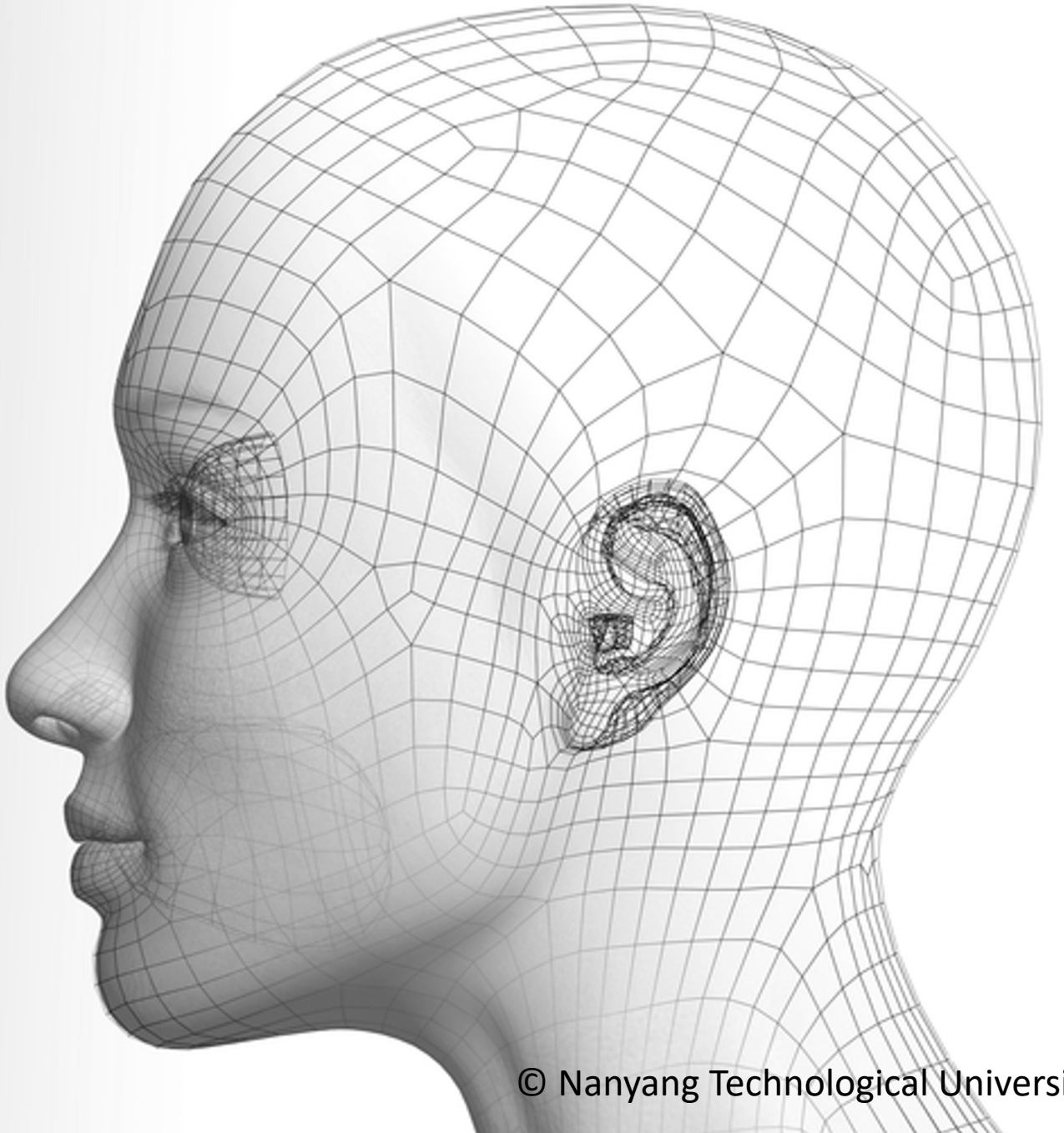
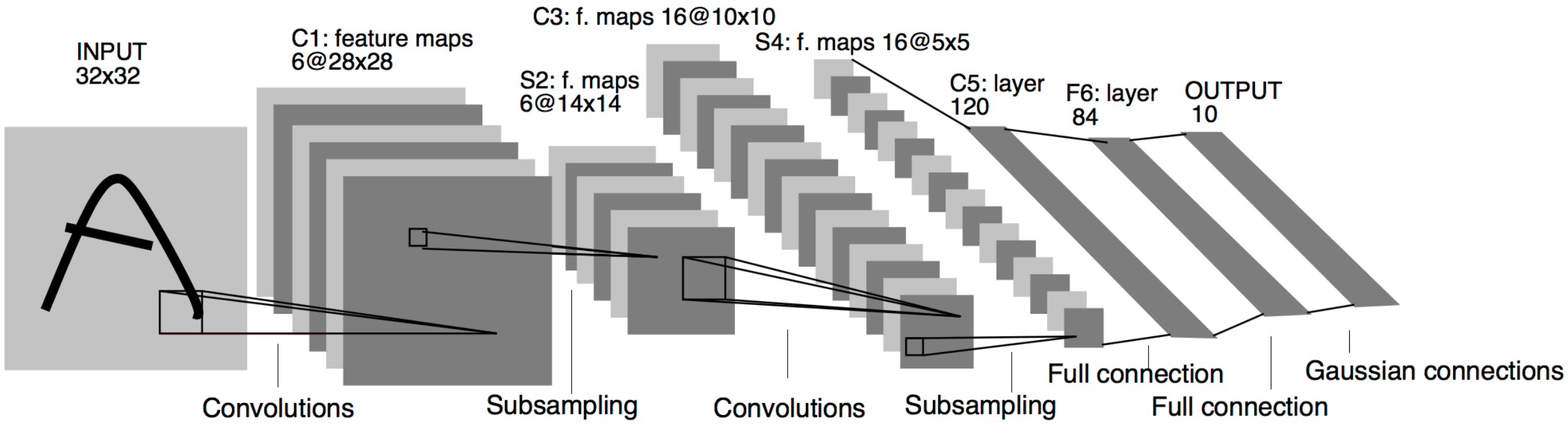
Recurrent Neural Networks (RNN)

Neural networks and deep learning

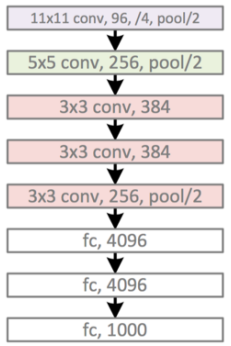


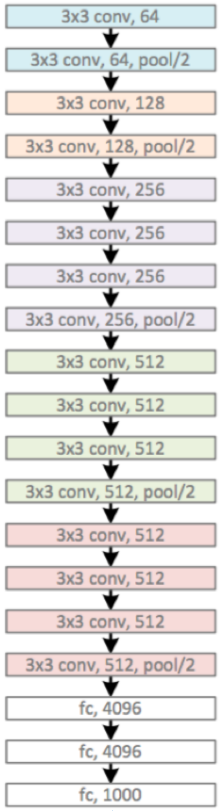
Previous: Convolutional Neural Networks (CNN)

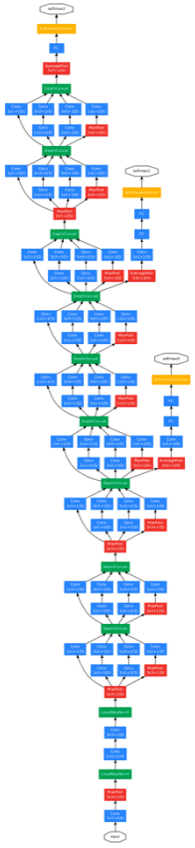


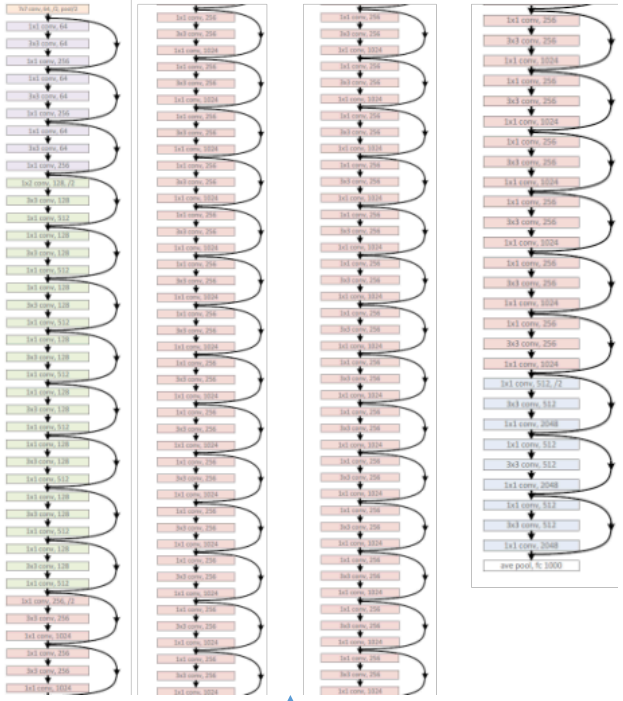
Previous: Convolutional Neural Networks (CNN)

AlexNet 8 layers





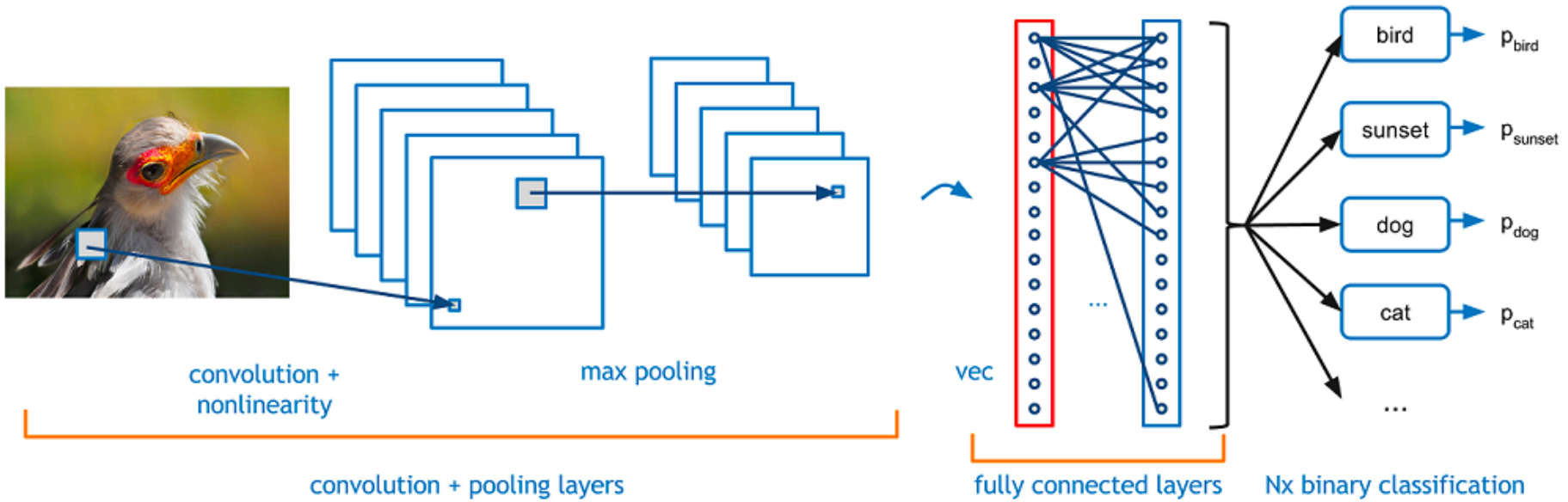




Deeper
More intricate and different connectivity structures

2012 2013 2014 2015 2016 2017 2018 ...

Previous: Convolutional Neural Networks (CNN)



How about sequential information?

- Turn a sequence of sound pressures into a sequence of word identities?
- Speech recognition?
- Video prediction?

Recurrent Neural Networks (RNN)

Recurrent neural networks (RNN) are designed to process sequential information. That is, the data presented in a sequence.

The next data point in the sequence is usually dependent on the current data point.

RNN attempts to capture dependency among the data points in the sequence.

Examples:

- Natural language processing (spoken words and written sentences). The next word in a sentence depends on the word which comes before it.
- Genomic sequences: a nucleotide in a DNA sequence is dependent on its neighbors.

Text Generation with an RNN

QUEENE:
I had thought thou hadst a Roman; for the oracle,
Thus by All bids the man against the word,
Which are so weak of care, by old care done;
Your children were in your holy love,
And the precipitation through the bleeding throne.

BISHOP OF ELY:
Marry, and will, my lord, to weep in such a one were prettiest;
Yet now I was adopted heir
Of the world's lamentable day,
To watch the next way with his father with his face?

Text Generation with an RNN

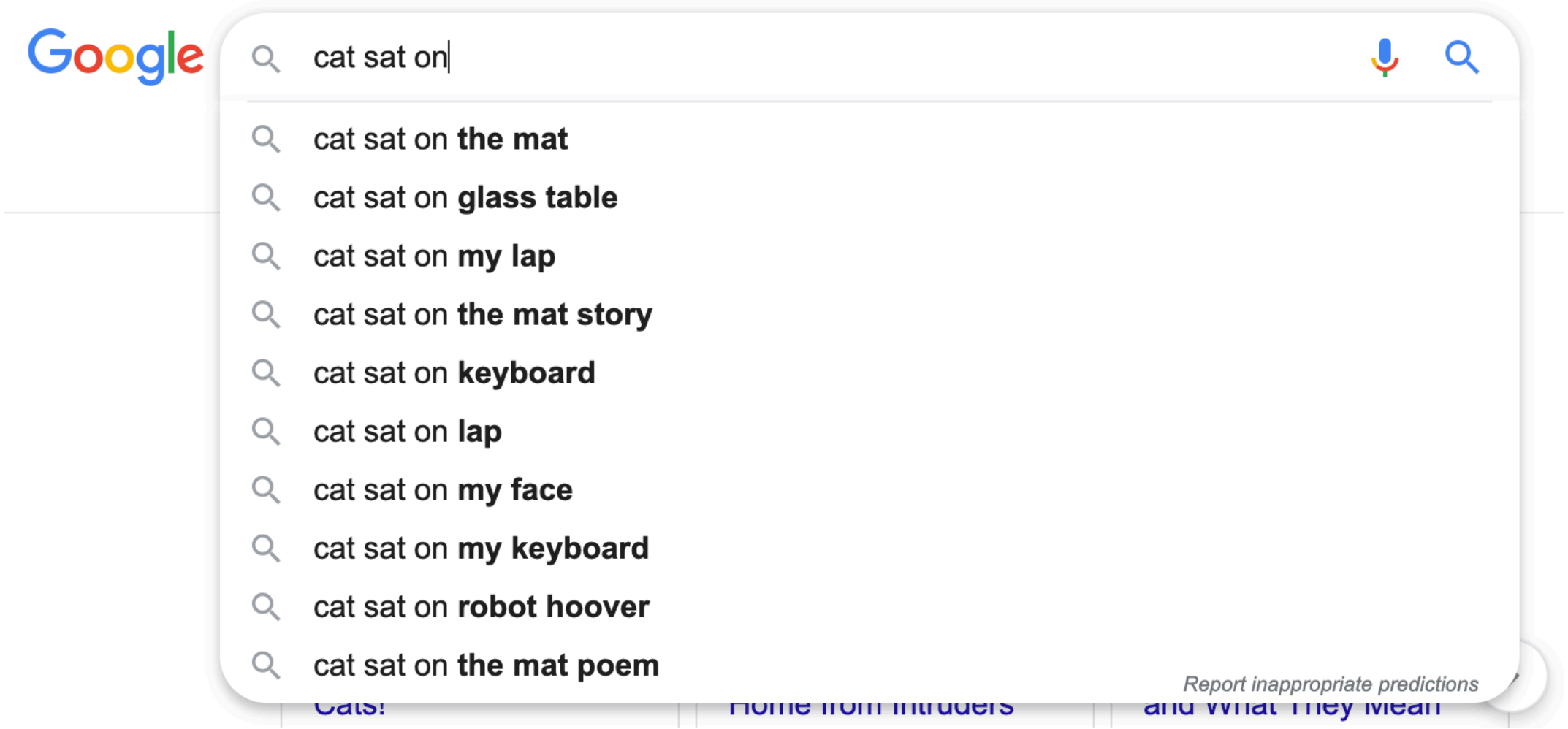
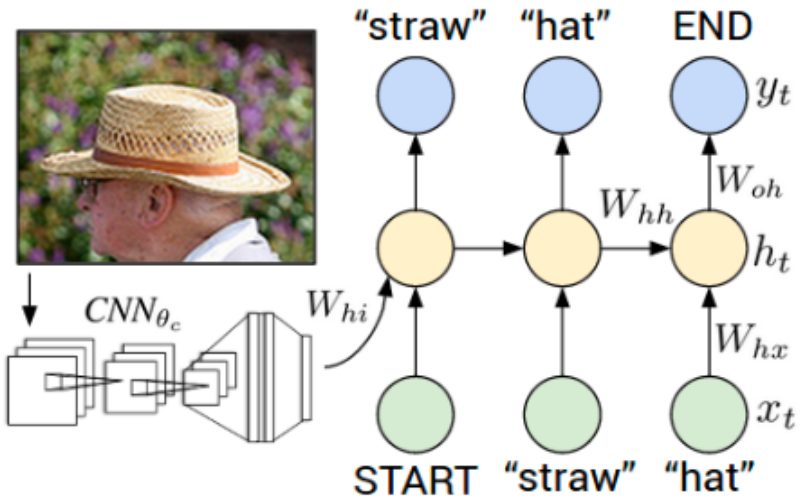


Image Captioning

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."



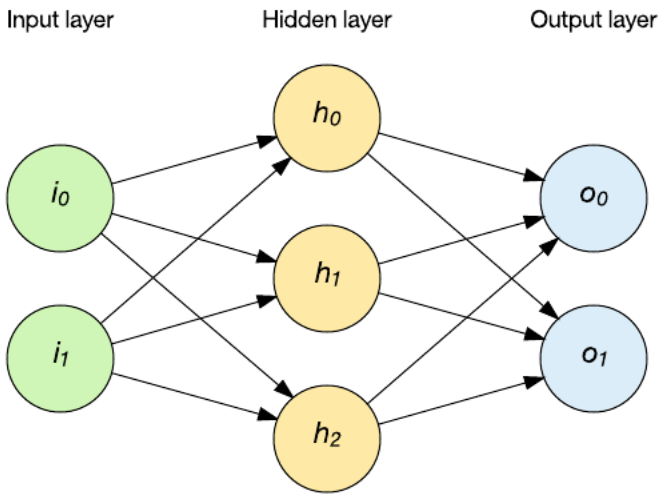
Recurrent Neural Networks (RNN)

RNNs are called recurrent because they perform the same task for every data element (frame) of a sequence, with the output depending on the previous computations.

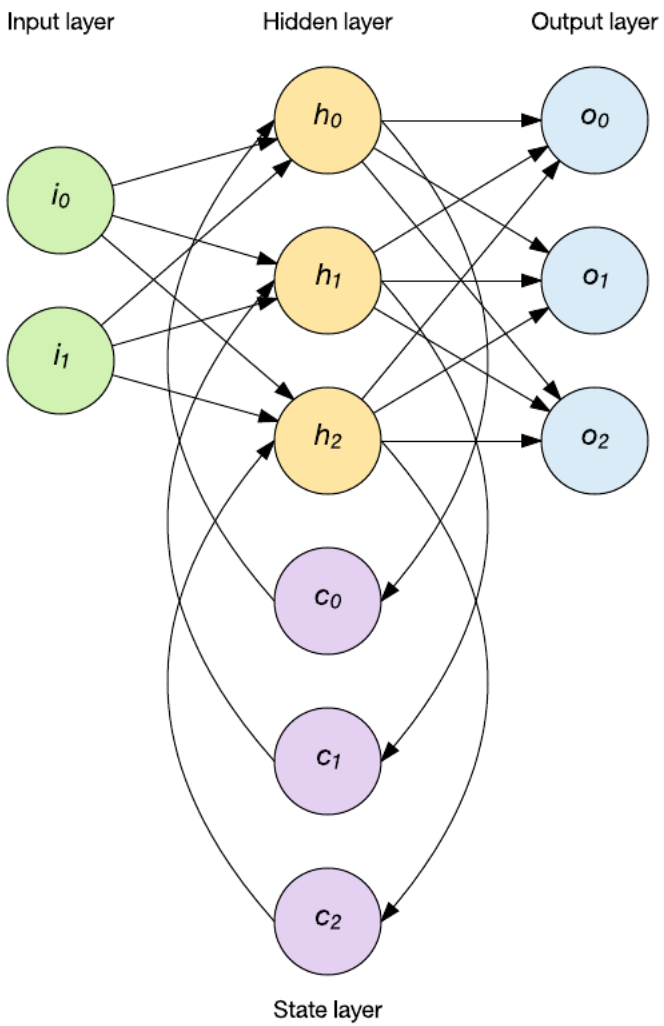
In other words, it has a memory that captures information about what has been processed so far.

RNN achieves its function by using feedback connections that enables learning of sequential (temporal) information of sequences.

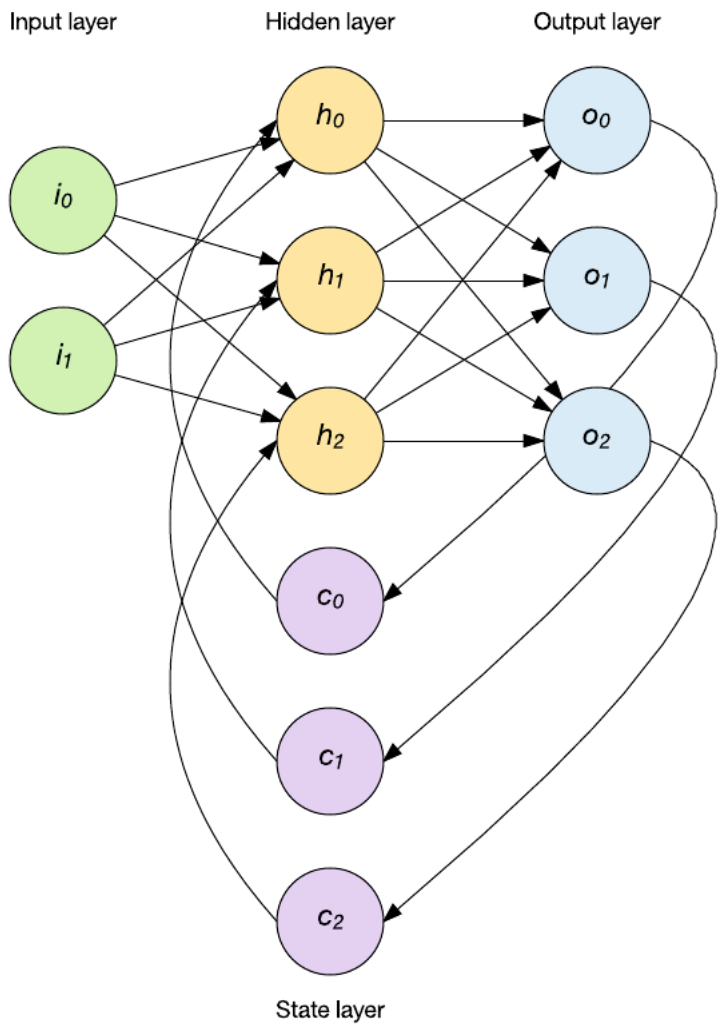
Types of RNN



Feedforward NN



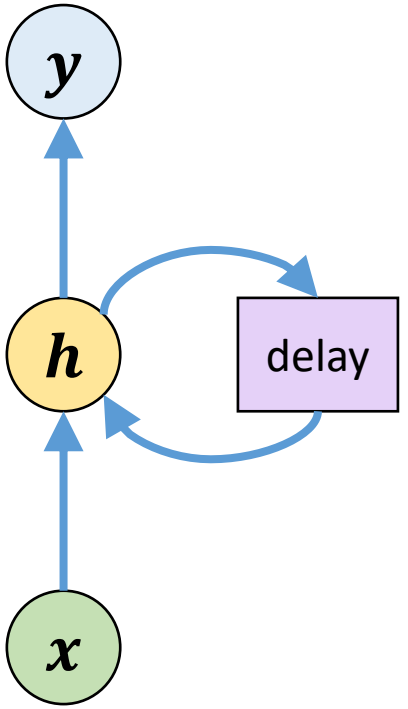
RNN with hidden recurrence
(Elman-type)



RNN with top-down recurrence
(Jordan-type)

RNN with hidden recurrence (Elman type)

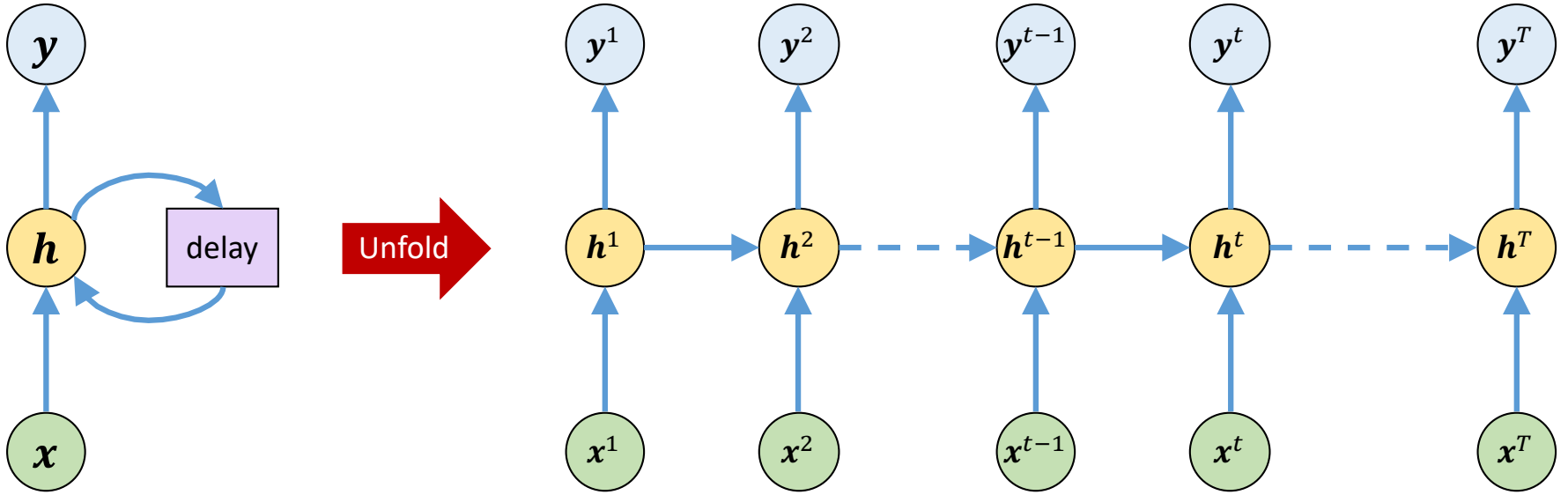
Elman type networks (sometimes called a “Vanilla RNN”) that produce an output at each time step and have recurrent connections between hidden units.



Data is presented as a sequence of instance t (time) that is discretized and activations are updated at each time instant t. The hidden-layer activation at time $t - 1$ is kept by the delay unit and fed to the hidden layer at time t together with the raw input $\boldsymbol{x}(t)$.

The **delay unit** represents that the activation is held for one time unit until the next time instance. Here, one unit represents the time between two adjacent data points in the sequence .

RNN with hidden recurrence

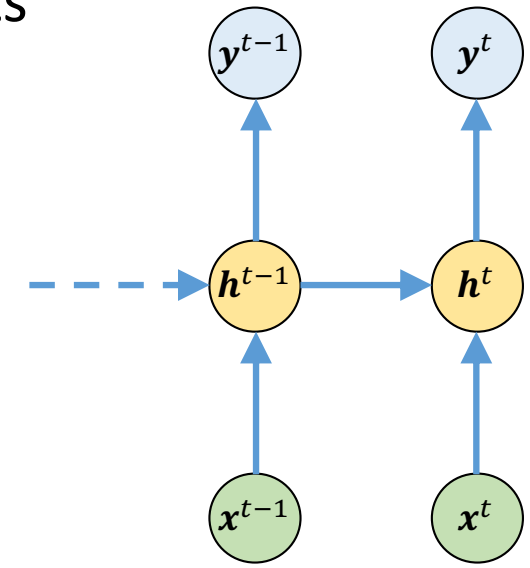


- Note that the purple square in the computational graph indicates that a feedback takes place with a delay of a unit time, from state at time t to the state at time $t + 1$.
- The recurrent connections in the hidden-layer can be unfolded and represented in time for processing a time series $\left(x(t)\right)_{t=1}^{T}$

RNN with hidden recurrence

By considering the unfolded structure, $\boldsymbol{h}(t)$ is dependent on all the inputs before time t:

$$\boldsymbol{h}(t) = f^t(\boldsymbol{x}(t), \boldsymbol{x}(t-1), \dots, \boldsymbol{x}(2), \boldsymbol{x}(1))$$



By using recurrence relationship, we can write:

new state $\boldsymbol{h}(t) = f(\boldsymbol{h}(t-1), \boldsymbol{x}(t))$

some function — old state — input vector at some time step

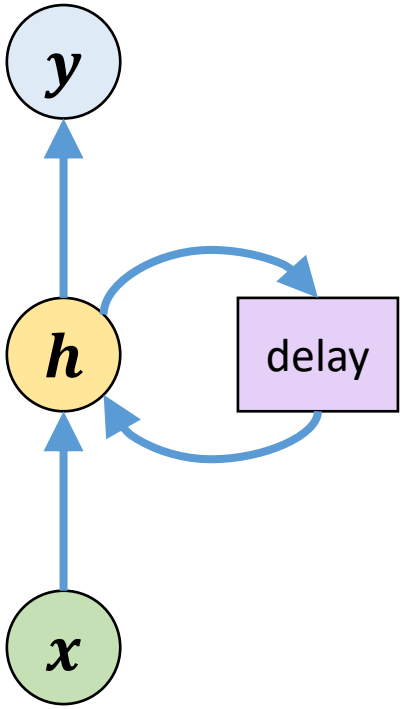
The function f^t takes the whole past sequence ,$(\boldsymbol{x}(t), \boldsymbol{x}(t-1), \dots, \boldsymbol{x}(2), \boldsymbol{x}(1))$ as input and produce the hidden layer activation.

The past inputs are represented by the hidden-layer activations in the previous instant. The recurrent structure allows us to factorize f^t into repeated applications of a function f.

RNN with hidden recurrence

The folded structure introduces two major advantages:

1. Regardless of the sequence length, the learned model always has the same size, rather than specified in terms of a variable-length history of states.
2. It is possible to use same transition function f with the same parameters at every time step.



RNN with hidden recurrence

$\boldsymbol{U}$: weight vector that transforms raw inputs to the hidden-layer

$\boldsymbol{W}$: recurrent weight vector connecting previous hidden-layer output to hidden input

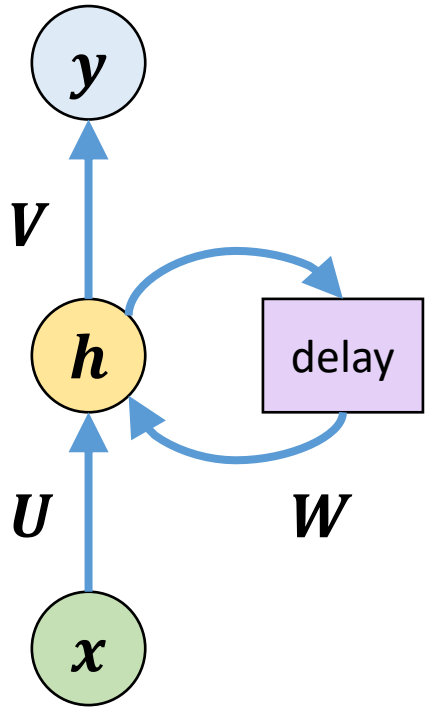
$\boldsymbol{V}$: weight vector of the output layer

$\boldsymbol{b}$: bias connected to hidden layer

$\boldsymbol{c}$: bias connected to the output layer

ϕ: the tanh hidden-layer activation function

σ: the linear/softmax output-layer activation function

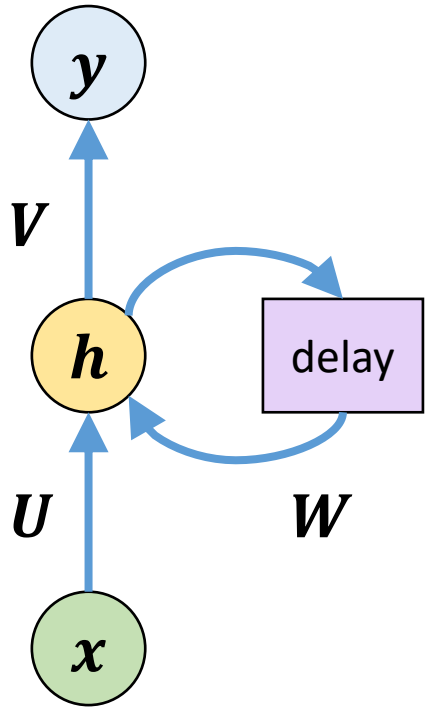


RNN with hidden recurrence

Let $\boldsymbol{x}(t)$, $\boldsymbol{y}(t)$, and $\boldsymbol{h}(t)$ be the input, output, and hidden output of the network at time t.

Activation of the Elman-type RNN with one hidden-layer is given by:

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$
$$\boldsymbol{y}(t) = \sigma\left(\boldsymbol{V}^{\mathsf{T}}\boldsymbol{h}(t) + \boldsymbol{c}\right)$$

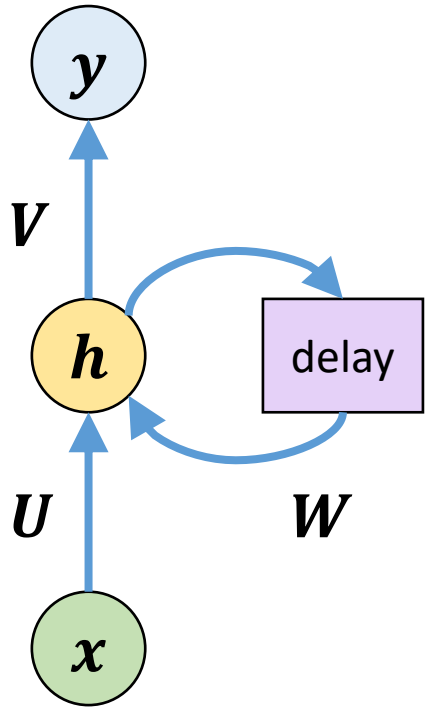


σ is a *softmax* function for classification and a *linear* function for regression.

RNN with hidden recurrence: batch processing

Given P patterns $\{\boldsymbol{x}_p\}_{p=1}^{P}$ where $x_p = \left(\boldsymbol{x}_p(t)\right)_{t=1}^{T}$,

$$\boldsymbol{X}(t) = \begin{pmatrix} \boldsymbol{x}_1(\boldsymbol{t})^{\mathsf{T}} \\ \boldsymbol{x}_2(\boldsymbol{t})^{\mathsf{T}} \\ \vdots \\ \boldsymbol{x}_P(\boldsymbol{t})^{\mathsf{T}} \end{pmatrix}$$



Let $\boldsymbol{X}(t)$, $\boldsymbol{Y}(t)$, and $\boldsymbol{H}(t)$ be batch input, output, and hidden output of the network at time t

Activation of the three-layer Elman-type RNN is given by:

$$\boldsymbol{H}(t) = \phi(\boldsymbol{X}(t)\boldsymbol{U} + \boldsymbol{H}(t-1)\boldsymbol{W} + \boldsymbol{B})$$
$$\boldsymbol{Y}(t) = \sigma(\boldsymbol{H}(t)\boldsymbol{V} + \boldsymbol{C})$$

Example 1

A recurrent neural network with hidden recurrence has two input neurons, three hidden neurons, and two output neurons. The parameters of the network are initialized as $\boldsymbol{U} = \begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix}$, $\boldsymbol{W} = \begin{pmatrix} 2.0 & 1.3 & -1.0 \\ 1.5 & 0.0 & -0.5 \\ -0.2 & 1.5 & 0.4 \end{pmatrix}$ and $\boldsymbol{V} = \begin{pmatrix} 2.0 & -1.0 \\ -1.5 & 0.5 \\ 0.2 & 0.8 \end{pmatrix}$.

Bias to the hidden layer $\boldsymbol{b} = \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \end{pmatrix}$ and to the output layer $\boldsymbol{c} = \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix}$.

For a sequence of inputs $\big(\boldsymbol{x}(1), \boldsymbol{x}(2), \boldsymbol{x}(3), \boldsymbol{x}(4)\big)$ where

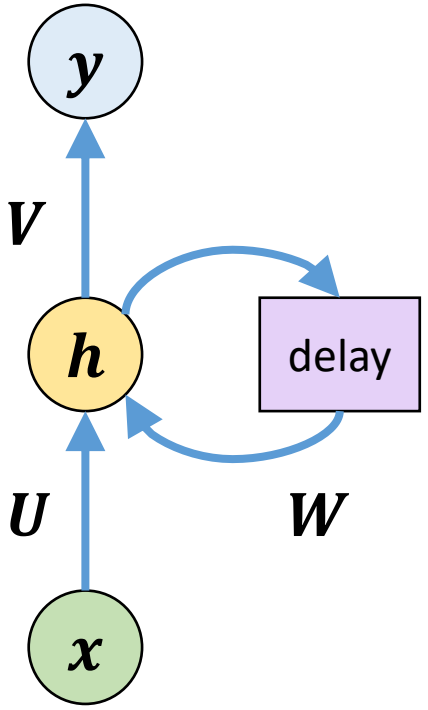
$$\boldsymbol{x}(1) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \boldsymbol{x}(2) = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \boldsymbol{x}(3) = \begin{pmatrix} 0 \\ 3 \end{pmatrix}, \text{ and } \boldsymbol{x}(4) = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$$

find the output of RNN.

Assume that hidden layer activations are initialized to zero and *tanh* and sigmoid functions for the hidden and output layer activation functions, respectively.

Example 1 (con't)

$$\boldsymbol{U} = \begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix}, \quad \boldsymbol{W} = \begin{pmatrix} 2.0 & 1.3 & -1.0 \\ 1.5 & 0.0 & -0.5 \\ -0.2 & 1.5 & 0.4 \end{pmatrix} \text{ and } \boldsymbol{V} = \begin{pmatrix} 2.0 & -1.0 \\ -1.5 & 0.5 \\ 0.2 & 0.8 \end{pmatrix}$$



$$\boldsymbol{b} = \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \end{pmatrix} \text{ and } \boldsymbol{c} = \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix}$$

Three hidden neurons and two output neurons

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$
$$\boldsymbol{y}(t) = \sigma\left(\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{h}(t) + \boldsymbol{c}\right)$$

$$\phi(u) = \tanh(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}}$$
$$\sigma(u) = \text{sigmoid}(u) = \frac{1}{1 + e^{-u}}.$$

Assume $\boldsymbol{h}(0) = (0 \quad 0 \quad 0)^T$.

Example 1 (con't)

At t=1, $\boldsymbol{x}(1) = \begin{pmatrix}1\\2\end{pmatrix}$:

$$\boldsymbol{h}(1) = \phi\left(\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{x}(1) + \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{h}(0) + \boldsymbol{b}\right)$$

$$= \tanh\left(\begin{pmatrix}-1.0 & 0.5\\ 0.5 & 0.1\\ 0.2 & -2.0\end{pmatrix}\begin{pmatrix}1\\2\end{pmatrix} + \begin{pmatrix}2.0 & 1.5 & -0.2\\ 1.3 & 0.0 & 1.5\\ -1.0 & -0.5 & 0.4\end{pmatrix}\begin{pmatrix}0\\0\\0\end{pmatrix} + \begin{pmatrix}0.2\\0.2\\0.2\end{pmatrix}\right)$$

$$= \begin{pmatrix}0.2\\0.72\\-1.0\end{pmatrix}$$

$$\boldsymbol{y}(1) = \sigma\left(\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{h}(1) + \boldsymbol{c}\right)$$

$$= \text{sigmoid}\left(\begin{pmatrix}2.0 & -1.5 & 0.2\\ -1.0 & 0.5 & 0.8\end{pmatrix}\begin{pmatrix}0.2\\0.72\\-1.0\end{pmatrix} + \begin{pmatrix}0.5\\0.1\end{pmatrix}\right)$$

$$= \begin{pmatrix}0.41\\0.37\end{pmatrix}$$

Example 1 (con't)

At t=2, $\boldsymbol{x}(2) = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$:

$$\boldsymbol{h}(2) = \phi\left(\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{x}(2) + \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{h}(1) + \boldsymbol{b}\right)$$

$$= \tanh\left(\begin{pmatrix} -1.0 & 0.5 \\ 0.5 & 0.1 \\ 0.2 & -2.0 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \end{pmatrix} + \begin{pmatrix} 2.0 & 1.5 & -0.2 \\ 1.3 & 0.0 & 1.5 \\ -1.0 & -0.5 & 0.4 \end{pmatrix}\begin{pmatrix} 0.2 \\ 0.72 \\ -1.0 \end{pmatrix} + \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 1.0 \\ -0.89 \\ -0.99 \end{pmatrix}$$

$$\boldsymbol{y}(2) = \sigma\left(\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{h}(2) + \boldsymbol{c}\right)$$

$$= \text{sigmoid}\left(\begin{pmatrix} 2.0 & -1.5 & 0.2 \\ -1.0 & 0.5 & 0.8 \end{pmatrix}\begin{pmatrix} 1.0 \\ -0.89 \\ -0.99 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix}\right) = \begin{pmatrix} 0.97 \\ 0.11 \end{pmatrix}$$

Example 1 (con't)

Similarly,

at t=3, $\boldsymbol{x}(3) = \begin{pmatrix} 0 \\ 3 \end{pmatrix}$; $\boldsymbol{h}(3) \begin{pmatrix} 0.99 \\ 0.3 \\ -1.0 \end{pmatrix}$ and $\boldsymbol{y}(3) = \begin{pmatrix} 0.86 \\ 0.18 \end{pmatrix}$

at t=4, $\boldsymbol{x}(4) = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$; $\boldsymbol{h}(4) \begin{pmatrix} 0.31 \\ 0.71 \\ 0.79 \end{pmatrix}$ and $\boldsymbol{y}(4) = \begin{pmatrix} 0.55 \\ 0.68 \end{pmatrix}$

The output is $\big(\boldsymbol{y}(1), \boldsymbol{y}(2), \boldsymbol{y}(3), \boldsymbol{y}(4)\big)$ where

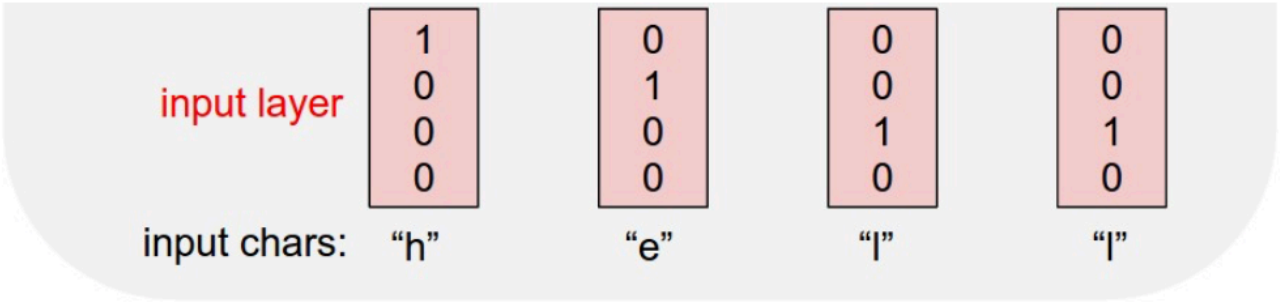
$$\boldsymbol{y}(1) = \begin{pmatrix} 0.41 \\ 0.37 \end{pmatrix}, \boldsymbol{y}(2) = \begin{pmatrix} 0.97 \\ 0.11 \end{pmatrix}, \boldsymbol{y}(3) = \begin{pmatrix} 0.86 \\ 0.18 \end{pmatrix} \boldsymbol{y}(4) = \begin{pmatrix} 0.55 \\ 0.68 \end{pmatrix}$$

An example – Languange modelling

Given characters 1, 2, …, t,
model predicts character t

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



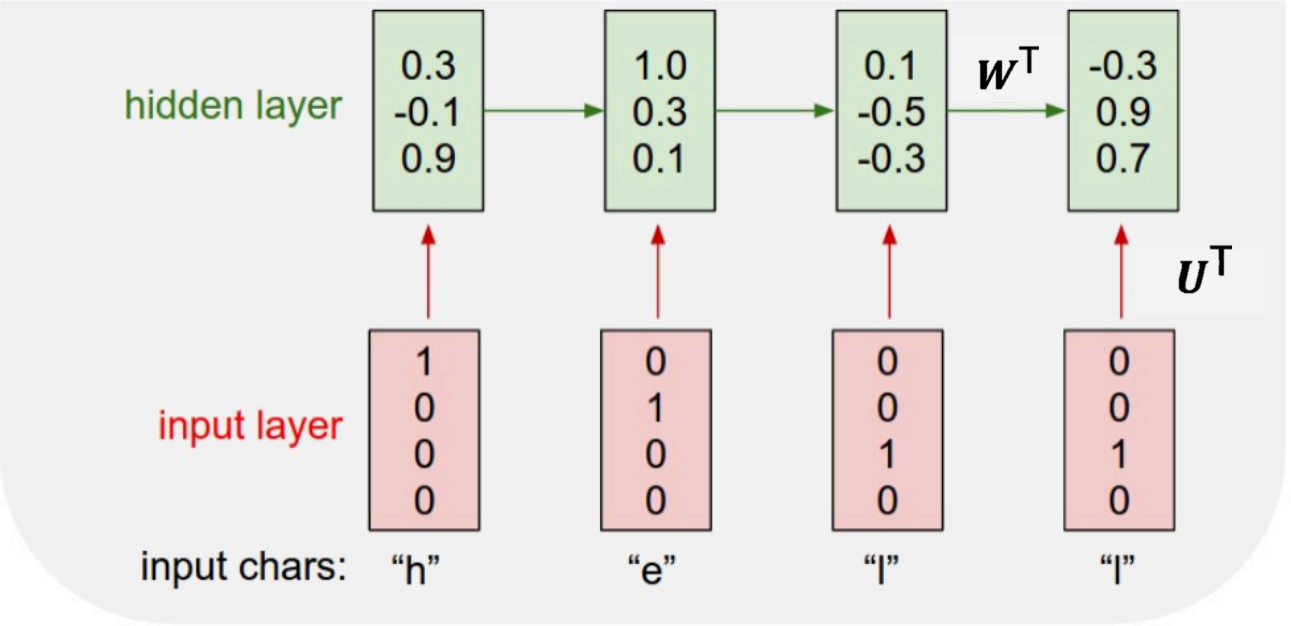
An example – Languange modelling

Given characters 1, 2, …, t,
model predicts character t

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



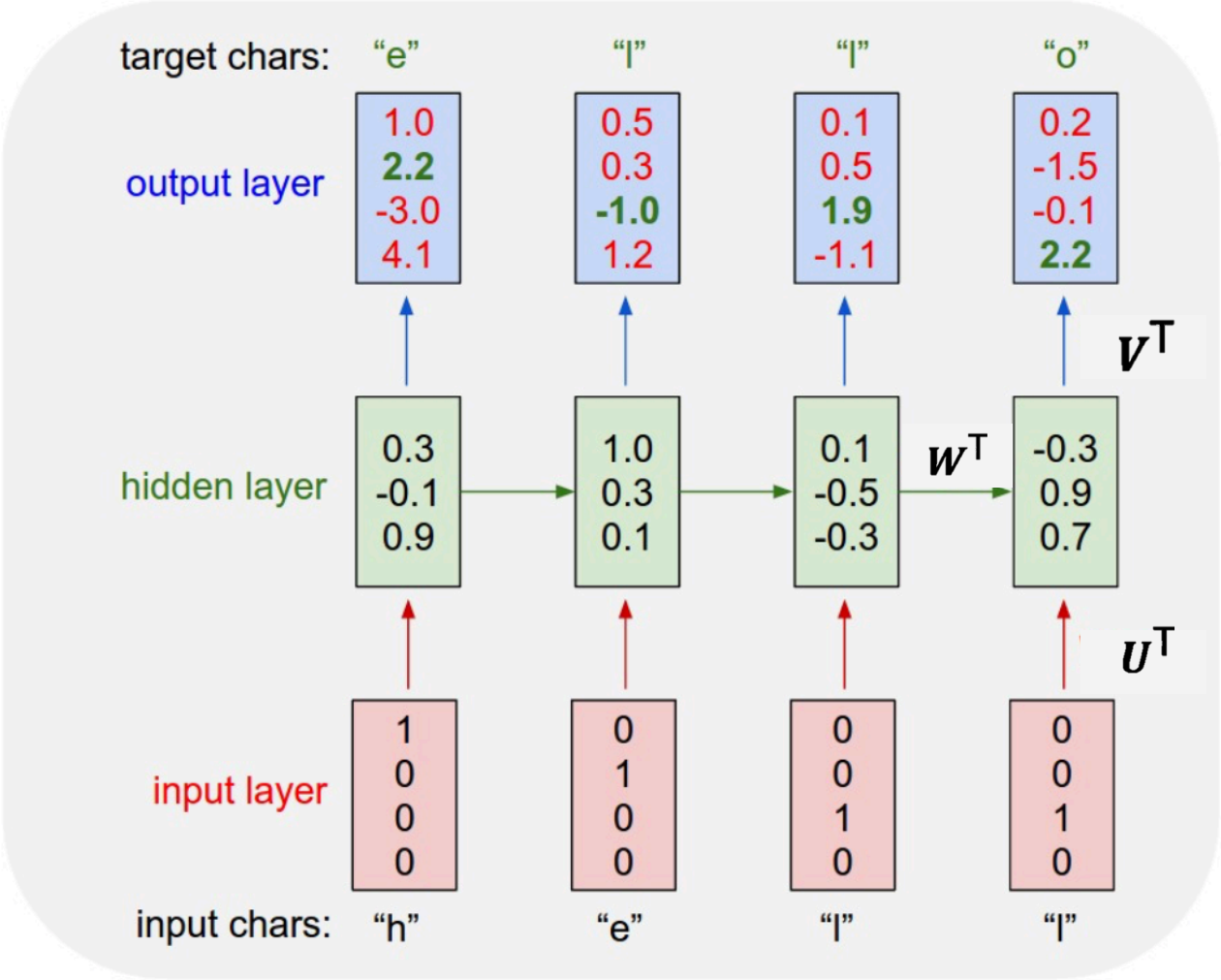
An example – Languange modelling

Given characters 1, 2, …, t,
model predicts character t

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



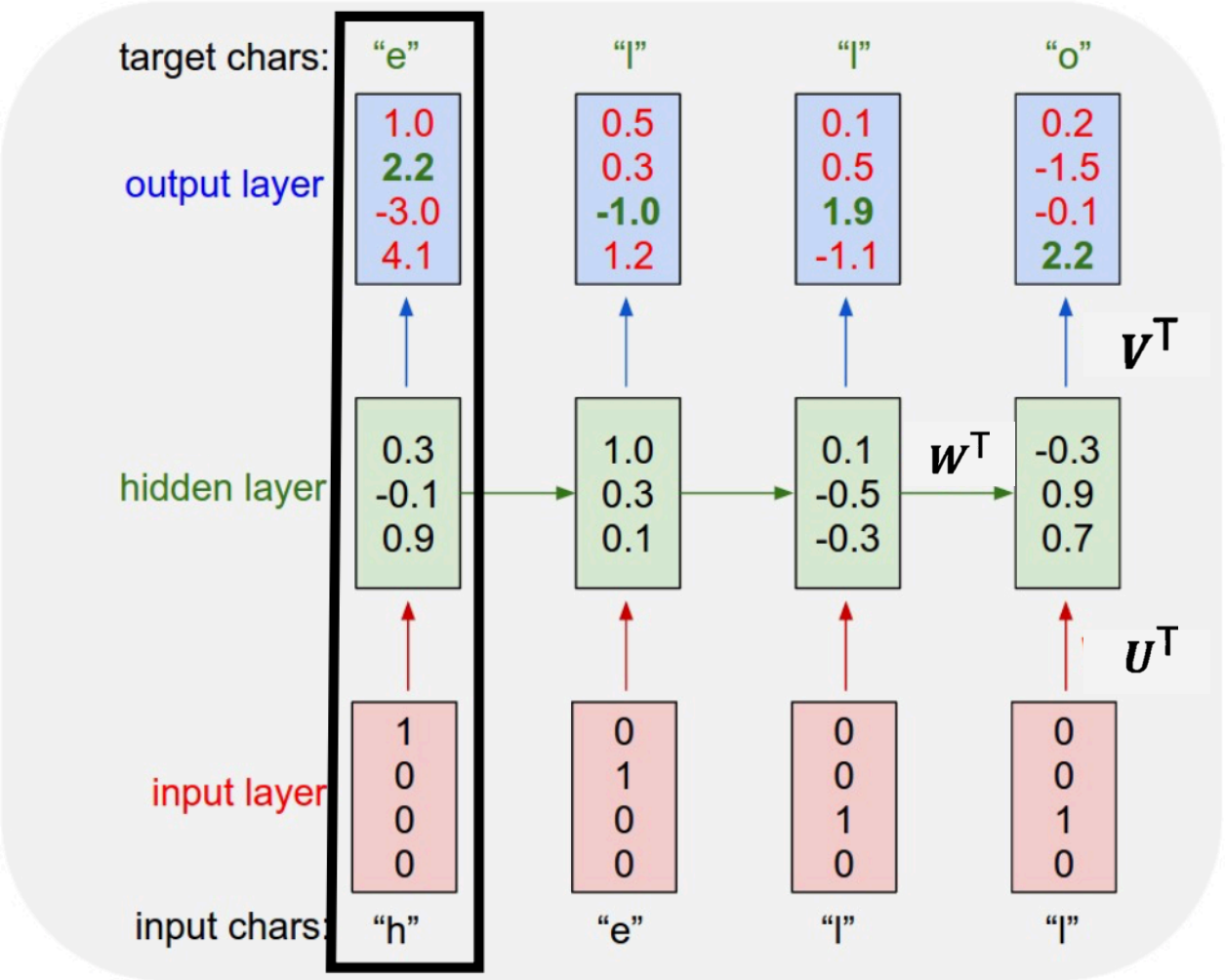
An example – Languange modelling

Given characters 1, 2, …, t,
model predicts character t

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



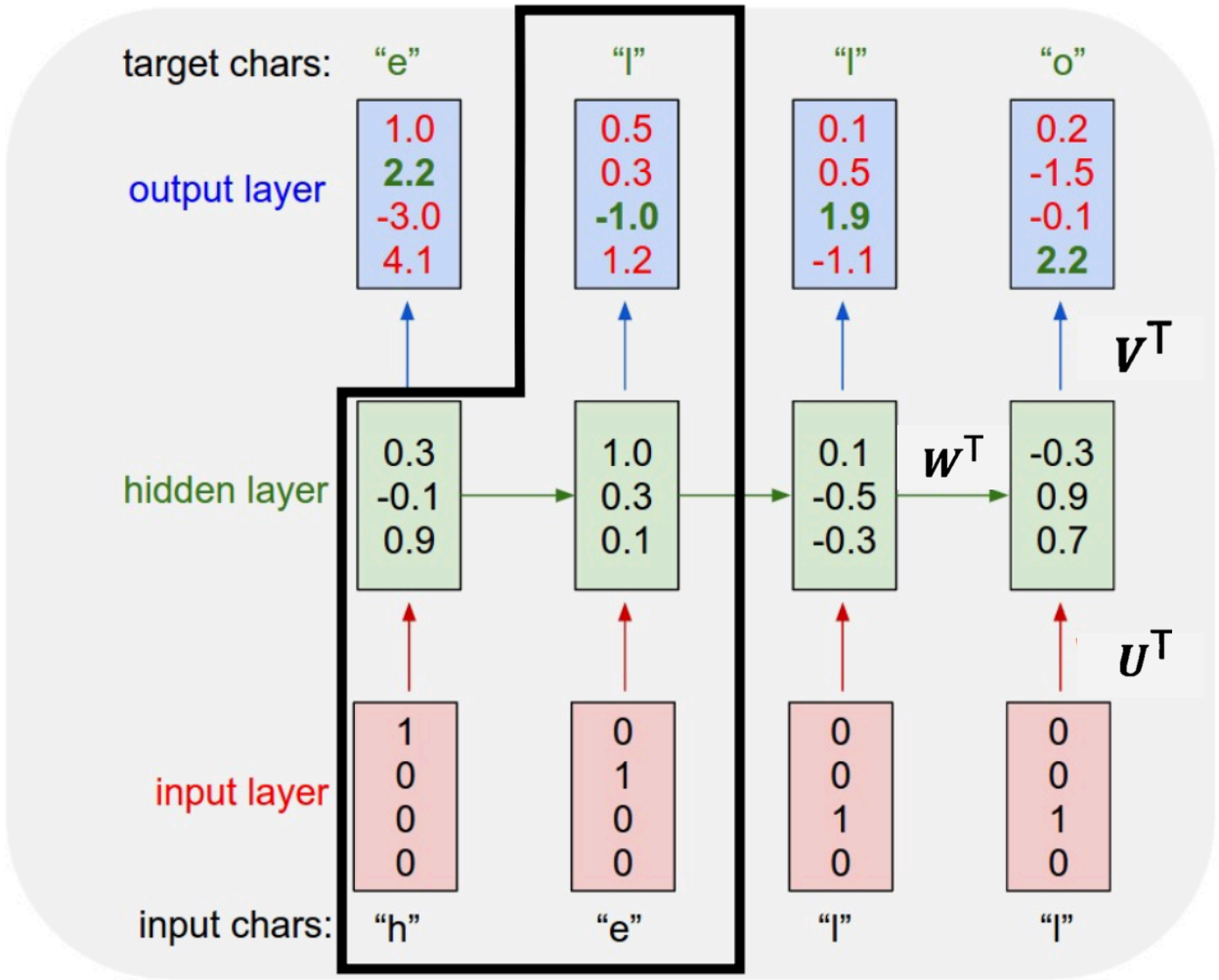
An example – Languange modelling

Given characters 1, 2, …, t,
model predicts character t

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



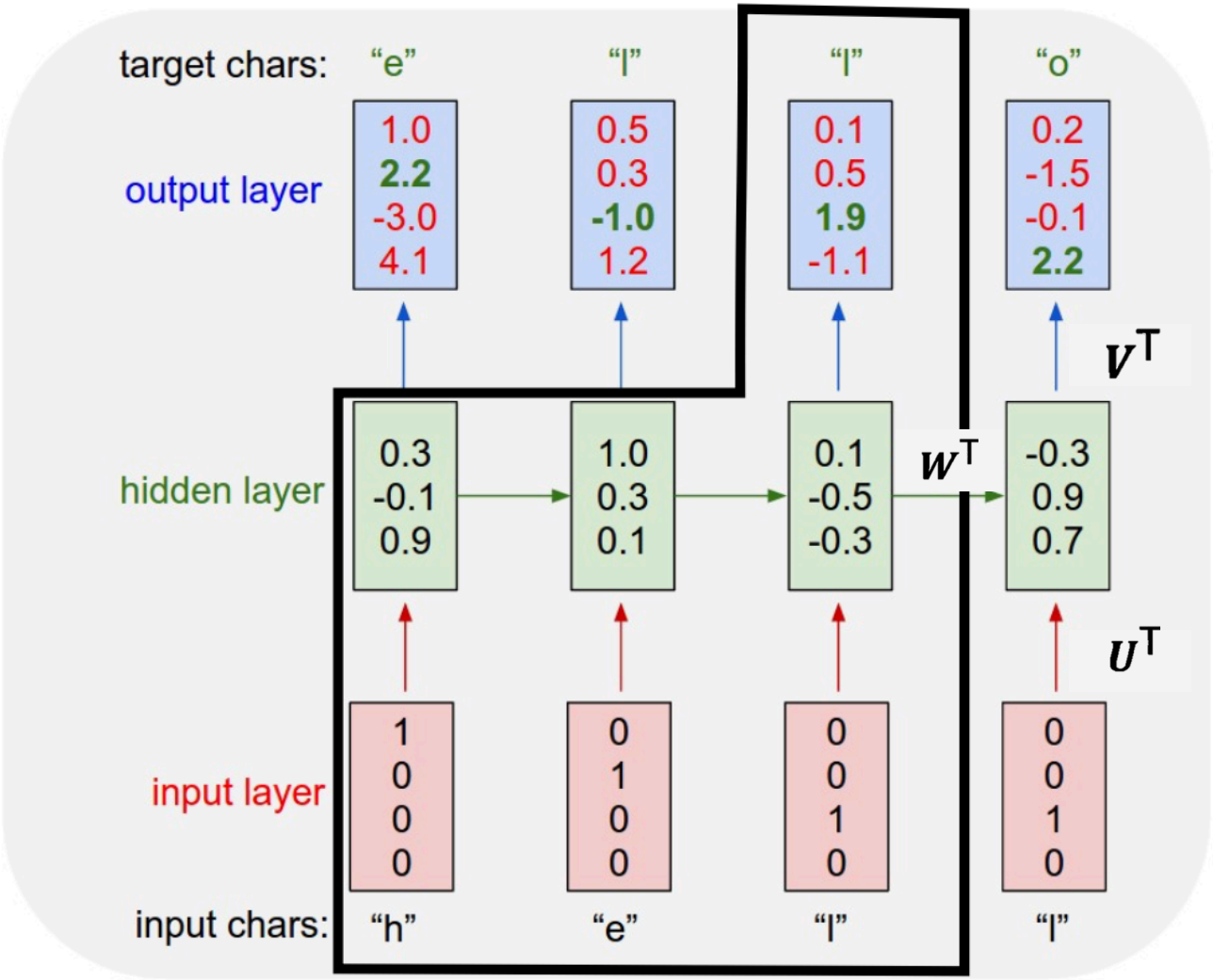
An example – Languange modelling

Given characters 1, 2, …, t,
model predicts character t

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



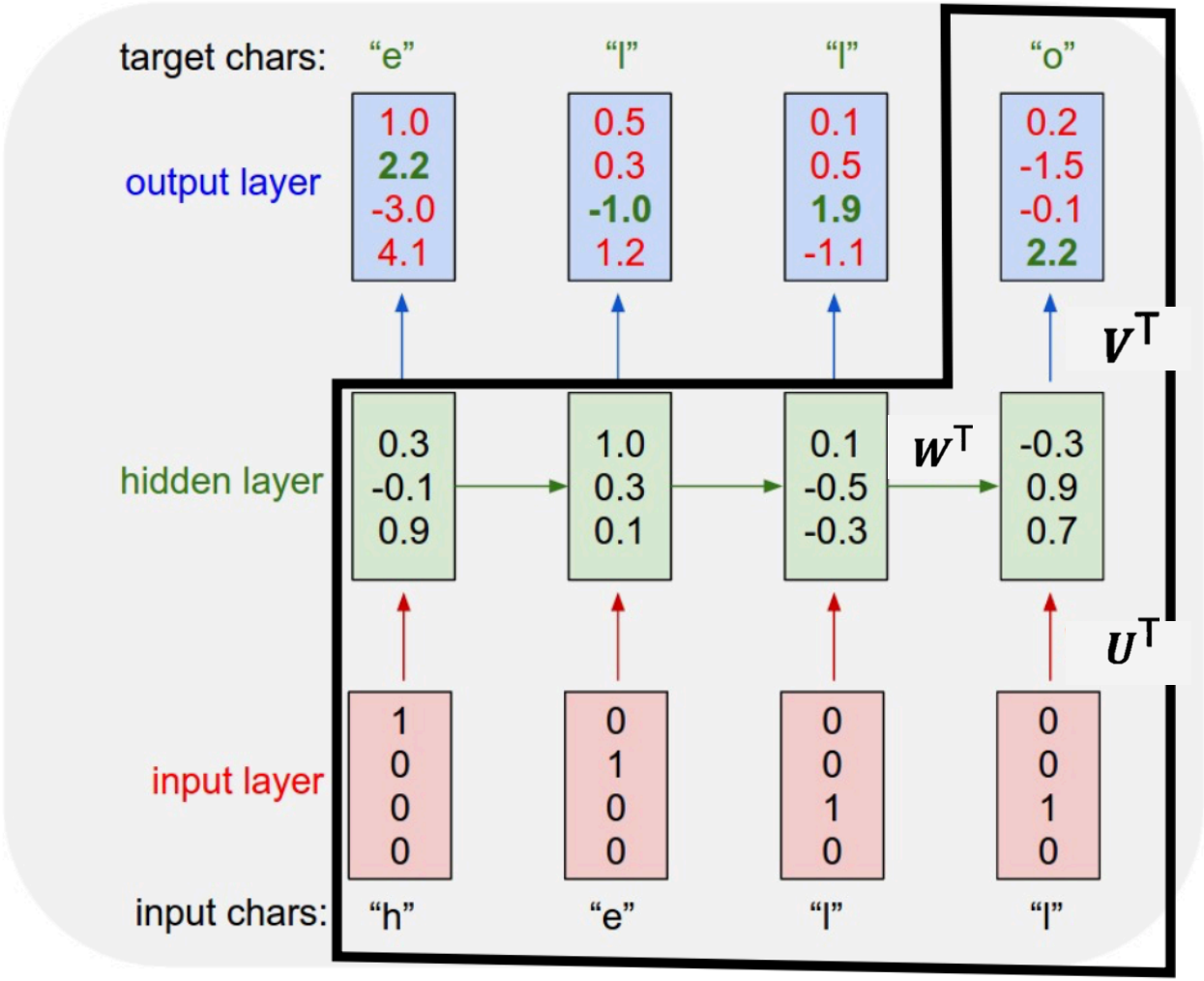
An example – Languange modelling

Given characters 1, 2, …, t,
model predicts character t

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



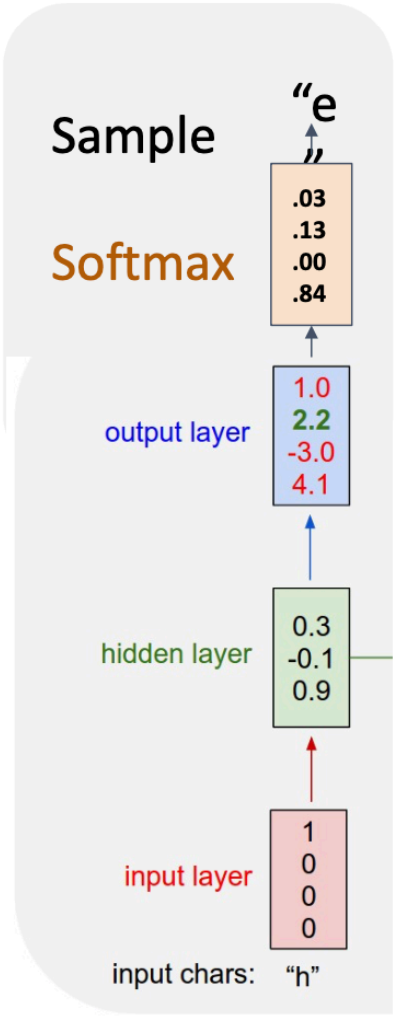
An example – Languange modelling

Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



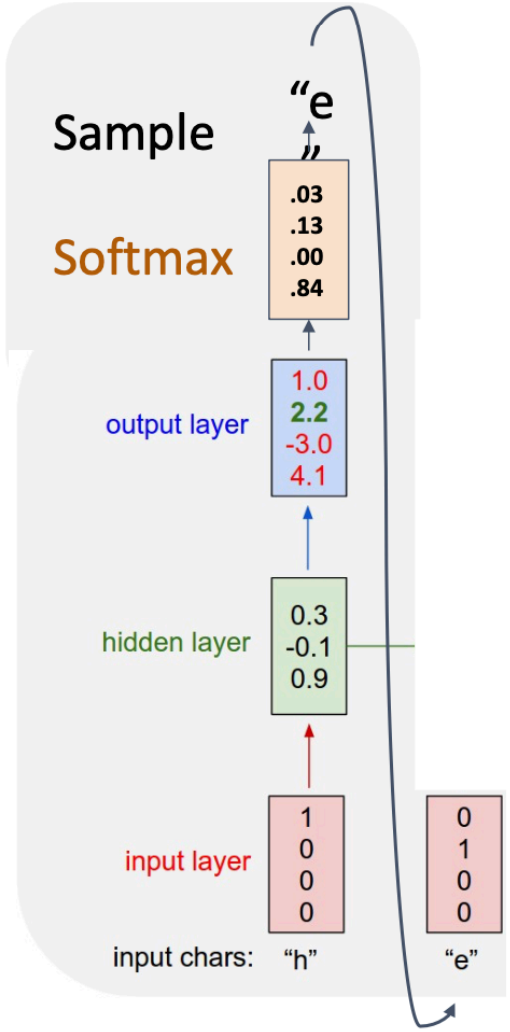
An example – Languange modelling

Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



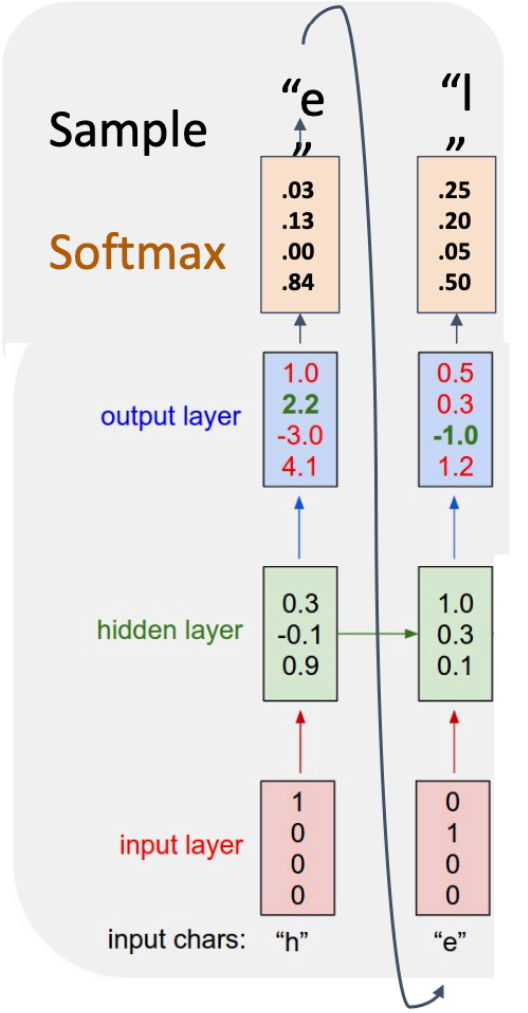
An example – Languange modelling

Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



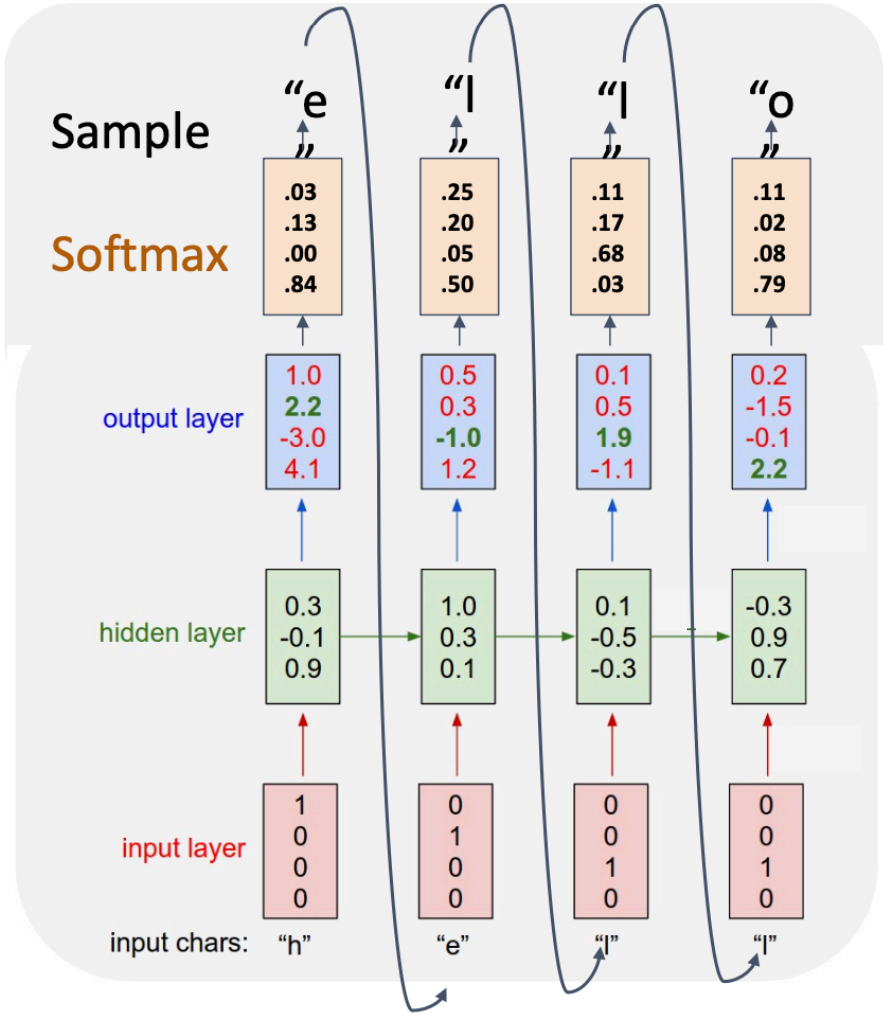
An example – Languange modelling

Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



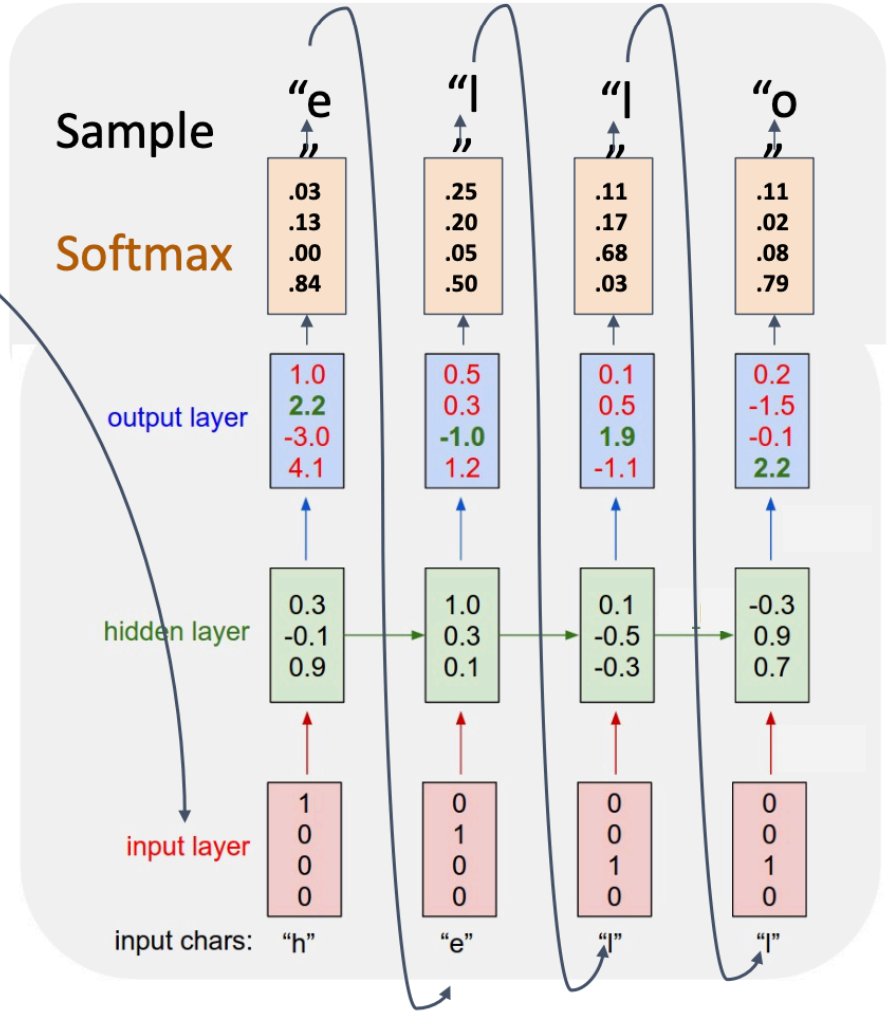
An example – Languange modelling

Example: Language Modeling

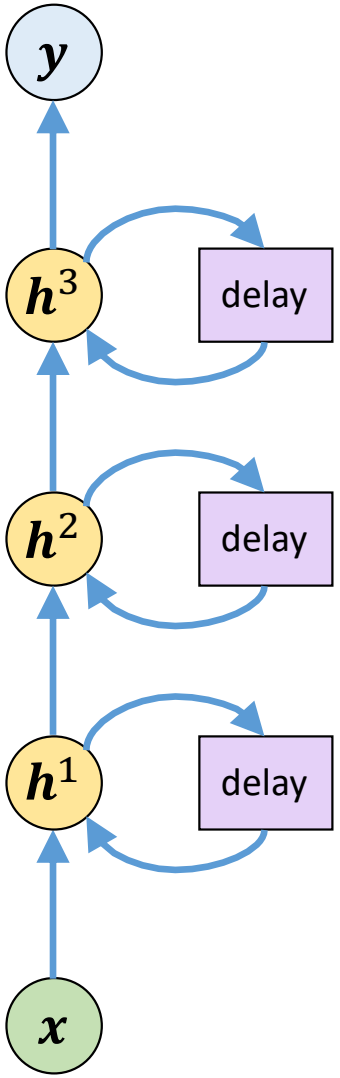
So far: encode inputs as **one-hot-vector**

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. Often extract this into a separate **embedding** layer



Deep RNN with hidden recurrence



$\boldsymbol{U}^l$: weight matrix connecting output of hidden-layer $l - 1$ to hidden-layer l.
$\boldsymbol{W}^l$: weight matrix connecting hidden-layer output of the previous instant to the current output of the hidden-layer.
$\boldsymbol{b}^l$: the bias vector connected to the layer l

First hidden-layer $l = 1$:

$$\boldsymbol{h}^1(t) = \phi\left(\boldsymbol{U}^{1\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{1\mathsf{T}}\boldsymbol{h}^1(t-1) + \boldsymbol{b}^1\right)$$

For layer $l = 2, \ldots, L-1$:

$$\boldsymbol{h}^l(t) = \phi\left(\boldsymbol{U}^{l\mathsf{T}}\boldsymbol{h}^{l-1}(t) + \boldsymbol{W}^{l\mathsf{T}}\boldsymbol{h}^l(t-1) + \boldsymbol{b}^l\right)$$

First output layer $l = L$:

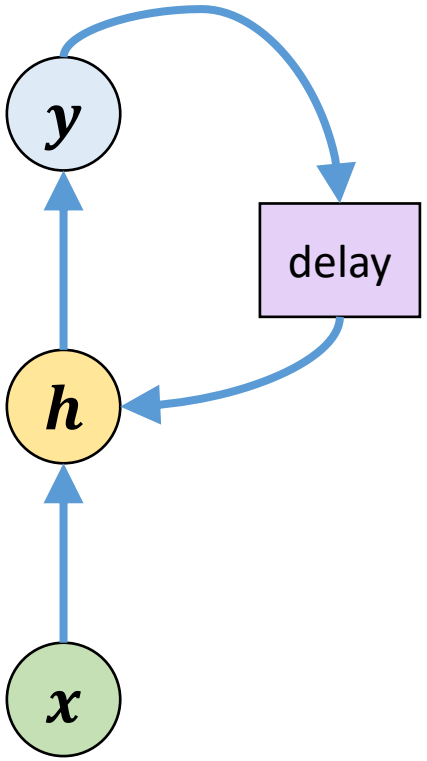
$$\boldsymbol{y}(t) = \sigma\left(\boldsymbol{U}^{L\mathsf{T}}\boldsymbol{h}^{L-1}(t) + \boldsymbol{b}^L\right)$$

RNN with top-down recurrence (Jordan type)

In Jordan-type RNN (that is, RNN with top-down recurrence), the output of the output-layer at time $t - 1$ is kept and fed to the hidden layer at time t together with the raw input $x(t)$ for time t.

Note that time t is assumed to be discretized and activations are updated at each time instant t.

The **delay unit** is to indicate that the output is held until the next time instance and fed to the hidden layer.



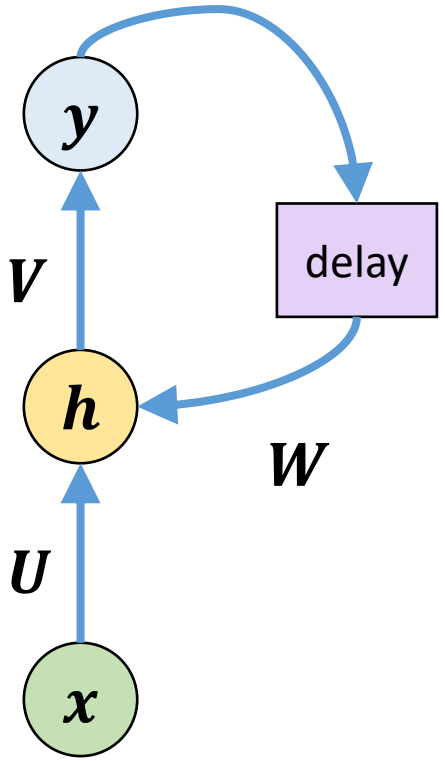
RNN with top-down recurrence

Let $\boldsymbol{x}(t)$, $\boldsymbol{y}(t)$, and $\boldsymbol{h}(t)$ be the input, output, and hidden output of the network at time t.

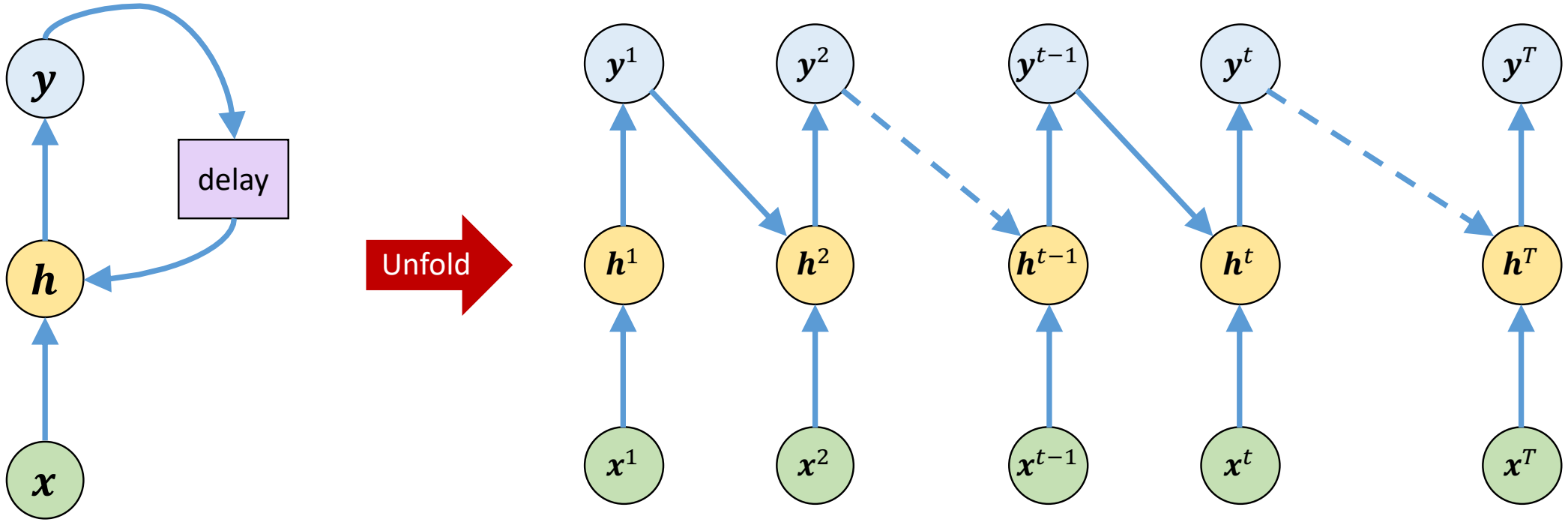
Activation of the Jordan-type RNN with one hidden-layer is given by:

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{y}(t-1) + \boldsymbol{b}\right)$$
$$\boldsymbol{y}(t) = \sigma\left(\boldsymbol{V}^{\mathsf{T}}\boldsymbol{h}(t) + \boldsymbol{c}\right)$$

Note that output of the previous time instant is fed back to the hidden layer and $\boldsymbol{W}$ represents the recurrent weight matrix connecting previous output to the current hidden input

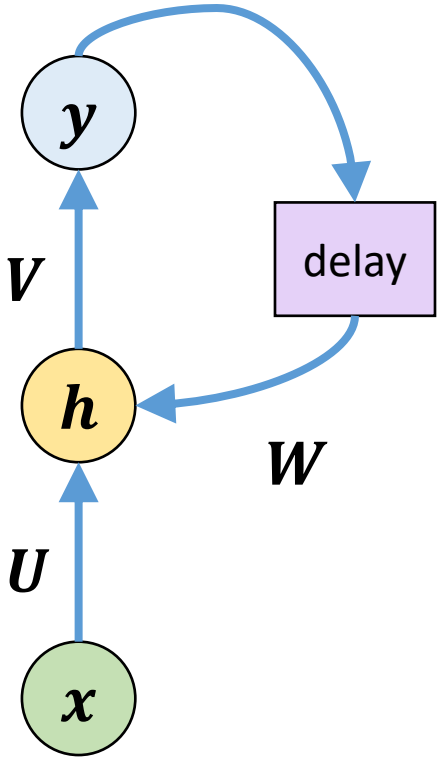


RNN with top-down recurrence



The recurrent connections in the hidden-layer is unfolded and represented in time for processing a time series $\left(x(t)\right)_{t=1}^{T}$

RNN with top-down recurrence: batch processing



Given P patterns $\{\boldsymbol{x}_p\}_{p=1}^{P}$ where $x_p = \left(\boldsymbol{x}_p(t)\right)_{t=1}^{T}$,

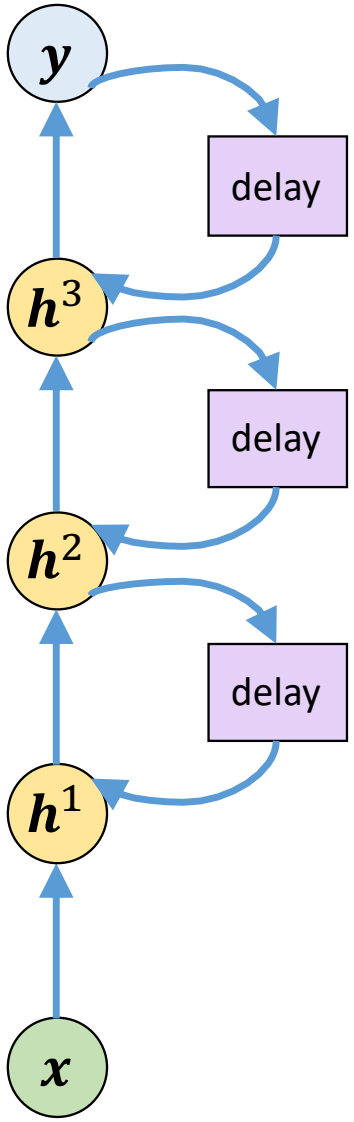
$$\boldsymbol{X}(t) = \begin{pmatrix} \boldsymbol{x}_1(\boldsymbol{t})^{\mathsf{T}} \\ \boldsymbol{x}_2(\boldsymbol{t})^{\mathsf{T}} \\ \vdots \\ \boldsymbol{x}_P(\boldsymbol{t})^{\mathsf{T}} \end{pmatrix}$$

Let $\boldsymbol{X}(t)$, $\boldsymbol{Y}(t)$, and $\boldsymbol{H}(t)$ be batch input, output, and hidden output of the network at time t

Activation of the three-layer Jordan-type RNN is given by:

$$\boldsymbol{H}(t) = \phi(\boldsymbol{X}(t)\boldsymbol{U} + \boldsymbol{Y}(t-1)\boldsymbol{W} + \boldsymbol{B})$$
$$\boldsymbol{Y}(t) = \sigma(\boldsymbol{H}(t)\boldsymbol{V} + \boldsymbol{C})$$

Deep RNN with top-down recurrence



$\boldsymbol{U}^l$: weight matrix connecting output of hidden-layer $l - 1$ to hidden-layer l.
$\boldsymbol{W}^l$: weight matrix connecting hidden-layer output of the previous instant to the current output of the hidden-layer.
$\boldsymbol{b}^l$: the bias vector connected to the layer l

First hidden-layer $l = 1$:

$$\boldsymbol{h}^1(t) = \phi\left(\boldsymbol{U}^{1\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{1\mathsf{T}}\boldsymbol{h}^2(t-1) + \boldsymbol{b}^1\right)$$

For hidden-layer $l = 2, \ldots, L-2$:

$$\boldsymbol{h}^l(t) = \phi\left(\boldsymbol{U}^{l\mathsf{T}}\boldsymbol{h}^{l-1}(t) + \boldsymbol{W}^{l\mathsf{T}}\boldsymbol{h}^{l+1}(t-1) + \boldsymbol{b}^l\right)$$

For hidden-layer $l = L-1$:

$$\boldsymbol{h}^{L-1}(t) = \phi\left(\boldsymbol{U}^{L-1\mathsf{T}}\boldsymbol{h}^{L-2}(t) + \boldsymbol{W}^{L-1\mathsf{T}}\boldsymbol{y}(t-1) + \boldsymbol{b}^{L-1}\right)$$

First output layer $l = L$:

$$\boldsymbol{y}(t) = \sigma\left(\boldsymbol{U}^{L\mathsf{T}}\boldsymbol{h}^{L-1}(t) + \boldsymbol{b}^L\right)$$

Example 2

A recurrent neural network with top-down recurrence receives 2-dimensional input sequences and produce 1-dimensional output sequences. It has three hidden neurons and following weight matrices and biases:

Weight matrix connecting input the hidden layer $\boldsymbol{U} = \begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix}$

Recurrence weight matrix connecting previous output to the hidden layer $\boldsymbol{W} = \begin{pmatrix} 2.0 & 1.3 & -1.0 \end{pmatrix}$

Weight matrix connecting hidden layer to the output $\boldsymbol{V} = \begin{pmatrix} 2.0 \\ -1.5 \\ 0.2 \end{pmatrix}$

Bias to the hidden layer $\boldsymbol{b} = \begin{pmatrix} 0.2 \\ 0.2 \\ 0.2 \end{pmatrix}$ and bias to the output layer $\boldsymbol{c} = 0.1$.

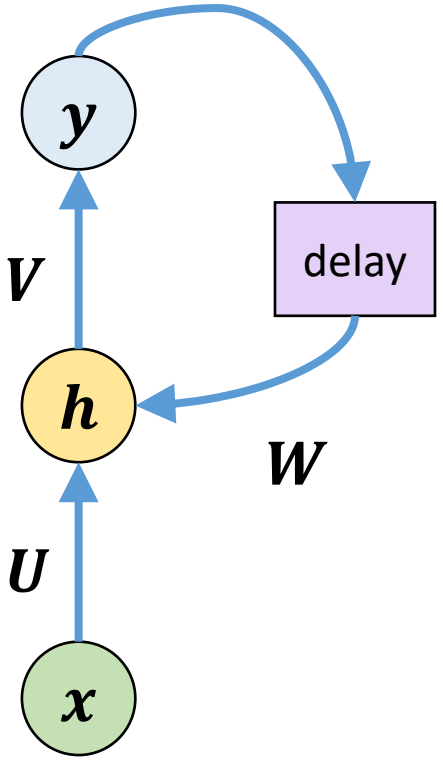
Given the following two input sequences:

$$\boldsymbol{x}_1 = \left(\begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad \begin{pmatrix} -1 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 3 \end{pmatrix} \right)$$

$$\boldsymbol{x}_2 = \left(\begin{pmatrix} -1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 2 \\ -1 \end{pmatrix} \quad \begin{pmatrix} 3 \\ -1 \end{pmatrix} \right)$$

Using batch processing, find output sequences. Assume outputs are initialized to zero at the beginning. Assume tanh and sigmoid activations for hidden and output layer, respectlvely.

Example 2 (con't)



$$\boldsymbol{x}_1 = \left(\begin{pmatrix}1\\2\end{pmatrix} \quad \begin{pmatrix}-1\\1\end{pmatrix} \quad \begin{pmatrix}0\\3\end{pmatrix}\right)$$
$$\boldsymbol{x}_2 = \left(\begin{pmatrix}-1\\0\end{pmatrix} \quad \begin{pmatrix}2\\-1\end{pmatrix} \quad \begin{pmatrix}3\\-1\end{pmatrix}\right)$$

Two sequences as batch of sequences:

$$\boldsymbol{X} = \left(\begin{pmatrix}1 & 2\\-1 & 0\end{pmatrix} \quad \begin{pmatrix}-1 & 1\\2 & -1\end{pmatrix} \quad \begin{pmatrix}0 & 3\\3 & -1\end{pmatrix}\right)$$

Three hidden neurons and one output neuron.

$$\boldsymbol{H}(t) = \phi(\boldsymbol{X}(t)\boldsymbol{U} + \boldsymbol{Y}(t-1)\boldsymbol{W} + \boldsymbol{B})$$
$$\boldsymbol{Y}(t) = \sigma(\boldsymbol{H}(t)\boldsymbol{V} + \boldsymbol{C})$$

Initially,

$$\boldsymbol{Y}(0) = \begin{pmatrix}0.0\\0.0\end{pmatrix}$$

Example 2 (con't)

$$\text{At } t = 1 : \boldsymbol{X}(1) = \begin{pmatrix} 1 & 2 \\ -1 & 0 \end{pmatrix}$$

$$\begin{aligned} \boldsymbol{H}(1) &= tanh(\boldsymbol{X}(1)\boldsymbol{U} + \boldsymbol{Y}(0)\boldsymbol{W} + \boldsymbol{B}) \\ &= tanh\left(\begin{pmatrix} 1 & 2 \\ -1 & 0 \end{pmatrix}\begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix} + \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}\begin{pmatrix} 2.0 & 1.3 & -1.0 \end{pmatrix} + \begin{pmatrix} 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 \end{pmatrix}\right) \\ &= \begin{pmatrix} 0.2 & 0.72 & -1.0 \\ 0.83 & -0.29 & 0.0 \end{pmatrix} \end{aligned}$$

$$\begin{aligned} \boldsymbol{Y}(1) &= sigmoid(\boldsymbol{H}(1)\boldsymbol{V} + \boldsymbol{C}) \\ &= sigmoid\left(\begin{pmatrix} 0.2 & 0.72 & -1.0 \\ 0.83 & -0.29 & 0.0 \end{pmatrix}\begin{pmatrix} 2.0 \\ -1.5 \\ 0.2 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix}\right) \\ &= \begin{pmatrix} 0.31 \\ 0.90 \end{pmatrix} \end{aligned}$$

Example 2 (con't)

$$\text{At } t = 2 : \boldsymbol{X}(2) = \begin{pmatrix} -1 & 1 \\ 2 & -1 \end{pmatrix}$$

$$\boldsymbol{H}(2) = tanh(\boldsymbol{X}(2)\boldsymbol{U} + \boldsymbol{Y}(1)\boldsymbol{W} + \boldsymbol{B})$$

$$= tanh\left(\begin{pmatrix} -1 & 1 \\ 2 & -1 \end{pmatrix}\begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix} + \begin{pmatrix} 0.31 \\ 0.9 \end{pmatrix}\begin{pmatrix} 2.0 & 1.3 & -1.0 \end{pmatrix} + \begin{pmatrix} 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 0.98 & 0.21 & -0.98 \\ -0.46 & 0.98 & 0.94 \end{pmatrix}$$

$$\boldsymbol{Y}(2) = sigmoid(\boldsymbol{H}(2)\boldsymbol{V} + \boldsymbol{C})$$

$$= sigmoid\left(\begin{pmatrix} 0.98 & 0.21 & -0.98 \\ -0.46 & 0.98 & 0.94 \end{pmatrix}\begin{pmatrix} 2.0 \\ -1.5 \\ 0.2 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 0.83 \\ 0.11 \end{pmatrix}$$

Example 2 (con't)

$$\text{At } t = 3 : \boldsymbol{X}(3) = \begin{pmatrix} 0 & 3 \\ 3 & -1 \end{pmatrix}$$

$$\boldsymbol{H}(3) = tanh(\boldsymbol{X}(3)\boldsymbol{U} + \boldsymbol{Y}(1)\boldsymbol{W} + \boldsymbol{B})$$

$$= tanh\left(\begin{pmatrix} 0 & 3 \\ 3 & -1 \end{pmatrix}\begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix} + \begin{pmatrix} 0.83 \\ 0.11 \end{pmatrix}\begin{pmatrix} 2.0 & 1.3 & -1.0 \end{pmatrix} + \begin{pmatrix} 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 1.0 & 0.92 & -1.0 \\ -1.00 & 0.94 & 0.99 \end{pmatrix}$$

$$\boldsymbol{Y}(3) = sigmoid(\boldsymbol{H}(3)\boldsymbol{V} + \boldsymbol{C})$$

$$= sigmoid\left(\begin{pmatrix} 1.0 & 0.92 & -1.0 \\ -1.0 & 0.94 & 0.99 \end{pmatrix}\begin{pmatrix} 2.0 \\ -1.5 \\ 0.2 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 0.63 \\ 0.04 \end{pmatrix}$$

Example 2 (con't)

Output (batch):

$$\boldsymbol{Y} = \begin{pmatrix} \boldsymbol{Y}(1) & \boldsymbol{Y}(2) & \boldsymbol{Y}(3) \end{pmatrix}$$
$$= \left(\begin{pmatrix} 0.31 \\ 0.9 \end{pmatrix} \quad \begin{pmatrix} 0.83 \\ 0.11 \end{pmatrix} \quad \begin{pmatrix} 0.63 \\ 0.04 \end{pmatrix} \right)$$

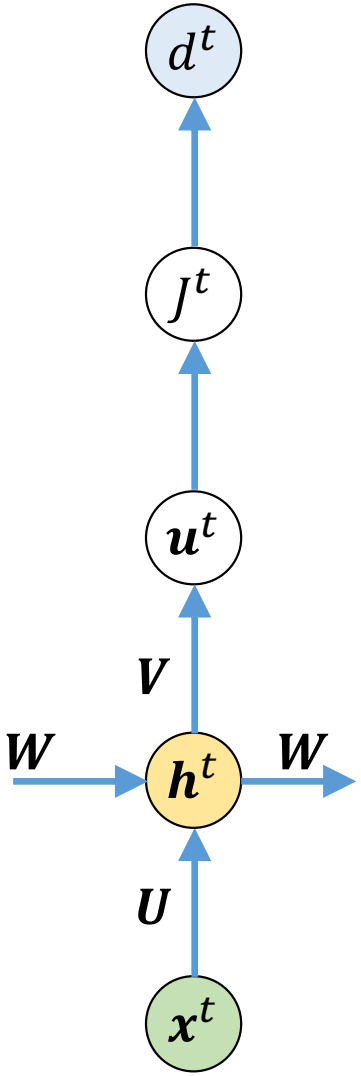
Outputs for inputs

$$x_1 = \left(\begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad \begin{pmatrix} -1 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 3 \end{pmatrix} \right)$$
$$x_2 = \left(\begin{pmatrix} -1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 2 \\ -1 \end{pmatrix} \quad \begin{pmatrix} 3 \\ -1 \end{pmatrix} \right)$$

are

$$y_1 = (0.31, 0.83, 0.63)$$
$$y_2 = (0.9, 0.11, 0.04)$$

Backpropagation through time (BPTT)



Let's consider vanilla-RNN with hidden recurrence.
Forward propagation equations:

$$\boldsymbol{h}(t) = tanh\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\mathsf{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$
$$\boldsymbol{u}(t) = \boldsymbol{V}^{\mathsf{T}}\boldsymbol{h}(t) + \boldsymbol{c}$$

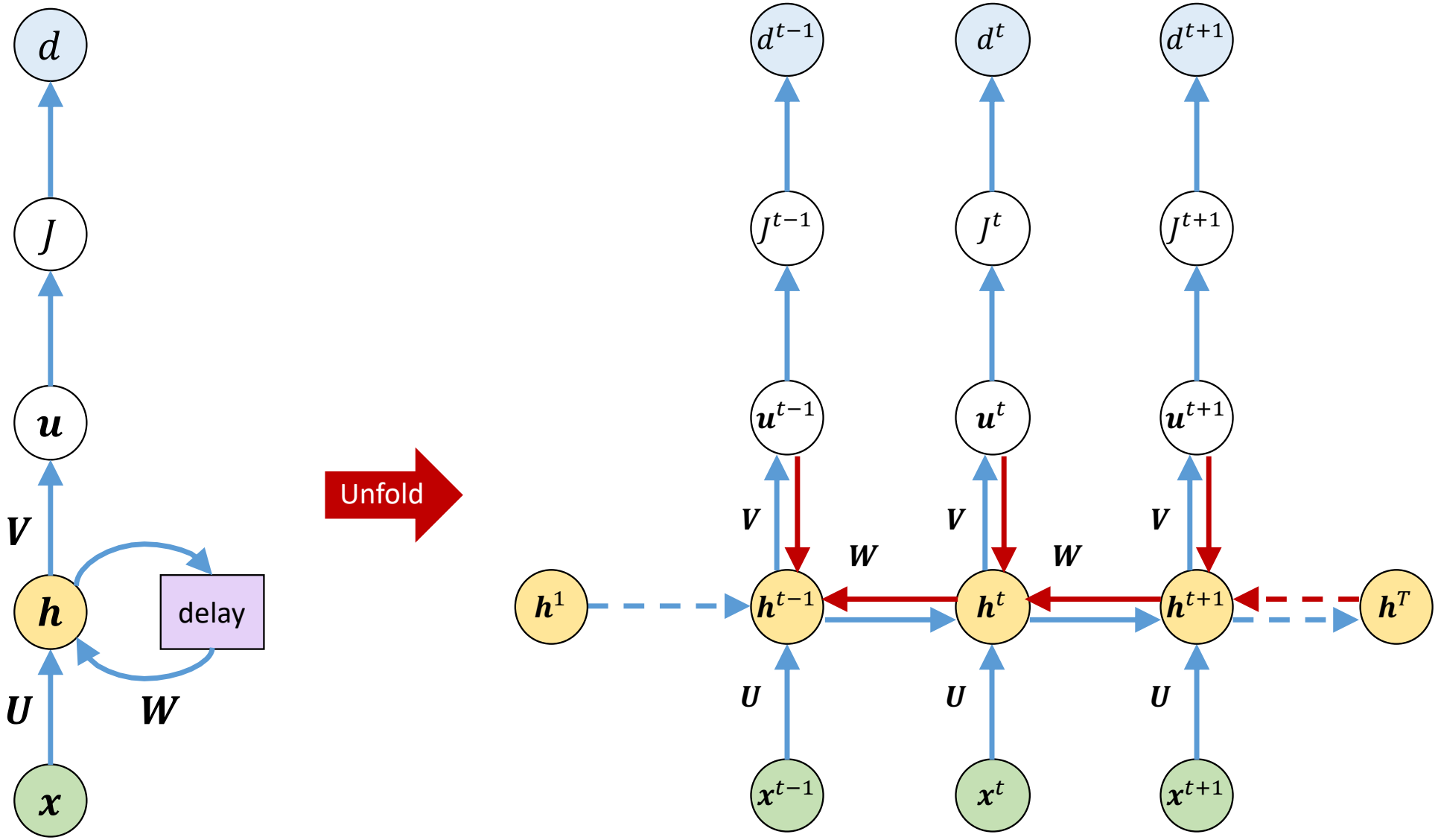
For classification,

$$\boldsymbol{y}(t) = softmax\left(\boldsymbol{u}(t)\right)$$

For regression,

$$\boldsymbol{y}(t) = \boldsymbol{u}(t)$$

Backpropagation through time (BPTT)



Backpropagation through time (BPTT)

Total cost:

$$J = \sum_{t=1}^{T} J(t)$$

where T is the length of the sequence and $J(t)$ represent the cost at the instant t.

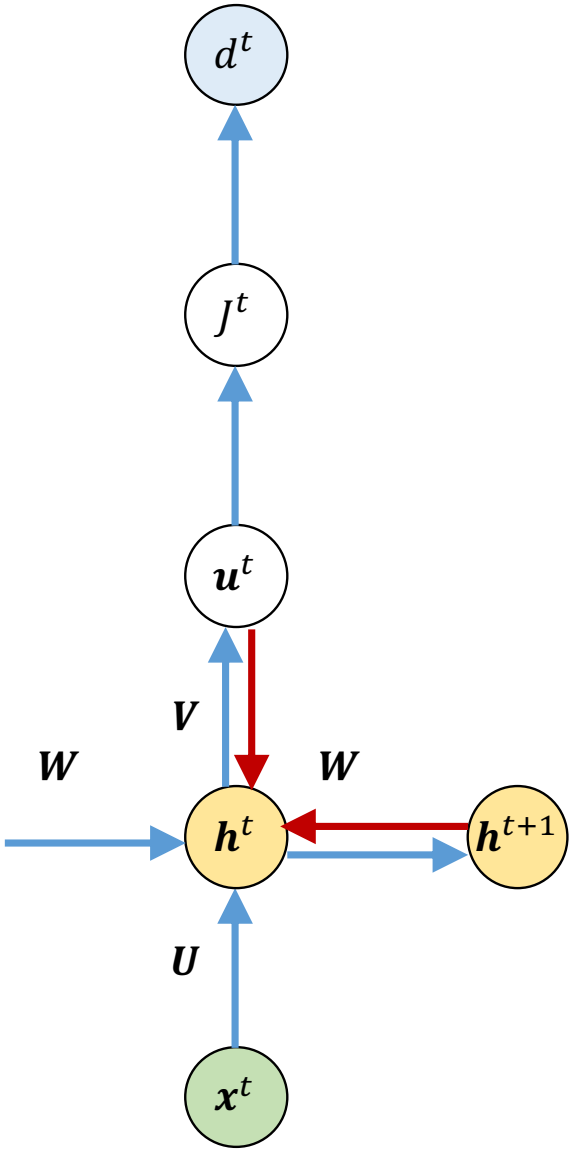
Note that

$$\frac{\partial J}{\partial J(t)} = 1$$

Error gradients at the output:

$$\nabla_{\boldsymbol{u}(t)} J = \nabla_{\boldsymbol{u}(t)} J(t) = \begin{cases} -\big(1(\boldsymbol{k} = d(t)) - \sigma(\boldsymbol{u}(t))\big), & for\ softmax \\ -\big(\boldsymbol{d}(t) - \boldsymbol{u}(t)\big), & for\ linear\ regression \end{cases}$$

Backpropagation through time (BPTT)



The gradients propagates backward, starting from the end of the sequence from two directions: top-down direction and reverse sequence direction.

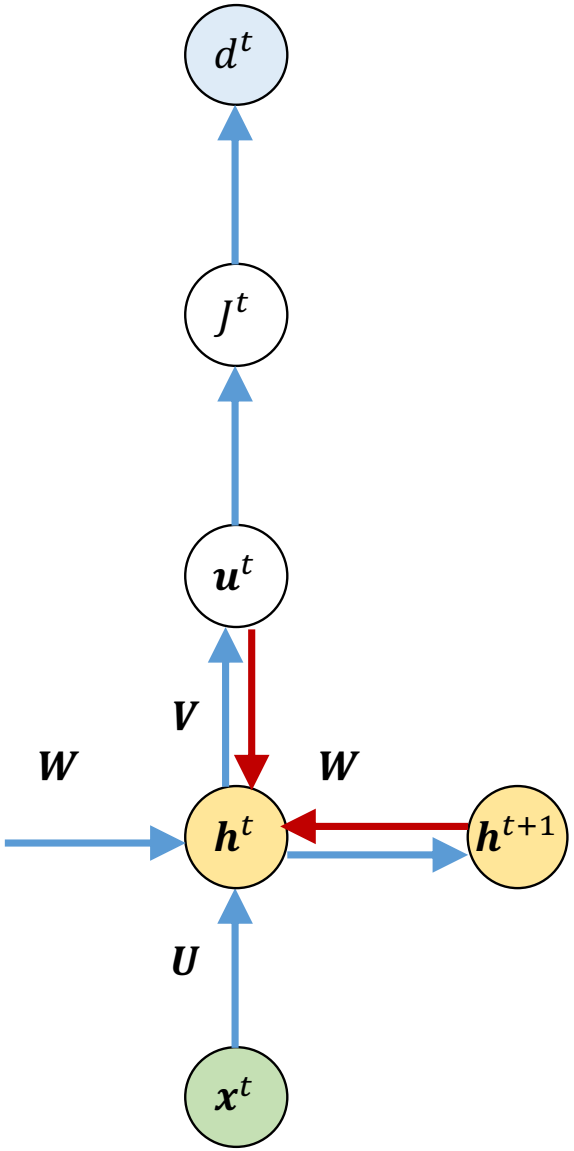
At the final time step T, $\boldsymbol{h}(t)$ has only one descendant.
For $t = T$:

$$\nabla_{\boldsymbol{h}(T)} J = \boldsymbol{V}^T \nabla_{\boldsymbol{u}(T)} J \qquad \text{(A)}$$

For $t < T$, $\boldsymbol{h}(t)$ has descendants of gradients from both $\boldsymbol{h}(t)$ and $\boldsymbol{u}(t)$. That is, for $t = 1, 2, \cdots T-1$, from chain rule for multi-dimensions,

$$\nabla_{\boldsymbol{h}(t)} J = \left(\frac{\partial \boldsymbol{h}(t+1)}{\partial \boldsymbol{h}(t)}\right)^T \nabla_{\boldsymbol{h}(t+1)} J + \left(\frac{\partial \boldsymbol{u}(t)}{\partial \boldsymbol{h}(t)}\right)^T \nabla_{\boldsymbol{u}(t)} J \qquad \text{(B)}$$

Backpropagation through time (BPTT)



For $t = 1, 2, \cdots T-1$

$$\nabla_{\boldsymbol{h}(t)} J = \left(\frac{\partial \boldsymbol{h}(t+1)}{\partial \boldsymbol{h}(t)}\right)^T \nabla_{\boldsymbol{h}(t+1)} J + \left(\frac{\partial \boldsymbol{u}(t)}{\partial \boldsymbol{h}(t)}\right)^T \nabla_{u(t)} J$$

where:

$$\frac{\partial \boldsymbol{h}(t+1)}{\partial \boldsymbol{h}(t)} = \frac{\partial \boldsymbol{h}(t+1)}{\partial \boldsymbol{z}(t+1)} \frac{\partial \boldsymbol{z}(t+1)}{\partial \boldsymbol{h}(t)}$$

and $\boldsymbol{z}(t)$ is the synaptic input to the hidden-layer at instant t and the hidden-layer has a *tanh* activation function.

$$\frac{\partial \boldsymbol{h}(t+1)}{\partial \boldsymbol{z}(t+1)} = \phi'\big(\boldsymbol{z}(t+1)\big) = \mathrm{diag}\big(1 - \boldsymbol{h}^2(t+1)\big) \qquad \text{(C)}$$

And

$$\frac{\partial \boldsymbol{z}(t+1)}{\partial \boldsymbol{h}(t)} = \boldsymbol{W}^T \text{ and } \frac{\partial \boldsymbol{u}(t)}{\partial \boldsymbol{h}(t)} = \boldsymbol{V}^T \qquad \text{(D)}$$

Backpropagation through time (BPTT)

Substituting (C) and (D) in (B);

$$\nabla_{\boldsymbol{h}(t)} J = \boldsymbol{W}\mathrm{diag}\left(1 - \boldsymbol{h}^2(t+1)\right) \nabla_{\boldsymbol{h}(t+1)} J + \boldsymbol{V}\nabla_{u(t)} J \qquad \text{(E)}$$

Starting from (A):

$$\nabla_{\boldsymbol{h}(T)} J = \boldsymbol{V}\nabla_{\boldsymbol{u}(T)} J$$

The above equation (E) can be recursively used to compute $\nabla_{\boldsymbol{h}(t)} J$ backwards.

Note that each backward propagation through time, the $\nabla_{\boldsymbol{h}(t)} J$ is multiplied by $\boldsymbol{W}^T$.

Backpropagation through time (BPTT)

Using gradients $\nabla_{\boldsymbol{u}(t)}J$ and $\nabla_{\boldsymbol{h}(t)}J$, the error gradients with respect to the parameters $\{\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{b}, \boldsymbol{c}\}$ can be obtained:

$$\nabla_{\boldsymbol{c}}J = \sum_t \frac{\partial J(t)}{\partial \boldsymbol{c}} = \sum_t \left(\frac{\partial \boldsymbol{u}(t)}{\partial \boldsymbol{c}}\right)^T \nabla_{\boldsymbol{u}(t)}J = \sum_t \nabla_{\boldsymbol{u}(t)}J$$

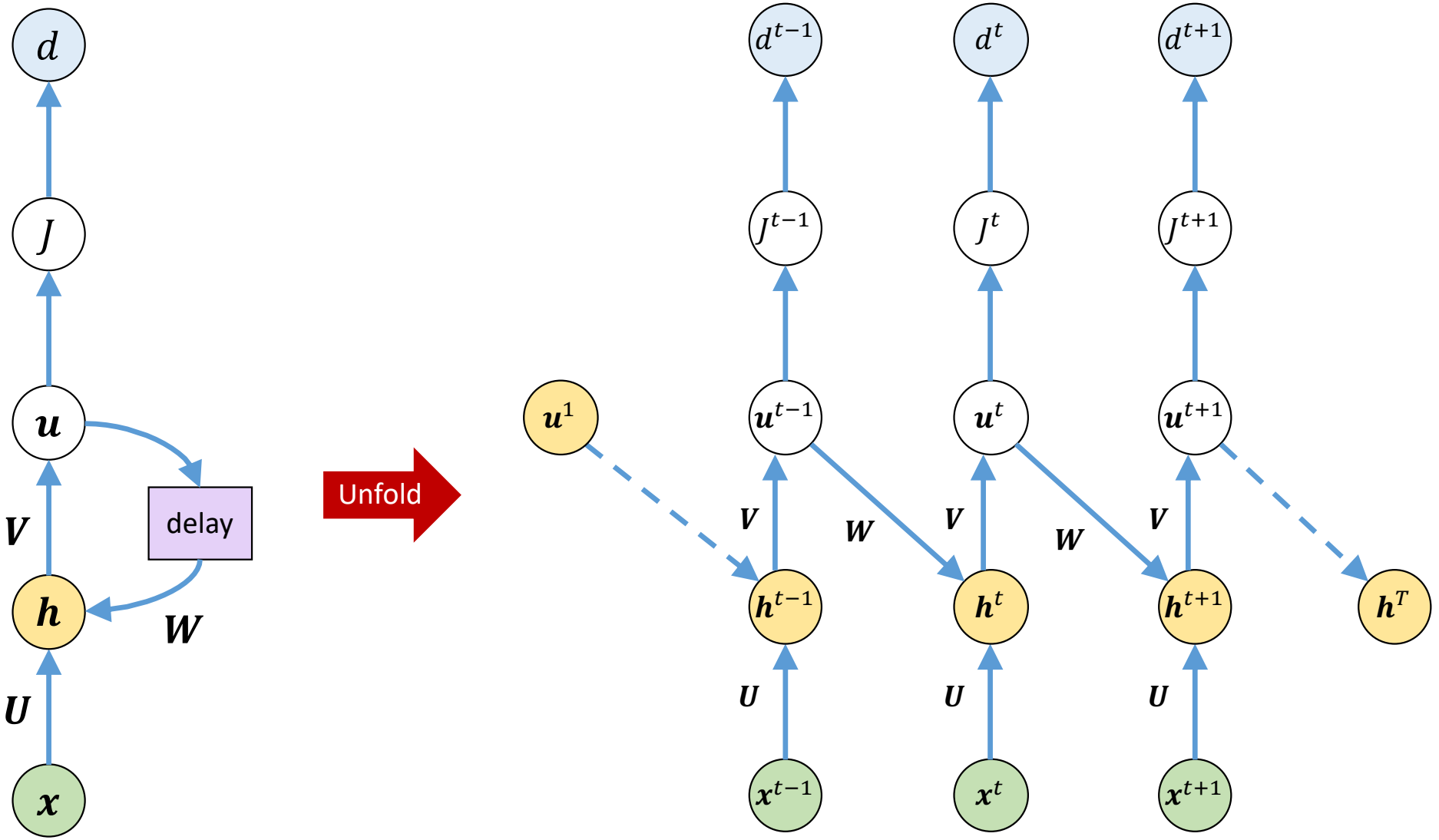
$$\nabla_{\boldsymbol{b}}J = \sum_t \left(\frac{\partial \boldsymbol{h}(t)}{\partial \boldsymbol{b}}\right)^T \nabla_{\boldsymbol{h}(t)}J = \sum_t \left(\frac{\partial \boldsymbol{h}(t)}{\partial \boldsymbol{z}(t)}\frac{\partial \boldsymbol{z}(t)}{\partial \boldsymbol{b}}\right)^T \nabla_{\boldsymbol{h}(t)}J = \sum_t diag(1 - \boldsymbol{h}(t)^2)\nabla_{\boldsymbol{h}(t)}J$$

$$\nabla_{\boldsymbol{V}}J = \sum_t \boldsymbol{h}(t)\left(\nabla_{\boldsymbol{u}(t)}J\right)^T$$

$$\nabla_{\boldsymbol{W}}J = \sum_t \boldsymbol{h}(t-1)\left(\nabla_{\boldsymbol{z}(t)}J\right)^T = \sum_t \boldsymbol{h}(t-1)\left(\nabla_{\boldsymbol{h}(t)}J\right)^T diag(1 - \boldsymbol{h}(t)^2)$$

$$\nabla_{\boldsymbol{U}}J = \sum_t \boldsymbol{x}(t)\left(\nabla_{\boldsymbol{z}(t)}J\right)^T = \sum_t \boldsymbol{x}(t)\left(\nabla_{\boldsymbol{h}(t)}J\right)^T diag(1 - \boldsymbol{h}(t)^2)$$

BPTT for RNN with top-down recurrence



Backpropagation through time (BPTT)

The gradient computation involves performing a forward propagation pass moving left to right through the unfolded graph, followed by a backpropagation pass moving right to left through the graph. The gradients are propagated from the final time point to the initial time points .

The runtime is $O(T)$ where T is the length of input sequence and cannot be reduced by parallelization because the forward propagation is inherently sequential. The back-propagation algorithm applied to the unrolled graph with $O(T)$ cost is called *back-propagation through time (BPTT).*

The network with recurrence between hidden units is thus very powerful but also expensive to train.

Example 3

Generate 8-dimensional 16 input sequences of 64 time-steps with each input is a random number between [0.0, 1.0].

Generate the corresponding 1-dimensional labels by randomly generating a number from [0, 1, 2].

Create an RNN with one hidden layer of 5 neurons.

Plot the learning curves and predicted labels.

Example 3 (con't)

```
# Initialize hyper-parameters
n_in = 8
n_hidden = 5
n_out = 3
n_steps = 64
n_seqs = 16
n_iters = 25000
lr = 0.0001
```

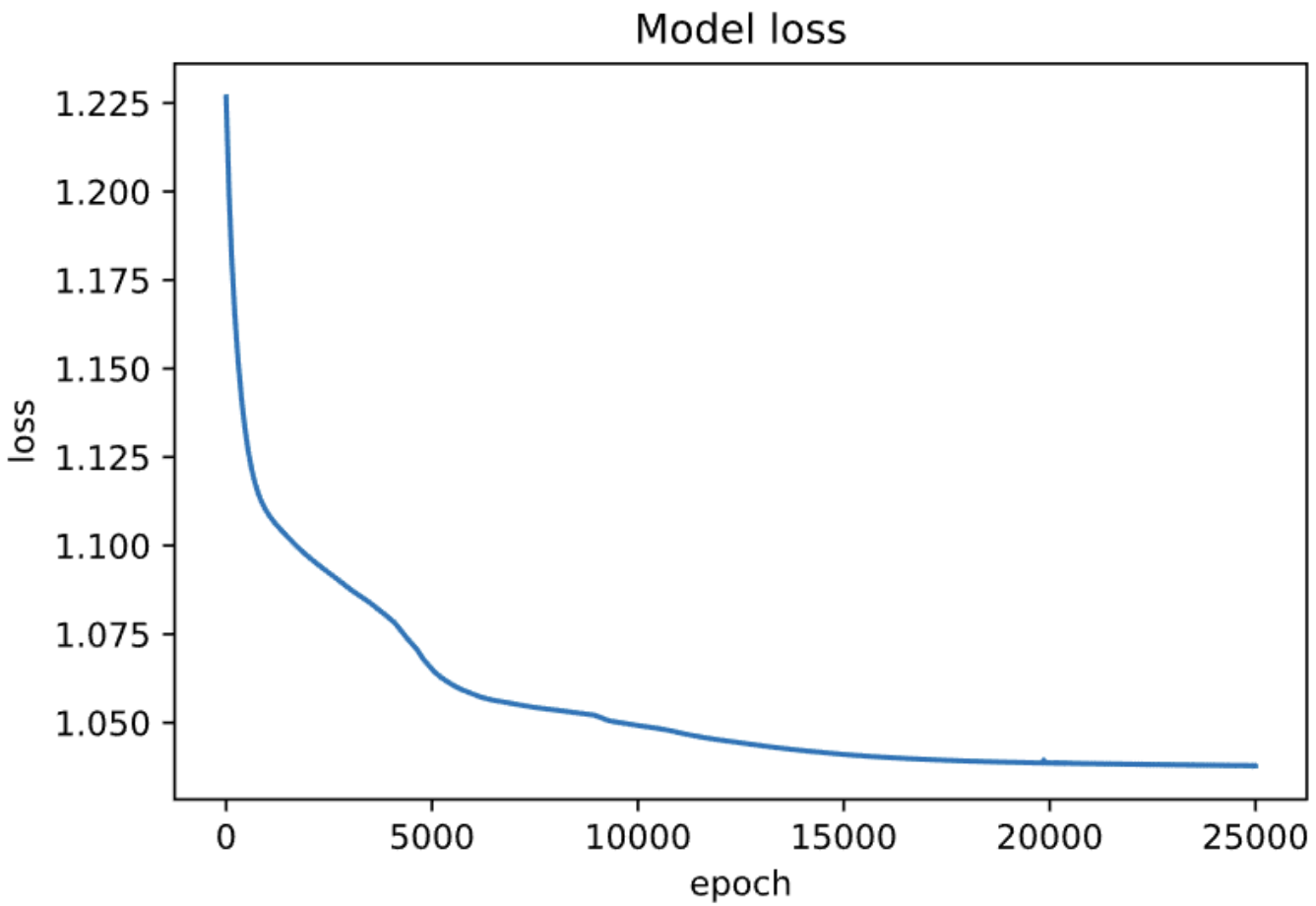
```
# Generate training data
x_train = tf.constant(np.random.rand(n_seqs, n_steps, n_in), dtype=tf.float32)
x_train = tf.split(x_train, n_steps, axis=1)
y_train = np.random.randint(size=(n_seqs, n_steps), low=0, high=n_out)
y_train = tf.reshape(tf.constant(y_train, dtype=tf.float32), [n_seqs, n_steps, 1])
```

Example 3 (con't)

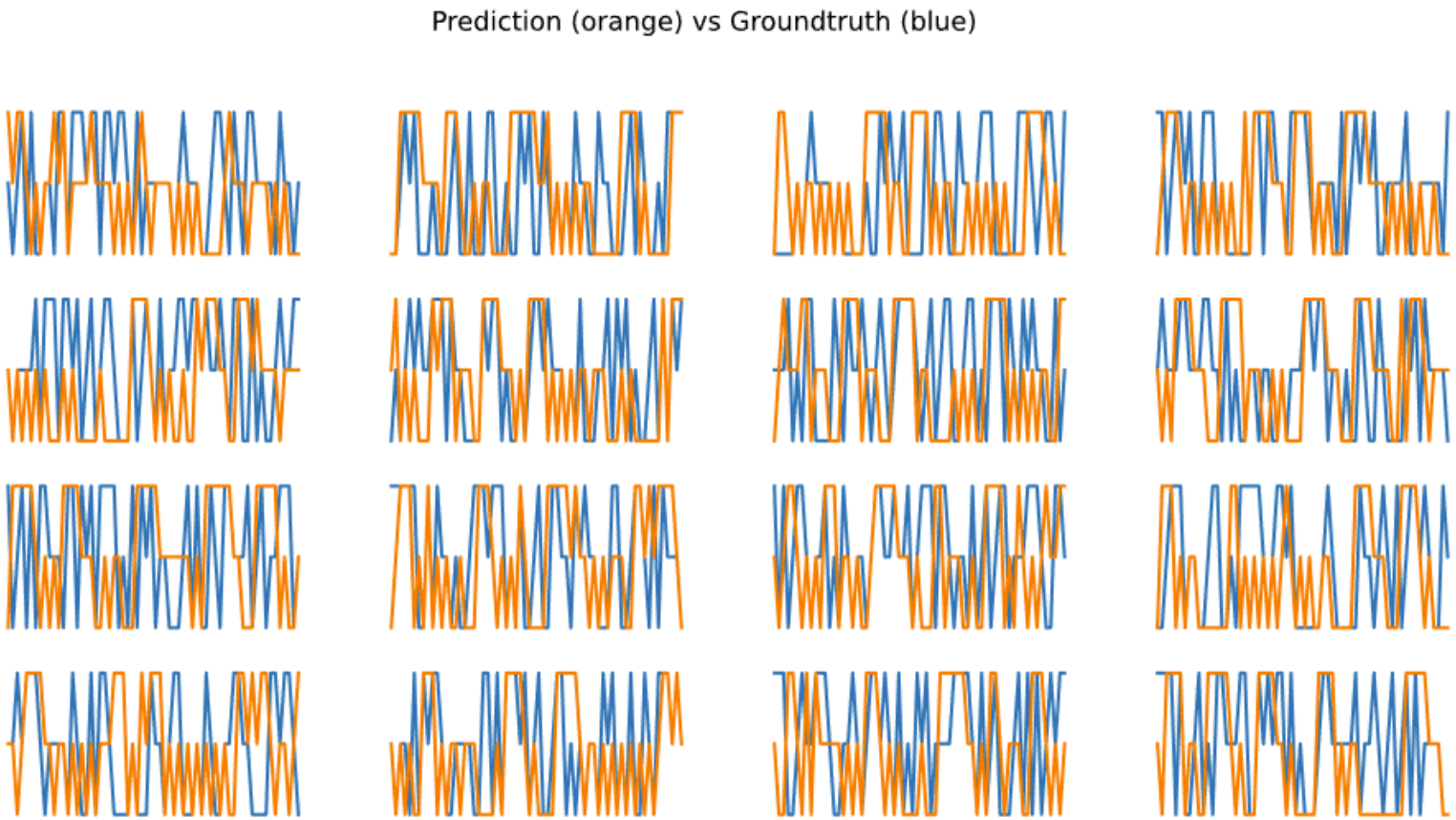
```
class RNN(Model):
    def __init__(self, n_in, n_hidden, n_out):

        super(RNN, self).__init__()

        self.n_hidden = n_hidden

        # Specify the weights and biases
        self.u = tf.Variable(tf.random.truncated_normal([n_in, n_hidden], stddev=1 / np.sqrt(n_in)))
        self.w = tf.Variable(tf.random.truncated_normal([n_hidden, n_hidden], stddev=1 / np.sqrt(n_hidden)))
        self.v = tf.Variable(tf.random.truncated_normal([n_hidden, n_out], stddev=1 / np.sqrt(n_hidden)))
        self.b = tf.Variable(tf.zeros([n_hidden]))
        self.c = tf.Variable(tf.zeros([n_out]))

    def call(self, x):

        # The initial hidden state, which is a zero vector
        h = tf.constant(np.array([0] * self.n_hidden), dtype=tf.float32)
        h = tf.tile(tf.reshape(h, [1, h.shape[0]]), [x[0].shape[0], 1])

        ys = []
        for i in range(0, len(x)):
            h = tf.tanh(tf.matmul(tf.squeeze(x[i]), self.u) + tf.matmul(h, self.w) + self.b)
            u = tf.matmul(h, self.v) + self.c
            ys.append(u)

        return tf.stack(ys, axis=1)
```

```
# Initialize the model, optimizer and loss for training
rnn = RNN(n_in, n_hidden, n_out)
optimizer = keras.optimizers.Adam(lr)
loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True)

# Training
print('Training starts...')
rnn.compile(optimizer=optimizer, loss=loss)
history = rnn.fit(x_train, y_train,
                  batch_size=n_seqs,
                  epochs=n_iters,
                  verbose=0)
print('Training ended...')
```

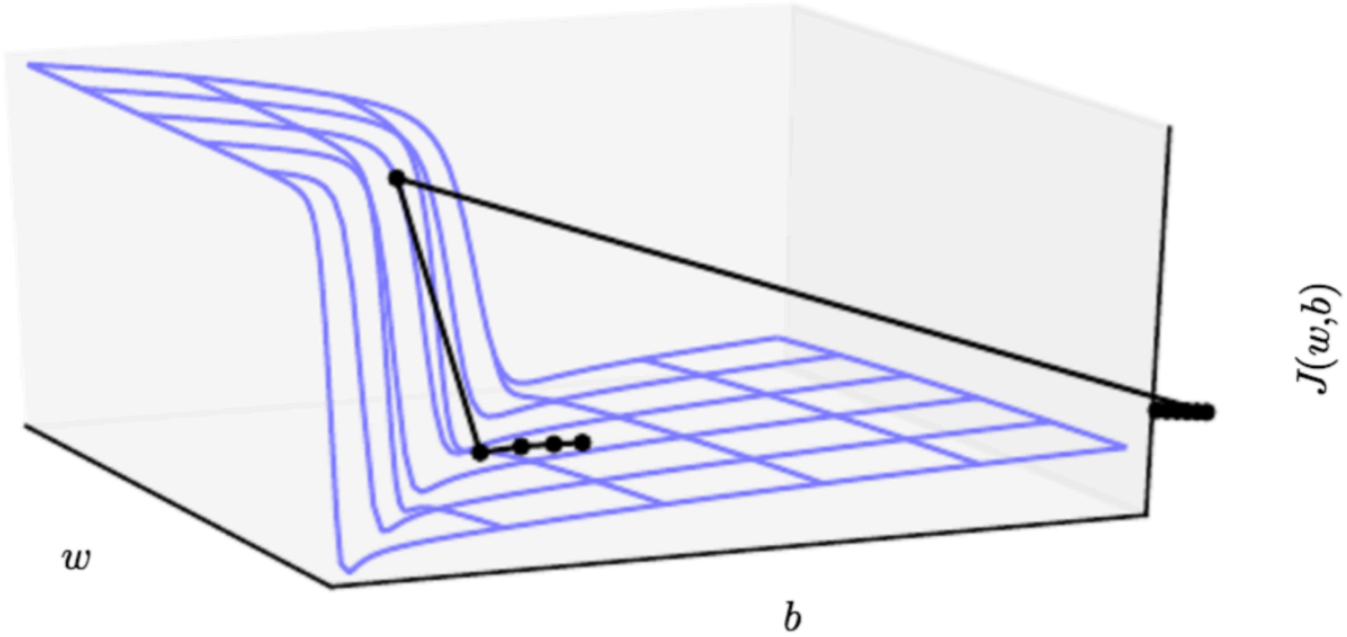
Example 3 (con’t)



Example 3 (con’t)



Gradient Clipping



Due to long term dependencies, RNN tend to have gradients having very large or very small magnitudes. The large gradients resemble cliffs in the error landscape and when the gradient descent encounters the gradient updates can move the parameters away from true minimum.

A gradient clipping is employed to avoid the gradients to become too large.

Gradient Clipping

Commonly, two methods are used:

1. Clip the gradient g when it exceeds a threshold:

$$\text{if} \|g\| > \nu\text{:}$$
$$\|g\| \leftarrow \nu$$

2. Normalize the gradient when it exceeds a threshold:

$$\text{if} \|g\| > \nu\text{:}$$
$$\|g\| \leftarrow \frac{g}{\|g\|} \nu$$

Image Credits

- https://cs.stanford.edu/people/karpathy/sfmltalk.pdf
- https://developer.ibm.com/articles/cc-cognitive-recurrent-neural-networks/