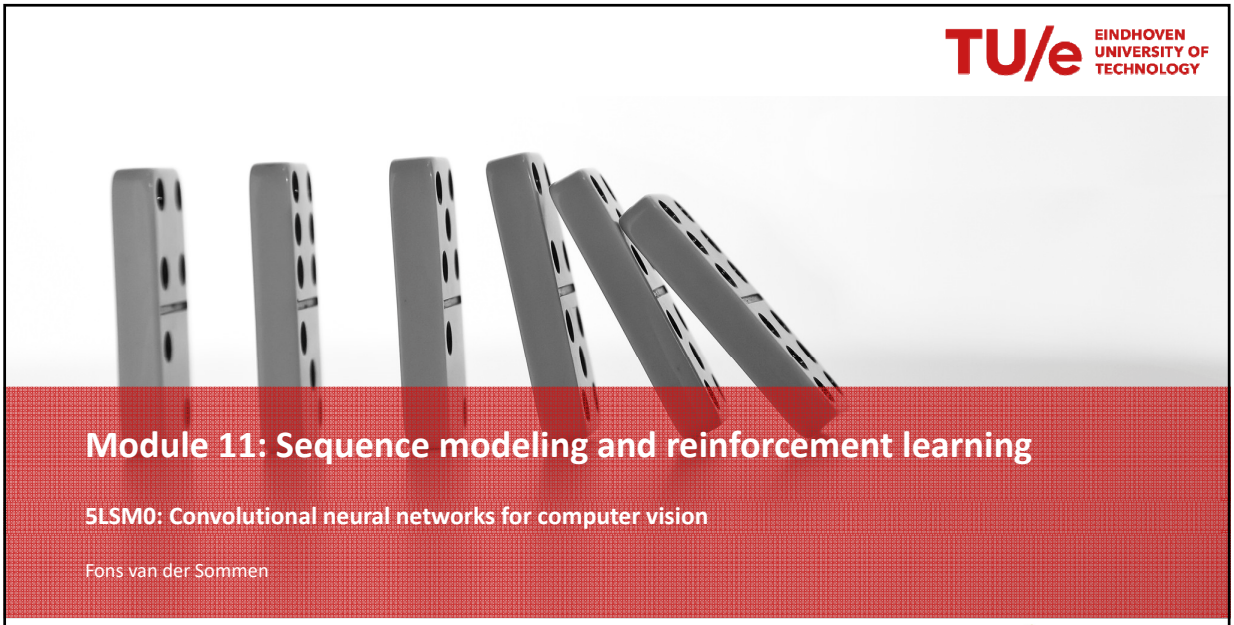


Module 11: Sequence modeling and reinforcement learning

5LSM0: Convolutional neural networks for computer vision

Fons van der Sommen

Electrical Engineering / VCA research group

Administrative

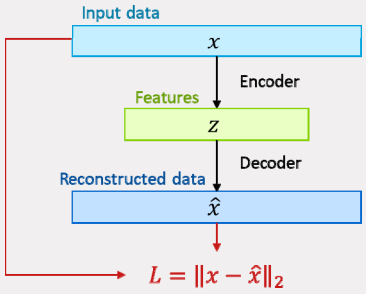
Paper sessions

- Next Thursday (March 28th) hours 1-4
- Please indicate a preferred time-slot on Canvas (also check for more details)
- Upload presentations through Canvas Assignments by Wednesday March 27th 23:59
  - *Otherwise penalty on grade for presentation*

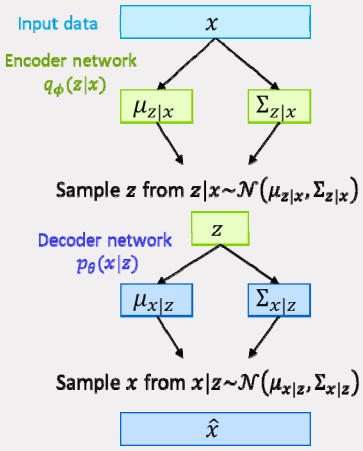
Final assignment

- Detailed announcement next week…

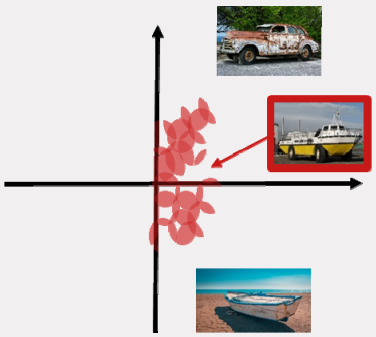
Last time(s)



Auto-encoders



Variational Auto-encoders



$$\mathcal{L}(x^{(i)}, \theta, \phi) = \mathbf{E}_z[\log p_\theta(x^{(i)}|z)] - D_{KL}\left(q_\phi(z|x^{(i)})||p_\theta(z)\right)$$

Last time(s)

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x\sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)} \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)\right]$$

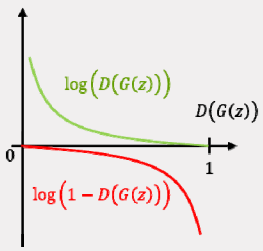
Generator network

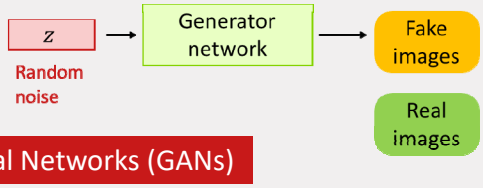
z

$$\max_{\theta_d} \left[\mathbb{E}_{x\sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)} \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)\right]$$

Gradient ascent for both θ_d and θ_g

$$\max_{\theta_g} \left[\mathbb{E}_{z\sim p(z)} \log\left(D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)\right]$$

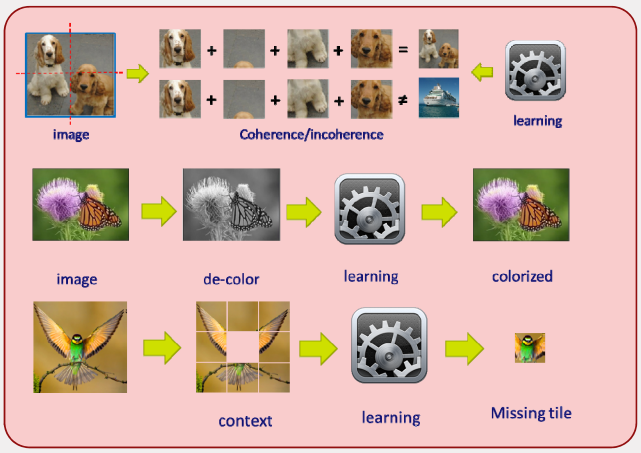




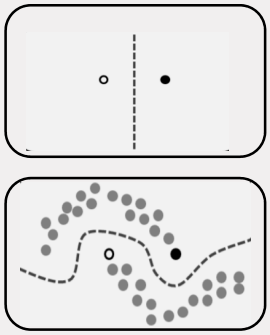
Generative Adversarial Networks (GANs)

Last time(s)

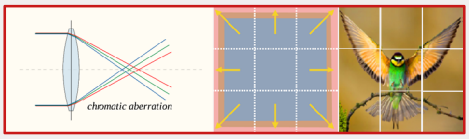
Self-supervised learning



Unlabeled data may reveal structure



Beware of pit-falls!



This time

Sequential modeling

How can we model sequential relations using deep neural networks?

- Recurrent Neural Networks (RNNs)

How can we optimize gradient flow over the sequential dimension? (e.g. time)

- Long Short-Term Memory (LSTM)

Reinforcement learning

Can we learn to play Atari games with deep learning?:D

- (And do other cool stuff too…)
- Deep Q-learning

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Sequential modeling

So far…

CNNs to do

- Image classification; Object detection/segmentation; Density estimation for data generation

Mostly fixed size input → fixed size output

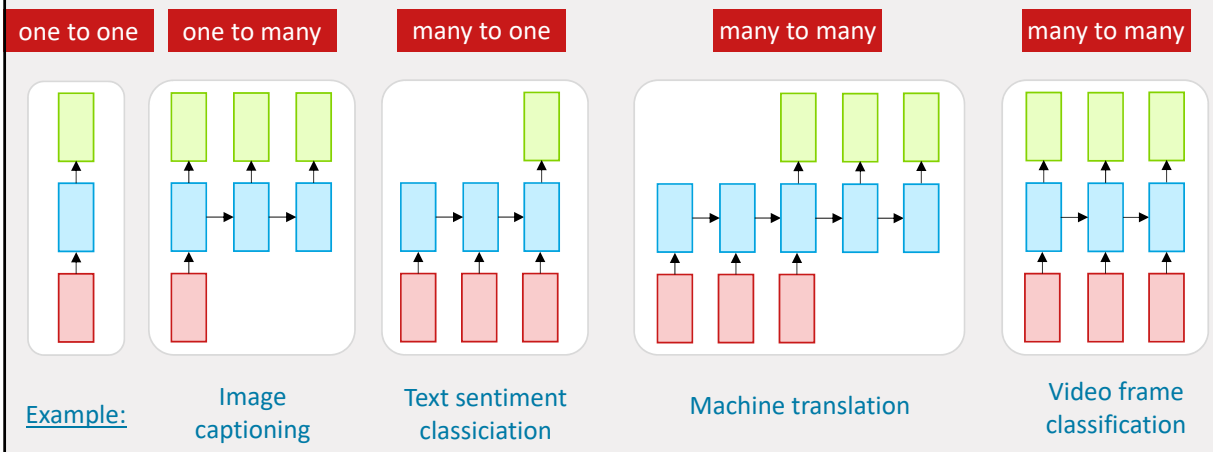
- If we didn't we required several passes through network or auxiliary methods

How would we do

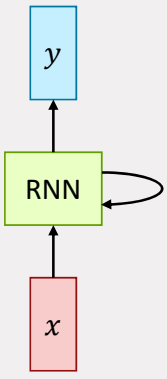
- Image captioning [one image to variable number of words]
- Sentiment classification [variable number of words to one sentiment]
- Translation [variable number of words to variable number of words]

Recurrent Neural Networks (RNNs)

Recurrent neural networks



Recurrent neural networks



Recurrence relation

- For a certain time t:

Function with parameter W — Input

$$h_t = f_W(h_{t-1}, x_t)$$

New state — Old state

Note: parameters W fixed over time

Straightforward option:

- Two linear relations:
  - *old state* → *new state*
  - *Input* → *new state*
- Linear output relation:
  - *New state* → *output*

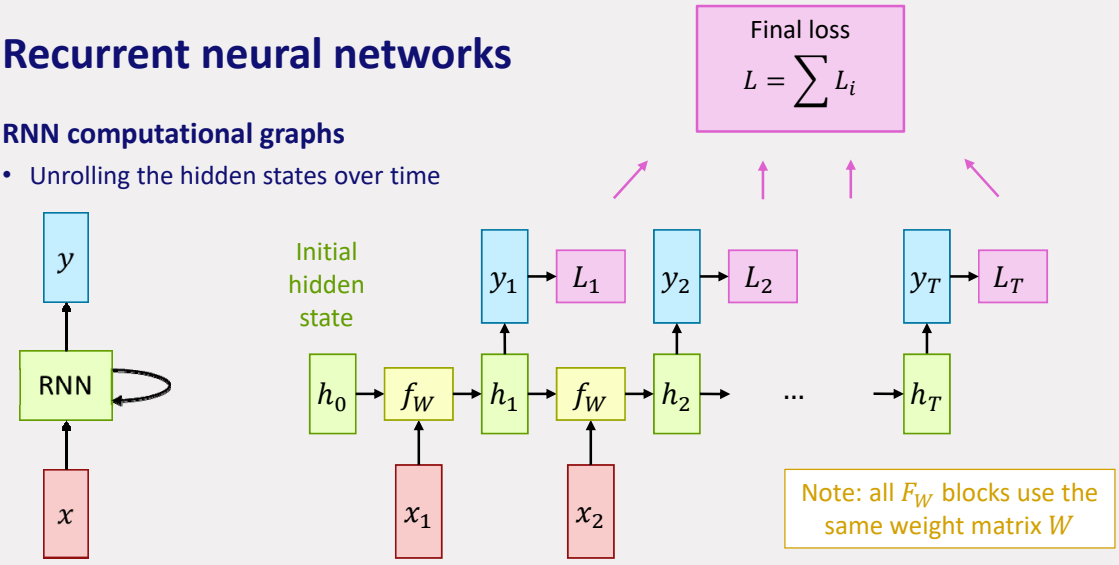
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Recurrent neural networks

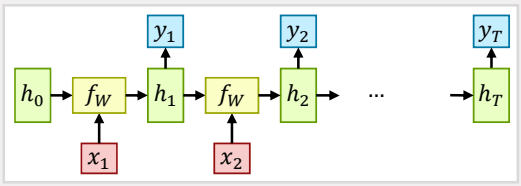
RNN computational graphs

- Unrolling the hidden states over time



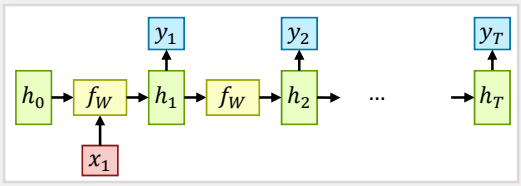
Recurrent neural networks

RNN computational graphs

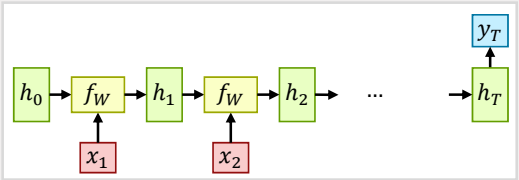


Many to many

Use single input to initialize the hidden state of the model



Many to one



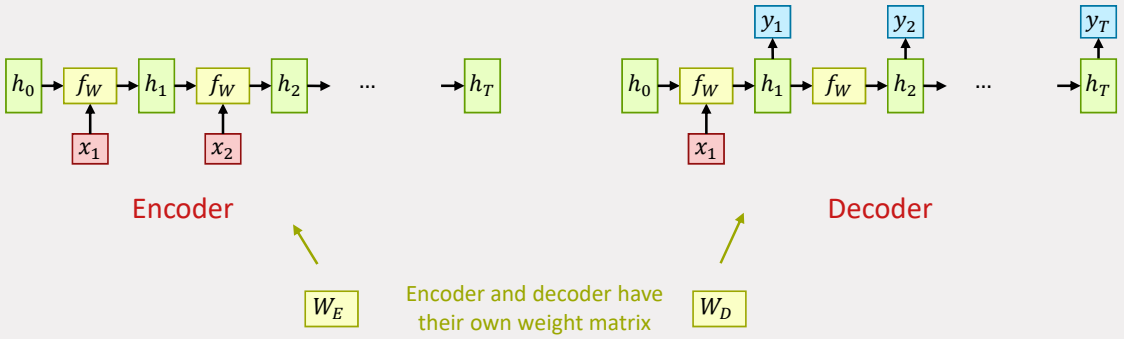
Final hidden state summarized all of the context of the entire sequence

One to many

Recurrent neural networks

RNN computational graphs: Sequence to sequence

- Many-to-one followed by one-to-many



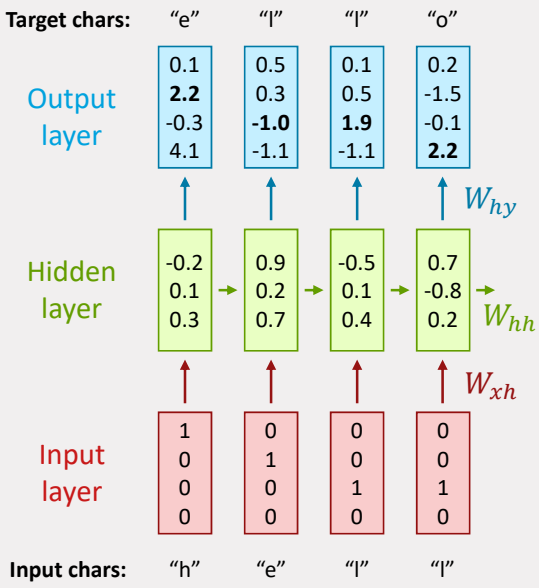
Recurrent neural networks

RNN computational graphs: Sequence to sequence

- Many-to-one followed by one-to-many

Example: character-level language model

- Predict next letter
- Vocabulary: [h, e, l, o]
- Training sequence: "hello"



**Example from cs231n lecture 10 slides 34-36*

Recurrent neural networks

RNN computational graphs: Sequence to sequence

- Many-to-one followed by one-to-many

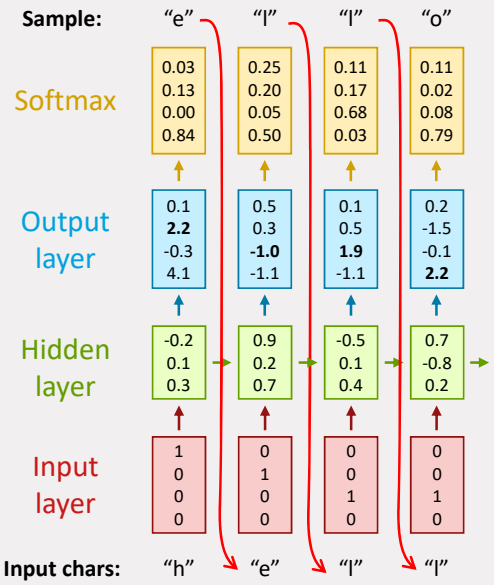
Example: character-level language model

- Predict next letter
- Vocabulary: [h, e, l, o]
- Training sequence: "hello"

Q: Why not feed back max sample?

At test time we can sample!

- Start with a letter, and sample from output to use for next input in time

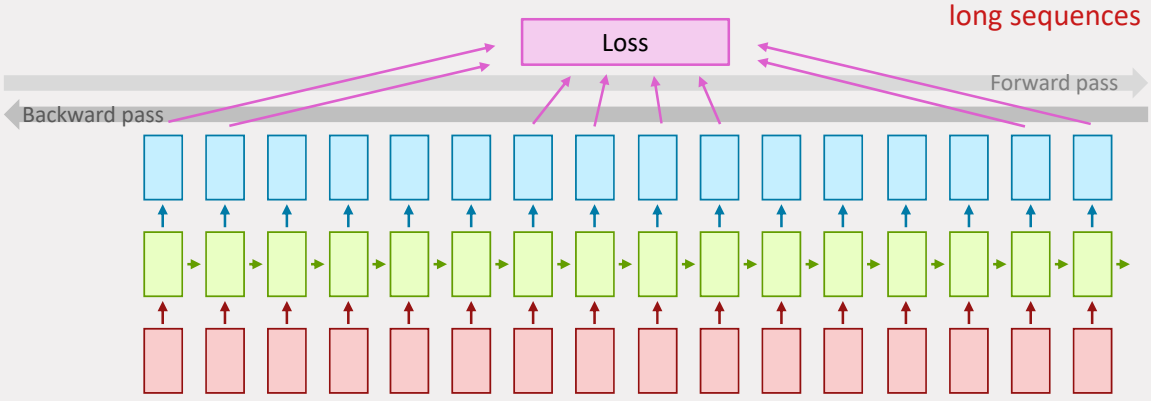


*Example from cs231n lecture 10 slides 34-36

Recurrent neural networks

RNN training: backpropagation through time

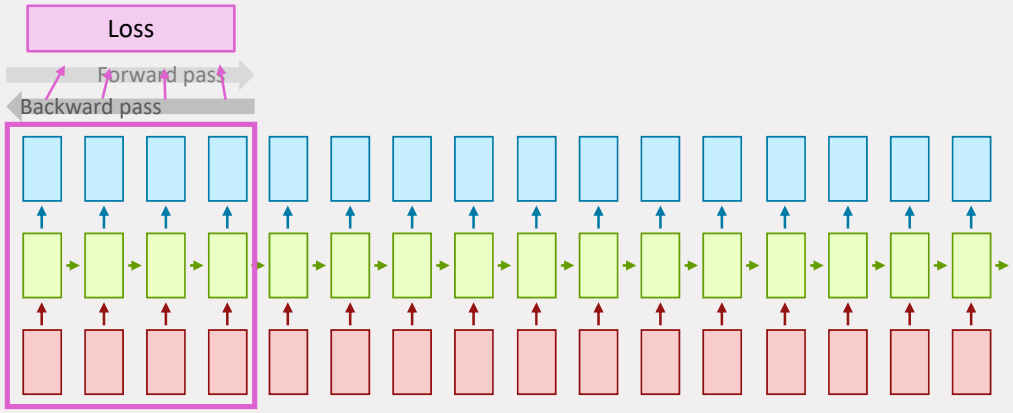
Problem: backprop could take ages for long sequences



*Example from cs231n lecture 10 slides 34-36

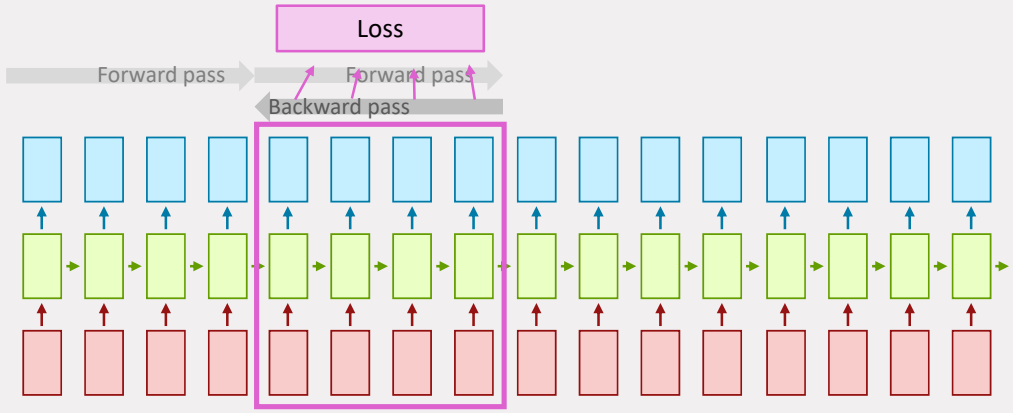
Recurrent neural networks

RNN training: truncated backpropagation through time



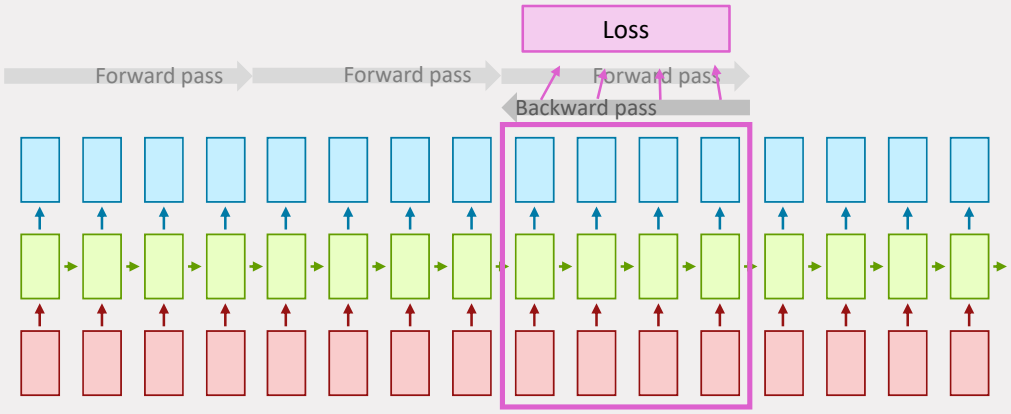
Recurrent neural networks

RNN training: truncated backpropagation through time



Recurrent neural networks

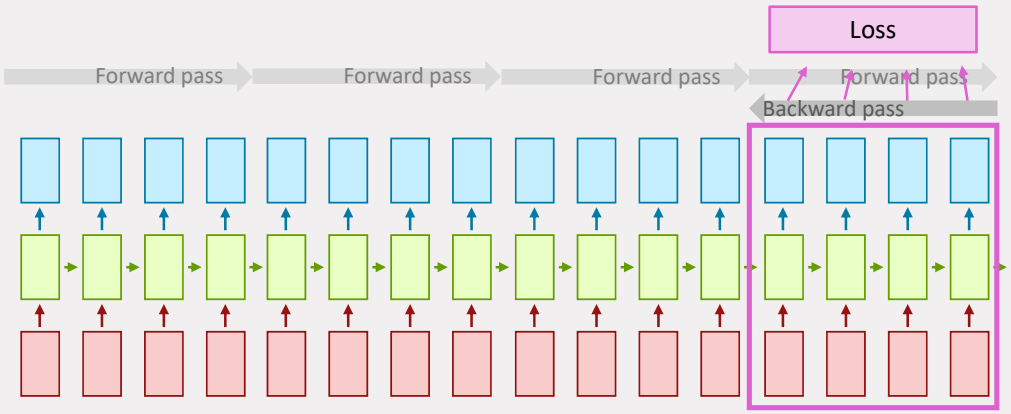
RNN training: truncated backpropagation through time



Recurrent neural networks

Q: What other training approach looks similar?

RNN training: truncated backpropagation through time



Cool stuff you can do 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.
```

```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, 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 council 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.
```

Write in Shakespeare style

*Example from cs231n lecture 10 slides 34-36

TU/e

Cool stuff you can do 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 compariconly 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 $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',s''}$ 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 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

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

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 $X_{spaces,étale}$ 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* $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ *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 $U \cap U = \coprod_{n=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes 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. □

Produce mathy LaTeX source code

*Example from cs231n lecture 10 slides 34-36

TU/e

Cool stuff you can do with RNNs

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x0000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

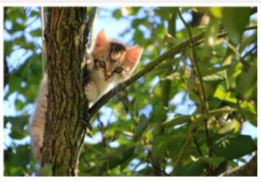
Produce code

*Example from cs231n lecture 10 slides 34-36

Cool stuff you can do with RNNs

Image captioning

A cat sitting on a suitcase on the floor



A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

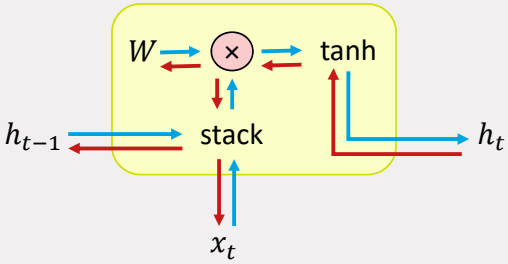
A man riding a dirt bike on a dirt track

*Example from cs231n lecture 10 slides 34-36

RNN gradient flow

What happens during RNN training?

During backwards pass need to multiply by W



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

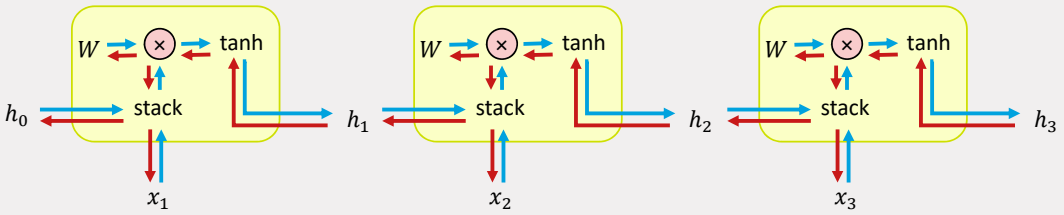
$$= \tanh\left((W_{hh} \quad W_{xh})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

We can stack h_{t-1} and x_t together

RNN gradient flow

What happens during RNN training?



Backprop involves successive multiplication by W^T

Largest singular value > 1
Exploding gradient
→ Hack: gradient clipping

Largest singular value < 1
Vanishing gradient
→ Change architecture: **LSTMs**

RNN gradient flow

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation **(1997)**

Long Short-Term Memory (LSTM)

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma(\cdot) \\ \sigma(\cdot) \\ \sigma(\cdot) \\ \tanh(\cdot) \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

Cell state

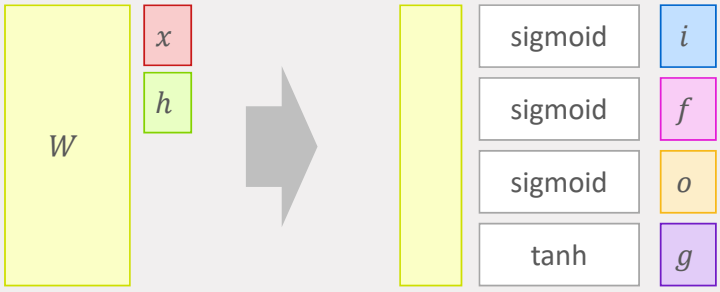
$$h_t = o \odot \tanh(c_t)$$

Hidden state

Long Short Term Memory (LSTM)

Introduction of a "cell state"

i	Input gate
f	Forget gate
o	Output gate
g	… Gate



$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

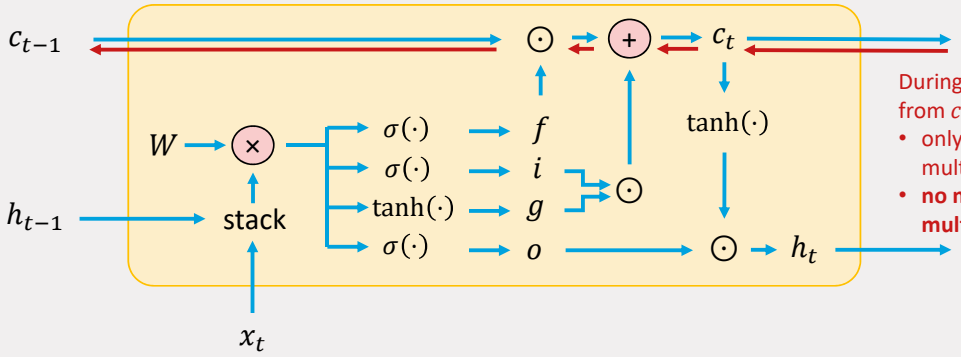
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma(\cdot) \\ \sigma(\cdot) \\ \sigma(\cdot) \\ \tanh(\cdot) \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

Long Short Term Memory (LSTM)

$$c_t = f \odot c_{t-1} + i \odot g$$

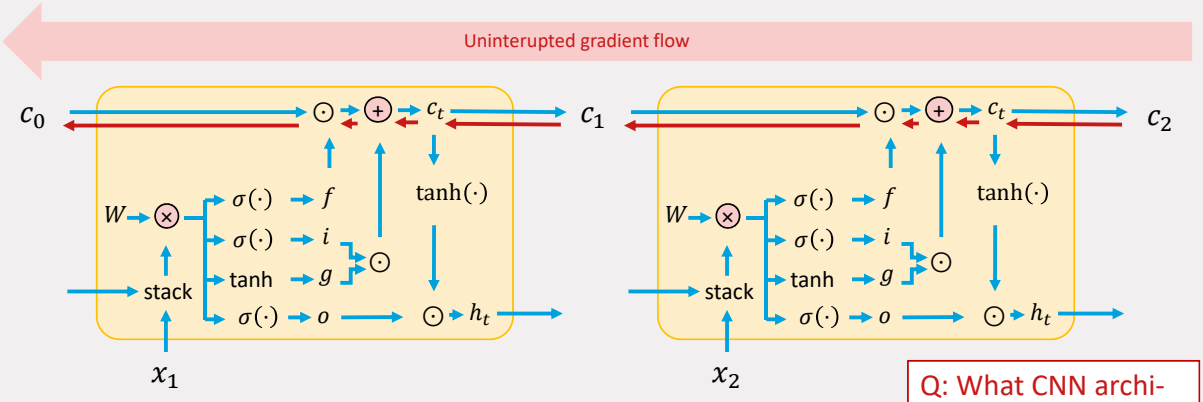
$$h_t = o \odot \tanh(c_t)$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma(\cdot) \\ \sigma(\cdot) \\ \sigma(\cdot) \\ \tanh(\cdot) \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$



During backprop from c_t to c_{t-1}:
- only elementwise multiplication by f
- **no matrix multiplication by W**

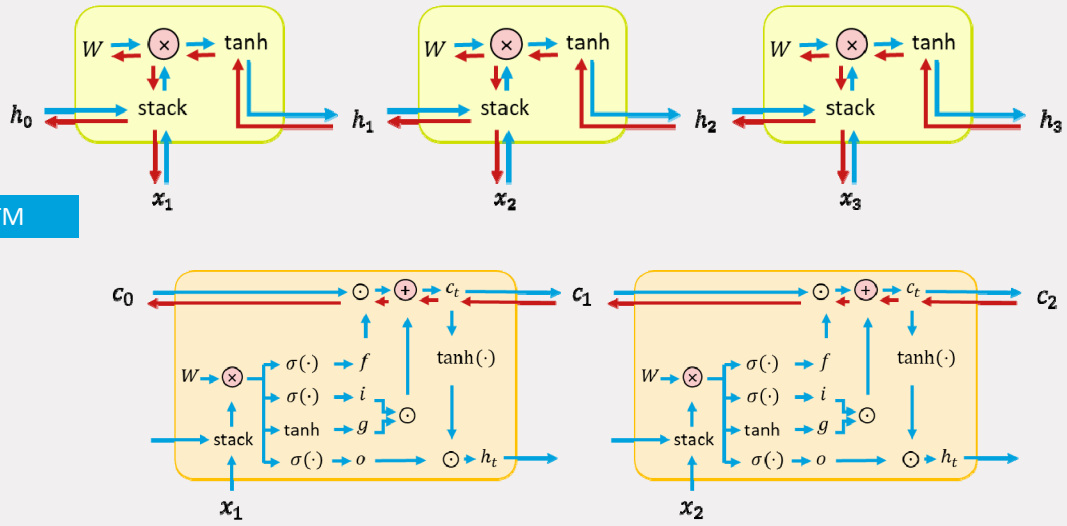
Long Short Term Memory (LSTM)



Q: What CNN architecture looks similar?

Vanilla RNN

LSTM



Temporal modeling summary

Recurrent Neural Networks (RNNs)

- Add recurrent relation in the network to model sequential relations
- Unroll the network over time and apply standard backpropagation to train
- In practice: vanilla RNNs don't work that well due to vanishing/exploding gradients

Long Short-Term Memory (LSTM)

- Introduce an additional cell state, governing the input and output of the hidden state
  - *Input-, output-, forget-, "gating"- gates*
- Uninterupted gradient flow

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Reinforcement learning

Slides from cs231n lecture 14 [1-62]
(http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture14.pdf)

Machine learning so far…

Supervised learning

- We have data and we have labels

Semi-/self supervised learning

- We have data and we kinda create our own labels

Unsupervised learning

- We have data, but no labels

Reinforcement learning

- We have no data..
  - *(But we do have an environment that can provide us with data! And even with rewards!)*

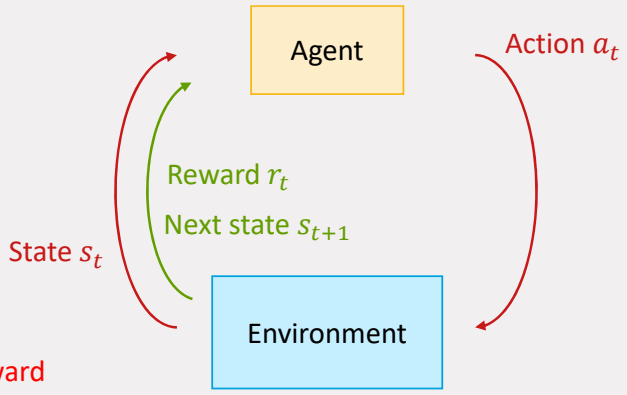
Reinforcement learning

Reinforcement learning (RL) setup

- Environment
  - *Gives agent a state s_t*
- Agent
  - *Takes action a_t*
  - *Receives reward r_t and next state s_{t+1} from environment*

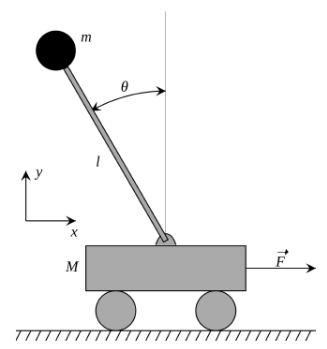
→ Goal: take actions that maximize reward

Process continues until agent receives a terminal state



Reinforcement learning

Example: cart-pole problem



Objective:

- Balance a pole on top of a movable cart

State:

- Angle, angular speed, position, horizontal velocity

Action:

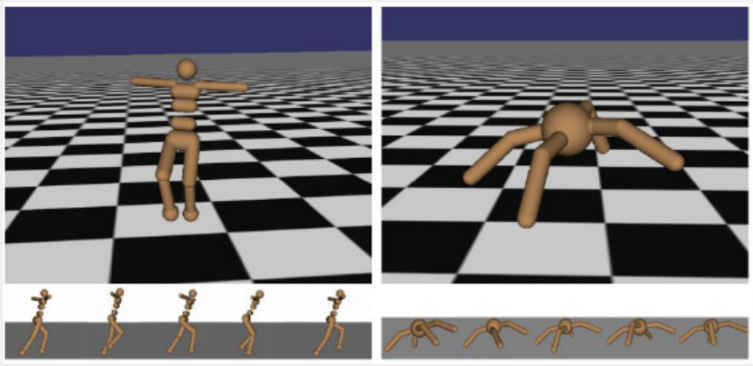
- Horizontal force applied on the cart

Reward:

- 1 at each time step if the pole is upright

Reinforcement learning

Example: Robot locomotion



Objective:

- Make the robot move forward

State:

- Angle and position of the joints

Action:

- Torques applied on joints

Reward:

- 1 at each time step upright + forward movement

Reinforcement learning

Example: Atari games

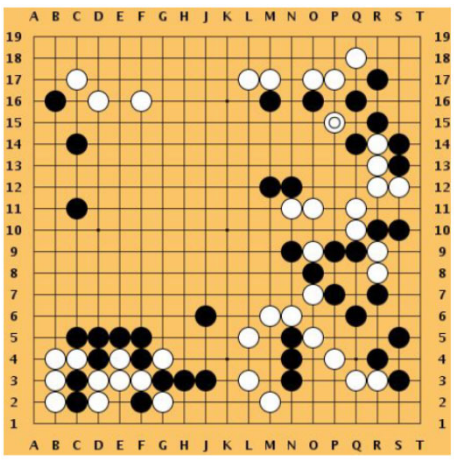
Objective:
- Complete the game with the highest score

State:
- Raw pixel inputs of the game state

Action:
- Game controls e.g. Left, Right, Up, Down

Reward:
- Score increase/decrease at each time step

Reinforcement learning



Example: Go

Objective:
- Win the game!

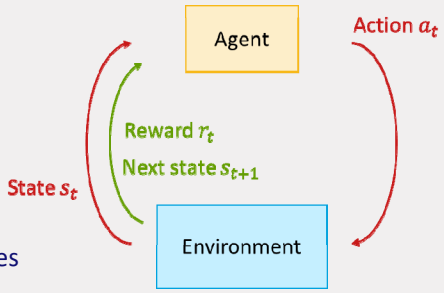
State:
- Position of all the pieces

Action:
- Where to put the next piece down

Reward:
- 1 if win the game, 0 otherwise

Reinforcement learning

How to mathematically formalize this?



Markov Decision Process (MDP)

- Markov Property: Current state completely characterizes the state of the world.
- Defined by $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$	Set of possible states
$\mathcal{A}$	Set of possible actions
$\mathcal{R}$	Distribution over reward, given state-action pair
$\mathbb{P}$	Transition probability: distribution over next state given state-action pair
γ	Discount factor: how much do we value early vs. late rewards

Reinforcement learning

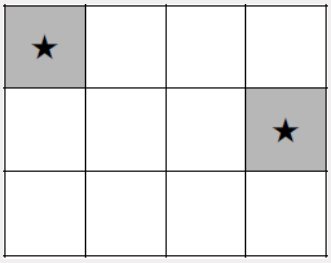
How does the MDP work?

- At time step $t = 0$, environment samples initial state $\boldsymbol{s_0}$ from initial state distribution $\boldsymbol{p(s_0)}$
- Then for $t = 0$ until done:
  - *Agent selects action a_t*
  - *Environment samples reward $r_t \sim \mathcal{R}(\cdot \mid s_t, a_t)$*
  - *Environment samples next state*
  - *Agent receives reward r_t and mext state s_{t+1}*
- A policy $\boldsymbol{\pi}$ is a function from S to A that specifies what action to take in each state
- **Objective**: find policy $\boldsymbol{\pi}^*$ hat maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$

Reinforcement learning

A simple MDP: Grid World

Actions = {
1. Right
2. Left
3. Up
4. Down
}

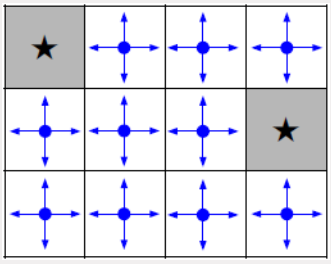


Set a negative "reward" for each transition (e.g. $r = -1$)

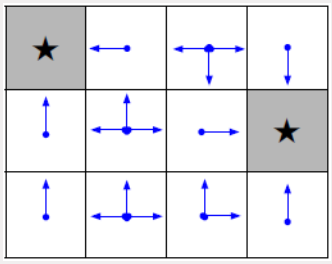
Objective: reach one of the terminal states in the least number of actions

Reinforcement learning

A simple MDP: Grid World



Random policy



Optimal policy

Reinforcement learning

Want to find the optimal policy π^*

- How do we handle the randomness (initial state, transition probability,…)?

→ Maximize the expected sum of the rewards!

- Formally:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid \pi\right] \quad \text{with} \quad s_0 \sim p(s_0), a_t \sim \pi(\cdot \mid s_t), s_{t+1} \sim \mathbb{P}(\cdot \mid s_t, a_t)$$

Reinforcement learning

Definitions: value function & Q-value function

How good is a state?

- The **value function** at state s, is the expected cumulative reward from following the policy from that state s:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, \pi\right]$$

How good is a state-action pair?

- The **Q-value function** at state s and action a, is the expected cumulative reward from taking action a in state s and ten following the policy:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

Reinforcement learning

Bellman equation

- The optimal Q-value function Q^* is the maximum expected cumulative reward achievable from a given state-action pair:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \,|s_0 = s, a_0 = a, \pi\right]$$

- Q^* satisfies the **Bellman equation**:

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') \,|s,a\right]$$

- Intuition: if the optimal state-action values for the next time-step are known $Q(s',a')$, then the optimal strategy is to take the action that maximizes the expected value of $\mathrm{r} + \gamma Q^*(s',a')$

The optimal policy π^* corresponds to taking the best action in any state specified by Q^*

Reinforcement learning

Value iteration algorithm: use Bellman equation as an iterative update

$$Q^*_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*_i(s',a') \,|s,a\right]$$

- Q_i will converge to Q^* as $i \to \infty$

Problem: this is not scalable:

- We must compute $Q(s,a)$ for every state-action pair.
- If stat is e.g. current game state pixels, this is computationaly infeasible to compute for the entire state space!

Deep Q-learning

Solution: use a function approximator to estimate $Q(s,a)$, e.g. a neural network!

Reinforcement learning

We want to find a Q-function that satisfies the Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') \,|s,a\right]$$

Forward pass

- Loss function

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \mathcal{R}}\left[\left(y_i - Q(s,a;\theta_i)\right)^2\right]$$

where

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') \,|s,a\right]$$

Iteratively try to make the Q-value close to the target value y_i it should have

If Q-function corresponds to optimal Q^* (and optimal policy π^*)

Backward pass

Gradient update (with respect to Q-function parameters θ)

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \mathcal{R}; s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a';\theta_{i-1}) - Q(s,a;\theta_i) \nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

Reinforcement learning

Case study: Atari games

Objective:

- Complete the game with the highest score

State:

- Raw pixel inputs of the game state

Action:

- Game controls e.g. Left, Right, Up, Down

Reward:

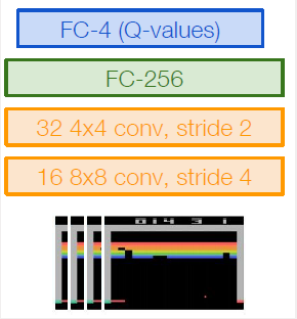
- Score increase/decrease at each time step

Reinforcement learning

Case Study: Playing Atari Games

- $Q(s, a; \theta)$ neural network with weights θ
- Current state s_t: 84×84×4 stack of last 4 frames
  - *(after RGB->grayscale conversion, downsampling, and cropping)*

A single feedforward pass to compute Q-values for all actions from the current state → efficient!



Last FC layer has 4-d output for 4 actions

Conv layers

Input state s_t

Reinforcement learning

Learning from batches of consecutive samples is problematic:

- Samples are correlated → inefficient learning
- Current Q-network parameters determines next training samples
  - *(e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) → can lead to bad feedback loops*

Address these problems using experience replay

- Continually update a replay memory table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Reinforcement learning

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Play M episodes (full games)

For each time-step of the game

With small probability, select a random action, otherwise select greedy action from current policy

Take the action a_t, and observe the reward r_t and next state s_t

Store transition in replay memory

Experience Replay: Sample a random minibatch of transitions from replay memory and perform a gradient descent step

Reinforcement learning

https://www.youtube.com/watch?v=V1eYniJ0Rnk

Reinforcement learning

Summary

Agent in an environment

- The agent takes actions and receives rewards from the environment
- The environment also controls the state of the agent

Q-function evaluates the value of a certain state-action pair

- Infeasible to compute…
- We can use neural networks to approximate the Q-function!