Assignment-14
RNNs, Unsupervised Learning and Applications

Harshvardhan Patidar
Department of Artificial Intelligence
Indian Institute of Technology Hyderabad
ai24btech11015@iith.ac.in

1 Recurrent Neural Networks

RNNs provide a way to make cycles in the computation graphs we had discussed till now. Each cycle has a delay, so that the output produced by that particular node can be a part of the input of the same node in the next cycle. RNN can perform more general computations instead of the boolean calculations which was the limit of the traditional feedforward neural networks. RNNs are mostly used to analyze sequential data, where a new input vector arrives at every timestep. RNNs add expressive power to the feedforward networks in the case of sequential data. If a feedforward network had been used, the input layer has a fixed size, because of which only a fixed length of input data would be analyzed, ignoring the rest of the data.

1.1 Training a basic RNN

For a basic model, we will consider it to have an input layer **x**, an hidden layer **z**, and an output layer **y**. We will observe both the input and output at each time step. The following equations give the values of variables at a time step t.

$$\mathbf{z}_t = g_z\left(\mathbf{W}_{z,z}\mathbf{z}_{t-1} + \mathbf{W}_{x,z}\mathbf{x}_t\right) \equiv g_z(\text{in}_{z,t})$$
$$\hat{\mathbf{y}}_t = g_y\left(\mathbf{W}_{z,y}\mathbf{z}_t\right) \equiv g_y(\text{in}_{y,t})$$

Here **g** are the activation function for the respective layer. One important and interesting observation is that we can express this RNN in the form of a feedforward network by unrolling the network for some T steps if the number of input vectors is T. The drawback for this feedforward network is that the gradient calculation will be comparatively more complicated, as in RNN the gradients are not needed to be calculated repeatedly.

For calculating and updating the weights of hidden layer, we need to find the gradient of the loss function w.r.t. these weights. The calculations are below:

$$
\begin{aligned}
\frac{\partial L}{\partial \mathbf{w}_{z,z}} &= \frac{\partial}{\partial \mathbf{w}_{z,z}} \sum_{t=1}^{T} (y_t - \hat{\mathbf{y}}_t)^2 \\
&= \sum_{t=1}^{T} -2(y_t - \hat{\mathbf{y}}_t) \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{w}_{z,z}} \\
&= \sum_{t=1}^{T} -2(y_t - \hat{\mathbf{y}}_t) \frac{\partial}{\partial \mathbf{w}_{z,z}} g_y(\text{in}_{y,t}) \\
&= \sum_{t=1}^{T} -2(y_t - \hat{\mathbf{y}}_t) g'_y(\text{in}_{y,t}) \frac{\partial}{\partial \mathbf{w}_{z,z}} \text{in}_{y,t} \\
&= \sum_{t=1}^{T} -2(y_t - \hat{\mathbf{y}}_t) g'_y(\text{in}_{y,t}) \frac{\partial}{\partial \mathbf{w}_{z,z}} (\mathbf{w}_{z,y}\mathbf{z}_t + \mathbf{w}_{0,y}) \\
&= \sum_{t=1}^{T} -2(y_t - \hat{\mathbf{y}}_t) g'_y(\text{in}_{y,t}) \mathbf{w}_{z,y} \frac{\partial \mathbf{z}_t}{\partial \mathbf{w}_{z,z}}
\end{aligned}
$$

Also, the gradient of the hidden unit z_t is calculated below.

$$
\begin{aligned}
\frac{\partial \mathbf{z}_t}{\partial \mathbf{w}_{z,z}} &= \frac{\partial}{\partial \mathbf{w}_{z,z}} g_z(\text{in}_{z,t}) \\
&= g'_z(\text{in}_{z,t}) \frac{\partial}{\partial \mathbf{w}_{z,z}} (\mathbf{w}_{z,z}\mathbf{z}_{t-1} + \mathbf{W}_{x,z}\mathbf{x}_t + \mathbf{w}_{0,z}) \\
&= g'_z(\text{in}_{z,t}) \left(\mathbf{z}_{t-1} + \mathbf{w}_{z,z} \frac{\partial \mathbf{z}_{t-1}}{\partial \mathbf{w}_{z,z}} \right) \qquad (22.15)
\end{aligned}
$$

The gradient calculations are recursive, meaning that we can calculate the gradient from time step t using the contribution of the step $t-1$. Right ordering of the calculations can make the complexity of the function linear with the size of the network. This algorithm is called back-propagation through time. This is usually executed automatically by the deep learning softwares.

1.2 Long short-term memory RNNs

LSTM has been designed with the goal to preserve information for many time steps. The long term memory component is called memory cell, denoted by **c**, and is copied from one time step to another. Apart from this, we have gating units, which control and check the flow of information from one step to another.

1. The Forget gate, f determines if all the memory cells of the previous time step are copied or not.
2. The input gate, i checks if each element in the input vectors has been considered and added to the memory cell.
3. The output gate, o determines if each of the elements from the memory cell is transferred back to the short term memory **z** (it is similar to the hidden layer in the basic RNNs).

The gate functions here are not boolean, instead they are soft and in the range of $[0, 1]$. This is useful in the cases when the elements of the memory cell are partially forgotten, when the values of the gate are smaller than 1, but not 0. The update equations are :

$$
\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_{x,f}\mathbf{x}_t + \mathbf{W}_{z,f}\mathbf{z}_{t-1}) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_{x,i}\mathbf{x}_t + \mathbf{W}_{z,i}\mathbf{z}_{t-1}) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{x,o}\mathbf{x}_t + \mathbf{W}_{z,o}\mathbf{z}_{t-1}) \\
\mathbf{c}_t &= \mathbf{c}_{t-1} \odot \mathbf{f}_t + \mathbf{i}_t \odot \tanh(\mathbf{W}_{x,c}\mathbf{x}_t + \mathbf{W}_{z,c}\mathbf{z}_{t-1}) \\
\mathbf{z}_t &= \tanh(\mathbf{c}_t) \odot \mathbf{o}_t
\end{aligned}
$$

2 Unsupervised Learning and Transfer Learning

The Supervised learning that we discussed till now is not the best possible method for all the real-world problems as it requires a lot of labelled data for the training purpose. Labelled data is often quite expensive as it would require expensive human labour to sit and label each of the examples in the data set. Unsupervised learning plays its role in such cases. Unsupervised learning algorithms learn from unlabelled data, which is available abundantly at low costs. It typically produces a generative model. Transfer learning and semi supervised learning are also such two algorithms, which require only a little labelled data, and later improved by learning from unlabelled data.

2.1 Unsupervised Learning

Unsupervised learning typically has the goal of making a generative model. Suppose we learn some joint model $P_W(\mathbf{x}, \mathbf{z})$. Here $\mathbf{z}$ is a set of some unobserved variables, which in some way represent the data or the content in $\mathbf{x}$. It is the choice of a model on how to associate $\mathbf{z}$ with $\mathbf{x}$. For example, if a model is trained on images of handwritten digits, then the model might choose to use one variable in $\mathbf{z}$ to represent the thickness of the stroke, another for ink color, another for background color, etc. In this way, a learned probability model $P_W(\mathbf{x}, \mathbf{z})$ achieves both representation learning and generative modeling. As it has constructed a meaningful $\mathbf{z}$ out of raw input, it achieves representation learning. On the other hand if we don't integrate z in the $P_W(\mathbf{x}, \mathbf{z})$, we get $P_W(\mathbf{x})$, which acts like an generative model.

2.1.1 Probabilistic PCA : A simple generative model

The simplest model whose form $P_W(\mathbf{x}, \mathbf{z})$ might take is probabilistic principal components analysis(PPCA). In this model, we choose $\mathbf{z}$ from a spherical gaussian, whose mean is non-zero.

$$P(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, \mathbf{I})$$

Then $\mathbf{x}$ is generated from this $\mathbf{z}$ by applying a weight matrix $\mathbf{W}$, and some spherical gaussian noise, to make it more robust.

$$P_{\mathbf{W}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \mathbf{W}\mathbf{z}, \sigma^2\mathbf{I})$$

Now, the weight matrix $\mathbf{W}$ will be learned by maximizing the likelihood of the data

$$P_{\mathbf{W}}(\mathbf{x}) = \int P_{\mathbf{W}}(\mathbf{x}, \mathbf{z})\, d\mathbf{z} = \mathcal{N}(\mathbf{x}; 0, \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I})$$

There are several methods to do the maximization. We can either use the gradient methods, or the iterative method of EM algorithms. This way once we learn $\mathbf{W}$, we can generate new data samples directly from the $P_W(\mathbf{x})$ using the above equation. We take the dimensionality of $\mathbf{z}$ to be much less than that of the input so that the model learns to explain the input well with fewer features.

2.1.2 Autoencoders

These are a type of models used in Unsupervised Learning. It has two parts. The first part maps input data $\mathbf{x}$ to a latent representation $\hat{\mathbf{z}}$ using a function f, and is called the encoder. The second part is decoder, which maps the latent representation $\hat{\mathbf{z}}$ back to the original input $\mathbf{x}$, using a function g. The goal is to train the model so that the decoding process produces output that is approximately equal to the original input.

For a simplest encoder, i.e. linear autoencoder, both f and g are linear functions having a shared weight matrix $\mathbf{W}$:

$$\hat{\mathbf{z}} = f(\mathbf{x}) = \mathbf{W}\mathbf{x} \tag{1}$$

$$\mathbf{x} = g(\hat{\mathbf{z}}) = \mathbf{W}^\top\hat{\mathbf{z}} \tag{2}$$

These equations show that the reconstruction process involves using the transpose of the weight matrix.

The objective is to minimize the squared error between the original input and the restored input, given by:

$$\sum_j |\mathbf{x}_j - g(f(\mathbf{x}_j))|^2 \tag{3}$$

The goal of this training process is to learn a matrix $\mathbf{W}$ such that the low-dimensional vector $\hat{\mathbf{z}}$ retains maximum possible information from the high-dimensional input data $\mathbf{x}$.

2.1.3 Deep Regressive Model

These are the models in which each element $\mathbf{x}_i$ of an input vector $\mathbf{x}$ is predicted on the basis of the other elements of the vector. If $\mathbf{x}$ is of fixed size, an AR model can be viewed as fully observable and might also be a fully connected Bayes net. In this framework, calculating the likelihood of a given data vector $\mathbf{x}$, predicting the value of some missing variable, and sampling a data vector from the model are all easy and straightforward tasks.

For timed data, an AR model of order k predicts $\mathbf{x}t$ given $\mathbf{x}t-k,\ldots,\mathbf{x}_{t-1}$. Mathematically, this is expressed as:

$$P(\mathbf{x}_t|\mathbf{x}_{t-k},\ldots,\mathbf{x}_{t-1})$$

In the classical models, where variables were real-valued, the conditional distribution $P(\mathbf{x}t|\mathbf{x}t-k,\ldots,\mathbf{x}t-1)$ was usually a linear-Gaussian model with constant variance. The mean was a weighted linear combination of $\mathbf{x}t-k,\ldots,\mathbf{x}_{t-1}$, similar to the standard linear regression model. The maximum likelihood for this model was given by the Yule–Walker equation.

In deep autoregressive models, our assumption of linear-Gaussian is replaced by a deep network which has a flexible and nonlinear structure. The output of the network depends on whether $\mathbf{x}_t$ is discrete or continuous. One famous application is WaveNet by DeepMind, which uses a deep autoregressive model for speech generation.

2.1.4 Generative Adversarial Networks

A GAN is a pair of networks that combine to form the complete model. The first is Generator, which maps a latent vector $\mathbf{z}$ to data space $\mathbf{x}$, producing some samples. And the second model is discriminator, which simply classifies the input as real or fake. Here, real means the ones given as input in data, and fake are the ones which are produced by the generator.

Both the models are trained in a competitive manner, where generators try to fool the discriminator by generating better data each time, and the discriminator tries to classify it correctly. At the end, the generator starts generating very similar data as the given input, and the discriminator starts to just randomly guess if it's real or fake. Such models are very successful in generating high-quality images.

2.1.5 Unsupervised Translation

The main task of such models is to transform a multi-dimensional input **x** to a multi-dimensional output **y**. For example, converting a night image to a day image as if it was taken in day only. In the case of supervised translation, we gather (**x,y**) and then learn from these, but this data is not alway available. In these cases we use Unsupervised Translation. This is accomplished by training two GANs, first one is trained to generate output **y** out of the input **x**, and the other does the reverse.

2.2 Transfer Learning and Multitask Learning

2.2.1 Transfer Learning

It basically means using already learned models to learn some new similar task. For example if someone already knows how to play tennis, he will find learning to play badminton easy. This is accomplished by simply copying the weights of an already trained model for some task A, to a new model and then fine tuning those weights so that it fits for some task B. Human intervention

is required to choose which pretrained model to use for the new model which is to be developed. RoBERTA is a common pretrained model which serves as a starting point for developing natural language models.

2.2.2 Multitask Learning

It is nothing but a specialized form of transfer learning in which we simultaneously train the model on multiple tasks rather than copying the weights to a new model, and then further training that model. Learning all the tasks simultaneously helps it to improve and perform better than what it would have when trained individually for those tasks.

3 Applications

Deep learning has been applied in various important areas in AI. Some of them are

3.1 Vision

Computer vision is the field which has been most impacted by Deep Learning, especially after the victory of AlexNet in the 2012 ImageNet competition. AlexNet was based on deep learning, and that's why its victory led to the focus of researchers to shift to deep learning. We have improved a lot in computer vision by improvements in all the tasks like network design , training methods, and compute resources. The top-5 error rate has come down to less than 2%, much less than that of a well trained human (5%). CNNs have been used for self-driving cars, grading cucumbers, etc.

3.2 Natural Language Processing

NLP applications such as machine translation and speech recognition are also impacted by Deep Learning, which has allowed the possibility of end-to-end learning, automatic generation of internal layers/representations for words. For example, for the simple task of translation of a sentence from one language to another, earlier, separate models were used to accomplish the tasks by using the pipeline method. But now, a single trained model does the complete task reducing the possibility of error by a great factor. In Word embeddings, the internal meaning of the words is extracted from a learned model into some high-dimensional vector. The words with similar meanings are placed close to each other, which helps the model to make better predictions for the next word of some statement depending on the context.

3.3 Reinforcement Learning

In RL, an agent is made to learn by giving a sequence of rewards based on the quality of its behavior. The goal is to optimize the sum of the future rewards. Deep Reinforcement Learning is the field of research in which deep learning is used for reinforcement learning. DeepMinds's DQN and AlphaGO are two successful examples of it. Still it is difficult to implement RL, as it is difficult to obtain good performance, and the models may perform very unpredictably in slightly modified environments.