Applied Machine Learning

Neural Networks for Sequences

Isabeau Prémont-Schwarz
Deep Neural Networks

- Neural Networks for **Tabular Data**
  - MLP
- Neural Networks for **Images**
  - CNN
- Neural Networks for **Sequences**
  - input is a sequence, the output is a sequence, or both are sequences
    - *e.g. machine translation, speech recognition, text classification, image captioning*
Learning objectives

• Recurrent neural networks (RNNs)
  ▪ 3 different models for different input/output
  ▪ training with back propagation through time
• understand the attention mechanisms
• The architecture of transformer
Recurrent neural networks (RNNs)

maps sequences to sequences in a stateful way

i.e. prediction $\hat{y}_t$ depends on $x_t$ and hidden state of the network $h_t$, which is updated over time

- Vec2Seq (sequence generation)
- Seq2Vec (sequence classification)
- Seq2Seq (sequence translation)
Recurrent neural networks (RNNs)

- Vec2Seq (sequence generation)
  - output, $y_{1:T}$ is generated one token at a time
  - at each step we sample $y_t$ from the hidden state $h_t$
    and then feed it back to the model to get $h_{t+1}$

conditional generative model:

$$p(y_{1:T} | x) = \sum_{h_{1:T}} p(y_{1:T}, h_{1:T} | x) = \sum_{h_{1:T}} \prod_{t=1}^{T} p(y_t | h_t) p(h_t | h_{t-1}, y_{t-1}, x)$$

with the initial hidden state $p(h_1 | h_0, y_0, x) = p(h_1 | x)$
Recurrent neural networks (RNNs)

- Vec2Seq (sequence generation)

**Conditional generative model:**

\[
p(y_{1:T} | x) = \sum_{h_{1:T}} p(y_{1:T}, h_{1:T} | x) = \sum_{h_{1:T}} \prod_{t=1}^{T} p(y_t | h_t) p(h_t | h_{t-1}, y_{t-1}, x)
\]

- Real-valued output:
  \[
  \hat{y}_t = W_{hy} h_t
  \]
  \[
  p(y_t | h_t) = \mathcal{N}(y_t | \hat{y}_t, \mathbf{I})
  \]

- Categorical output:
  \[
  \hat{y}_t = \text{softmax}(W_{hy} h_t)
  \]
  \[
  p(y_t | h_t) = \text{Categorical}(y_t | \hat{y}_t)
  \]

**Per usual dropping bias terms for simplicity**
Recurrent neural networks (RNNs)

- Vec2Seq (sequence generation)

conditional generative model:

\[
p(y_{1:T}|x) = \sum_{h_{1:T}} p(y_{1:T}, h_{1:T}|x) = \sum_{h_{1:T}} \prod_{t=1}^{T} p(y_t|h_t) p(h_t|h_{t-1}, y_{t-1}, x)
\]

hidden state:

\[
h_t = \varphi(W_{xh} [x; y_{t-1}] + W_{hh} h_{t-1})
\]
Recurrent neural networks (RNNs)

- Vec2Seq (sequence generation)

\[
\begin{align*}
\hat{y}_t &= g(W_{hy} h_t) \\
h_t &= \varphi(W_{xh}[x; y_{t-1}] + W_{hh} h_{t-1})
\end{align*}
\]

RNNs are powerful

- In theory can have unbounded memory and are as powerful as a Turing machine
- In practice, memory size is determined by the size of the latent space and strength of the parameters

\[f_\theta: \mathbb{R}^D \rightarrow \mathbb{R}^{TC}\]
Recurrent neural networks (RNNs)

- Vec2Seq (sequence generation)

**Conditional generative model:**

\[
p(y_{1:T} | x) = \sum_{h_{1:T}} p(y_{1:T}, h_{1:T} | x) = \sum_{h_{1:T}} \prod_{t=1}^{T} p(y_t | h_t) p(h_t | h_{t-1}, y_{t-1}, x)
\]

**Language modelling:** generating sequences unconditionally (by setting \( x = \emptyset \)) which is learning joint probability distributions over sequences of discrete tokens, i.e., \( p(y_1, \ldots, y_T) \)

**Example:**

Character level RNN trained on the book *The Time Machine* by H. G. Wells (32,000 words and 170k character)

Output when given prefix "the":

*the githa some thong the time traveller held in his hand was a glitteringmetallic framework scarcely larger than a small clock and verydelicately made there was ivory in it and the latter than s bettyre tat howhong s ie time thave ler simk you a dimensions le ghat dionthat shall travel indifferently in any direction of space and timeas the driver determinesfilby contented himself with laughterbut i have experimental verification said the time travellerit would be remarkably convenient for the histo*

See the code [here](#), read more [here](#)
Recurrent neural networks (RNNs)

- Vec2Seq (sequence generation)

*conditional generative model:*

\[ p(y_{1:T} | x) = \sum_{h_{1:T}} p(y_{1:T}, h_{1:T} | x) \]

**Example:**

CNN-RNN model for image captioning when \( x \) is embedding by a CNN

See more [here](#)
Recurrent neural networks (RNNs)

- **Seq2Vec (sequence classification)**
  - predict a single fixed-length output vector given a variable length sequence as input
  \[ y \in \{1, \ldots, C\} \]
  - use the final state:
  \[ \hat{y} = \text{softmax}(W h_T) \]
  \[ p(y|x_{1:T}) = \text{Categorical}(y|\hat{y}) \]

Bi-directional RNN:
- the hidden states of the RNN depend on the past and future context
- gives better results

\[ f_\theta : \mathbb{R}^{TD} \rightarrow \mathbb{R}^C \]
Recurrent neural networks (RNNs)

- Seq2Vec (sequence classification)
  - predict a single fixed-length output vector given a variable length sequence as input

  \[
  h_t^\rightarrow = \varphi \left( W_{xh} x_t + W_{hh} h_{t-1}^\rightarrow \right)
  \]

  \[
  h_t^\leftarrow = \varphi \left( W_{xh} x_t + W_{hh} h_{t+1}^\leftarrow \right)
  \]

  \[
  h_t = [h_t^\rightarrow, h_t^\leftarrow]
  \]

  \[
  \bar{h} = \frac{1}{T} \sum_{t=1}^{T} h_t
  \]

  \[
  \hat{y} = \text{softmax}(W \bar{h})
  \]

  \[
  p(y|x_{1:T}) = \text{Categorical}(y|\hat{y})
  \]

Bi-directional RNN:
the hidden states of the RNN depend on the past and future context

This gives better results
Recurrent neural networks (RNNs)

- Seq2Vec (sequence classification)
  - predict a single fixed-length output vector given a variable length sequence as input

**Example:**

Sentiment classification with word level **bidirectional** LSTM trained on a subset of the Internet Movie Database (IMDB) reviews. (20k positive and 20k negative examples)

Prediction examples for two inputs:
- 'this movie is so great' ⇒ 'positive'
- 'this movie is so bad' ⇒ 'negative'

see the code [here](#), and read more [here](#)
Recurrent neural networks (RNNs)

- Seq2Seq (sequence translation)
  - aligned: $T = T'$
  - unaligned: $T \neq T'$

$$f_\theta : \mathbb{R}^{TD} \rightarrow \mathbb{R}^{T'C}$$
Recurrent neural networks (RNNs)

- Seq2Seq (sequence translation)
  - aligned: $T = T'$

modify the RNN as:

$$p(y_{1:T} \mid x_{1:T}) = \sum_{h_{1:T}} \prod_{t=1}^{T} p(y_t \mid h_t) \mathbb{I}(h_t = f(h_{t-1}, x_t))$$

initial state: $h_1 = f(h_0, x_1) = f_0(x_1)$

dense sequence labeling:
predict one label per location
Recurrent neural networks (RNNs)

- Seq2Seq (sequence translation)
  - aligned: $T = T'$

modify the RNN as:

$p(y_{1:T} \mid x_{1:T}) = \sum_{h_{1:T}}^{T} \prod_{t=1}^{T} p(y_t \mid h_t) \mathbb{I}(h_t = f(h_{t-1}, x_t))$

more depth to be more expressive

$h_t^l = \varphi_l (W_{xh}^l h_{t-1}^l + W_{hh}^l h_{t-1}^l)$

$y_t = W_{hy} h_t^L$
Recurrent neural networks (RNNs)

- Seq2Seq (sequence translation)
  - unaligned: $T \neq T'$

- **encode** the input sequence to get the context vector, the last state of an RNN, $c = f_e(x_{1:T})$

- generate the output sequence using an RNN decoder, $y_{1:T'} = f_d(c)$

\[
f_{\theta} : \mathbb{R}^{TD} \rightarrow \mathbb{R}^{T'C}
\]

Example: translating English to French

See code [here](#)
Training: Backpropagation through time (BPTT)

unroll the computation graph, then apply the backpropagation

Example:

\[
\begin{align*}
  h_t &= W_{hx} x_t + W_{hh} h_{t-1} \\
  \hat{y}_t &= W_{hy} h_t \\
  L &= \frac{1}{T} \sum_{t=1}^{T} \ell(y_t, \hat{y}_t)
\end{align*}
\]
Training: Backpropagation through time (BPTT)

unroll the computation graph, then apply the backpropagation

Example:

\[ h_t = W_{hx} x_t + W_{hh} h_{t-1} = f(x_t, h_{t-1}, w_h) \]
\[ \hat{y}_t = W_{hy} h_t = g(h_t, w_y) \]
\[ L = \frac{1}{T} \sum_{t=1}^{T} \ell(y_t, \hat{y}_t) \]

\[ \frac{\partial L}{\partial W_{hx}} \]
\[ \frac{\partial L}{\partial W_{hh}} \]
\[ \frac{\partial L}{\partial W_{hy}} \]

\[ \frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1} \left( \prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}} \right) \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} \]

see code here
Gating and long term memory

Vanishing and exploding gradients
activations can decay or explode as we go forwards and backwards in time

RNN variations that circumvent this:

- Gated recurrent units (GRU)
  - learns when to update the hidden state, by using a gating unit
- Long short term memory (LSTM)
  - augments the hidden state with a memory cell
The Gated Recurrent Unit (GRU):

\[ z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \]
\[ r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \]
\[ \hat{h}_t = \phi(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h) \]
\[ h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t \]
Attention

\[ z = g(Wx) \]

Instead of linear combination of the input activations, the model dynamically decides (in an input dependent way) which one to use based on how similar the input **query** vector \( q \in \mathbb{R}^q \) is to a set of m **keys** \( K \in \mathbb{R}^{m \times k} \). If \( q \) is most similar to key \( i \), then we use value \( v_i \).

\[
\text{Attn} \ (q, (k_1, v_1), \ldots, (k_m, v_m)) = \text{Attn} \ (q, (k_{1:m}, v_{1:m})) = \sum_{i=1}^{m} \alpha_i (q, k_{1:m}) v_i \in \mathbb{R}^v
\]

\[
\alpha_i (q, k_{1:m}) = \text{softmax}_i ([a(q, k_1), \ldots, a(q, k_m)]) = \frac{\exp(a(q,k_i))}{\sum_{j=1}^{m} \exp(a(q,k_j))}
\]

The attention weights are computed from an attention score function \( a(q, k_i) \in \mathbb{R} \), which gives the similarity of query \( q \) to key \( k_i \).
Parametric Attention

The attention weights are computed from an attention score function $a(q, k_i) \in \mathbb{R}$, which gives the similarity of query $q \in \mathbb{R}^q$ to key $k_i \in \mathbb{R}^k$

- queries and keys both have different sizes
  - map them to a common embedding space of size $h$, then pass these into an MLP
    \[
a(q, k) = w_v^T \tanh (W_q q + W_k k) \in \mathbb{R}
    \]
    \[
    \in \mathbb{R}^{h \times q} \in \mathbb{R}^{h \times k}
    \]

- queries and keys both have length $d = q = k$
  - so we can compute $q^T k$ directly:
    \[
a(q, k) = q^T k / \sqrt{d} \in \mathbb{R}
    \]
  - for minibatches of $n$ vectors this gives:
    \[
    \text{Attn}(Q, K, V) = \text{softmax} \left( \frac{Q K^T}{\sqrt{d}} \right) V \in \mathbb{R}^{n \times v}
    \]
Seq2Seq with attention

use attention to the input sequence in order to capture contextual embeddings of each input
- query is the hidden state of the decoder at the previous step
- keys and values are the hidden states from the encoder

Gives better results for machine translations

**self attention:**
we can also modify the model so the encoder attends to itself

Example: translating English to French
Transformers

a seq2seq model which uses attention in the encoder as well as the decoder, thus eliminating the need for RNNs

- Self-attention
- Multi-headed attention
- Positional encoding
Transformers: **self-attention**

given a sequence of input tokens $x_1, \ldots, x_n$, generate a sequence of outputs of the same size with:

$$y_i = \text{Attn}(x_i, (x_1, x_1), \ldots, (x_n, x_n))$$

for decoder we set $x_i = y_{i-1}$ and $n = i - 1$

this gives improved representations of context

**Example:**

**coreference resolution:**
encoder self-attention for the word “it” differs depending on the input context which is important in translation, e.g. what pronoun to use in French
Transformers: multi-headed attention

use multiple attention matrices, to capture different notions of similarity with projection matrices: $W_i^{(q)} \in \mathbb{R}^{p_q \times d_q}$, $W_i^{(k)} \in \mathbb{R}^{p_k \times d_k}$, and $W_i^{(v)} \in \mathbb{R}^{p_v \times d_v}$

$$h_i = \text{Attn} \left( W_i^{(q)} q, \{ W_i^{(k)} k_j, W_i^{(v)} v_j \} \right) \in \mathbb{R}^{p_v}$$

We then stack the h heads together, and project with $W_o \in \mathbb{R}^{p_o \times hp_v}$:

$$h = \text{MHA} \left( q, \{ k_j, v_j \} \right) = W_o \begin{pmatrix} h_1 \\ \vdots \\ h_h \end{pmatrix} \in \mathbb{R}^{p_o}$$
Transformers: positional encoding

Attention is permutation invariant, and hence ignores the input word ordering. To overcome this, we can concatenate the word embeddings with a positional embedding so that the model knows what order the words occur in.

\[
p_{i,2j} = \sin \left( \frac{i}{10000^{2j/d}} \right)
\]

\[
p_{i,2j+1} = \cos \left( \frac{i}{10000^{2j/d}} \right)
\]

\[
\text{POS(Embed}(X)) = X + P \in \mathbb{R}^{n \times d}
\]

Example:

- Lower columns have higher frequencies.
- Read more [here](#).

Example:

- \( n = 60, d = 32 \)
Transformers: putting it all together

A transformer is a seq2seq model that uses self-attention for the encoder and decoder rather than an RNN. The encoder uses a series of encoder blocks, each of which uses multi-headed attention, residual connections, and layer normalization.
Language models

- **ELMo** (Embeddings from Language Model)
  - RNN based, trained unsupervised to minimize the negative log likelihood of the input sentence, i.e. $y_t = x_{t-1}$

- **BERT** (Bidirectional Encoder Representations from Transformers)
  - Transformer-based: map a modified version of a sequence back to the unmodified form and compute the loss at the masked locations: fill-in-the-blank:

  Let’s make [MASK] chicken! [SEP] It [MASK] great with orange sauce

- **GPT** (Generative Pre-training Transformer)
  - uses a masked transformer as the decoder, see an open-source model [here](#) (20 billion parameters)
Summary

- Recurrent neural networks (RNNs)
  - Vec2Seq (sequence generation)
  - Seq2Vec (sequence classification)
  - Seq2Seq (sequence translation)
  - training with back propagation through time
- attention mechanisms, self-attention and multi-headed attention
- The architecture of transformer
- language models with transformer