paperswithcode.com

ICLR (download paper)

arxiv.org --> paper

icml.cc

# *Modern Recurrent Neural Networks*

DR. RISMAN ADNAN

TELKOM UNIVERSITY

Perbedaan utama antara GRU dan LSTM adalah bahwa GRU memiliki dua gerbang (reset dan memperbarui gerbang) sedangkan LSTM memiliki tiga gerbang (yaitu input , output dan gerbang lupa ).

GRU lebih sederhana dan karenanya lebih mudah untuk dimodifikasi, misalnya menambahkan gerbang baru jika input tambahan ke jaringan. Itu hanya sedikit kode pada umumnya.

LSTM secara teori harus mengingat urutan yang lebih panjang dari GRU dan mengungguli mereka dalam tugas yang membutuhkan pemodelan hubungan jarak jauh.

# *Outline*

- Gated Recurrent Units (GRU)
- Long Short-Term Memory (LSTM)
- Deep Recurrent Neural Networks
- Bidirectional Recurrent Neural Networks
- Machine Translation and Dataset
- Encoder-Decoder Architecture
- Sequence to Sequence Learning

Tujuan utama dari LSTM pada kasus peramalan (forecasting) adalah membuat prediksi.
LSTM merupakan salah satu jenis dari Recurrent Neural Network (RNN) dimana dilakukan modifikasi pada RNN dengan menambahkan memory cell yang dapat menyimpan informasi untuk jangka waktu yang lama (Manaswi, 2018).
LSTM diusulkan sebagai solusi untuk mengatasi terjadinya vanishing (menghilang) gradient pada RNN saat memproses data sequential yang panjang.

GRU: Learn predictive, classification, clustering, and information retrieval (mendapatkan kembali) ML algorithms.
Classification uses predefined classes in which objects are assigned, while clustering identifies similarities between objects.
Retrieval: the process of getting something back from somewhere

*Gated Recurrent Units (GRU)*

- The key distinction between vanilla RNNs and GRUs is that the latter support **gating of the hidden state**.

  perbedaan
  ?
  yang terakhir
  gerbang

- GRU has dedicated mechanisms for when a hidden state should be *updated* and also when it should be *reset*.

- These mechanisms are **learned** and they address the concerns listed above.

  menangani

- For instance, if the first token is of great importance we will learn not to update the hidden state after the first observation.

- Likewise, we will learn to **skip irrelevant temporary observations**. Last, we will learn to reset the latent state whenever needed.

  terpendam

## Benefits of GRU

- Gated RNNs can better capture dependencies for sequences with large time step distances.

- Reset gates help capture short-term dependencies in sequences.

- Update gates help capture long-term dependencies in sequences.

- GRUs contain basic RNNs as their extreme case whenever the reset gate is switched on. They can also skip subsequences by turning on the update gate.

# *GRU Architecture*

- Reset gates help capture short-term dependencies in sequences.

- Update gates help capture long-term dependencies in sequences.
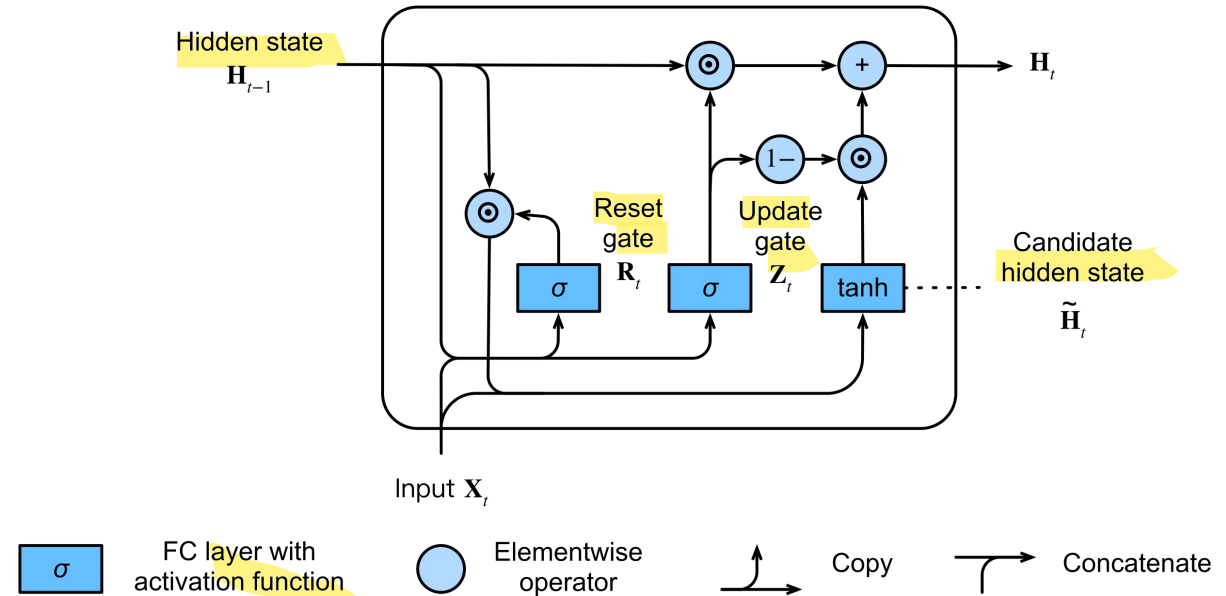


$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$

$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z),$$

$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h),$$

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t.$$

# Long Short Term Memory (LSTM)

to address: untuk mengatasi

The challenge to address long-term information preservation and short-term input skipping in latent variable models has existed for a long time.

One of the earliest approaches to address this was the long short-term memory (LSTM). It shares many of the properties of the GRU. Interestingly, LSTMs have a slightly more complex design than GRUs but predates GRUs by almost two decades.

predates: mendahului
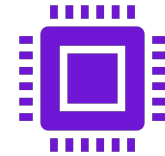
Arguably LSTM's design is inspired by logic gates of a computer. LSTM introduces a *memory cell* (or *cell* for short) that has the same shape as the hidden state (some literatures consider the memory cell as a special type of the hidden state), engineered to record additional information.

# LSTM Architecture



- LSTMs have three types of gates: input gates, forget gates, and output gates that control the flow of information.

- The hidden layer output of LSTM includes the hidden state and the memory cell. Only the hidden state is passed into the output layer. The memory cell is entirely internal.

- LSTMs can alleviate vanishing and exploding gradients.
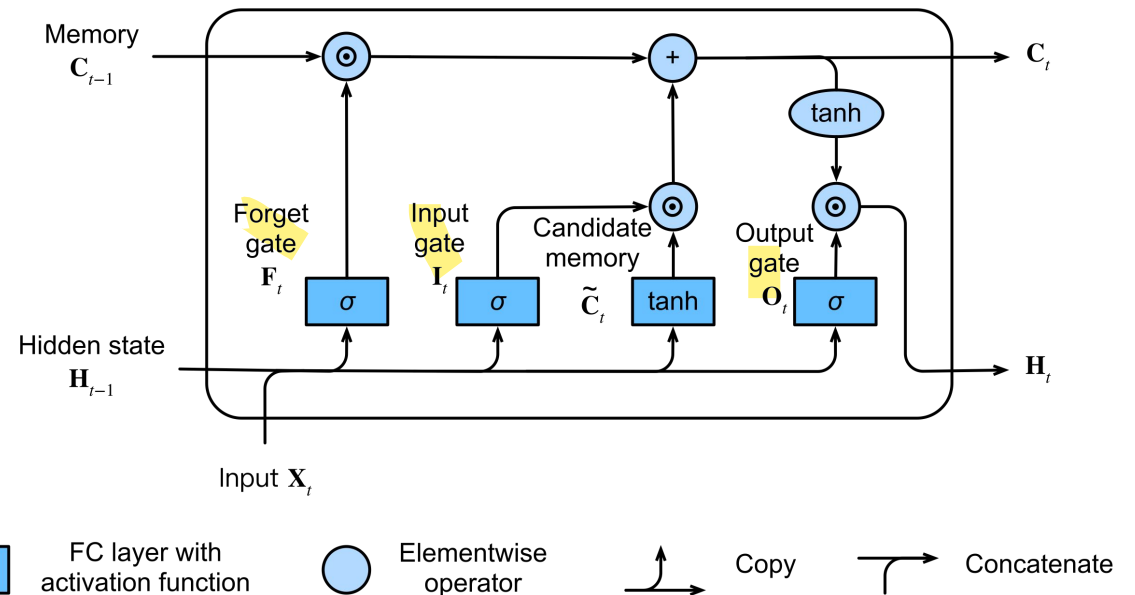
$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i),$$
$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f),$$
$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o),$$
$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c),$$
$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t.$$
$$H_t = O_t \odot \tanh(C_t).$$

# *Deep Recurrent Neural Networks*

latent: terpendam;     arbitrary: sewenang-wenang

- Up to now, we only discussed RNNs with a single unidirectional hidden layer. In it the specific functional form of how latent variables and observations interact is rather arbitrary. This is not a big problem as long as we have enough flexibility to model different types of interactions. With a single layer, however, this can be quite challenging. In the case of the linear models, we fixed this problem by adding more layers. Within RNNs this is a bit trickier, since we first need to decide how and where to add extra nonlinearity.

stack: tumpukan

- In fact, we could stack multiple layers of RNNs on top of each other. This results in a flexible mechanism, due to the combination of several simple layers. In particular, data might be relevant at different levels of the stack. For instance, we might want to keep high-level data about financial market conditions (bear or bull market) available, whereas at a lower level we only record shorter-term temporal dynamics.

# DRNN Architecture

- In deep RNNs, the hidden state information is passed to the next time step of the current layer and the current time step of the next layer.

  <span style="color:blue">conveniently: dgn baik sekali</span>

- There exist many different flavors of deep RNNs, such as LSTMs, GRUs, or vanilla RNNs. Conveniently these models are all available as parts of the high-level APIs of deep learning frameworks.

- Initialization of models requires care. Overall, deep RNNs require considerable amount of work (such as learning rate and clipping) to ensure proper convergence.
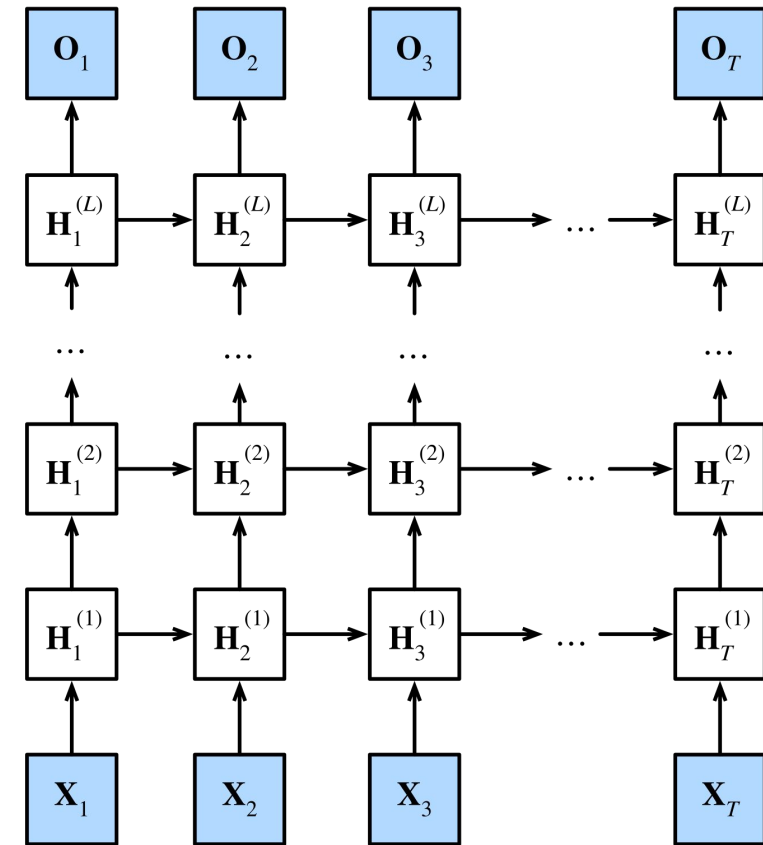


$$\mathbf{H}_t^{(l)} = \phi_l(\mathbf{H}_t^{(l-1)}\mathbf{W}_{xh}^{(l)} + \mathbf{H}_{t-1}^{(l)}\mathbf{W}_{hh}^{(l)} + \mathbf{b}_h^{(l)}),$$

$$\mathbf{O}_t = \mathbf{H}_t^{(L)}\mathbf{W}_{hq} + \mathbf{b}_q,$$

# Bidirectional Recurrent Neural Networks

- In sequence learning, so far we assumed that our goal is to model the next output given what we have seen so far, e.g., in the context of a time series or in the context of a language model. While this is a typical scenario, it is not the only one we might encounter. To illustrate the issue, consider the following three tasks of filling in the blank in a text sequence:

    - I am ___.
    - I am ___ hungry.
    - I am ___ hungry, and I can eat half a pig.

- Depending on the amount of information available, we might fill in the blanks with very different words such as "happy", "not", and "very". Clearly the end of the phrase (if available) conveys significant information about which word to pick. A sequence model that is incapable of taking advantage of this will perform poorly on related tasks. For instance, to do well in named entity recognition (e.g., to recognize whether "Green" refers to "Mr. Green" or to the color) longer-range context is equally vital. To get some inspiration for addressing the problem let us take a detour to probabilistic graphical models.

    detour: a long or roundabout (tdk langsung) route that is taken to avoid something or to visit somewhere along the way
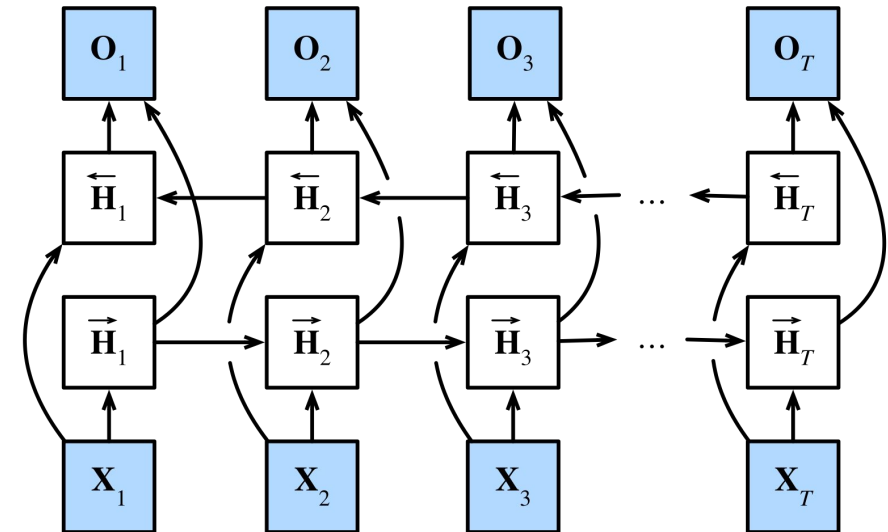
    convey: menyampaikan

# DP in Markov Models

- This subsection serves to illustrate the dynamic programming problem. The specific technical details do not matter for understanding the deep learning models but they help in motivating why one might use deep learning and why one might pick specific architectures.

- If we want to solve the problem using probabilistic graphical models we could for instance design a latent variable model as follows. At any time step $t$t, we assume that there exists some latent variable $h_t$ht that governs our observed emission $x_t$xt via $P(x_t|h_t)$P(xt|ht). Moreover, any transition $h_t \rightarrow h_{t+1}$ht→ht+1 is given by some state transition probability $P(h_{t+1}|h_t)$P(ht+1|ht). This probabilistic graphical model is then a *hidden Markov model* as

governs: mengatur
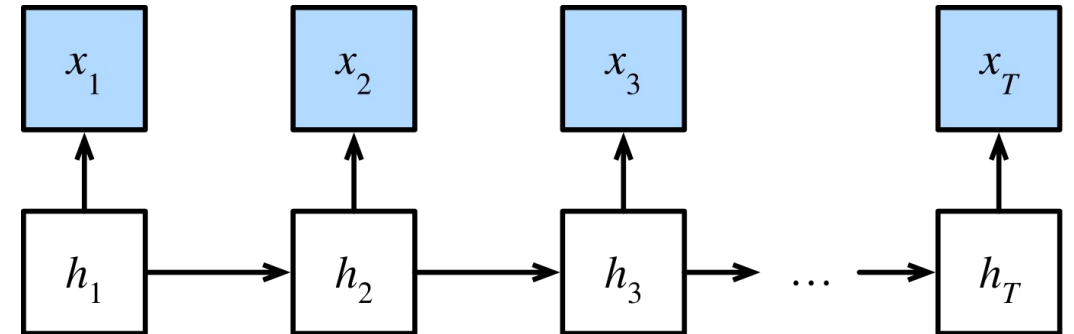
# Bidirectional Model

- If we want to have a mechanism in RNNs that offers comparable look-ahead ability as in hidden Markov models, we need to modify the RNN design that we have seen so far. Fortunately, this is easy conceptually. Instead of running an RNN only in the forward mode starting from the first token, we start another one from the last token running from back to front. *Bidirectional RNNs* add a hidden layer that passes information in a backward direction to more flexibly process such information.
devoid: tanpa
- In fact, this is not too dissimilar to the forward and backward recursions in the dynamic programming of hidden Markov models. The main distinction is that in the previous case these equations had a specific statistical meaning. Now they are devoid of such easily accessible interpretations and we can just treat them as generic and learnable functions. This transition epitomizes many of the principles guiding the design of modern deep networks: first, use the type of functional dependencies of classical statistical models, and then parameterize them in a generic form.

- In bidirectional RNNs, the hidden state for each time step is simultaneously determined by the data prior to and after the current time step.

- Bidirectional RNNs bear a striking resemblance with the forward-backward algorithm in probabilistic graphical models.

- Bidirectional RNNs are mostly useful for sequence encoding and the estimation of observations given bidirectional context.

- Bidirectional RNNs are very costly to train due to long gradient chains.

$$\overrightarrow{\mathbf{H}}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh}^{(f)} + \overrightarrow{\mathbf{H}}_{t-1} \mathbf{W}_{hh}^{(f)} + \mathbf{b}_h^{(f)}),$$
$$\overleftarrow{\mathbf{H}}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh}^{(b)} + \overleftarrow{\mathbf{H}}_{t+1} \mathbf{W}_{hh}^{(b)} + \mathbf{b}_h^{(b)}),$$
$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q.$$

# Machine Translation

- *Machine translation* refers to the automatic translation of a sequence from one language to another. In fact, this field may date back to 1940s soon after digital computers were invented, especially by considering the use of computers for cracking language codes in World War II. For decades, statistical approaches had been dominant in this field before the rise of end-to-end learning using neural networks. The latter is often called *neural machine translation* to distinguish itself from *statistical machine translation* that involves statistical analysis in components such as the translation model and the language model. corpus: kumpulan tulisan

- Emphasizing end-to-end learning, this book will focus on neural machine translation methods. Different from our language model problem whose corpus is in one single language, machine translation datasets are composed of pairs of text sequences that are in the source language and the target language, respectively. Thus, instead of reusing the preprocessing routine for language modeling, we need a different way to preprocess machine translation datasets. In the following, we show how to load the preprocessed data into minibatches for training.
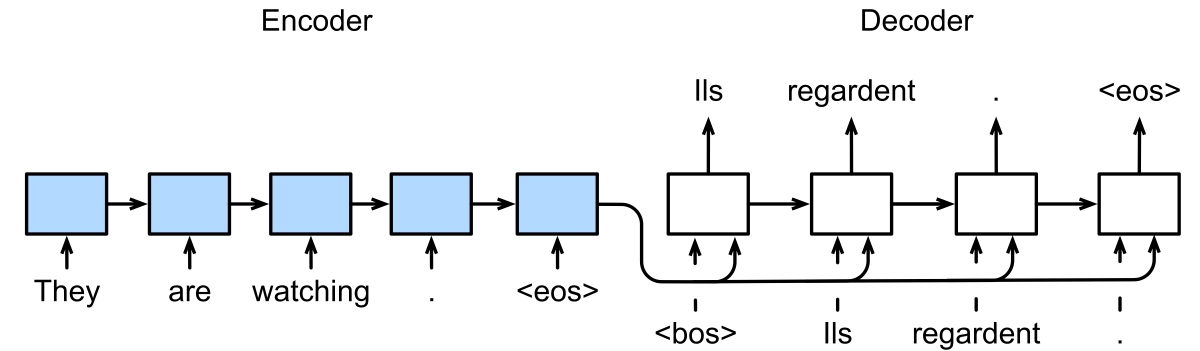
# Encoder-Decoder Architecture

- Machine translation is a major problem domain for sequence transduction models, whose input and output are both variable-length sequences. To handle this type of inputs and outputs, we can design an architecture with two major components. The first component is an *encoder*: it takes a variable-length sequence as the input and transforms it into a state with a fixed shape. The second component is a *decoder*: it maps the encoded state of a fixed shape to a variable-length sequence. This is called an *encoder-decoder* architecture

- Let us take machine translation from English to French as an example. Given an input sequence in English: "They", "are", "watching", ".", this encoder-decoder architecture first encodes the variable-length input into a state, then decodes the state to generate the translated sequence token by token as the output: "Ils", "regardent", ".". Since the encoder-decoder architecture forms the basis of different sequence transduction models in subsequent sections, this section will convert this architecture into an interface that will be implemented later.

- The encoder-decoder architecture can handle inputs and outputs that are both variable-length sequences, thus is suitable for sequence transduction problems such as machine translation.

- The encoder takes a variable-length sequence as the input and transforms it into a state with a fixed shape.

- The decoder maps the encoded state of a fixed shape to a variable-length sequence.

Input → Encoder → State → Decoder → Output
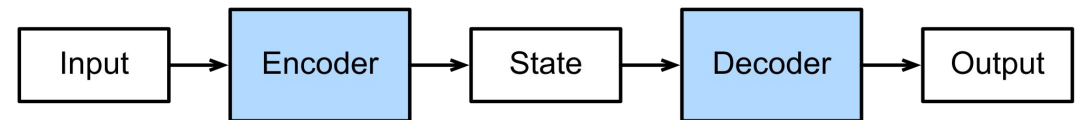
# *Sequence to Sequence Learning*

- S2S will use two RNNs to design the encoder and the decoder of this architecture and apply it to *sequence to sequence* learning for machine translation [Sutskever et al., 2014][Cho et al., 2014b].

- Following the design principle of the encoder-decoder architecture, the RNN encoder can take a variable-length sequence as the input and transforms it into a fixed-shape hidden state. In other words, information of the input (source) sequence is *encoded* in the hidden state of the RNN encoder. To generate the output sequence token by token, a separate RNN decoder can predict the next token based on what tokens have been seen (such as in language modeling) or generated, together with the encoded information of the input sequence.

- To predict the output sequence token by token, at each decoder time step the predicted token from the previous time step is fed into the decoder as an input. Similar to training, at the initial time step the beginning-of-sequence ("<bos>") token is fed into the decoder.

- When implementing the encoder and the decoder, we can use multilayer RNNs.

- We can use masks to filter out irrelevant computations, such as when calculating the loss.

- In encoder-decoder training, the teacher forcing approach feeds original output sequences (in contrast to predictions) into the decoder.

- BLEU is a popular measure for evaluating output sequences by matching $n$-grams between the predicted sequence and the label sequence.

Encoder                                                      Decoder

Ils    regardent    .    <eos>

They    are    watching    .    <eos>

<bos>    Ils    regardent    .

# *Home Work*

- Encoder Decoder
- Sequence to Sequence