Recurrent Neural Networks Part 2

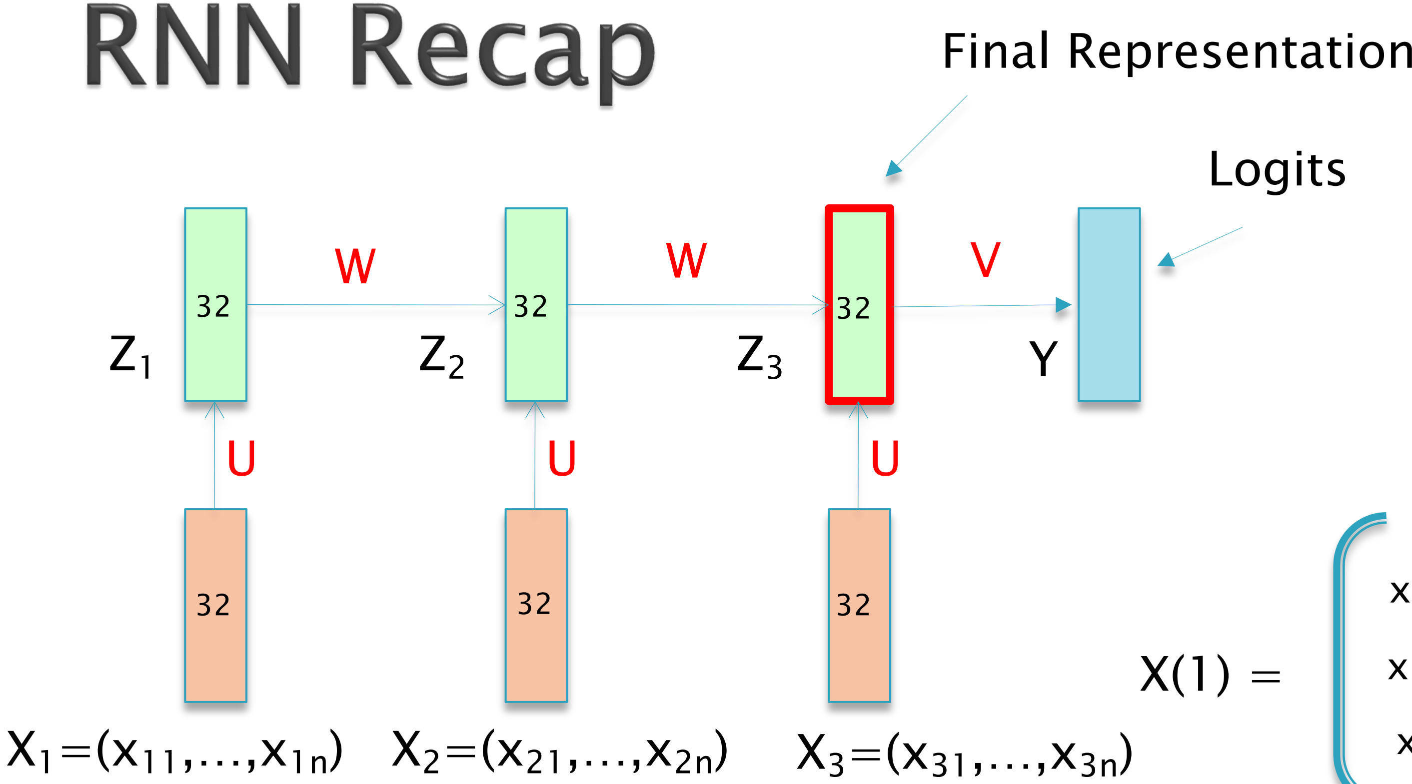
Lecture 14
Subir Varma

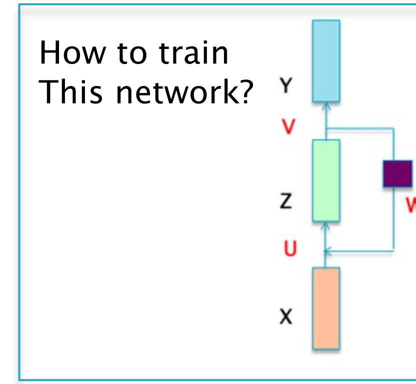
Today's Lecture

- Training RNNs
  - Back Propagation Through Time (BPTT) Algorithm
- Issues with BPTT
  - Vanishing and Exploding Gradients
- Solution: LSTMs
- 1D Convolutions

BackPropagation Through Time (BPTT)

RNN Recap





features

$$X(1) = \begin{pmatrix} x11 & x12 & \ldots & x1n \\ x21 & x22 & \ldots & x2n \\ x31 & x32 & \ldots & x3n \end{pmatrix}$$

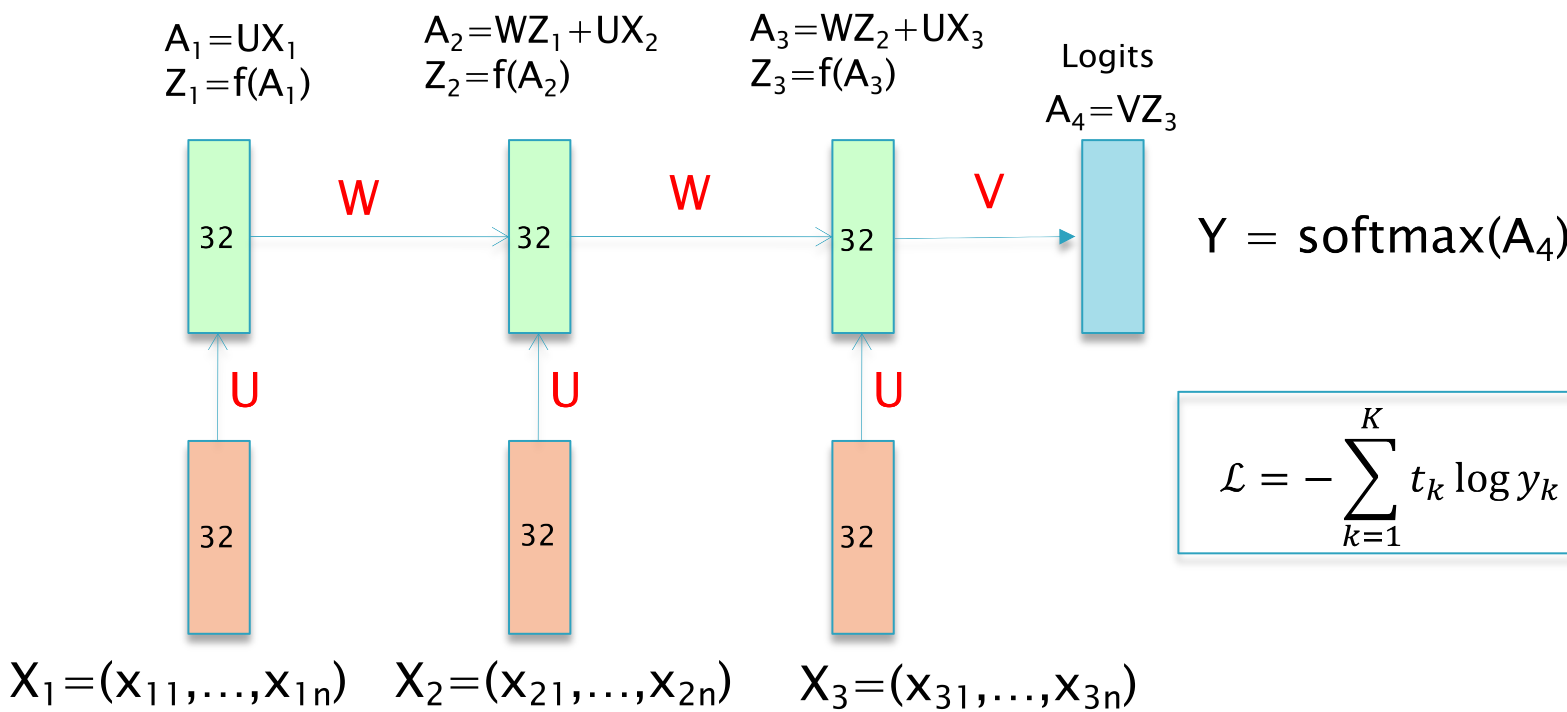
The matrix X(1) forms a single training sample

X(1) X(2) . . . X(N)

Training Data Set

NW Type	Input
Dense FF →	1D Tensors
ConvNet →	3D Tensors
RNN →	2D Tensors

BPTT: Forward Pass



$$\mathcal{L} = -\sum_{k=1}^{K} t_k \log y_k$$

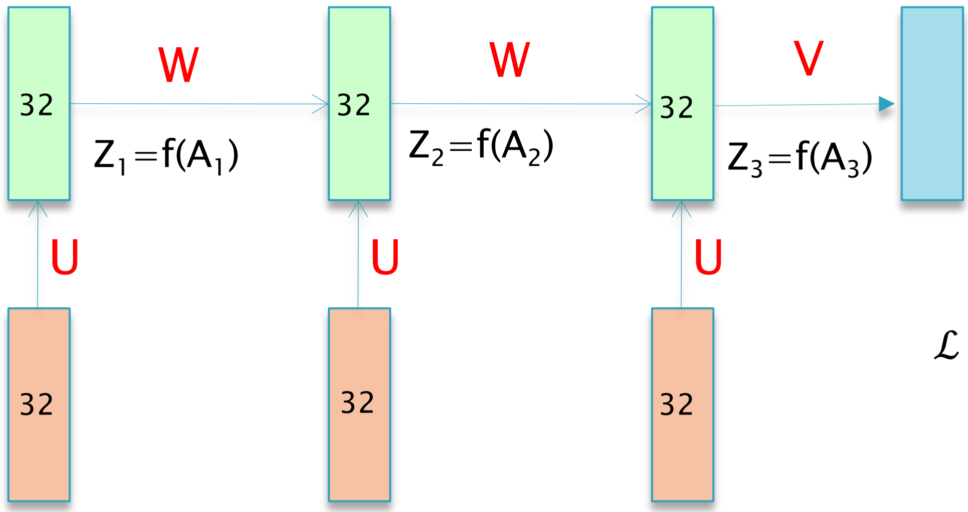
BPTT: Backward Pass

$\Delta_1 = f'(A_1) \circledast W^T \Delta_2$

$\Delta_3 = f'(A_3) \circledast V^T \Delta_4$

$\Delta_2 = f'(A_2) \circledast W^T \Delta_3$

$\Delta_4 = Y - T$



$Y = \mathrm{softmax}(VA_3)$

$A_4 = VZ_3$

$$\mathcal{L} = -\sum_{k=1}^{K} t_k \log y_k$$

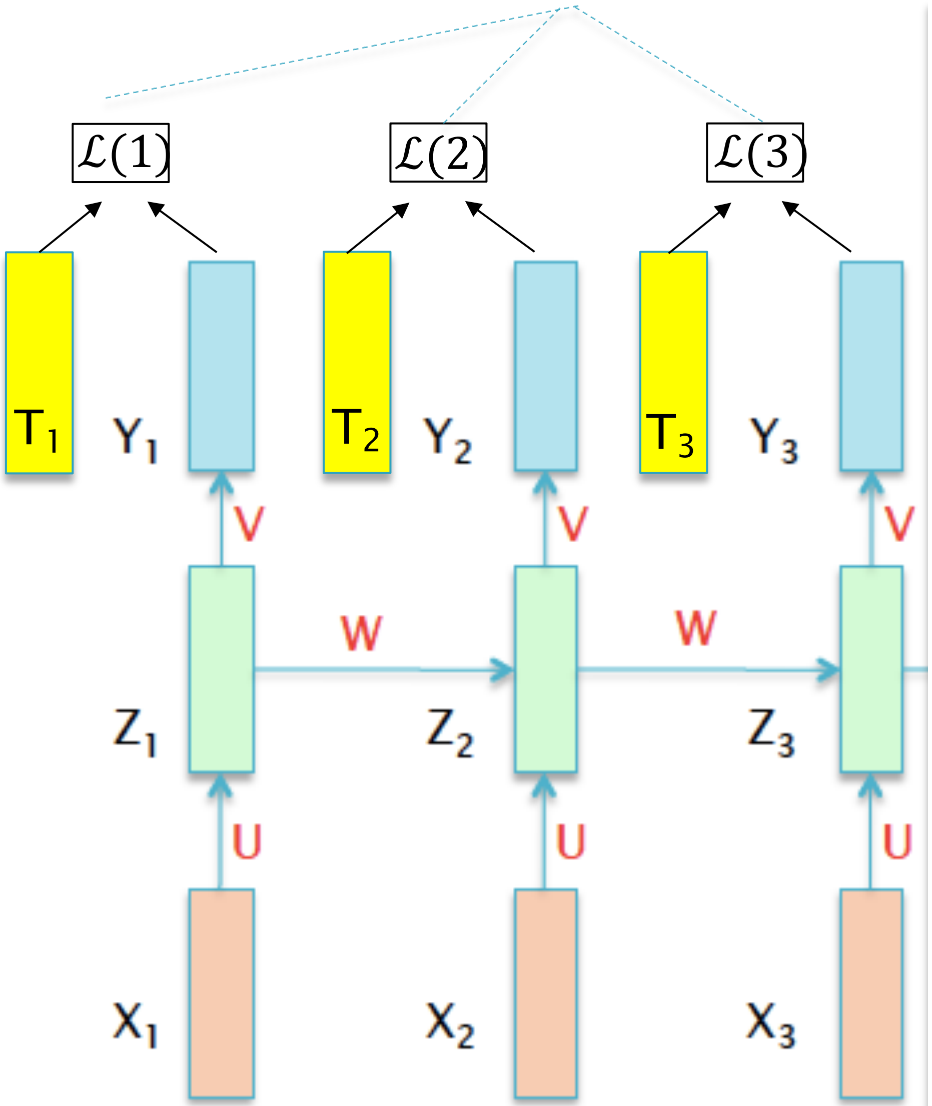
$X_1 = (x_{11}, \ldots, x_{1n})$ $X_2 = (x_{21}, \ldots, x_{2n})$ $X_3 = (x_{31}, \ldots, x_{3n})$

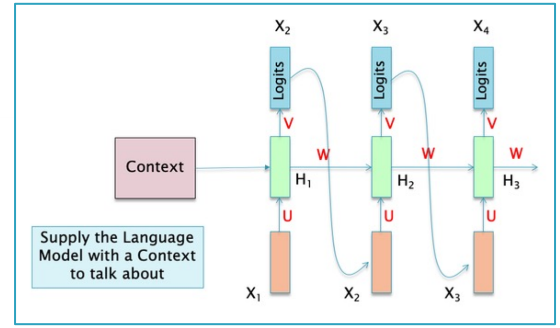
$$\frac{\partial \mathcal{L}}{\partial V} = Z_3^T \Delta_4 \qquad \frac{\partial \mathcal{L}}{\partial W} = Z_1^T \Delta_2 + Z_2^T \Delta_3$$

$$\frac{\partial \mathcal{L}}{\partial U} = X_1^T \Delta_1 + X_2^T \Delta_2 + X_3^T \Delta_3$$

Multiple Outputs

$$L = \sum_{m=1}^{M} \mathcal{L}(m)$$





$$\Delta \;\; = Y_3 - T_3$$

$$\Delta_3 = f'(A_3) \circledast V^T(Y_3 - T_3)$$

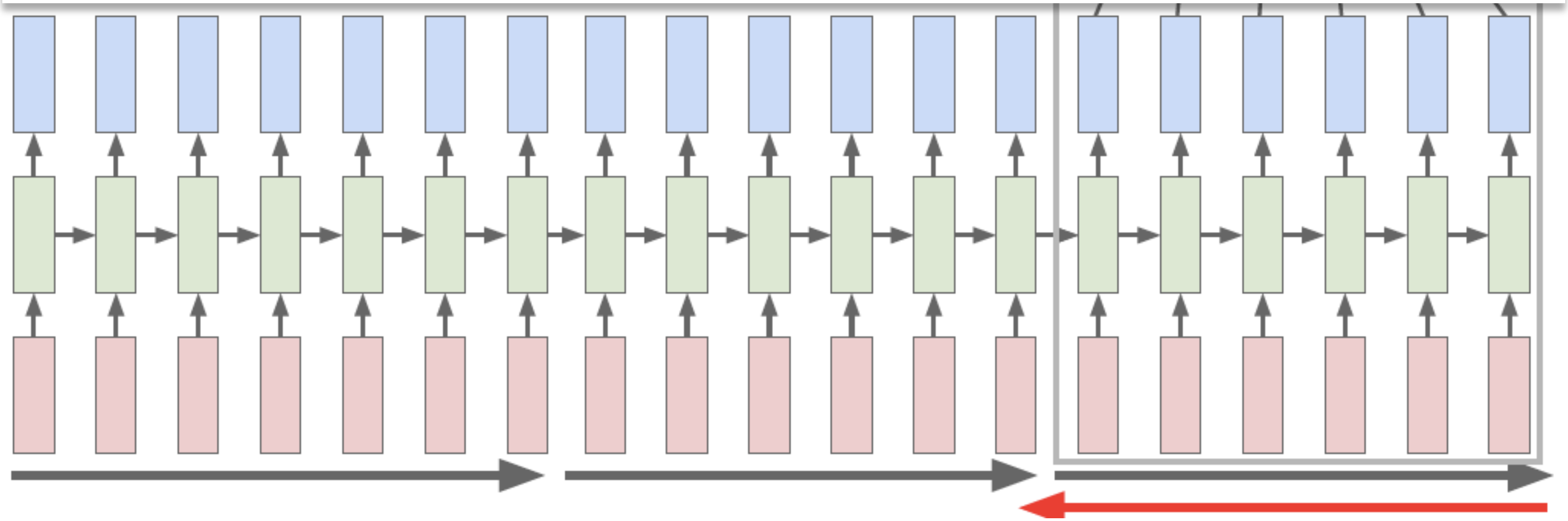
$$\Delta_2 = f'(A_2) \circledast W^T \Delta_3 + f'(A_2) \circledast V^T(Y_2 - T_2)$$

Long Sequences

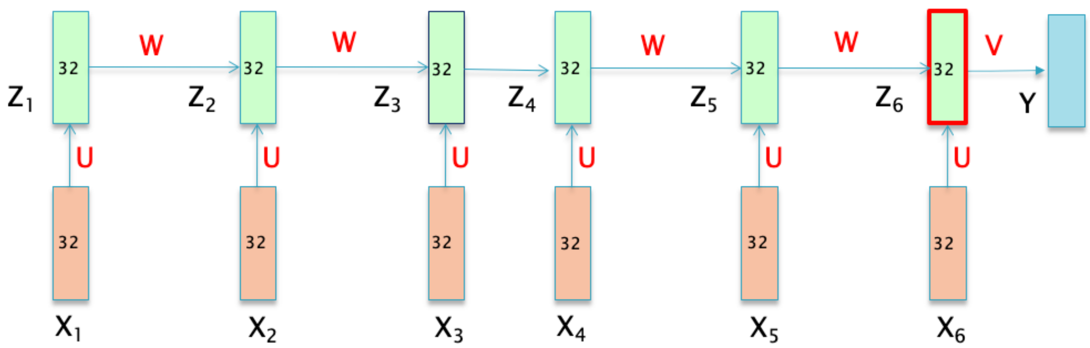
The number of RNN Stages depends upon the input data!

```
from keras.layers import Dense

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

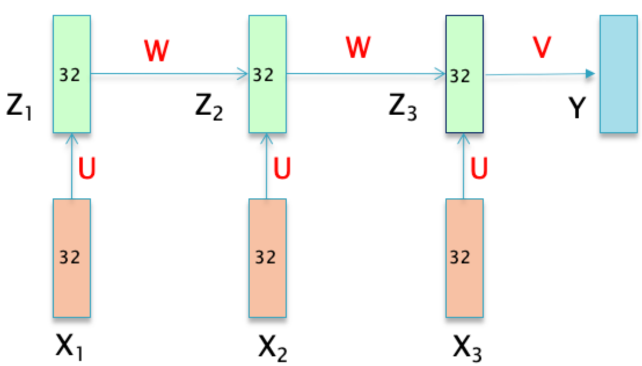
The Vanishing Gradient Problem Re-appears!

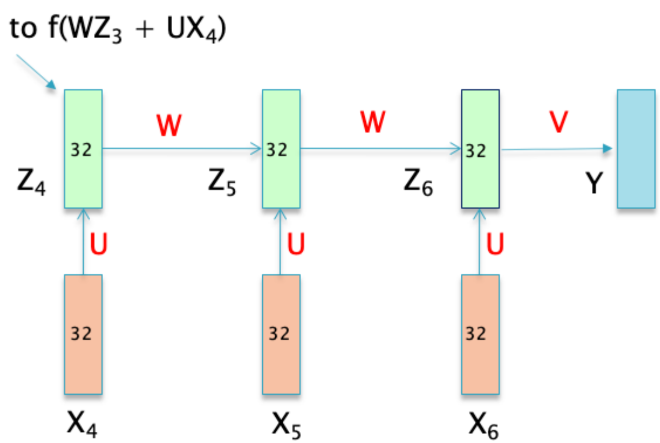


Truncated BPTT



(a)





(b) BPTT on Segment 1

(b) BPTT on Segment 2

Problems with BPTT

$$\frac{\partial \mathcal{L}}{\partial U} = X_1^T \Delta_1 + X_2^T \Delta_2 + X_3^T \Delta_3$$

Ideally if BPTT based training goes well, then ALL the inputs should be able to interact with each other by influencing the gradient updates

In practice, BPTT may run into the following problems:

- Vanishing Gradient Problem: the gradients Δ in the initial stages of the network become progressively smaller and get close to zero during Backprop, as the number of RNN stages increases.
- Exploding Gradient Problem

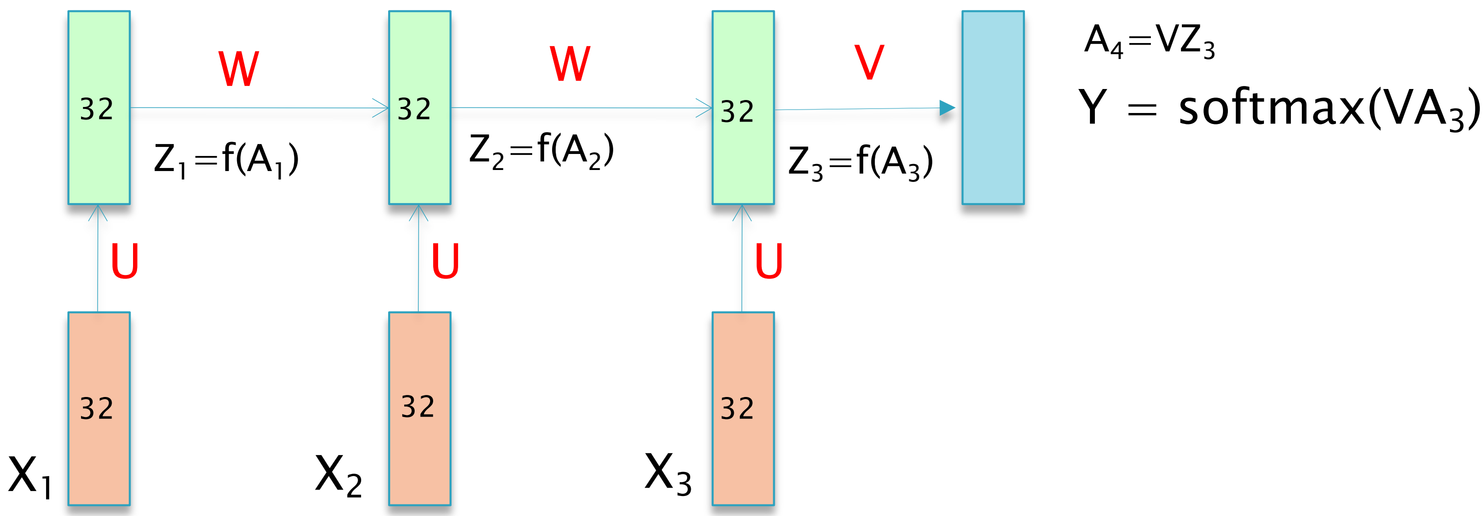
BPTT Issues

$$\Delta_1 = f'(A_1) \circledast W^T \Delta_2$$

$$\Delta_3 = f'(A_3) \circledast V^T \Delta_4$$

$$\Delta_2 = f'(A_2) \circledast W^T \Delta_3$$

$$\Delta_4 = Y - T$$



$$\Delta_3 = V^T \Delta_4$$

$$\Delta_2 = W^T \Delta_3 = W^T V^T \Delta_4$$

$$\Delta_1 = W^T \Delta_2 = (W^T)^2 \; V^T \Delta_4$$

In General

$$\Delta_1 = (W^T)^n \; V^T \Delta_n$$

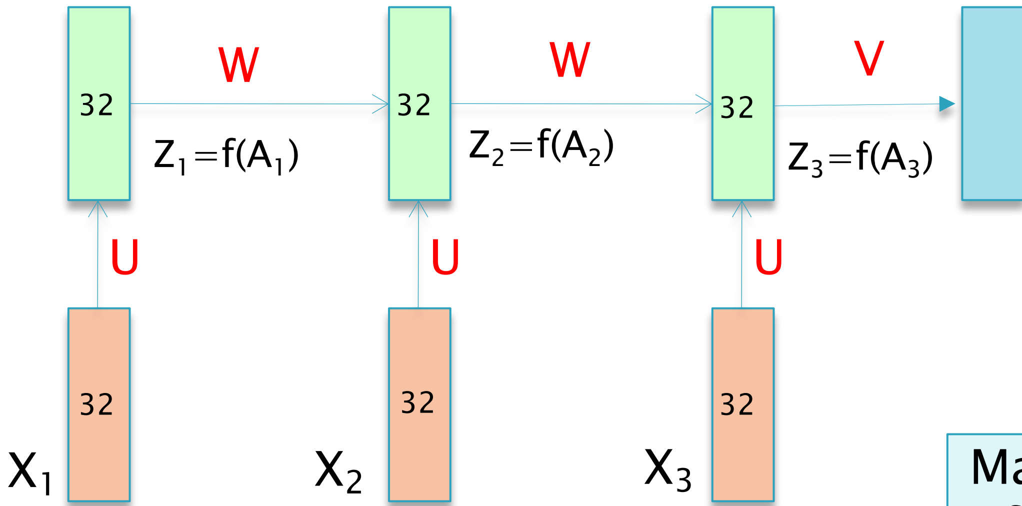
BPTT Issues

$\Delta_1 = f'(A_1) \circledast W^T \Delta_2$

$\Delta_3 = f'(A_3) \circledast V^T \Delta_4$

$\Delta_2 = f'(A_2) \circledast W^T \Delta_3$

$\Delta_4 = Y - T$



$A_4 = VZ_3$

$Y = \text{softmax}(VA_3)$

Matrix Factoring result from Linear Algebra

In General

$\Delta_1 = (W^T)^n \, V^T \Delta_V$

$W^T = R \Lambda R^T$

Λ is a diagonal matrix with eigenvalues λ_i

Then $(W^T)^n = R \Lambda^n R^T$

Λ^n is a diagonal matrix with eigenvalues λ_i^n

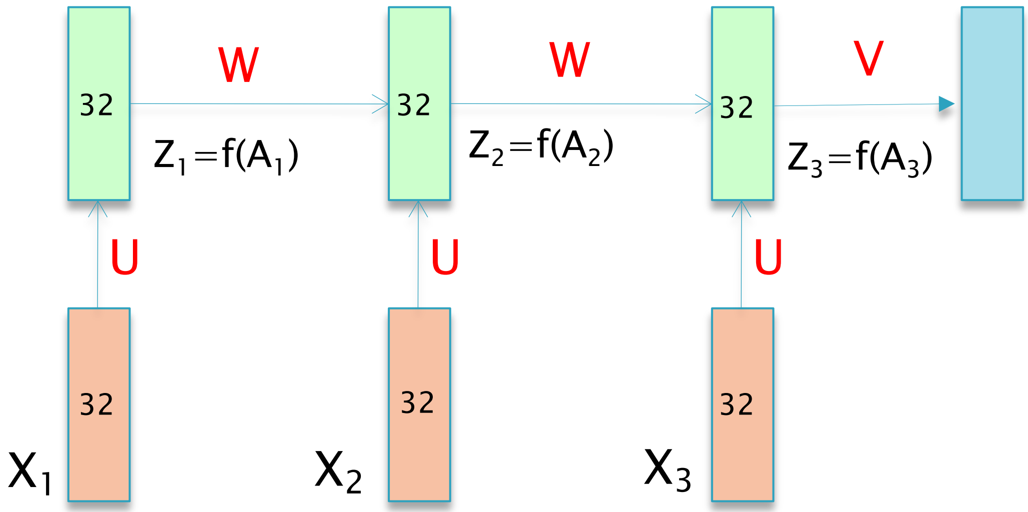
BPTT Issues

$\Delta_1 = f'(A_1) \circledast W^T \Delta_2$

$\Delta_2 = f'(A_2) \circledast W^T \Delta_3$

$\Delta_3 = f'(A_3) \circledast V^T \Delta_4$

$\Delta_4 = Y - T$



$A_4 = VZ_3$

$Y = \text{softmax}(VA_3)$

In General

$\Delta_1 = (W^T)^n \, V^T \Delta_f$

Then $(W^T)^n = R\Lambda^n R^T$

Λ^n is a diagonal matrix with eigenvalues λ_i^n

As $n \to \infty$: $if\ \lambda_i < 1,\ \ then\ \Delta_1 \to 0$ — Vanishing Gradients

As $n \to \infty$: $if\ \lambda_i > 1,\ \ then\ \Delta_1 \to \infty$ — Exploding Gradients

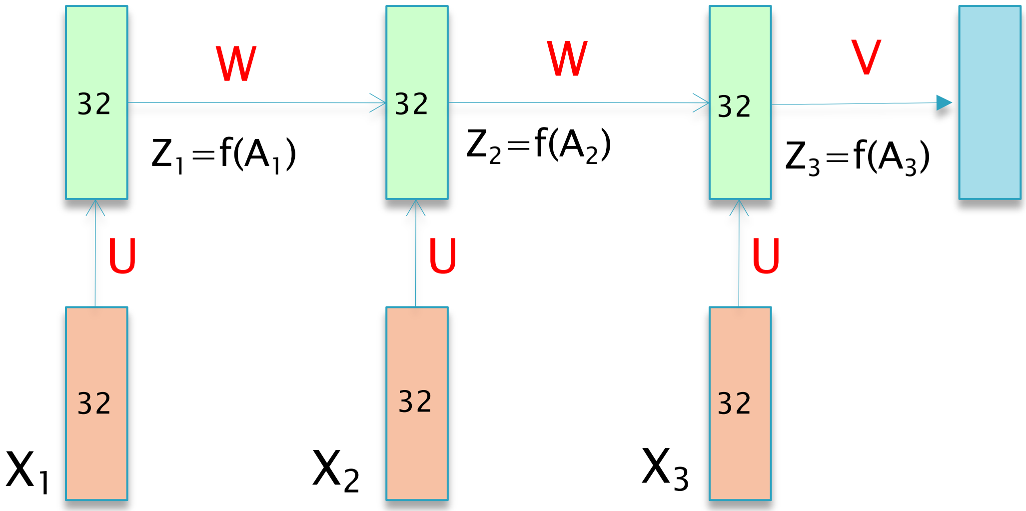
BPTT Issues: Vanishing Gradients

$\Delta_1 = f'(A_1) \circledast W^T \Delta_2$

$\Delta_3 = f'(A_3) \circledast V^T \Delta_4$

$\Delta_2 = f'(A_2) \circledast W^T \Delta_3$

$\Delta_4 = Y - T$



$A_4 = VZ_3$

$Y = \text{softmax}(VA_3)$

$$\frac{\partial \mathcal{L}}{\partial V} = Z_3^T \Delta_4$$

$$\frac{\partial \mathcal{L}}{\partial W} = Z_1^T \Delta_2 + Z_2^T \Delta_3$$

$$\frac{\partial \mathcal{L}}{\partial U} = X_1^T \Delta_1 + X_2^T \Delta_2 + X_3^T \Delta_3$$

In General

$\Delta_1 = (W^T)^n \, V^T \Delta_f$

Then $(W^T)^n = R \Lambda^n R^T$

Λ^n is a diagonal matrix with eigenvalues λ_i^n

As $n \to \infty$: $if \ \lambda_i < 1, \ \ then \ \Delta_1 \to 0$

$$\frac{\partial \mathcal{L}}{\partial U} = X_1^T \Delta_1 + X_2^T \Delta_2 + \ \ldots \ + X_n^T \Delta_n$$

The early inputs stop influencing the gradient updates

Solution to Exploding Gradient Problem

- The solution first introduced by Mikolov is to clip gradients to a maximum value.

Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

- Makes a big difference in RNNs.

Vanishing Gradient Problem

- More difficult problem to solve
  - Problem caused to multiple matrix multiplications, which is intrinsic to the architecture
- Solution requires a major change in RNN architecture

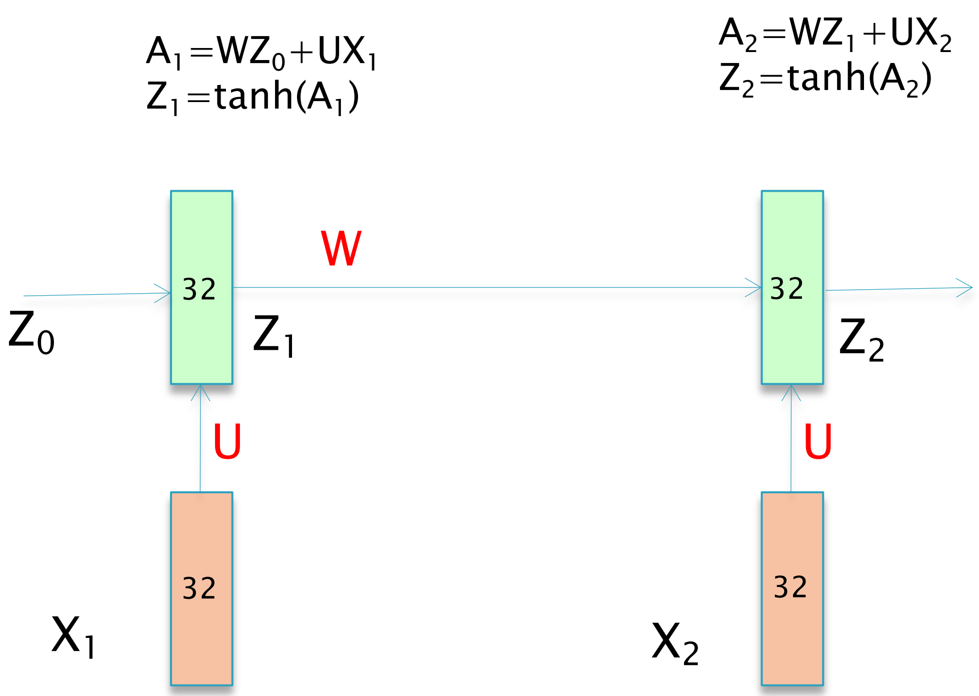
LSTM
Long Short Term Memory

LSTMs

- LSTMs were designed with the objective of solving the Vanishing Gradients Problem in RNNs
- They have been very successful in doing so, indeed almost all of the successful applications of Recurrent Networks in recent years have been with LSTMs rather than with plain RNNs

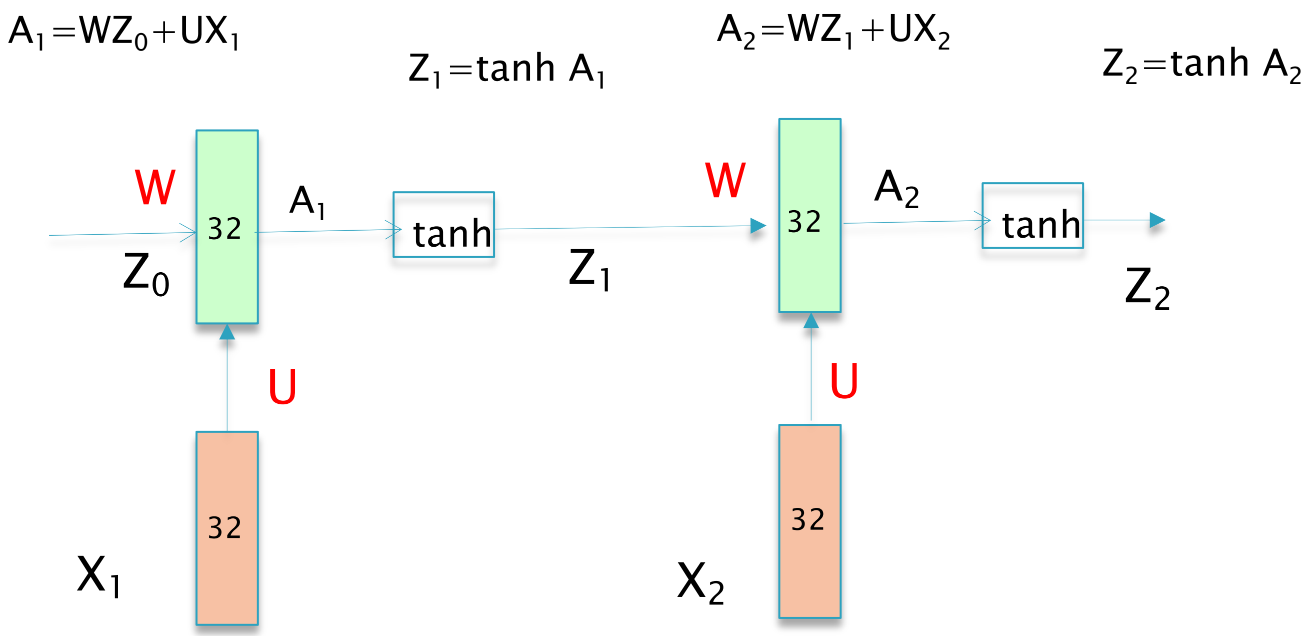
ConvNets	← →	Image Processing
LSTMs, Transformers	← →	Sequence Processing/NLP

LSTM – 1



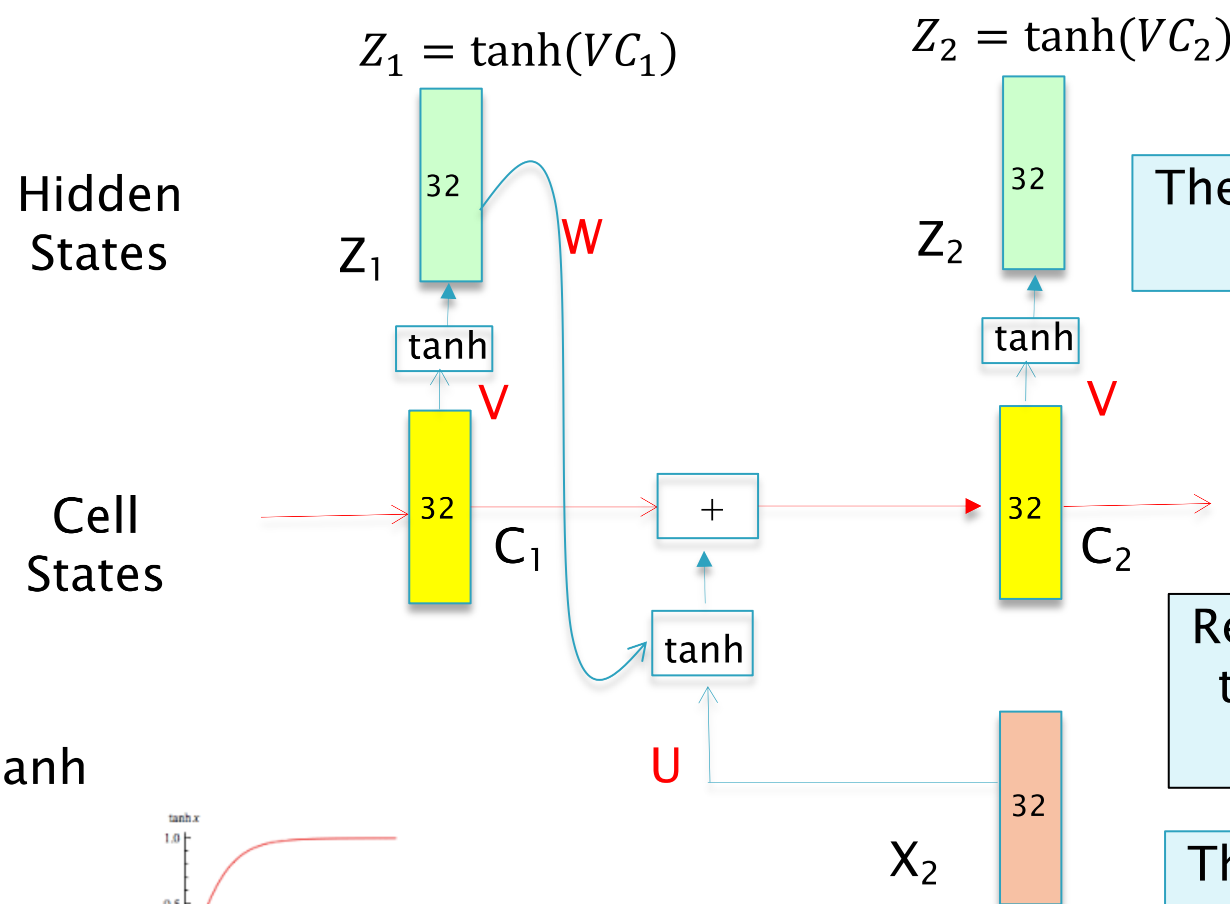
Just a regular RNN

LSTM – 2



Just a regular RNN

LSTM – 3: Cell States

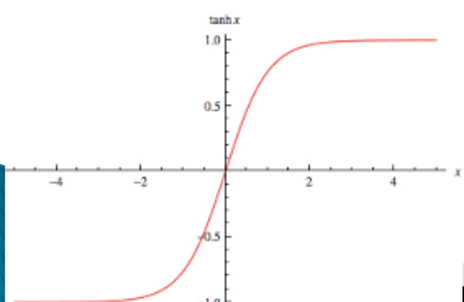


The Hidden State is Derived from the Cell State

Recurrence has shifted to the Cell State from the Hidden State

The Cell State is updated Additively!

tanh

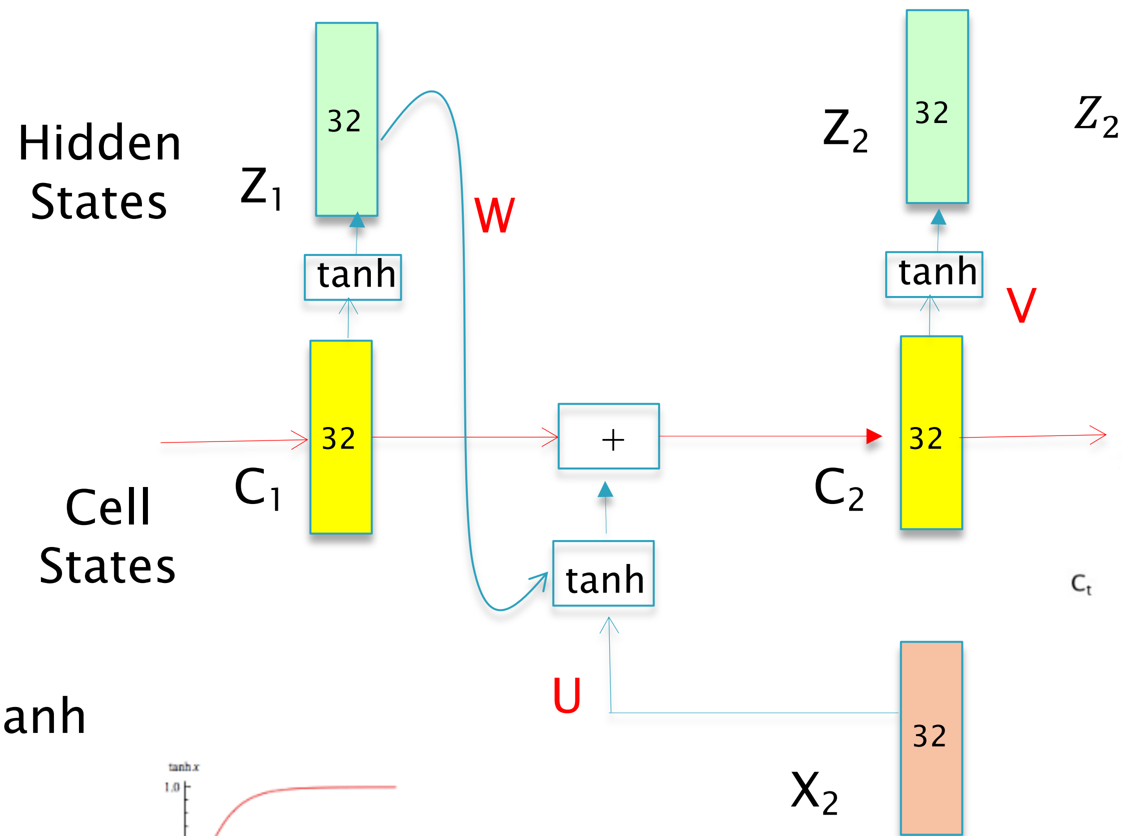


$$C_2 = C_1 + \tanh(WZ_1 + UX_2)$$

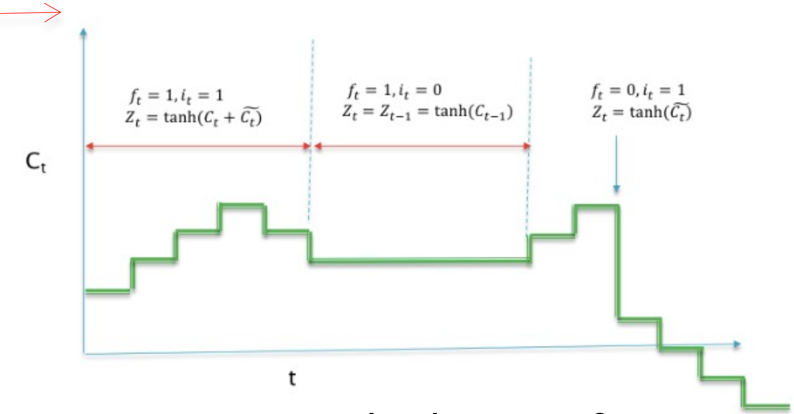
Cell State is a Vector

$$C_1 = (c_{11}, c_{12}, \ldots, c_{1n})$$

LSTM – 4

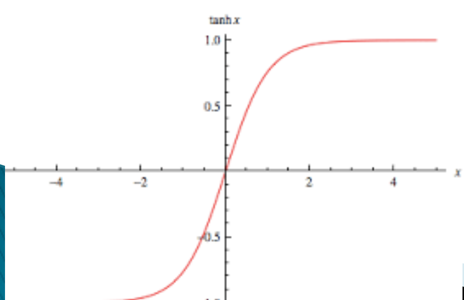


$$Z_2 = \tanh(VC_2)$$



A single element of the cell state vector

tanh

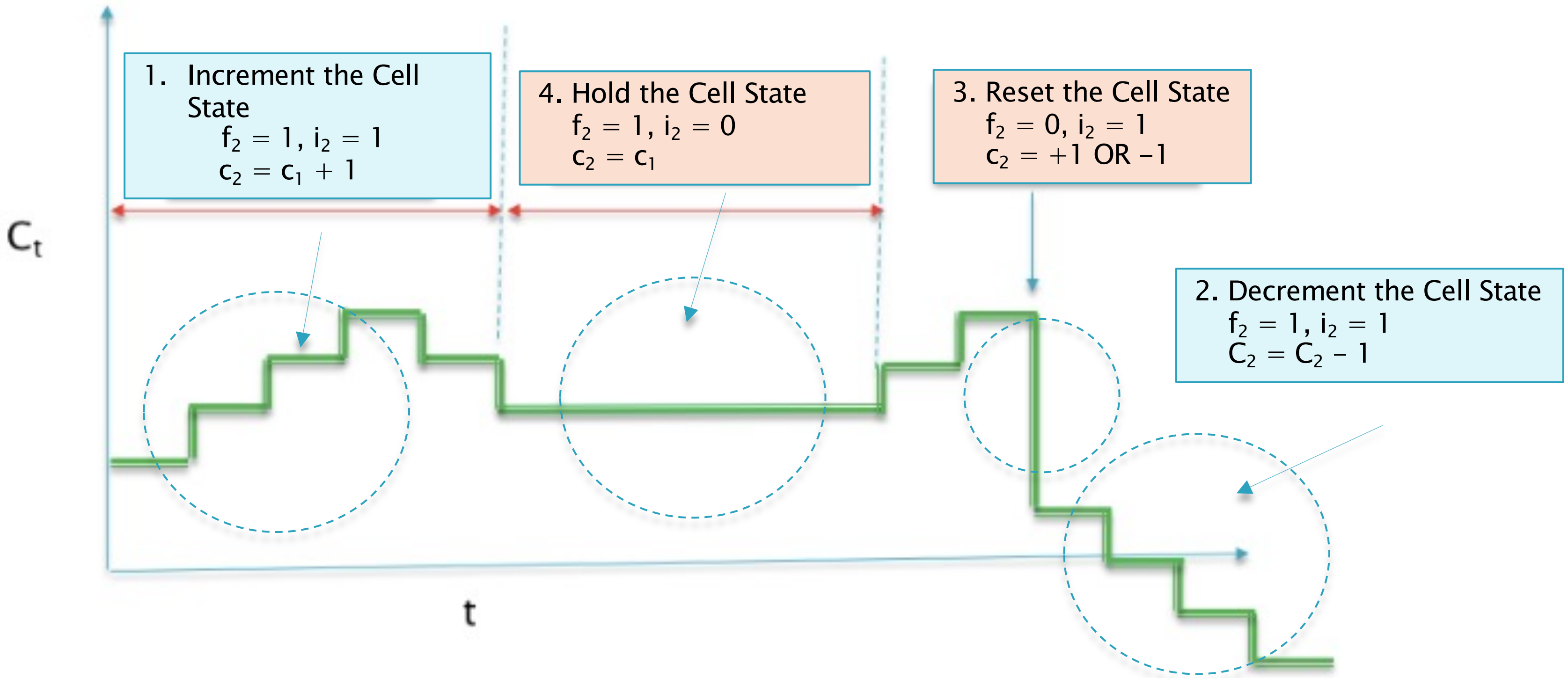


$$C_2 = C_1 + \tanh(WZ_1 + UX_2)$$

Cell State is a Vector

$$C_1 = (c_{11}, c_{12}, \ldots, c_{1n})$$

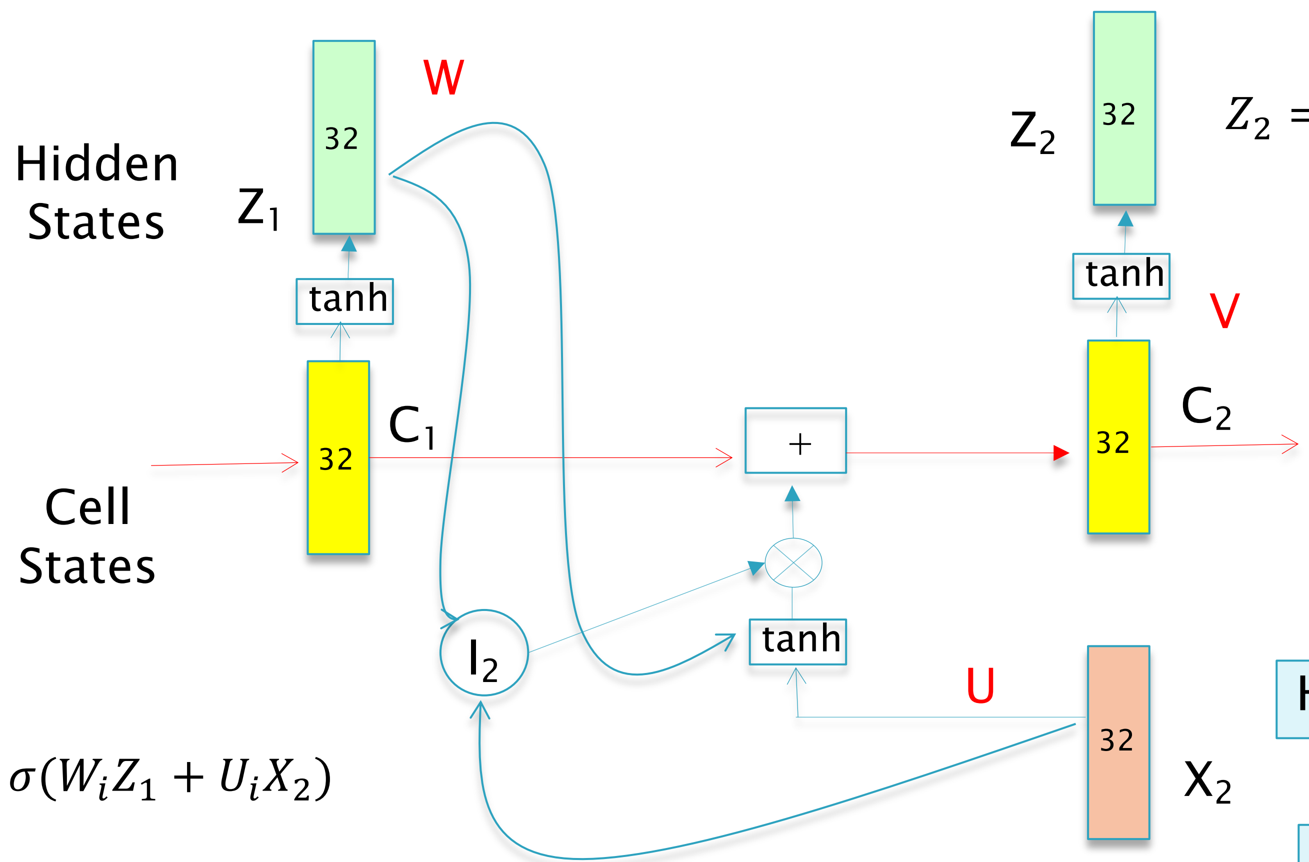
Cell State Changes: Introducing <u>Gates</u>



$$C_2 = F_2 \circledast C_1 + I_2 \circledast \tanh(WZ_1 + UX_2)$$

Gates f=1, i=0 implements the Hold function
Gates f=0, i=1 implements the Reset function

LSTM – 5: Input Gate i



$Z_2 = \tanh(VC_2)$

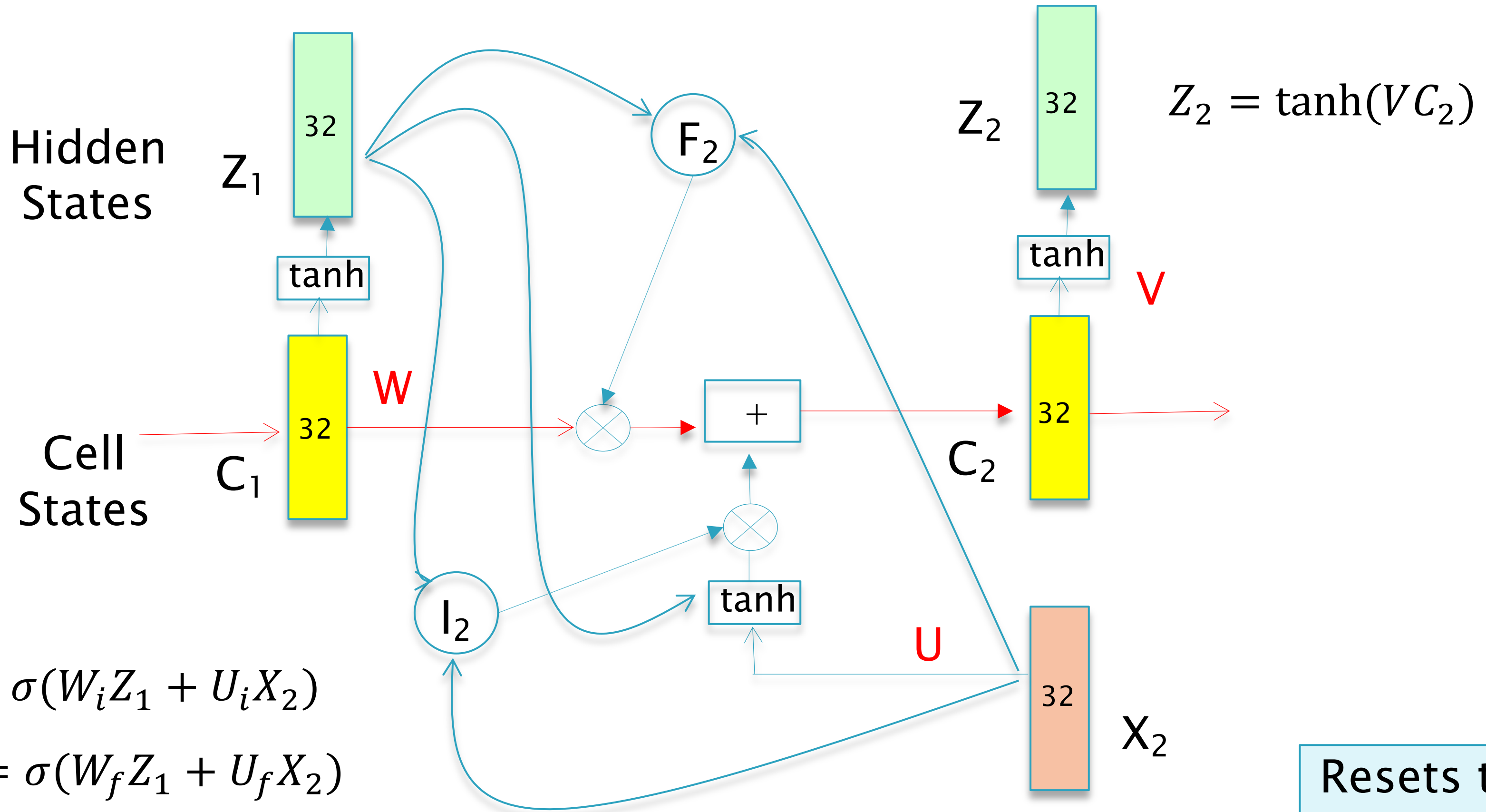
$I_2 = \sigma(W_i Z_1 + U_i X_2)$

$C_2 = C_1 + I_2 \circledast \tanh(WZ_1 + UX_2)$

Hold the Cell State

Blocks the input from changing Cell State

LSTM – 6: Forget Gate F



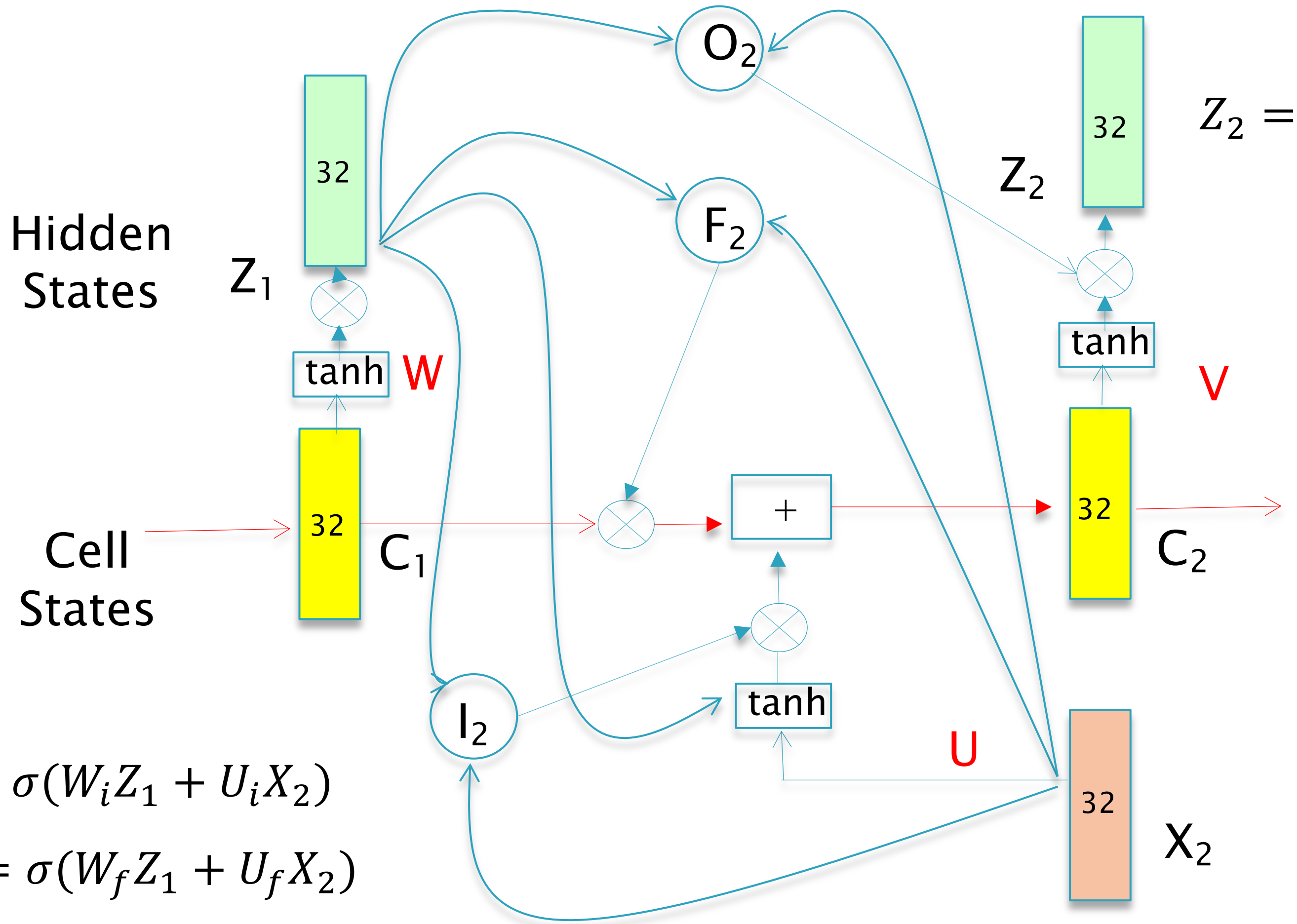
$$I_2 = \sigma(W_i Z_1 + U_i X_2)$$

$$F_2 = \sigma(W_f Z_1 + U_f X_2)$$

Resets the Cell State

$$C_2 = F_2 \circledast C_1 + I_2 \circledast \tanh(WZ_1 + UX_2)$$

LSTM – 7: Output Gate O



$$Z_2 = O_2 \circledast \tanh(VC_2)$$

Regulates the Cell State to Hidden State connection

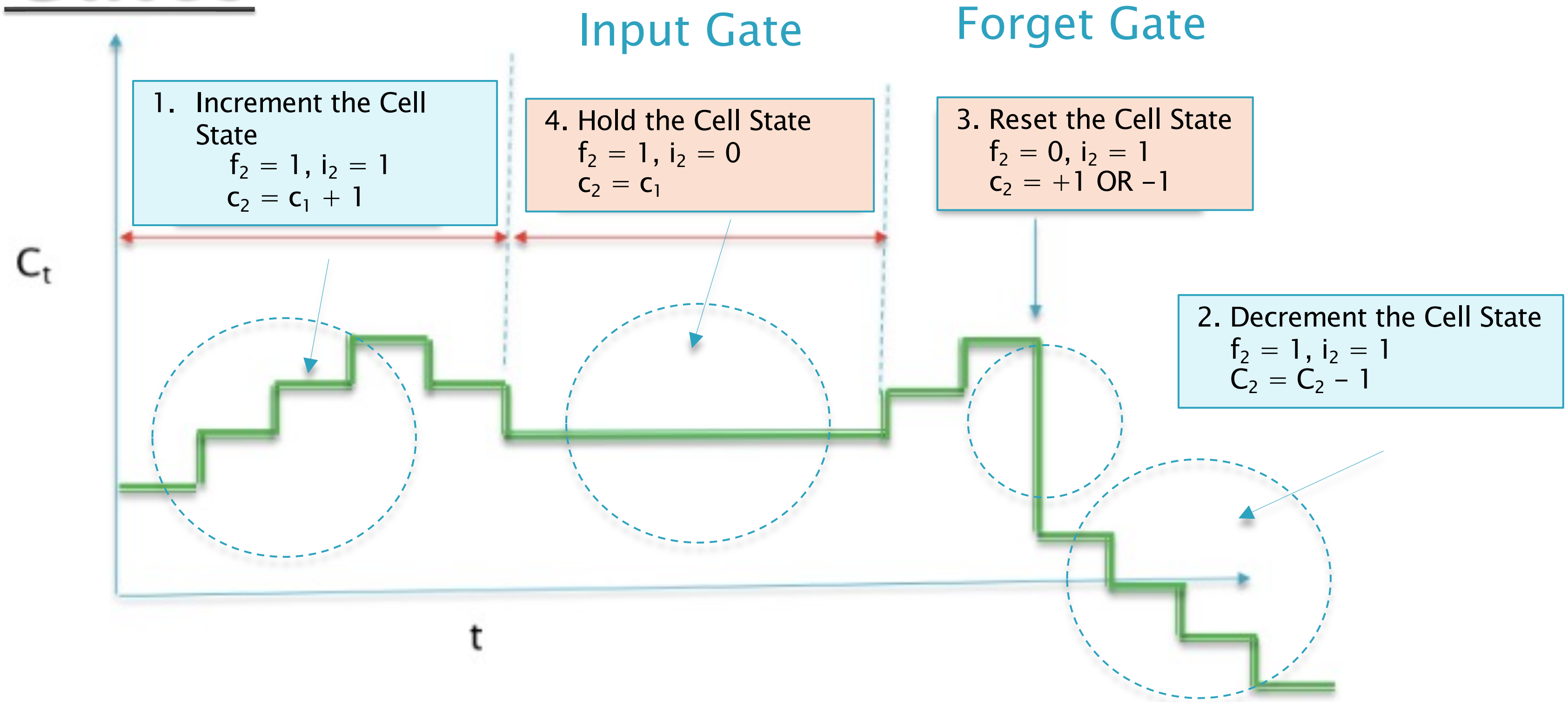
$$I_2 = \sigma(W_i Z_1 + U_i X_2)$$

$$F_2 = \sigma(W_f Z_1 + U_f X_2)$$

$$O_2 = \sigma(W_o Z_1 + U_o X_2)$$

$$C_2 = F_2 \circledast C_1 + I_2 \circledast \tanh(WZ_1 + UX_2)$$

Cell State Changes: Introducing Gates



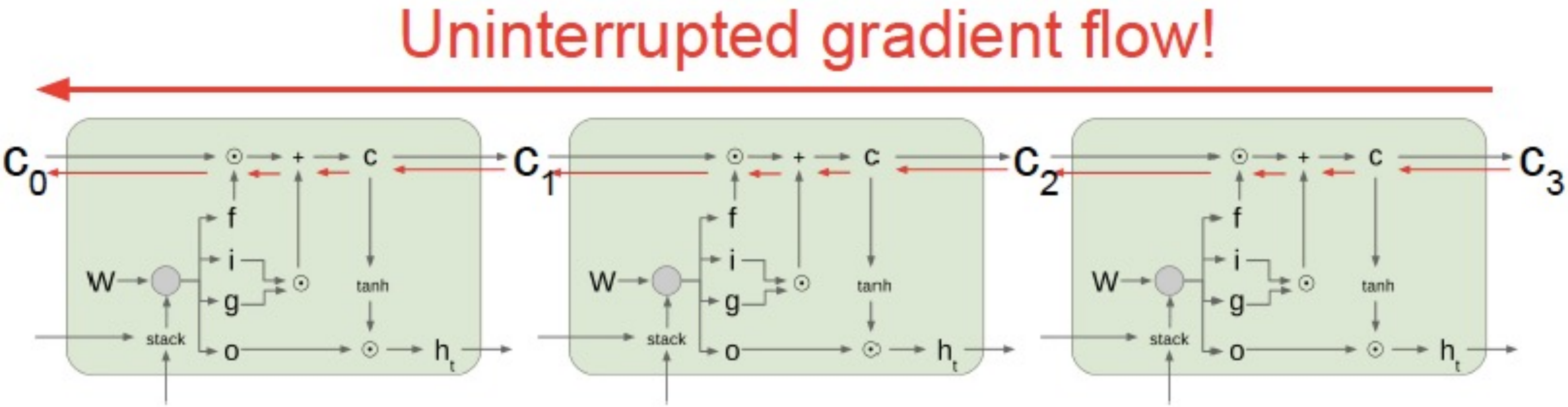
$$C_2 = F_2 \circledast C_1 + I_2 \circledast \tanh(WZ_1 + UX_2)$$

Gates f=1, i=0 implements the Hold function

Gates f=0, i=1 implements the Reset function

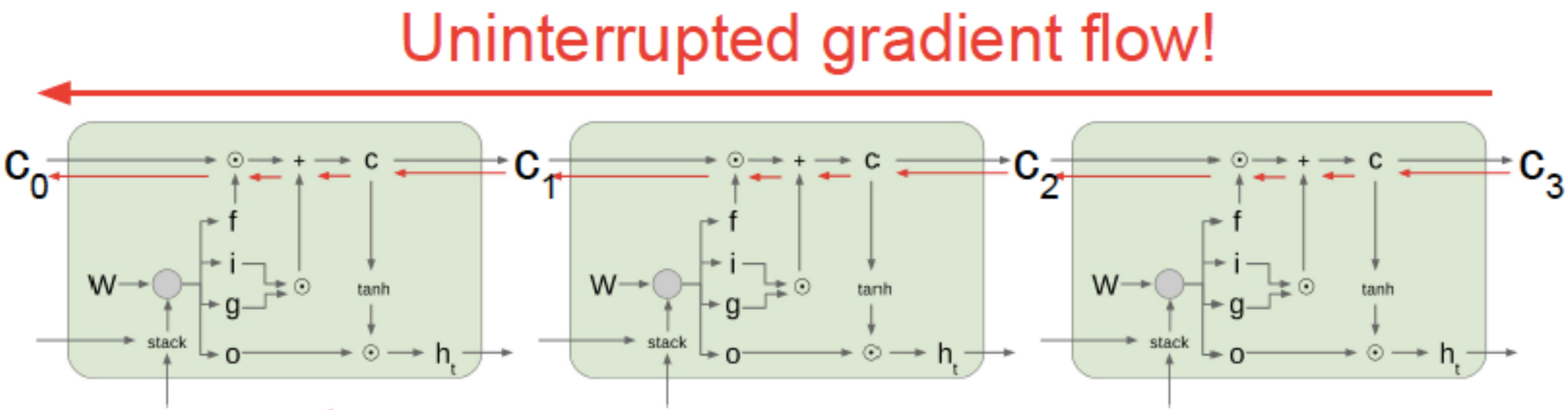
LSTM: Backprop

Long Short Term Memory (LSTM): Gradient Flow
[Hochreiter et al., 1997]

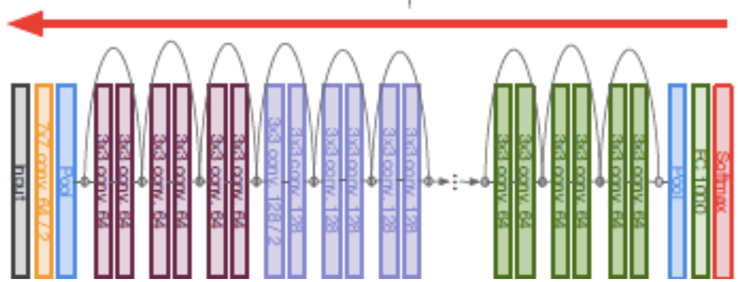


LSTM: Backprop

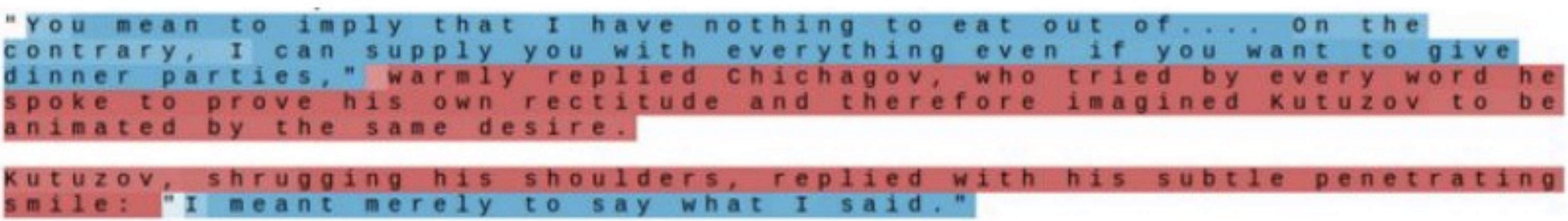
Long Short Term Memory (LSTM): Gradient Flow
[Hochreiter et al., 1997]



Similar to ResNet!



Searching for Interpretable Cells

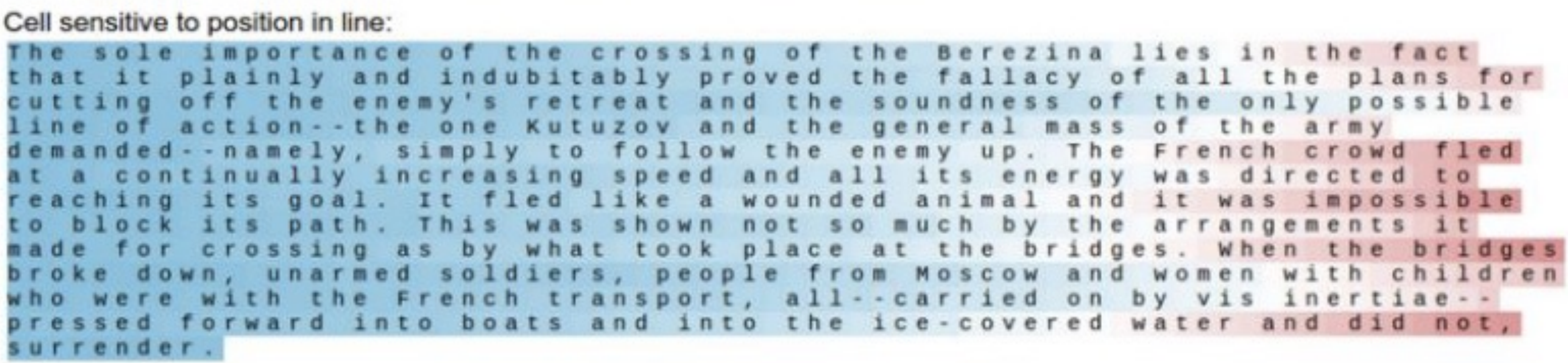


quote detection cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

$$C_1 = (c_{11}, c_{12}, \ldots, c_{1n})$$

Searching for Interpretable Cells



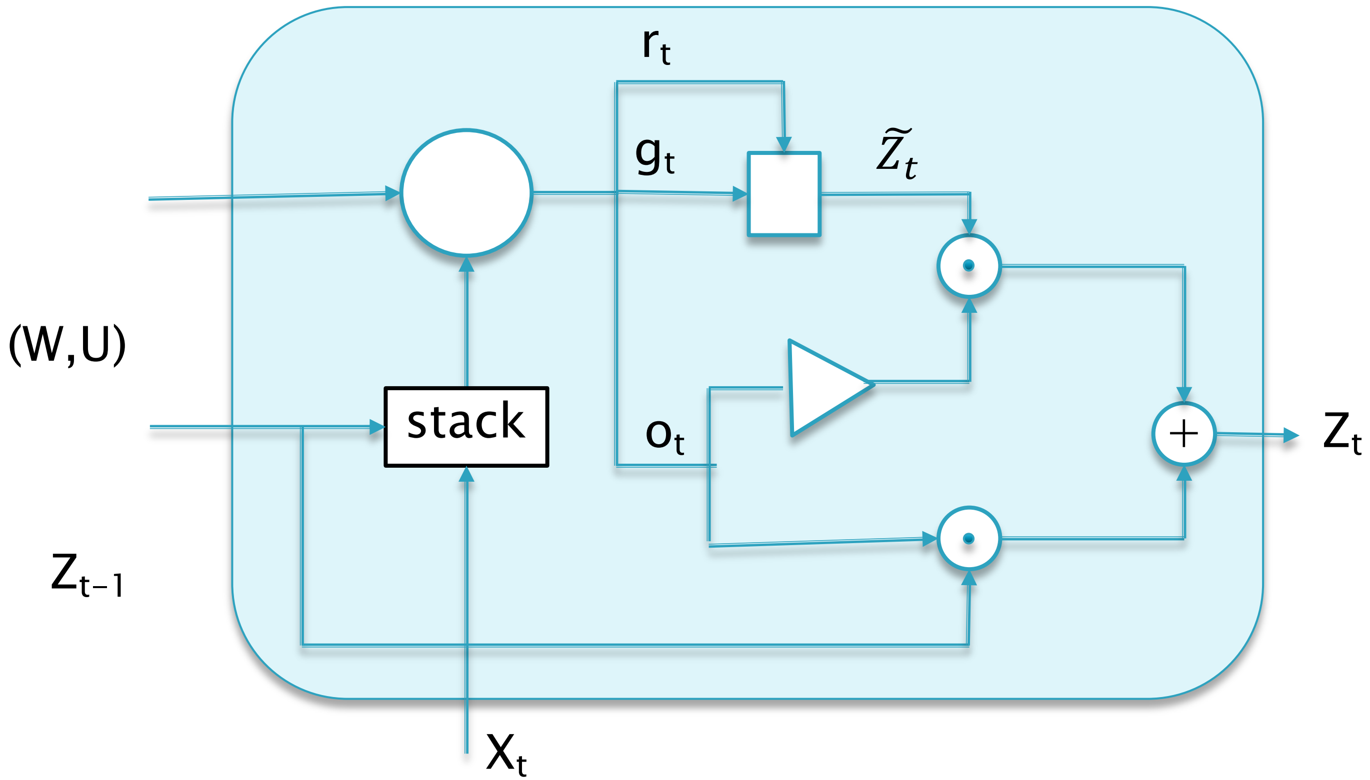
line length tracking cell

Searching for Interpretable Cells

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
  int i;
  if (classes[class]) {
    for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
      if (mask[i] & classes[class][i])
        return 0;
  }
  return 1;
}
```

code depth cell

Gated Recurrent Units (GRUs)



The design of the GRU features two gates, the Update Gate o_t, given by:

$$o_t = \sigma(W^o X_t + U^o Z_{t-1}) \qquad (**GRU1**)$$

and the Reset Gate r_t given by:

$$r_t = \sigma(W^r X_t + U^r Z_{t-1}) \qquad (**GRU2**)$$

The Hidden State vector Z_{t-1} from the prior stage is combined with the Input vector X_t from the current stage, to generate an intermediate Hidden State value $\tilde{Z}_t$ given by:

$$\tilde{Z}_t = \tanh(r_t \odot U Z_{t-1} + W X_t) \qquad (**GRU3**)$$

Finally the new Hidden State Z_t is generated by combining the prior Hidden State Z_{t-1} with the intermediate value $\tilde{Z}_t$ as follows:

$$Z_t = (1 - o_t) \odot \tilde{Z}_t + o_t \odot Z_{t-1} \qquad (**GRU4**)$$

Invoking LSTM or GRU in Keras

```
from keras import layers

model = keras.models.Sequential()
model.add(layers.LSTM(128, input_shape=(maxlen, len(chars))))
model.add(layers.Dense(len(chars), activation='softmax'))
```

```
model = Sequential()
model.add(layers.GRU(32, input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))
```

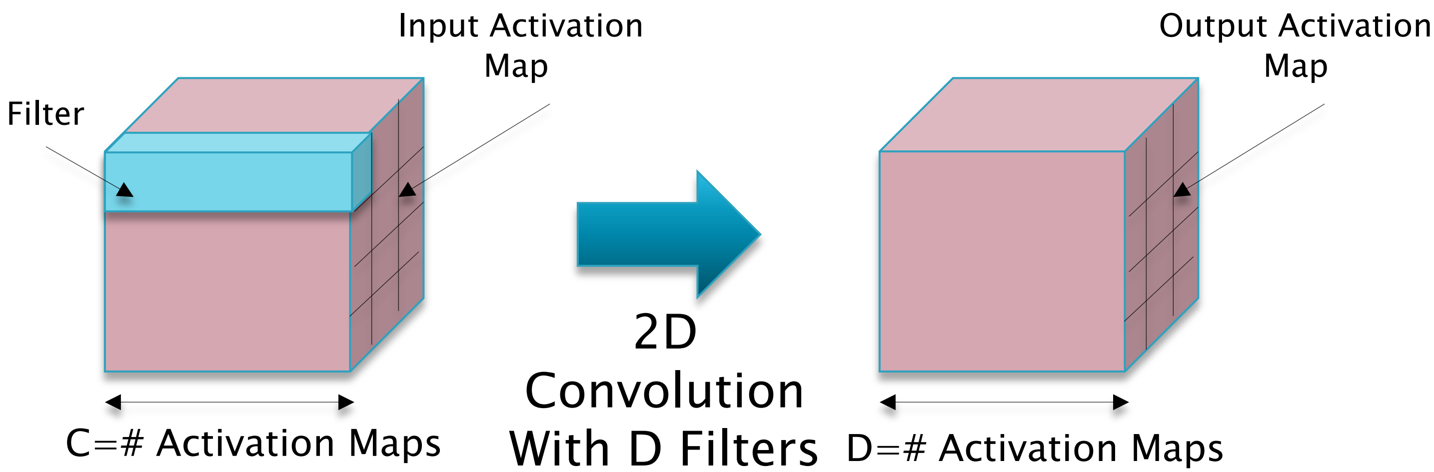
One Dimensional Convolutions

Why Use 1D Convolutions?

A way in which Convolutional Networks can be used for processing sequences:

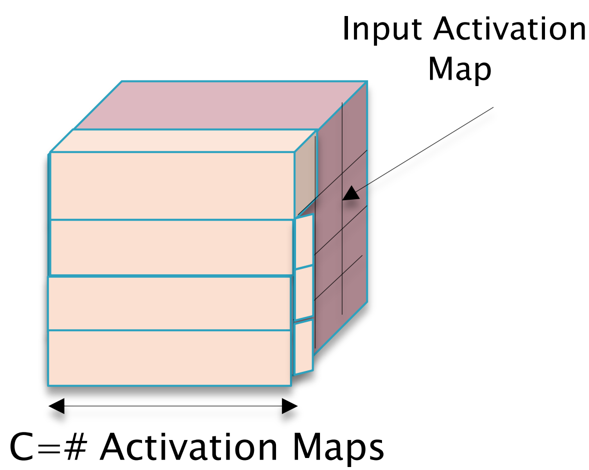
- Natural Language Processing
- Structured Data (CSV or Excel)

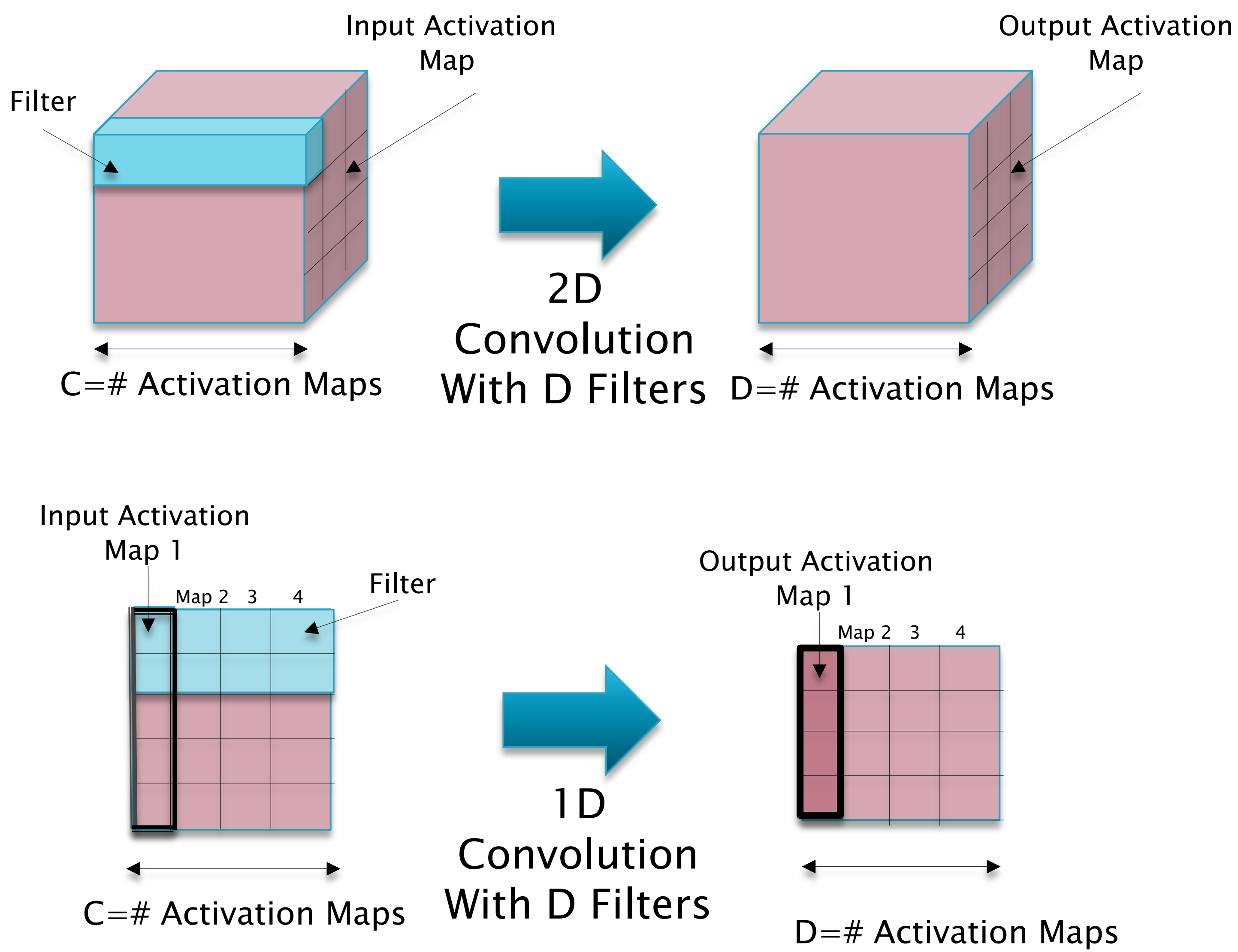
An Alternative to using RNNs/LSTMs



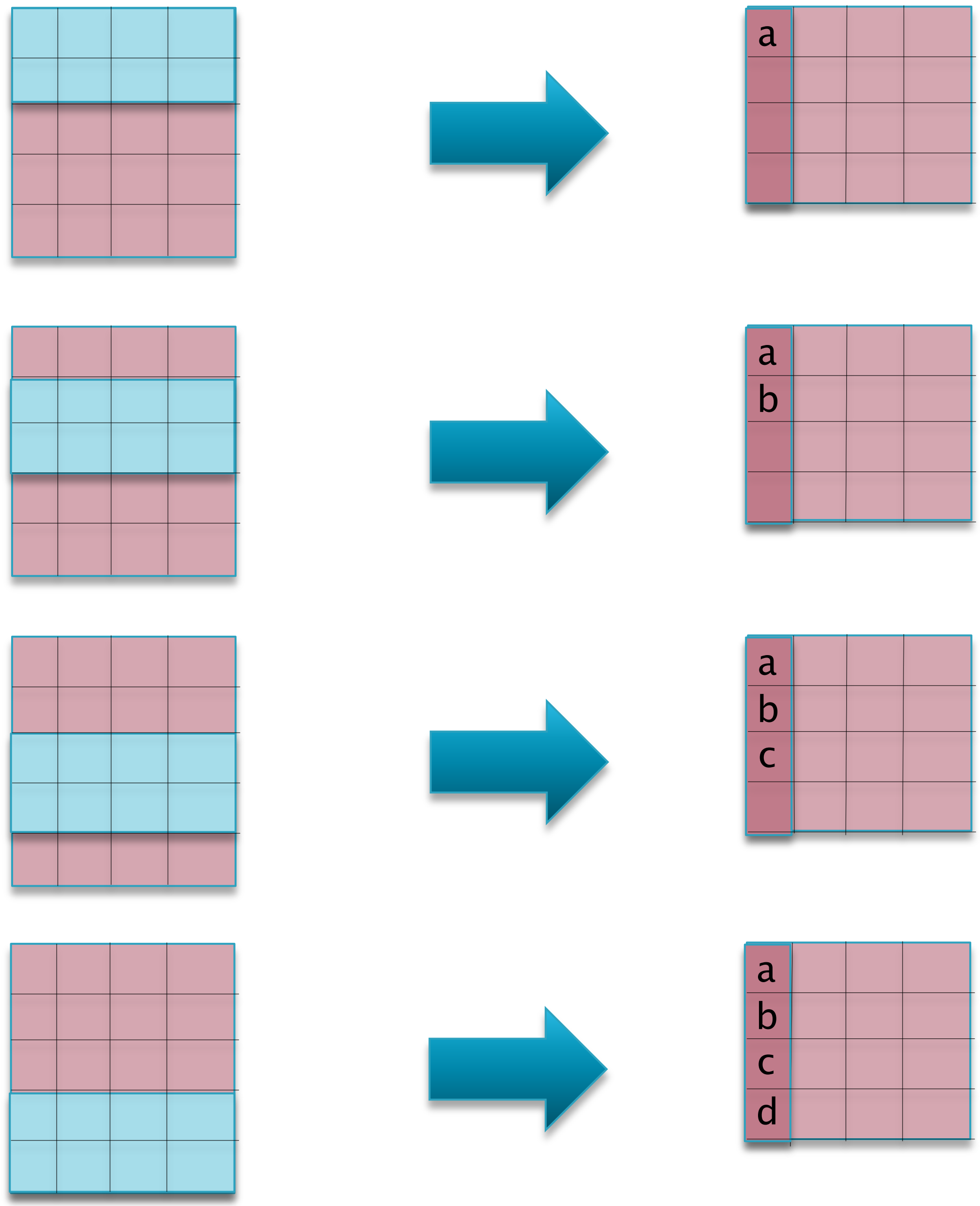
Slice the First Column to create a 2D Tensor

1D Input





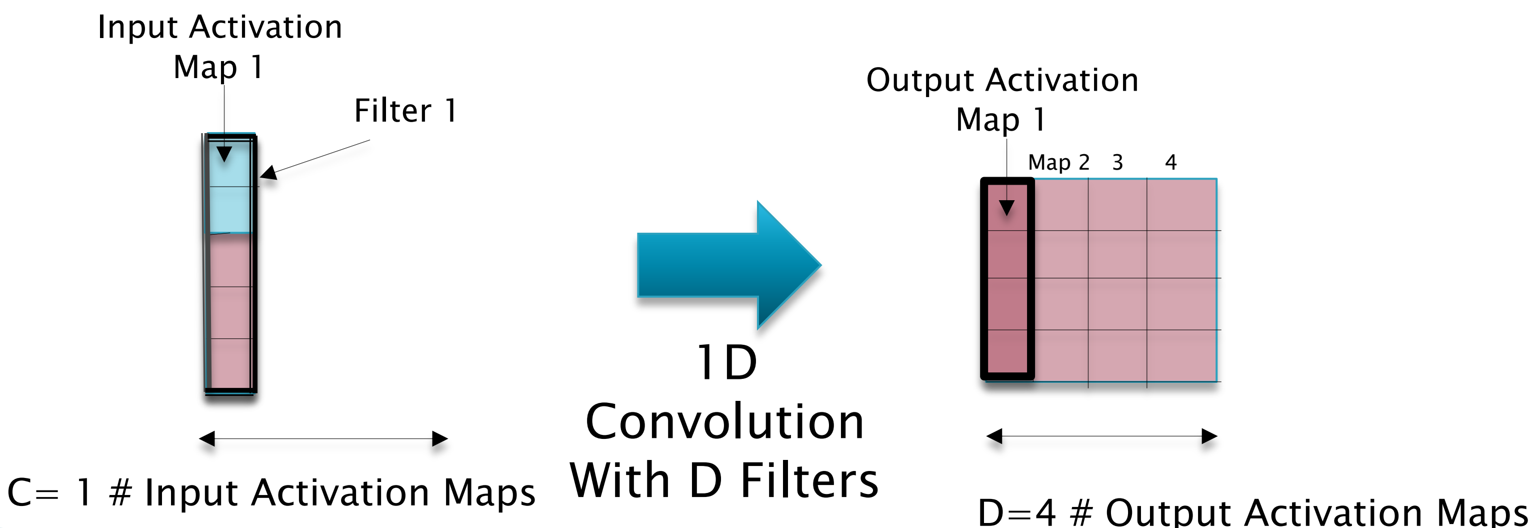
Input Activation Map
Filter
C=# Activation Maps
2D Convolution With D Filters
Output Activation Map
D=# Activation Maps
Input Activation Map 1
Map 2 3 4
Filter
C=# Activation Maps
1D Convolution With D Filters
Output Activation Map 1
Map 2 3 4
D=# Activation Maps



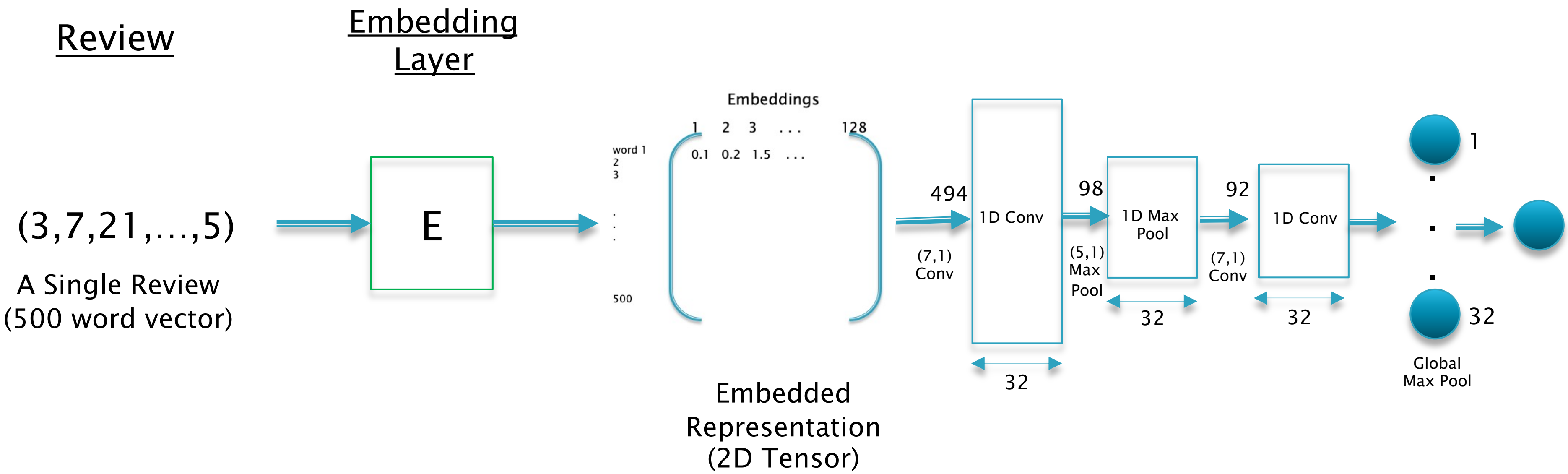
1D Convolution
With 2x1 Filter
With Depth 4

Why are 1D Convolutions Useful?

- 1-D Convolutions can be used to process 2D and 1D input data. Alternative to using RNN/LSTMs
  Examples:
  - NLP
  - Tabular Data



Processing IMDB Reviews with 1D ConvNets



$$N2 = \frac{N1 - F + 2P}{S} + 1$$

1D Convolutions in Keras

```
model_m = Sequential()
model_m.add(Reshape((TIME_PERIODS, num_sensors), input_shape=(input_shape,)))
model_m.add(Conv1D(100, 10, activation='relu', input_shape=(TIME_PERIODS, num_sensors)))
model_m.add(Conv1D(100, 10, activation='relu'))
model_m.add(MaxPooling1D(3))
model_m.add(Conv1D(160, 10, activation='relu'))
model_m.add(Conv1D(160, 10, activation='relu'))
model_m.add(GlobalAveragePooling1D())
model_m.add(Dropout(0.5))
model_m.add(Dense(num_classes, activation='softmax'))
print(model_m.summary())
```

Lots of Tools for Processing Sequence Data

- RNNs
- LSTMs
- GRUs
- 1D ConvNets

- Transformers

Further Reading

- Das and Varma: Chapter RNNs, Chapter ConvNets Part 1 for ID CNNs