# Lecture 12
# Recurrent Neural Networks II

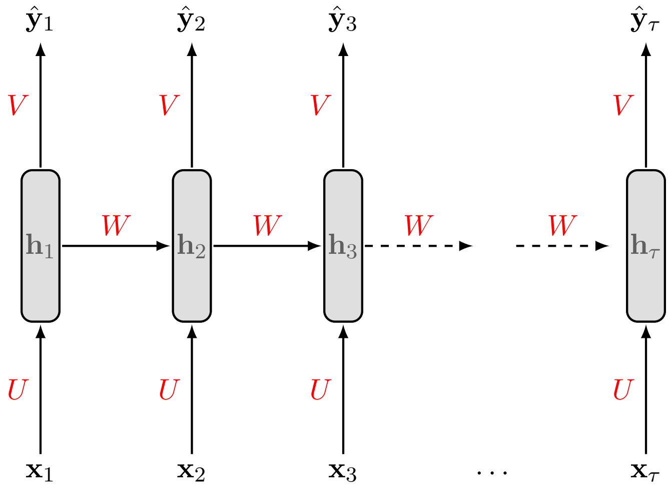
## CMSC 35246: Deep Learning

Shubhendu Trivedi
&
Risi Kondor

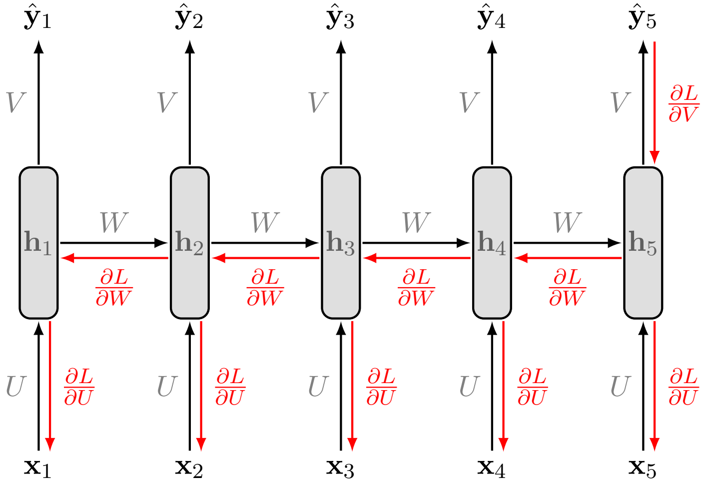
University of Chicago

May 03, 2017

# Recap: Plain Vanilla RNNs



# Recap: BPTT



$\hat{\mathbf{y}}_1$
$\hat{\mathbf{y}}_2$
$\hat{\mathbf{y}}_3$
$\hat{\mathbf{y}}_4$
$\hat{\mathbf{y}}_5$
$V$
$\frac{\partial L}{\partial V}$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_4$
$\mathbf{h}_5$
$W$
$\frac{\partial L}{\partial W}$
$U$
$\frac{\partial L}{\partial U}$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$

# 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 $\rightarrowtail$ 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 $\rightarrowtail$ network parameters oscillate
  - Vanishing $\rightarrowtail$ no learning

# 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 $\rightarrowtail$ network parameters oscillate
  - Vanishing $\rightarrowtail$ 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 $\rightarrowtail$ network parameters oscillate
  - Vanishing $\rightarrowtail$ no learning
- Problem first analyzed by Hochreiter and Schmidhuber, 1991 and Bengio *et al.*, 1993
- Reference: Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, TU Munich, 1991

# Why do gradients explode or vanish?

- Recall the expression for $\mathbf{h}_t$ in RNNs:

$$\mathbf{h}_t = \tanh(W\mathbf{h}_{t-1} + V\mathbf{x}_t)$$

# Why do gradients explode or vanish?

- Recall the expression for $\mathbf{h}_t$ in RNNs:

$$\mathbf{h}_t = \tanh(W\mathbf{h}_{t-1} + V\mathbf{x}_t)$$

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

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{h}_t} &= \frac{\partial L}{\partial \mathbf{h}_T}\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} \\ &= \frac{\partial L}{\partial \mathbf{h}_T}\prod_{k=t}^{T-1}\frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \\ &= \frac{\partial L}{\partial \mathbf{h}_T}\prod_{k=t}^{T-1} D_{k+1} W_k^T\end{aligned}$$

# Why do gradients explode or vanish?

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{h}_T} \prod_{k=t}^{T-1} D_{k+1} W_k^T$$

# Why do gradients explode or vanish?

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{h}_T} \prod_{k=t}^{T-1} D_{k+1} W_k^T$$

- Reminder: $D_{k+1} = diag(1 - tanh^2(W\mathbf{h}_{t-1} + V\mathbf{x}_t))$ is the Jacobian matrix of the pointwise nonlinearity

# Why do gradients explode or vanish?

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{h}_T} \prod_{k=t}^{T-1} D_{k+1} W_k^T$$

- Reminder: $D_{k+1} = diag(1 - tanh^2(W\mathbf{h}_{t-1} + V\mathbf{x}_t))$ is the Jacobian matrix of the pointwise nonlinearity
- The quantity of interest is the norm of the gradient $\left\| \frac{\partial L}{\partial \mathbf{h}_t} \right\|$:

# Why do gradients explode or vanish?

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{h}_T} \prod_{k=t}^{T-1} D_{k+1} W_k^T$$

- Reminder: $D_{k+1} = diag(1 - tanh^2(W\mathbf{h}_{t-1} + V\mathbf{x}_t))$ is the Jacobian matrix of the pointwise nonlinearity
- The quantity of interest is the norm of the gradient $\left\| \frac{\partial L}{\partial \mathbf{h}_t} \right\|$:
- Which is simply:

$$\left\| \frac{\partial L}{\partial \mathbf{h}_t} \right\| = \left\| \frac{\partial L}{\partial \mathbf{h}_T} \prod_{k=t}^{T-1} D_{k+1} W_k^T \right\|$$

- Note: $\| \mathbf{.} \|$ 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 $\mathbf{v}$: $\|A\mathbf{v}\| \leq \|A\|\|\mathbf{v}\|$ and $\|AB\| \leq \|A\|\|B\|$, we have the trivial bound:

$$\left\|\frac{\partial L}{\partial \mathbf{h}_t}\right\| = \left\|\frac{\partial L}{\partial \mathbf{h}_T}\prod_{k=t}^{T-1} D_{k+1}W_k^T\right\| \leq \left\|\frac{\partial L}{\partial \mathbf{h}_T}\right\| \prod_{k=t}^{T-1} \|D_{k+1}W_k^T\|$$

# Why do gradients explode or vanish?

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

$$\left\|\frac{\partial L}{\partial \mathbf{h}_t}\right\| = \left\|\frac{\partial L}{\partial \mathbf{h}_T} \prod_{k=t}^{T-1} D_{k+1} W_k^T\right\| \leq \left\|\frac{\partial L}{\partial \mathbf{h}_T}\right\| \prod_{k=t}^{T-1} \|D_{k+1} W_k^T\|$$

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

$$\left\|\frac{\partial L}{\partial \mathbf{h}_t}\right\| \leq \left\|\frac{\partial L}{\partial \mathbf{h}_T}\right\| \prod_{k=t}^{T-1} \|D_{k+1} W_k^T\| = \left\|\frac{\partial L}{\partial \mathbf{h}_T}\right\| \prod_{k=t}^{T-1} \sigma_{D_k} \sigma_{W_k}$$

# Why do gradients explode or vanish?

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

$$\left\|\frac{\partial L}{\partial \mathbf{h}_t}\right\| = \left\|\frac{\partial L}{\partial \mathbf{h}_T}\prod_{k=t}^{T-1} D_{k+1}W_k^T\right\| \leq \left\|\frac{\partial L}{\partial \mathbf{h}_T}\right\|\prod_{k=t}^{T-1}\|D_{k+1}W_k^T\|$$

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

$$\left\|\frac{\partial L}{\partial \mathbf{h}_t}\right\| \leq \left\|\frac{\partial L}{\partial \mathbf{h}_T}\right\|\prod_{k=t}^{T-1}\|D_{k+1}W_k^T\| = \left\|\frac{\partial L}{\partial \mathbf{h}_T}\right\|\prod_{k=t}^{T-1}\sigma_{D_k}\sigma_{W_k}$$

- 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:

$$\mathbf{h}^{(t)} = W^T \mathbf{h}^{(t-1)}$$

# Simplified Model

- Consider the recurrence relationship:

$$\mathbf{h}^{(t)} = W^T \mathbf{h}^{(t-1)}$$

- This could be thought of as a very simple recurrent neural network without a nonlinear activation and lacking $\mathbf{x}$

# Simplified Model

- Consider the recurrence relationship:

$$\mathbf{h}^{(t)} = W^T \mathbf{h}^{(t-1)}$$

- This could be thought of as a very simple recurrent neural network without a nonlinear activation and lacking $\mathbf{x}$
- Essentially describes the power method:

$$\mathbf{h}^{(t)} = (W^t)^T \mathbf{h}^{(0)}$$

- If $W$ admits a decomposition $W = Q \Lambda Q^T$ with orthogonal $Q$

# Simplified Model

- Consider the recurrence relationship:

$$\mathbf{h}^{(t)} = W^T \mathbf{h}^{(t-1)}$$

- This could be thought of as a very simple recurrent neural network without a nonlinear activation and lacking $\mathbf{x}$
- Essentially describes the power method:

$$\mathbf{h}^{(t)} = (W^t)^T \mathbf{h}^{(0)}$$

- 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:

$$\mathbf{h}_t \leftarrow \alpha \mathbf{h}_{t-1} + (1 - \alpha)\mathbf{h}_t$$

# Idea 2: Leaky Units

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

$$\mathbf{h}_t \leftarrow \alpha \mathbf{h}_{t-1} + (1-\alpha)\mathbf{h}_t$$

- 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:

$$\mathbf{h}_t \leftarrow \alpha \mathbf{h}_{t-1} + (1-\alpha)\mathbf{h}_t$$

- 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 $\mathbf{g}$, after one step of backpropagation it would be $J\mathbf{g}$ and after $n$ steps it would be $J^n\mathbf{g}$

# 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 $J^n(\mathbf{g} + \delta\mathbf{v})$

# 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 $J^n(\mathbf{g} + \delta\mathbf{v})$
- Infact, the separation is exactly $\delta|\lambda|^n$

# 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 $J^n(\mathbf{g} + \delta\mathbf{v})$
- 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 $\mathbf{h}$, when a linear map $W$ always shrinks $\mathbf{h}$, the mapping is said to be contractive

# Echo State Networks

- For a vector $\mathbf{h}$, when a linear map $W$ always shrinks $\mathbf{h}$, 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 $\mathbf{h}$, when a linear map $W$ always shrinks $\mathbf{h}$, 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 $\mathbf{h}$, when a linear map $W$ always shrinks $\mathbf{h}$, 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 $\mathbf{h}$, when a linear map $W$ always shrinks $\mathbf{h}$, 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

# Echo State Networks

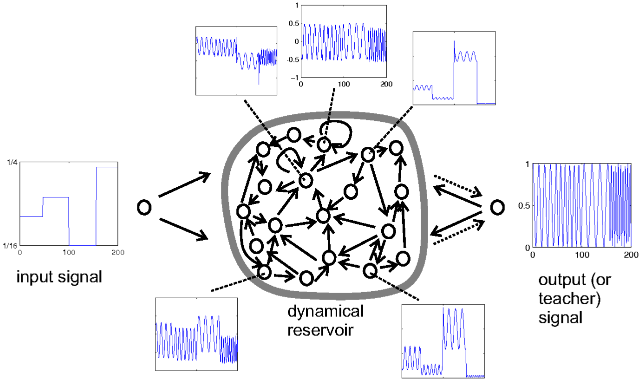



Figure: Scholarpedia

- Solid arrows represent fixed, random connections. Dashed arrows represent learnable weights

# A Popular Solution: Gated Architectures

# Back to Plain Vanilla RNN

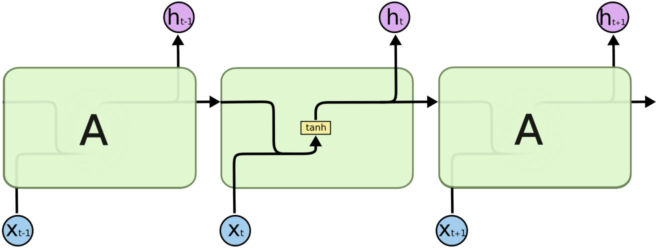



Figure: Chris Olah

# Long Short Term Memory

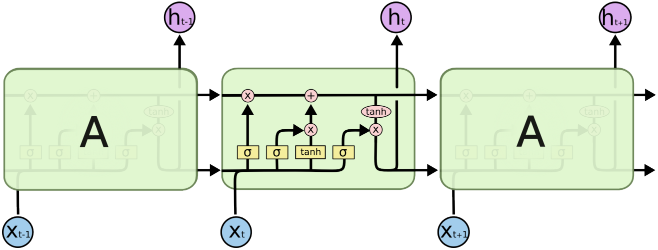



Figure: Chris Olah

- Proposed by Hochreiter and Schmidhuber (1997)

# Long Short Term Memory

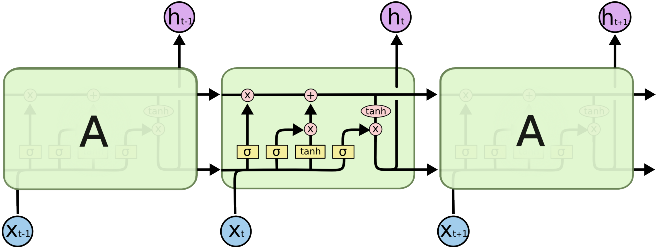
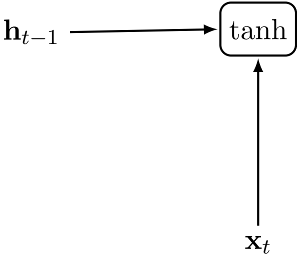



Figure: Chris Olah

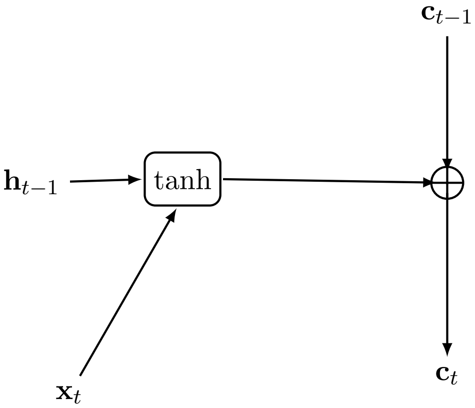
- Proposed by Hochreiter and Schmidhuber (1997)
- Now let's try to understand each memory cell!

# Long Short Term Memory



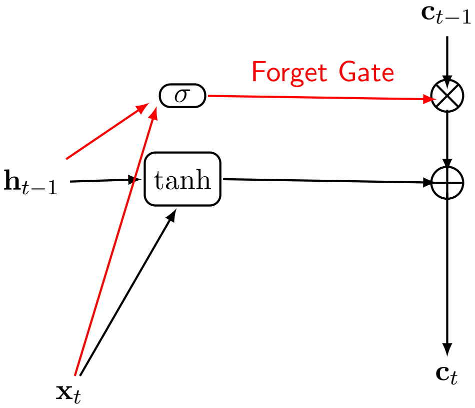

$$\mathbf{h}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$

# Long Short Term Memory




$$\tilde{\mathbf{c}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$
$$\mathbf{c}_t = \mathbf{c}_{t-1} + \tilde{\mathbf{c}}_t$$

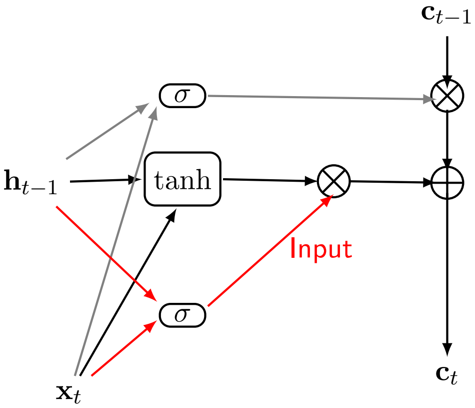
# Long Short Term Memory




$$f_t = \sigma(W_f \mathbf{h}_{t-1} + U_f \mathbf{x}_t)$$

$$\tilde{\mathbf{c}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$
$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + \tilde{\mathbf{c}}_t$$

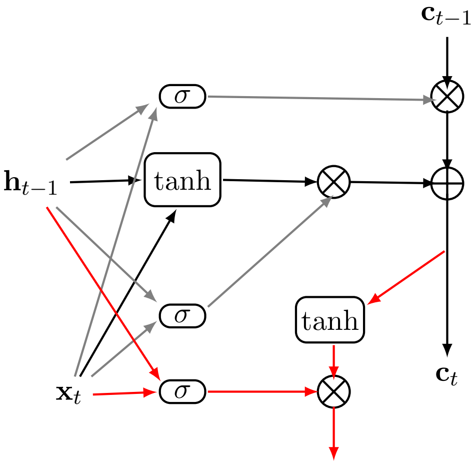
# Long Short Term Memory




$$f_t = \sigma(W_f \mathbf{h}_{t-1} + U_f \mathbf{x}_t)$$
$$i_t = \sigma(W_i \mathbf{h}_{t-1} + U_i \mathbf{x}_t)$$

$$\tilde{\mathbf{c}}_t = \tanh(W \mathbf{h}_{t-1} + U \mathbf{x}_t)$$
$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{\mathbf{c}}_t$$

# Long Short Term Memory




$$f_t = \sigma(W_f \mathbf{h}_{t-1} + U_f \mathbf{x}_t)$$
$$i_t = \sigma(W_i \mathbf{h}_{t-1} + U_i \mathbf{x}_t)$$
$$o_t = \sigma(W_o \mathbf{h}_{t-1} + U_o \mathbf{x}_t)$$

$$\tilde{\mathbf{c}}_t = \tanh(W \mathbf{h}_{t-1} + U \mathbf{x}_t)$$
$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{\mathbf{c}}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

# LSTM: Further Intuition

- The Cell State

$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{\mathbf{c}}_t \text{ with } \tilde{\mathbf{c}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$

# LSTM: Further Intuition

- The Cell State

$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{\mathbf{c}}_t \text{ with } \tilde{\mathbf{c}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$

- 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

$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{\mathbf{c}}_t \text{ with } \tilde{\mathbf{c}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$

- 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

$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{\mathbf{c}}_t \text{ with } \tilde{\mathbf{c}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$

- 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
- Gates are constructed by using a sigmoid and a pointwise multiplication

# LSTM: Further Intuition

- The Forget Gate

$$f_t = \sigma(W_f \mathbf{h}_{t-1} + U_f \mathbf{x}_t)$$

# LSTM: Further Intuition

- The Forget Gate

$$f_t = \sigma(W_f \mathbf{h}_{t-1} + U_f \mathbf{x}_t)$$

- Helps to decide what information to throw away from the cell state

# LSTM: Further Intuition

- The Forget Gate

$$f_t = \sigma(W_f \mathbf{h}_{t-1} + U_f \mathbf{x}_t)$$

- 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

$$i_t = \sigma(W_i \mathbf{h}_{t-1} + U_i \mathbf{x}_t)$$

# LSTM: Further Intuition

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

$$i_t = \sigma(W_i \mathbf{h}_{t-1} + U_i \mathbf{x}_t)$$

- We then update the cell state:

$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{\mathbf{c}}_t$$

# LSTM: Further Intuition

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

$$i_t = \sigma(W_i \mathbf{h}_{t-1} + U_i \mathbf{x}_t)$$

- We then update the cell state:

$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{\mathbf{c}}_t$$

- We then need to output, and use the output gate $o_t = \sigma(W_o \mathbf{h}_{t-1} + U_o \mathbf{x}_t)$ to pass on the filtered version

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

# Gated Recurrent Unit

- Let $\tilde{\mathbf{h}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$ and $\mathbf{h}_t = \tilde{\mathbf{h}}_t$

# Gated Recurrent Unit

- Let $\tilde{\mathbf{h}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$ and $\mathbf{h}_t = \tilde{\mathbf{h}}_t$
- Reset gate: $r_t = \sigma(W_r\mathbf{h}_{t-1} + U_r\mathbf{x}_t)$

# Gated Recurrent Unit

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

# Gated Recurrent Unit

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

# Gated Recurrent Unit

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

# Gated Recurrent Unit

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

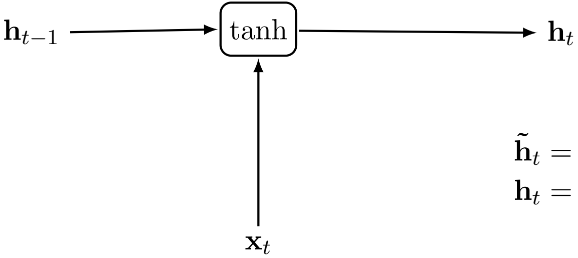
# Gated Recurrent Unit

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

# Gated Recurrent Unit

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

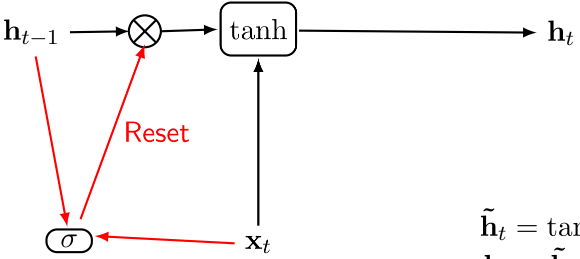
# Gated Recurrent Unit




$$\tilde{\mathbf{h}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$
$$\mathbf{h}_t = \tilde{\mathbf{h}}_t$$

# Gated Recurrent Unit

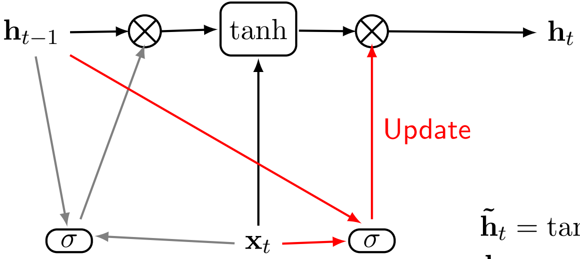
$$r_t = \sigma(W_r \mathbf{h}_{t-1} + U_r \mathbf{x}_t)$$




$$\tilde{\mathbf{h}}_t = \tanh(W(r_t \odot \mathbf{h}_{t-1}) + U\mathbf{x}_t)$$
$$\mathbf{h}_t = \tilde{\mathbf{h}}_t$$

# Gated Recurrent Unit

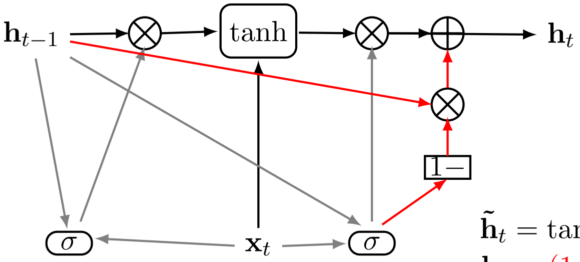
$$r_t = \sigma(W_r \mathbf{h}_{t-1} + U_r \mathbf{x}_t)$$
$$z_t = \sigma(W_z \mathbf{h}_{t-1} + U_z \mathbf{x}_t)$$




$$\tilde{\mathbf{h}}_t = \tanh(W(r_t \odot \mathbf{h}_{t-1}) + U\mathbf{x}_t)$$
$$\mathbf{h}_t = z_t \odot \tilde{\mathbf{h}}_t$$

# Gated Recurrent Unit

$$r_t = \sigma(W_r \mathbf{h}_{t-1} + U_r \mathbf{x}_t)$$
$$z_t = \sigma(W_z \mathbf{h}_{t-1} + U_z \mathbf{x}_t)$$




$$\tilde{\mathbf{h}}_t = \tanh(W(r_t \odot \mathbf{h}_{t-1}) + U\mathbf{x}_t)$$
$$\mathbf{h}_t = (1 - z_t) \odot \mathbf{h}_{t-1} + z_t \odot \tilde{\mathbf{h}}_t$$

## Attention Models

# 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.*

# 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.*
- Let us consider Machine Translation first

# Attention Models: Motivation

- Recall our encoder-decoder model for machine translation

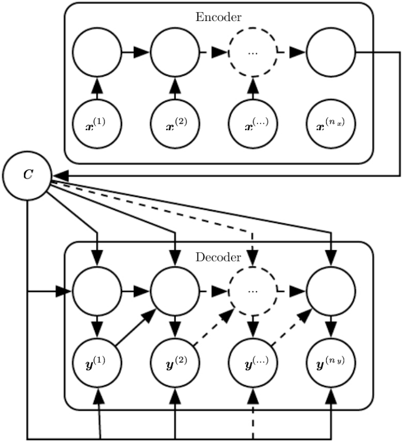



Figure: Goodfellow *et al.*

# Attention Models: Motivation

- Let's look at the steps for translation again:

# Attention Models: Motivation

- Let's look at the steps for translation again:
- The input sentence $\mathbf{x}_1, \ldots, \mathbf{x}_n$ via hidden unit activations $\mathbf{h}_1, \ldots, \mathbf{h}_n$ is encoded into the *thought vector* $C$

# Attention Models: Motivation

- Let's look at the steps for translation again:
- The input sentence $\mathbf{x}_1, \ldots, \mathbf{x}_n$ via hidden unit activations $\mathbf{h}_1, \ldots, \mathbf{h}_n$ is encoded into the *thought vector* $C$
- Using $C$, the decoder then *generates* the output sentence $\mathbf{y}_1, \ldots, \mathbf{y}_p$

# Attention Models: Motivation

- Let's look at the steps for translation again:
- The input sentence $\mathbf{x}_1, \ldots, \mathbf{x}_n$ via hidden unit activations $\mathbf{h}_1, \ldots, \mathbf{h}_n$ is encoded into the *thought vector* $C$
- Using $C$, the decoder then *generates* the output sentence $\mathbf{y}_1, \ldots, \mathbf{y}_p$
- 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 $\mathbf{x}_1, \dots, \mathbf{x}_n$ via hidden unit activations $\mathbf{h}_1, \dots, \mathbf{h}_n$ is encoded into the *thought vector* $C$
- Using $C$, the decoder then *generates* the output sentence $\mathbf{y}_1, \dots, \mathbf{y}_p$
- 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

# 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
- Maybe it would be more efficient to also be able to attend to these words *while decoding*

# Machine Translation Using Attention

- First Observation: For each word we had a hidden unit. This encoded a representation for each word.

# 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

# 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

# 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

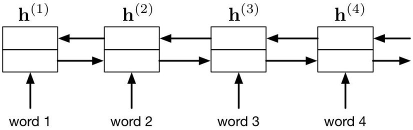



Figure: Roger Grosse

# 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

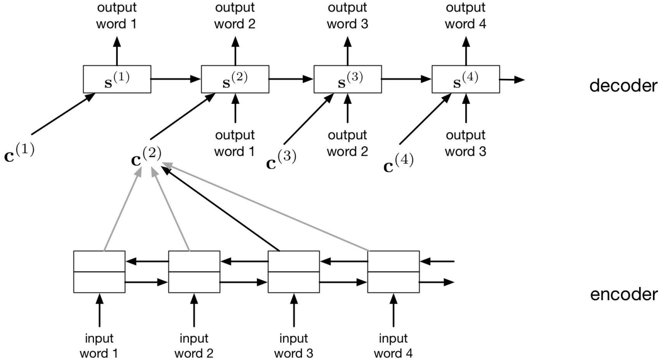



Figure: Roger Grosse

# Machine Translation Using Attention

- How do we *learn* $C^{(t)}$'s so that they attend to relevant words?

# Machine Translation Using Attention

- 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 $\mathbf{h}_i$

# Machine Translation Using Attention

- 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 $\mathbf{h}_i$
- Define $C^{(t)}$ to be the weighted average of encoder's representations:

$$C^{(t)} = \sum_i \alpha_{ti} \mathbf{h}_i$$

# Machine Translation Using Attention

- 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 $\mathbf{h}_i$
- Define $C^{(t)}$ to be the weighted average of encoder's representations:

$$C^{(t)} = \sum_i \alpha_{ti} \mathbf{h}_i$$

- $\alpha_t$ 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 the previous state of the decoder

# Machine Translation Using Attention

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

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

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

# 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 $e_{ti} = a(\mathbf{s}^{(t-1)}, \mathbf{h}^{(i)})$
- This is a form of content-based addressing

# Machine Translation Using Attention

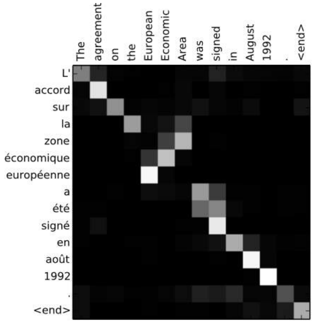
- $\alpha_{ti}$ is a function of the representations of the words, as well the previous state of the decoder

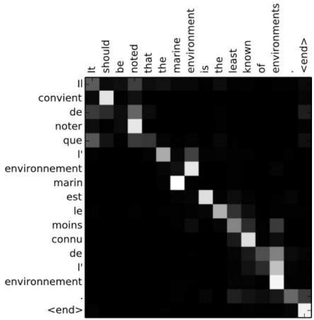
$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_k \exp(e_{tk})}$$

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

- 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

# 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

# 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

# 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

# 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

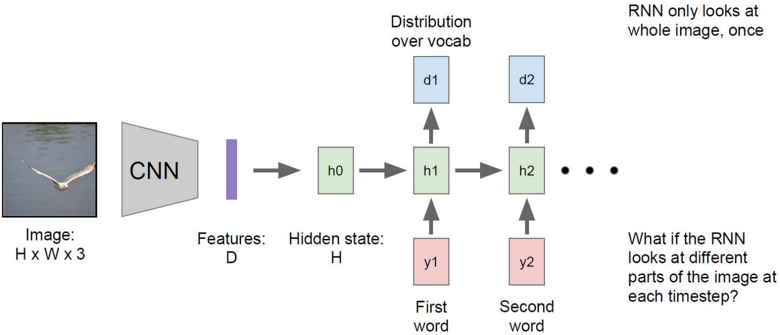



Figure: Andrej Karpathy

# Caption Generation with Attention

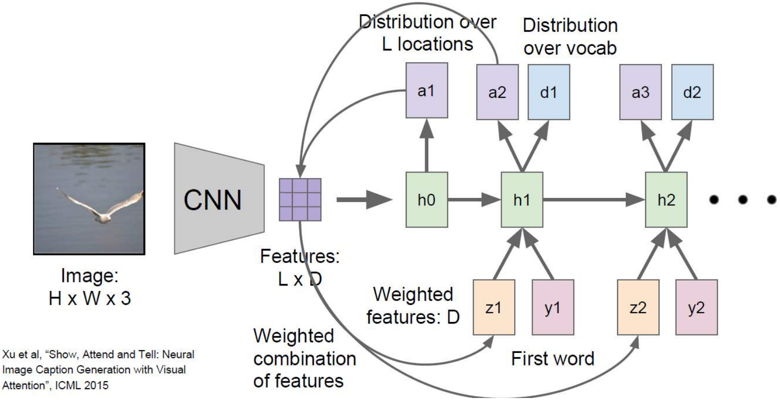
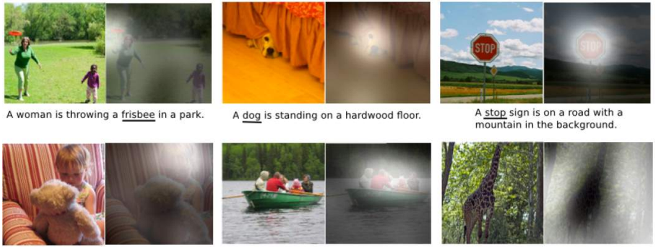



Figure: Andrej Karpathy

# Caption Generation using Attention

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


A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

# 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