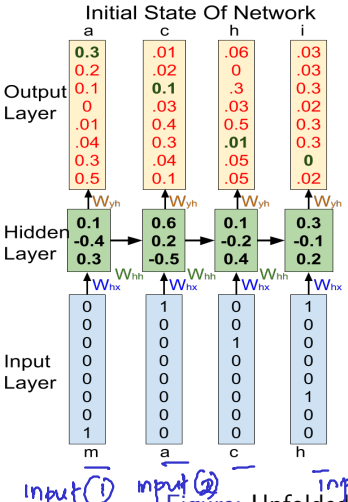
Lecture 23: Recurrent Neural Networks, Long Short Term Memory Networks, Conntectionist Temporal Classification, Decision Trees

Instructor: Prof. Ganesh Ramakrishnan

Recap: The Lego Blocks in Modern Deep Learning

1. Depth/Feature Map
2. Patches/Kernels (provide for spatial interpolations) - **Filter**
3. Strides (enable downsampling)
4. Padding (shrinking across layers)
5. Pooling (More downsampling) - **Filter**
6. **RNN and LSTM** (Backpropagation through time and Memory cell)
7. **Connectionist Temporal Classification**
8. Embeddings (Later, with unsupervised learning)

RNN: Language Model Example with one hidden layer of 3 neurons



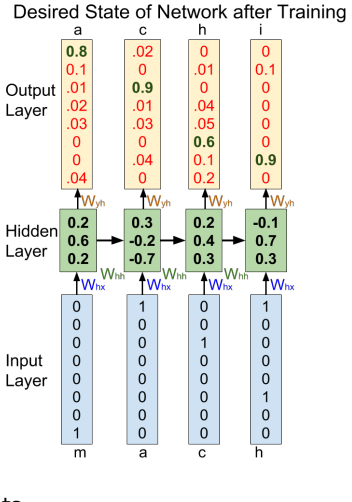


Figure: Unfolded RNN for 4 time units

Vanishing Gradient Problem

The sensitivity(derivative) of network w.r.t input($@t = 1$) decays exponentially with time, as shown in the unfolded (for 7 time steps) RNN below. Darker the shade, higher is the sensitivity w.r.t to x_1.

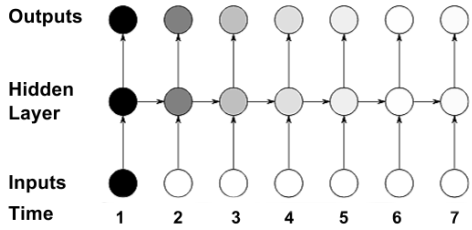


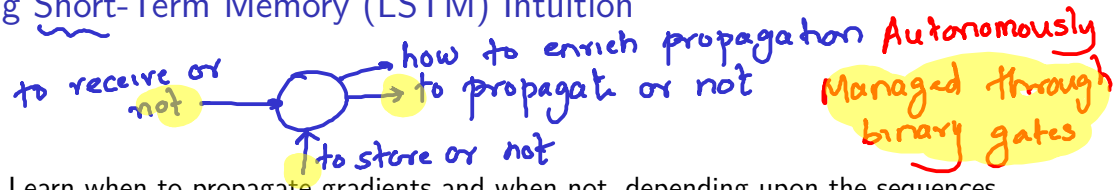
Image reference: Alex Graves 2012.

Q1: Is such ineffective update cost warranted

Q2: More meaningful long term dependence?

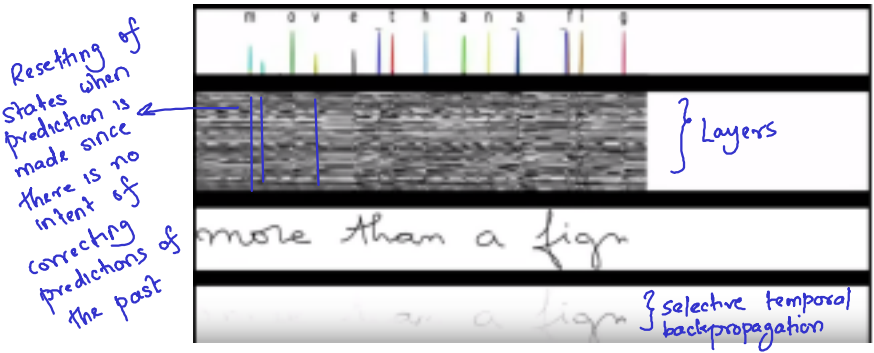
Multiplicative effect ← self loop → vanishing

Long Short-Term Memory (LSTM) Intuition



- Learn when to propagate gradients and when not, depending upon the sequences.
- Use the memory cells to store information and reveal it whenever needed.
- I live in **India**.... I visit **Mumbai** regularly. I speak Marathi (most probable)
- For example: Remember the context "India", as it is generally related to many other things like language, region etc. and forget it when the words like "Hindi", "Mumbai" or End of Line/Paragraph appear or get predicted.

Demonstration of Alex Graves's system working on pen coordinates



1. Top: Characters as recognized, without output delayed but never revised.
2. Second: States in a subset of the memory cells, that get reset when character recognized.
3. Third: Actual writing (input is x and y coordinates of pen-tip and up/down location).
4. Fourth: Gradient backpropagated all the way to the xy locations. Notice which bits of the input are affecting the probability that it's that character (how decisions depend on past).

LSTM Equations

- $f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)$
- $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)$
- We learn the forgetting (f_t) of previous cell state and insertion (i_t) of present input depending on the present input, previous cell state(s) and hidden state(s).

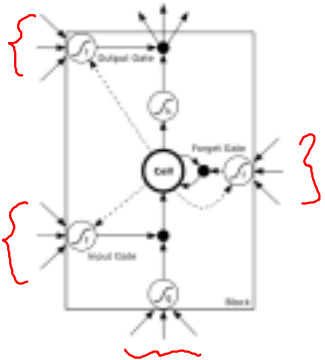


Image reference: Alex Graves 2012.

LSTM Equations

- $c_t = f_t c_{t-1} + i_t tanh(W_{hc} h_{t-1} + W_{xc} x_t + b_c)$
- The new cell state c_t is decided according to the firing of f_t and i_t.

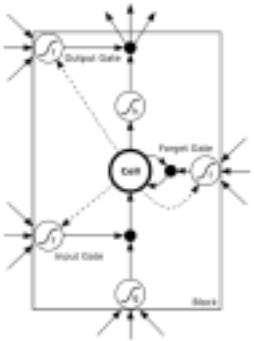
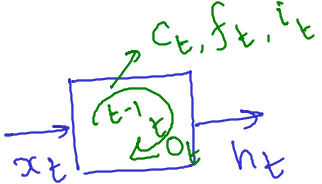


Image reference: Alex Graves 2012.

LSTM Equations

- $c_t = f_t c_{t-1} + i_t tanh(W_{hc} h_{t-1} + W_{xc} x_t + b_c)$
- $f_t = \sigma(W_{xf} x_t + W_{hf} h_{t-1} + W_{cf} c_{t-1} + b_f)$
- $i_t = \sigma(W_{xi} x_t + W_{hi} h_{t-1} + W_{ci} c_{t-1} + b_i)$
- Each gate is a vector of cells; keep the constraint of W_{c*} being diagonal so that each element of LSTM unit acts independently.
- $o_t = \sigma(W_{xo} x_t + W_{ho} h_{t-1} + W_{co} c_{t-1} + b_f)$
- $h_t = o_t \ tanh(c_t)$



LSTM Gradient Information remain preserved

The opening 'O' or closing '-' of input, forget and output gates are shown below, to the left and above the hidden layer respectively.

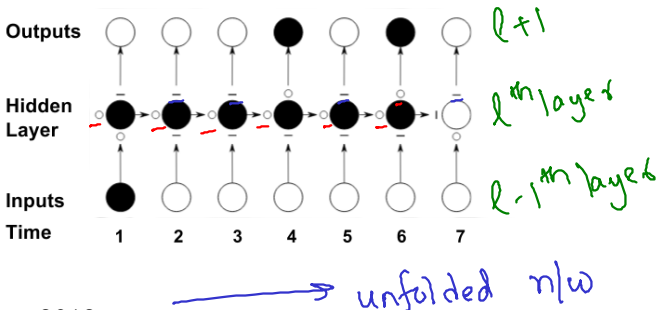


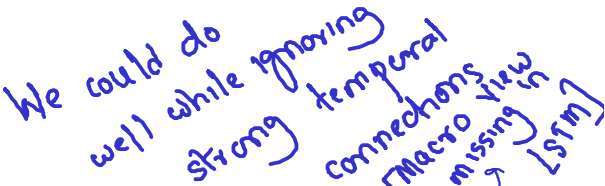
Image reference: Alex Graves 2012.

LSTM V/S RNN results on Novel writing

A RNN and a LSTM, when trained appropriately with a Shakespeare Novel write the following output (for few time steps) upon random initialization.

RNN's Novel	LSTM's Novel
kentages, whom falled me, Compassion, that the wind of England toposed To grief all go. AUFIDIUS: Marry, it were any graven, And high againss. PAULINA: Notus my plateries, this are the sitty, but continue stillen sonsable Tybalt. GREMIO: God sues we charpet. Since this is shoulders to the king. KING EDWARD IV: Look, myself. Sir Juige: ...	Monsum and Parustage, is no mews Desiraud is a death barshers: Her much. LEONTES: JULIET: O Wamswhat that she were a pituhpy, that nothing. Second Lord: They are of my foul me: power and your ocibing. Lore and the glatishal indeed, stamp: hell, if procing dages. LADY GREY: No noter them belipsew the burges That Enrilth these out with her good ...

Sequence Labeling



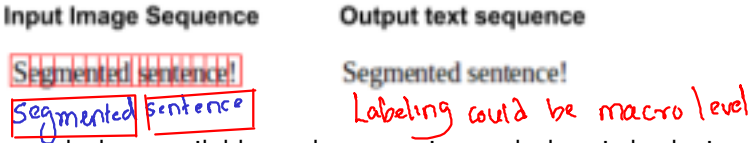
- The task of labeling the sequence with discrete labels. Examples: Speech recognition, handwriting recognition, part of speech tagging.
- Humans while reading/hearing make use of context much more than individual components. For example:- Yoa can undenstard dis, tough itz an eroneous text.
- The sound or image of individual characters may appear similar and may cause confusion to the network, if the proper context is unknown. For example: "in" and "m" may look similar whereas "dis" and "this" may sound similar.

Type of Sequence Labeling Tasks

- Sequence Classification: Label sequence is constrained to be of unit length.

Type of Sequence Labeling Tasks

- Segment Classification: Target sequence consist of multiple labels and the segment locations of the input is known in advance, e.g. the timing where each character ends and another character starts is known in a speech signal.



- We generally do not have such data available, and segmenting such data is both tiresome and erroneous.

Type of Sequence Labeling Tasks

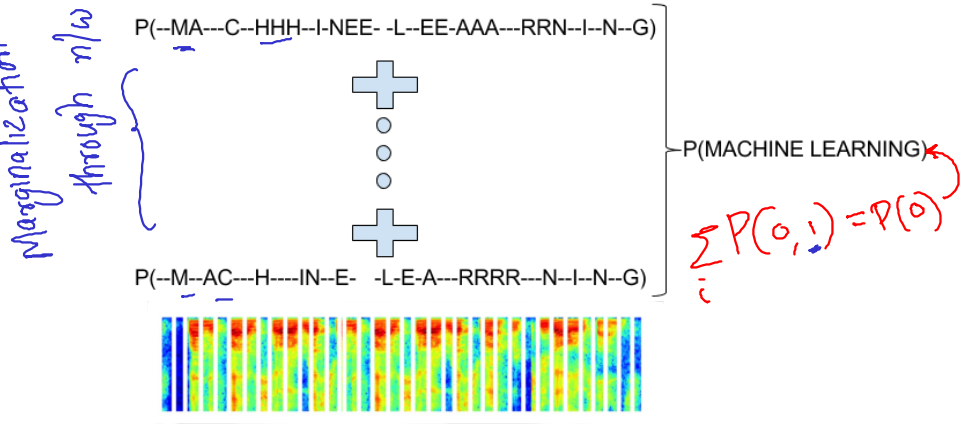
- Temporal Classification: Tasks in which temporal location of each label in the input image/signal does not matter.

- Very useful, as generally we have higher level labeling available for training, e.g. word images and the corresponding strings, or it is much easier to automate the process of segmenting the word images from a line, than to segment the character images from a word.

Connectionist Temporal Classification (CTC) Layer

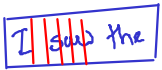
- For temporal classification task: length of label sequence $<$ length of input sequence.
- CTC label predictions at any time in input sequence.
- Predict an output at every time instance, and then decode the output using probabilities we get at output layer in vector form.
- e.g. If we get output as sequence "–m–aa-ccch-i-nee– -lle–a-rr-n-iinnn-g", we will decode it to "machine learning".
- While training we may encode "machine learning" to"-m-a-c-h-i-n-e- -l-e-a-r-n-i-n-g-" via C.T.C. Layer.

CTC Intuition [Optional]



CTC Intuition [Optional]

NN function : $f(x_T) = y_T$



- x_T: input image/signal x of length T.
  - For image: each element of x_T is a column(or its feature) of the image.
- y_T: output sequence y of length T.
  - each element of y_T is a vector of length |A'|(where A' = A∪"-" i.e. alphabet set ∪ blank label).

ℓ_U : Label of length U(<T).

Intuition behind CTC:

- generate a PDF at every time-step t ∈ 1,2,...,T.
- Train NN with objective function that forces Max. Likelihood to decode x_T to ℓ_U(desired label).

CTC Layer: PDF [Optional]

$P(\pi|x) = \prod_{t=1}^{T} y_t(\pi_t)$

- path π : a possible string sequence of length T, that we expect to lead to ℓ. For example: "-p-a-t-h-", if ℓ = "path".
- y_i(n): probability assigned by NN when character n($\in$ A') is seen at time i. "-" is symbol for blank label.
- π_t : t^{th} element of path π.

$P(\ell|x) = \sum_{label(\pi)=\ell} P(\pi|x)$

CTC Layer: PDF [Optional]

$P(\ell|x) = \sum_{label(\pi)=\ell} P(\pi|x) = \sum_{label(\pi)=\ell} \prod_{t=1}^{T} y_t(\pi_t)$

- Question: What could be possible paths of length T = 9 that lead to ℓ = "path"?
- Answer:
- Question: How do we take care of cases like ℓ = "Mongoose"?
- Answer:

CTC Layer: PDF [Optional]

$P(\ell|x) = \sum_{label(\pi)=\ell} P(\pi|x) = \sum_{label(\pi)=\ell} \prod_{t=1}^{T} y_t(\pi_t)$

- Question: What could be possible paths of length T = 9 that lead to ℓ = "path"?
- Answer: "-p-a-t-h-", "pp-a-t-h-", "-paa-t-h-", "-ppa-t-h-", "-p-aat-h-" etc.
- Question: How do we take care of cases like ℓ = "Mongoose"?
- Answer: We change ℓ = "Mongoose" to ℓ = "Mongo-ose".
- Question: During training ℓ is known, what to do at testing stage?
- Answer: Next Slide.

CTC Layer: Forward Pass Decoding [Optional]

$P(\ell|x) = \sum_{label(\pi)=\ell} P(\pi|x) = \sum_{label(\pi)=\ell} \prod_{t=1}^{T} y_t(\pi_t)$

Question: During training ℓ is known, what to do at testing stage?

1. Brute force: try all possible ℓ's, all possible π's for each ℓ to get $P(\ell|x)$ and choose best ℓ.
   - Rejected as expensive.
2. Best Path Decoding - most likely path corresponds to the most likely label.
   - $P(A1) = 0.1$, where A1 is the only path corresponding to label A.
   - $P(B1) = P(B2) = ... = P(B10) = 0.05$, where B1..B10 are the 10 paths corresponding to label B.
   - Clearly B is preferable over A as $P(B|x) = 0.5$.
   - But Best Path Decoding will select A.

   - Rejected as inaccurate.
3. Prefix Search Decoding - NEXT

CTC Layer: Prefix Search Decoding [Optional]

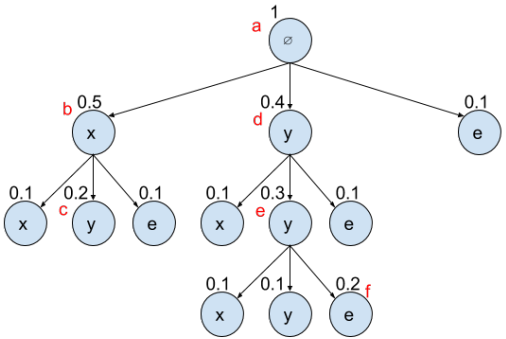
1. Initialize prefix p* = ϕ; //p* is prefix of *ell*
2. P(ϕ) = 1; // as ϕ is prefix of every word.
3. try p_{new} = p* + k, for all k $\in$ A$\cup${eos}; // + represent concatenation.
4. Maintain L_p: list of growing prefixes; // |A| + 1 new values per iteration.
5. Maintain P_p: list of probabilities of corresponding elements in Lp; //How to find P(p*+ k)? Next Slide.
6. if P(p*+eos) >= max(P_p): stop and go to step 8;
7. else: update p* with prefix having max prob. and repeat from step 3;
8. p* is the required prefix.

In practice beam-search is used to limit the exponentially growing Lp and make the decoding faster.

CTC Layer: Prefix Search Decoding

Consider the DAG shown below with A = {x,y} and e representing the end of string. The steps a-f represent the path followed by Prefix Search Decoding Algorithm.

- What ℓ would the Best Path Decoding Produce?
- What ℓ would the Prefix Search Decoding Produce?



CTC Layer: Extending Prefix Probabilities [Optional]

P(p|x)= $Y_t(p_n) + Y_t(p_b)$

- Y_t(p): probability that prefix p(of ℓ) is seen at time t.
- $Y_t(p_n)$: prob. that p seen at t and last seen output is non-blank.
- $Y_t(p_b)$: prob. that p seen at t and last seen output is blank.

Initial Conditions :

- $Y_t(\phi_n) = 0$.
- $Y_t(\phi_b) = \prod_{i=1}^{t} y_i$(b).

Extended Probabilities: Consider, initial p=ϕ, ℓ^* : growing output labeling, p* : current prefix, and p' = p* + k; k A$\cup${eos}.

- $Y_1(p'_b) = 0$(as k $\in$ A$\cup${eos}, and A excludes blank)
- $Y_1(p'_n)$ = y1(k) (as p' ends with k)

CTC Layer: Extending Prefix Probabilities [Optional]

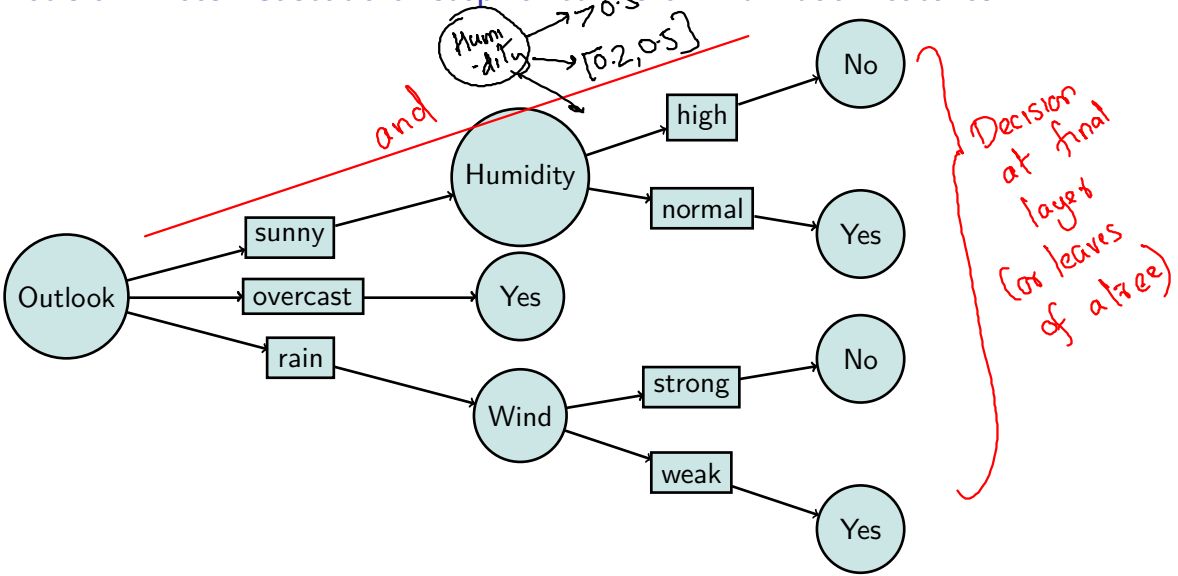
P_{new}(t) : Prob. to see a new character k at time t.

P_{new}(t) =
$Y_{t-1}(p*_b)$, if p* ends with k.
$Y_{t-1}(p*_b) + Y_{t-1}(p*_n)$, otherwise.
Thus:

- $Y_t(p'_n) = y_t$(k)((P_{new}(t)+Y_{t-1}(p'n)))
- $Y_t(p'_b) = y_t$(b)(y_{t-1}(p'b)+y_{t-1}(p'n)))

Other (Non-linear) Classifiers: Decision Trees and Support Vector Classification

Decision Trees: Cascade of step functions on individual features



Use cases for Decision Tree Learning

(1) When features already known and you want a small/compact model on top that is interpretable & sparse

Does branching on features

(2) Implicit feature selection

(3) Recursively partitioning of data

The Canonical Playtennis Dataset

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision tree representation

H/w: Try learning these trees

Principle: Recursive "pure/precise" partitioning

- Each internal node tests an attribute
- Each branch corresponds to attribute value
- Each leaf node assigns a classification

How would we represent:

- $\wedge, \vee$, XOR
- $(A \wedge B) \vee (C \wedge \neg D \wedge E)$
- M of N → $(X_1, X_2 \ldots X_N)$: o/p 1 if M of these are 1 & 0 o/w.