RECURRENT NEURAL NETWORKS AND SEQUENCE MODELS

Machine Learning for Autonomous Robots

Dr. Matias Valdenegro
Assistant Professor at University of Groningen

January 16, 2023 – Bremen, Deutschland

University of Bremen

German Research Center for Artificial Intelligence

1. Introduction
2. Advanced RNNs
3. Visualization
4. Advanced Sequence Models
5. Attention Mechanisms
6. Regularization

Introduction

Sequence Modeling

- Many problems are naturally modeled as sequences instead of just fixed-size vectors.
- In these problems, there is structure in the sequence (the order) of elements, such as short or long term dependencies.
- For example, text is a sequence of characters, and the order of specific characters changes the meaning of the sentence.
- Another clear example is Time Series, where the time dimension defines a sequence.

Sequence Modeling

Some examples of tasks that require sequential modeling.

- Analyzing and processing video frames.
- Processing temporal data (accelerometers, cameras, etc).
- Textual information processing (translation, classification, etc).
- Speech synthesis and analysis. Activity recognition.

Sequence Modeling

- Sequences for ML models are typically input as tensors/arrays having shape (`samples`, *timesteps*, *features*).
- The *timesteps* dimension represents a dimension across the sequence.
- The *features* dimension corresponds to multi-dimensional features specific to the problem. In case the problem needs scalar features, then this dimension is set to one.

Recurrent Neural Networks

An RNN is a neural network that has two major differences over a feed forward network:

- An RNN has memory, through an internal state that is persisted between timesteps.
- The internal state (denoted by h) is meant to capture temporal or sequence information, which is also learned from data.
- An RNN is able to receive variable-sized inputs, and to produce variable-size outputs, usually with an encoder-decoder architecture.

In an RNN, h_t is the hidden state for timestep t, x_t is the input given at timestep t, and y_t is the output produced at each timestep.

$$h_t = \sigma_h(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$
$$y_t = \sigma_y(W_{hy}h_t + b_y)$$

These equations are vectorized if features are multi-dimensional. Note that instead of a single weight/bias matrix, there are three weight matrices (W_{hh}, W_{xh}, W_{hy}) and two bias vectors (b_h, b_y). σ denotes activation functions.

- An important feature of an RNN is that the weight/bias matrices are shared among timesteps.
- This means that these weights/biases are the same across timesteps, the operation that they represent is invariant to the sequence.
- This has the advantage of reducing the number of parameters needed to learn a task, and to force the RNN cell to learn timestep-invariant features. It is similar to how CNNs work with respect to translation invariance.

- The dimensionality of the hidden state h_t does not have to be the same as the input.
- As in feedforward NNs, an RNN could project the input into a higher dimensional space, which also contains the hidden state.
- This allows for sequence features in the hidden state that activate or correlate with features in the sequence, modeling long and short term dependencies.

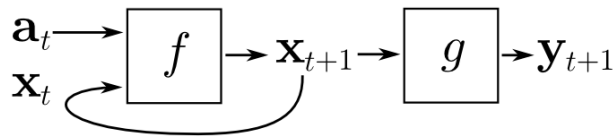
RNN Hyper-Parameters

- The most basic hyper-parameter is the number of units/neurons in a recurrent layer, which also defines the hidden space dimensionality.
- Activations σ_h is usually sigmoid or tanh, while activation σ_y can be configurable by the user. The selection of σ_h is done in a way to prevent the hidden state from growing without control.
- An RNN can be configured to return a sequence of outputs y_t, or only the output of the last timestep y_n.
- RNNs can also be stateful, meaning that the hidden state is kept across batches of inputs, and the state can be manually reset (to zero).

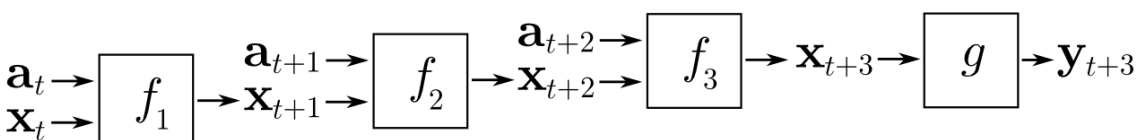
Training RNNs

- RNNs are trained using Back-propagation Through Time.
- The RNN is **unrolled** through timesteps (time), and the gradients are computed with respect to the trainable parameters using standard back-propagation (or in modern ML, using automatic differentiation).
- To do this one usually needs to know by how many timesteps to unroll the network.
- Unrolling transforms an RNN into a feed-forward network, for which we already know some training algorithms.
- But unrolling is not a requirement to train an RNN, using automatic differentiation gradients can be computed without performing unrolling, saving memory.

Back-propagation Through Time



⇩ unfold through time ⇩



Vanishing or Exploding Gradients

An RNN basically applies a repetitive multiplication by a weight matrix W. If we use an eigen decomposition $W = Q\Lambda Q^T$, then:

$$h_t = Q^T \Lambda^t Q$$

Any eigenvalues that are less than one will eventually converge to zero, and any eigenvalue bigger than one will explode into infinity. This happens when sequences are very long. This then translates into vanishing and exploding gradients when training RNNs.

Long and Short Term Dependencies

- Vanilla RNNs have big issues dealing with long-term dependencies.
- Typically the problem is that the gradient in direction of long-term dependencies is has exponentially smaller magnitude than the gradient in direction of short-term dependencies.
- It has been experimentally shown that the regions in parameter space that have good long-term gradients also have vanishing gradients, contributing to making learning difficult.

Tricks for Exploding Gradients

Element-Wise Clipping

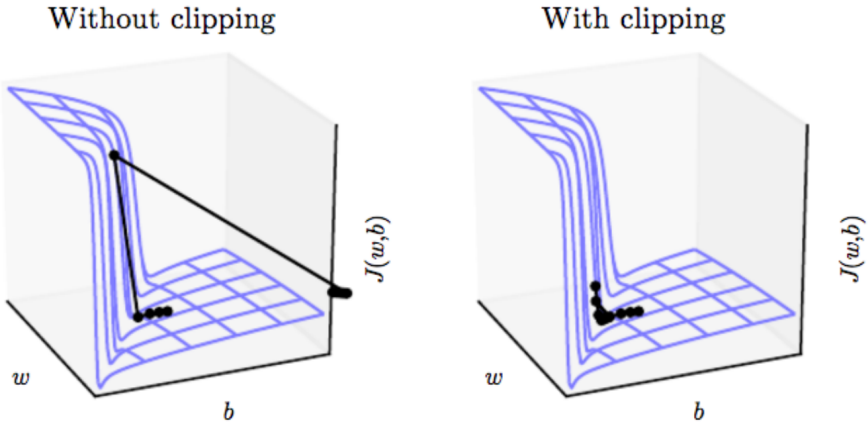
Given the gradient $\mathbf{g}$ of a batch, clip any component of the gradient that has absolute value bigger than v. That is for element where $g_i > v$, set $g_i = v$.

Norm Clipping

Clip the norm of the gradient. If $||\mathbf{g}|| > v$ set $g = \frac{\mathbf{g}v}{||\mathbf{g}||}$

The clip parameter v has to be decided. It can be obtained by monitoring gradients during training and trial and error.

Gradient Clipping



— Goodfellow et al., *Deep Learning*

Advanced RNNs

Long-Short Term Memory [Hochreiter & Schmidhuber. 1997]

- Developed by Sepp Hochreiter and Jürgen Schmidhuber in 1997.
- An LSTM is a more advanced kind of RNN cell, which has a much smaller chance of producing vanishing or exploding gradients.
- Additionally, it has a more advanced ability to model long-term dependencies inside the sequence.
- It works by using gated connections, and by splitting the state h_t into parts that are useful for output prediction, and parts to learn features from the sequence.

First, an LSTM computes three gates (forget, input, output):

$$g_f = \sigma(W_{hf} h_{t-1} + W_{xf} x_t + b_f)$$
$$g_i = \sigma(W_{hi} h_{t-1} + W_{xi} x_t + b_i)$$
$$g_o = \sigma(W_{ho} h_{t-1} + W_{xo} x_t + b_o)$$

These gates control the flow of information inside the LSTM. They use a sigmoid activation (the σ) to produce a value between 0 and 1.

Long-Short Term Memory

The hidden state is divided into two parts. h_t is state to predict output, and C_t is called cell state which models cross-timestep dependencies. First, a cell state proposal $\hat{C}$ is computed.

$$\hat{C} = \tanh(W_{hc} h_{t-1} + W_{xc} x_t + b_c)$$

And then the final cell state C_t is updated using the forget g_f and input gates g_i:

$$C_t = g_f \cdot C_{t-1} + g_i \cdot \hat{C}$$

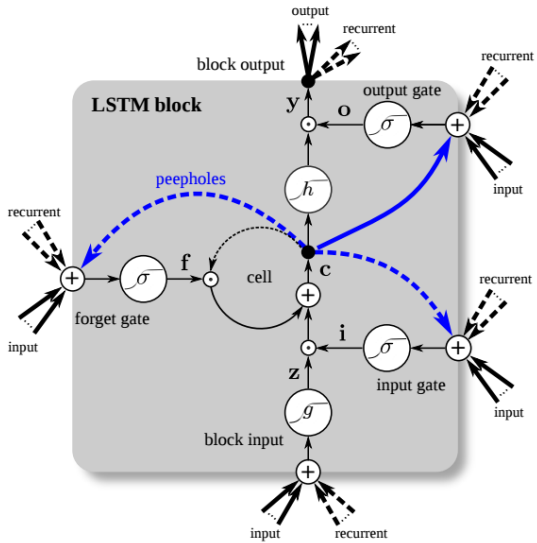
Here the forget gate determines how much of the previous cell state is used, and the input gate determines how the proposal is added to the final cell state. $\cdot$ performs component-wise products.

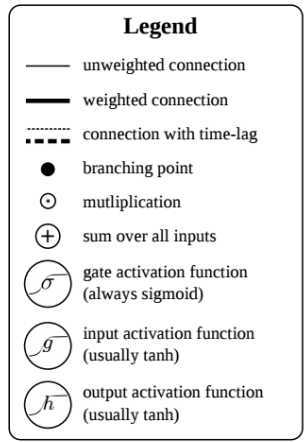
The output/hidden state is computed as:

$$h_t = g_o \cdot \sigma_y(C_t)$$

Here the output gate controls which parts of the cell state are used as output, and σ_y is the activation function for the output (user configurable).
The LSTM is a complicated cell, but this additional complexity greatly helps model long-term dependencies and has less issues with vanishing/exploding gradients.

Long-Short Term Memory





Gated Recurrent Unit [Cho et al. 2014]

Another state of the art unit is the GRU, which is simpler than an LSTM but still can perform similarly. It uses two gates (update and reset) with sigmoid activation:

$$g_r = \sigma(W_{hr} h_{t-1} + W_{xr} x_t + b_r)$$
$$g_u = \sigma(W_{hu} h_{t-1} + W_{xu} x_t + b_u)$$

GRU does not use cell states. The hidden state proposal is computed as:

$$\hat{h}_t = \tanh(W_{xh} x_t + W_{hh}\, g_r \cdot h_{t-1} + b_h)$$

And $\cdot$ represents component-wise multiplication.

Gated Recurrent Unit[Cho et al. 2014]

The final hidden state is:

$$h_t = (1 - g_u) \cdot h_{t-1} + g_u \cdot \hat{h}_t$$

Which corresponds to a linear interpolation between the previous hidden state and the hidden state proposal, with weight determined by the update gate g_u.

The output y_t of a GRU is computed the same way as a standard RNN layer.

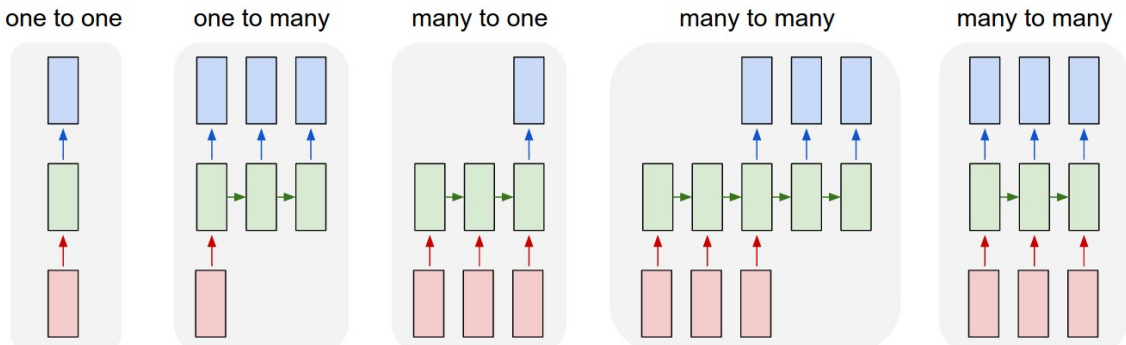
Convolutional RNNs

- A natural extension of any recurrent layer is to replace the dot product with spatial convolution:

$$h_t = \sigma_h(W_{hh} \star h_{t-1} + W_{xh} \star x_t + b_h)$$
$$y_t = \sigma_y(W_{hy} \star h_t + b_y)$$

- Where the weights W_{hh}, W_{xh}, W_{hy} are replaced with learnable convolutional filters.
- The input x_t, state h_t, and outputs y_t are now convolutional feature maps. Shapes are usually 5D as `(samples, timesteps, width, height, channels)`.
- Similar extensions can be done for LSTM and GRU. Usually very memory intensive.

Basic RNN Architectures



Basic RNN Architectures

- **One-to-One**. This is a feed-forward neural network, no timesteps or sequences are used.
- **One-to-Many**. One input timestep, multiple output timesteps. This is usually implemented by feeding back the output of one timestep to the next time step as input.
- **Many-to-one**. Multiple input timesteps, a single output timesteps. Implemented as a normal RNN, predicting an output for each timestep, but discarding these outputs except for the last timestep.

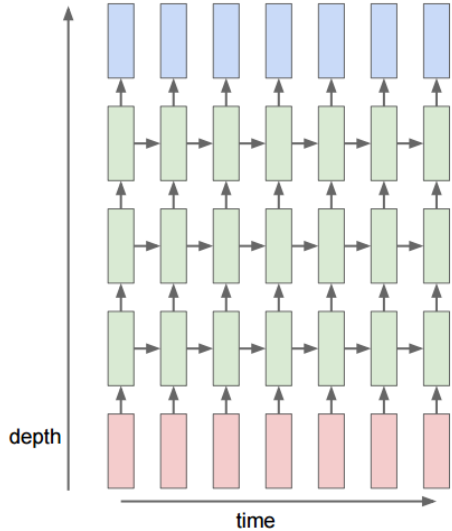
Basic RNN Architectures

- **Many-to-Many**. Multiple inputs timesteps, multiple output timesteps. There are two basic cases, when the number of input and output timesteps is the same, or it is variable. The model can learn to decide when to stop by outputting a specially market output (a token).
- When using variable output timesteps, usually first the network consumes all input timesteps, and then can produce output timesteps. In this case the input for each additional timestep is the output at the previous timestep.
- Encoder-Decoder architectures are typically used for variable-sized inputs and outputs.

Stacking RNNs

- RNNs can also be stacked, just as Convolutional/Fully Connected layers can be.
- This corresponds to the *depth* dimension of the network, as deeper networks usually perform better and learns a feature hierarchy.
- At each level in the hierarchy, the output is a sequence of features that is input at the next set of layers in the hierarchy.
- You can also combine this with any basic RNN architecture.

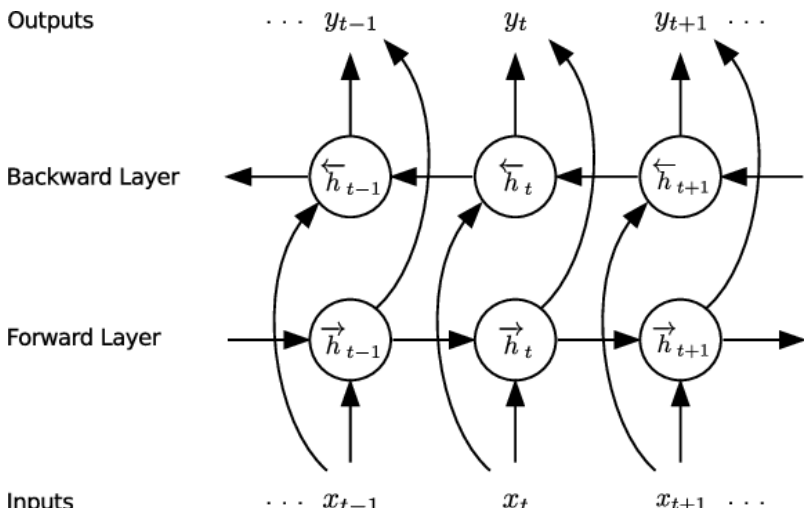
Stacking RNNs



Bi-Directional RNNs

- Normal RNNs are *causal*, that is, they only consider information from the past to produce a prediction.
- There are applications which require to look at the whole sequence information before making a prediction, such as machine translation. For this Bi-Directional RNNs can be used.
- The basic idea is to have a set of RNN cells looking in the forward timesteps dimension, and another looking into the backwards timestep dimension, starting from the farthest future.
- Then both forward and backwards sequences are combined and used to make a prediction into a final RNN layer.

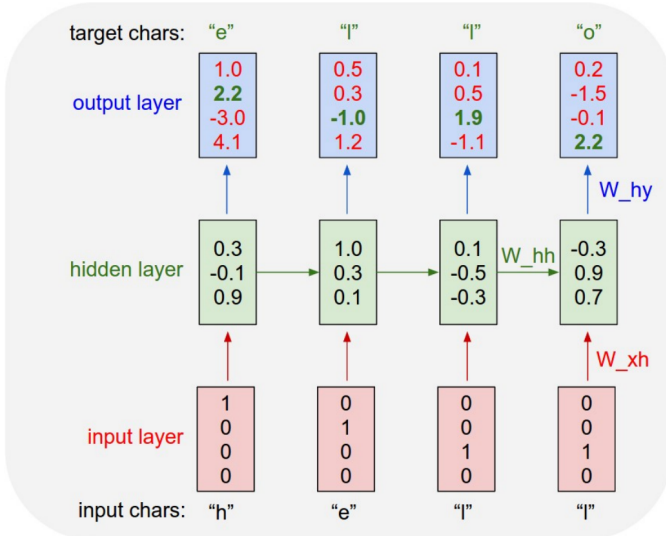
Bi-Directional RNNs



Text Modeling

- RNNs are very appropriate to model different kinds of text. There are two basic approaches.
- **Character-Level**. Text is modeled as sequences of characters, where each character is one-hot encoded from a pre-defined alphabet.
- **Word-Level**. Text is modeled as sequences of words, and each word is one-hot encoded from a pre-defined vocabulary.
- In general character-level is preferred as the size of the alphabet is smaller than a vocabulary.

Text Modeling



Text Generation with RNNs

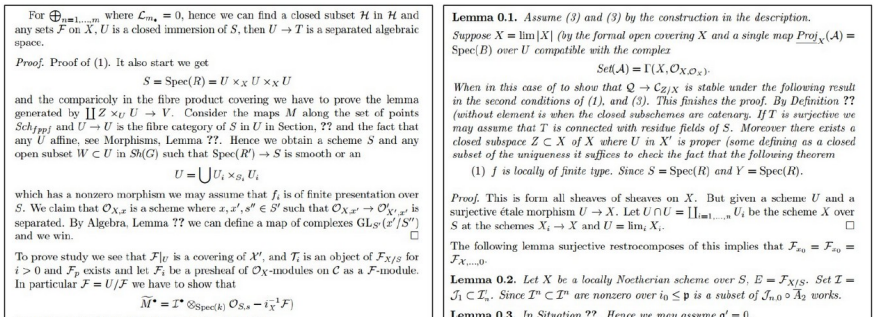
"The surprised in investors weren't going to raise money. I'm not the company with the time there are all interesting quickly, don't have to get off the same programmers. There's a super-angel round fundraising, why do you can do. If you have a different physical investment are become in people who reduced in a startup with the way to argument the acquirer could see them just that you're also the founders will part of users' affords that and an alternation to the idea. [2] Don't work at first member to see the way kids will seem in advance of a bad successful startup. And if you have to act the big company too."

Generated from an RNN trained with Paul Graham's works.

Text Generation with RNNs

```
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.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

Text Generation with RNNs



For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,\overline{x}}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{\overline{X}',\overline{x}'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace S by $X_{spaces,\acute{e}tale}$ which gives an open subspace of X and T equal to S_{Zar}. see Descent., Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ *(by the formal open covering* X *and a single map* $\underrightarrow{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ *over* U *compatible with the complex*

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If* T *is surjective we may assume that* T *is connected with residue fields of* S*. Moreover there exists a closed subspace* $Z \subset X$ *of* X *where* U *in* X' *is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) f *is locally of finite type. Since* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restroecomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{X,\ldots,0}$.

Lemma 0.2. *Let* X *be a locally Noetherian scheme over* S*.* $E = \mathcal{F}_{X/S}$*. Set* $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$*. Since* $\mathcal{I}^n \subset \mathcal{I}^n$ *are nonzero over* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \overline{A}_2$ *works.*

Lemma 0.3. *In Situation ??. Hence we may assume* $\mathfrak{q}' = 0$.

Proof. We will use the property we see that $\mathfrak{p}$ is the next functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

Sampled (fake) algebraic geometry. Here's the actual pdf.

Text Generation with RNNs

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

Visualization

Visualizing RNNs [Karpathy et al. 2015]

- They trained RNN/LSTM/GRUs to predict a character in text from the previous one.
- The models were trained on two text datasets: The War and Peace novel, and the Linux Kernel source code.
- Then the activations of the hidden state are visualized, overlaid on top of the text that is being input.

Visualizing RNNs [Karpathy et al. 2015]

Cell sensitive to position in line:

```
The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and did not,
surrender.
```

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."
```

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;
}
```

Visualizing RNNs [Karpathy et al. 2015]

Cell that turns on inside comments and quotes:

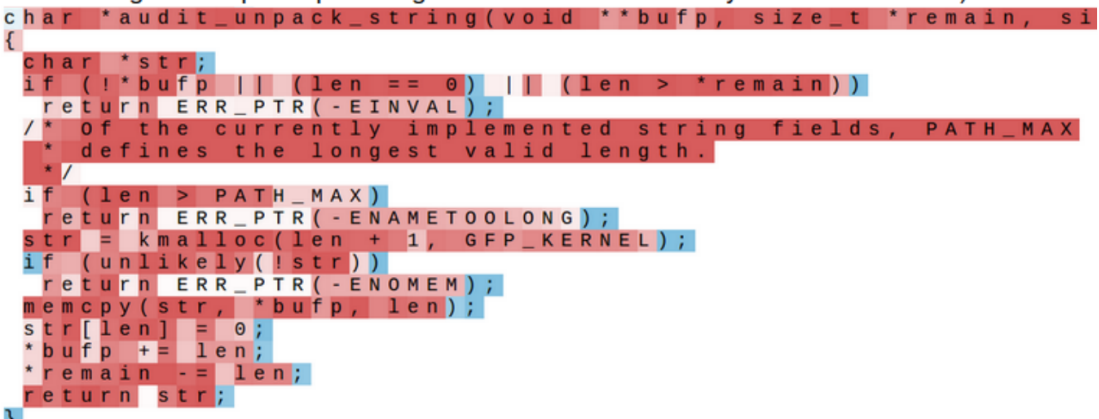
```
/* Duplicate LSM field information.  The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
                   struct audit_field *sf)
{
 int ret = 0;
 char *lsm_str;
 /* our own copy of lsm_str */
 lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
 if (unlikely(!lsm_str))
  return -ENOMEM;
 df->lsm_str = lsm_str;
 /* our own (refreshed) copy of lsm_rule */
 ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                (void **)&df->lsm_rule);
 /* Keep currently invalid fields around in case they
  * become valid after a policy reload. */
 if (ret == -EINVAL) {
  pr_warn("audit rule for LSM \'%s\' is invalid\n",
   df->lsm_str);
  ret = 0;
 }
 return ret;
}
```

Cell that is sensitive to the depth of an expression:

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
 int i;
 if (classes[class]) {
  for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
   if (mask[i] & classes[class][i])
    return 0;
 }
 return 1;
}
```

Visualizing RNNs [Karpathy et al. 2015]

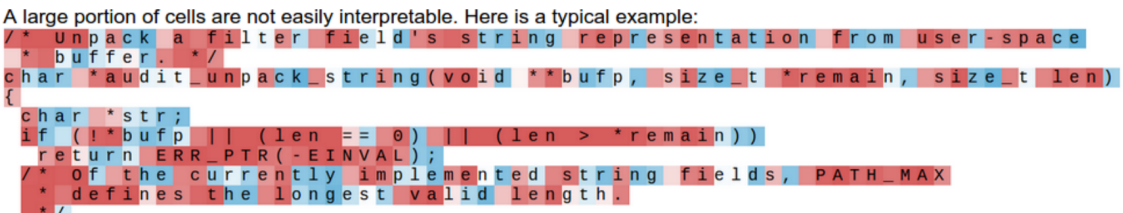
Cell that might be helpful in predicting a new line. Note that it only turns on for some ")":



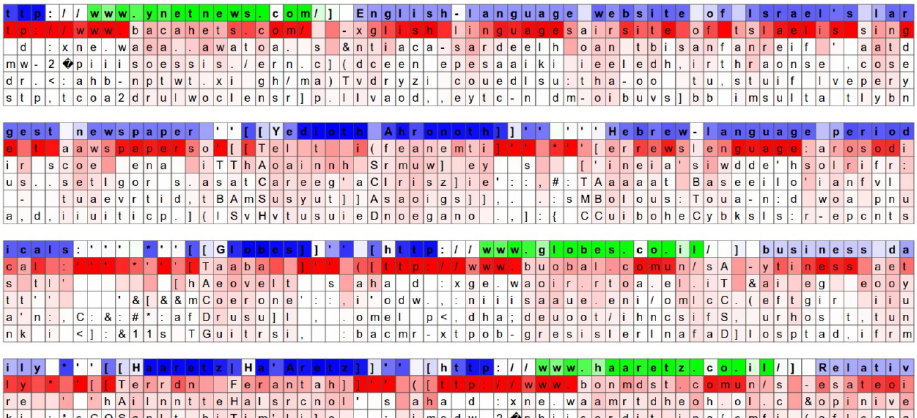
```
char *audit_unpack_string(void **bufp, size_t *remain, si
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /* Of the currently implemented string fields, PATH_MAX
   * defines the longest valid length.
   */
  if (len > PATH_MAX)
    return ERR_PTR(-ENAMETOOLONG);
  str = kmalloc(len + 1, GFP_KERNEL);
  if (unlikely(!str))
    return ERR_PTR(-ENOMEM);
  memcpy(str, *bufp, len);
  str[len] = 0;
  *bufp += len;
  *remain -= len;
  return str;
}
```

Visualizing RNNs [Karpathy et al. 2015]

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

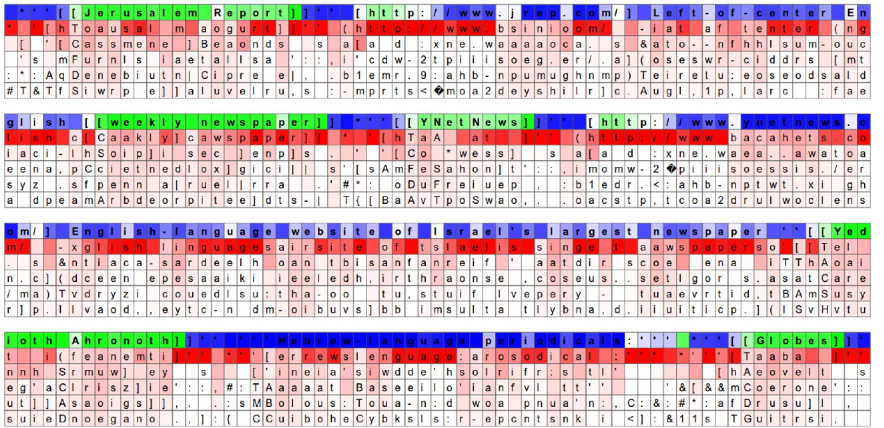


Visualizing RNNs [Karpathy et al. 2015]



The neuron highlighted in this image seems to get very excited about URLs and turns off outside of the URLs. The LSTM is likely using this neuron to remember if it is inside a URL or not.

Visualizing RNNs [Karpathy et al. 2015]



The highlighted neuron here gets very excited when the RNN is inside the [[ ]] markdown environment and turns off outside of it. Interestingly, the neuron can't turn on right after it sees the character "[", it must wait for the second "[" and then activate. This task of counting whether the model has seen one or two "[" is likely done with a different neuron.

Visualizing RNNs [Karpathy et al. 2015]

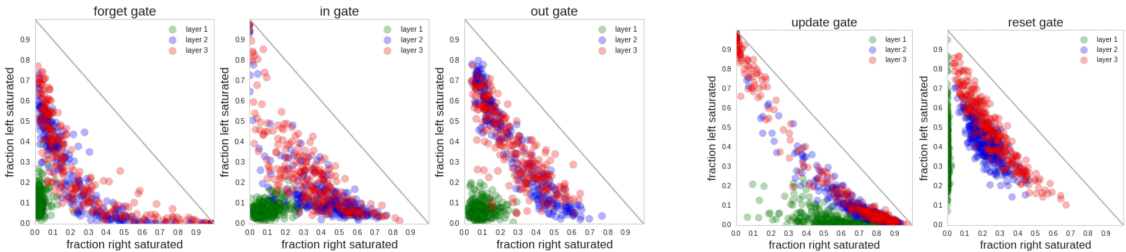


Figure 3: **Left three:** Saturation plots for an LSTM. Each circle is a gate in the LSTM and its position is determined by the fraction of time it is left or right-saturated. These fractions must add to at most one (indicated by the diagonal line). **Right two:** Saturation plot for a 3-layer GRU model.

Advanced Sequence Models

One-Dimensional Convolutions

- RNNs are not the only kind of models that are able to process sequences.
- Since 1D convolutions will perform sliding window across the width dimension, one can consider this as processing a sequence with shape `(samples, width, channels)`, which has the same structure as for a sequence.
- Using convolution would learn features that are linear combinations of neighboring elements in the input, meaning it only makes sense if there is such structure in the data.

Causal One-Dimensional Convolutions [Oord et al. 2016]

- One important detail when processing sequences is to consider causality, which is simply that for timestep t, only information from timesteps 0 to t are considered to make a prediction.
- Meaning only past information is used for a prediction.
- This is important because convolutions at each layer work in both directions, and do not have to limit themselves to past information. They can leak future information into predictions.

Causal One-Dimensional Convolutions [Oord et al. 2016]

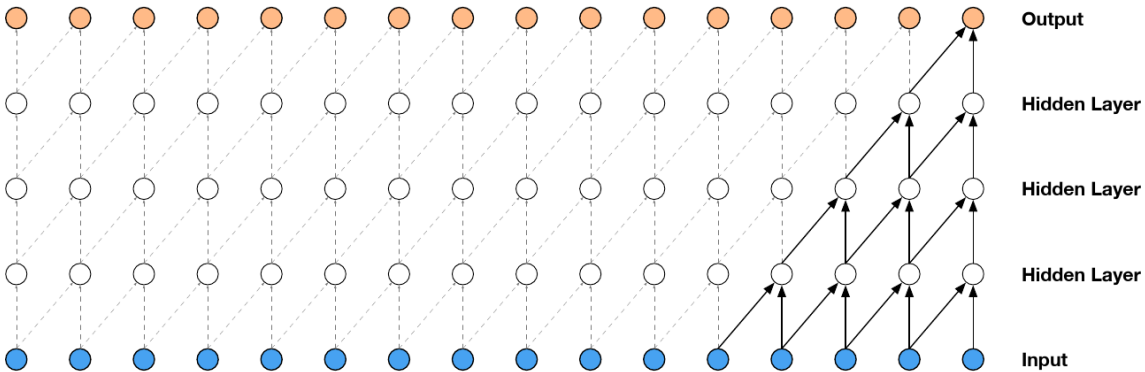


Figure 2: Visualization of a stack of causal convolutional layers.

Causal Dilated Convolutions [Oord et al. 2016]

- As we saw before, you can also add dilation to convolutions, in order to move the sliding window for more than one element.
- This is useful as a way to increase the effective **receptive field** of the network.
- For sequential models, the receptive field means how much of the input sequence is needed to produce one prediction.

Dilated Causal One-Dimensional Convolutions [Oord et al. 2016]

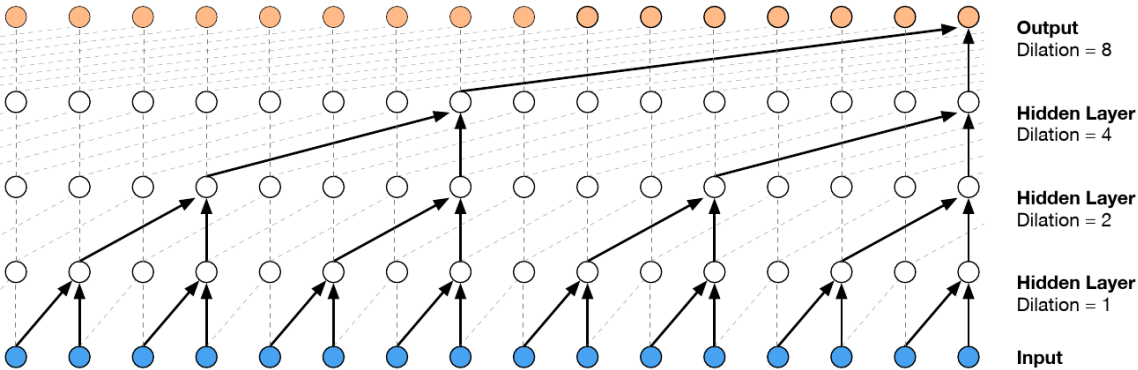
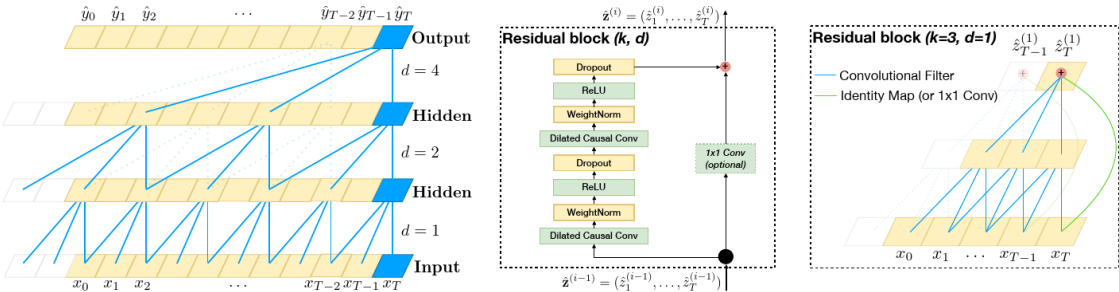


Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

Temporal Convolutional Networks [Bai et al. 2018]

- A generalization of the previous concept, its a family of neural network architectures.
- It combined a fully convolutional network (FCN) with causal convolutions, and residual connections.
- It can provide performance that is very similar to a recurrent network like LSTM or GRU.

Temporal Convolutional Networks [Bai et al. 2018]



(a) (b) (c)

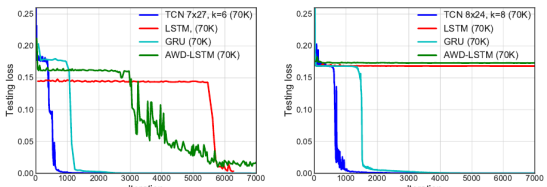
Figure 1. Architectural elements in a TCN. (a) A dilated causal convolution with dilation factors $d = 1, 2, 4$ and filter size $k = 3$. The receptive field is able to cover all values from the input sequence. (b) TCN residual block. An 1x1 convolution is added when residual input and output have different dimensions. (c) An example of residual connection in a TCN. The blue lines are filters in the residual function, and the green lines are identity mappings.

Temporal Convolutional Networks [Bai et al. 2018]

Table 1. Evaluation of TCNs and recurrent architectures on synthetic stress tests, polyphonic music modeling, character-level language modeling, and word-level language modeling. The generic TCN architecture outperforms canonical recurrent networks across a comprehensive suite of tasks and datasets. Current state-of-the-art results are listed in the supplement. ^{h} means that higher is better. $^{\ell}$ means that lower is better.

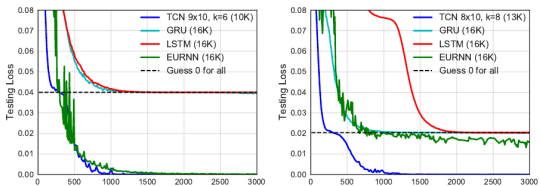
Sequence Modeling Task	Model Size ($\approx$)	Models			
		LSTM	GRU	RNN	**TCN**
Seq. MNIST (accuracy^{h})	70K	87.2	96.2	21.5	**99.0**
Permuted MNIST (accuracy)	70K	85.7	87.3	25.3	**97.2**
Adding problem T=600 (loss$^{\ell}$)	70K	0.164	**5.3e-5**	0.177	**5.8e-5**
Copy memory T=1000 (loss)	16K	0.0204	0.0197	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**

Temporal Convolutional Networks [Bai et al. 2018]



(a) $T = 200$ (b) $T = 600$

Figure 2. Results on the adding problem for different sequence lengths T. TCNs outperform recurrent architectures.



(a) $T = 500$ (b) $T = 1000$

Figure 4. Result on the copy memory task for different sequence lengths T. TCNs outperform recurrent architectures.

Temporal Convolutional Networks [Bai et al. 2018]

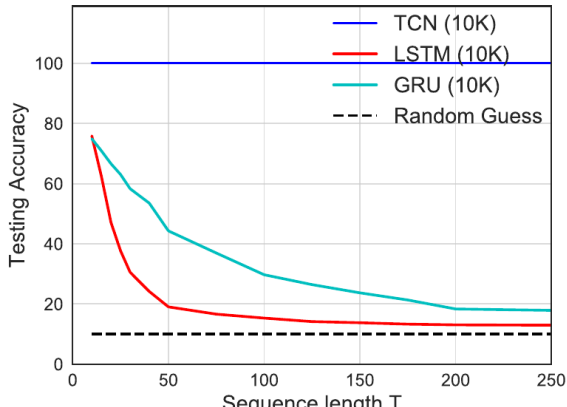


Figure 5. Accuracy on the copy memory task for sequences of different lengths T. While TCN exhibits 100% accuracy for all sequence lengths, the LSTM and GRU degenerate to random guessing as T grows.

Sequence-to-Sequence Models

- A set of models that can easily model problems with an input sequence, and an output predicted sequence.
- Generally this is done with RNNs.
- The most basic model is called an Encoder-Decoder, because all the information about the input sequence can be encoded into a hidden state vector, which is used by the decoder to produce the output.
- Only the hidden state of the last sequence element is usually used.

Sequence-to-Sequence Models

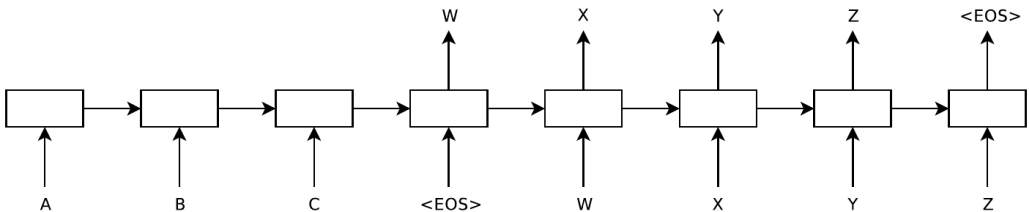


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

Issues with Seq2Seq Models

- In general they work, but have issues with long-term dependencies.
- For example, in Neural Machine Translation, all the information about the input word sequence has to be stored in a single hidden state vector.
- Even if the hidden state vector has a high dimensionality, in general it cannot hold all the information.
- This has been found experimentally.

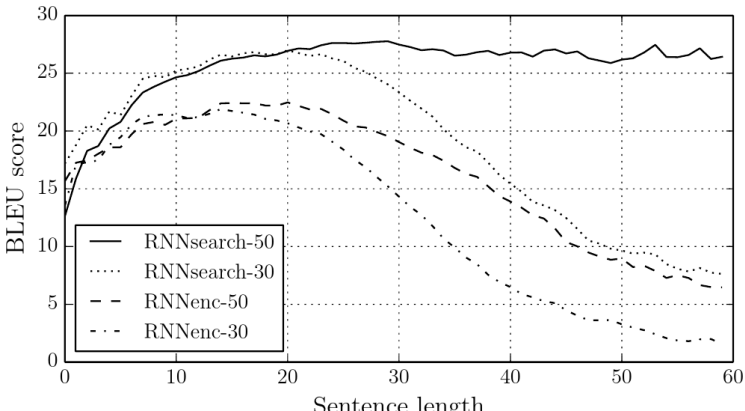


Figure 2: The BLEU scores of the generated translations on the test set with respect to the lengths of the sentences. The results are on the full test set which includes sentences having unknown words to the models.

Attention Mechanisms

The Concept of Attention

- Attention is currently becoming an important basic ML concept.
- In simple words, it is an addition to a model so the model can decide by itself where to look in the input, in a way that reduces processing and the most important input or feature is seen by the model.
- Usually an Attention model has a part that predicts "where to look", and a way to combine this with the input or features.
- We will first visit [Bahdanau, Cho, and Bengio 2014] with Soft-Attention for Seq2Seq models.

Soft-Attention [Bahdanau, Cho, and Bengio 2014]

- For a simple seq2seq model for NMT, all the hidden state vectors h_t across timesteps are used, and linearly combined:

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j$$

Where the α_{ij} are the weights computed as:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})}$$

The values e_{ij} are produced by an attention model, which is the one that decides "where to look".

Soft-Attention [Bahdanau, Cho, and Bengio 2014]

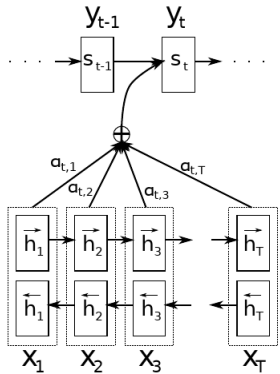


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Soft-Attention for Neural Machine Translation

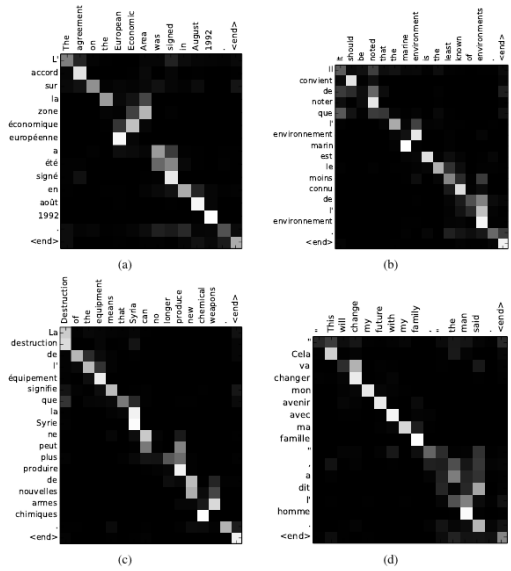


Figure 3: Four sample alignments found by RNNsearch-50. The x-axis and y-axis of each plot

Multi-Head Attention [Vaswani et al. 2017]

The literature contains (too) many models for attention:

- There are different models that decide where to perform attention.
- Combination of hidden state and attention vectors does not have to be linear.
- Attention can be multi-scale or even multiple attentions can be done at the same time. This has led to the developed of Multi-Head Attention and the Transformer Architecture.

Scaled Dot-Product Attention [Vaswani et al. 2017]

- It is a popular Attention approach, given by:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

- Where Q is a query, K is a key, and V is a value, all vectors of the same dimension.
- This corresponds to a generalization of the Soft-Attention previously shown.
- **Encoder-Decoder Attention**. The query Q comes from the previous decoder, while the key K and value V come from the encoder output.
- **Self-Attention**. Q, K, and V all come from the same module, but from a previous layer.

Multi-Head Attention [Vaswani et al. 2017]

- Corresponds to using multiple Attention "instances" at the same time, each of them learning to focus at different scales or parts of the input.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, ..., \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- Matrices W correspond to learnable projections for the attention parameters.
- The authors use 8 Attention heads, which are concatenated to form the output.
- This kind of Attention is used to build the Transformer architecture.

Multi-Head Attention [Vaswani et al. 2017]

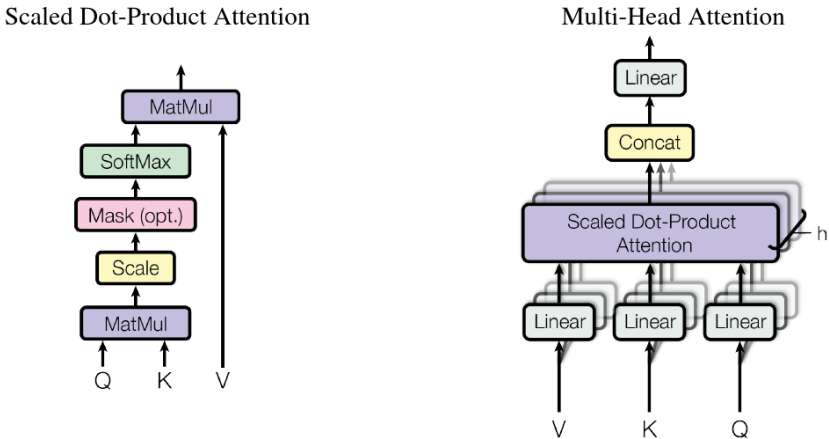


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Transformer Architecture [Vaswani et al. 2017]

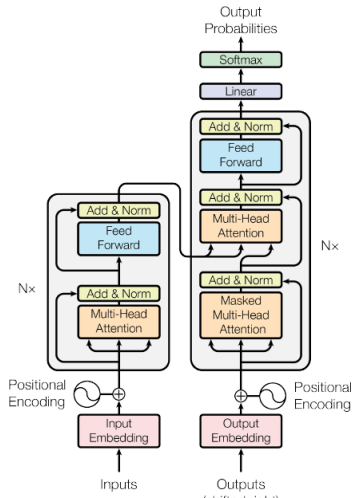


Figure 1: The Transformer - model architecture.

Visualization of Attention [Vaswani et al. 2017]

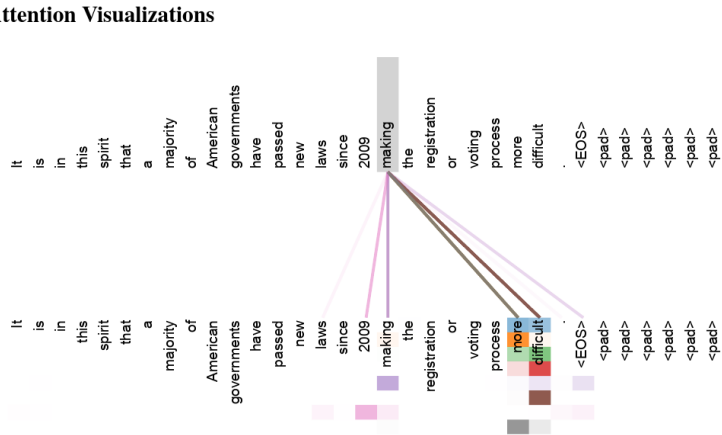


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

Visualization of Attention [Vaswani et al. 2017]

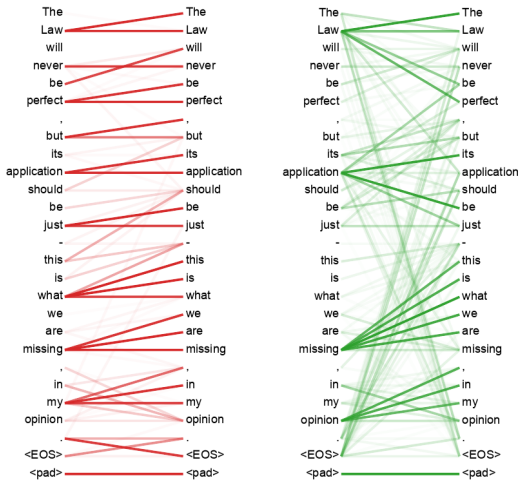


Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.

Other Kinds of Attention

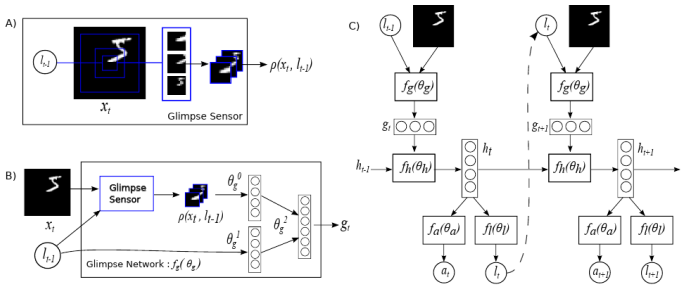
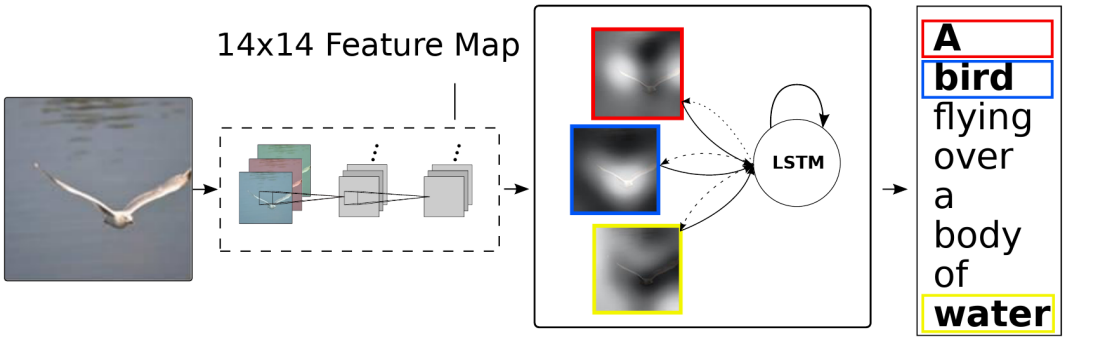


Figure 1: **A) Glimpse Sensor:** Given the coordinates of the glimpse and an input image, the sensor extracts a *retina-like* representation $\rho(x_t, l_{t-1})$ centered at l_{t-1} that contains multiple resolution patches. **B) Glimpse Network:** Given the location (l_{t-1}) and input image (x_t), uses the glimpse sensor to extract retina representation $\rho(x_t, l_{t-1})$. The retina representation and glimpse location is then mapped into a hidden space using independent linear layers parameterized by θ_g^0 and θ_g^1 respectively using rectified units followed by another linear layer θ_g^2 to combine the information from both components. The glimpse network $f_g(.; \{\theta_g^0, \theta_g^1, \theta_g^2\})$ defines a trainable bandwidth limited sensor for the attention network producing the glimpse representation g_t. **C) Model Architecture:** Overall, the model is an RNN. The core network of the model $f_h(.; \theta_h)$ takes the glimpse representation g_t as input and combining with the internal representation at previous time step h_{t-1}, produces the new internal state of the model h_t. The location network $f_l(.; \theta_l)$ and the action network $f_a(.; \theta_a)$ use the internal state h_t of the model to produce the next location to attend to l_t and the action/classification a_t respectively. This basic RNN iteration is repeated for a variable number of steps.

Other Kinds of Attention



Other Kinds of Attention

Figure 2. Attention over time. As the model generates each word, its attention changes to reflect the relevant parts of the image. "soft" (top row) vs "hard" (bottom row) attention. (Note that both models generated the same captions in this example.)

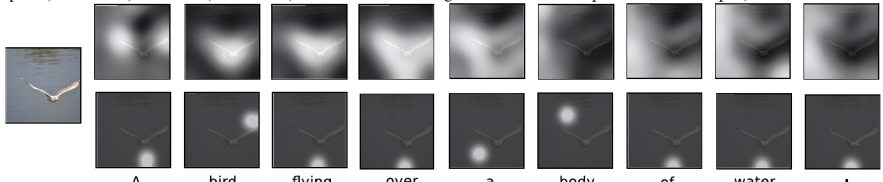
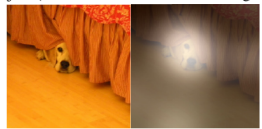


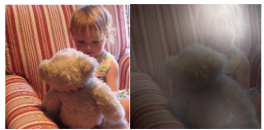
Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)

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.

Other Kinds of Attention

Figure 5. Examples of mistakes where we can use attention to gain intuition into what the model saw.

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.

Regularization

Regularization

- L1/L2 regularization can be directly applied, since through weight sharing, there are no major differences.
- But Dropout and Batch Normalization don't work correctly if applied directly to a RNN. This is due to weight sharing, and due the dynamics across timesteps.
- Changes are needed in the regularization techniques to make them compatible with the recurrent connections in an RNN.

Recurrent Dropout

There are multiple ways to use Dropout in RNNs.

- Use Dropout only on the non-recurrent parts of the network, like inputs and outputs of an RNN/LSTM/GRU.
- Use Dropout in the recurrent parts of the network, but use the same Dropout mask for all time-steps.
- This last modification allows the information to flow without problems across timesteps.

Recurrent Dropout

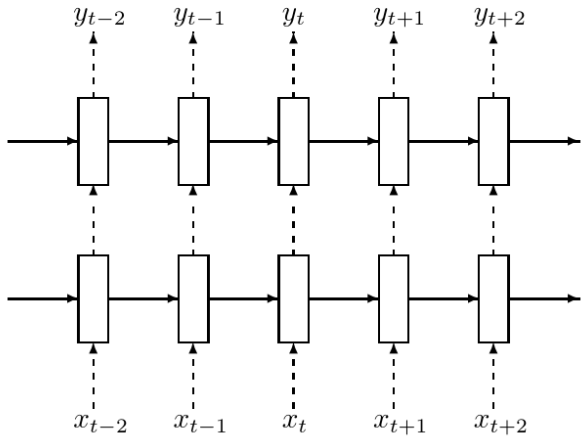
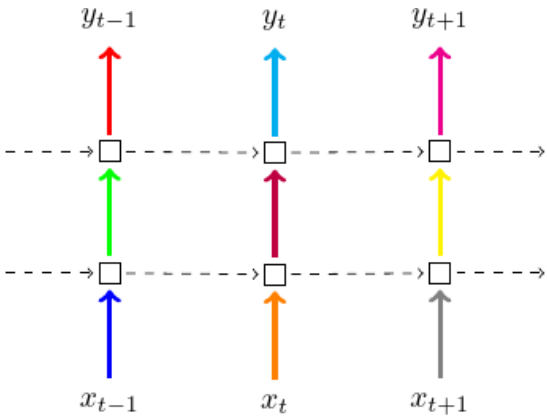
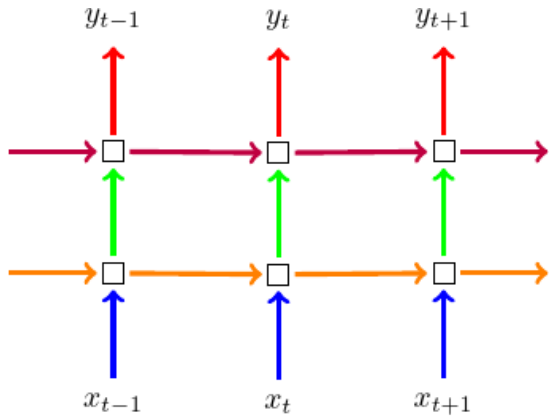


Figure 2: Regularized multilayer RNN. The dashed arrows indicate connections where dropout is applied, and the solid lines indicate connections where dropout is not applied.

Recurrent Dropout



(a) Naive dropout RNN



(b) Variational RNN

Batch Normalization

- Batch Normalization has issues as well due to the recurrent connections.
- The batch statistics are not the same at each timestep, meaning that in order to use Batch Normalization, each timestep has to be normalized differently.
- This limits models with variable-sized inputs and outputs, since pre-computation of batch statistics per timestep is no longer possible.
- Some variations of BatchNorm don't have this issue, such as Layer Normalization.

Layer Normalization on LSTMs

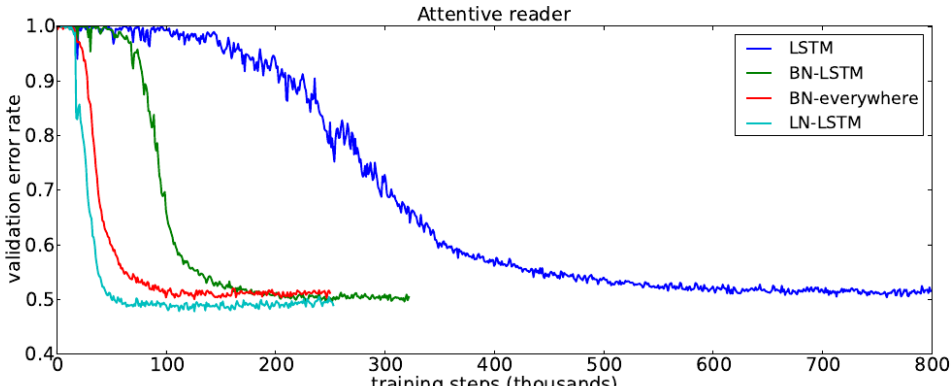


Figure 2: Validation curves for the attentive reader model. BN results are taken from [Cooijmans et al., 2016].

Thank You!
Please feel free to ask questions in the forums.

Bibliography I

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio.
Neural machine translation by jointly learning to align and translate.
arXiv preprint arXiv:1409.0473, 2014.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun.
An empirical evaluation of generic convolutional and recurrent networks for sequence modeling.
arXiv preprint arXiv:1803.01271, 2018.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio.
Learning phrase representations using rnn encoder–decoder for statistical machine translation.
In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

Bibliography II


Sepp Hochreiter and Jürgen Schmidhuber.
Long short-term memory.
Neural computation, 9(8):1735–1780, 1997.


Andrej Karpathy, Justin Johnson, and Li Fei-Fei.
Visualizing and understanding recurrent networks.
arXiv preprint arXiv:1506.02078, 2015.


Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu.
Wavenet: A generative model for raw audio.
arXiv preprint arXiv:1609.03499, 2016.

Bibliography III


Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin.
Attention is all you need.
In *Advances in neural information processing systems*, pages 5998–6008, 2017.