If We have to convert DNN to CNN -

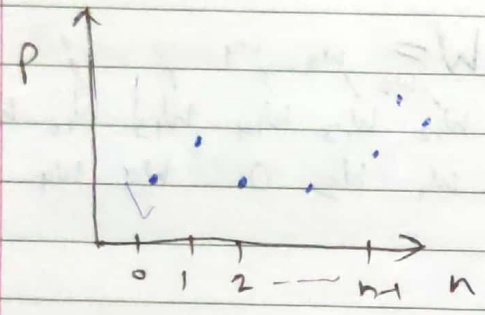
size of filter = I/p size


filter size  Output

# Softmax Activation fn

- for 2 classes, sigmoid is used. Softmax is used for multi-class classification.

- Softmax uses one-hot vector to give probability distribution




→ This is what we're looking for

$$\begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

1 is put where probab is max. Here, $P_3$ has the highest probab.

Probab. distri. is converted to One-Hot Vectors.

$$z^{[l]} = w^{[l]} A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g(z^{[l]})$$

→ ReLU is used most of the time

→ At last layer, sigmoid is used

$z^{[l]} \rightarrow (n^c \times 1)$  $n^c$: # classes.

Let $n^{[l]} = 3$,

$$z^{[l]} = \begin{bmatrix} z_1^{[l]} \\ z_2^{[l]} \\ z_3^{[l]} \end{bmatrix}, \quad e^{z^{[l]}} = \begin{bmatrix} e^{z_1^{[l]}} \\ e^{z_2^{[l]}} \\ e^{z_3^{[l]}} \end{bmatrix}$$

$$a^{[l]} = \begin{bmatrix} a_1^{[l]} \\ a_2^{[l]} \\ a_3^{[l]} \end{bmatrix} \qquad a_1^{[l]} = \frac{e^{z_1^{[l]}}}{\sum_{i=1}^{3} e^{z_i^{[l]}}}, \quad a_3^{[l]} = \frac{e^{z_3^{[l]}}}{\sum_{i=1}^{3} e^{z_i^{[l]}}}$$

* Imagenet - 1 million images with 1000 classes.

* We're calculating probab. in order to back propagate. Else we could have just find the $z_i^{[l]}$ which is max. & declare it as ans. inside one hot vector

I/p → 7 x 7 x (512) → # repr of an image

Since, we're convolving so many times, we get diff. repr of base image.

- With DNN, we get more # rich features for model to perform well

How powerful are DNNs as compared to CNNs?

↓

has more no. of parameters (& don't take overfitting into account)

?

CNNs require [order] in images but DNNs don't

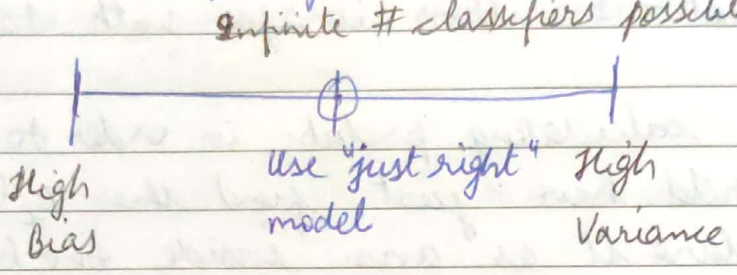
As DNNs are prone to overfitting, they perform poorly.

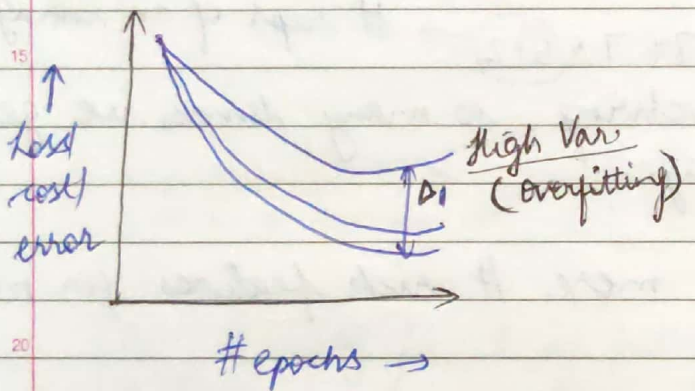
Date
15/10/19

# Datasets

- All 3 datasets - train / dev / test one from same distribution
- Earlier we didn't have too many hyperparameters so no need as such to have validation set.

**High bias** - Underfitting
**High Variance** - Overfitting

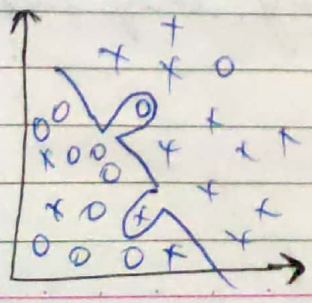



**Underfitting** - model performs poorly on both train & test set.
(High Bias)
eg- 15% on train set
16% on test set

(High Bias & High Variance)
eg- 15% on train set
30% on test set
[Performs bad on train set but even worse on test set

(High Variance) eg- 1% on train set
15% on test set



## Regularization

$$J(y, \hat{y}) = \underbrace{\left[-\frac{1}{m}\sum_{i=1}^{m}\left(y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right)\right]}_{} + \underbrace{\frac{\lambda}{2m}\|w\|^2}_{}$$

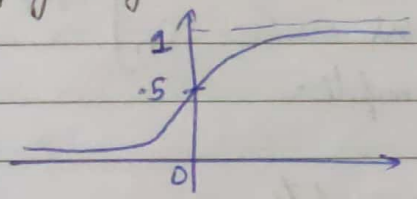
$$\frac{\lambda}{2m}\sum_{j=1}^{p} w_j^2 \quad (p \Rightarrow \#\text{parameters})$$

sq. of all parameters in our model

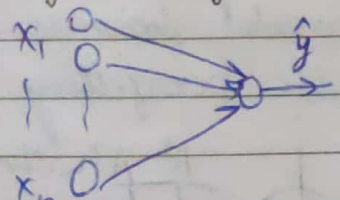
Regularization term

The model ultimately tries to min. cost fn. Now, addition of sq. of parameters will penalize if para. take very large values. (as goal is to min J).

Apply Regularization to avoid overfitting




## Logistic Regression




$$J(w, b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}_i, y_i) + \frac{\lambda}{2m}\sum_{i=1}^{n} w_i^2 + \frac{\lambda}{2m} b^2 \to 0$$

Ignore as b is very small as compared to w

**$L_2$ regularization**: $\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

$$w_i = w_i - \alpha \frac{\partial (J + J')}{\partial w} = w_i - \alpha \frac{\partial J}{\partial w} - \alpha \frac{\partial J'}{\partial w}$$

$$w_i = w_i - \alpha \frac{\partial J}{\partial w} - \alpha \frac{\lambda}{2m}(2w_i)$$

$\alpha, \lambda > 0$, $m > 0$

$$\boxed{w_i = \left(1 - \frac{\alpha\lambda}{m}\right) w_i - \alpha \frac{\partial J}{\partial w_i}}$$

$\Rightarrow$ We're reducing w value even further.

}} [Reg may contribute to overcome exploding grad. prblm]

Camlin | Page
Date / /

($\lambda$): Regularization parameter

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}_i, y_i) + \boxed{\frac{\lambda}{2m}\sum_{l=1}^{L}\sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}} \left(w_{ij}^{[l]}\right)^2}$$

[Very high bias - Regulari doesn't solve this.

[Weight-decay]

[Reg turns complex classifier to simple classifier

$[z = w_0 + w_1 x + w_2 x^2 + \cdots + w_n x^n]$ close to

Make non-linear to linear class

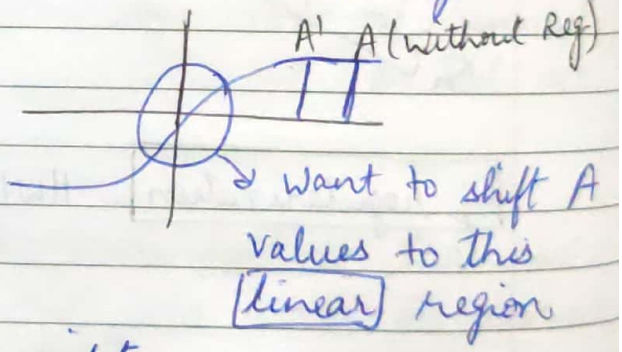
Regularization reduces $[w_0, w_1 \cdots w_n]$
to make classifier simpler

Regularization helps to decay weights.

(non-linear)

[The more layers we add, our model becomes more complex & becomes more prone to overfitting.

- By reducing wts., we shift values o/p by act. f$^n$ to linear region.
(less complex)




- Reg only penalizes high weights to reduce cost function.

Q: How Reg deals with $[10, 0.01, 0.01, \cdots 0.01]$?
It tries to redistribute these weights so that all of them contribute almost equally.
& value of $\left(\frac{\lambda}{2m}\sum w_i^2\right)$ is reduced

Effect of Reg: Tend to spread out weights. Doesn't make them zero.



* L1 Regularization $= \frac{\lambda}{2m} \sum_{l=1}^{L} \sum_{i=1} \sum_{j=1} |w_{ij}^{[l]}|$ → may get sparse matrix (many-zeros)

[While L2 will give very low W values but not sparse.]

# Dropout - Drop some of the neurons (keep_prob = 0.5) randomly. makes n/w simpler

* [While training, weights are distributed uniformly using drop out]
* [But at test time, we don't use dropout. As we should use all trained neurons in the n/w to contribute to the actual prediction.]

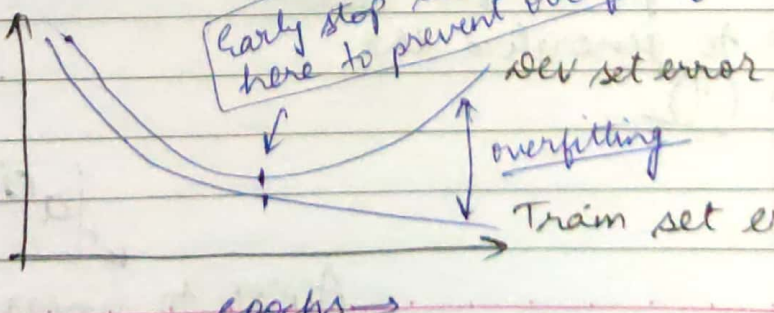
* More data means less overfitting possible But still we can design very very huge NN that will overfit. Then we use [dropout] instead of using smaller NN.

* [Dropout is strongest form of Regularization that helps to curb over-fitting. (Has same effect as L2-Reg.)]

# Other regularization techniques

(1) Data Augmentation - Slight rotations
- Zooming

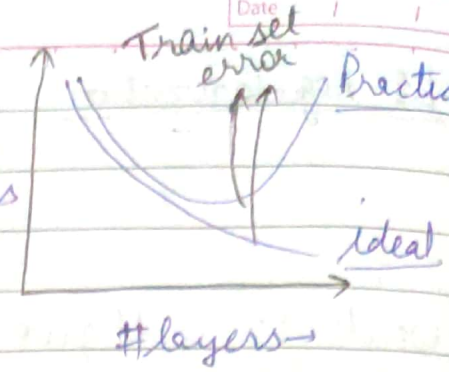
(2) Early stopping



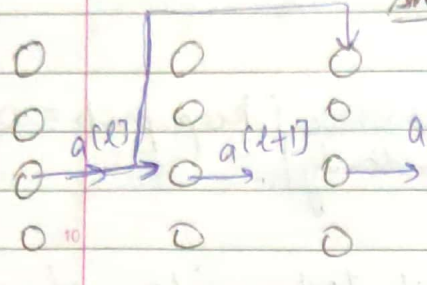
Date
22/10/19

# ResNets

## 152 layer architecture




O/p of $l^{th}$ layer is directly fed to $(l+2)^{th}$ layer before applying ReLV.

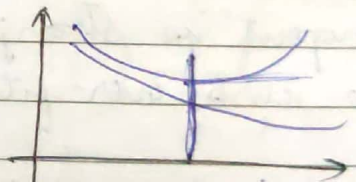



- Very diff for optimizer to optimize weights (parameters) as our model becomes complex
- Learning becomes difficult

$$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]}.$$
$$a^{[l+1]} = g(z^{[l+1]})$$
$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}.$$
$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]}).$$
$$= g(W^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]}).$$

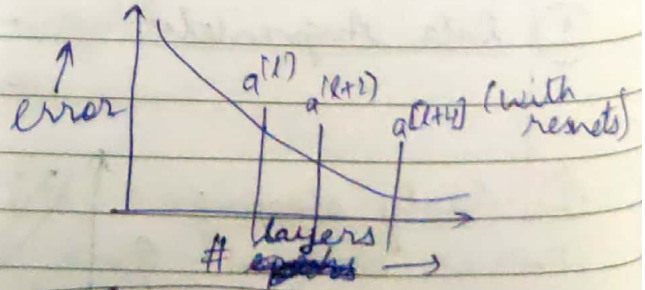
$a^{[l+2]} \approx a^{[l]}$

- ResNets help us overcome the above problm in plot.
- It tries to make model learn at least identity $f^{n}$ so that at least our accuracy is maintained instead of decreasing.



Using regularization, we try to decay weights & biases so that output of $a^{[l+2]} \approx a^{[l]}$.

We add $a^{[l]}$ just before ReLV inorder to generalize the model. (?)




$a^{[l+2]} \approx a^{[l]}$ so err reduces on increasing # layers

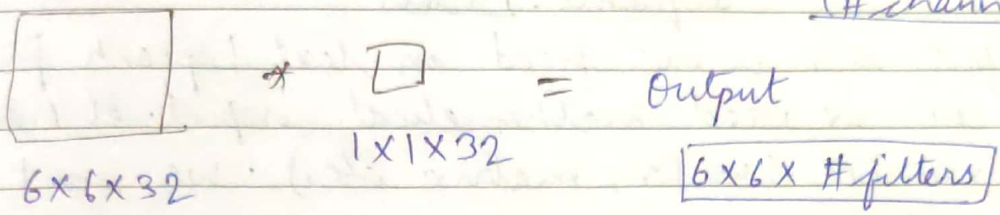
Approx. to linear identity $f^{n}$.

## Backprop in ResNets -?

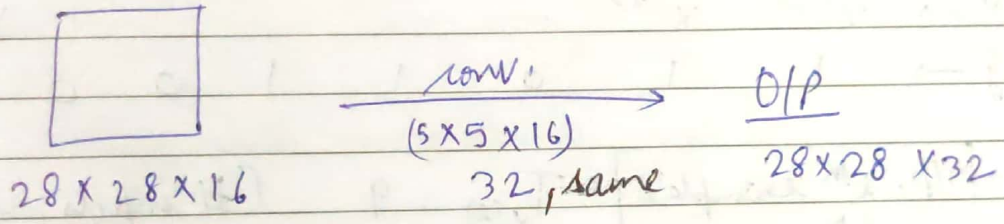
- Helps us to overcome little bit of vanishing / exploding gradient prblm as gradients are also backpropagated from $n^{[l+2]}$ to $n^{[l]}$.

# Why does a 1×1 convolution do?

(1×1) filter helps reducing the depth of inputs. (# channels)




We can reduce depth from 32 to (# filters).



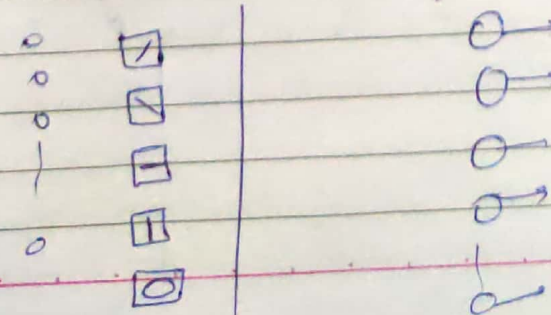

# multiplication operations = (28×28×32) × (5×5×16)

(28×28×32): total o/p nos.; (5×5×16): computation done for each o/p no.

≈ 10.0 M

# In Inception n/w, we've softmax Act. Layers that have regularization effect (dropout) as we're capable of getting o/p in s/w n/w as well (dropout effect).

### Transfer Learning



In case we've less data, we use transfer learning by freezing the weights of pretrained model.

We normalize inputs (/255) so that large pixel values (that only repr color intensities) don't dominate

## Sequence models

- Input size is not fixed in text/speech processing
- For DL, we need mathematical input. It has to be a number (vector, matrix, etc.). We can't input text here.

eg: Harry Potter and Hermaine Granger invented a new spe

$x \rightarrow x^{<1>}\ x^{<2>}\ \ldots\ x^{<n>}$

$\hat{y} \rightarrow 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0$

$x^{(i)<t>}$ / $y^{(i)<t>}$ → [ o/p, $i^{th}$ example's $t^{th}$ element ]

$T_{x^{(i)}} = 9$ (# inputs, words)

$T_y^{(i)}$

$y^{(i)<t>}$ → output of $t^{th}$ element of $i^{th}$ example.

# Representing words :-

[ ] 10000 × 1

9999 → 0's
1 → 1's

One-hot encoding

* Disadv
- Highly sparse.
- When our vocabulary changes, our representation changes.

CS230 - Stanford slides.

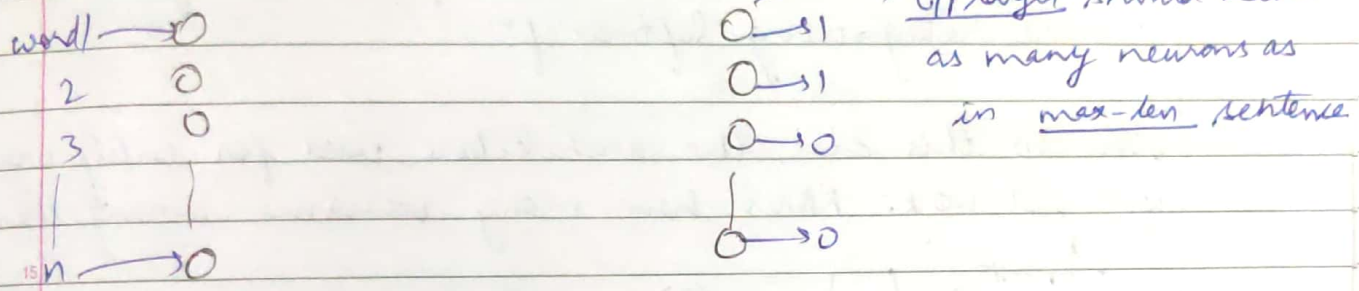
Named Entity Problem

Q. Stmt - "Hermoine gives a new spell."

How to input it to a NN & find that whether a word is name or not?

- CNN shares features & parameters. So, they don't overfit as much that of DNN.

→ Use one-hot vector after constructing a corpus.

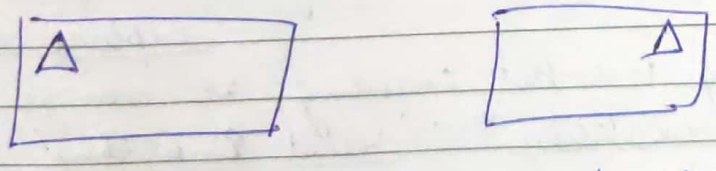



Sentence with n words as I/P.

* What if we've variable input length sentences?
We will 0-pad for rest of the words.

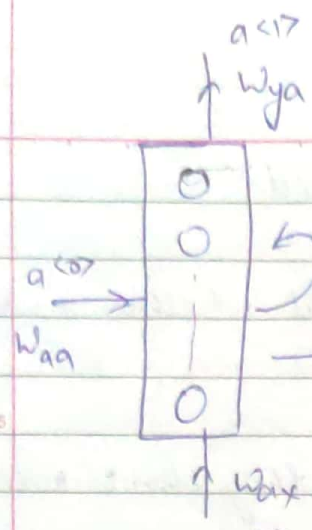
Take 1 sent with max. words & take that many neurons.
0-pad everyone else (not a very good approach).

eg.



These 2 imgs if we input to DNN will not be able to classify but will do with CNN.
What we want is to learn to recognise the data irrespective of where it's placed.

We need same thing in RNN. If Harry Potter appears somewhere in corpus (our sentence). Same parameters should be able to learn to recognize the name.



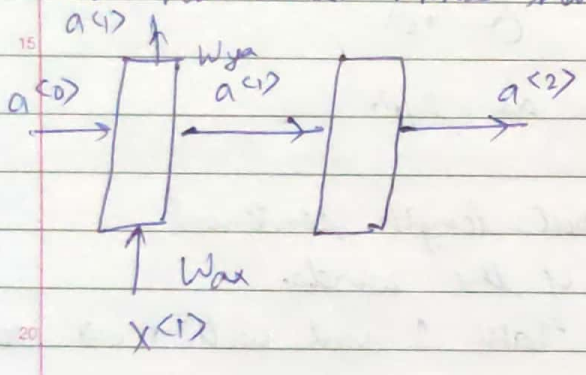

He said Teddy Roosevelt Name✓ (Teddy)

He said Teddy Bears. Name ✗ (Teddy)

$W_{xy}$ → I/p is of type x & output is of kind y.

Q) How to tell whether a sent. is positive or negative. Like a movie rating → using Softmax f$^n$

Now in this case, the architecture used for entity words will not work. RNNs have many variance among them.




This archi is Turing complete, but if you do not pass $a^{<1>}$ & pass $y^{<1>}$, it's not turing complete.

## NLP & Word Embeddings

Bigger problem of one-hot Encoding than sparse — don't capture semantics.

Instead of using one-hot encoding, we can go with "Featurized representation" : 'Word Embedding'.

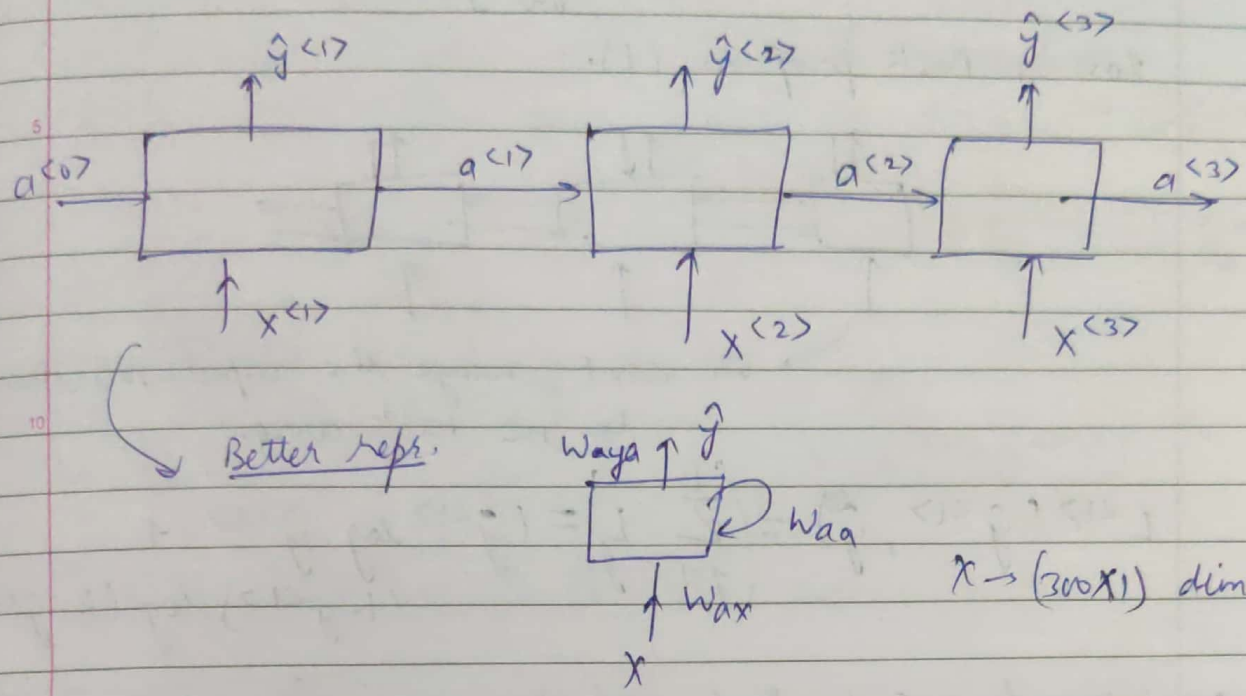
Man, Woman — King, Queen

Gender
Age
Food

(denoting the correlation)

Now each word is just repr by a vector of size 300. Dot prod. will not be 0, it will be some value denoting correlation.

Problem of sparsity as well as semantics both are solved if we use 'Featurized representation'.




$W_{xa}$: Outputs a & inputs X.

Equations: $z^{<t>} = b_a + W_{ax} x^{<t>} + W_{aa} a^{<t-1>}$

$a^{<t>} = g_1(z^{<t>})$ ← tanh / ReLU

$z'^{<t>} = W_{ya} a^{<t>} + b_y$

$\hat{y}^{<t>} = g_2(z'^{<t>})$ ← sigmoid

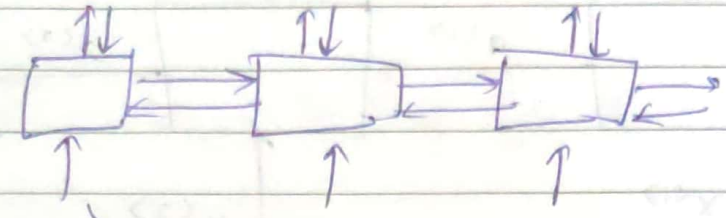
- RNN's have an equivalent power to that of a Turing Machine.
- Tanh can also be used unlike CNN that only uses ReLU.

Q. What problem is with $W_{ya}^{<2>}$ → No use of parameter sharing then, for some words, we want features to be shared instead of another parameter.

* Back Propagation

$$W = W - \alpha \frac{\partial L}{\partial W}$$

Loss in Back prop is (L).

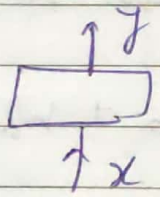


→ We can't change the input, so there can be no back arrow

$$L^{<t>}(\hat{y}^{<t>}, y^{<t>}) = \left(\sum_{j=1}^{t} L_j\right) = (y^{<t>} \log \hat{y}^{<t>} + (1-y^{<t>}) \log (1-\hat{y}^{<t>}))$$

• Many-to-one practical application → Sentiment Analysis

One-to-one



→ DNN's

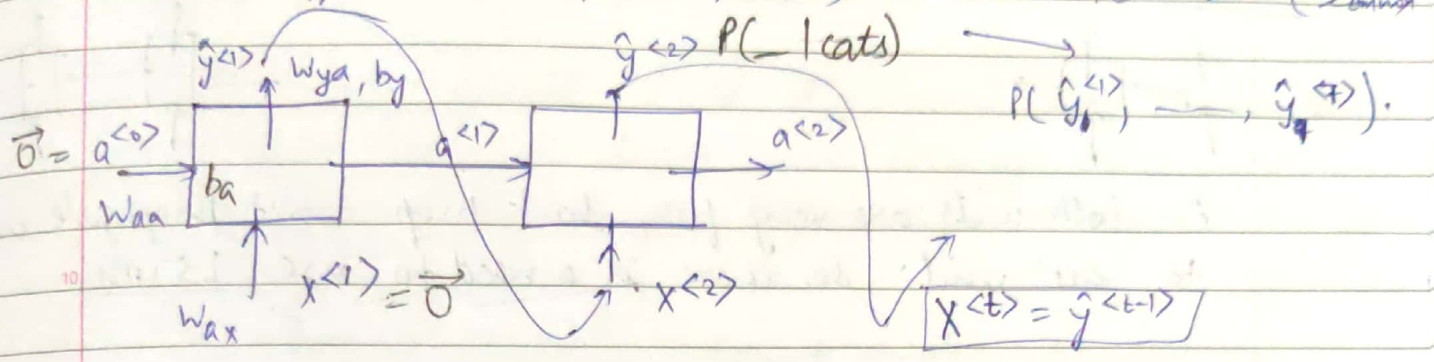
(No timestamp means whatever we did in DNNs)

Date 14/11/19

* **Language Modelling** - What is the probab. associated with the sentence or word?

P(The apple & the pair salad) $\approx 3.3 \times 10^{-13}$

P(The apple & the pear salad) $\approx 5.7 \times 10^{-10}$ ✓ (More common)




$P(\hat{y}_0^{<1>}, \ldots, \hat{y}_4^{<7>})$.

$a^{<0>}$ is initialised to $\vec{0}$ (No input)

$x^{<7>}$ = Most frequently occurring words is not a good idea as all our generated sent. by RNNs will start with 'The'.

$$a^{<t>} = g_1(W_{ax} x^{<t>} + W_{aa} a^{<t-1>} + b_a)$$

($g_1$ → tanh)

$$\hat{y}^{<t>} = g_2(W_{ya} a^{<t>} + b_y).$$

($g_2$ → softmax)

Idea:- We select the 1st word, based on the probab of words in the corpus. Eg- P(the) = 0.07, there are 7% chances of choosing 'the' out of all possible words/choices

P(cats) = 0.0004

$P(\hat{y}^{<1>} \mid \text{cats})$

$P(\hat{y}^{<3>} \mid \text{cats average})$

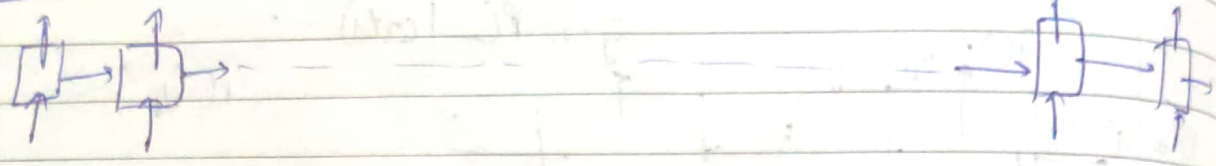
- <EOS> : End of sent. marker
- To deal with <UNK>, we can even have a model at character level as no chr. (ASCII) can be unknown.

# LSTMs

RNNs suffer from Vanishing Gr. prblm much more than DNNs or CNNs.

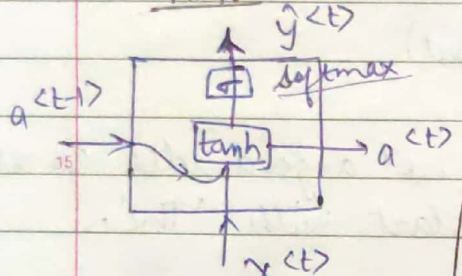
The cat, which already ate fruits like apple, pear, etc., was full

RNNs

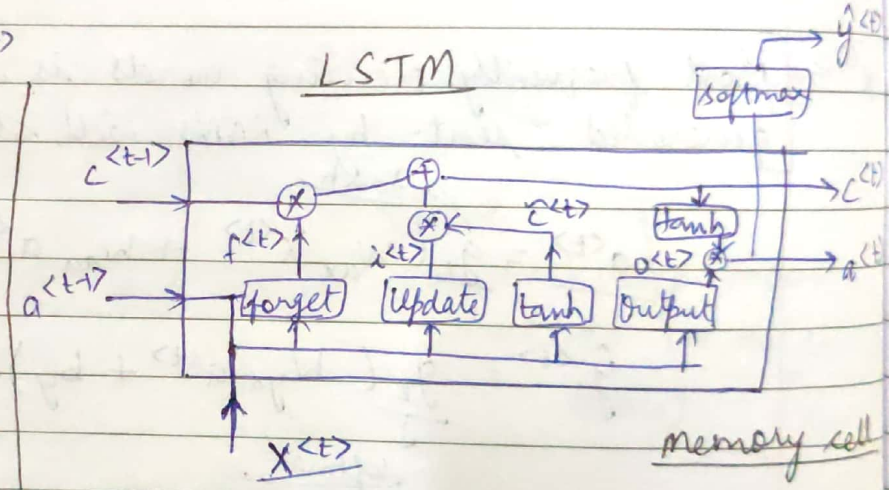


As both units are very far, back prop. won't propagate err to 'cat' unit. So, there is a need to use LSTMs.

RNN ⟶ Doesn't capture long term dependencies.

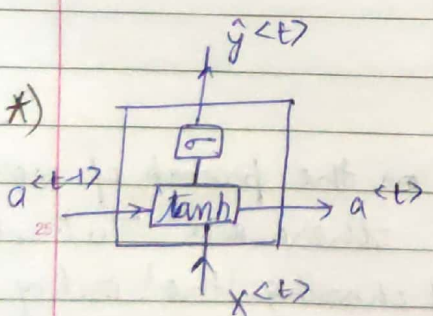



LSTM




memory cell

*)




(forget gate)

$$\Gamma_f = \sigma(W_f [a^{<t-1>}, x^{<t>}] + b_f).$$

$$\Gamma_u = \sigma(W_u [a^{<t-1>}, x^{<t>}] + b_u).$$

$$\Gamma_o = \sigma(W_o [a^{<t-1>}, x^{<t>}] + b_o).$$

$$a^{<t>} = g_1(W_{ax} x^{<t>} + W_{aa} a^{<t-1>} + b_a)$$

$$\hat{y}^{<t>} = g_2(W_{ya} a^{<t>} + b_y).$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

update $c^{<t-1>}$ & change it to $c^{<t>}$.

$$\tilde{c}^{<t>} = \tanh(W_c [a^{<t-1>}, x^{<t>}] + b_c).$$

$$a^{<t>} = \Gamma_o * c^{<t>}.$$

In our terminology, $W_f = [W_{fa}, W_{fx}]$. Composed of 2 wts. (Andrew Ng combined both in 1 slide).

$c^{\langle t \rangle}$: Cell state (for memorizing)
$a^{\langle t \rangle}$: Activation of previous unit

[Before LSTMs, people tried skip connections to maintain long term dep. But there were lots of extra conn. resulting in very high # parameters. Hence, LSTMs were invented.

## GRUs (much simpler than LSTMs).

$$\tilde{c}^{\langle t \rangle} = \tanh(W_c [\Gamma_r * c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c).$$

$$\Gamma_u = \sigma(W_u [c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u).$$

(Relevance gate)

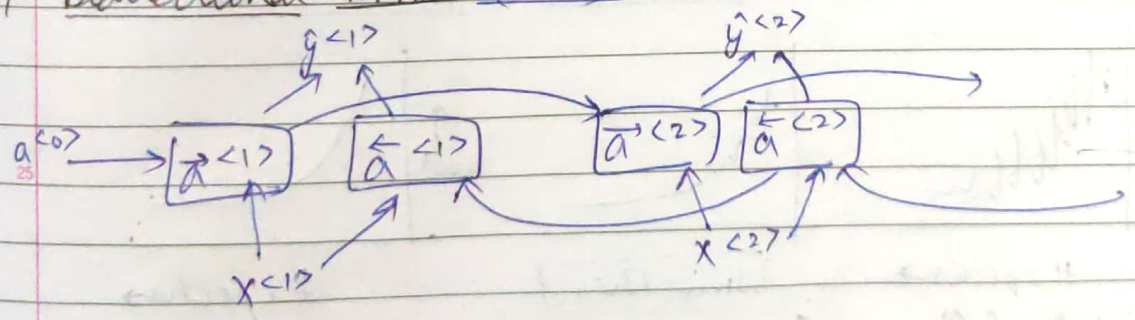
$$\Gamma_r = \sigma(W_r [c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_r).$$

$$c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + (1-\Gamma_u) * c^{\langle t-1 \rangle}.$$

$$a^{\langle t \rangle} = c^{\langle t \rangle}.$$

[has only 2 gates – update and Relevance.

## # Bidirectional RNNs (BRNN)



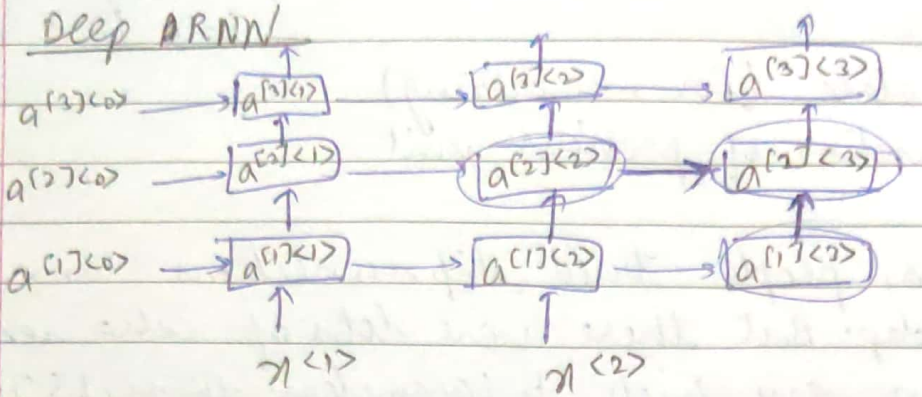

$$\hat{y}^{\langle t \rangle} = g(W_{ya} a^{\langle t \rangle} + b_y) \quad \text{— One dir}$$

$$= g(W_{ya} \overrightarrow{a}^{\langle t \rangle} + W_{ya} \overleftarrow{a}^{\langle t \rangle} + b_y)$$

Fwd Bcwd

$a^{[1]<2>}$ means 1st layer, 2nd unit / timestamp.

* Deep RNN



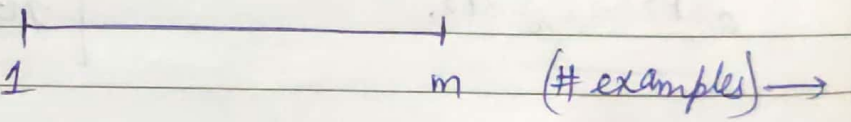

$$a^{[2]<3>} = g\left(W_a^{[2]}\left[a^{[2]<2>}, a^{[1]<3>}\right] + b_a^{[2]}\right).$$

Turing complete archi

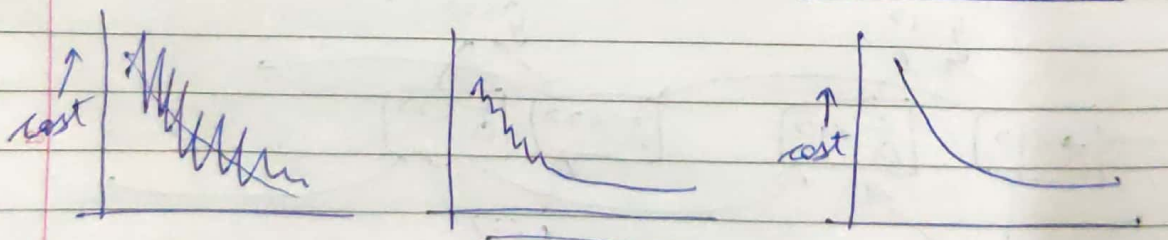
Date 21/11/19

SGD (Mini-batch Gr. Descent)

We can consider following # examples in 1 gr. desc update -




for 1 ex. | 1 < _ < m | for m exs.




(Stochastic GD) | Smoother (Mini-Batch GD) | (Batch GD)

$x^{\{1\}}$ → Mini-batch 1. | $x^{\{2\}}$ → Mini-batch 2.

$x^{(1)}$ → 1st tr. ex. | $x^{(2)}$ → 2nd tr. ex.

Stochastic Gr. Descent → When batch size = 1. We train on just 1 ex. before updating the parameters (exs. choosen ~~randomly~~ epoch-wise)

With Batch-GD, learning is very slow
With SGD, learning is very fast
While Mini-Batch GD, takes good of both the things.

We require:
1) Learning should be moderately fast
2) Should reach to min. as fast as possible.

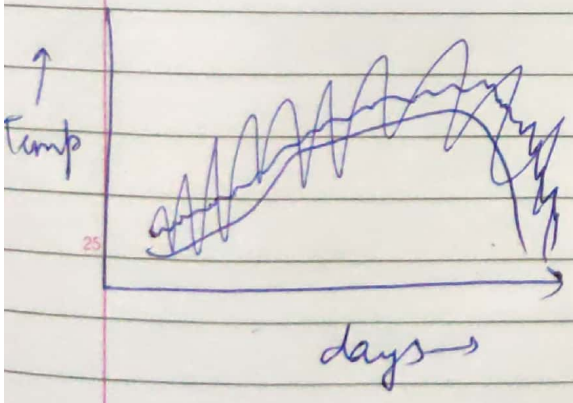
Moving Avg.: Taking avg of obs within a window. Assign higher wts. to nearer obs. & so on

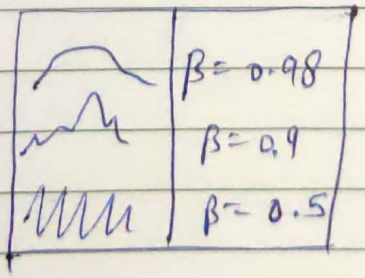
$$\begin{cases} V_0 = 0 \\ V_1 = 0.9 V_0 + 0.1 \theta_1 \\ V_2 = 0.9 V_1 + 0.1 \theta_2 \\ \vdots \\ V_t = 0.9 V_{t-1} + 0.1 \theta_t \end{cases}$$

$$\boxed{V_t = \beta V_{t-1} + (1-\beta)\theta_t}$$ exponentially weighted average

$\beta = 1$, just considering past avg

$\beta = 0$, just considering current value




# Momentum