A Neural Words Encoding Model

Dayiheng Liu
Machine Intelligence Laboratory
College of Computer Science
Sichuan University
Chengdu 610065, P. R. China

Jiancheng Lv
Machine Intelligence Laboratory
College of Computer Science
Sichuan University
Chengdu 610065, P. R. China

Xiaofeng Qi
and Jiangshu Wei
Machine Intelligence Laboratory
College of Computer Science
Sichuan University
Chengdu 610065, P. R. China

***Abstract*—This paper proposes a neural network model and learning algorithm that can be applied to encode words. The model realizes the function of words encoding and decoding which can be applied to text encryption/decryption and word-based compression. The model is based on Deep Belief Networks (DBNs) and it differs from traditional DBNs in that it is asymmetric structured and the output of it is a binary vector. With pre-training of multi-layer Restricted Boltzmann Machines (RBMs) and fine-tuning to reconstruct word set, the output of code layer can be used as a kind of representation code of words. We can change the number of neurons of code layer to control the length of representation code for different applications. This paper reports on experiments using English words of American National Corpus to train a neural words encoding model which can be used to encode/decode English words, realizing text encryption and data compression.**

I. Introduction

Text encryption and compression have a wide range of application. An interesting approach to text compression is not taking this textual data as sequences of characters or bytes, but as sequences of words, these words may be real words from spoken language, but also sequences of characters [1]. Word-based compression algorithm is a new type of text compression algorithm while most traditional algorithms perform compression at the character level [2]. It can fast compression and decompression natural language texts and allows fast and flexible searching of words [3].

We proposes a neural network model called neural words encoding model which can be applied to encode words. It has advantages in fast compression/decompression of natural language text and words searching as well as word-based compression algorithm, besides, by nonlinear transformation of data in neural network, the model can also encrypt and decrypt words fast.

Neural words encoding model is a neural network model based on Deep Belief Networks. Deep belief networks (DBNs) model is a typical deep learning structure which was first proposed by Hinton [4]. It is a probabilistic model consisting of multiple layers of stochastic hidden variables. DBNs can be viewed as a composition of several Restricted Boltzmann Machines (RBMs). Restricted Boltzmann Machines is a parameterized generative model representing a probability distribution which consist of two types of units called visible and hidden neurons [5]. Given some observations in visible neurons, the training data, learning a RBM means adjusting the RBM parameters such that the probability distribution represented by the hidden neurons of RBM fits the training data as well as possible. They can be viewed as non-linear feature detectors [6]. Applying contrastive divergence [7] to the layer-by-layer unsupervised training procedure of each RBM in turn, DBNs is able to generate training data with the maximum probability [8].

The neural words encoding model differs from traditional DBNs in that it is asymmetric structured and the output of it is a binary vector. The model consists of two parts, encoder and decoder. The part of encoder can encode and encrypt words fast by nonlinear transformation while the decoder part can decode and decrypt words rapidly. Encoder and decoder are connected with code layer whose outputs are representation codes of words which are m-dimensions binary vectors, that means the code of one word only needs m-bits to store. Because neuron number of code layer is flexible, we can change the length of word code for different applications. In this paper, approximate 60,000 English words of American National Corpus are used to train a neural words encoding model. The learning procedure of the model can be divided into two stages: pre-training with unlabelled samples and fine-tuning the whole network with labeled samples. After training, English words of training data were encoded uniformly, the outputs of code layer are representation codes of English words which are 64-dimensions binary vectors. Besides, many English words which are not in training data and many random strings can also be encoded and decoded by the model.

II. A Neural Words Encoding Model

The training set V is a set of words, for example, the vocabulary of American National Corpus which includes approximate 60,000 different English words. The goal is to learn a good model to

1) **Reconstruct word set:** To obtain a unified encoding and decoding method of word set, reconstructing every word is necessary. After training a model to learn features of the word set, the output of code layer is a kind of unified presentation code.
2) **Output binary code:** The output of code layer is encoding of word set which are m-dimensions vectors. As binary code can be converted into integers by binary conversion, if the output vectors are binary vectors, the code of one word only needs m-bits to store. Compared

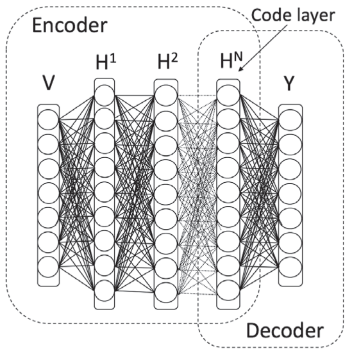

Fig. 1. Architecture of neural words encoding model

with real-value code, binary code requires less memory to store and has wider applications, for example, as input of simHash [9, 10].

Neural words encoding model is a neural network model based on Deep Belief Networks. In order to get binary output code of code layer, the structure of the model is asymmetric that is different from DBNs. Figure 1 shows a typical model with one input layer and N hidden layers H^1, H^2, ..., H^N. V is input, a word set which has been vectorized, and Y is learning target, the word itself which we want to reconstruct. The last hidden layer H^N is code layer whose output is the representation code of input. The whole deep network consists of two parts, encoder and decoder, as Figure 1 shows. The part of encoder can encode and encrypt words fast by nonlinear transformation while the decoder part can decode and decrypt words rapidly. The learning procedure of the model can be divided into two stages: pre-training with unlabelled samples and fine-tuning the whole deep network with labeled samples.

A. Pre-Training

The word set can be very large, so multiple hidden layers are necessary to reconstruct the whole word set. With random initialized weights, the model typically find poor local minima and the gradients in the early layers are tiny. The goal of pre-training is to find weights which are close to good solutions and make gradient descent work well.

Pre-training is unsupervised learning, in this stage, each pair of layers groups together to reconstruct the input layer to the output layer. Figure 2 shows the layer-wise reconstruction happens between V and H^1, H^1 and H^2 , ... , H^{N-1} and H^N, respectively, which is implemented by a family of Restricted Boltzmann Machines (RBMs) [11].

We take the input layer V and the first hidden layer H^1 for example to illustrate the training algorithm of each pair of layers. The input layer V and the first hidden layer H^1 are modeled as a Restricted Boltzmann Machine, while $|V|$ is the neuron number in V and $|H^1|$ is the neuron number in H^1.

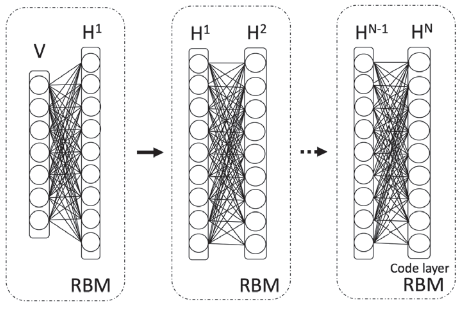

Fig. 2. Layer-wise pre-training

The energy of the state $(\boldsymbol{v}, \boldsymbol{h^1})$ in this RBM is

$$\begin{aligned} E(\boldsymbol{v}, \boldsymbol{h^1}; \boldsymbol{\theta^1}) &= -\left(\boldsymbol{v}\boldsymbol{W^1}\boldsymbol{h^1} + \boldsymbol{b^1}\boldsymbol{v} + \boldsymbol{c^1}\boldsymbol{h^1}\right) \\ &= -\sum_{i=1}^{|V|}\sum_{j=1}^{|H^1|} v_i W_{i,j}^1 h_j^1 \\ &\quad - \sum_{i=1}^{|V|} b_i^1 v_i - \sum_{j=1}^{|H^1|} c_j^1 h_j^1 \ . \end{aligned} \tag{1}$$

Where θ^1= $(\boldsymbol{W^1}$,$\boldsymbol{b^1}$,$\boldsymbol{c^1})$ are the parameters in this RBM, $W_{i,j}^1$ is the symmetric weight from i_{th} neuron in V to j_{th} neuron in H^1. b_i and c_j are bias of i_{th} neuron in V and j_{th} neuron in H^1. Joint distribution of this RBM is

$$\begin{aligned} P(\boldsymbol{v}, \boldsymbol{h^1}; \boldsymbol{\theta^1}) &= \frac{1}{Z} \mathrm{e}^{-E(\boldsymbol{v}, \boldsymbol{h^1}; \boldsymbol{\theta^1})} \\ &= \frac{\mathrm{e}^{-E(\boldsymbol{v}, \boldsymbol{h^1}; \boldsymbol{\theta^1})}}{\sum_v \sum_{h^1} \mathrm{e}^{-E(\boldsymbol{v}, \boldsymbol{h^1}; \boldsymbol{\theta^1})}} \ . \end{aligned} \tag{2}$$

Here Z is the normalization parameter, the conditional distributions over the input state $\boldsymbol{v}$ in layer V and hidden state $\boldsymbol{h^1}$ in H^1 are able to be given by the logistic function respectively as follows:

$$\begin{aligned} P(h_k^1 = 1 | \boldsymbol{v}; \boldsymbol{\theta^1}) &= \sigma\Big(\sum_{i=1}^{|V|} c_k^1 + w_{k,i}^1 v_i\Big) \ , \\ P(v_k = 1 | \boldsymbol{h^1}; \boldsymbol{\theta^1}) &= \sigma\Big(\sum_{j=1}^{|H^1|} b_k^1 + w_{j,k}^1 h_j^1\Big) \ . \end{aligned} \tag{3}$$

Where $\sigma(x)$ is logistic function 1/[1 + exp (-x)].

The parameter θ is initialized with random Gaussian distribution values, θ can be updated step by step with Contrastive Divergence algorithm [12], the change in parameter is given by

$$\begin{aligned} &\Delta w_{i,j}^1 = \epsilon\Big(\Big\langle v_i(0) h_j^1(0)\Big\rangle_{data} - \Big\langle v_i(t) h_j^1(t)\Big\rangle_{recon}\Big) \ , \\ &\Delta b_i^1 = \epsilon(v_i(0) - v_i(t)) \ , \\ &\Delta c_j^1 = \epsilon(h_j^1(0) - h_j^1(t)) \ . \end{aligned} \tag{4}$$

In which epsilon is a learning rate, $< \cdot >_{data}$ means the expectation with respect to the data distribution and $< \cdot >_{recon}$ means the reconstruction distribution after one step, here t is step size which is set to be one typically [12].

After training first group(V, H^1), the first hidden layer H^1 becomes the input of next group(H^1, H^2) for training the next RBM. The whole pre-training repeats this layer-by-layer learning as many times as desired. The last hidden layer H^N is code layer. We can set a proper neuron number of code layer to control the length of encoding for different applications. After the pre-training discussed above, the features of words are progressively combined from loose low-level representations into more compact high-level representations.

B. Fine-Turning

In this supervised stage, our goal is to reconstruct word set and make the output of code layer H^N binary. After pre-training multiple layers of words features, we get a set of decent weights $\boldsymbol{\theta}(W^1, W^2, ..., W^N)$ to initiate the parameters of encoder part of the model, by attaching decoder part with random Gaussian distribution weights W^r and target layer Y to the encoder part we get a neural words encoding model as Figure 1 shows.

We add a threshold function to the output of code layer H^N to make it binary, the output of k_{th} neuron of code layer is

$$a_k^N = \begin{cases} 1, & \text{if } \sigma(\sum_{j=1}^{|H^{N-1}|} w_{kj}^{N-1} a_j^{N-1} + c_k^{N-1}) > 0.5 \\ 0, & \text{others} \end{cases} . \tag{5}$$

Here c_k^{N-1} is bias of hidden layers H^{N-1} which has been trained in pre-training stage and σ is logistic function 1/[1 + exp (-x)].

The output of k_{th} neuron of other layers H^l is

$$a_k^l = \sigma(\sum_{j=1}^{|H^{l-1}|} w_{kj}^{l-1} a_j^{l-1} + c_k^{l-1}) . \tag{6}$$

To reconstruct word set, learning purpose is to minimize the cost function J as follows:

$$J = \frac{1}{2|Y|} \sum_{j=1}^{|Y|} (\hat{y}_j - y_j)^2 \tag{7}$$

in which y is the input itself which we want to reconstruct and $\hat{y}$ is the real output of our model, $|Y|$ is neuron number of target layer.

At last, the whole neural network is refined using back propagation algorithm via a global gradient-based optimization strategy. We use steepest descent method to update weights $\boldsymbol{\theta}(W^1, W^2, ..., W^N)$

$$w_{j,i}^l \leftarrow w_{j,i}^l - \alpha \frac{\partial J}{\partial w_{j,i}^l} . \tag{8}$$

Here α is learning rate which can be a constant or dynamically set [13, 14]. To compute $\frac{\partial J}{\partial w_{j,i}^l}$ we define z_k^l and δ_k^l as follows:

$$z_k^l = \sum_{j=1}^{|H^{l-1}|} w_{kj}^{l-1} a_j^{l-1} + c_k^{l-1} , \tag{9}$$

$$\delta_k^l = \frac{\partial J}{\partial z_k^l} . \tag{10}$$

According to chain rule we have

$$\frac{\partial J}{\partial w_{j,i}^l} = \frac{\partial J}{\partial z_j^{l+1}} \cdot \frac{\partial z_j^{l+1}}{\partial w_{j,i}^l} = \delta_j^{l+1} \cdot a_i^l , \tag{11}$$

δ_i^l can be computed by back propagation [15] as follows:

$$\delta_i^l = \sigma'(z_i^l) \cdot (\sum_{j=1}^{|H^{l+1}|} \delta_j^{l+1} \cdot w_{j,i}^l) . \tag{12}$$

and δ_i^L can be computed as follows:

$$\delta_i^L = \frac{\partial J}{\partial z_i^L} = (\hat{y} - y_i) \frac{\partial \hat{y}_i}{\partial z_i^L} = (\hat{y} - y_i) \sigma'(z_i^L) . \tag{13}$$

Each iteration we compute J and update weights as above until the algorithm converges.

After fine-turning, for any word of word set which is input, the output of code layer is its encoding.

III. Experiment and Analysis

In this section we will take an experiment on the vocabulary of American National Corpus which includes over 60,000 different English words. Each English word is a string with whose length between one and twenty-eight. Figure 3 shows the distribution of English word lengths. The distribution is

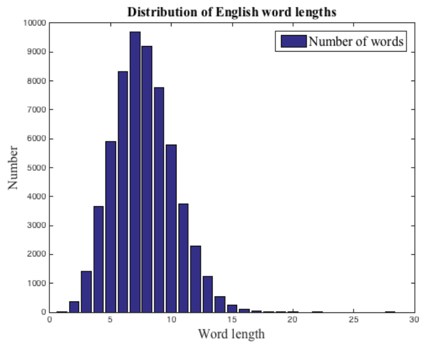

Fig. 3. Distribution of English word lengths

in concordance with the Gaussian distribution, besides, the proportion of words whose length are less than thirteen is 96.37% in this vocabulary. We take the whole words of the vocabulary whose length are less than thirteen as training data of our model which includes 58,194 different English words.

A. Vectorizing Input

Before training the proposed model, as English words string cannot be input of the network, preprocess to vectorize English words is essential. In the stage of pre-training, first group(V , H^1) are modeled as RBM and we need binary vectors as its input.

Ignoring case, there are twenty-six letters ('a'-'z') in all so that each letter can be represented by 5 bits ($2^5 > 26$). For example, letter 'a' can be represented as [00001] while letter 'z' can be represented as [11010]. Because the maximum

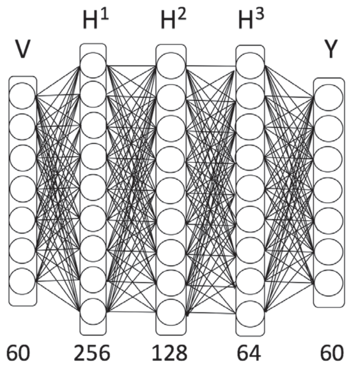

Fig. 4. 60-256-128-64-60 neural words encoding model

length of the string in training data is twelve, using a 60-dimensions binary vector can represent any English word in training data. For example, string ‘abc’ and ‘what’ can be represent by binary vector as follows:

$$'abc' \Rightarrow [00001\,00010\,00011\,00000\,00...00]_{60\times 1}$$
$$'what' \Rightarrow [10111\,01000\,00001\,11010\,00...00]_{60\times 1}$$

Here we use this method to vectorize input instead of word embedding or one-hot [16, 17] because we want to use a unique binary vector to represent each word whose dimension is as small as possible. This method can also be applied to other languages, for example, there are around 20000 characters in simplified Chinese and the length of every character is one, therefore a 15-dimension ($2^{15} > 20000$) binary vector is enough to present every single Chinese character.

By using this preprocess method, every English word of training data is converted into a 60-dimensions binary vector, finally we get training data X which is a matrix of 60×58194. We use X to train the model to obtain a set of parameters which can reconstruct training data by a series of unified non-linear transformation that can be used as encryption and decryption.

B. Training Model and Analysis

We used the proposed method to train a 60-256-128-64-60 neural words encoding model as Figure 4 shows. In pre-training stage, we trained three groups of RBM (60,256), (256,128) and (128,64) layer-by-layer, in fine-tuning stage, we set learning rate $\alpha = 0.03$ and use back propagation algorithm to train the whole model. For neuron number of the code layer is 64 whose output is binary vector, 64bits exactly meet the requirements of storage, which means, by using this model one English word which may contain twelve letters can be compressed and encrypted into a 64bits integer that can be exactly saved by unsigned long long in C++/Java, so a sentence can be encrypted into a series of integers as Figure 5 shows.

To evaluate the performance of the model, we define a parameter R called reconstruction rate as follows:

$$R = \sum_{i=1}^{|X|} \frac{\mathbb{1}\left\{\hat{y}^{(i)} = y^{(i)}\right\}}{|X|} \tag{14}$$

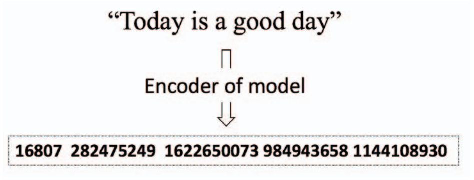

Fig. 5. An example of encrypting a sentence

TABLE I
PERFORMANCE OF MODELS

Model	Code layer binary output	R
Neural words encoding model	**Yes**	**99.04%**
RBM 60-128	Yes	85.46%
RBM 60-256	Yes	89.16%
Auto-encoder 120-128-120	No	89.75%
Auto-encoder 120-256-120	No	90.12%
DBNs 60-256-128-64-128-256-60	No	99.87%

Here $|X|$ is the number of words in training data and $y^{(i)}$ is the result of i_{th} training data, $\mathbb{1}\left\{\hat{y}^{(i)} = y^{(i)}\right\}$ is equal to 1 if output $\hat{y}^{(i)}$ and $y^{(i)}$ are identical and if there is any bit that $\hat{y}^{(i)}$ differs from $y^{(i)}$, $\mathbb{1}\left\{\hat{y}^{(i)} = y^{(i)}\right\}$ is equal to 0.

The reconstruction rate R of 60-256-128-64-60 neural words encoding model is 99.04%, we trained three kinds of another models to compare the performances of different models. The first kind model is a single layer RBM, the second is three layer auto-encoder [18, 19] without pre-training, and the third is DBNs which is proposed by Hinton [4] with pre-training, unfolding and fine-tuning. The results are shown in Table 1.

The reconstruction rate R of DBNs is also very good whose output of code layer is real-value, but when we added threshold function to DBNs's code layer to make output binary, we found it hard to converge and reconstruction rate decreased sharply.

From the compared results, it can be concluded that neural words encoding model has an excellent performance of encoding English words. We found it was also able to reconstruct some strings which are not English words or not in training data. We randomly generated two groups of strings to test reconstruction rate of the model, the first group contained 10,000 random strings whose lengths are between one and twelve. Because 80% of training data has length between five and ten, we randomly generated the second group which contained 10,000 random strings whose lengths are between five and ten. The results are shown in Table 2.

We can see this model is also able to encode and decode random strings. When we use this model, in case there is a word whose length is over twelve which is the maximum length of training data, we can split the word to several strings. For example, word ‘chemiluminescence’ can be split to ‘chemilumine’ and ‘scence’ , then we can use our model to encode each string and concatenate the encode of each part.

TABLE II
RECONSTRUCTION RATE OF RANDOM STRING

Test data	Reconstruct number	Reconstruction rate
Random string (length: 1-12)	8745/10000	87.45%
Random string (length: 5-10)	9671/10000	96.71%

IV. CONCLUSION

In this work, we trained a neural network model which can encode most English words to 64bits integers and with this model each 64bits integer can be decoded to an English word. Many random strings can also be encoded and decoded by the model. It realizes the function of encoding and decoding of English words by a series of unified non-linear transformation which can be applied to text encryption, decryption and compression.

Using the proposed model and training algorithm, other language word sets can also be encoded and decoded. By adding punctuations such as comma, period, question mark, and exclamation mark, etc in training data, the model can encode and decode text, it has advantages in fast compression/decompression of natural language text and words searching as well as word-based compression algorithm. In order to meet the needs of different applications, we can change neuron number of code layer to control the length of the representation code.

The future work of our research will focus on adding some restrictions to make encoding satisfy a certain distribution for different applications. For instance, make the encoding satisfy uniform distribution so that it can be used as the input of simHash [9, 10].

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation of China (Grant No. 61375065), partially supported by the State Key Program of National Science Foundation of China (Grant No. 61432012 and 61432014).

REFERENCES

[1] A. Moffat, "Word-based text compression," *Software: Practice and Experience*, vol. 19, no. 2, pp. 185–198, 1989.

[2] R. N. Horspool and G. V. Cormack, "Constructing word-based text compression algorithms." in *Data Compression Conference*, 1992, pp. 62–71.

[3] E. Silva de Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates, "Fast and flexible word searching on compressed text," *ACM Transactions on Information Systems (TOIS)*, vol. 18, no. 2, pp. 113–139, 2000.

[4] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[5] A. Fischer and C. Igel, "An introduction to restricted boltzmann machines," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Springer, 2012, pp. 14–36.

[6] G. E. Hinton, "Boltzmann machine," *Scholarpedia*, vol. 2, no. 5, p. 1668, 2007.

[7] Y. Bengio and O. Delalleau, "Justifying and generalizing contrastive divergence," *Neural computation*, vol. 21, no. 6, pp. 1601–1621, 2009.

[8] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[9] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, 2002, pp. 380–388.

[10] G. S. Manku, A. Jain, and A. Das Sarma, "Detecting near-duplicates for web crawling," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 141–150.

[11] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[12] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[13] J. C. Lv, Y. Zhang, and T. Kok Kiong, "Global convergence of gha learning algorithm with nonzero-approaching learning rates," *IEEE Transactions on Neural Networks (TNN)*, vol. 18, no. 6, pp. 1557–1571, 2007.

[14] J. C. Lv, T. Kok Kiong, Y. Zhang, and S. Huang, "Convergence analysis of a class of hyvärinen–oja's ica learning algorithms with constant learning rates," *IEEE Transactions on Signal Processing (TSP)*, vol. 57, no. 5, pp. 1811–1824, 2009.

[15] D. Ruhmelhart, G. Hinton, and R. Wiliams, "Learning representations by back-propagation errors," *Nature*, vol. 323, pp. 533–536, 1986.

[16] Y. Bengio, H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain, "Neural probabilistic language models," in *Innovations in Machine Learning*. Springer, 2006, pp. 137–186.

[17] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations." in *HLT-NAACL*, 2013, pp. 746–751.

[18] A. Ng, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, 2011.

[19] A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, and C. Suen, "Unsupervised feature learning and deep learning tutorial," 2011.