Recap: Plain Vanilla RNNs
Recap: BPTT

\[
\begin{align*}
\hat{y}_1 & = h_1(U^T x_1) \\
\hat{y}_2 & = h_2(U^T x_2, h_1) \\
\hat{y}_3 & = h_3(U^T x_3, h_2) \\
\hat{y}_4 & = h_4(U^T x_4, h_3) \\
\hat{y}_5 & = h_5(U^T x_5, h_4)
\end{align*}
\]
Challenge of Long Term Dependencies
Challenge of Long-Term Dependencies

- **Basic problem**: Gradients propagated over many stages tend to vanish (most of the time) or explode (relatively rarely)

Challenge of Long-Term Dependencies

- Basic problem: Gradients propagated over many stages tend to vanish (most of the time) or explode (relatively rarely)
  - Blow up \(\rightarrow\) network parameters oscillate
Challenge of Long-Term Dependencies

- **Basic problem:** Gradients propagated over many stages tend to vanish (most of the time) or explode (relatively rarely)
  - Blow up $\Rightarrow$ network parameters oscillate
  - Vanishing $\Rightarrow$ no learning
Basic problem: Gradients propagated over many stages tend to vanish (most of the time) or explode (relatively rarely)
- Blow up $\rightarrow$ network parameters oscillate
- Vanishing $\rightarrow$ no learning

Problem first analyzed by Hochreiter and Schmidhuber, 1991 and Bengio et al., 1993
Challenge of Long-Term Dependencies

- **Basic problem**: Gradients propagated over many stages tend to vanish (most of the time) or explode (relatively rarely)
  - Blow up $\implies$ network parameters oscillate
  - Vanishing $\implies$ no learning
- Problem first analyzed by Hochreiter and Schmidhuber, 1991 and Bengio *et al.*, 1993
Why do gradients explode or vanish?

- Recall the expression for $h_t$ in RNNs:

$$h_t = \tanh(Wh_{t-1} + Vx_t)$$
Why do gradients explode or vanish?

- Recall the expression for $h_t$ in RNNs:

$$h_t = \tanh(W h_{t-1} + V x_t)$$

- $L$ was our loss, so we have by the chain rule:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T} \frac{\partial h_T}{\partial h_t}$$

$$= \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} \frac{\partial h_{k+1}}{\partial h_k}$$

$$= \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} D_{k+1} W_k^T$$
Why do gradients explode or vanish?

\[
\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} D_{k+1} W_k^T
\]

Reminder:

\[D_{k+1} = \text{diag}(1 - \tanh^2 (W h_t - 1 + V x_t))\]

The quantity of interest is the norm of the gradient:

\[\| \frac{\partial L}{\partial h_t} \| \]

Note: \(\| \cdot \|\) represents the L2 norm for a vector and the spectral norm for a matrix.
Why do gradients explode or vanish?

\[ \frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} D_{k+1} W_k^T \]

Reminder: \( D_{k+1} = \text{diag}(1 - \tanh^2(W h_{t-1} + V x_t)) \) is the Jacobian matrix of the pointwise nonlinearity.
Why do gradients explode or vanish?

\[
\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} D_{k+1} W_k^T
\]

- Reminder: \( D_{k+1} = \text{diag}(1 - \tanh^2(W h_{t-1} + V x_t)) \) is the Jacobian matrix of the pointwise nonlinearity.

- The quantity of interest is the norm of the gradient \( \left\| \frac{\partial L}{\partial h_t} \right\| : \)
Why do gradients explode or vanish?

\[
\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} D_{k+1} W_k^T
\]

- Reminder: \( D_{k+1} = \text{diag}(1 - \tanh^2(Wh_{t-1} + Vx_t)) \) is the Jacobian matrix of the pointwise nonlinearity
- The quantity of interest is the norm of the gradient \( \left\| \frac{\partial L}{\partial h_t} \right\| : \)
- Which is simply:

\[
\left\| \frac{\partial L}{\partial h_t} \right\| = \left\| \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} D_{k+1} W_k^T \right\|
\]

- Note: \( \left\| . \right\| \) represents the L2 norm for a vector and the spectral norm for a matrix
Why do gradients explode or vanish?

Given that for any matrices $A, B$ and vector $v$:
\[ \|Av\| \leq \|A\|\|v\| \text{ and } \|AB\| \leq \|A\|\|B\|, \]
we have the trivial bound:
\[
\left\| \frac{\partial L}{\partial h_t} \right\| = \left\| \frac{\partial L}{\partial h_T} T^{-1} \prod_{k=t}^{T-1} D_{k+1} W_k^T \right\| \leq \left\| \frac{\partial L}{\partial h_T} \right\| T^{-1} \prod_{k=t}^{T-1} \| D_{k+1} W_k^T \| \]

Given that $\|A\|$ is the spectral norm (largest singular value):
Why do gradients explode or vanish?

- Given that for any matrices $A, B$ and vector $v$:
  \[ \|Av\| \leq \|A\|\|v\| \text{ and } \|AB\| \leq \|A\|\|B\|, \]
  we have the trivial bound:

  \[
  \|\frac{\partial L}{\partial h_t}\| = \left\| \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} D_{k+1} W_k^T \right\| \leq \left\| \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} \|D_{k+1} W_k^T\| \right\|
  \]

- Given that $\|A\|$ is the spectral norm (largest singular value $\sigma_A$):

  \[
  \left\| \frac{\partial L}{\partial h_t} \right\| \leq \left\| \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} \|D_{k+1} W_k^T\| \right\| = \left\| \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} \sigma_{D_k} \sigma_{W_k} \right\|
  \]
Why do gradients explode or vanish?

- Given that for any matrices $A, B$ and vector $v$:
  
  \[ \| A v \| \leq \| A \| \| v \| \text{ and } \| A B \| \leq \| A \| \| B \| , \]

  we have the trivial bound:

  \[
  \left\| \frac{\partial L}{\partial h_t} \right\| = \left\| \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} D_{k+1} W_k^T \right\| \leq \left\| \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} \| D_{k+1} W_k^T \| \right\|
  \]

- Given that $\| A \|$ is the spectral norm (largest singular value $\sigma_A$):

  \[
  \left\| \frac{\partial L}{\partial h_t} \right\| \leq \left\| \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} \| D_{k+1} W_k^T \| = \left\| \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} \sigma D_k \sigma W_k \right\|
  \]

- The above tells us that the gradient norm can shrink to zero or blow up exponentially fast depending on the gain $\sigma$.
Simplified Model

Consider the recurrence relationship:

\[ h(t) = W^T h(t-1) \]
Simplified Model

- Consider the recurrence relationship:

\[ h(t) = W^T h(t-1) \]

- This could be thought of as a very simple recurrent neural network without a nonlinear activation and lacking \( x \)
Simplified Model

- Consider the recurrence relationship:
  \[ h(t) = W^T h(t-1) \]

- This could be thought of as a very simple recurrent neural network without a nonlinear activation and lacking \( x \).

- Essentially describes the power method:
  \[ h(t) = (W^t)^T h(0) \]

- If \( W \) admits a decomposition \( W = Q \Lambda Q^T \) with orthogonal \( Q \).
Simplified Model

- Consider the recurrence relationship:
  \[ h(t) = W^T h(t-1) \]

- This could be thought of as a very simple recurrent neural network without a nonlinear activation and lacking \( x \)
- Essentially describes the power method:
  \[ h(t) = (W^t)^T h(0) \]

- If \( W \) admits a decomposition \( W = Q \Lambda Q^T \) with orthogonal \( Q \)
- The recurrence becomes:
  \[ h(t) = (W^t)^T h(0) = Q^T \Lambda^t Q h(0) \]
Simplified Model

\[ h^{(t)} = (W^t)^T h^{(0)} = Q^T \Lambda^t Q h^{(0)} \]
Simplified Model

\[ h^{(t)} = (W^t)^T h^{(0)} = Q^T \Lambda^t Q h^{(0)} \]

- Eigenvalues are raised to \( t \): Quickly decay to zero or explode
Simplified Model

\[ h^{(t)} = (W^t)^T h^{(0)} = Q^T \Lambda^t Q h^{(0)} \]

- Eigenvalues are raised to \( t \): Quickly decay to zero or explode
- Problem particular to RNNs
Simplified Model

\[ h(t) = (W^t)^T h(0) = Q^T \Lambda^t Q h(0) \]

- Eigenvalues are raised to \( t \): Quickly decay to zero or explode
- Problem particular to RNNs
- Can be avoided in feedforward networks (atleast in principle)
Some Solutions
Idea 1: Skip Connections

- Add connections from the distant past to the present
Idea 1: Skip Connections

- Add connections from the distant past to the present
- Plain Vanilla RNNs: Recurrence goes from a unit at time $t$ to a unit at time $t + 1$
Idea 1: Skip Connections

- Add connections from the distant past to the present
- Plain Vanilla RNNs: Recurrence goes from a unit at time $t$ to a unit at time $t + 1$
- Gradients vanish/explode w.r.t number of time steps
Idea 1: Skip Connections

- Add connections from the distant past to the present
- Plain Vanilla RNNs: Recurrence goes from a unit at time \( t \) to a unit at time \( t + 1 \)
- Gradients vanish/explode w.r.t number of time steps
- With recurrent connections with a time-delay of \( d \), gradients explode/vanish exponentially as a function of \( \frac{\tau}{d} \) rather than \( \tau \)
Idea 2: Leaky Units

- Keep a running average for a hidden unit by adding a linear self connection:

\[ h_t \leftarrow \alpha h_{t-1} + (1 - \alpha) h_t \]
Idea 2: Leaky Units

- Keep a running average for a hidden unit by adding a linear self connection:

$$h_t \leftarrow \alpha h_{t-1} + (1 - \alpha) h_t$$

- Such hidden units are called leaky units
Idea 2: Leaky Units

- Keep a running average for a hidden unit by adding a linear self connection:

\[ h_t \leftarrow \alpha h_{t-1} + (1 - \alpha) h_t \]

- Such hidden units are called leaky units

- Ensures hidden units can easily access values from the past
Idea 3: Echo State Networks

- **Idea**: Set the recurrent weights such that they do a *good job* of capturing past history and learn only the output weights.
Idea 3: Echo State Networks

- **Idea:** Set the recurrent weights such that they do a *good job* of capturing past history and learn only the output weights.
- **Methods:** Echo State Machines, Liquid State Machines
Idea 3: Echo State Networks

- **Idea:** Set the recurrent weights such that they do a *good job* of capturing past history and learn only the output weights.
- **Methods:** Echo State Machines, Liquid State Machines.
- The general methodology is called Reservoir Computing.
Idea 3: Echo State Networks

- **Idea:** Set the recurrent weights such that they do a *good job* of capturing past history and learn only the output weights.
- **Methods:** Echo State Machines, Liquid State Machines
- The general methodology is called Reservoir Computing
- How to choose the recurrent weights?
Echo State Networks: Motivation

Choose recurrent weights such that the hidden-to-hidden transition Jacobian has eigenvalues close to 1
Echo State Networks: Motivation

- Choose recurrent weights such that the hidden-to-hidden transition Jacobian has eigenvalues close to 1
- In particular we pay attention to the spectral radius of $J_t$
Echo State Networks: Motivation

- Choose recurrent weights such that the hidden-to-hidden transition Jacobian has eigenvalues close to 1.
- In particular we pay attention to the spectral radius of $J_t$.
- Consider gradient $g$, after one step of backpropagation it would be $Jg$ and after $n$ steps it would be $J^n g$.

Lecture 12 Recurrent Neural Networks II
Echo State Networks: Motivation

- Choose recurrent weights such that the hidden-to-hidden transition Jacobian has eigenvalues close to 1.
- In particular we pay attention to the spectral radius of $J_t$.
- Consider gradient $g$, after one step of backpropagation it would be $Jg$ and after $n$ steps it would be $J^n g$.
- Now consider a perturbed version of $g$ i.e. $g + \delta v$, after $n$ steps we will have $J^n (g + \delta v)$.
Echo State Networks: Motivation

- Choose recurrent weights such that the hidden-to-hidden transition Jacobian has eigenvalues close to 1.
- In particular we pay attention to the spectral radius of $J_t$.
- Consider gradient $\mathbf{g}$, after one step of backpropagation it would be $J\mathbf{g}$ and after $n$ steps it would be $J^n\mathbf{g}$.
- Now consider a perturbed version of $\mathbf{g}$ i.e. $\mathbf{g} + \delta \mathbf{v}$, after $n$ steps we will have $J^n(\mathbf{g} + \delta \mathbf{v})$.
- In fact, the separation is exactly $\delta |\lambda|^n$. 

Lecture 12 Recurrent Neural Networks II
CMSC 35246
Choose recurrent weights such that the hidden-to-hidden transition Jacobian has eigenvalues close to 1.

In particular we pay attention to the spectral radius of $J_t$.

Consider gradient $g$, after one step of backpropagation it would be $Jg$ and after $n$ steps it would be $J^ng$.

Now consider a perturbed version of $g$ i.e. $g + \delta v$, after $n$ steps we will have $J^n(g + \delta v)$.

Infact, the separation is exactly $\delta |\lambda|^n$.

When $|\lambda| > 1$, $\delta |\lambda|^n$ grows exponentially large and vice-versa.
Echo State Networks

- For a vector \( h \), when a linear map \( W \) always shrinks \( h \), the mapping is said to be contractive
Echo State Networks

- For a vector \( h \), when a linear map \( W \) always shrinks \( h \), the mapping is said to be contractive.
- The strategy of echo state networks is to make use of this intuition.
Echo State Networks

- For a vector $h$, when a linear map $W$ always shrinks $h$, the mapping is said to be contractive.
- The strategy of echo state networks is to make use of this intuition.
- The Jacobian is chosen such that the spectral radius corresponds to stable dynamics.
Echo State Networks

- For a vector $h$, when a linear map $W$ always shrinks $h$, the mapping is said to be contractive.
- The strategy of echo state networks is to make use of this intuition.
- The Jacobian is chosen such that the spectral radius corresponds to stable dynamics.
- Then we only learn the output weights!
Echo State Networks

- For a vector $h$, when a linear map $W$ always shrinks $h$, the mapping is said to be contractive.
- The strategy of echo state networks is to make use of this intuition.
- The Jacobian is chosen such that the spectral radius corresponds to stable dynamics.
- Then we only learn the output weights!
- Can be used to initialize a fully trainable RNN.
• Solid arrows represent fixed, random connections. Dashed arrows represent learnable weights.
A Popular Solution: Gated Architectures
Back to Plain Vanilla RNN

Figure: Chris Olah
Long Short Term Memory

Proposed by Hochreiter and Schmidhuber (1997)

Figure: Chris Olah

Now let's try to understand each memory cell!
Long Short Term Memory

- Proposed by Hochreiter and Schmidhuber (1997)
- Now let’s try to understand each memory cell!
Long Short Term Memory

\[ h_t = \tanh(W h_{t-1} + U x_t) \]
Long Short Term Memory

\[ \tilde{c}_t = \tanh(W h_{t-1} + U x_t) \]
\[ c_t = c_{t-1} + \tilde{c}_t \]
Long Short Term Memory

\[ f_t = \sigma(W_f h_{t-1} + U_f x_t) \]

\[ \tilde{c_t} = \tanh(W h_{t-1} + U x_t) \]

\[ c_t = f_t \odot c_{t-1} + \tilde{c_t} \]
Long Short Term Memory

\[ f_t = \sigma(W_f h_{t-1} + U_f x_t) \]
\[ i_t = \sigma(W_i h_{t-1} + U_i x_t) \]
\[ \tilde{c}_t = \tanh(W h_{t-1} + U x_t) \]
\[ c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \]
\[
\begin{align*}
\hat{c}_t &= \tanh(Wh_{t-1} + Ux_t) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \hat{c}_t \\
h_t &= o_t \odot \tanh(c_t)
\end{align*}
\]

Lecture 12 Recurrent Neural Networks II
LSTM: Further Intuition

- The Cell State

\[ c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad \text{with} \quad \tilde{c}_t = \tanh(W_h t - 1 + U x_t) \]
LSTM: Further Intuition

- The Cell State

\[ c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \text{ with } \tilde{c}_t = \tanh(W h_{t-1} + U x_t) \]

- Useful to think of the cell as a *conveyor belt* (Olah), which runs across time; only interrupted with linear interactions
LSTM: Further Intuition

- The Cell State

\[ c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \text{ with } \tilde{c}_t = \tanh(W h_{t-1} + U x_t) \]

- Useful to think of the cell as a *conveyor belt* (Olah), which runs across time; only interrupted with linear interactions
- The memory cell can add or delete information from the cell state by *gates*
LSTM: Further Intuition

- The Cell State

\[ c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \text{ with } \tilde{c}_t = \tanh(Wh_{t-1} + Ux_t) \]

- Useful to think of the cell as a *conveyor belt* (Olah), which runs across time; only interrupted with linear interactions
- The memory cell can add or delete information from the cell state by gates
- Gates are constructed by using a sigmoid and a pointwise multiplication
LSTM: Further Intuition

- The Forget Gate

\[ f_t = \sigma(W_f h_{t-1} + U_f x_t) \]
LSTM: Further Intuition

- The Forget Gate

\[ f_t = \sigma(W_f h_{t-1} + U_f x_t) \]

- Helps to decide what information to throw away from the cell state
LSTM: Further Intuition

- The Forget Gate

\[ f_t = \sigma(W_f h_{t-1} + U_f x_t) \]

- Helps to decide what information to throw away from the cell state
- Once we have thrown away what we want from the cell state, we want to update it
LSTM: Further Intuition

- First we decide how much of the input we want to store in the updated cell state via the Input Gate

\[ i_t = \sigma(W_i h_{t-1} + U_i x_t) \]
LSTM: Further Intuition

- First, we decide how much of the input we want to store in the updated cell state via the Input Gate

\[ i_t = \sigma(W_i h_{t-1} + U_i x_t) \]

- We then update the cell state:

\[ c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \]
LSTM: Further Intuition

- First we decide how much of the input we want to store in the updated cell state via the Input Gate

\[ i_t = \sigma(W_i h_{t-1} + U_i x_t) \]

- We then update the cell state:

\[ c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \]

- We then need to output, and use the output gate

\[ o_t = \sigma(W_o h_{t-1} + U_o x_t) \] to pass on the filtered version

\[ h_t = o_t \odot \tanh(c_t) \]
Gated Recurrent Unit

• Let $\tilde{h}_t = \tanh(W h_{t-1} + U x_t)$ and $h_t = \tilde{h}_t$
Gated Recurrent Unit

- Let $\tilde{h}_t = \tanh(W h_{t-1} + U x_t)$ and $h_t = \tilde{h}_t$
- Reset gate: $r_t = \sigma(W_r h_{t-1} + U_r x_t)$
Gated Recurrent Unit

- Let $\tilde{h}_t = \tanh(W h_{t-1} + U x_t)$ and $h_t = \tilde{h}_t$
- Reset gate: $r_t = \sigma(W_r h_{t-1} + U_r x_t)$
- New $\tilde{h}_t = \tanh(W (r_t \odot h_{t-1}) + U x_t)$

Comes from attempting to factor LSTM and reduce gates

Example: One gate controls forgetting as well as decides if the state needs to be updated
Gated Recurrent Unit

- Let $\tilde{h}_t = \tanh(Wh_{t-1} + Ux_t)$ and $h_t = \tilde{h}_t$
- Reset gate: $r_t = \sigma(W_rh_{t-1} + U_rx_t)$
- New $\tilde{h}_t = \tanh(W(r_t \odot h_{t-1}) + Ux_t)$
- Find: $z_t = \sigma(Wzh_{t-1} + Uzx_t)$
Let $\tilde{h}_t = \tanh(W h_{t-1} + U x_t)$ and $h_t = \tilde{h}_t$

Reset gate: $r_t = \sigma(W_r h_{t-1} + U_r x_t)$

New $\tilde{h}_t = \tanh(W (r_t \odot h_{t-1}) + U x_t)$

Find: $z_t = \sigma(W_z h_{t-1} + U_z x_t)$

Update $h_t = z_t \odot \tilde{h}_t$
Gated Recurrent Unit

- Let $\tilde{h}_t = \tanh(W h_{t-1} + U x_t)$ and $h_t = \tilde{h}_t$
- Reset gate: $r_t = \sigma(W_r h_{t-1} + U_r x_t)$
- New $\tilde{h}_t = \tanh(W(r_t \odot h_{t-1}) + U x_t)$
- Find: $z_t = \sigma(W_z h_{t-1} + U_z x_t)$
- Update $h_t = z_t \odot \tilde{h}_t$
- Finally: $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$
Gated Recurrent Unit

- Let $\tilde{h}_t = \tanh(W_h h_{t-1} + U x_t)$ and $h_t = \tilde{h}_t$
- Reset gate: $r_t = \sigma(W_r h_{t-1} + U_r x_t)$
- New $\tilde{h}_t = \tanh(W (r_t \odot h_{t-1}) + U x_t)$
- Find: $z_t = \sigma(W_z h_{t-1} + U_z x_t)$
- Update $h_t = z_t \odot \tilde{h}_t$
- Finally: $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$
- Comes from attempting to factor LSTM and reduce gates
Gated Recurrent Unit

- Let $\tilde{h}_t = \tanh(W h_{t-1} + U x_t)$ and $h_t = \tilde{h}_t$
- Reset gate: $r_t = \sigma(W_r h_{t-1} + U_r x_t)$
- New $\tilde{h}_t = \tanh(W (r_t \odot h_{t-1}) + U x_t)$
- Find: $z_t = \sigma(W_z h_{t-1} + U_z x_t)$
- Update $h_t = z_t \odot \tilde{h}_t$
- Finally: $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$

- Comes from attempting to factor LSTM and reduce gates
- Example: One gate controls forgetting as well as decides if the state needs to be updated
Gated Recurrent Unit

\[ \tilde{h}_t = \tanh(W h_{t-1} + U x_t) \]

\[ h_t = \tilde{h}_t \]
Gated Recurrent Unit

\[ r_t = \sigma(W_r h_{t-1} + U_r x_t) \]

\[ \tilde{h}_t = \tanh(W (r_t \odot h_{t-1}) + U x_t) \]

\[ h_t = \tilde{h}_t \]
Gated Recurrent Unit

\[ r_t = \sigma(W_r h_{t-1} + U_r x_t) \]
\[ z_t = \sigma(W_z h_{t-1} + U_z x_t) \]

\[ \tilde{h}_t = \tanh(W (r_t \odot h_{t-1}) + U x_t) \]
\[ h_t = z_t \odot \tilde{h}_t \]
Gated Recurrent Unit

\[ r_t = \sigma(W_r h_{t-1} + U_r x_t) \]
\[ z_t = \sigma(W_z h_{t-1} + U_z x_t) \]

\[ h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \]

\[ \tilde{h}_t = \tanh(W (r_t \odot h_{t-1}) + U x_t) \]
Attention Models
To illustrate the fundamental idea of attention, we will look at two classic papers on the topic.
Attention Models

- To illustrate the fundamental idea of attention, we will look at two classic papers on the topic

- **Machine Translation:** *Neural Machine Translation by Jointly Learning to Align and Translate* by Bahdanau *et al.*
Attention Models

- To illustrate the fundamental idea of attention, we will look at two classic papers on the topic
- **Machine Translation:** *Neural Machine Translation by Jointly Learning to Align and Translate* by Bahdanau et al.
- **Image Caption Generation:** *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015* by Xu et al.
To illustrate the fundamental idea of attention, we will look at two classic papers on the topic

**Machine Translation:** *Neural Machine Translation by Jointly Learning to Align and Translate* by Bahdanau et al.

**Image Caption Generation:** *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015* by Xu et al.

Let us consider Machine Translation first
Attention Models: Motivation

- Recall our encoder-decoder model for machine translation

Figure: Goodfellow et al.
Let’s look at the steps for translation again:
Let’s look at the steps for translation again:

- The input sentence $x_1, \ldots, x_n$ via hidden unit activations $h_1, \ldots, h_n$ is encoded into the thought vector $C$.
Attention Models: Motivation

- Let’s look at the steps for translation again:
- The input sentence $x_1, \ldots, x_n$ via hidden unit activations $h_1, \ldots, h_n$ is encoded into the thought vector $C$
- Using $C$, the decoder then generates the output sentence $y_1, \ldots, y_p$
Attention Models: Motivation

Let’s look at the steps for translation again:

- The input sentence $x_1, \ldots, x_n$ via hidden unit activations $h_1, \ldots, h_n$ is encoded into the thought vector $C$
- Using $C$, the decoder then generates the output sentence $y_1, \ldots, y_p$
- We stop when we sample a terminating token i.e. $\langle END \rangle$
Attention Models: Motivation

Let's look at the steps for translation again:

- The input sentence $x_1, \ldots, x_n$ via hidden unit activations $h_1, \ldots, h_n$ is encoded into the *thought vector* $C$
- Using $C$, the decoder then *generates* the output sentence $y_1, \ldots, y_p$
- We stop when we sample a terminating token i.e. $\langle END \rangle$
- A Problem? For long sentences, it might not be useful to only give the decoder access to the vector $C$
Attention Models: Motivation

- When we ourselves are translating a sentence from one language to another, we don’t consider the whole sentence at all times.
Attention Models: Motivation

- When we ourselves are translating a sentence from one language to another, we don’t consider the whole sentence at all times.

- Intuition: Every word in the output only depends on a word or a group of words in the input sentence.
Attention Models: Motivation

- When we ourselves are translating a sentence from one language to another, we don’t consider the whole sentence at all times
- **Intuition:** Every word in the output only depends on a word or a group of words in the input sentence
- We can help the decoding process by allowing the decoder to refer to the input sentence
Attention Models: Motivation

- When we ourselves are translating a sentence from one language to another, we don’t consider the whole sentence at all times.
- **Intuition:** Every word in the output only depends on a word or a group of words in the input sentence.
- We can help the decoding process by allowing the decoder to refer to the input sentence.
- We would like the decoder, while it is about to generate the next word, to *attend* to a group of words in the input sentence most relevant to predicting the right next word.
When we ourselves are translating a sentence from one language to another, we don’t consider the whole sentence at all times.

**Intuition:** Every word in the output only depends on a word or a group of words in the input sentence.

We can help the decoding process by allowing the decoder to refer to the input sentence.

We would like the decoder, while it is about to generate the next word, to *attend* to a group of words in the input sentence most relevant to predicting the right next word.

Maybe it would be more efficient to also be able to *attend* to these words *while decoding*.
First Observation: For each word we had a hidden unit. This encoded a representation for each word.
First Observation: For each word we had a hidden unit. This encoded a representation for each word.

Let use first try to incorporate both forwards and backward context for each word using a bidirectional RNN and concatenate the resulting representations.
Machine Translation Using Attention

- **First Observation:** For each word we had a hidden unit. This encoded a representation for each word.
- Let use first try to incorporate both forwards and backward context for each word using a bidirectional RNN and concatenate the resulting representations.
- We have already seen why using a bidirectional RNN is useful.
First Observation: For each word we had a hidden unit. This encoded a representation for each word.

Let use first try to incorporate both forwards and backward context for each word using a bidirectional RNN and concatenate the resulting representations.

We have already seen why using a bidirectional RNN is useful.

Figure: Roger Grosse
Machine Translation Using Attention

- The decoder generates the sentence one word at a time conditioned on $C$
Machine Translation Using Attention

- The decoder generates the sentence one word at a time conditioned on \( C \)
- We can instead have a context vector for every time step
Machine Translation Using Attention

- The decoder generates the sentence one word at a time conditioned on $C$
- We can instead have a context vector for every time step
- These vectors $C(t)$ learn to attend to specific words of the input sentence
Machine Translation Using Attention

- The decoder generates the sentence one word at a time conditioned on $C$
- We can instead have a context vector for every time step
- These vectors $C^{(t)}$ learn to attend to specific words of the input sentence

Figure: Roger Grosse
How do we learn $C^{(t)}$’s so that they attend to relevant words?
How do we learn $C^{(t)}$’s so that they attend to relevant words?

First: Let the representations of the bidirectional RNN for each word be $h_i$. 

$$C^{(t)} = \sum_i \alpha_t h_i$$
How do we learn $C^{(t)}$’s so that they attend to relevant words?

First: Let the representations of the bidirectional RNN for each word be $h_i$

Define $C^{(t)}$ to be the weighted average of encoder’s representations:

$$C^{(t)} = \sum_i \alpha_{ti} h_i$$
How do we learn $C^{(t)}$’s so that they attend to relevant words?

First: Let the representations of the bidirectional RNN for each word be $h_i$

Define $C^{(t)}$ to be the weighted average of encoder’s representations:

$$C^{(t)} = \sum_i \alpha_{ti} h_i$$

$\alpha_t$ defines a probability distribution over the input words
Machine Translation Using Attention

- $\alpha_{ti}$ is a function of the representations of the words, as well as the previous state of the decoder.

\[ \alpha_{ti} = \exp(e_{ti}) \sum_k \exp(e_{tk}) \]
\[ e_{ti} = a(s(t-1), h(i)) \]

This is a form of content-based addressing.

Example: The language model says the next word should be an adjective, give me an adjective in the input.
Machine Translation Using Attention

- $\alpha_{ti}$ is a function of the representations of the words, as well as the previous state of the decoder.

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_k \exp(e_{tk})}$$

This is a form of content-based addressing.

Example: The language model says the next word should be an adjective, give me an adjective in the input.
Machine Translation Using Attention

- $\alpha_{ti}$ is a function of the representations of the words, as well the previous state of the decoder

\[
\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_k \exp(e_{tk})}
\]

With $e_{ti} = a(s^{(t-1)}, h^{(i)})$
\( \alpha_{ti} \) is a function of the representations of the words, as well the previous state of the decoder

\[
\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_k \exp(e_{tk})}
\]

With \( e_{ti} = a(s^{(t-1)}, h^{(i)}) \)

This is a form of content-based addressing
Machine Translation Using Attention

- $\alpha_{ti}$ is a function of the representations of the words, as well the previous state of the decoder

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_k \exp(e_{tk})}$$

With $e_{ti} = a(s^{(t-1)}, h^{(i)})$

- This is a form of content-based addressing
- Example: The language model says the next word should be an adjective, give me an adjective in the input
Machine Translation Using Attention

- For each word in the translation, the matrix gives the degree of focus on all the input words.
- A linear order is not forced, but it figures out that the translation is approximately linear.
Attention in Computer Vision

We will look at only one illustrative example: *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015*
Attention in Computer Vision

- We will look at only one illustrative example: *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015*
- Attention can also be used to understand images
Attention in Computer Vision

- We will look at only one illustrative example: *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015*
- Attention can also be used to understand images
- Humans don’t process a visual scene all at once. The Fovea gives high resolution vision in only a tiny region of our field of view
Attention in Computer Vision

- We will look at only one illustrative example: *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015*
- Attention can also be used to understand images
- Humans don’t process a visual scene all at once. The Fovea gives high resolution vision in only a tiny region of our field of view
- A series of glimpses are then integrated
Caption Generation using Attention

Here we have an encoder and decoder as well:
Caption Generation using Attention

- Here we have an encoder and decoder as well:
  - **Encoder:** A trained network like ResNet that extracts features for an input image
  - **Decoder:** Attention based RNN, which is like the decoder in the translation model of Bahdanau

While generating the caption, at every time step, the decoder must decide which region of the image to attend to.

The decoder here too receives a context vector, which is the weighted average of the convolutional network features.

The $\alpha$'s here would define a distribution over the pixels indicating what pixels we would like to focus on to predict the next word.
Here we have an encoder and decoder as well:

- **Encoder:** A trained network like ResNet that extracts features for an input image
- **Decoder:** Attention based RNN, which is like the decoder in the translation model of Bahdanau
Caption Generation using Attention

Here we have an encoder and decoder as well:

- **Encoder**: A trained network like ResNet that extracts features for an input image
- **Decoder**: Attention based RNN, which is like the decoder in the translation model of Bahdanau

While generating the caption, at every time step, the decoder must decide which region of the image to attend to
Here we have an encoder and decoder as well:

**Encoder:** A trained network like ResNet that extracts features for an input image

**Decoder:** Attention based RNN, which is like the decoder in the translation model of Bahdanau

While generating the caption, at every time step, the decoder must decide which region of the image to attend to.

The decoder here too receives a context vector, which is the weighted average of the convolutional network features.
Caption Generation using Attention

- Here we have an encoder and decoder as well:
- **Encoder:** A trained network like ResNet that extracts features for an input image
- **Decoder:** Attention based RNN, which is like the decoder in the translation model of Bahdanau
- While generating the caption, at every time step, the decoder must decide which region of the image to attend to
- The decoder here too receives a context vector, which is the weighted average of the convolutional network features
- The $\alpha$’s here would define a distribution over the pixels indicating what pixels we would like to focus on to predict the next word
Caption Generation without Attention

Image: H x W x 3

Features: D

Hidden state: H

First word

Second word

Distribution over vocab

d1

d2

RNN only looks at whole image, once

What if the RNN looks at different parts of the image at each timestep?

Figure: Andrej Karpathy
Caption Generation with Attention


Figure: Andrej Karpathy
Caption Generation using Attention

- Not only generates good captions, but we also get to see where the decoder is looking at in the image.
Caption Generation using Attention

- Can also see the networks mistakes

A large white bird standing in a forest.
A woman holding a clock in her hand.
A man wearing a hat and a hat on a skateboard.

A person is standing on a beach with a surfboard.
A woman is sitting at a table with a large pizza.
A man is talking on his cell phone while another man watches.
Next Time:
Neural Networks with Explicit Memory