Recurrent Networks
Designing, Visualizing and Understanding Deep Neural Networks

CS W182/282A

Instructor: Sergey Levine
UC Berkeley
What if we have variable-size inputs?

Before:

\[ x_i \rightarrow \text{”cat”: 0.64} \]

Now:

\[ x_1 = (x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}) \]
\[ x_2 = (x_{2,1}, x_{2,2}, x_{2,3}) \]
\[ x_3 = (x_{3,1}, x_{3,2}, x_{3,3}, x_{3,4}, x_{3,5}) \]

Examples:

- classifying sentiment for a phrase (sequence of words)
- recognizing phoneme from sound (sequence of sounds)
- classifying the activity in a video (sequence of images)
What if we have variable-size inputs?

\[ x_1 = (x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}) \]
\[ x_2 = (x_{2,1}, x_{2,2}, x_{2,3}) \]
\[ x_3 = (x_{3,1}, x_{3,2}, x_{3,3}, x_{3,4}, x_{3,5}) \]

Simple idea: zero-pad up to length of longest sequence

+ very simple, and can work in a pinch

- doesn’t scale very well for very long sequences
One input per layer?

\[ x_1 = (x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}) \]
\[ x_2 = (x_{2,1}, x_{2,2}, x_{2,3}) \]
\[ x_3 = (x_{3,1}, x_{3,2}, x_{3,3}, x_{3,4}, x_{3,5}) \]

Each layer:

\[ \bar{a}^{\ell-1} = \begin{bmatrix} a_1^{\ell-1} \\ x_{i,t} \end{bmatrix} \]
\[ z^{\ell} = W^{\ell} \bar{a}^{\ell-1} + b^{\ell} \]
\[ a^{\ell} = \sigma(z^{\ell}) \]

**Note:** this doesn’t actually work very well in practice, we’ll discuss this more later

**Obvious question:** what happens to the missing layers?
Variable layer count?

This is more efficient than always 0-padding the sequence up to max length.

Each layer is much smaller than the giant first layer we would need if we feed in the whole sequence at the first layer.

The shorter the sequence, the fewer layers we have to evaluate.

But the total number of weight matrices increases with max sequence length!

each layer:

\[
\tilde{a}^{\ell-1} = \begin{bmatrix}
  a^{\ell-1} \\
  x_{i,t}
\end{bmatrix}
\]

\[
z^{\ell} = W^{\ell} \tilde{a}^{\ell-1} + b^{\ell}
\]

\[
a^{\ell} = \sigma(z^{\ell})
\]
Can we share weight matrices?

what if $W^\ell$ is the same for all these layers?

i.e., $W^{\ell_i} = W^{\ell_j}$ for all $i, j$

$b^{\ell_i} = b^{\ell_j}$ for all $i, j$

we can have as many “layers” as we want!

this is called a **recurrent** neural network (RNN)

could also call this a “variable-depth” network perhaps?

\[
\begin{align*}
a^{\ell-1} & = \begin{bmatrix} a^{\ell-1} \\ x_{i,t} \end{bmatrix} \\
z^\ell & = W^\ell a^{\ell-1} + b^\ell \\
a^\ell & = \sigma(z^\ell)
\end{align*}
\]
Aside: RNNs and time

What we just learned:

\[ a^\ell = 0 \quad \uparrow \quad x_{1,1} \quad \uparrow \quad x_{1,2} \quad \uparrow \quad x_{1,3} \quad \uparrow \quad x_{1,4} \]

“a recurrent neural network extends a standard neural network along the time dimension”

(or some other assertion of this sort)

This is technically true, but somewhat unhelpful for actually understanding how RNNs work, and makes them seem more mystical than they are

RNNs are just neural networks that share weights across multiple layers, take an input at each layer, and have a variable number of layers

What you often see in textbooks/classes:

\[ x_t \quad \rightarrow \]

this funny thing represents the fact that this layer also gets its own “previous” value as input
How do we train this?

Backpropagation:

forward pass: calculate each $a^{(i)}$ and $z^{(i)}$
backward pass:
initialize $\delta = \frac{d\mathcal{L}}{dz^{(n)}}$
for each $f$ with input $x_f$ & params $\theta_f$ from end to start:

\[
\frac{d\mathcal{L}}{d\theta_f} = \frac{df}{d\theta_f} \delta \quad \text{taken literally, gradient at } \ell - 1 \text{ will “overwrite” gradient at } \ell \\
\delta \leftarrow \frac{df}{dx_f} \delta + \frac{d\mathcal{L}}{d\theta_f} \frac{df}{d\theta_f} \delta \quad \text{“accumulate” the gradient during the backward pass}
\]

$W, b$ are shared at all these layers
shared = the same

To convince yourself that this is true:

\[
f(x) = g(x, h(x)) \quad \text{how does this resemble role of } W \text{ in the RNN?}
\]

\[
\frac{d}{dx} f(x) = \frac{dg}{dx} + \frac{dh}{dx} \frac{dg}{dh}
\]

derivative through first argument (via chain rule)
What if we have variable-size outputs?

Examples:

- generating a text caption for an image
- predicting a sequence of future video frames
- generating an audio sequence

**Before:** an input at every layer

**Now:** an output at every layer

\[ \hat{y}_i,1 \uparrow \hat{y}_i,2 \uparrow \hat{y}_i,3 \uparrow \hat{y}_i,4 \]

\[ x_i \rightarrow \rightarrow \rightarrow \rightarrow \]
An output at every layer

each of these have their own loss!

at each step:

\[ z^\ell = W^\ell a^{\ell-1} + b^\ell \]
\[ a^\ell = \sigma(z^\ell) \]
\[ \hat{y}_\ell = f(a^\ell) \]

just like before

we have a loss on each \( \hat{y}_\ell \)

(e.g., cross-entropy)

\[ L(\hat{y}_{1:T}) = \sum_\ell L_\ell(\hat{y}_\ell) \]

some kind of readout function

“decoder”

could be as simple as a linear layer + softmax
Let’s draw the computation graph!

\[
\begin{align*}
\hat{y}_{i,1} & \quad \hat{y}_{i,2} & \quad \hat{y}_{i,3} & \quad \hat{y}_{i,4} \\
\uparrow & & & \uparrow \\
x_i & \quad & \quad & \quad \\
\end{align*}
\]

\[
z^\ell = W^\ell a^{\ell-1} + b^\ell \\
a^\ell = \sigma(z^\ell) \\
\hat{y}_\ell = f(a^\ell)
\]

\[
\mathcal{L}(\hat{y}_{1:T}) = \sum_\ell \mathcal{L}_\ell(\hat{y}_\ell)
\]

\[
\begin{align*}
\mathcal{L}_1 & \quad f_1 & \quad \text{Lin}_1 & \quad \sigma_1 \\
\mathcal{L}_2 & \quad f_2 & \quad \text{Lin}_2 & \quad \sigma_2 \\
\mathcal{L}_3 & \quad f_3 & \quad \text{Lin}_3 & \quad \sigma_3 \\
\sum & \quad & \mathcal{L} & \quad \\
\end{align*}
\]

not completely obvious how to do backprop on this!
Graph-structured backpropagation

Also called reverse-mode automatic differentiation

do the following at each layer \( f(x_f) \to y_f \)
starting with the last function, where \( \delta = 1 \)

\[
\delta_{x_f} = \frac{dL}{dx_f} = \frac{df}{dx_f} \delta_{y_f} \\
\delta_{y_f} = \delta_{y_f}^1 + \delta_{y_f}^2 \\
f(x_f) = y_f
\]

\[
\delta_{y_f}^1 = \frac{dL}{dy_f} \delta_{y_f} \\
\delta_{y_f}^2 = \frac{dL}{dy_f} \frac{1}{y_f} \delta_{y_f}
\]

Very simple rule:

For each node with multiple descendants in the computational graph:

Simply add up the delta vectors coming from all of the descendants

\[
e.g., a^\ell = \text{ReLU}(z^\ell) \\
z^\ell = W^\ell a^{\ell-1} + b^\ell
\]
Inputs and outputs at each step?

Examples:
- generating a text caption for an image
- translating some text into a different language

At each step:
\[
\hat{a}^{\ell-1} = \begin{bmatrix} a^{\ell-1} \\ x_{i,t} \end{bmatrix}
\]
\[
z^\ell = W^{\ell} \hat{a}^{\ell-1} + b^\ell
\]
\[
a^\ell = \sigma(z^\ell)
\]
\[
\hat{y}_\ell = f(a^\ell)
\]

A bit subtle why there are inputs at each time step! We’ll discuss this later.
What makes RNNs difficult to train?
RNNs are extremely deep networks

Imagine our sequence length was 1000+
that’s like backpropagating through 1000+ layers!

If we multiply many numbers together, what will we get?
- If most of the numbers are < 1, we get 0
- If most of the numbers are > 1, we get infinity
- We only get a reasonable answer if the numbers are all close to 1!

Intuitively:
vanishing gradients = gradient signal from later steps never reaches the earlier steps
very bad – this prevents the RNN from “remembering” things from the beginning!

“vanishing gradients”
big problem!

“exploding gradients”
could fix with gradient clipping
Promoting better gradient flow

**Basic idea:** (similar to what we saw before) we would really like the gradients to be close to 1 which gradients?

each layer:

\[
\begin{align*}
\bar{a}_{t-1} &= \begin{bmatrix} a_{t-1} \\ x_t \end{bmatrix} \\
\bar{z}_t &= W\bar{a}_{t-1} + b \quad a_t = \sigma(\bar{z}_t)
\end{align*}
\]

\[
a_t = q(a_{t-1}, x_t) \quad \text{“RNN dynamics”}
\]

dynamics Jacobian \[
\frac{dq}{da_{t-1}} \approx I
\]

best gradient flow

not always good – only good when we want to **remember**
sometimes we may want to **forget**
Promoting better gradient flow

**Basic idea:** (similar to what we saw before) we would really like the gradients to be close to 1

**Intuition:** want $\frac{dq_i}{da_{t-1,i}} \approx 1$ if we choose to remember $a_{t-1,i}$

for each unit, we have a little “neural circuit” that decides whether to remember or overwrite

if “remembering,” just copy the previous activation as it is

if “forgetting,” just overwrite it with something based on the current input

$$f_t \in [0, 1]$$

$$a_t = a_{t-1}f_t + g_t$$

$$\frac{dq_i}{da_{t-1,i}} = f_t \in [0, 1]$$
LSTM cells

Long short-term memory

Isn’t this all a little arbitrary?

Well, yes, but it ends up working quite well in practice, and much better than a naïve RNN!

output is 4x larger in dimensionality than RNN cell!
Why do LSTMs train better?

- The RNN output at previous time step changes very little step to step!
- "long term" memory changes all the time (multiplicative)
- "short term" memory

Mathematical equations:

\[ a_t = a_{t-1} f_t + g_t \]

- changes very little step to step!
- "long term" memory changes all the time (multiplicative)
- "short term" memory
Some practical notes

- In practice, RNNs almost always have both an input and an output at each step (we’ll see why in the next section).
- In practice, naïve RNNs like in part 1 almost never work.
- LSTM units are OK – they work fine in many cases, and dramatically improve over naïve RNNs.
  - Still require way more hyperparameter tuning than standard fully connected or convolutional networks.
- Some alternatives (that we’ll learn about later) can work better for sequences.
  - Temporal convolutions.
  - Transformers (temporal attention).
- LSTM cells are annoyingly complicated, but once implemented, they can be used the same as any other type of layer (hurray for abstraction!)
- There some variants of the LSTM that are a bit simpler and work just as well.
  - Gated recurrent unit (GRU).
Using RNNs
Autoregressive models and structured prediction

Most RNNs used in practice look like this:

\[
\begin{array}{cccc}
\hat{y}_{i,1} & \hat{y}_{i,2} & \hat{y}_{i,3} & \hat{y}_{i,4} \\
\uparrow & \uparrow & \uparrow & \uparrow \\
\downarrow & \downarrow & \downarrow & \downarrow \\
x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4}
\end{array}
\]

Why?

Most problems that require multiple outputs have strong dependencies between these outputs. This is sometimes referred to as structured prediction.

Example: text generation

Think: 0.3
Like: 0.3
Am: 0.4
Therefore: 0.3
Machine: 0.3
Learning: 0.3
Not: 0.4
Just: 0.4

I think therefore I am
I like machine learning
I am not just a neural network

We get a nonsense output even though the network had exactly the right probabilities!
Autoregressive models and structured prediction

Most RNNs used in practice look like this:

Why?

Most problems that require multiple outputs have strong dependencies between these outputs. This is sometimes referred to as structured prediction.

**Example:** Text generation

I think therefore I am
I like machine learning
I am not just a neural network

**Key idea:** Past outputs should influence future outputs!

We get a nonsense output even though the network had exactly the right probabilities!
Autoregressive models and structured prediction

How do we train it?

**Basic version:** just set inputs to be entire training sequences, and ground truth outputs to be those same sequences (offset by one step)

\[ x_{1:5} = (\text{“I”}, \text{“think”}, \text{“therefore”}, \text{“I”}, \text{“am”}) \]

\[ y_{1:5} = (\text{“think”}, \text{“therefore”}, \text{“I”}, \text{“am”}, \text{stop\_token}) \]

This teaches the network to output “am” if it sees “I think therefore I”

**Example:** text generation

I think therefore I am

I like machine learning

I am not just a neural network
Aside: distributional shift

The problem: this is a training/test discrepancy: the network always saw true sequences as inputs, but at test-time it gets as input its own (potentially incorrect) predictions.

This is called distributional shift, because the input distribution shifts from true strings (at training) to synthetic strings (at test time).

Even one random mistake can completely scramble the output!
Aside: scheduled sampling

An old trick from reinforcement learning adapted to training RNNs

Randomly decide whether to give the network a ground truth token as input during training, or its own previous prediction.

At the beginning of training, mostly feed in ground truth tokens as input, since model predictions are mostly nonsense.

At the end of training, mostly feed in the model’s own predictions, to mitigate distribution shift.

schedules for probability of using ground truth input token

Different ways to use RNNs

- **One to one**: e.g., image captioning
- **One to many**: e.g., activity recognition
- **Many to one**: e.g., machine translation
- **Many to many**: e.g., frame-level video annotation

In reality, we almost always use autoregressive generation like this.
RNN encoders and decoders

output is processed by a (non-recurrent) decoder before getting outputted
a bit less common, since we could just use more RNN layers

input is processed by a (non-recurrent) encoder before going into the RNN
very common with image inputs
RNNs with many layers

easy to stack as many RNN layers as necessary

could even implement recurrent RNN (or LSTM) convolutional layers!

just replace this with convolution
each “filter” becomes a little LSTM cell

\[
W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b = \begin{bmatrix} \tilde{f}_t \\ \tilde{i}_t \\ \tilde{g}_t \\ \tilde{o}_t \end{bmatrix}
\]
Bidirectional models

Example: speech recognition

Problem: the word at a particular time step might be hard to guess without looking at the rest of the utterance!

(for example, can’t tell if a word is finished until hearing the ending)

This is an even bigger problem in machine translation, but there we use slightly different types of models
Some (vivid) examples

THE SONNETS
by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his name.
But thou, coeffected to those own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a fountain where abundance lies:
Thy self thy self to thy self most true cast.
Then art thou new, the world's first born sonnet,
And only herald in the glory spring.
Within thee rules not teen but honest心愿,
And render chart made wise in rigging:
Ply the word, or else the time glidest to.
'To out the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep grooves into thy beaut's field,
Thy youth's jouful beauty so gazed on now,
Will be a cearl's word of small worth held.
Then being asked, where all thy beaut's lies,
Where all the honour of thy happy days,
To say, within thine own deep weather eyes,
Worn an all eating shame, and useless praise.
How much more praise deserveth thy beauty's use,
If thou couldst answer 'This is fair child of mine'
Shall earn my comfort, and make my old renown
Even by his beauty by succession crown.
'Tis won to be new made when thou art old.
And use thy blood warm when thou feel'st it cool.

Source: Andrej Karpathy
Some (vivid) examples

at first:

"Tyntd-iaphatawiaohrdemot lytdws e,tfti, astai f ogoh eoase rrranbyne 'nhthnee e plia tklrgd t o idoe ns,smtn  h ne etie h,hregtrs nigtike,aoaenns lng"

train more

"Tmont thithey" fomesscerliund Keushey. Thom here sheulke, ammerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome coaniogenn 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, pavi uay falling mistfot 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 oftended him. Pierre aking his soul came to the packs and drove up his father-in-law women.
Some (vivid) examples

PANDARUS:
Alas, I think he shall be come approachd and the day
When little strain 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.

VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not apt, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father’s world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy counsel I am great,
Murdered and by thy master’s ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder’d at the deeds,
So drop upon your lordship’s head, and your opinion
Shall be against your honour.
Some (vivid) examples

The Stacks Project: open source algebraic geometry textbook

Source: Andrej Karpathy
Some (vivid) examples

Source: Andrej Karpathy
Some (vivid) examples

OpenAI GPT-2 generated text

Input: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Output: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Source: GPT-2
Recurrent neural networks (RNNs): neural networks that can process variable-length inputs and outputs
- Could think of them as networks with an input & output at each layer
- Variable depth
- Depth = time

Training RNNs is very hard
- Vanishing and exploding gradients
- Can use special cells (LSTM, GRU)
- Generally need to spend more time tuning hyperparameters

In practice, we almost always have both inputs and outputs at each step
- This is because we usually want structured prediction
- Can use scheduled sampling to handle distributional shift

Many variants for various purposes
- Sequence to sequence models (more on this later)
- Bidirectional models
- Can even “RNN-ify” convolutional layers!