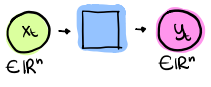
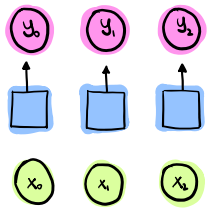
AVA SOLEIMANY MIT 6.S191

SEQUENCE MODELING APPLICATION
- MANY TO ONE (SENTIMENT CLASSIFICATION)
- ONE TO MANY (IMAGE CAPTIONING)
- MANY TO MANY (MACHINE TRANSLATION)

THE PERCEPTRON REVISITED



WE CAN REPLICATE THE SAME PROCESS FOR EACH TIMESTEP



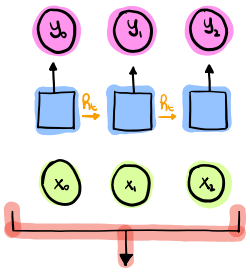
$\hat{y}_t = f(x_t)$

WE NEED A WAY TO RELATE THE NETWORK COMPUTATION AT A PARTICULAR TIME TO ITS HISTORY!

WE CAN LINK THE INFORMATION AND THE COMPUTATION OF THE NETWORK AT DIFFERENT TIMESTEPS TO EACH OTHER!

RECURRENCE RELATION

THIS CAN BE DONE BY HAVING AN INTERNAL MEMORY OR STATE h_t

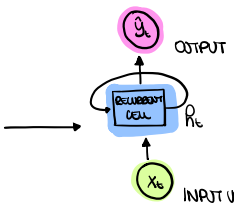


$\hat{y}_t = f(x_t, h_{t-1})$

OUTPUT INPUT PAST MEMORY STEP

NOW THE OUTPUT IS DEPENDENT NOT ONLY ON THE INPUT AT A PARTICULAR TIME STEP BUT ALSO ON THE STATE OF THE PREVIOUS

THIS CAN BE SEEN AS



RNNs → RECURRENT NEURAL NETWORKS

↳ RNNs HAVE A STATE h_t, THAT IS UPDATED AT EACH TIMESTEP AS A SEQUENCE IS PROCESSED

THIS IS DONE BY APPLY A RECURRENCE RELATION AT EVERY TIMESTEP TO PROCESS A SEQUENCE

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

CELL STATE — FUNCTION WITH WEIGHTS W — INPUT — OLD STATE

IN PSEUDO CODE:

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0] # THE FIRST HIDDEN STATE
sentence = ["I", "love", "nebuly"]
for word in sentence: # THE RECURRENCE RELATION IS CAPTURED BY THIS LOOP
    prediction, hidden_state = my_rnn(word, hidden_state)
next_word_prediction = prediction # TOKEN PREDICTION
```

KEY POINT → THE SAME FUNCTION AND SET OF PARAMETERS ARE USED AT EVERY TIMESTEP.

THE INTERNAL COMPUTATION OF THE RNN INCLUDES BOTH THIS INTERNAL STATE OUTPUT AS WELL AS ULTIMATELY TRYING TO OUTPUT THE PREDICTION WE ARE INTERESTED IN (THE OUTPUT VECTOR $\hat{y}_t$)

GIVEN AN INPUT VECTOR x_t WE PASS THAT IN TO COMPUTE THE RNN INTERNAL STATE COMPUTATION

BECAUSE WE HAVE THESE TWO INPUTS, WE HAVE TWO INDEPENDENT WEIGHT MATRICES

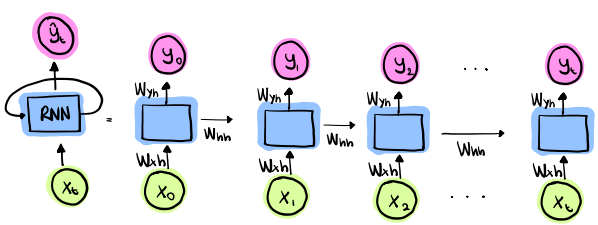
$$h_t = \tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t)$$ UPDATE HIDDEN STATE

BREAKING THIS FUNCTION DOWN, WHAT IT'S DOING IT'S JUST A STANDARD NN OPERATION. IN THIS CASE WE MULTIPLY BOTH THE PAST HIDDEN STATE BY A WEIGHT MATRIX W AS WELL AS THE CURRENT INPUT x_t BY ANOTHER WEIGHT MATRIX, AND THEN WE ADD THEM TOGETHER AND APPLY A NON-LINEARITY.

$$\hat{y}_t = W_{hy}^T h_t$$

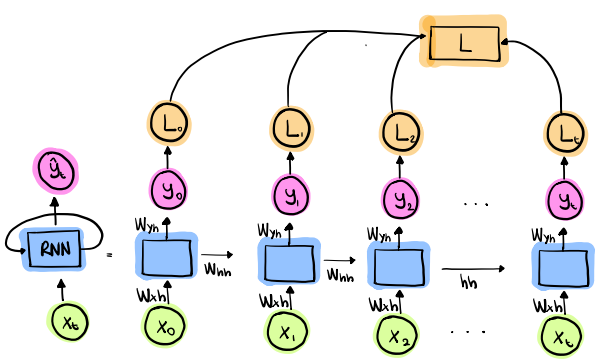
THE FINAL STEP IS TO GENERATE THE OUTPUT FOR A GIVEN TIMESTEP. THIS IS DONE BY TAKING THAT INTERNAL STATE AND SIMPLY MODIFYING IT, FOLLOWING A MULTIPLICATION BY ANOTHER WEIGHT MATRIX AND THEN USING THIS AS OUR GENERATED OUTPUT.

RNN REPRESENTED AS A COMPUTATIONAL GRAPH UNROLLED ACROSS TIME:



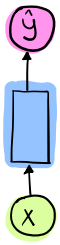
WE RE-USE THE SAME WEIGHT MATRICES AT EVERY TIMESTEP.

BUT HOW WE TRAIN THIS NETWORK? → WE NEED A LOSS!!

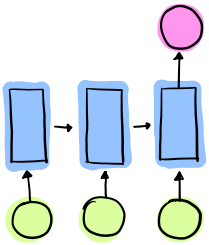


WE CAN GENERATE AN INDIVIDUAL LOSS FOR EACH OF THIS INDIVIDUAL TIMESTEPS ACCORDING TO WHAT THE OUTPUT OF THE TIMESTEP IS. AND WE CAN GENERATE A TOTAL SUM LOSS BY TAKING THIS TIMESTEPS AND SUM THEM TOGETHER.

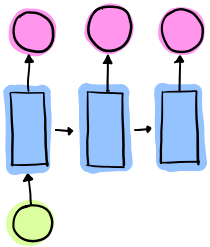
RNNs FOR SEQUENCE MODELING



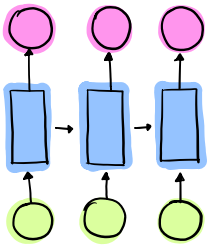
ONE TO ONE "VANILLA" NEURAL NETWORK (E.G. BINARY CLASSIFICATION)



MANY TO ONE (E.G. SENTIMENT ANALYSIS)



ONE TO MANY (E.G. TEXT GENERATION, IMAGE CAPTIONING)



MANY TO MANY (E.G. TRANSLATION AND FORECASTING)

SEQUENCE MODELING - DESIGN CRITERIA

↳ WE NEED TO:

1) HANDLE VARIABLE-LENGTH SEQUENCES → FEEDFORWARD NEURAL NETWORKS ARE NOT ABLE TO HANDLE VARIABLE SEQUENCE LENGTH BECAUSE THEY HAVE INPUT OF FIXED DIMENSIONALITY. BUT BECAUSE WITH RNNs WE ARE UNROLLING ACROSS TIME WE ARE ABLE TO HANDLE THIS.
2) TRACK LONG-TERM DEPENDENCIES → WE NEED INFORMATION FROM THE DISTANT PAST TO ACCURATE PREDICT THE NEXT WORD.
3) MAINTAIN INFORMATION ABOUT ORDER
4) SHARE PARAMETERS ACROSS THE SEQUENCE TO BE ABLE TO KEEP TRACK OF THESE DEPENDENCIES

REPRESENTING THE LANGUAGE TO A NN

↳ NEURAL NETWORKS REQUIRE NUMERICAL INPUTS, WE NEED A WAY TO REPRESENT THE WORDS NUMERICALLY → EMBEDDING → TRANSFORM INDEXES INTO A VECTOR OF FIXED SIZE

① VOCABULARY

CORPUS OF WORDS

↓

WALK AM MY FOR THIS I TOOK CAT MORNING

② INDEXING

WORD TO INDEX

↓

A → 1
CAT → 2
... ...
WALK → N

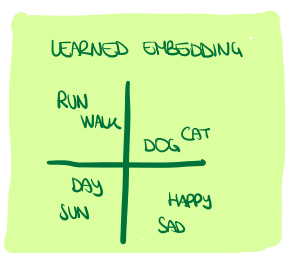
③ EMBEDDING

THESE INDECES CAN BE MAPPED TO AN EMBEDDING WHICH IS JUST A FIXED LENGTH VECTOR.

ONE-HOT EMBEDDING
"CAT" = [0, 1, 0, 0, .., 0]
↑ i-TH INDEX

ANOTHER OPTION TO GENERATE AN EMBEDDING IS TO USE SOME SORT OF MACHINE LEARNING MODEL TO LEARN AN EMBEDDING

IDEA: TAKING AN INPUT OF WORDS THAT ARE GOING TO BE INDEXED NUMERICALLY, WE CAN LEARN AN EMBEDDING OF THOSE WORDS IN SOME LOWER DIMENSIONAL SPACE.



USING MACHINE LEARNING WE CAN MAP THE MEANING OF THE WORDS TO AN ENCODING THAT IS MORE INFORMATIVE ⇒ WORDS THAT HAVE SIMILAR MEANING WILL HAVE SIMILAR EMBEDDINGS.

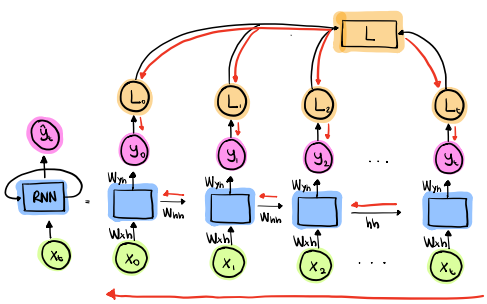
BACKPROPAGATION THROUGH TIME (BPTT)

↳ IN THE RNNs THE FORWARD PASS THROUGH THE NETWORK CONSISTS OF GOING FORWARD ACROSS TIME, COMPUTING THE INDIVIDUAL LOSS VALUES AT THE INDIVIDUAL TIME STEPS AND THEN SUMMING THEM TOGETHER.

↓

TO BACKPROPAGATE, INSTEAD OF BACKPROPAGATING ERRORS THROUGH A SINGLE FEEDFORWARD NETWORK, WE HAVE TO BACKPROPAGATE ERRORS INDIVIDUALLY ACROSS EACH TIME STEP AND ACROSS EACH TIME STEP.

↦



BETWEEN EACH TIMESTEP WE NEED TO PERFORM THESE INDIVIDUAL MATRIX MULTIPLICATION → COMPUTING THE GRADIENT WITH RESPECT TO h_0 INVOLVES MANY FACTORS OF W_{hh} AND REPEATED GRADIENT COMPUTATION. **WHY THIS CAN BE PROBLEMATIC:**

- MANY VALUES (WEIGHTS OR GRADIENTS) > 1 → **EXPLODING GRADIENT**, THE GRADIENTS ARE BECOMING EXTREMELY LARGE DUE TO THIS REPEATED MULTIPLICATION OPERATION. **POSSIBLE SOLUTION:** GRADIENT CLIPPING: SCALE BIG GRADIENTS.
- MANY VALUES < 1 → **VANISHING GRADIENT**: MULTIPLY MANY SMALL VALUES TOGETHER ⇒ ERRORS DUE TO FURTHER BACK TIME STEPS HAVE SMALLER AND SMALLER GRADIENTS ⇒ BAD PARAMETERS TO CAPTURE SHORT TERM DEPENDENCIES

⇓

THE MODEL WILL IGNORE THE LONG TERM DEPENDENCIES THAT MAY EXIST.

T

POSSIBLE SOLUTIONS TO THE VANISHING GRADIENT PROBLEM

① ACTIVATION FUNCTIONS

USING RELU PREVENTS f' FROM SHRINKING THE GRADIENTS WHEN $x > 0$

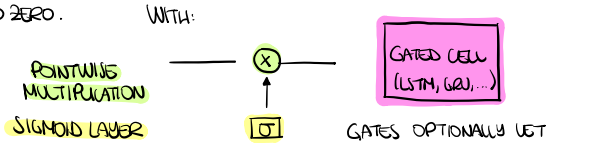
② PARAMETER INITIALIZATION

INITIALIZE WEIGHTS TO IDENTITY MATRIX AND INITIALIZE BIASES TO ZERO. THIS HELPS PREVENT THE WEIGHTS SHRINKING TO ZERO.

$$I_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

③ GATED CELLS

USE GATES TO SELECTIVELY ADD OR REMOVE INFORMATION WITHIN EACH RECURRENT UNIT WITH:



GATES OPTIONALLY LET INFORMATION THROUGH THE CELL.

LSTMs (LONG SHORT TERM MEMORY)

NETWORK RELY ON GATED CELL TO TRACK INFORMATION THROUGH OUT MANY TIME STEPS. GATED LSTM CELLS CONTROL INFORMATION FLOW:

FORGET
FORGET GATE GETS RID OF IRRELEVANT INFORMATION

STORE
STORE RELEVANT INFORMATION FROM CURRENT INPUT

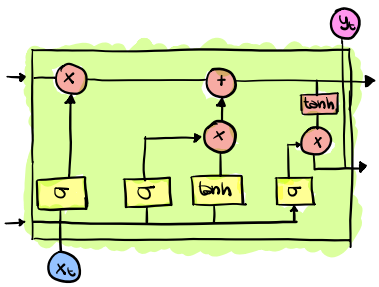
UPDATE
SELECTIVELY UPDATE CELL STATE

OUTPUT
OUTPUT GATE RETURNS A FILTERED VERSION OF THE CELL STATE

THANKS TO THIS GATED STRUCTURE THE BACKPROPAGATION THROUGH TIME ALGORITHM BECOMES MUCH MORE STABLE! THE VANISHING GRADIENT PROBLEM IS MITIGATED BY HAVING FEWER REPEATED MATRIX MULTIPLICATIONS.

↓

BPTT WITH PARTIALLY UNINTERRUPTED GRADIENT FLOW



LSTM CELL ARE ABLE TO TRACK INFORMATION THROUGHOUT MANY TIMESTEPS

LIMITATIONS OF RNNs

① ENCODING BOTTLENECK

WE TAKE A LOT OF CONTENT (E.G. A VERY LONG BODY OF TEXT) AND CONDENSE IT INTO A REPRESENTATION THAT CAN BE PREDICTED ON.

↑

INFORMATION CAN BE LOST IN THIS ENCODING OPERATION

② SLOW, NO PARALLELIZATION

RECURRENT NEURONS AND RECURRENT MODELS ARE NOT EFFICIENT AT ALL.

↓

THEY REQUIRE TO PROCESS INFORMATION SEQUENTIALLY ⇒ THEY ARE VERY INEFFICIENT ON MODERN GPU HARDWARE

③ NOT LONG MEMORY

RECURRENT MODELS DON'T HAVE AN HIGH MEMORY CAPACITY, WHILE THEY CAN HANDLE SEQUENCES IN THE ORDER OF 100 OR 1000 TIME STEPS, THEY DON'T REALLY SCALE WELL TO SEQUENCES THAT ARE OF LENGTH OF THOUSANDS OR TEN THOUSANDS OF TIME STEPS.

THE MAIN LIMITATION OF RNN'S IS THAT THEY PROCESS THOSE TIME STEPS INDIVIDUALLY DUE TO THE RECURRENCE RELATION.

↓

WE WANT TO ELIMINATE THE RECURRENCE OPERATION. WE CAN'T SIMPLY TAKE THE INPUT, SQUASH IT AND PASS IT INTO A DENSE NETWORK BECAUSE:

- IT'S NOT SCALABLE FOR VERY LONG SEQUENCES
- THERE IS NO ORDER
- NO LONG MEMORY
- THERE IS NO NOTION OF WHAT POINT IN OUR SEQUENCE IS IMPORTANT.

ATTENTION IS ALL YOU NEED ← MECHANISM BEHIND TRANSFORMER

IDEA: IDENTIFY AND ATTEND TO WHAT'S IMPORTANT

↓

SELF-ATTENTION → THE ABILITY TO TAKES AN INPUT AND ATTEND TO THE MOST IMPORTANT PARTS OF THAT INPUT: 1) IDENTIFY WHICH PARTS TO ATTEND TO
2) EXTRACT THE FEATURES WITH HIGH ATTENTION

UNDERSTANDING ATTENTION WITH SEARCH

WHEN WE SEARCH SOMETHING ON THE INTERNET, FOR EXAMPLE A VIDEO ON YOUTUBE WE INPUT SOME QUERY (Q) INTO THE SEARCH BAR. FOR EVERY VIDEO IN THE DB WE ARE GOING TO EXTRACT SOME KEY INFORMATION WHICH WE CALL THE KEY (K). THE OVERLAPS BETWEEN THE QUERY AND THE KEYS IN THE DATABASE ARE GOING TO BE COMPUTED, AND AS WE DO THIS, AT EACH CHECK WE MAKE WE'LL ASK HOW SIMILAR IS THAT KEY TO OUR QUERY.

THIS IS THE IDEA OF COMPUTING AN ATTENTION MASK

THE NEXT AND FINAL STEP IS TO EXTRACT THE INFORMATION THAT WE CARE ABOUT BASED ON THIS COMPUTATION, AND WE CALL THIS THE VALUE (V).

LEARNING SELF-ATTENTION WITH NEURAL NETWORKS

WE DON'T WANT TO PROCESS THE INFORMATION TIMESTEP BY TIME STEP ⇒ WE NEED A WAY TO ENCODE THE POSITION INFORMATION TO UNDERSTAND ORDER.

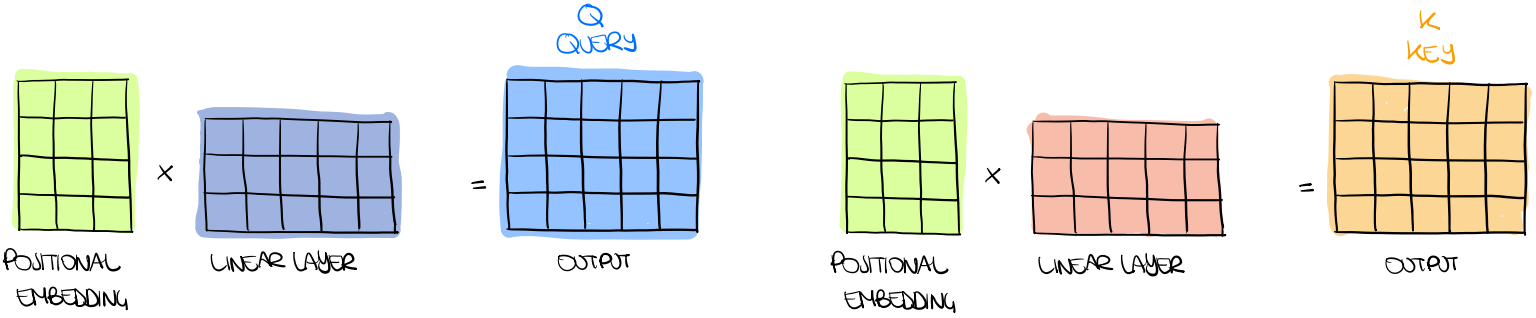
↳ DATA IS FED ALL AT ONCE!

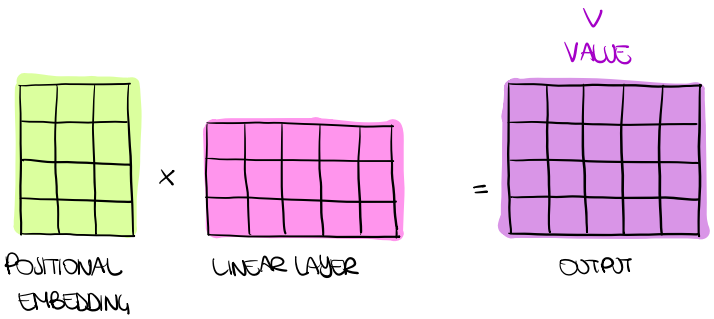
↓

WE USE AN EMBEDDING THAT IS GOING TO INCORPORATE SOME NOTION OF POSITION.

↳ COMPUTE THE WORD EMBEDDING, TAKE SOME METRIC THAT CAPTURES POSITION INFORMATION WITHIN THAT SEQUENCE AND THEN COMBINE THEM TOGETHER.

AFTER WE COMPUTE THE POSITIONAL EMBEDDING WE NEED TO FIGURE OUT HOW TO EXTRACT THE QUERY, KEY AND VALUE.



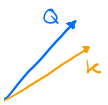


THE NEXT STEP IS TO COMPUTE THE ATTENTION USING THESE THREE MATRICES, FIGURING OUT HOW MUCH ATTENTION TO PAY TO AND WHERE

↳ WE FOCUS ON THE SIMILARITY BETWEEN OUR QUERY AND KEY.

↓

ATTENTION SCORE ← COMPUTE PAIRWISE SIMILARITY BETWEEN EACH QUERY AND VALUE: $\frac{QK^T}{\text{SCALING}}$



$\frac{QK^T}{\text{SCALING}}$ ← SIMILARITY METRIC

ATTENTION WEIGHTING → $\text{SOFTMAX}\left(\frac{QK^T}{\text{SCALING}}\right)$ ← THE SIMILARITY METRIC IS PASSED IN A SOFTMAX FUNCTION THAT SQUASH EVERYTHING BETWEEN 0 AND 1

THE FINAL STEP IS TO USE THE WEIGHTING MATRIX TO EXTRACT FEATURES WITH HIGH ATTENTION:

$$\text{SOFTMAX}\left(\frac{QK^T}{\text{SCALING}}\right)V = A(Q,K,V)$$

WE CAN DO THIS MULTIPLE TIMES ⇒ WE CAN HAVE MULTIPLE INDIVIDUAL ATTENTION HEADS FOR ATTENDING TO DIFFERENT THINGS.