$, with a softmax output layer (i.e. $f_i > 0$ and $\sum_i f_i = 1$). The i-th element of the output vector, f_i is an estimate of the conditional probability $P(y_t = i | y_{t-n}^{t-1})$. The number of weights of such a network, with H hidden units (and no direct input-output connections) is $O(nmH + |V|H)$.
The maximum likelihood training criterion is simply

$$L(w, v) = \sum_t \log f_{y_t}(v_{y_{t-1}}, v_{y_{t-2}}, \cdots, v_{y_{t-n}}; w).$$

where the sum is over the whole training corpus (seen as a very long sequence). Note that we only consider the probability associated to the "correct" next word, f_{y_t}.
One can also view this model as a neural network with an additional hidden layer with shared weights and local connectivity, and whose input layer would have a 0/1 one-hot encoding for the input words. The input units would be organized as n groups of V units. In this additional first hidden layer there would also be n groups, each with m linear units (m is the dimension of the code vector for a word). The i-th group of hidden units of this first layer would only receive connections from the i-th group of input units, and its input weight matrix would be v, i.e. the same for all the groups. Because there would be only a single non-zero entry in the input units for word i, the vector computed at the i-th hidden group would be $v_{y_{t-i}}$. In practice, one would not implement this first hidden layer with actual matrix multiplications (that would be very wasteful), but using a simple table look-up (to map y_{t-i} to $v_{y_{t-i}}$).
Because of the immense size of the training data set (several hundred million words), the only training algorithm that seems practicable is stochastic gradient. Note that stochastic gradient has already been successful at training very large neural networks for character recognition with hundreds of thousands of examples (Bottou et al., 1994).
To initialize v, we propose to use a singular value decomposition of a large sparse matrix whose entries are co-occurence counts and other informations about each word, such that words that can easily be replaced in the same context have similar entries. An extension of the above model would deal in a straightforward way with *new words* not already seen in the training corpus, and could also be used in a second phase of initialization of v. Instead of assigning a dummy symbol and an arbitrary low probability to unseen words, as in the count-based statistical models such as trigrams, we could use the model itself to assign a code vector to new words, depending on the context in which they occur. The idea is to train the ANN not only to compute the probability of the next word within the seen vocabulary V, but also to train it to predict the *expected value* of the next word in v-space, i.e. $\mu_t = E[v_{y_t} | v_{y_{t-1}}, v_{y_{t-2}}, \cdots, v_{y_{t-n}}]$. This can be done by adding m extra output units which will be compared with v_{y_t} in the training criterion. In this way, when a new word is encountered, we can use this expected vector as the initial code-word for the new word, and feed it back in the input window in order to predict the words that follow. In order to properly train these extra outputs, one has to form an estimate of $P(y_t | y_{t-n}^{t-1})$ from μ_t. This can be obtained by postulating a conditional Gaussian model for each word, and forming and maximizing the products of $P(y_t | y_{t-n}^{t-1}) = e^{-0.5(\mu_t - v_{y_t})^2} / \sum_i e^{-0.5(\mu_t - v_i)^2}$ (i.e. a softmax of the squared distances).
The number of computations remain fairly large, but not exponential in n. The computational bottleneck is in the output layer, which requires $O(|V|H)$ computations. However, it is likely that this can be easily reduced by pre-selecting a "small" (compared to V) set of likely next words. This could be done for example with a language model based on a table look-up, such as a trigram model. Another solution is to form a hierarchical clustering in v-space and train the ANN to predict in which clusters the next word may occur. Then one needs only compute the probabilities of the words from the clusters with significantly non-zero probability.
Since we know that there is a lot of local sequential structure in language, one could use convolutional (or time-delay) neural networks instead of ordinary feedforward neural networks (to extract progressively higher-level contextual information from groups of neighboring words). One could also attempt to use recurrent networks to learn a continuous state function in order to capture longer-term dependencies (although some learning problems may be anticipated in this case (Bengio et al., 1994)).

5 Conclusion

Starting from a brief overview of how ANNs have been incorporated into probabilistic models, in particular those that concern sequential data, such as HMMs, we have discussed how ANNs could be useful to address the generalization problem also known as the curse of dimensionality when trying to model the joint or conditional distributions of

many discrete variables. We have then proposed a new approach to language modeling that takes advantage of the power of ANNs to represent the joint distribution between many variables without involving an exponential number of parameters. Taking stock of earlier work on connectionist models of language, this approach is based on *learning a low-dimensional distributed representation* for each word in a vocabulary, in such a way that this encoding provides the best predictive probabilities of the next word given the previous words in a training corpus.

References

Bahadur, R. (1961). A representation of the joint distribution of responses to n dichotomous items. In Solomon, H., editor, *Studies in Item Analysis and Predictdion*, pages 158–168. Stanford University Press, California.

Bengio, S. and Bengio, Y. (1996). An EM algorithm for asynchronous input/output hidden Markov models. In Xu, L., editor, *International Conference On Neural Information Processing*, pages 328–334, Hong-Kong.

Bengio, S. and Bengio, Y. (2000). Taking on the curse of dimensionality in joint distributions using neural networks. *IEEE Transactions on Neural Networks, special issue on Data Mining and Knowledge Discovery*. to appear.

Bengio, Y., , Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166. Special Issue on Recurrent Neural Networks, March 94.

Bengio, Y. (1999). Markovian models for sequential data. *Neural Computing Surveys*, 2:129–162.

Bengio, Y. and Frasconi, P. (1995). An input/output HMM architecture. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems 7*, pages 427–434. MIT Press, Cambridge, MA.

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, London, UK.

Bottou, L., Cortes, C., Denker, J., Drucker, H., Guyon, I., Jackel, L., LeCun, Y., Muller, U., Sackinger, E., Simard, P., and Vapnik, V. (1994). Comparison of classifier methods: a case study in handwritten digit recognition. In *International Conference on Pattern Recognition*, Jerusalem, Israel.

Cacciatore, T. W. and Nowlan, S. J. (1994). Mixtures of controllers for jump linear and non-linear plants. In Cowan, J., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, San Mateo, CA. Morgan Kaufmann.

Deerwester, S., Dumais, S., Furnas, G., Landauer, T., and R.Harshman (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

Duda, R. and Hart, P. (1973). *Pattern Classification and Scene Analysis*. Wiley, New York.

Frey, B. (1998). *Graphical models for machine learning and digital communication*. MIT Press.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixture of local experts. *Neural Computation*, 3:79–87.

Jelinek, F. and Mercer, R. (1980). Interpolated estimation of markov source parameters from sparse data. In Gelsema, E. and Kanal, L., editors, *Pattern Recognition in Practice*. North-Holland, Amsterdam.

Meila, M. and Jordan, M. (1996). Learning fine motion by markov mixtures of experts. In Mozer, M., Touretzky, D., and Perrone, M., editors, *Advances in Neural Information Processing Systems 8*. MIT Press, Cambridge, MA.

Miikkulainen, R. and Dyer, M. (1991). Natural language processing with modular neural networks and distributed lexicon. *Cognitive Science*, 15:343–399.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Ron, D., Singer, Y., and Tishby, N. (1996). The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25.

Schütze, H. (1992). Dimensions of meaning. In *Supercomputing'92*, pages 787–796, Minneapolis MN.

White, H. (1989). Learning in artificial neural networks: A statistical perspective. *Neural Computation*, 1(4):425–464.