Deep Learning

AlejandroMllo

This document serves as a very brief summary of the topics covered in each chapter of the book Deep Learning [1].

Disclaimer: This document is completely extracted from [1], the author does not attribute any ownership over the material.

10 Sequence Modeling: Recurrent and Recursive Nets

- Recurrent neural networks (RNNs) are a family of NN for processing sequential data. They can scale to much longer sequences than unspecialized NN and can also process sequences of variable length.
- RNNs share parameters across different parts of a model to extend and apply the model to examples of different forms and generalize across them. - Each member of the output is produced using the same update rule applied to previous outputs.
- RNNs operate on sequences that contain vectors $\boldsymbol{x}^{(t)}$ with the time step index t ranging from 1 to τ. In practice, they usually operate on minibatches of such sequences, with a different sequence length τ for each member of the minibatch.
- A computational graph is a way to formalize the structure of a set of computations.
- The classical form of a dynamical system $\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta})$, where $\boldsymbol{s}^{(t)}$ is the state of the system, is recurrent because the definition of $\boldsymbol{s}$ at time t refers back to the same definition at time $t-1$.
- The network typically learns to use the state as a kind of lossy summary of the task-relevant aspects of the past sequence of inputs up to t.
- It is possible to represent the unfolded recurrence after t steps: $\boldsymbol{h}^{(t)} = g^{(t)}(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \cdots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}) = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$, where the function$g^{(t)}$ takes the whole past sequence as input and produces the current state.
- Design patterns for RNNs include:
  - RNN that produce an output at each time step and have recurrent connections between hidden units. Any function computable by a Turing machine can be computed by this network.
    Forward propagation begins with a specification of the initial state $\boldsymbol{h}^{(0)}$. Then, for each time step from $t = 1$ to $t = \tau$, the following updates are applied:

    $$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)},$$

    $$\boldsymbol{h}^{(t)} = \tanh \boldsymbol{a}^{(t)},$$

    $$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)},$$

    $$\hat{\boldsymbol{y}}^{(t)} = softmax(\boldsymbol{o}^{(t)}).$$

    The total loss for a given sequence of $\boldsymbol{x}$ values paired with a sequence of $\boldsymbol{y}$ values would the just be the sum of the losses over all the time steps.
  - RNN that produce an output at each time step and have recurrent connections only from the output at one timestep to the hidden units at the next time step. This option is less powerful because it lacks hidden-to-hidden recurrent connections. Training can be parallelized.
  - RNN with recurrent connections between hidden units, that read an entire sequence and then produce a single output.
- Teacher forcing is a procedure that emerges from the maximum likelihood criterion, in which during training the model receives the ground truth output $y^{(t)}$ as input at time $t + 1$. Its disadvantage arises if the network is going to be later used in an open-loop mode, with the network outputs fed back as input.
- When using a predictive log-likelihood training objective, the RNN is trained to estimate the conditional distribution of the next sequence element $\boldsymbol{y}^{(t)}$ given the past inputs.

- It is possible to view the RNN as defining a graphical model whose structure is the complete graph, able to represent direct dependencies between any pair of y values. Every past observation $y^{(i)}$ may influence the conditional distribution of some $y^{(t)}$ (for $t > i$), given the previous values.
- The parameter sharing used in RNNs relies on the assumption that the same parameters can be used for different time steps.
- Some common ways of providing an extra input to an RNN are: as an extra input at each time step, or as the initial state $\boldsymbol{h}^{(0)}$, or both.
- Up to now RNNs have a casual structure, meaning that the state at time t captures only information from the past, $\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(t-1)}$, and the present input $\boldsymbol{x}^{(t)}$. Also, some models allow information from past $\boldsymbol{y}$ values to affect the current state when the $\boldsymbol{y}$ values are available.
- Many applications need to output a prediction of $\boldsymbol{y}^{(t)}$ that may depend on the whole input sequence. Bidirectional RNNs were invented to address that need. They combine an RNN that moves forward through time, beginning from the start of the sequence, with another RNN that moves backward through time, beginning from the end of the sequence.
- An RNN can be trained to map an input sequence to an output sequence which is not necessarily of the same length.
- The input to an RNN is sometimes called the "context". The objective is to produce a representation of this context, C. C might be a vector or sequence of vectors that summarize the input sequence $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(n_x)})$.
- Encoder-decoder or sequence-to-sequence RNN architecture: it is composed of an encoder RNN that reads the input sequence as well as a decoder RNN that generates the output sequence. The final hidden stat of the encoder RNN is used to compute a generally fixed-size context variable C, which represents a semantic summary of the input sequence and is given as input to the decoder RNN. The two RNNs are trained jointly to maximize the average of $\log P(\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{(n_y)} | \boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(n_x)})$ over all the pairs of $\boldsymbol{x}$ and $\boldsymbol{y}$ sequences in the training set. The lenghts of n_x and n_y can vary from each other.
- The computation in most RNNs can be decomposed into three blocks of parameters and associated transformations:
  1. From the input to the hidden state,
  2. From the previous hidden state to the next hidden state, and
  3. From the hidden state to the output.
- Recursive neural networks represent other generalization of RNNs, which is structured as a deep tree, rather than the chain-like structure of RNNs.
- A clear advantage of recursive nets over recurrent nets is that for a sequence of the same length τ, the depth (measured as the number of compositions of nonlinear operations) can be drastically reduced from τ to $O(\log \tau)$.
- Recurrent networks involve the composition of the same function multiple times, once per time step, which can result in extremely nonlinear behavior.
- It is possible to think of the recurrence relation $\boldsymbol{h}^{(t)} = \boldsymbol{W}^{\top} \boldsymbol{h}^{(t-1)}$ as a very simple RNN lacking of nonlinear activation function, and lacking inputs $\boldsymbol{x}$.
- The recurrent weights mapping from $\boldsymbol{h}^{(t-1)}$ to $\boldsymbol{h}^{(t)}$ and the input weights mapping from $\boldsymbol{x}^{(t)}$ to $\boldsymbol{h}^{(t)}$ are some of the most difficult parameters to learn in a recurrent network. One proposed approach to avoiding this difficulty is to set the recurrent weights such that the recurrent hidden units do a good job of capturing the history of past inputs, and only learn the output weights. This idea is proposed for echo state networks (ESN) or liquid state machines.
- Reservoir computing: denote the fact that the hidden units form a reservoir of temporal features that may capture different aspects of the history of inputs.
- To represent a rich set of histories in the RNN, view the recurrent net as a dynamical system, and set the input and recurrent weights such that the dynamical system is near the edge of stability.
- Everything about backpropagation via repeated matrix multiplication applies equally to forward propagation in a network with no nonlinearity, where the state $\boldsymbol{h}^{(t+1)} = \boldsymbol{h}^{(t)\top} \boldsymbol{W}$.
- One way to deal with long-term dependencies is to design a model that operates at multiple time scales, so that some parts of the model operate at fine-grained time scales and can handle small details, while other parts operate at coarse time scales and transfer information from the distant past to the present more efficiently.
- One way to obtain coarse time scales is to add direct connections from variables in the distant past to variables in the present.

- Another way to obtain paths on which the product of derivatives is close to one is to have units (known as leaky units) with linear self-connections and a weight near one on these connections.
- Leaky units allow the network to accumulate information over a long duration. Once that information has been used, however, it might be useful for the NN to forget the old state.
- Other approach to handling long-term dependencies involves actively removing length-one connections and replacing them with longer connections.
- The most effective sequence models used in practical applications are called gated RNNs. This include long short-term memory (LSTM) and networks based on the gated recurrent unit.
- Gated RNNs are based on the idea of creating paths through time that have derivatives that neither vanish nor explode.
- LSTM model: it uses self-loops to produce paths were the gradient can flow for long durations; also the weight on this self-loop is conditioned on the context, rather than fixed. By making the weight of this self-loop gated (controlled by another hidden unit), the time scale of integration can be changed dynamically.
- Instead of a unit that simply applies an element-wise nonlinearity to the affine transformation of inputs and recurrent units, LSTM recurrent networks have "LSTM cells" that have an internal recurrence (a self-loop), in addition to the outer recurrence of the RNN. Each cell has the same inputs and outputs as an ordinary recurrent network, but also has more parameters and a system of gating units that controls the flow of information. The most important component is the state unit $s_i^{(t)}$, which has a linear self-loop similar to the leaky units. The self-loop weight (or the associated time constant) is controlled by a forget gate unit $f_i^{(t)}$ (for time step t and cell i), which sets the weight via a sigmoid unit. Section 10.10.1 presents the corresponding equations.
- A single gating unit simultaneously controls the forgetting factor and the decision to update the state unit.
- It is often much easier to design a model that is easy to optimize than it is to design a more powerful optimization algorithm.
- Clipping gradients: this is used when the parameter gradient is very large, such that a gradient descent parameter update could throw the parameters very far into a region where the objective function is larger, undoing much of the work that had been done to reach the current solution. This helps to deal with exploding gradients.
- An idea to deal with vanishing gradients is to regularize or constrain the parameters so as to encourage "information flow". In particular, the objective is that the gradient vector $\nabla_{\boldsymbol{h}^{(t)}} L$ being back-propagated maintains its magnitude, even if the loss function only penalizes the output at the end of the sequence.
- NN excel at storing implicit knowledge, but they struggle to memorize facts.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.