# 9- Dynamics Modeling with Neural Nets

Melih Kandemir

University of Southern Denmark
Department of Mathematics and Computer Science (IMADA)
kandemir@imada.sdu.dk

Fall 2022

# Recurrent Neural Nets (RNNs)

The simplest choice is a **recurrent neural net**

$$x_{t+1} := f_\theta(x_t, t).$$

Common choice is $x_{t+1} := W \tanh(x_t)$. Note how dependencies grow

$$\begin{aligned}
x_1 &:= W \tanh(x_0, 0) \\
x_2 &:= W \tanh(W \tanh(x_0, 0)) \\
x_3 &:= W \tanh(W \tanh(W \tanh(x_0))) \\
&\vdots \\
x_N &:= W \tanh(W \tanh(\cdots W \tanh(x_0), \cdots)
\end{aligned}$$

Possible to enhance the nonlinearity of per-time-step update

$$x_{t+1} := W_1 \tanh(W_2 \tanh(x_t)).$$

# Latent state spaces and Seq2Seq design

Also possible to model dynamics in a **latent state space**:

$$x_{t+1} := W_s \tanh(x_t) + W_i \tanh(u_t),$$
$$y_{t+1} := W_o \tanh(x_{t+1}).$$

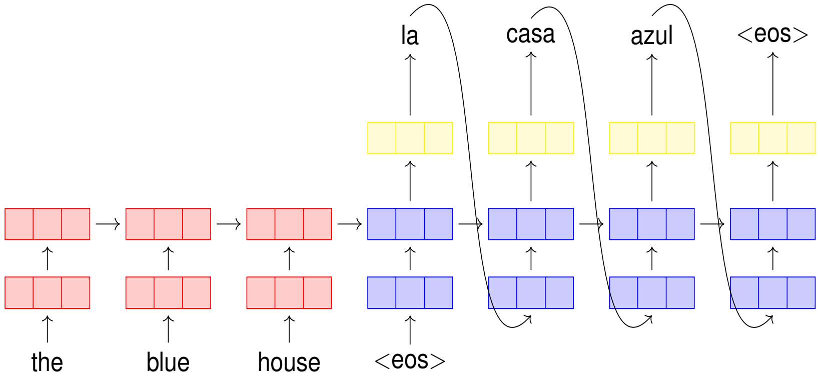
It is even possible to input a sequence $x_A, \ldots, x_B$ and output another not necessarily time-aligned sequence $y_C, \ldots, y_D$:

$$x_{t+1} := W_s \tanh(x_t) + W_i \tanh(u_t), \qquad t = A, \ldots, B$$
$$y_{t+1} := W_o \tanh(x_{t+1}), \qquad t = C, \ldots, D.$$

This is called the **seq2seq** design. A common choice in machine translation applications is $C = B + 1$, i.e. read the whole sentence first and translate afterwards.

# The bottleneck problem

In a seq2seq model, the only information the decoder has about the encoder is the last hidden state $x_B$ of the encoded sequence. This causes big information loss especially for long sequences.



# Chain rule for feedback loop systems

Given input $x_0$ and its intermediate representations $x_l = f_{\theta_l}(x_{l-1})$ For feedforward for $l = 1, 2, \ldots, L$ we have

$$\frac{d}{d\theta_l} f_{\theta_L}(f_{\theta_{L-1}}(\cdots f_{\theta_l}(x_{l-1})\cdots)) = \frac{dx_L}{dx_{L-1}} \frac{dx_{L-1}}{dx_{L-2}} \cdots \frac{dx_l}{dx_{l-1}} \frac{dx_l}{d\theta_l}$$

Things are tricker when the same function is repeated $x_l = f_\theta(x_{l-1})$ sharing the same parameter across nested operations. Above the chain rule applies backwards $L \to l$. Now it needs to be forward in **time** and $x_l$ depends both directly on $\theta$ and via $x_{l-1}$:

$$\frac{dx_2}{d\theta} = \frac{\partial x_2}{\partial \theta} + \frac{\partial x_2}{\partial x_1} \frac{dx_1}{d\theta}$$

Go one step further

$$\frac{dx_3}{d\theta} = \frac{\partial x_3}{\partial \theta} + \frac{\partial x_3}{\partial x_2} \frac{dx_2}{d\theta} = \frac{\partial x_3}{\partial \theta} + \frac{\partial x_3}{\partial x_2} \left( \frac{\partial x_2}{\partial \theta} + \frac{\partial x_2}{\partial x_1} \frac{dx_1}{d\theta} \right).$$

# Chain rule for feedback loop systems

Another step

$$\begin{aligned}\frac{dx_4}{d\theta} &= \frac{\partial x_4}{\partial \theta} + \frac{\partial x_4}{\partial x_3}\frac{\partial x_3}{\partial \theta} + \frac{\partial x_4}{\partial x_3}\frac{\partial x_3}{\partial x_2}\frac{dx_2}{d\theta} \\ &= \frac{\partial x_4}{\partial \theta} + \frac{\partial x_4}{\partial x_3}\frac{\partial x_3}{\partial \theta} + \frac{\partial x_4}{\partial x_3}\frac{\partial x_3}{\partial x_2}\left(\frac{\partial x_2}{\partial \theta} + \frac{\partial x_2}{\partial x_1}\frac{dx_1}{d\theta}\right) \\ &= \frac{\partial x_4}{\partial \theta} + \frac{\partial x_4}{\partial x_3}\frac{\partial x_3}{\partial \theta} + \frac{\partial x_4}{\partial x_3}\frac{\partial x_3}{\partial x_2}\frac{\partial x_2}{\partial \theta} + \frac{\partial x_4}{\partial x_3}\frac{\partial x_3}{\partial x_2}\frac{\partial x_2}{\partial x_1}\frac{dx_1}{d\theta}.\end{aligned}$$

Generalize to an arbitrary $t$th step

$$\frac{dx_t}{d\theta} = \sum_{j=1}^{t}\left(\prod_{k=j+1}^{t}\frac{\partial x_k}{\partial x_{k-1}}\right)\frac{\partial x_j}{\partial \theta}$$

# Back-Propagation Through Time (BPTT)

Denote ground truth as $\hat{x}_t$, model as $x_t = f_\theta(x_{t-1})$ and solve

$$\arg\min_\theta \sum_{t=1}^{N} \frac{1}{2}(\hat{x}_t - x_t)^2$$

by gradient-descent, hence

$$\begin{aligned}\theta :=& \theta - \gamma \sum_{t=1}^{N} (\hat{x}_t - x_t) \frac{dx_t}{d\theta} \\ =& \theta - \gamma \sum_{t=1}^{N} (\hat{x}_t - x_t) \sum_{j=1}^{t} \left( \prod_{k=j+1}^{t} \frac{\partial x_k}{\partial x_{k-1}} \right) \frac{\partial x_j}{\partial \theta}\end{aligned}$$

Unrolling the RNN to collect gradient signal from consecutive time steps to update a single parameter set is called **Back-Propagation Through Time (BPTT)**.

For details see `https://arxiv.org/pdf/1211.5063.pdf`

# The norm (magnitude) of the gradient

$$
\begin{aligned}
&\left\| \sum_{t=1}^{N} (\hat{x}_t - x_t) \sum_{j=1}^{t} \left( \prod_{k=j+1}^{t} \frac{\partial x_k}{\partial x_{k-1}} \right) \frac{\partial x_j}{\partial \theta} \right\| \\
&\qquad = \sum_{t=1}^{N} \left\| (\hat{x}_t - x_t) \sum_{j=1}^{t} \left( \prod_{k=j+1}^{t} \frac{\partial x_k}{\partial x_{k-1}} \right) \frac{\partial x_j}{\partial \theta} \right\| \\
&\qquad \leq \sum_{t=1}^{N} \left\| (\hat{x}_t - x_t) \right\| \cdot \left\| \sum_{j=1}^{t} \left( \prod_{k=j+1}^{t} \frac{\partial x_k}{\partial x_{k-1}} \right) \frac{\partial x_j}{\partial \theta} \right\| \\
&\qquad = \sum_{t=1}^{N} \left\| (\hat{x}_t - x_t) \right\| \cdot \sum_{j=1}^{t} \left\| \left( \prod_{k=j+1}^{t} \frac{\partial x_k}{\partial x_{k-1}} \right) \frac{\partial x_j}{\partial \theta} \right\| \\
&\qquad \leq \sum_{t=1}^{N} \left\| (\hat{x}_t - x_t) \right\| \cdot \sum_{j=1}^{t} \prod_{k=j+1}^{t} \left\| \frac{\partial x_k}{\partial x_{k-1}} \right\| \cdot \left\| \frac{\partial x_j}{\partial \theta} \right\|
\end{aligned}
$$

# The vanishing gradients problem

For the classical RNN design $x_k = W\sigma(x_{k-1})$ with some activation function $\sigma(\cdot)$ we have

$$\left|\left|\frac{\partial x_k}{\partial x_{k-1}}\right|\right| = \left|\left|W\frac{\partial \sigma(x_{k-1})}{\partial x_{k-1}}\right|\right| \leq \left|\left|W\right|\right| \cdot \left|\left|\frac{\partial \sigma(x_{k-1})}{\partial x_{k-1}}\right|\right|.$$

It is likely $||W|| < 1/||\partial\sigma(x_{k-1})/\partial x_{k-1}||$ to happen, which would cause

$$\left|\left|\frac{\partial x_k}{\partial x_{k-1}}\right|\right| < 1 \Rightarrow \prod_{j=t}^{k} \underbrace{\left|\left|\frac{\partial x_k}{\partial x_{k-1}}\right|\right|}_{\eta_j \in [0,1)} \Rightarrow \lim_{t\to\infty} \eta_*^{t-k} = 0$$

where $\eta_* = \max_j \eta_j$. The contribution of the terms dependent on past experience to the gradient **vanishes** exponentially with the time difference $t - k$.

# Old-school solution strategy

We can mitigate this problem if state information is encapsulated within a variable $c_t$ called a **cell**

$$x_t := a_{\theta'}(x_{t-1}) f_{\theta'}(c_t)$$

that has its own parameters $\theta$ that affect the next cell state **additively**:

$$c_t := g_{\theta'}(x_{t-1}) c_{t-1} + h_\theta(x_{t-1}).$$

Then the gradient wrt the cell parameter develops through time as

$$\begin{aligned}
\frac{dc_1}{d\theta} &= \frac{dh_\theta(x_0)}{d\theta} \\
\frac{dc_2}{d\theta} &= g_{\theta'}(x_1)\frac{dc_1}{d\theta} + \frac{dh_\theta(x_1)}{d\theta} = g_{\theta'}(x_1)\frac{dh_\theta(x_0)}{d\theta} + \frac{dh_\theta(x_1)}{dt} \\
\frac{dc_3}{d\theta} &= g_{\theta'}(x_2)\frac{dc_2}{d\theta} + \frac{dh_\theta(x_2)}{d\theta} \\
&= g_{\theta'}(x_2)\left( g_{\theta'}(x_1)\frac{dh_\theta(x_0)}{d\theta} + \frac{dh_\theta(x_1)}{d\theta} \right) + \frac{dh_\theta(x_2)}{d\theta}
\end{aligned}$$

# Old-school solution strategy

$$
\begin{aligned}
\frac{dc_3}{d\theta} &= g_{\theta'}(x_2)g_{\theta'}(x_1)\frac{dh_\theta(x_0)}{d\theta} + g_{\theta'}(x_2)\frac{dh_\theta(x_1)}{d\theta} + \frac{dh_\theta(x_2)}{d\theta} \\
\frac{dc_4}{d\theta} &= g_{\theta'}(x_3)\frac{dc_3}{d\theta} + \frac{dh_\theta(x_3)}{d\theta} \\
\frac{dc_4}{d\theta} &= g_{\theta'}(x_3)\left(g_{\theta'}(x_2)g_{\theta'}(x_1)\frac{dh_\theta(x_0)}{d\theta} + g_{\theta'}(x_2)\frac{dh_\theta(x_1)}{d\theta} + \frac{dh_\theta(x_2)}{d\theta}\right) \\
&\quad + \frac{dh_\theta(x_3)}{d\theta} \\
&= g_{\theta'}(x_3)g_{\theta'}(x_2)g_{\theta'}(x_1)\frac{dh_\theta(x_0)}{d\theta} + g_{\theta'}(x_3)g_{\theta'}(x_2)\frac{dh_\theta(x_1)}{dt} \\
&\quad + g_{\theta'}(x_3)\frac{dh_\theta(x_2)}{d\theta} + \frac{dh_\theta(x_3)}{d\theta}
\end{aligned}
$$

# The gradient highway

Generalized to $t$ steps

$$\frac{dc_t}{d\theta} = \sum_{k=0}^{t-1} \Big( \prod_{j=k+1}^{t-1} g_{\theta'}(x_j) \Big) \frac{dh_\theta(x_k)}{d\theta}$$

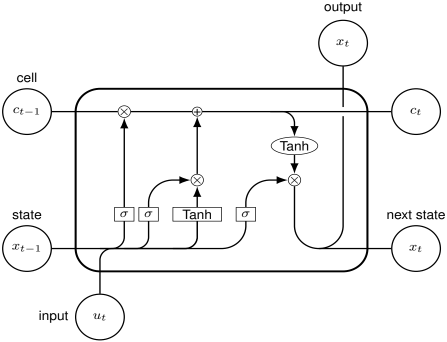
Hence the full gradient is

$$\frac{dx_t}{d\theta} = h_{\theta'}(x_{t-1}) \frac{df_{\theta'}(c_t)}{dc_t} \Big( \prod_{j=k+1}^{t-1} g_{\theta'}(x_j) \Big) \frac{dh_\theta(x_k)}{d\theta}$$

The gradient no longer has a product of terms that depend on the updated parameter $\theta$. The gradient may still vanish if $||g_{\theta'}(x_j)||$ is small for multiple time steps, however

- The risk is significantly lower as it is now a joint event, while in RNNs one occurrence of $||W||$ destroys the whole gradient signal.
- The gradient can no longer vanish due to $\frac{dh_\theta(x_k)}{d\theta}$.

# Long Short-Term Memory (LSTM)




$$c_t := f_t c_{t-1} + i_t \tanh(W_c[x_{t-1}, u_t])$$ cell update

$$x_t := o_t \tanh(c_t)$$ state update

$$f_t = \sigma(W_f[x_{t-1}, u_t]),$$ forget gate

$$i_t = \sigma(W_i[x_{t-1}, u_t]),$$ input gate

$$o_t = \sigma(W_o[x_{t-1}, u_t]),$$ output gate

for $\sigma(z) = 1/(1 + \exp(-z))$.

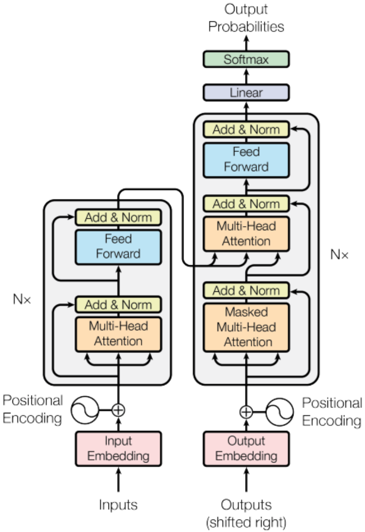
# Transformers[1]

## Dynamics modeling without feedback loops

**Problem:** RNNs (also LSTMs) compute hidden states sequentially:

- unparallelizable
- causes the bottleneck problem
- suffers from vanishing gradients

**Solution:** Use **self-attention** and **point-wise nonlinearity** successively to **transform** a sequence into another sequence.




[1] Figure: `https://arxiv.org/pdf/1706.03762.pdf`

# Elements of the Transformer architecture

The goal is to map $\{x_1, \dots, x_N\}$ to $\{y_1, \dots, y_{N'}\}$ for $x_t \in \mathbb{R}^{d_{in}}$

- **Encoder:** $h_t = f_\theta(x_t)$ for neural net $f_\theta$ and $h_t \in \mathbb{R}^{d_{emb}}$
- **Scaled Dot-Product Attention:** For key dimensionality $d_k$ and value dimensionality $d_v$ we have

$$\texttt{attend}(Q, K, V) = \mathrm{softmax}\Big(\frac{QK^T}{\sqrt{d_k}}\Big)V \in \mathbb{R}^{N' \times d_v}$$

  where $Q \in \mathbb{R}^{N' \times d_k}$, $K \in \mathbb{R}^{N \times d_k}$ and $V \in \mathbb{R}^{N \times d_v}$ with entries

$$v_t = g_\psi(h_t), \qquad k_t = u_\eta(h_t), \qquad q_t = u_\eta(h_t).$$

- **Self-Attention:** Use the same sequence for $K$ and $Q$, i.e.

$$\texttt{self-attend}(Q, K, V) = \mathrm{softmax}\Big(\frac{KK^T}{\sqrt{d_k}}\Big)V \in \mathbb{R}^{N \times d_v}$$

  which outputs a sequence of equal length to the input. When $K \neq Q$ then it is called **cross-attention**.

# Road blocker 1

## Lack of sequence information

**Do positional encoding.** The encoder $f_\theta$ processes each element $x_t$ of the sequence independently, which causes the position information $t$ get lost. Do

$$PE(t, 2i) = \sin(t/10000^{2i/d_{emb}})$$
$$PE(t, 2i+1) = \cos(t/10000^{2i/d_{emb}})$$

where $i \in \{1, \ldots, d_{emb}\}$ and use

$$h_t := f_\theta(x_t) + [PE(1), \ldots, PE(d_{emb})]$$

as the embedding function.

# Road blocker 2

## Softmax attends to a single element

### Do multi-head attention.

$$\texttt{multi-head}(Q, K, V) = \texttt{concat}(A_1, \ldots, A_R)W^O \in \mathbb{R}^{N' \times d_v}$$

where $A_i = \texttt{attend}(QW_i^Q, KW_i^K, VW_i^V)$ is called an **attention head** which has projection matrices

$$W_i^Q \in \mathbb{R}^{d_{emb} \times d_k}, \qquad W_i^K \in \mathbb{R}^{d_{emb} \times d_k}, \qquad W^O \in \mathbb{R}^{hd_v \times d_{emb}}$$

- Akin to using multiple filters in convolutional neural nets.
- Allows querying compound information (e.g. both subject and verb of a sentence to decide the modal verb in correct tense.)
- The classic transformer uses $R = 8$ and $d_v = 64$.

# Road blocker 3

**Transformed sequence is linear in the input sequence**

**Add position-wise nonlinearity.**

$$FFN(h_t') = \texttt{layernorm}\Big( \max(0, W_1^T h_t') W_2 + h_t' \Big)$$

where

$$h_t' = \texttt{layernorm}\Big( \texttt{multi-head}(q_t, K, v_t) + h_t \Big).$$

Attention is a memory fetch and this nonlinearity is an information processing step identical for each piece of information in the memory.

For `layernorm`, see `https://arxiv.org/abs/1607.06450`.

# Road blocker 4

## Cross-attention while decoding

**Use masked decoding.** Prevent attention lookups into the future (causal), because otherwise a circular dependency is introduced:

$$e_{l,t} = \begin{cases} q_l \cdot k_t, & \text{if } t \leq l, \\ -\infty, & \text{otherwise} \end{cases}$$

# Pros and cons of transformers

- (+) Memorizes longer-range distances than RNNs thanks to the one-step computation distance between any pair of variables in the sequence.
- (+) Parallelizable, i.e. allow building much deeper cascades than RNNs without having numerical issues.
- (-) Transformers have complexity $O(N^2)$ but in practice it is more like $O(N)$. They are more difficult to implement than RNNs for most programmers.

Useful to see code here:
`https://pytorch.org/tutorials/beginner/transformer_tutorial.html`.