Deep Learning

Lecture 8 – Sequence Models

Kumar Bipin

BE, MS, PhD (MMMTU, IISc, IIIT-Hyderabad)

Robotics, Computer Vision, Deep Learning, Machine Learning, System Software

Agenda

8.1 Recurrent Networks

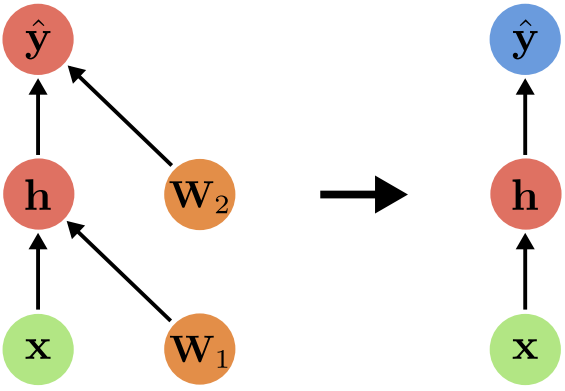
8.2 Recurrent Network Applications

8.3 Gated Recurrent Networks

8.4 Autoregressive Models

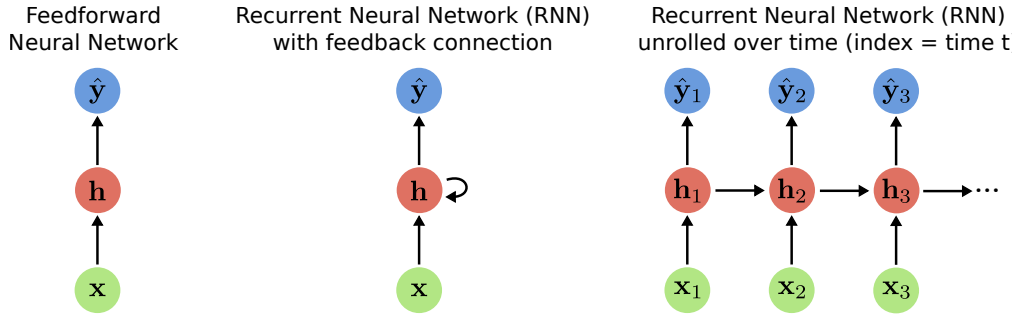
8.1
Recurrent Networks

Recap: Computation Graphs



- Example of a **feedforward neural network** (here: MLP with 1 hidden layer)
- In the following, we will drop all parameter nodes and highlight outputs in blue

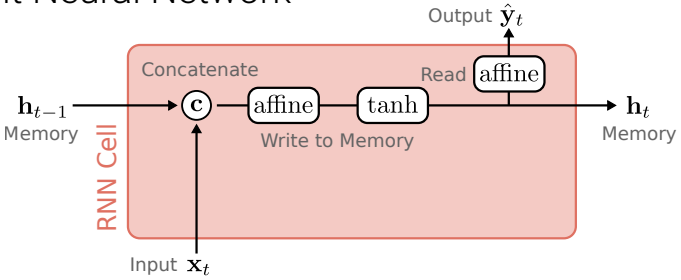
Feedforward vs. Recurrent Neural Networks



Recurrent Neural Networks (RNNs):

- Core idea: update **hidden state** $\mathbf{h}$ based on input and previous hidden state using same update rule (same/shared parameters) at each **time step**
- Allows for processing **sequences of variable length**, not only fixed-sized vectors
- **Infinite memory:** $\mathbf{h}$ is function of all previous inputs (long-term dependencies)

Basic Recurrent Neural Network



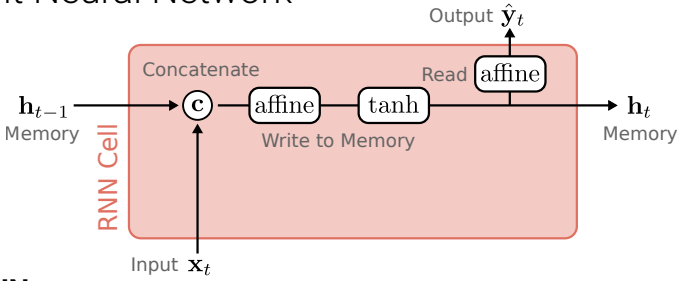
General Form:

$$\mathbf{h}_t = f_h(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$\hat{\mathbf{y}}_t = f_y(\mathbf{h}_t)$$

- We use t as the **time index** (in feedforward networks we used i as layer index)
- General form does not specify the form of the hidden and output mappings
- Important: f_h and f_y **do not change over time**, unlike in layers of feedforward net

Basic Recurrent Neural Network



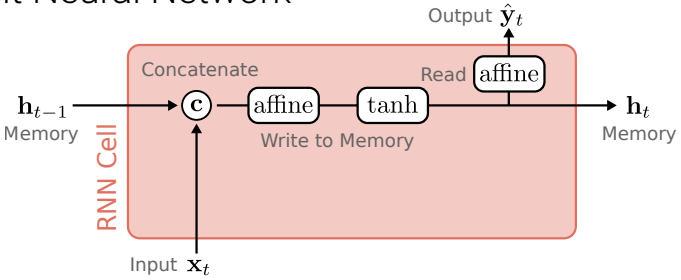
Single-Layer RNN:

$$\mathbf{h}_t = \tanh(\mathbf{A}_h \mathbf{h}_{t-1} + \mathbf{A}_x \mathbf{x}_t + \mathbf{b}_h)$$

$$\hat{\mathbf{y}}_t = \mathbf{A}_y \mathbf{h}_t + \mathbf{b}_y$$

- Hidden state $\mathbf{h}_t$ = linear combination of input $\mathbf{x}_t$ and previous hidden state $\mathbf{h}_{t-1}$
- Output $\hat{\mathbf{y}}_t$ = linear prediction based on current hidden state $\mathbf{h}_t$
- $\tanh(\cdot)$ is commonly used as activation function (data is in the range $[-1, 1]$)
- Parameters $\mathbf{A}_h$, $\mathbf{A}_x$, $\mathbf{A}_y$, $\mathbf{b}_h$, $\mathbf{b}_y$ are constant over time

Basic Recurrent Neural Network



Single-Layer RNN in Einstein Notation:

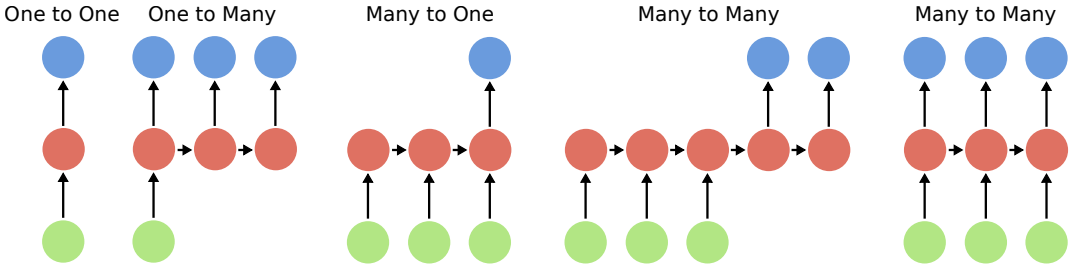
$$H_t[b, c_{out}] = \tanh\left(A_h[c_{out}, C_{in}]\, H_{t-1}[b, C_{in}] \;+\; A_x[c_{out}, C_{in}]\, X_t[b, C_{in}] \;+\; b[c_{out}]\right)$$

$$= \tanh\left(A[c_{out}, C_{in}] \begin{pmatrix} H_{t-1}[b, C_{in}] \\ X_t[b, C_{in}] \end{pmatrix} \;+\; b[c_{out}]\right)$$

$$\hat{Y}_t[b, c_{out}] = A_y[c_{out}, C_{in}]\, H_t[b, C_{in}] + b[c_{out}]$$

Note: c_{in} and c_{out} are (per-product) index names (not dimensionality). $\mathbf{A}_h$ is square.

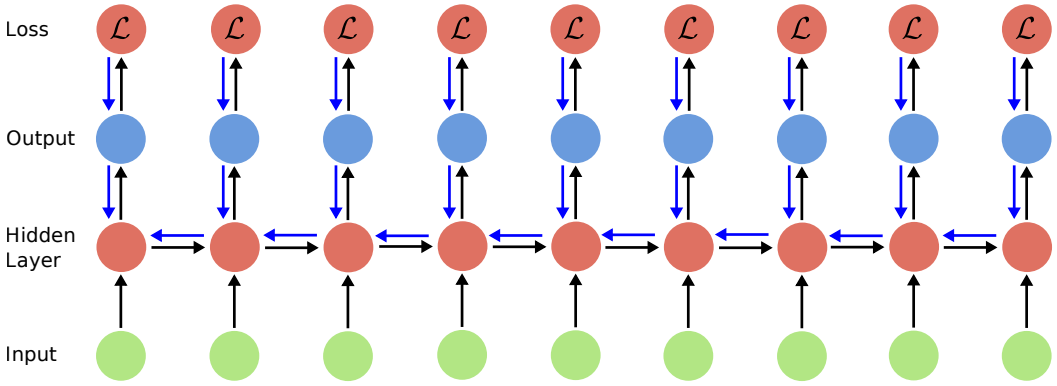
Mapping Types



RNNs allow for processing **variable length** inputs and outputs:

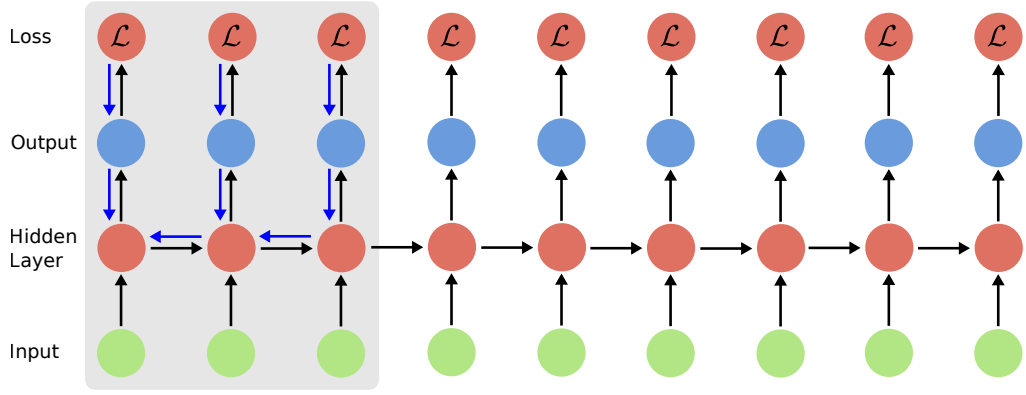
- **One to Many:** Image captioning (image to sentence)
- **Many to One:** Action recognition (video to action)
- **Many to Many:** Machine translation (sentence to sentence)
- **Many to Many:** Object tracking (video to object location per frame)
- To determine the length of the output sequence, a **stop symbol** can be predicted

Backpropagation through Time



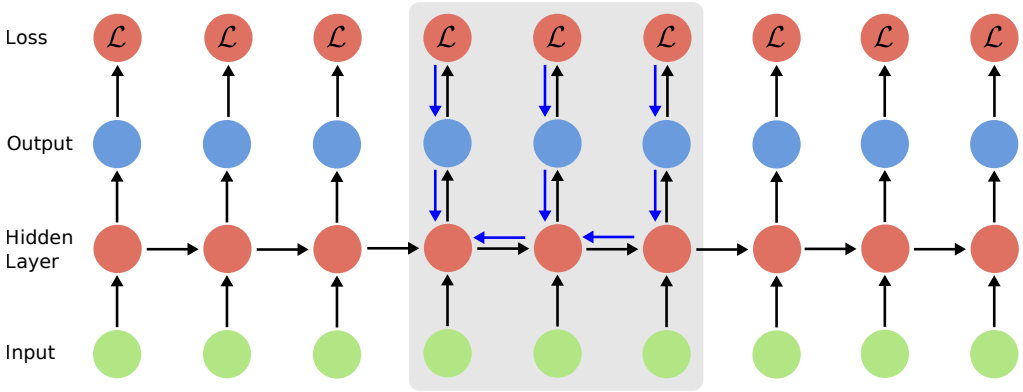
- To train RNNs, we **backpropagate gradients through time**
- As all hidden RNN cells share their parameters, gradients get accumulated
- However, very quickly intractable (memory) for larger sequences (e.g., Wikipedia)

Truncated Backpropagation through Time



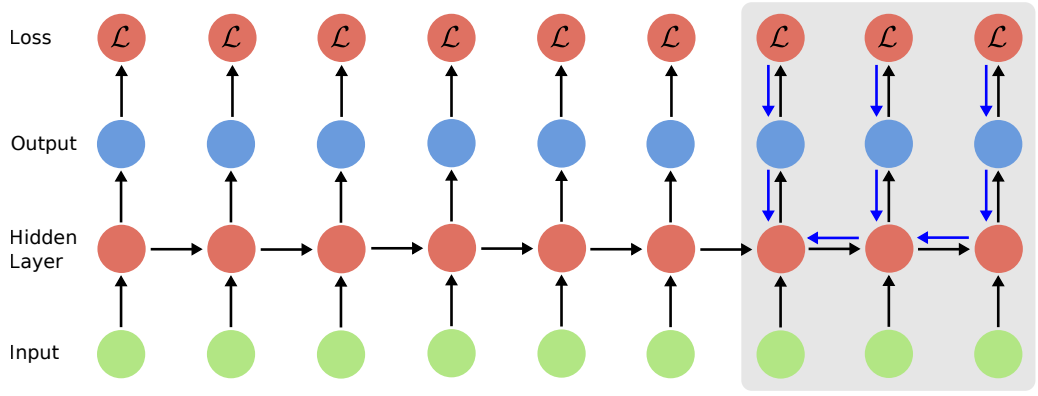
- Thus, one typically uses **truncated backpropagation through time** in practice
- Carry hidden states forward in time forever, but stop backpropagation earlier
- Total loss is sum of individual loss functions (= negative log likelihood)

Truncated Backpropagation through Time



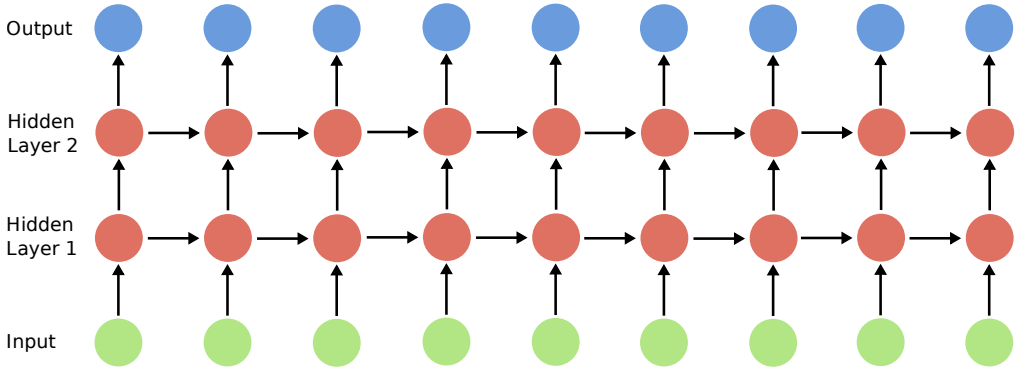
- Thus, one typically uses **truncated backpropagation through time** in practice
- Carry hidden states forward in time forever, but stop backpropagation earlier
- Total loss is sum of individual loss functions (= negative log likelihood)

Truncated Backpropagation through Time



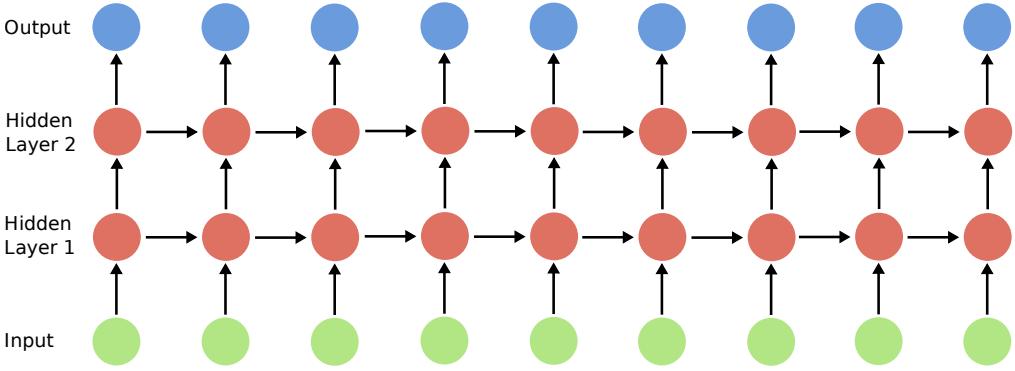
- Thus, one typically uses **truncated backpropagation through time** in practice
- Carry hidden states forward in time forever, but stop backpropagation earlier
- Total loss is sum of individual loss functions (= negative log likelihood)

Multi-Layer RNNs



- Deeper **multi-layer RNNs** can be constructed by stacking RNN layers
- An alternative is to make each individual computation (=RNN cell) deeper
- Today, often combined with residual connections in vertical direction

Multi-Layer RNNs



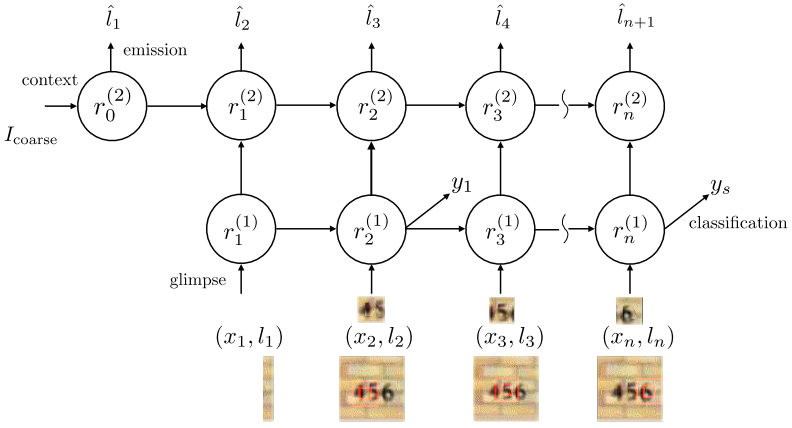
$$H_t^1[b, c_{out}] = \tanh\left(A_h^1[c_{out}, C_{in}]\, H_{t-1}^1[b, C_{in}] + A_x^1[c_{out}, C_{in}]\, X_t[b, C_{in}] + b[c_{out}]\right)$$

$$H_t^2[b, c_{out}] = \tanh\left(A_h^2[c_{out}, C_{in}]\, H_{t-1}^2[b, C_{in}] + A_x^2[c_{out}, C_{in}]\, H_t^1[b, C_{in}] + b[c_{out}]\right)$$

$$\hat{Y}_t[b, c_{out}] = A_y[c_{out}, C_{in}]\, H_t^2[b, C_{in}] + b[c_{out}]$$

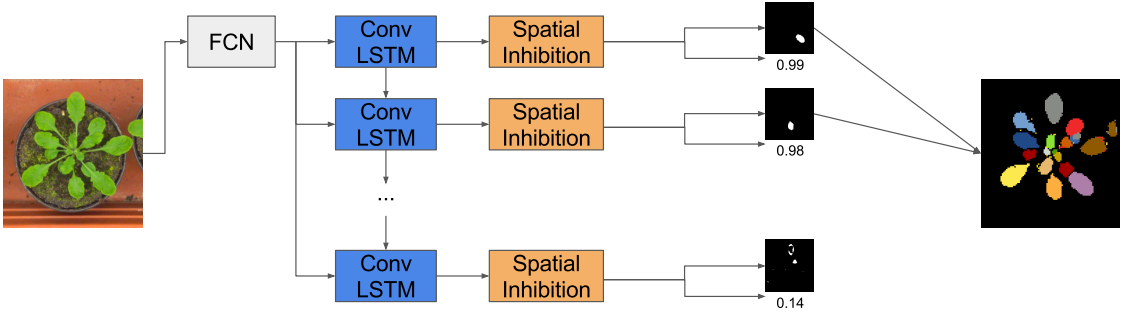
8.2
Recurrent Network Applications

Multiple Object Recognition



- At each time step, perceive a **glimpse** (= image region) and predict a **saccade**

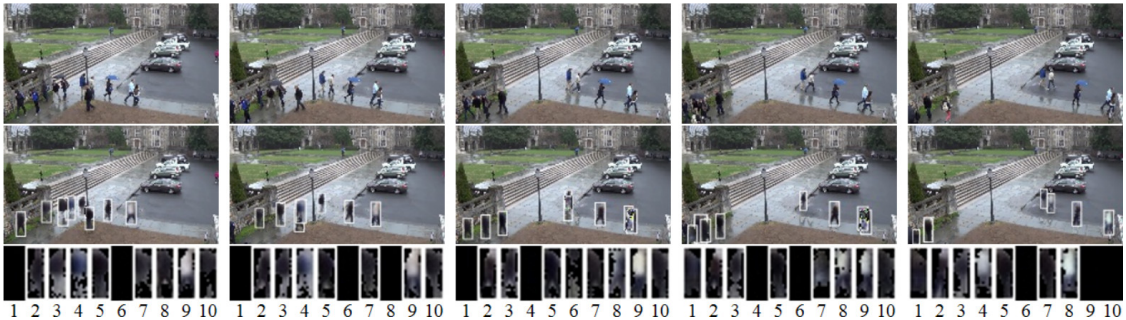
Recurrent Instance Segmentation



- At each time step, **segment** the next (not yet segmented) **part** of an object

Romera-Paredes, Torr: Recurrent Instance Segmentation. ECCV, 2016.

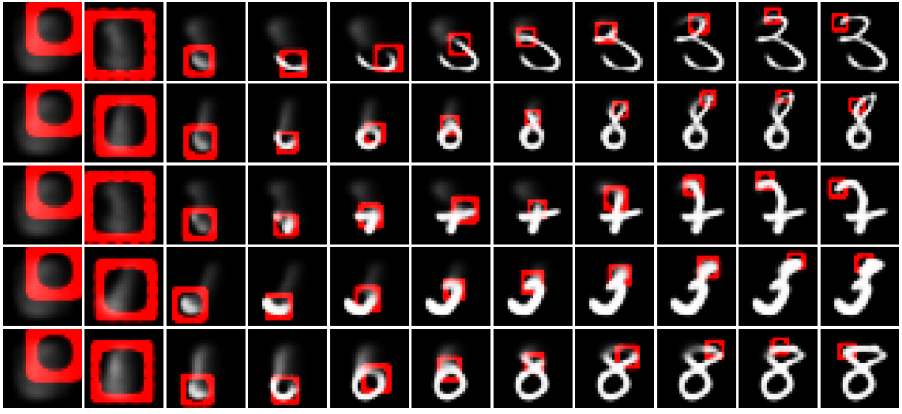
Object Tracking



- **Tracking of multiple objects** by updating each object's hidden state using an RNN

He, Li, Liu, He and Barber: Tracking by Animation: Unsupervised Learning of Multi-Object Attentive Trackers. CVPR, 2019.

Image Generation



- Model for **sequential image generation** (red rectangle = attended region)
- Demonstrates that RNNs can also process/generate non-sequential data

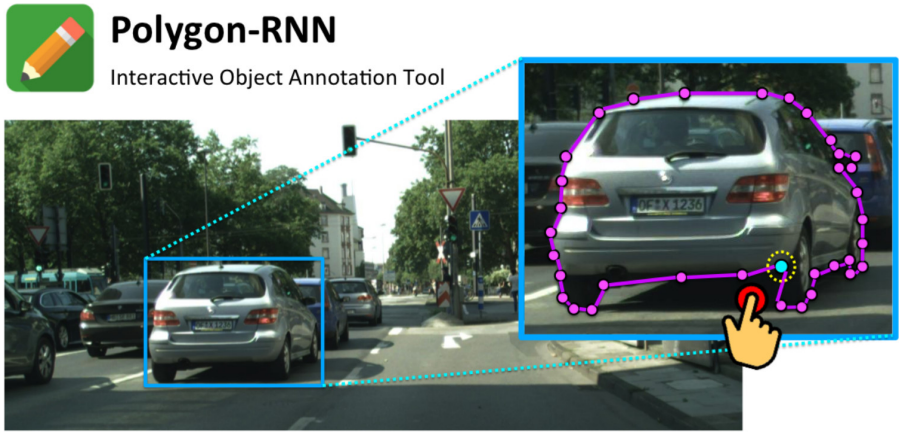
Gregor, Danihelka, Graves, Rezende and Wierstra: DRAW: A Recurrent Neural Network For Image Generation. ICML, 2015.

Image Generation

- Generated images based on partially occluded inputs
- Demonstrates that RNNs can also process/generate non-sequential data

Oord, Kalchbrenner and Kavukcuoglu: Pixel Recurrent Neural Networks. ICML, 2016.

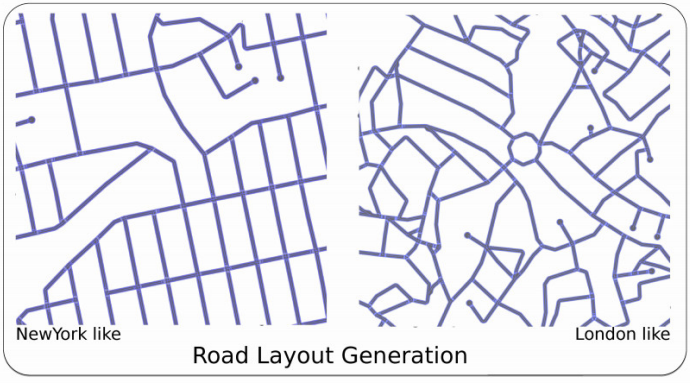
Image Annotation



- Iteratively **annotate** the outline of 2D object instances in an image with **polygons**

Castrejon, Kundu, Urtasun and Fidler: Annotating Object Instances with a Polygon-RNN. CVPR, 2017.

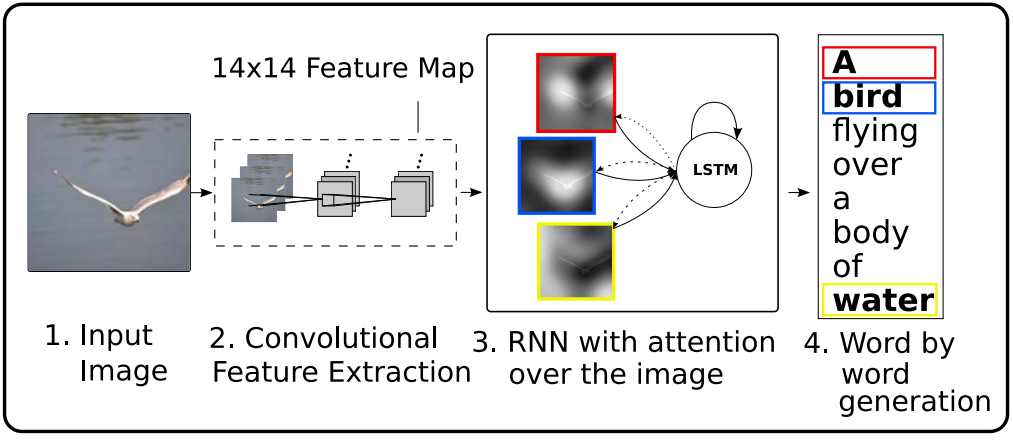
Modeling Road Layouts



- Iteratively generate or infer **road layouts** as spatial graphs

Chu et al.: Neural Turtle Graphics for Modeling City Road Layouts. ICCV, 2019.

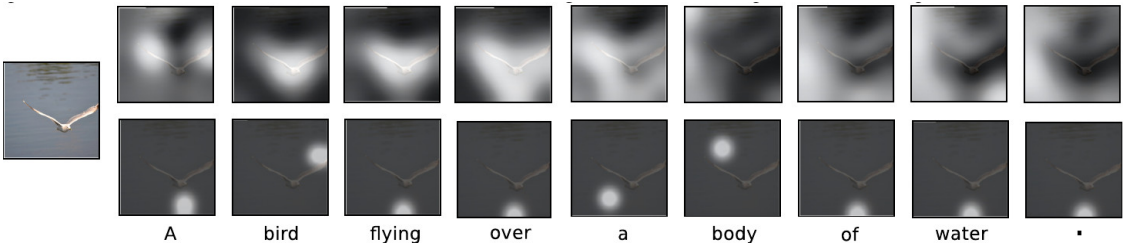
Image Captioning



- Generate **image description** by sequentially looking at an image

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

Image Captioning



- **Attention over time**
- Top: soft attention. Bottom: hard attention

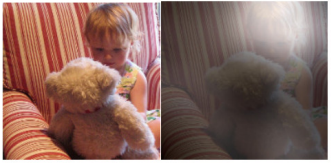
Xu et al.: Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. ICML, 2015.

Image Captioning

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.

- Successful caption generations

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

Image Captioning

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.

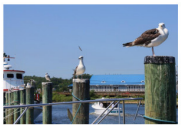
A woman is sitting at a table with a large pizza.

A man is talking on his cell phone while another man watches.

- Wrongly generated captions. Attention can reveal insights into what went wrong.

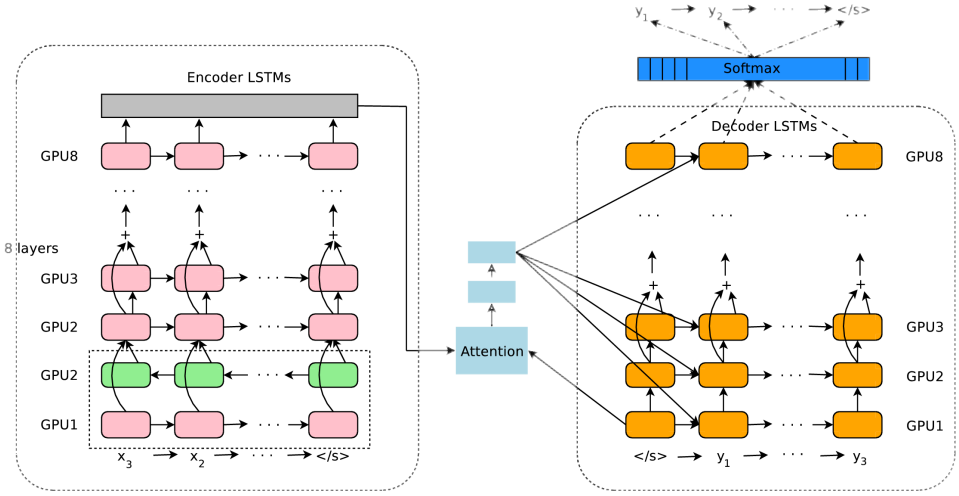
Xu et al.: Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. ICML, 2015.

Visual Question Answering

Image						
Multiple Choices	**Q: Who is behind the batter?** **A: Catcher.** A: Umpire. A: Fans. A: Ball girl.	**Q: What adorns the tops of the post?** **A: Gulls.** A: An eagle. A: A crown. A: A pretty sign.	**Q: How many cameras are in the photo?** **A: One.** A: Two. A: Three. A: Four.	**Q: Why is there rope?** **A: To tie up the boats.** A: To tie up horses. A: To hang people. A: To hit tether balls.	**Q: What kind of stuffed animal is shown?** **A: Teddy Bear.** A: Monkey. A: Tiger. A: Bunny rabbit.	**Q: What animal is being petted?** **A: A sheep.** A: Goat. A: Alpaca. A: Pig.
w/ Image	H: Catcher. ✓ M: Umpire. ✗	H: Gulls. ✓ M: Gulls. ✓	H: Three. ✗ M: One. ✓	H: To hit tether balls. ✗ M: To hang people. ✗	H: Monkey. ✗ M: Teddy Bear. ✓	H: A sheep. ✓ M: A sheep. ✓
w/o Image	H: Catcher. ✓ M: Catcher. ✓	H: Gulls. ✓ M: A crown. ✗	H: One. ✓ M: One. ✓	H: To tie up the boats. ✓ M: To hang people. ✗	H: Teddy Bear. ✓ M: Teddy Bear. ✓	H: Goat. ✗ M: A sheep. ✓

► Input: Image and question sentence. Output: answer sentence.

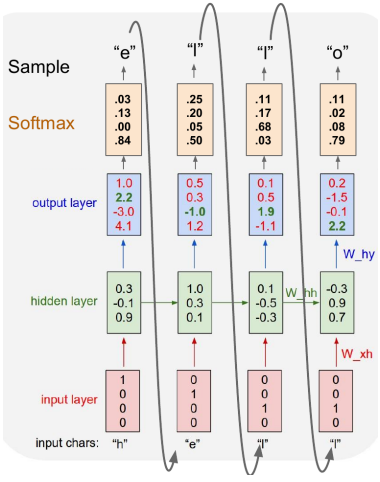
Neural Machine Translation



Wu et al.: Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. Arxiv, 2016.

Character-level Language Models

- Generating natural language text **character by character** with an RNN
- In the example on the right: Alphabet with 4 characters ('h', 'e', 'l', 'o')
- Each character is represented by a **1 hot vector**, e.g.: $(1, 0, 0, 0)^\top$
- Model predicts **distribution over next character** via softmax function
- Character drawn from distribution is **fed as input** to RNN at next time step



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Character-level Language Models

- 3-layer RNN
- 512 hidden nodes
- Trained on all works of William Shakespeare
- 4.4 Mio characters

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Character-level Language Models

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Character-level Language Models

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma **??**. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

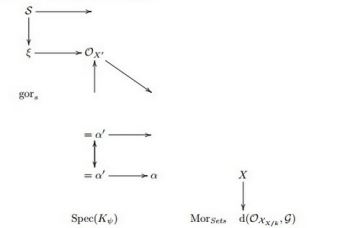
be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type $f_\star$. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

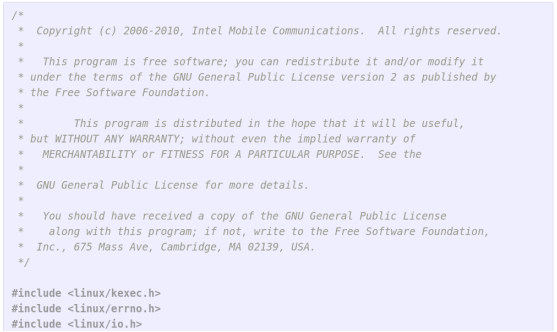
Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas **??**.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \text{ -1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_i}^{-1}\mathcal{O}_{X_i}(\mathcal{O}_{X_s}^{\mathrm{Pr}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition **??** and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma **??**. This is a sequence of $\mathcal{F}$ is a similar morphism.

Character-level Language Models



```
/*
 *  Copyright (c) 2006-2010, Intel Mobile Communications.  All rights reserved.
 *
 *   This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 *       This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *
 *  GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *    along with this program; if not, write to the Free Software Foundation,
 *  Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
```

```
#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)     (func)

#define SWAP_ALLOCATE(nr)     (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0));   \
  if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
          pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
  PUT_PARAM_RAID(2, sel) = get_state_state();
  set_pid_sum((unsigned long)state, current_state_str(),
           (unsigned long)-1->lr_full; low;
}
```

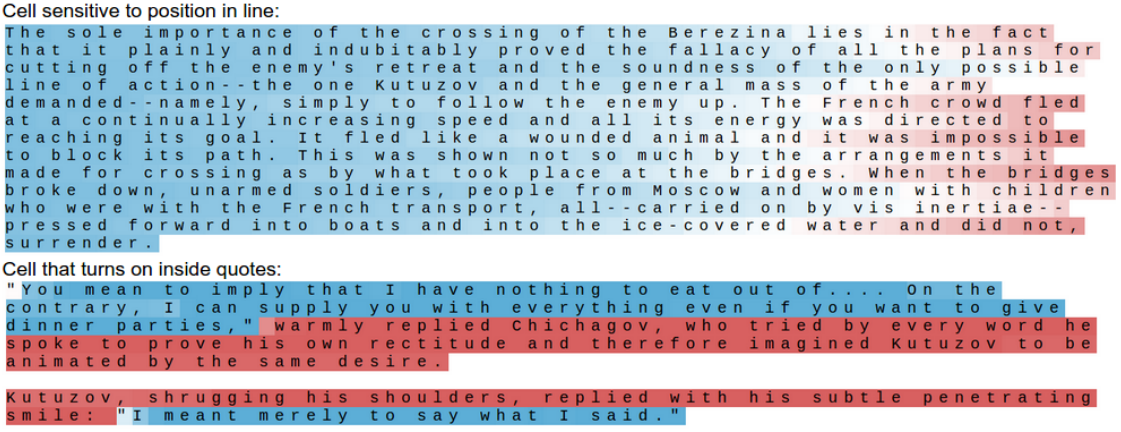
- 3-layer RNN trained for several days on **Linux source code** (474 MB)
- Sampled code snippets do not compile but look reasonable overall
- Learned that code starts with license, uses correct syntax, adds comments

Character-level Language Models

Samples from RNN trained on 8000 baby names:

Rudi Levette Berice Lussa Hany Mareanne Chrestina Carissy Marylen Hammine Janye
Marlise Jacacrie Hendred Romand Charienna Nenotto Ette Dorane Wallen Marly
Darine Salina Elvyn Ersia Maralena Minoria Ellia Charmin Antley Nerille Chelon Walmor
Evena Jeryly Stachon Charisa Allisa Anatha Cathanie Geetra Alexie Jerin Cassen
Herbett Cossie Velen Daurenge Robester Shermond Terisa Licia Roselen Ferine Jayn
Lusine Charyanne Sales Sanny Resa Wallon Martine Merus Jelen Candica Wallin Tel
Rachene Tarine Ozila Ketia Shanne Arnande Karella Roselina Alessia Chasty Deland
Berther Geamar Jackein Mellisand Sagdy Nenc Lessie Rasemy Guen Gavi Milea
Anneda Margoris Janin Rodelin Zeanna Elyne Janah Ferzina Susta Pey Castina

Character-level Language Models



- The behavior of some hidden neurons ($\sim 5\%$) is logical and human interpretable

Character-level Language Models

Cell that robustly activates inside if statements:

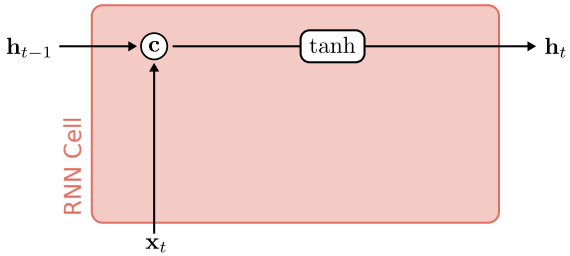
```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

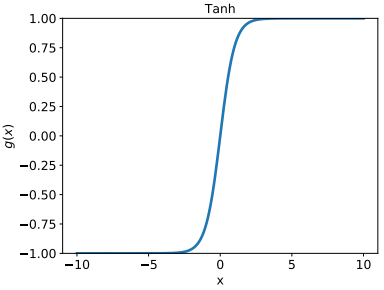
A large portion of cells are not easily interpretable. Here is a typical example:

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /* Of the currently implemented string fields, PATH_MAX
   * defines the longest valid length.
   */
```

8.3
Gated Recurrent Networks

Recurrent Neural Network

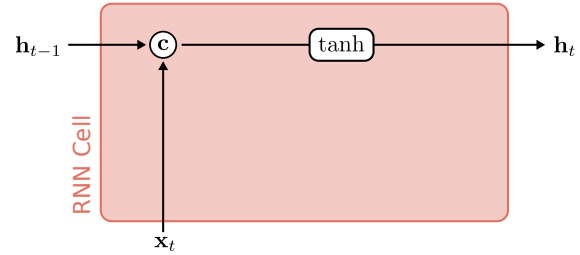




$$H_t[b, c_{out}] = \tanh\left(A_h[c_{out}, C_{in}]\, H_{t-1}[b, C_{in}] \;+\; A_x[c_{out}, C_{in}]\, X_t[b, C_{in}] \;+\; b[c_{out}]\right)$$

- The state update H_t is modeled using a zero-centered $\tanh(\cdot)$
- $\tanh(\cdot)$ assumes that the processed data is in the range $[-1, 1]$
- Remark: we omit the affine transformations and the output layer for clarity

Vanishing / Exploding Gradients



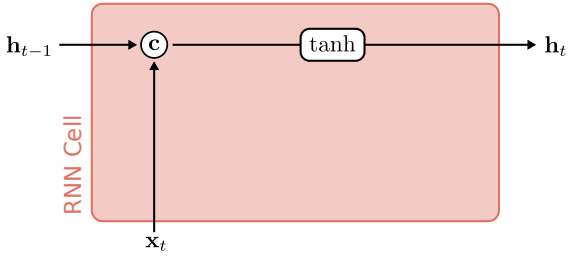
$$h_t = \tanh\left(a_h\, h_{t-1} \;+\; a_x\, x_t \;+\; b\right)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh' a_h \;\text{ with }\; \tanh' = \frac{\partial \tanh(x)}{\partial x}$$

What is the problem with vanilla RNNs?

- Let us consider an RNN with one dimensional hidden state $h_t \in \mathbb{R}$
- We have: $$\frac{\partial h_t}{\partial h_{t-k}} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{t-k+1}}{\partial h_{t-k}} = \left(\prod_{i=t-k+1}^{t} \tanh'_i \right) a_h^k$$
- Thus, the gradient vanishes if $\tanh(\cdot)$ saturates as in feedforward networks
- RNNs require careful initialization to avoid saturating activation functions

Vanishing / Exploding Gradients



$$h_t = \tanh\left(a_h\, h_{t-1} + a_x\, x_t + b\right)$$
$$\approx a_h\, h_{t-1} + a_x\, x_t + b$$
$$\frac{\partial h_t}{\partial h_{t-1}} \approx a_h$$

However, gradients might still misbehave:

- Let us now assume that the RNN has been initialized well such that the activation functions are not saturated $\Rightarrow a_h\, h_{t-1} + a_x\, x_t + b \in [-1, 1] \Rightarrow \tanh(x) \approx x$
- We now have:

$$\frac{\partial h_t}{\partial h_{t-k}} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{t-k+1}}{\partial h_{t-k}} = a_h^k$$

Vanishing / Exploding Gradients

$$\frac{\partial h_t}{\partial h_{t-k}} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{t-k+1}}{\partial h_{t-k}} = a_h^k$$

For $a_h > 1$ gradients will **explode** (become very large, cause divergence):

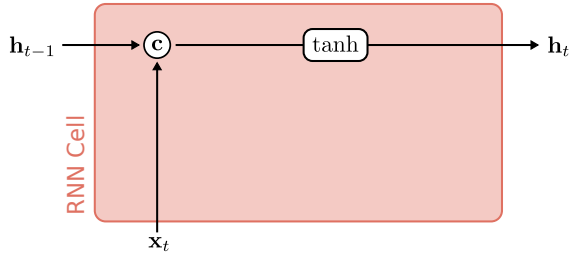
- Example: For $a_h = 1.1$ and $k = 100$ we have $\partial h_t / \partial h_{t-k} = a_h^k = 13781$
- This problem is often addressed in practice using **gradient clipping**
- Forward values do not explode due to bounded $\tanh(\cdot)$ activation function

For $a_h < 1$ gradients will **vanish** (no learning in earlier time steps):

- Example: For $a_h = 0.9$ and $k = 100$ we have $\partial h_t / \partial h_{t-k} = a_h^k = 0.0000266$
- Avoiding this problem requires an **architectural change**
- But residual connections do not work here as the parameters are shared across time and the input and desired output at each time step are different

Vanishing / Exploding Gradients

Similar situation for vector-valued hidden states:



$$\mathbf{h}_t = \tanh\left(\mathbf{A}_h\,\mathbf{h}_{t-1} + \mathbf{A}_x\,\mathbf{x}_t + \mathbf{b}\right)$$

$$\approx \mathbf{A}_h\,\mathbf{h}_{t-1} + \mathbf{A}_x\,\mathbf{x}_t + \mathbf{b}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \approx \mathbf{A}_h \quad \Rightarrow \quad \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-k}} \approx \mathbf{A}_h^k$$

- Let $\mathbf{A}_h = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}$ be the eigendecomposition of the square matrix $\mathbf{A}_h$
- We have $\mathbf{A}_h^k = \left(\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}\right)^k = \mathbf{Q}\mathbf{\Lambda}^k\mathbf{Q}^{-1}$ with diagonal eigenvalue matrix $\mathbf{\Lambda}$
- Components with eigenvalue $< 1 \Rightarrow$ vanishing gradient
- Components with eigenvalue $> 1 \Rightarrow$ exploding gradient
- Note that $\mathbf{A}_h$ is shared across time (unlike in layers of a feedforward network)

Gradient Clipping

The effect of exploding gradients can be dampened by a simple heuristic which clips the gradients to $\|\texttt{A.grad}\|_2 \leq \tau$ before applying the gradient update during SGD:

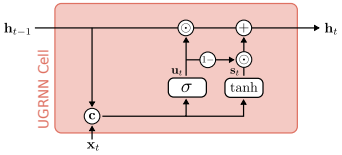
$$\texttt{A.grad} = \begin{cases} \texttt{A.grad} & \text{if } \|\texttt{A.grad}\|_2 \leq \tau \\ \tau \frac{\texttt{A.grad}}{\|\texttt{A.grad}\|_2} & \text{otherwise} \end{cases}$$

The maximal gradient magnitude τ is a hyperparameter, often $\tau \in [1, 10]$.

What can we do to avoid vanishing gradients?

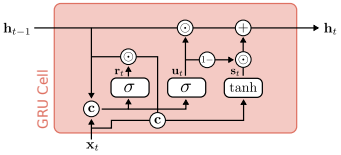
Gated Recurrent Networks

UGRNN



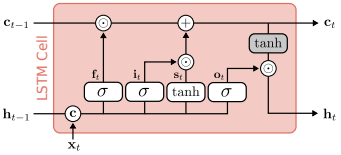
Collins, 2017

GRU



Cho, 2014

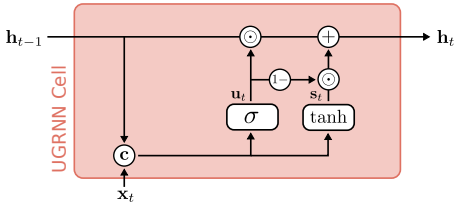
LSTM



Hochreiter, 1997

- **UGRNN:** Update Gate Recurrent Neural Network
- **GRU:** Gated Recurrent Unit
- **LSTM:** Long Short-Term Memory
- LSTM was the first and most transformative (revolutionized NLP in 2015, e.g. at Google), but also most complex model. UGRNN and GRU work similarly well.
- Common to all architectures: **gates** for filtering information

Update Gate RNN



$$U_t[b, c_{out}] = \sigma\left(A_{uh}[c_{out}, C_{in}]\, H_{t-1}[b, C_{in}] + A_{ux}[c_{out}, C_{in}]\, X_t[b, C_{in}] + b_u[c_{out}]\right)$$

$$S_t[b, c_{out}] = \tanh\left(A_{sh}[c_{out}, C_{in}]\, H_{t-1}[b, C_{in}] + A_{sx}[c_{out}, C_{in}]\, X_t[b, C_{in}] + b_s[c_{out}]\right)$$

$$H_t[b, c_{out}] = U_t[b, c_{out}]\, H_{t-1}[b, c_{out}] + (1 - U_t[b, c_{out}])\, S_t[b, c_{out}]$$

- U_t is called **update gate** as it determines if the hidden state H is updated or not
- S_t is the next **target state** that is added to H_{t-1} with element-wise weights U_t
- Remark: Gates use sigmoid ($\in [0, 1]$), state computation uses tanh ($\in [-1, 1]$)

Hadamard Product

This expression

$$H_t[b, c_{out}] = U_t[b, c_{out}]\, H_{t-1}[b, c_{out}] + (1 - U_t[b, c_{out}])\, S_t[b, c_{out}]$$

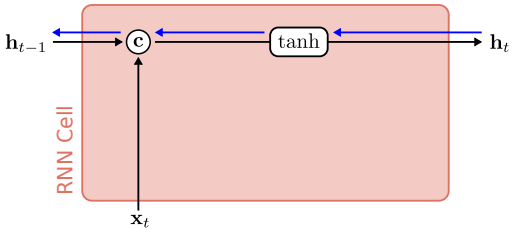
can be written as

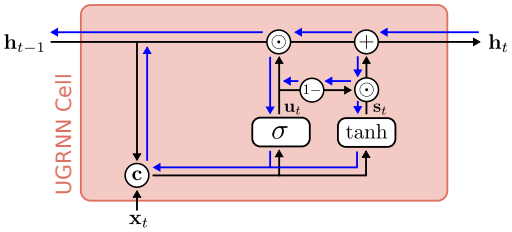
$$H_t[b, C_{out}] = U_t[b, C_{out}] \odot H_{t-1}[b, C_{out}] + (1 - U_t[b, C_{out}]) \odot S_t[b, C_{out}]$$

where $\odot$ denotes the **Hadamard product** (elementwise product).

We use this symbol to denote elementwise products in this lecture.

Gradient Flow





$$\mathbf{h}_t = \tanh\left(\mathbf{A}_h\mathbf{h}_{t-1} + \mathbf{A}_x\mathbf{x}_t + \mathbf{b}\right)$$

$$\mathbf{h}'_t = \tanh'\,\mathbf{A}_h \approx \mathbf{A}_h$$

with $\mathbf{h}'_t = \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$

$$\mathbf{u}_t = \sigma\left(\mathbf{A}_{uh}\mathbf{h}_{t-1} + \mathbf{A}_{ux}\mathbf{x}_t + \mathbf{b}_u\right)$$

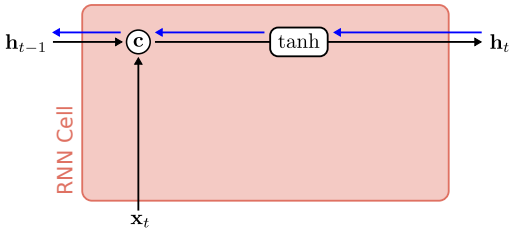
$$\mathbf{s}_t = \tanh\left(\mathbf{A}_{sh}\mathbf{h}_{t-1} + \mathbf{A}_{sx}\mathbf{x}_t + \mathbf{b}_s\right)$$

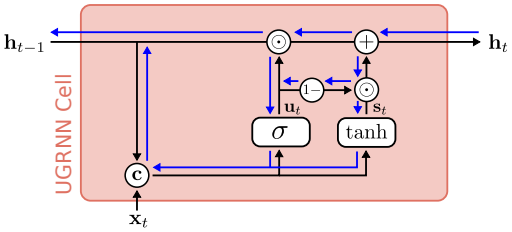
$$\mathbf{h}_t = \mathbf{u}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{u}_t) \odot \mathbf{s}_t$$

$$\mathbf{h}'_t = \mathbf{u}'_t \odot \mathbf{h}_{t-1} + \mathbf{u}_t + (1 - \mathbf{u}'_t) \odot \mathbf{s}_t + ..$$

- UGRNN can maintain gradient flow despite small $\mathbf{A}_h$ by setting its gate to $u \approx 1$

Gradient Flow





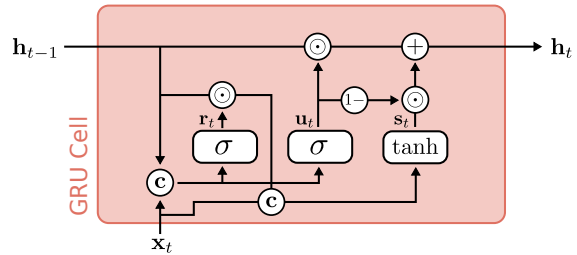
Cell that turns on inside quotes:

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

- UGRNN is able to **keep the state** of a variable **over a long time horizon** $(u \approx 1)$
- For example, it can implement a logic to keep track of being inside a quote or not

Gated Recurrent Unit

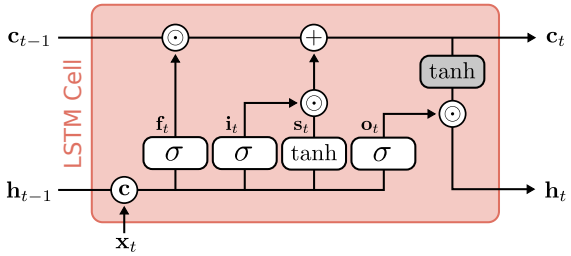


$$\mathbf{r}_t = \sigma(\mathbf{W}_{rh}\,\mathbf{h}_{t-1} + \mathbf{W}_{rx}\,\mathbf{x}_t + \mathbf{b}_r)$$
$$\mathbf{u}_t = \sigma(\mathbf{W}_{uh}\,\mathbf{h}_{t-1} + \mathbf{W}_{ux}\,\mathbf{x}_t + \mathbf{b}_u)$$
$$\mathbf{s}_t = \tanh(\mathbf{W}_{sh}\,(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{W}_{sx}\,\mathbf{x}_t + \mathbf{b}_s)$$
$$\mathbf{h}_t = \mathbf{u}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{u}_t) \odot \mathbf{s}_t$$

- **Reset gate** controls which parts of the state are used to compute next target state
- **Update gate** controls how much information to pass from previous time step

Cho et al.: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. EMNLP, 2014

Long Short-Term Memory



$$\mathbf{f}_t = \sigma(\mathbf{W}_{fh}\,\mathbf{h}_{t-1} + \mathbf{W}_{fx}\,\mathbf{x}_t + \mathbf{b}_f)$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_{ih}\,\mathbf{h}_{t-1} + \mathbf{W}_{ix}\,\mathbf{x}_t + \mathbf{b}_i)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{oh}\,\mathbf{h}_{t-1} + \mathbf{W}_{ox}\,\mathbf{x}_t + \mathbf{b}_o)$$
$$\mathbf{s}_t = \tanh(\mathbf{W}_{sh}\,\mathbf{h}_{t-1} + \mathbf{W}_{sx}\,\mathbf{x}_t + \mathbf{b}_s)$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{s}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Passes along an additional cell state $\mathbf{c}$ in addition to the hidden state $\mathbf{h}$. Has 3 gates:

- **Forget gate** determines information to erase from cell state
- **Input gate** determines which values of cell state to update
- **Output gate** determines which elements of cell state to reveal at time t

Remark: Cell update $\tanh(\cdot)$ creates new target values $\mathbf{s}_t$ for cell state

Hochreiter and Schmidhuber: Long Short-Term Memory. Neural Computation, 1997.

UGRNN vs. GRU vs. LSTM

UGRNN:

- One gate
- Expose entire state
- Single update gate
- Few parameters

GRU:

- Two gates
- Expose entire state
- Single update gate
- Medium parameters

LSTM:

- Three gates
- Control exposure
- Input/forget gates
- Many parameters

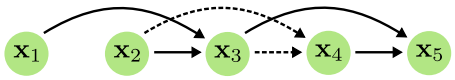
A systematic study [Collins et al., 2017] states:

"Our results point to the GRU as being the most learnable of gated RNNs for shallow architectures, followed by the UGRNN."

Collins, Sohl-Dickstein and Sussillo: Capacity and Trainability in Recurrent Neural Networks. ICLR, 2017

8.4
Autoregressive Models

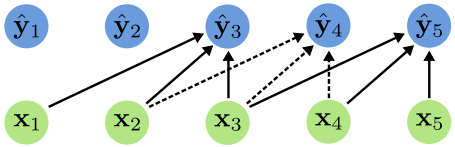
Autoregressive Models



$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \ldots, \mathbf{x}_{t-k})$$

- A k'th order **autoregressive model** is a **feedforward model** which predicts the next variable $\mathbf{x}_t$ in a time series based on the k previous variables $\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \ldots$
- As in RNNs, **parameters are shared** across time (same function $f(\cdot)$ at each t)
- Autoregressive models make a strong **conditional independence** assumption

Autoregressive Models

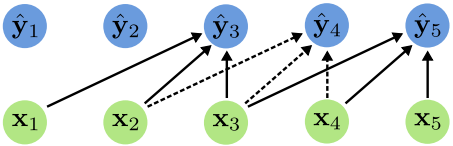


$$\hat{\mathbf{y}}_t = f(\mathbf{x}_t, \mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-k})$$

- This concept can be extended to situations with **different inputs and outputs**
- $\hat{\mathbf{y}}_t$ is **dependent** on $\{\mathbf{x}_i \mid t-k \leq i \leq t\}$ and **independent** of $\{\mathbf{x}_i \mid i < t-k\}$
- Remark: dashed arrows are only used to visually highlight different time steps

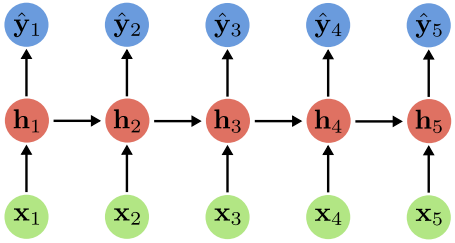
Autoregressive Models vs. RNNs

Autoregressive Network



$$\hat{\mathbf{y}}_t = f(\mathbf{x}_t, \mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-k})$$

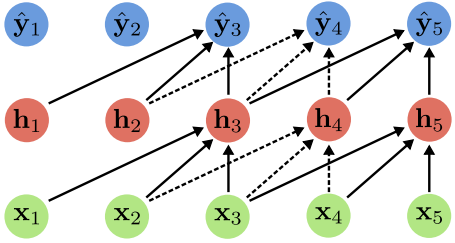
Recurrent Neural Network



$$\mathbf{h}_t = f_h(\mathbf{h}_{t-1}, \mathbf{x}_t)$$
$$\hat{\mathbf{y}}_t = f_y(\mathbf{h}_t)$$

- Recurrent models summarize past information through their **hidden state** $\mathbf{h}$
- In contrast to auto-regressive models, RNNs have **infinite memory**
- Autoregressive models are **easier to train** (no backprop through time)

Multi-Layer Autoregressive Models

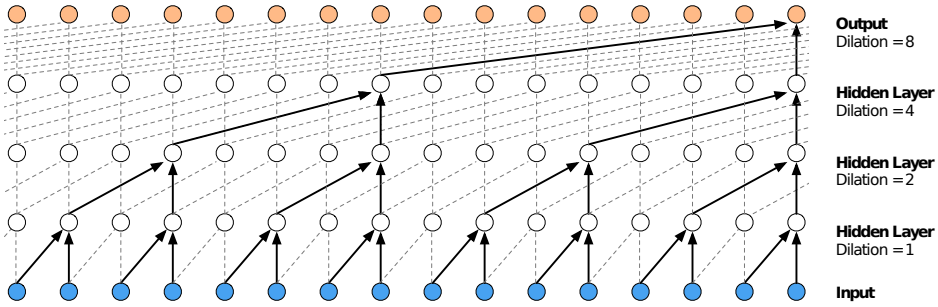


$$\mathbf{h}_t = f_1(\mathbf{x}_t, \mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-k})$$

$$\hat{\mathbf{y}}_t = f_2(\mathbf{h}_t, \mathbf{h}_{t-1}, \ldots, \mathbf{h}_{t-k})$$

- Autoregressive models can be extended to deep models with **multiple layers**
- They effectively perform multiple causal **temporal convolutions**
- They can be combined with residual connections and dilated convolutions

WaveNet



WaveNet:

- Generative model for **raw audio waveforms** which generates realistic speech
- Combines **dilated convolutions** with **residual and skip connections**
- Audio is hard: 16k samples per second, structure at **multiple time scales**

Oord et al.: WaveNet: A Generative Model for Raw Audio. ISCA Speech Synthesis Workshop, 2016.

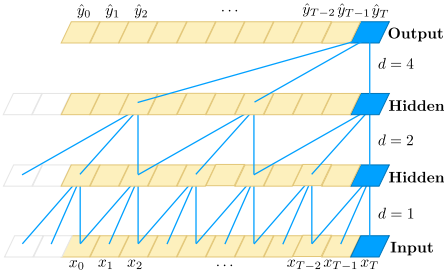
WaveNet

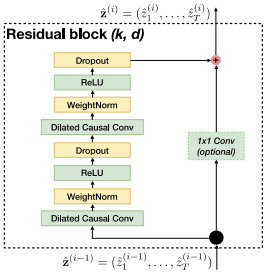
Speech samples	Subjective 5-scale MOS in naturalness	
	North American English	Mandarin Chinese
LSTM-RNN parametric	3.67 ± 0.098	3.79 ± 0.084
HMM-driven concatenative	3.86 ± 0.137	3.47 ± 0.108
WaveNet (L+F)	$\mathbf{4.21} \pm 0.081$	$\mathbf{4.08} \pm 0.085$
Natural (8-bit μ-law)	4.46 ± 0.067	4.25 ± 0.082
Natural (16-bit linear PCM)	4.55 ± 0.075	4.21 ± 0.071

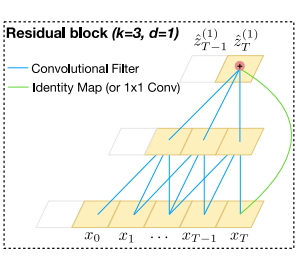
- WaveNet outperforms traditional LSTMs on speech synthesis by large margin

Oord et al.: WaveNet: A Generative Model for Raw Audio. ISCA Speech Synthesis Workshop, 2016.

Temporal Convolution Networks







Temporal Convolution Networks:

- Autoregressive model with **zero-padding** to handle sequences of arbitrary length
- **Deep multi-layer** network with **residual layers**, but otherwise simple architecture
- **Dilated convolutions** to increase the receptive field size (often called “context”)

Bai, Kolter and Koltun: An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. Arxiv, 2018.

Temporal Convolution Networks

Sequence Modeling Task	Model Size ($\approx$)	Models			
		LSTM	GRU	RNN	**TCN**
Seq. MNIST (accuracy^h)	70K	87.2	96.2	21.5	**99.0**
Permuted MNIST (accuracy)	70K	85.7	87.3	25.3	**97.2**
Adding problem T=600 (loss$^\ell$)	70K	0.164	**5.3e-5**	0.177	**5.8e-5**
Copy memory T=1000 (loss)	16K	0.0204	0.0197	0.0202	**3.5e-5**
Music JSB Chorales (loss)	300K	8.45	8.43	8.91	**8.10**
Music Nottingham (loss)	1M	3.29	3.46	4.05	**3.07**
Word-level PTB (perplexity$^\ell$)	13M	**78.93**	92.48	114.50	88.68
Word-level Wiki-103 (perplexity)	-	48.4	-	-	**45.19**
Word-level LAMBADA (perplexity)	-	4186	-	14725	**1279**
Char-level PTB (bpc$^\ell$)	3M	1.36	1.37	1.48	**1.31**
Char-level text8 (bpc)	5M	1.50	1.53	1.69	**1.45**

- TCNs outperform LSTMs, GRUs and RNNs on **several different sequence tasks**

Bai, Kolter and Koltun: An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. Arxiv, 2018.

Discussion

- The unlimited context offered by recurrent models is not strictly necessary for language modeling [Dauphin, 2016]
- Models that truncate at sequence lengths of 13 [Chelba, 2017] or 25 [Dauphin, 2016] are often competitive with models that have infinite memory
- The “infinite memory” advantage of RNNs is largely absent in practice, even when considering a variety of different models and tasks [Bai, 2018]
- Theoretical result: If the recurrent model is stable (meaning the gradients can not explode), then the model can be well-approximated by a feed-forward network for the purposes of both inference and training [Miller, 2018]
- `http://www.offconvex.org/2018/07/27/approximating-recurrent/`

Bai, Kolter and Koltun: An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. Arxiv, 2018.