# Deep Learning
Recurrent Networks

2/28/2018

# Recap: Recurrent networks can be incredibly effective

```c
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECON
  return segtable;
}
```
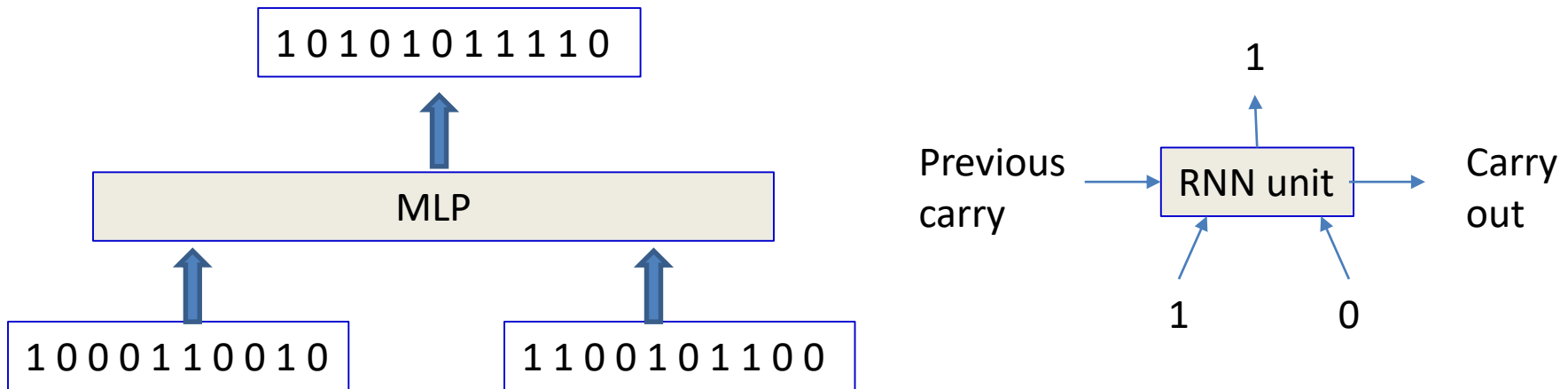
Story so far
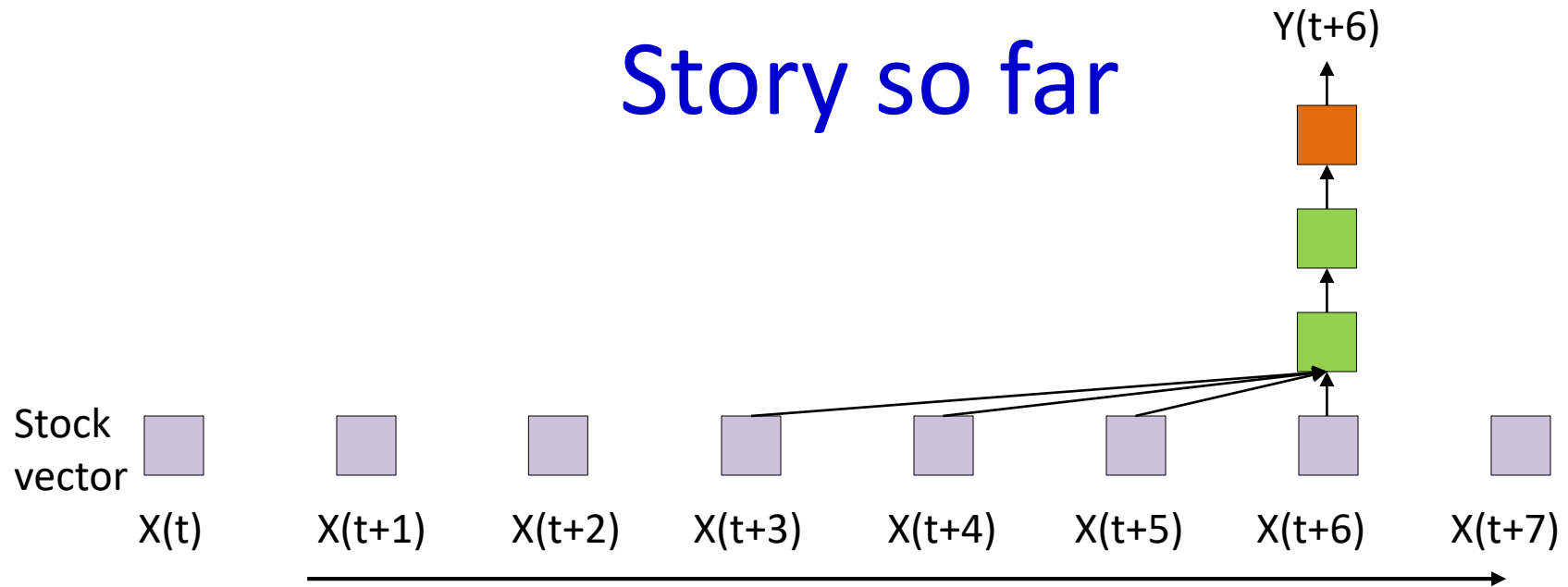
- ***Iterated structures*** are good for analyzing time series data with short-time dependence on the past
 - These are "***Time delay***" neural nets, AKA ***convnets***

Story so far
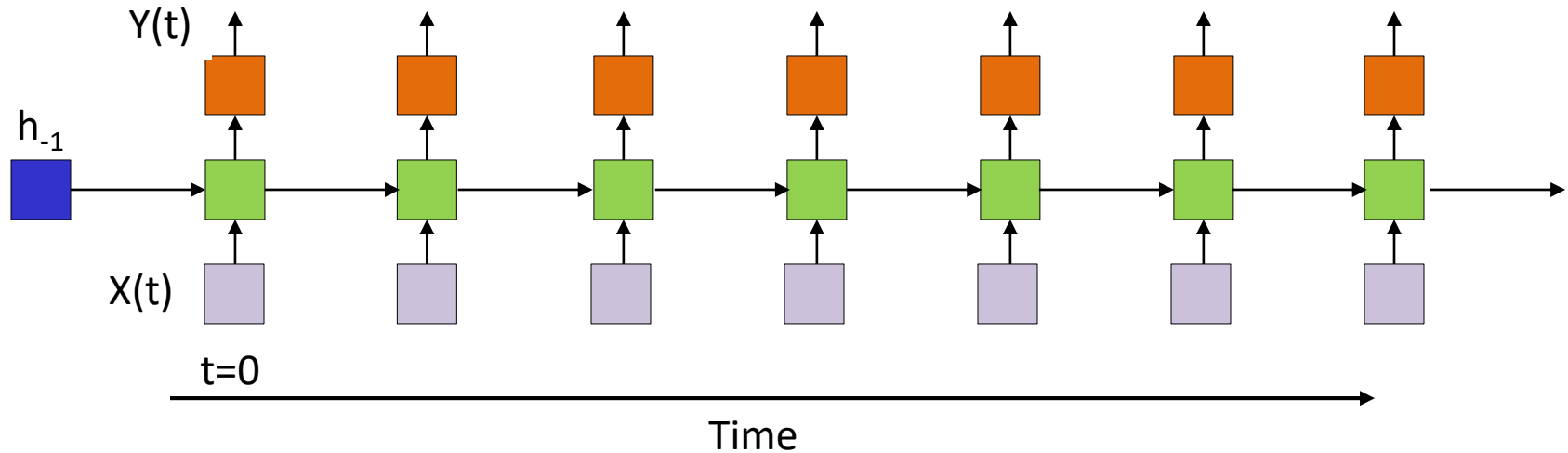
- Iterated structures are good for analyzing time series data with short-time dependence on the past
  - These are "Time delay" neural nets, AKA convnets
- *Recurrent structures* are good for analyzing time series data with *long-term* dependence on the past
  - These are *recurrent* neural networks

Recurrent structures can do what static structures cannot

```
1 0 1 0 1 0 1 1 1 1 0
```

MLP

```
1 0 0 0 1 1 0 0 1 0
```     ```
1 1 0 0 1 0 1 1 0 0
```

1

Previous carry → RNN unit → Carry out

1          0

- The addition problem:  Add two N-bit numbers to produce a N+1-bit number
  - Input is binary
  - Will require large number of training instances
    - Output must be specified for every pair of inputs
    - Weights that generalize will make errors
  - Network trained for N-bit numbers will not work for N+1 bit numbers

- An RNN learns to do this very quickly
  - With very little training data!

# Story so far
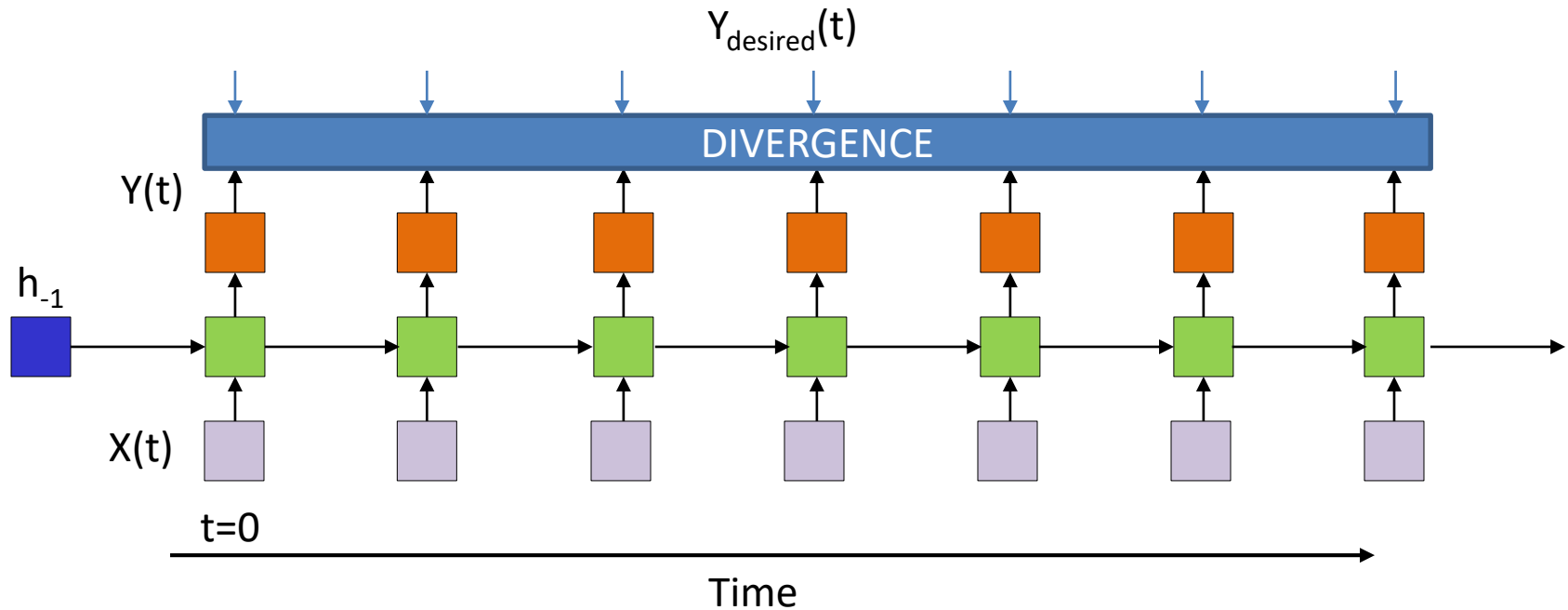


- Recurrent structures can be trained by minimizing the divergence between the *sequence* of outputs and the *sequence* of desired outputs
  - Through gradient descent and backpropagation
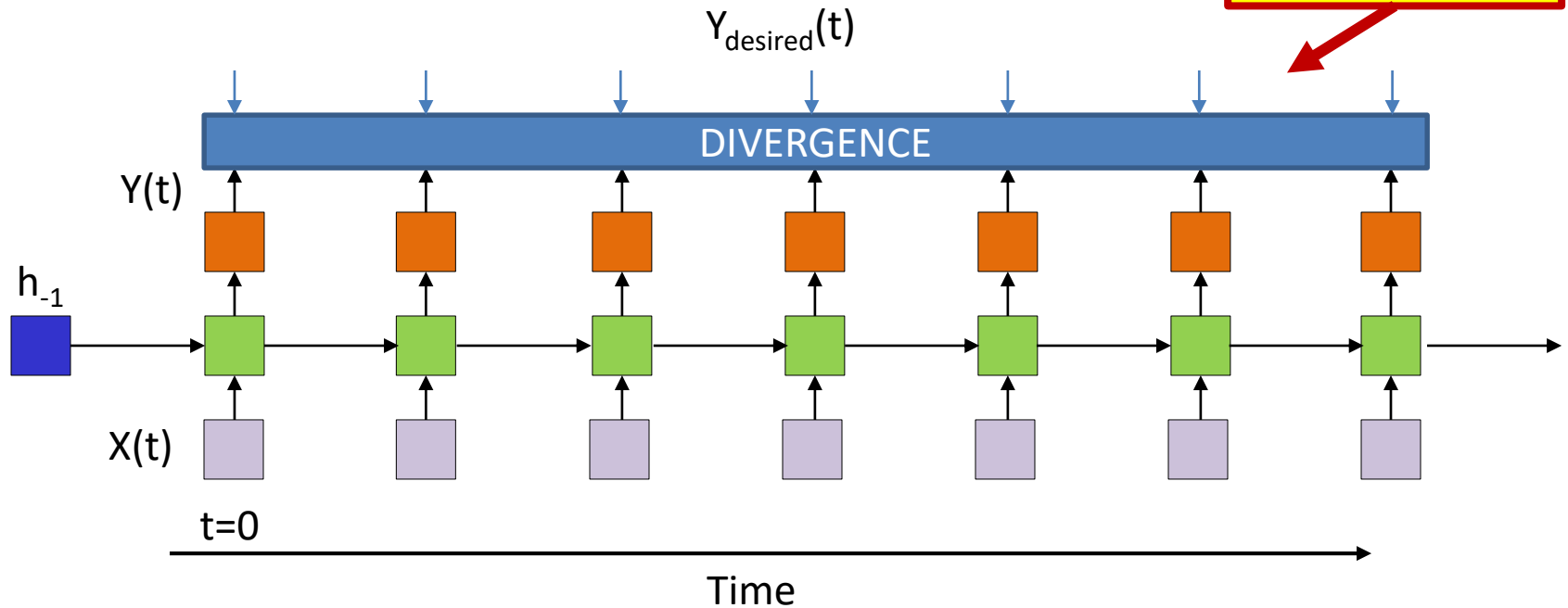
# Story so far

- Recurrent structures can be trained by minimizing the divergence between the *sequence* of outputs and the *sequence* of desired outputs
  - Through gradient descent and backpropagation

# Story so far: stability

- Recurrent networks can be unstable
  - And not very good at remembering at other times



sigmoid                                    tanh                                    relu
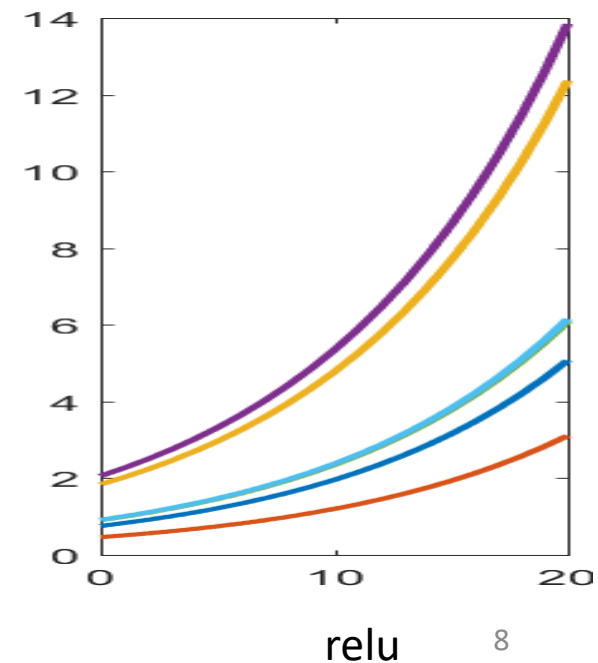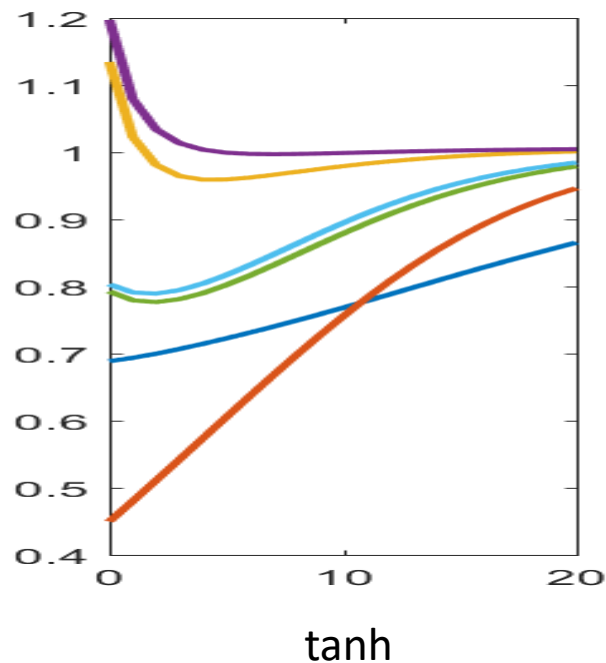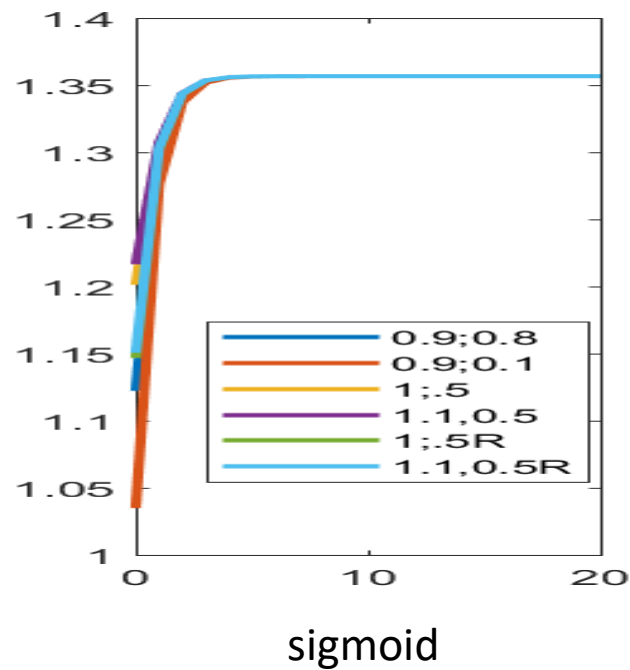
# Vanishing gradient examples..

ELU  activation,  Batch gradients

Input layer
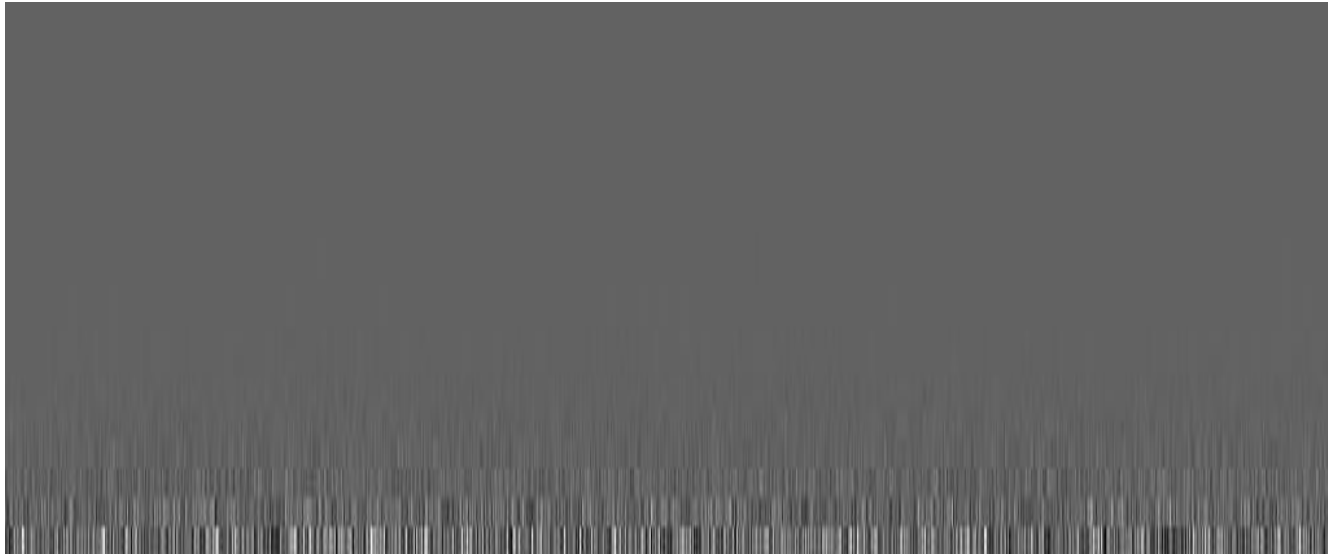
Output layer

- Learning is difficult: gradients tend to vanish..

# The long-term dependency problem

PATTERN1  [………………………..] PATTERN 2

*Jane* had a quick lunch in the bistro. Then *she..*

- Long-term dependencies are hard to learn in a network where memory behavior is an untriggered function of the *network*
  - Need it to be a triggered response to *input*

# Long Short-Term Memory
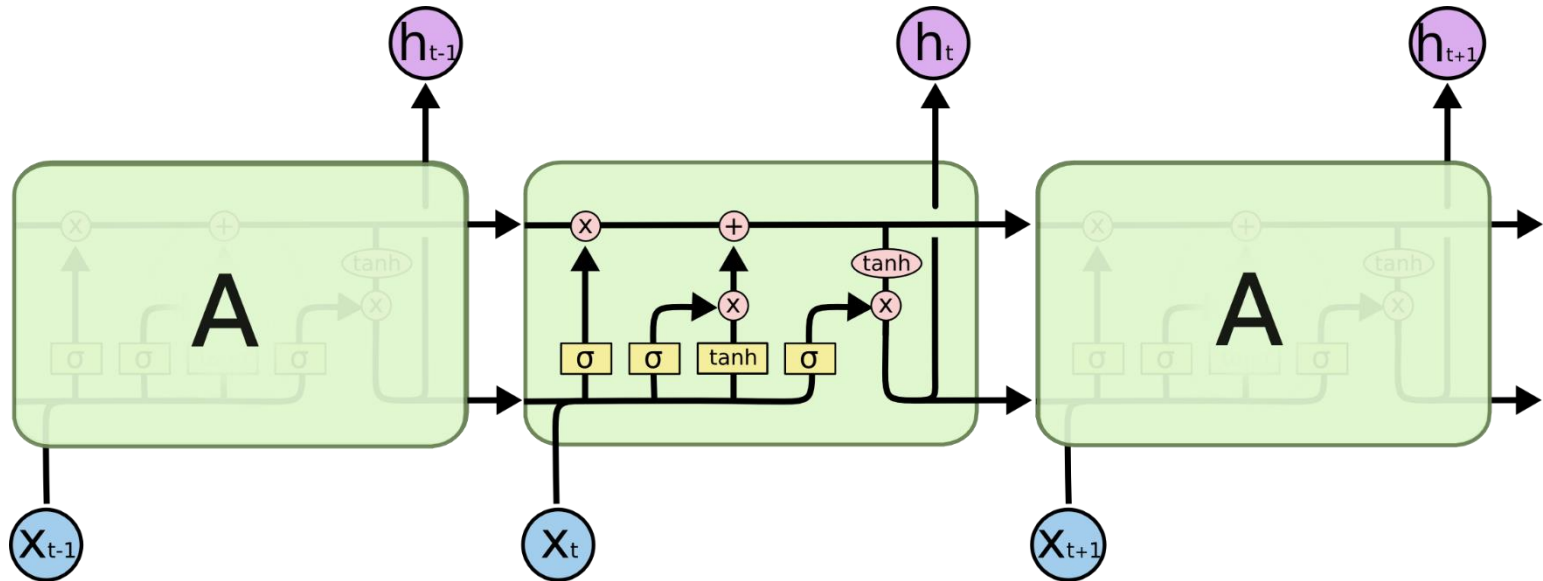


- The LSTM addresses the problem of *input-dependent* memory behavior

# LSTM-based architecture
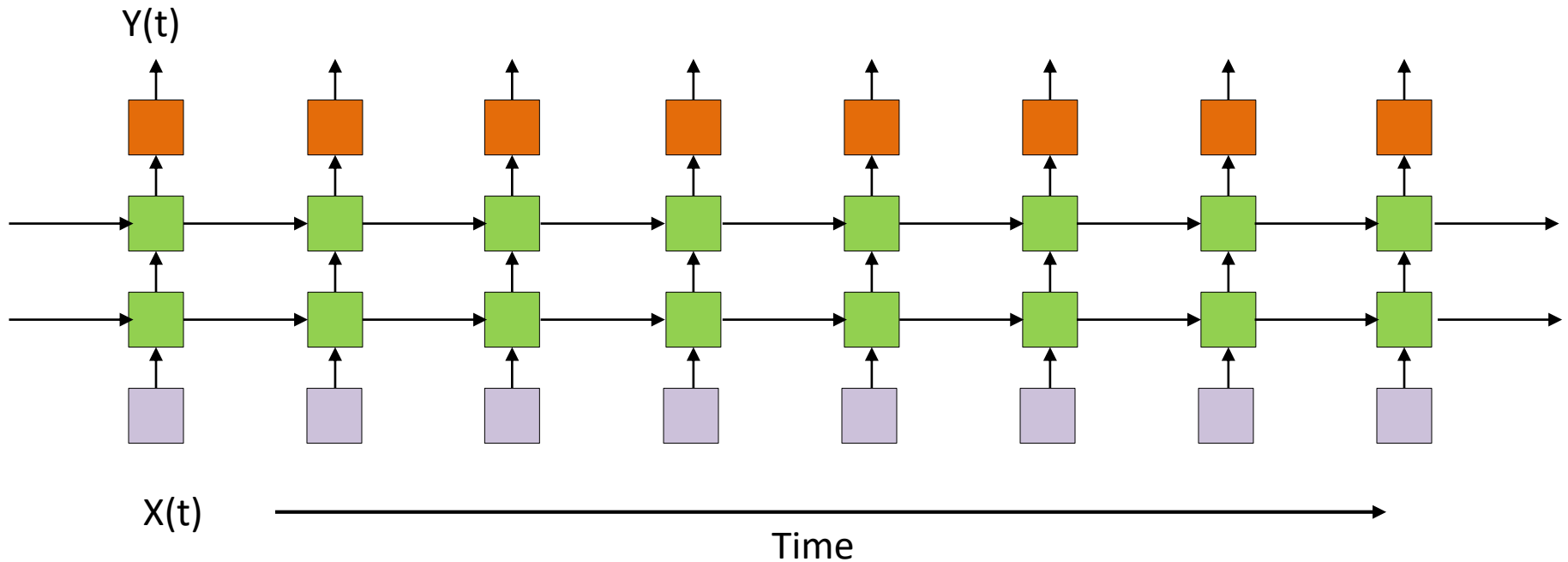


- LSTM based architectures are identical to RNN-based architectures

# Bidirectional LSTM
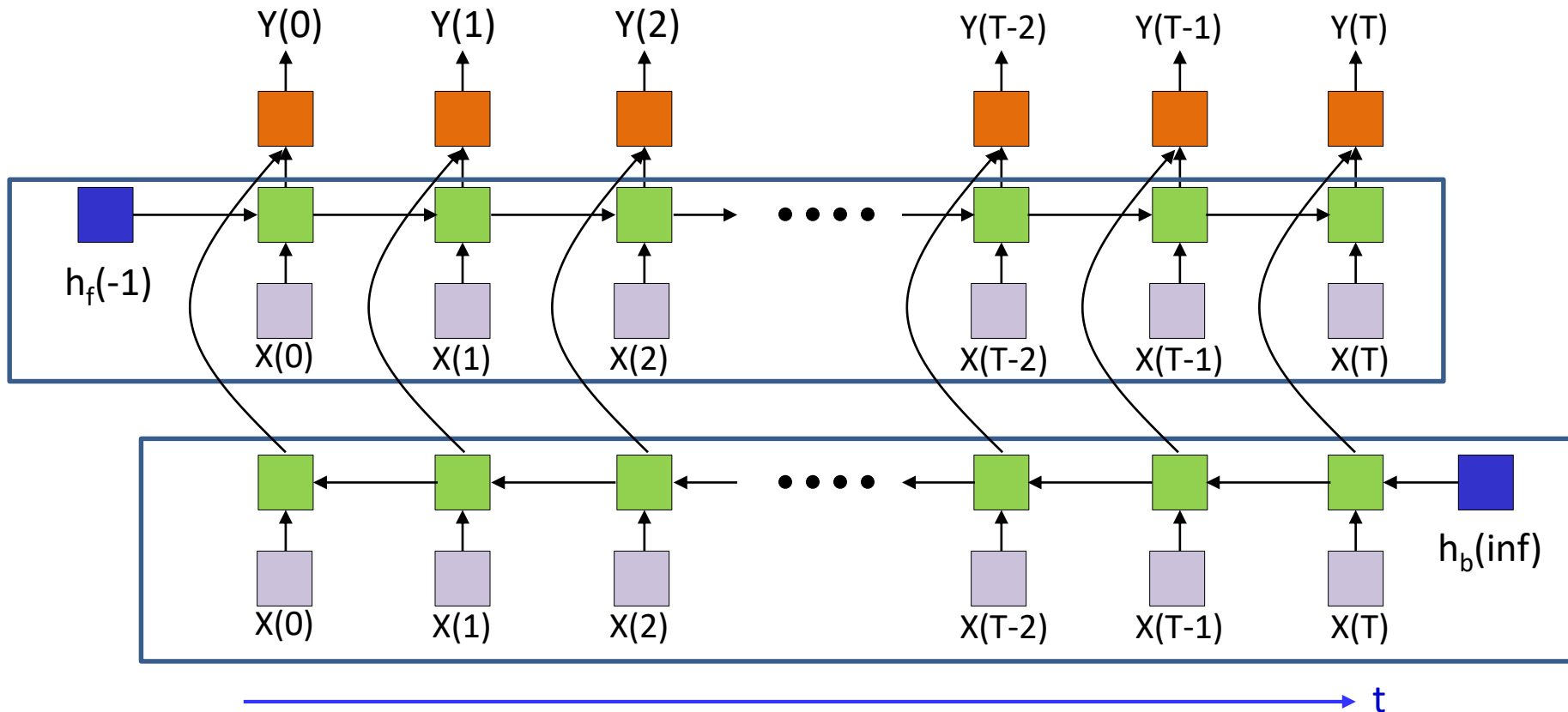


- Bidirectional version..

# Key Issue
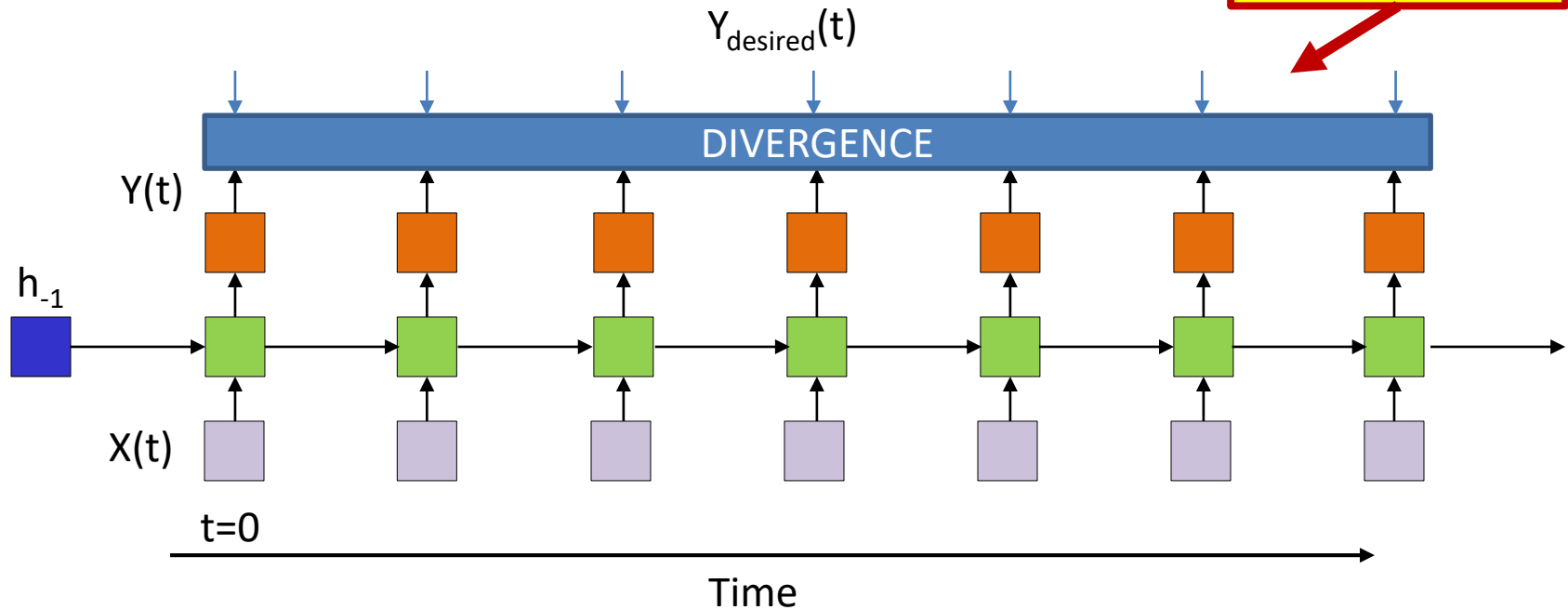


- How do we define the divergence

# How do we *train* the network
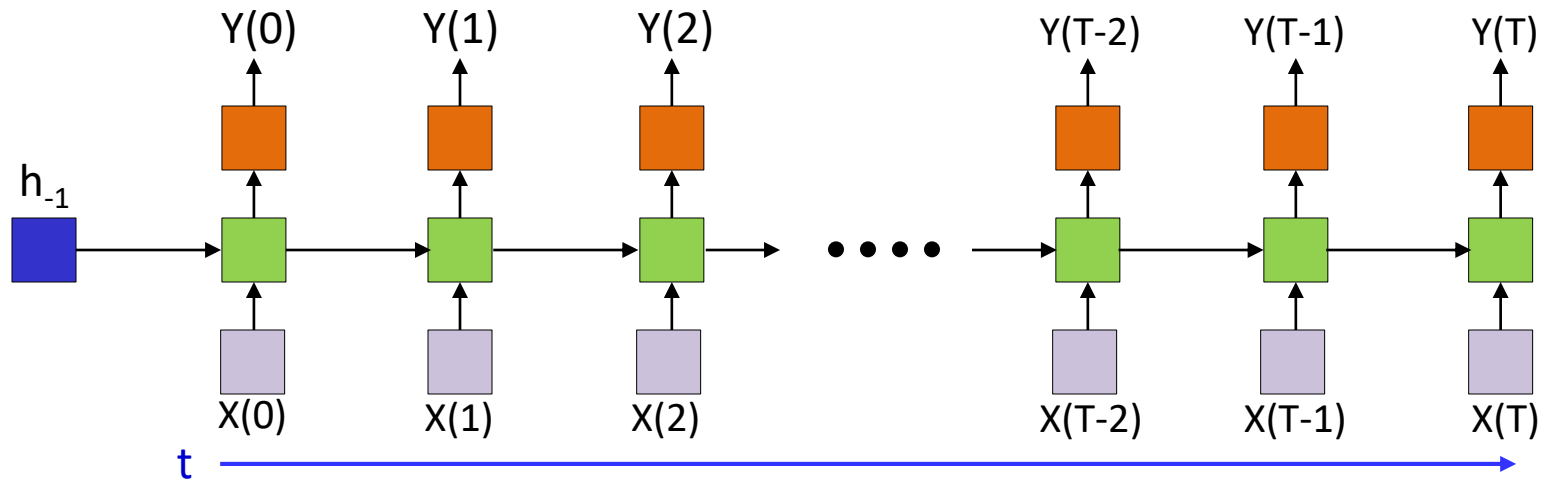


- Back propagation through time (BPTT)

- Given a collection of *sequence* inputs
    - $(\mathbf{X}_i, \mathbf{D}_i)$, where
    - $\mathbf{X}_i = X_{i,0}, \dots, X_{i,T}$
    - $\mathbf{D}_i = D_{i,0}, \dots, D_{i,T}$

# Training: Forward pass
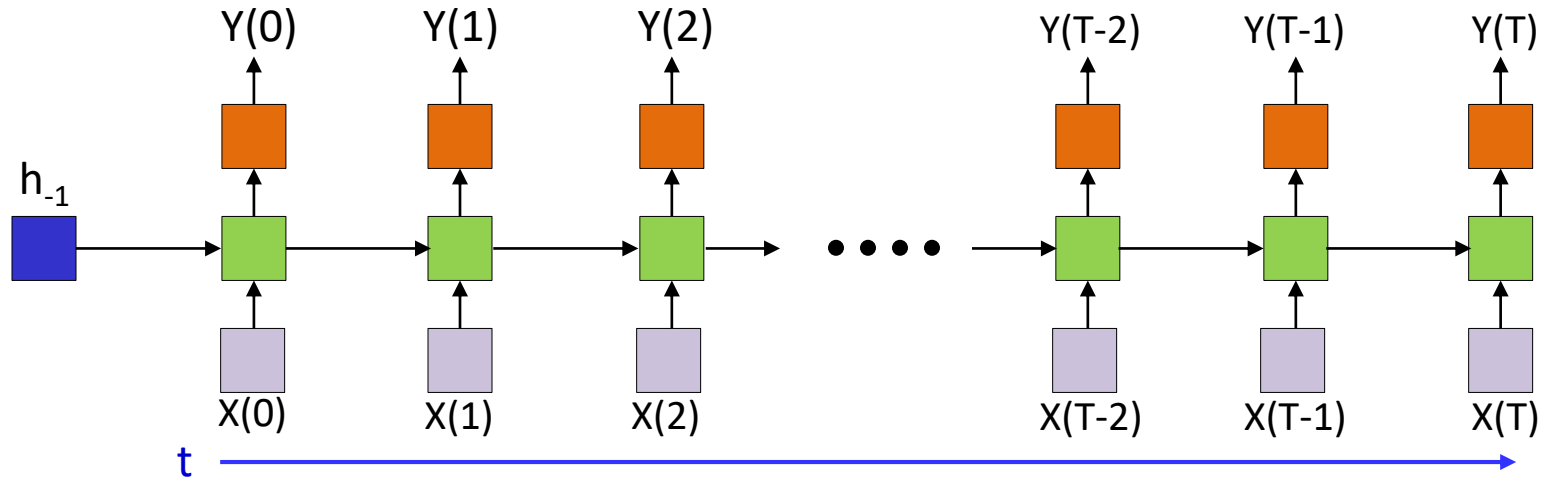


- For each training input:
- Forward pass:  pass the entire data sequence through the network, generate outputs

# Training: Computing gradients
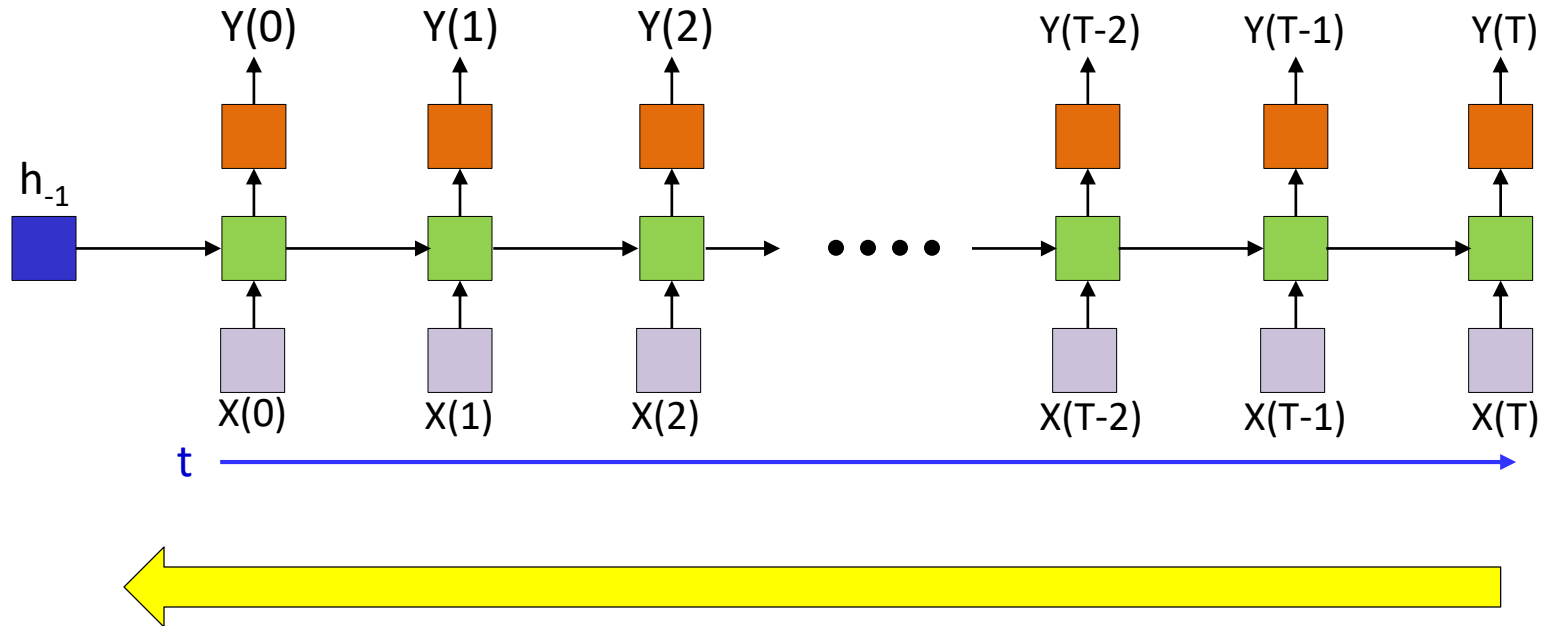


- For each training input:
- Backward pass: Compute gradients via backpropagation
  - *Back Propagation Through Time*

# Back Propagation Through Time
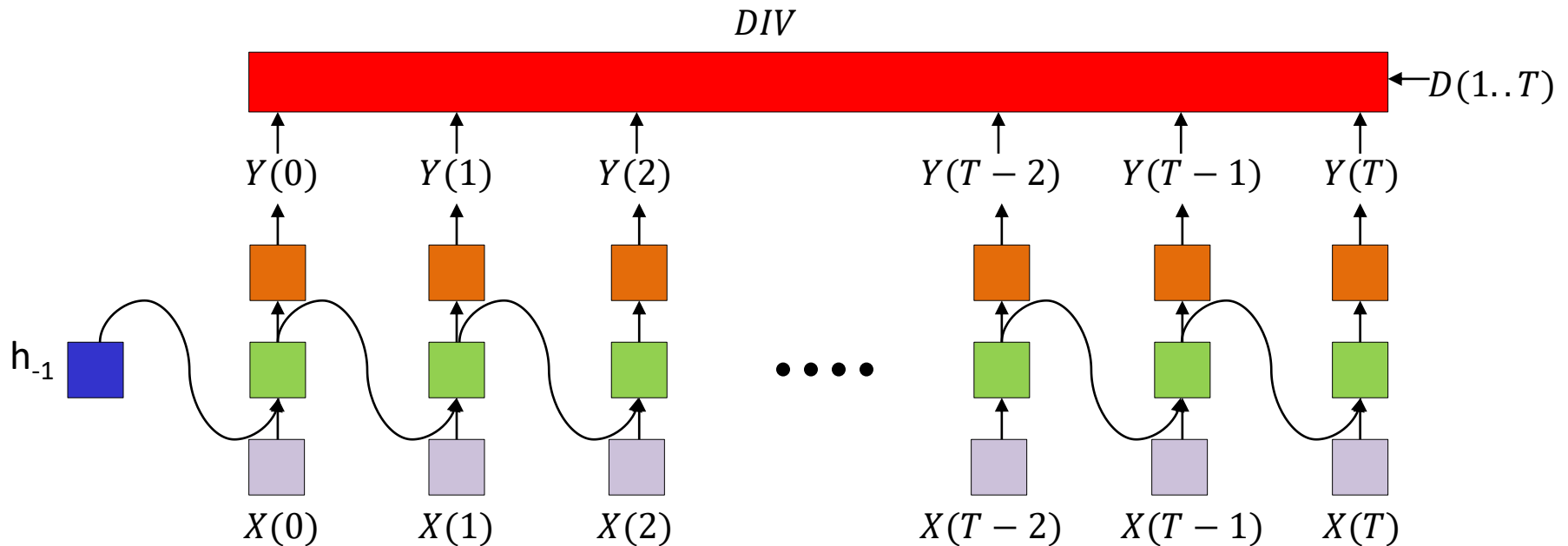


- The divergence computed is between the *sequence of outputs* by the network and the *desired sequence of outputs*

- This is *not* just the sum of the divergences at individual times
  - Unless we explicitly define it that way

# Back Propagation Through Time
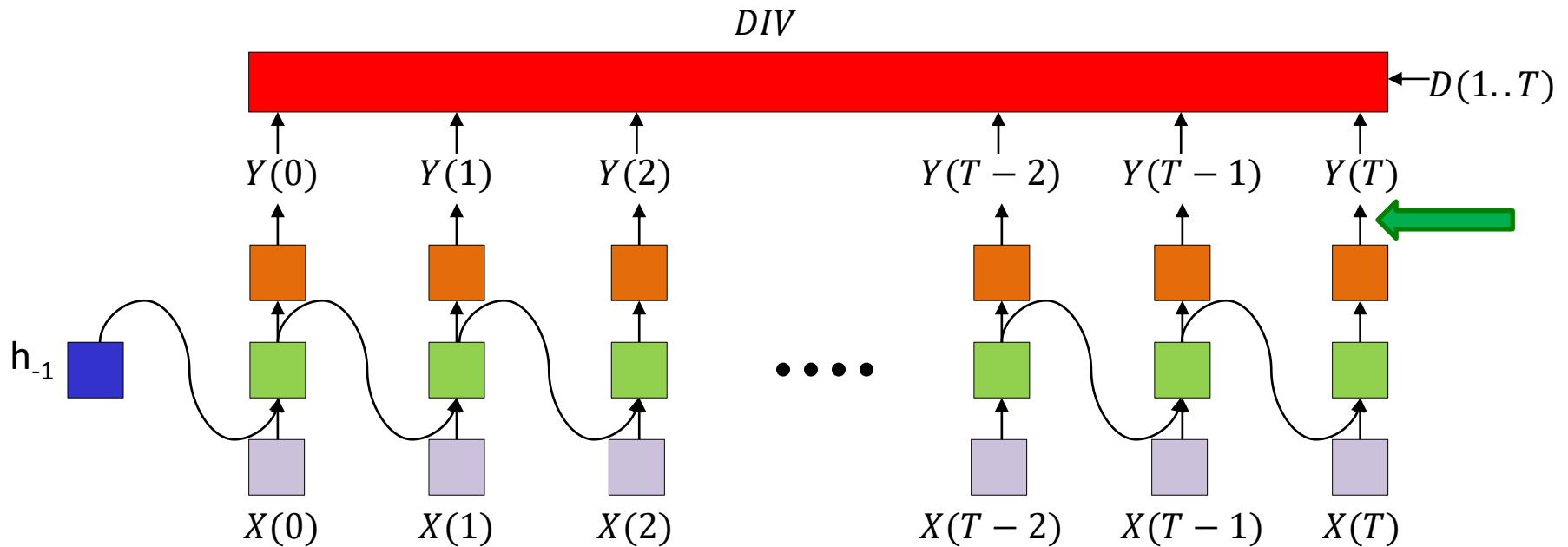


First step of backprop:   Compute $\nabla_{Y(t)} DIV$ for all $t$

The rest of backprop continues from there

# Back Propagation Through Time

$DIV$
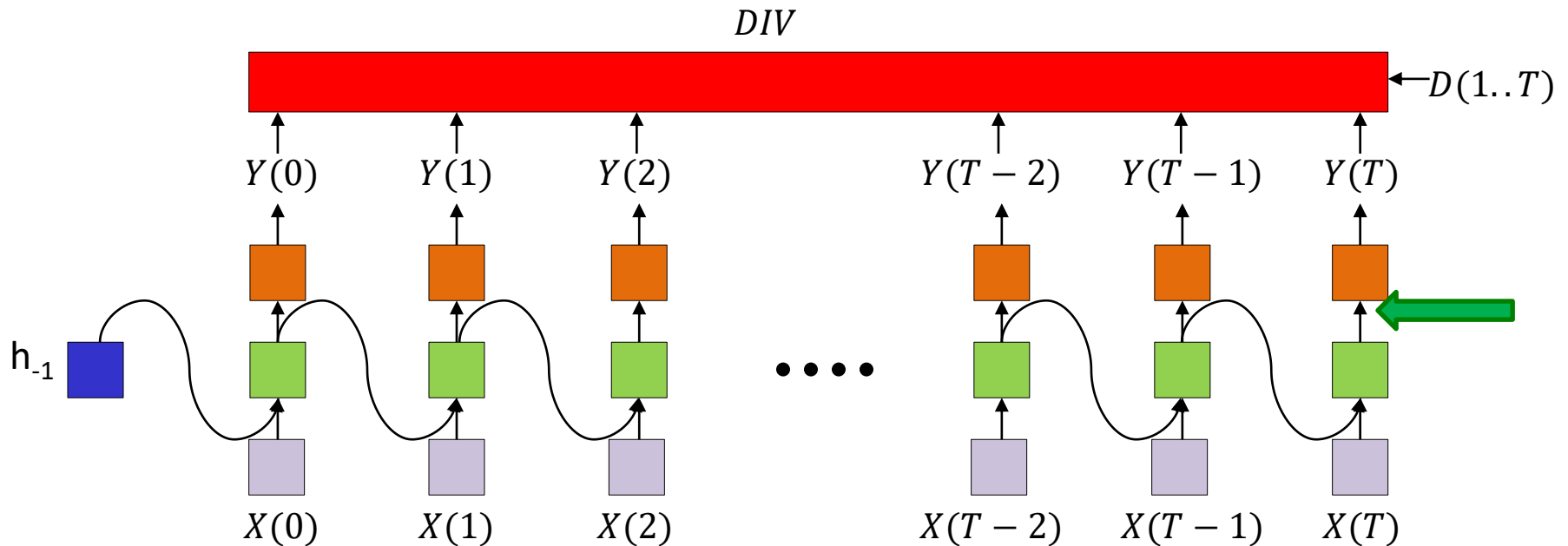


First step of backprop:   Compute $\nabla_{Y(t)} DIV$ for all $t$

$$\nabla_{Z^{(1)}(t)} DIV = \nabla_{Y(t)} DIV \ \nabla_{Z(t)} Y(t)$$

And so on!

# Back Propagation Through Time
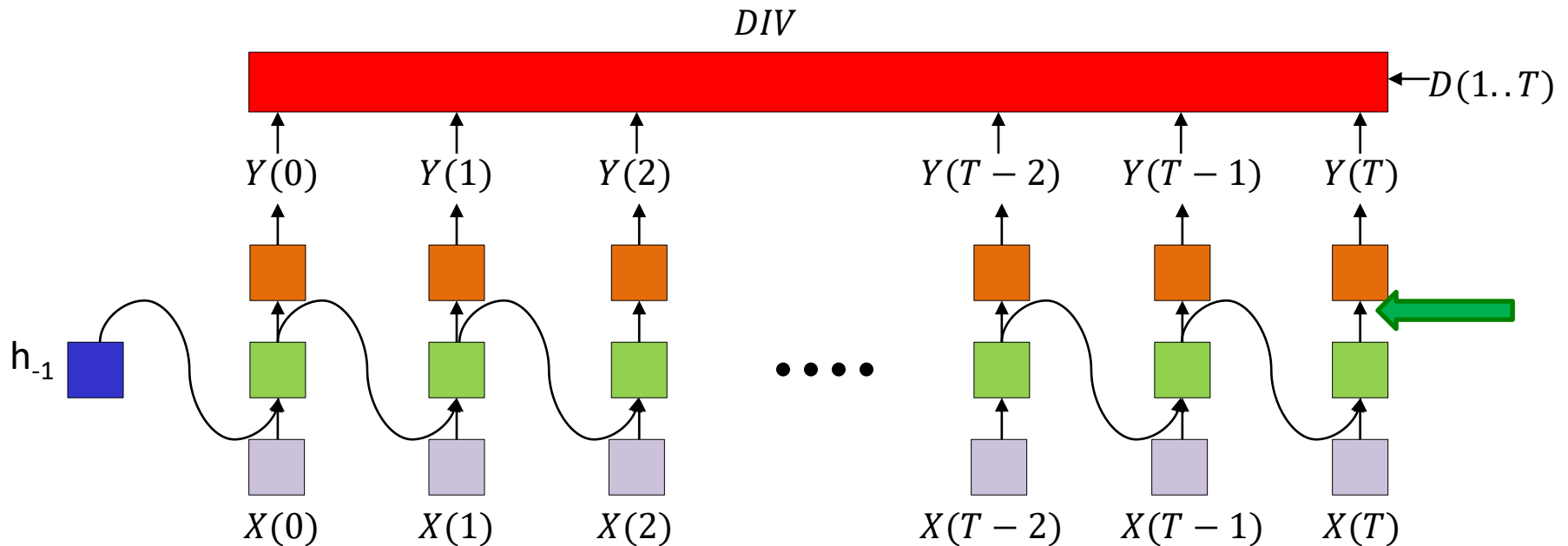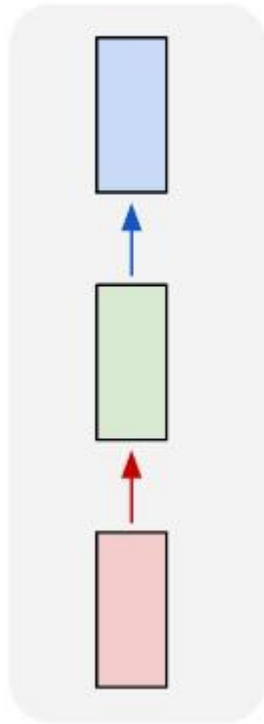


First step of backprop:   Compute $\nabla_{Y(t)} DIV$ for all $t$
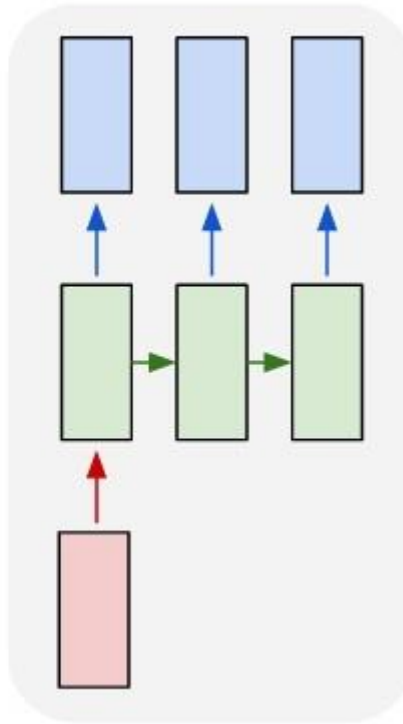
- The key component is the computation of this derivative!!
- This depends on the definition of "DIV"
- Which depends on the network structure itself

# Variants on recurrent nets
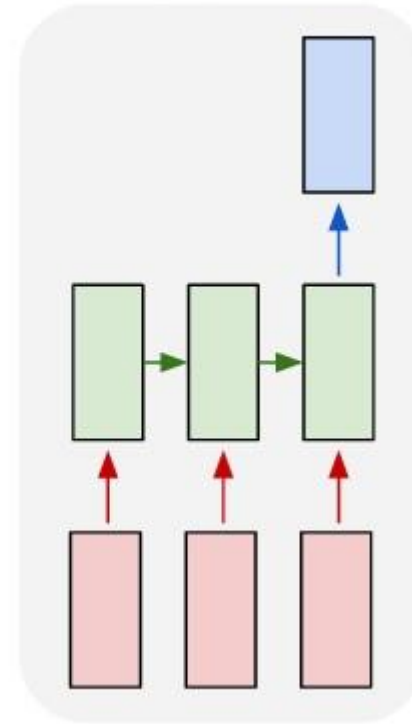


one to one     one to many     many to one

Images from Karpathy

- 1: Conventional MLP
- 2: Sequence *generation*, e.g. image to caption
- 3: Sequence based *prediction or classification,* e.g. Speech recognition, text classification

# Variants
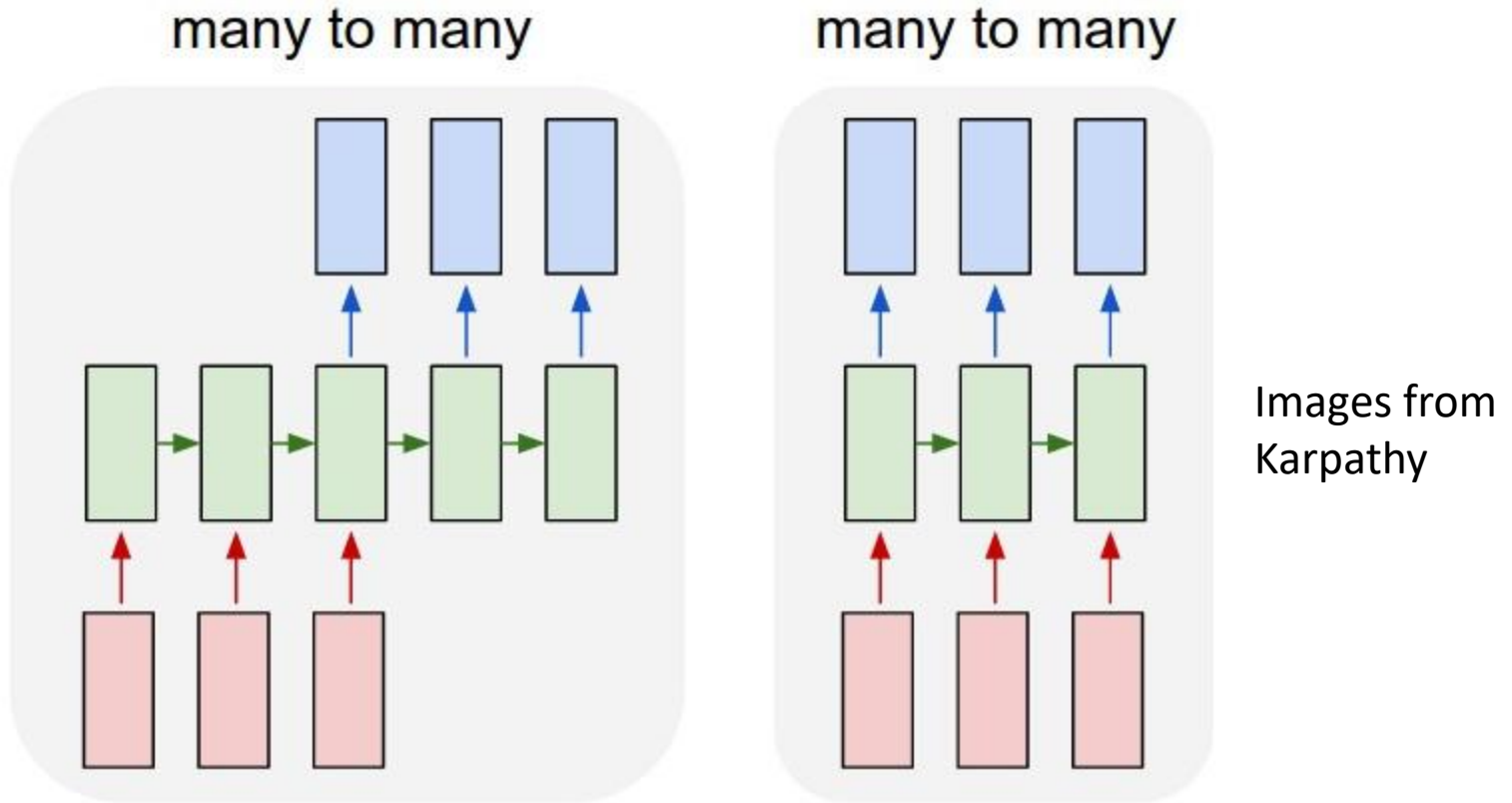


Images from Karpathy

- 1: *Delayed* sequence to sequence
- 2: Sequence to sequence, e.g. stock problem, label prediction
- Etc...

# Variants on recurrent nets
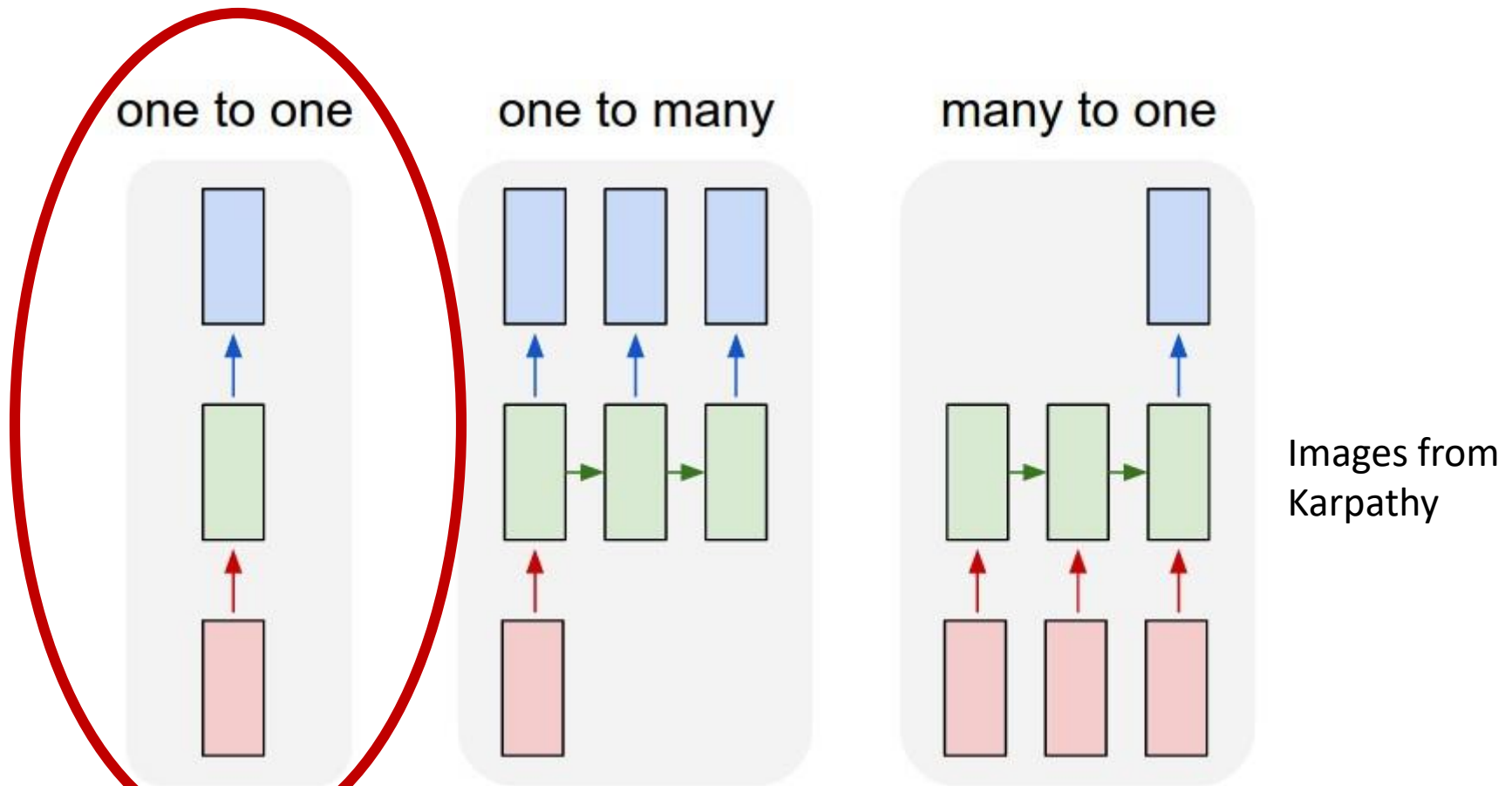


Images from Karpathy

- 1: Conventional MLP
- 2: Sequence *generation*, e.g. image to caption
- 3: Sequence based *prediction or classification,* e.g. Speech recognition, text classification

# Regular MLP
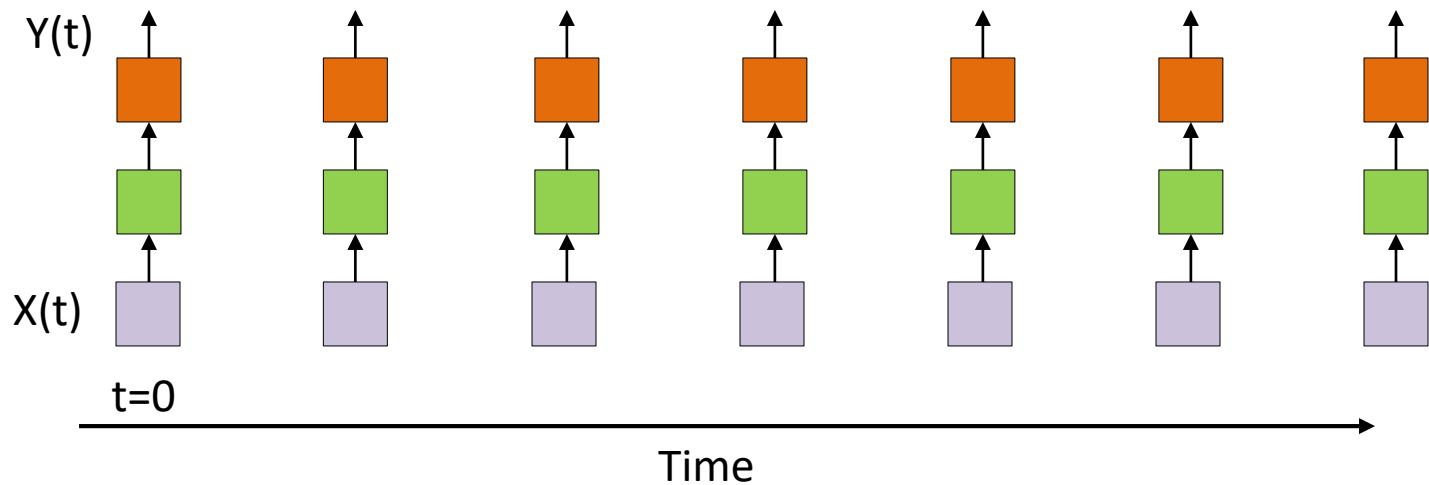


- No recurrence

  - Exactly as many outputs as inputs

  - Every input produces a unique output

# Learning in a Regular MLP
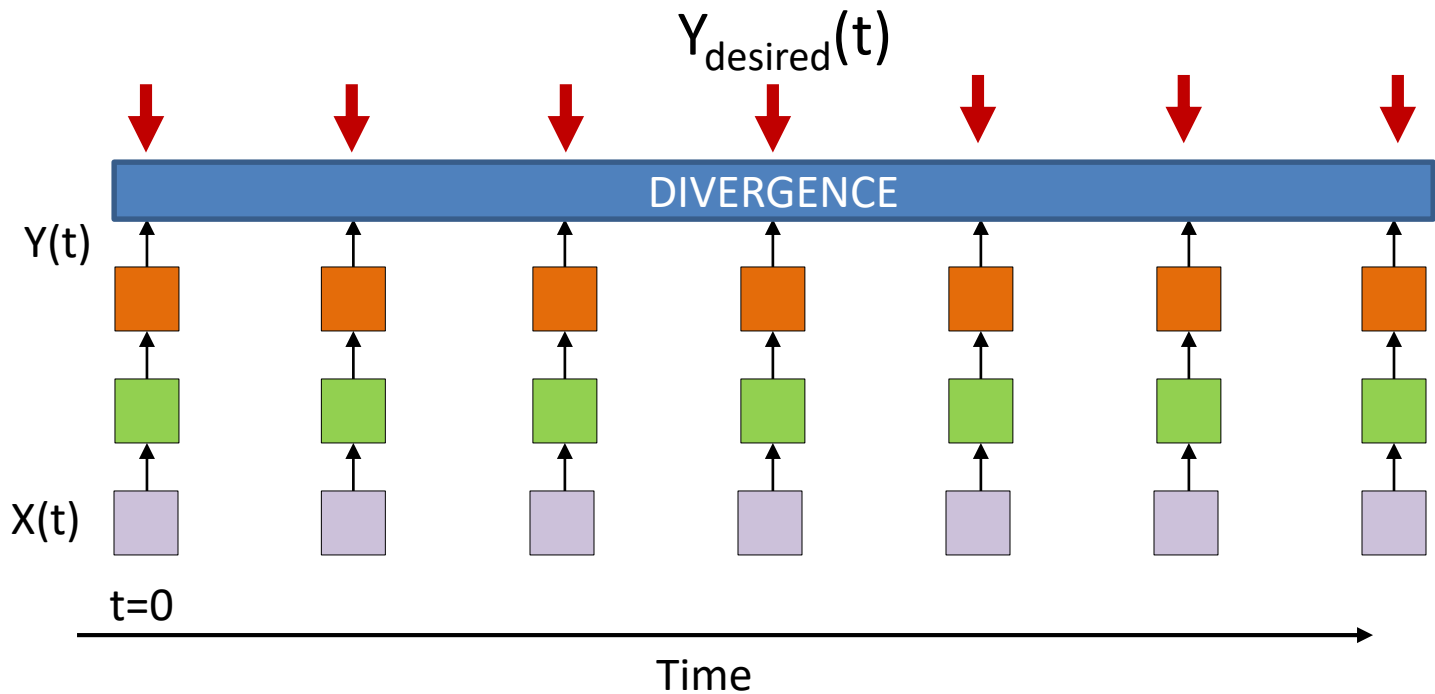


- No recurrence
  - Exactly as many outputs as inputs
    - **One to one correspondence between desired output and actual output**
  - The output at time $t$ is not a function of the output at $t' \neq t$.

# Regular MLP
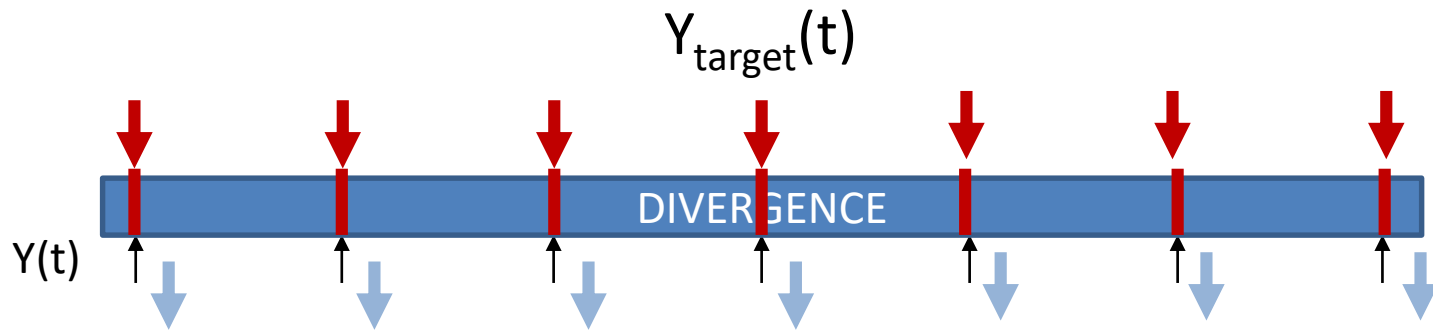


$Y_{target}(t)$

DIVERGENCE

$Y(t)$

- Gradient backpropagated at each time

$$\nabla_{Y(t)} Div\big(Y_{target}(1 \dots T), Y(1 \dots T)\big)$$

- Common assumption:

$$Div\big(Y_{target}(1 \dots T), Y(1 \dots T)\big) = \sum_t w_t Div\big(Y_{target}(t), Y(t)\big)$$

$$\nabla_{Y(t)} Div\big(Y_{target}(1 \dots T), Y(1 \dots T)\big) = w_t \nabla_{Y(t)} Div\big(Y_{target}(t), Y(t)\big)$$

  - $w_t$ is typically set to 1.0
  - This is further backpropagated to update weights etc

# Regular MLP
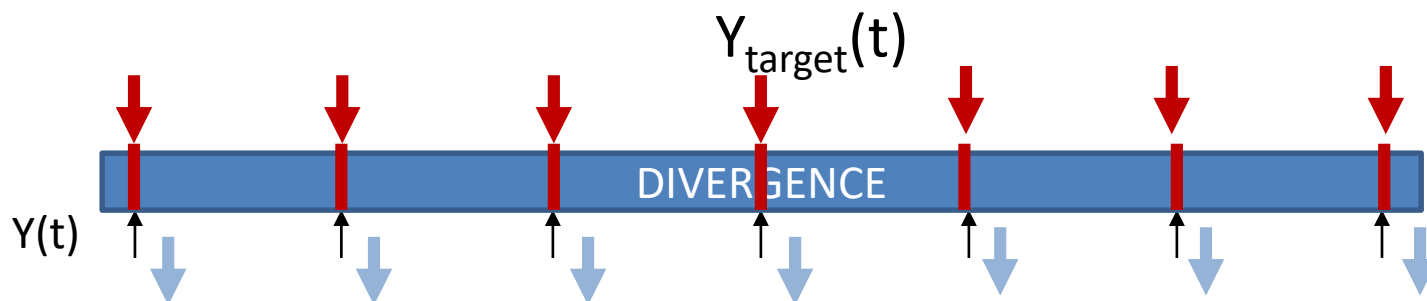


- Gradient backpropagated at each time

$$\nabla_{Y(t)} Div\big(Y_{target}(1 \dots T), Y(1 \dots T)\big)$$

- Common assumption:

$$Div\big(Y_{target}(1 \dots T), Y(1 \dots T)\big) = \sum_t Div\big(Y_{target}(t), Y(t)\big)$$

$$\nabla_{Y(t)} Div\big(Y_{target}(1 \dots T), Y(1 \dots T)\big) = \nabla_{Y(t)} Div\big(Y_{target}(t), Y(t)\big)$$

  – This is further backpropagated to update weights etc

Typical Divergence for classification: $Div\big(Y_{target}(t), Y(t)\big) = Xent(Y_{target}, Y)$

# Variants on recurrent nets
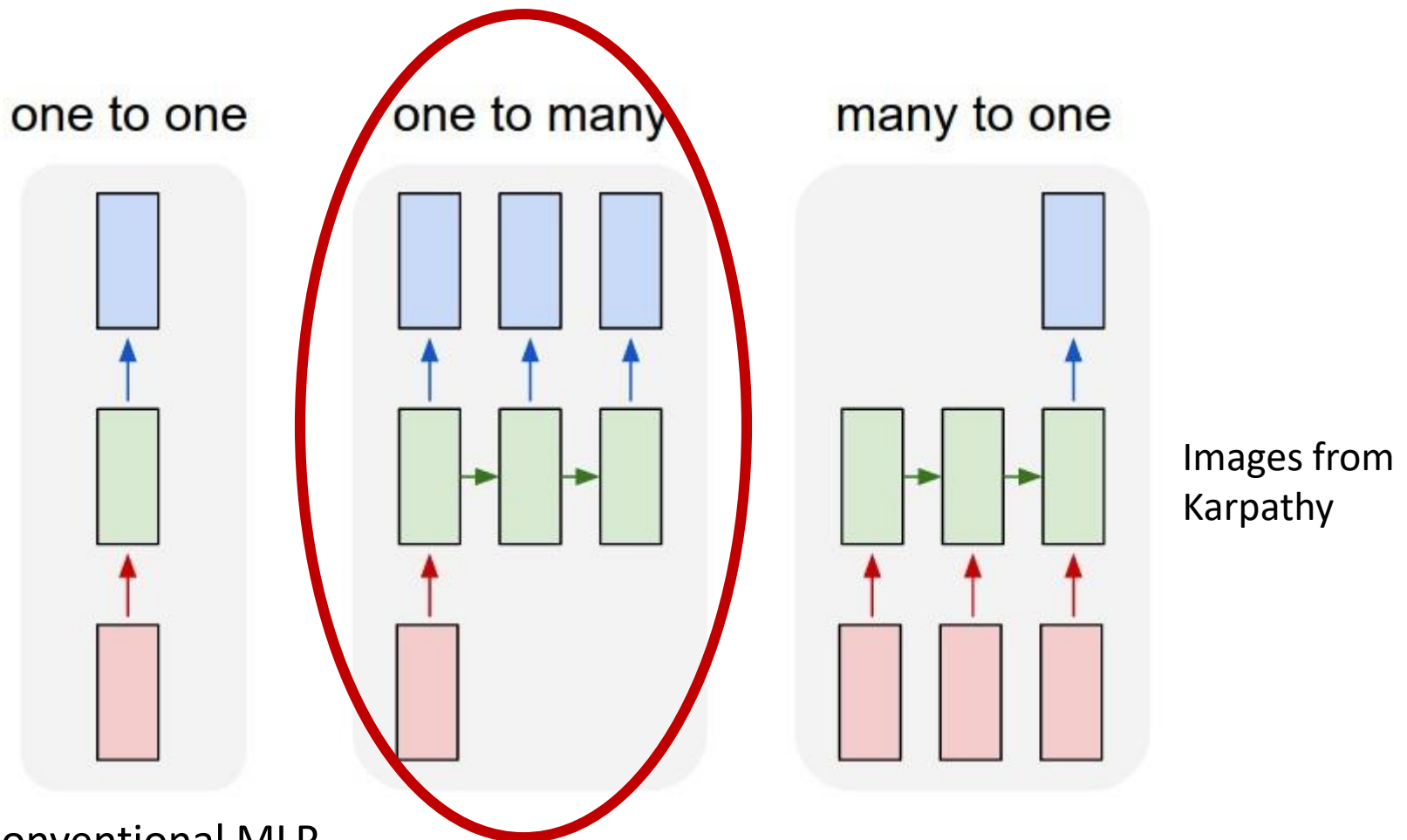


Images from Karpathy

- 1:  Conventional MLP
- 2: Sequence *generation*,  e.g. image to caption
- 3: Sequence based *prediction or classification,* e.g.  Speech recognition, text classification

# Sequence generation
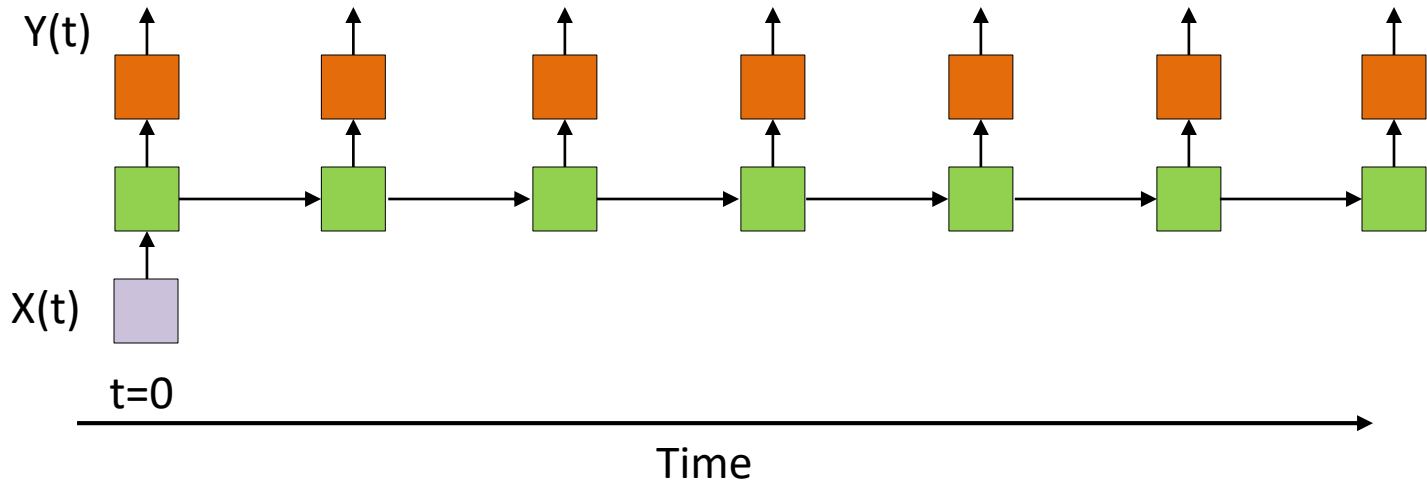


- Single input, sequence of outputs
  - Outputs are recurrently dependent
    - Units may be simple RNNs or LSTM/GRUs
  - A single input produces a sequence of outputs of arbitrary length
    - Until terminated by some criterion, e.g. max length
- Typical example: Predicting the state trajectory of a system (e.g. a robot) in response to an input

# Sequence generation: Learning, Case 1
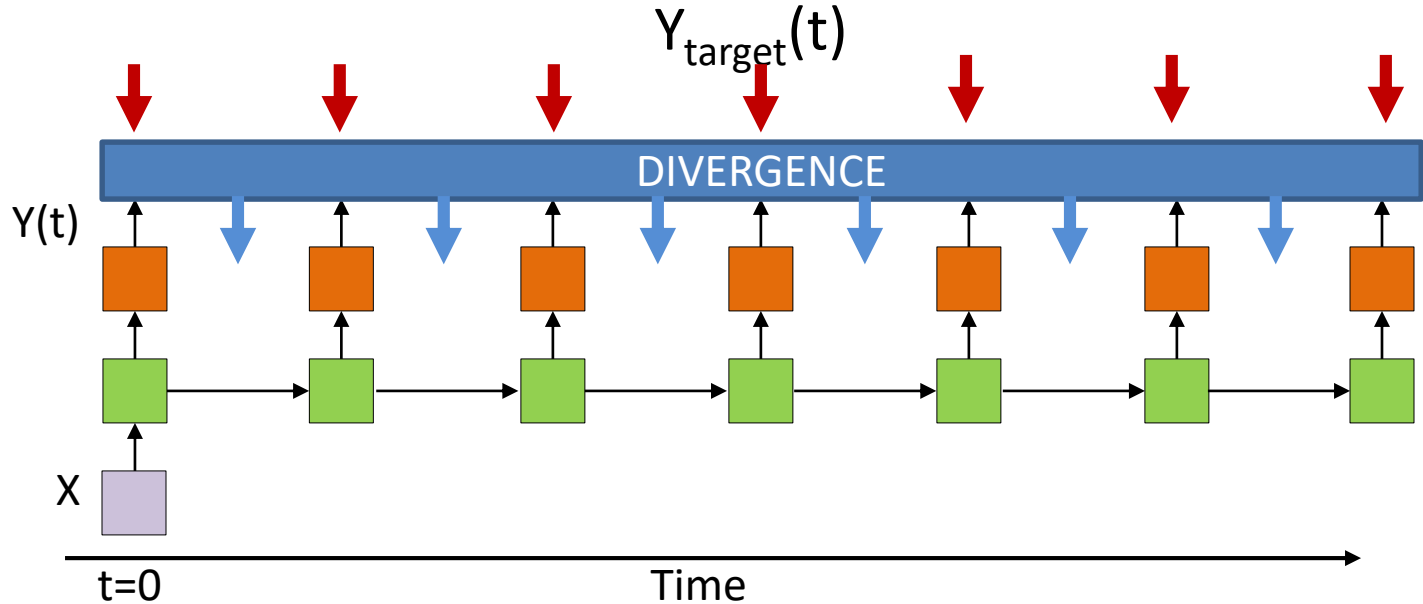


- Learning from training $(X, Y_{target}(1..T))$ pairs
- Gradient backpropagated at each time

$$\nabla_{Y(t)} Div(Y_{target}(1 \dots T), Y(1 \dots T))$$

- Common assumption: ***One-to-one correspondence between desired and actual outputs***

$$Div(Y_{target}(1 \dots T), Y(1 \dots T)) = \sum_t Div(Y_{target}(t), Y(t))$$

$$\nabla_{Y(t)} Div(Y_{target}(1 \dots T), Y(1 \dots T)) = \nabla_{Y(t)} Div(Y_{target}(t), Y(t))$$

31

# Variants



many to many

many to many

Images from Karpathy
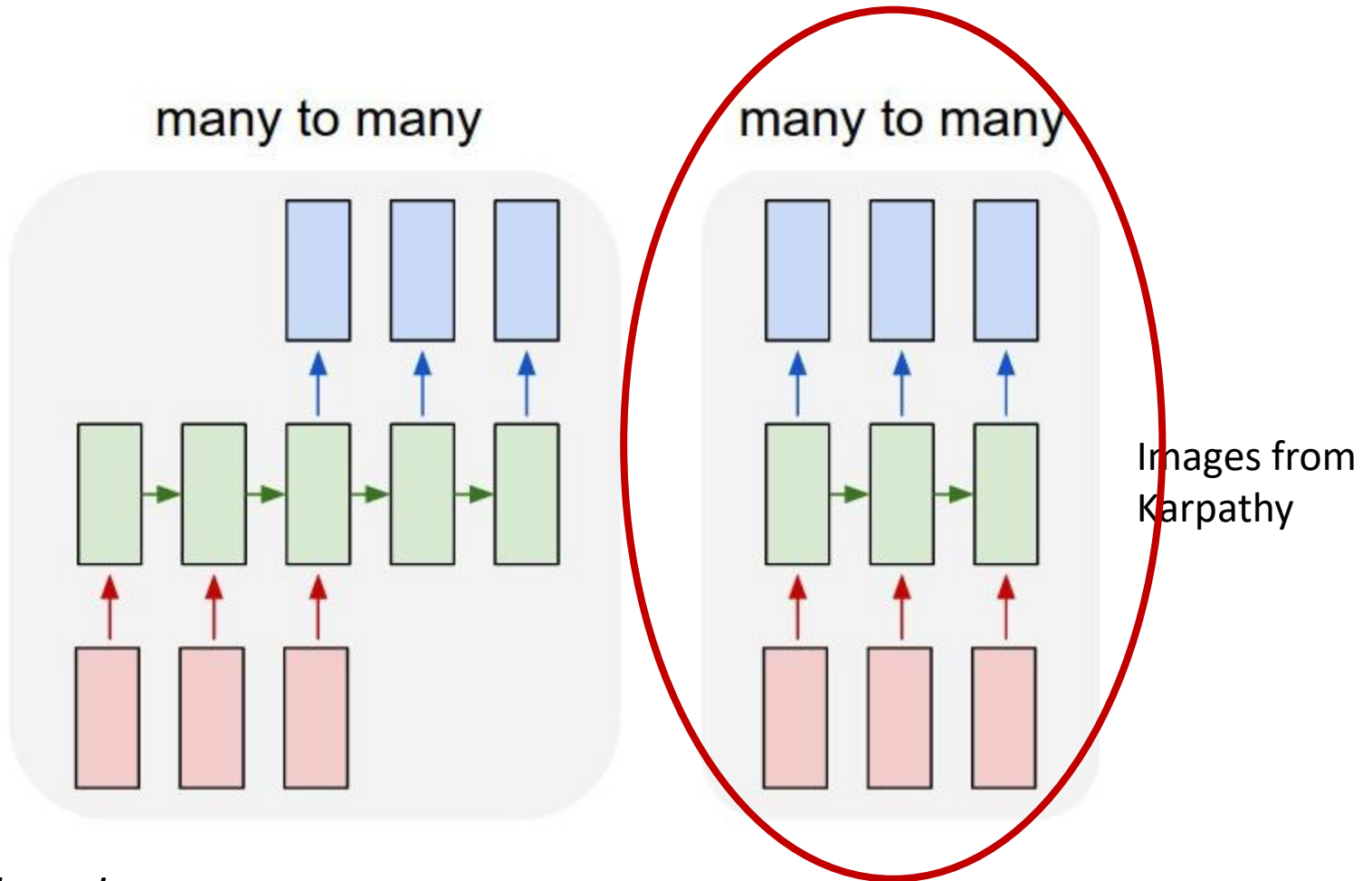
- 1: *Delayed* sequence to sequence
- 2: Sequence to sequence, e.g. stock problem, label prediction
- Etc...

# Recurrent Net
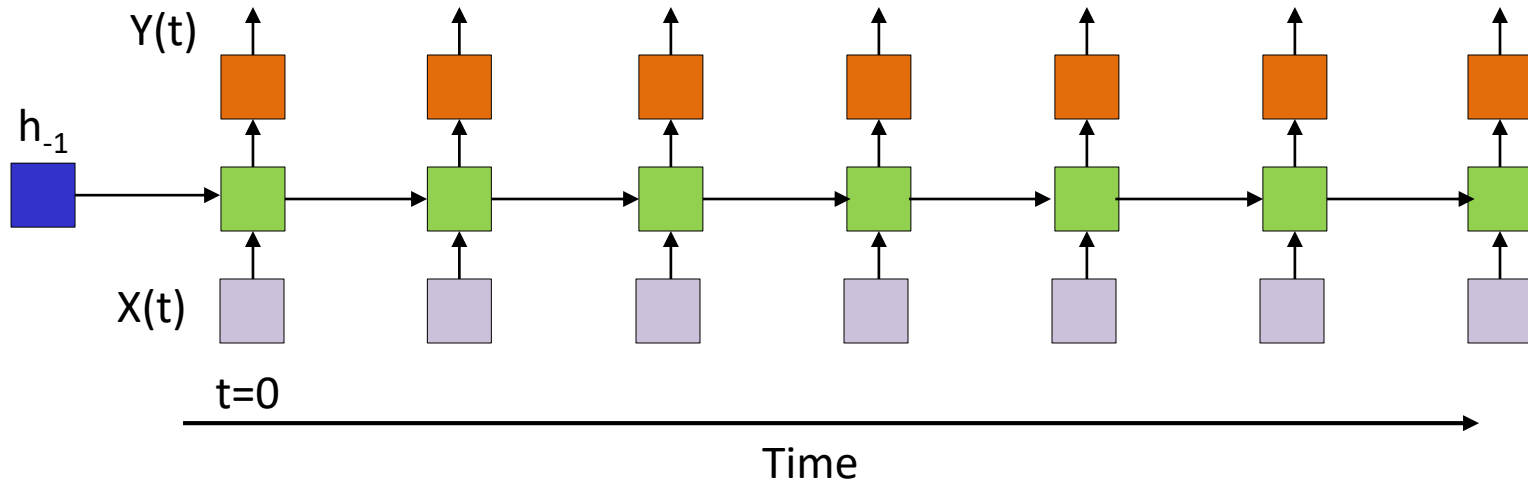


- Simple recurrent computation of outputs
  - Sequence of inputs produces a sequence of outputs
    - Exactly as many outputs as inputs
  - Output computation utilizes recurrence (RNN or LSTM) in hidden layer
    - Also generalizes to other forms of recurrence
- Example:
  - Predicting tomorrow's stocks
  - Predicting the next word…
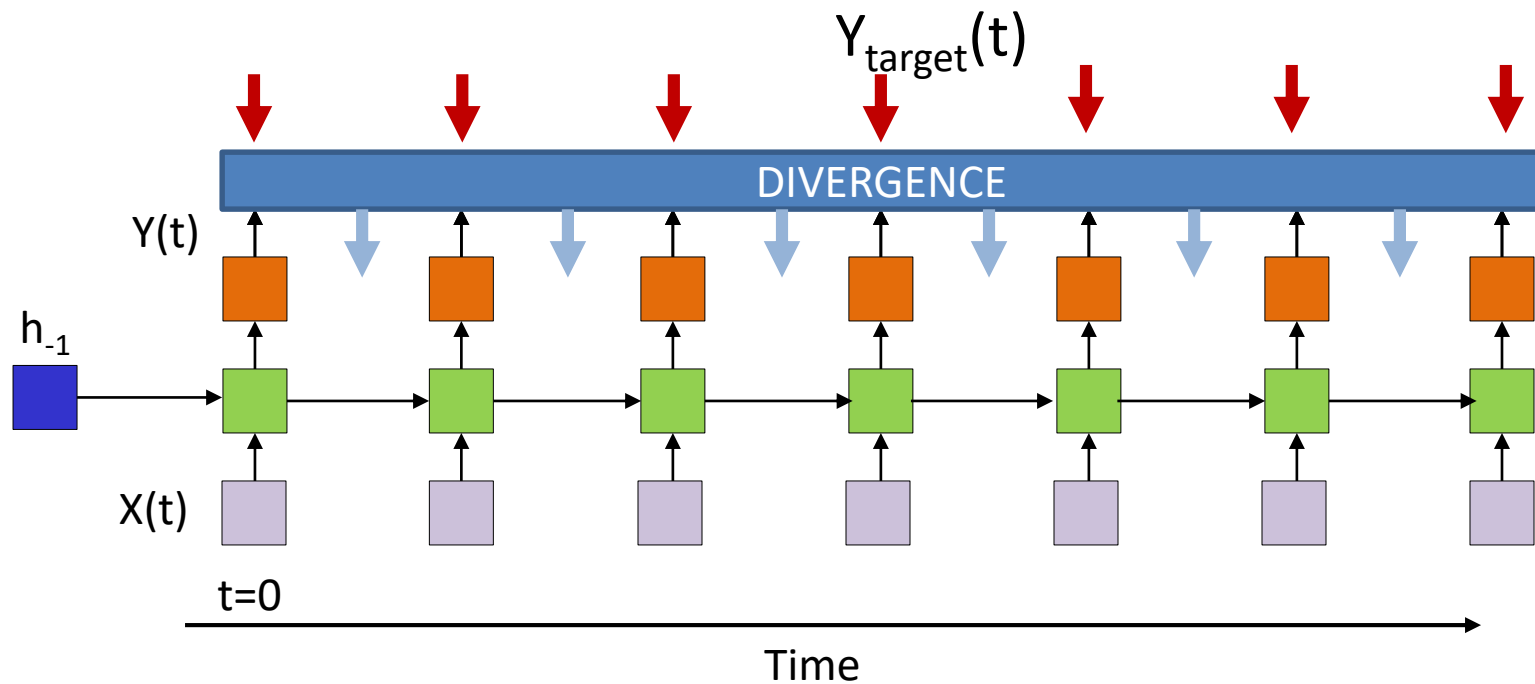
33

# Training Recurrent Net



- Learning from training $(X, Y_{target}(1..T))$ pairs

- Gradient backpropagated at each time

$$\nabla_{Y(t)} Div\big(Y_{target}(1 \dots T), Y(1 \dots T)\big)$$

  - General setting

# Training Recurrent Net
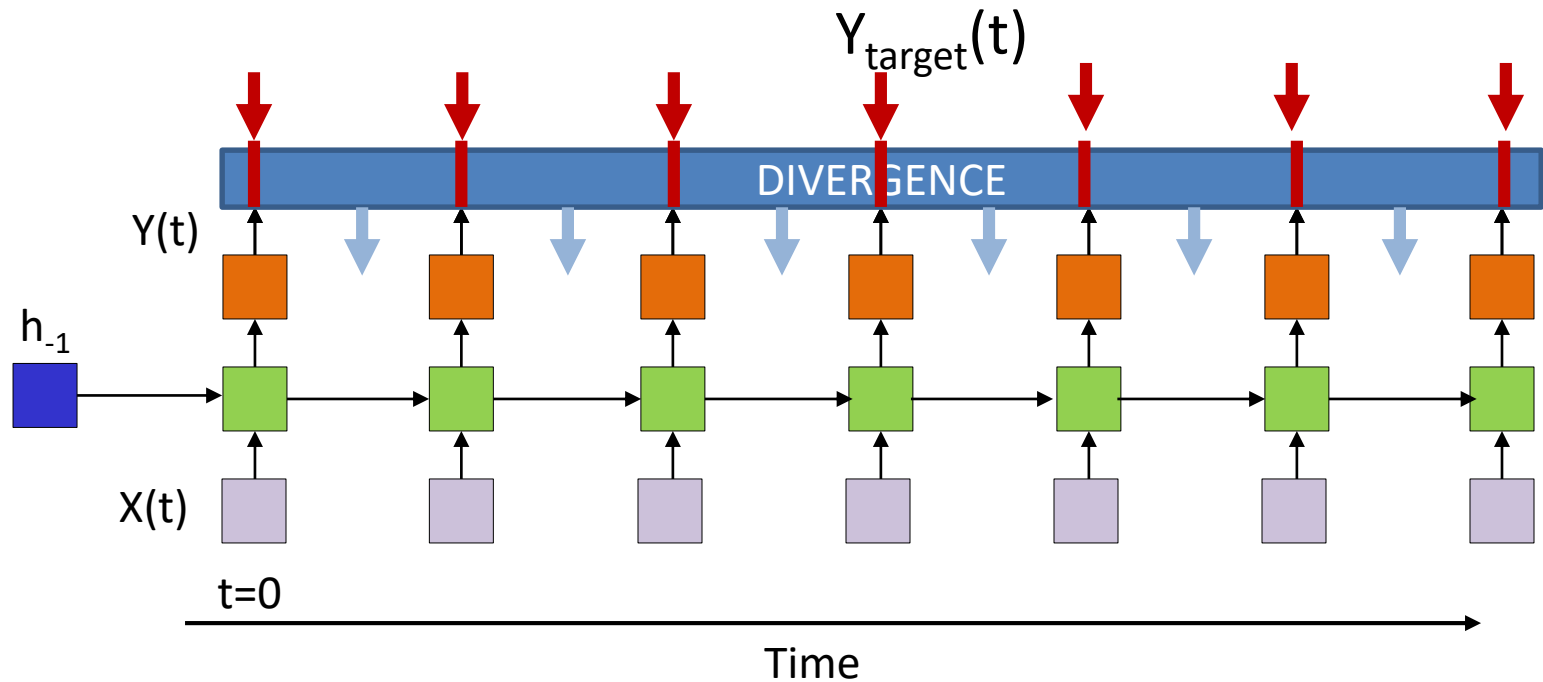


- Usual assumption: ***One-to-one correspondence between desired and actual outputs***

$$Div\big(Y_{target}(1\dots T), Y(1\dots T)\big) = \sum_t Div\big(Y_{target}(t), Y(t)\big)$$

$$\nabla_{Y(t)} Div\big(Y_{target}(1\dots T), Y(1\dots T)\big) = \nabla_{Y(t)} Div\big(Y_{target}(t), Y(t)\big)$$

# Simple recurrence example: Text Modelling
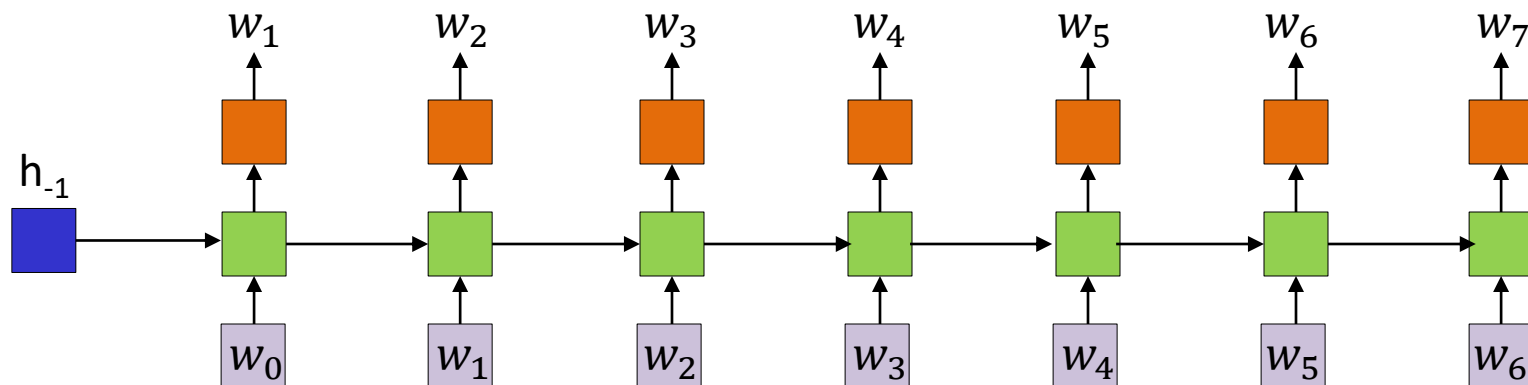


- Learn a model that can predict the next character given a sequence of characters
  - Or, at a higher level, words
- After observing inputs $w_0 \dots w_k$ it predicts $w_{k+1}$

# Simple recurrence example: Text Modelling

Figure from Andrej Karpathy.

Input:  Sequence of characters (presented as one-hot vectors).

Target output after observing "h e l l" is "o"
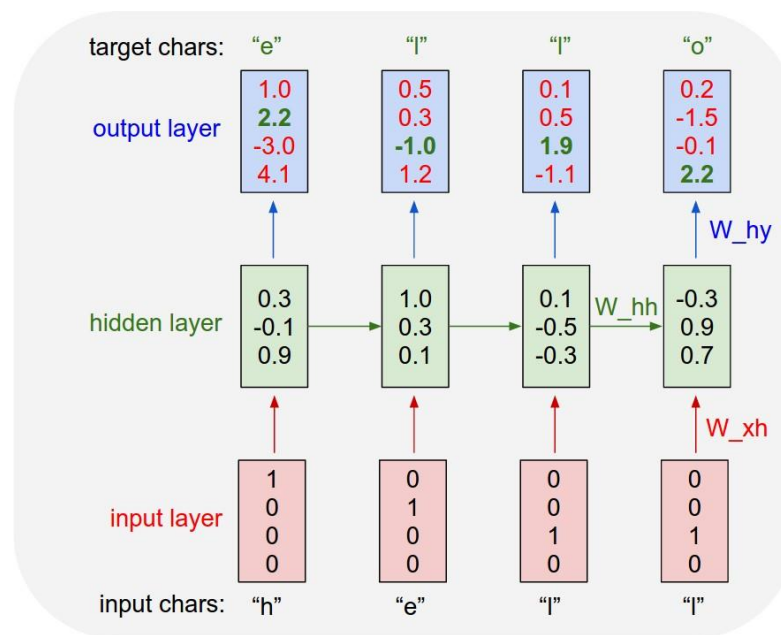


- Input presented as one-hot vectors
  - Actually "embeddings" of one-hot vectors
- Output: probability distribution over characters
  - Must ideally peak at the target character

# Training
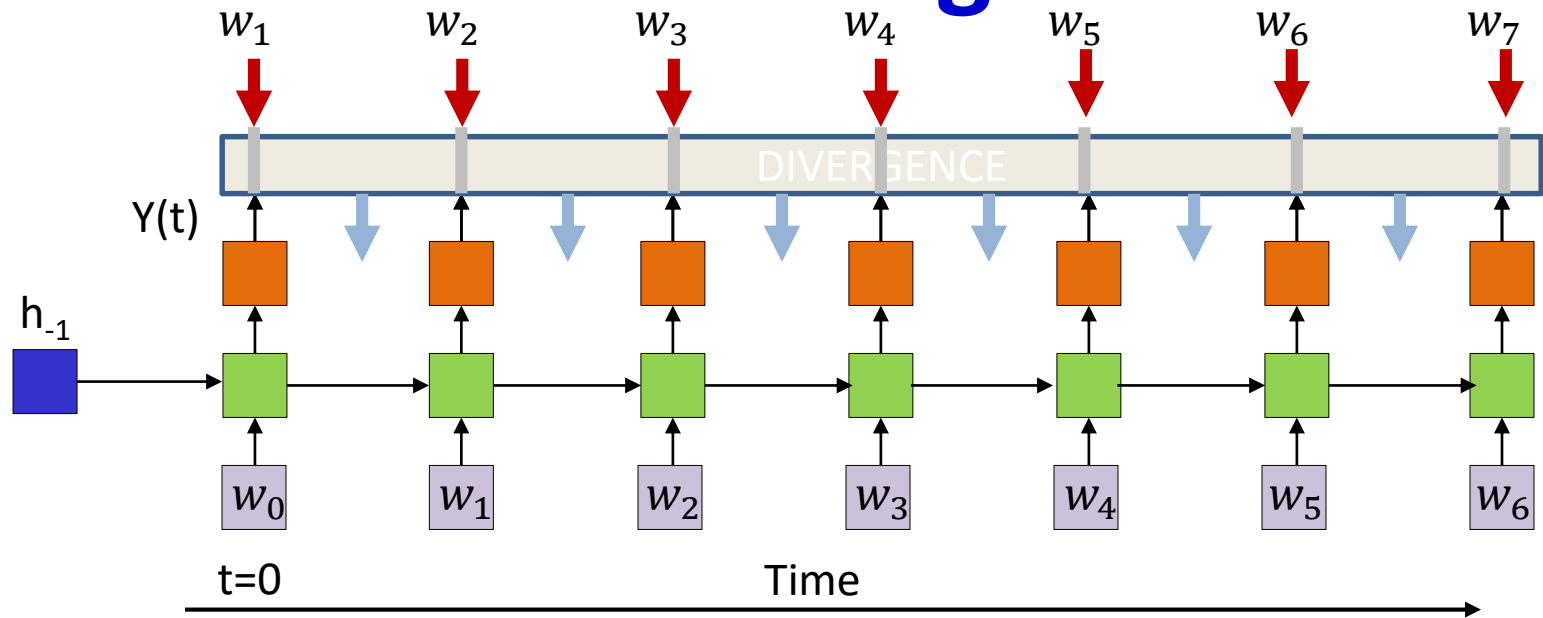


- Input: symbols as one-hot vectors
  - Dimensionality of the vector is the size of the "vocabulary"
- Output: Probability distribution over symbols
$$Y(t,i) = P(V_i | w_0 \dots w_{t-1})$$
  - $V_i$ is the i-th symbol in the vocabulary
- Divergence

$$Div\big(Y_{target}(1 \dots T), Y(1 \dots T)\big) = \sum_t Xent\big(Y_{target}(t), Y(t)\big) = -\sum_t \log Y(t, w_{t+1})$$

The probability assigned to the correct next word

38

# Brief Segue..

- Language Modeling using neural networks

# Which open source project?

```c
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECON
  return segtable;
}
```

Language modelling using RNNs

Four score and seven years ???

A B R A H A M L I N C O L ??

- Problem: Given a sequence of words (or characters) predict the next one

Language modelling: Representing words

- Represent words as one-hot vectors
  – Pre-specify a vocabulary of N words in fixed (e.g. lexical) order
    - *E.g. [A AARDVARD AARON ABACK ABACUS... ZZYP]*
  – Represent each word by an N-dimensional vector with N-1 zeros and a single 1 (in the position of the word in the ordered list of words)

- Characters can be similarly represented
  – English will require about 100 characters, to include both cases, special characters such as commas, hyphens, apostrophes, etc., and the space character

Predicting words

Four score and seven years ???

$$W_n = f(W\_1, W_1, \ldots, W_{n-1})$$
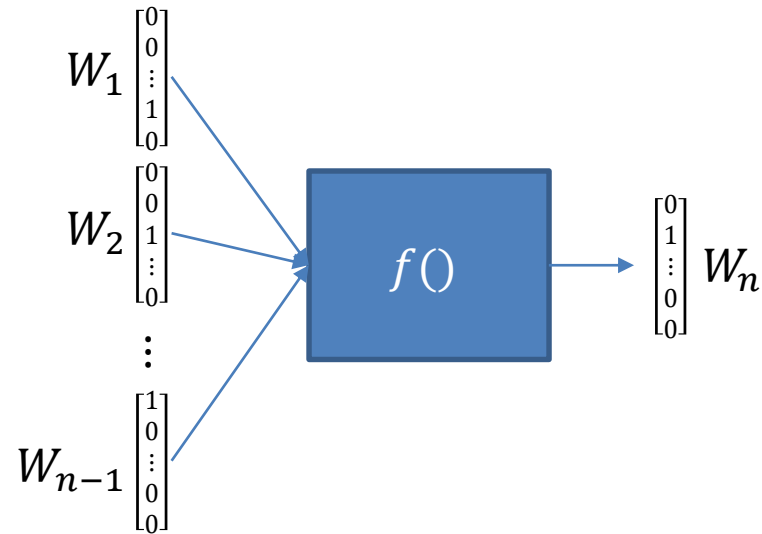
Nx1 one-hot vectors

- Given one-hot representations of $W_1 \ldots W_{n-1}$, predict W_n

Predicting words

Four score and seven years ???

$$W_n = f(W\_1, W_1, \ldots, W_{n-1})$$

Nx1 one-hot vectors
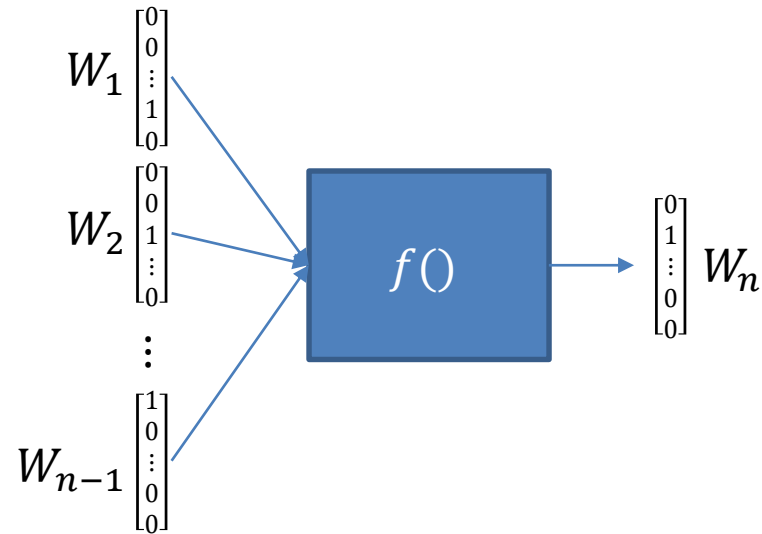
- Given one-hot representations of $W_1 \ldots W_{n-1}$, predict W_n

- **Dimensionality problem:** All inputs $W_1 \ldots W_{n-1}$ are both very high-dimensional and very sparse

The one-hot representation

(1,0,0)

(0,1,0)

(0,0,1)
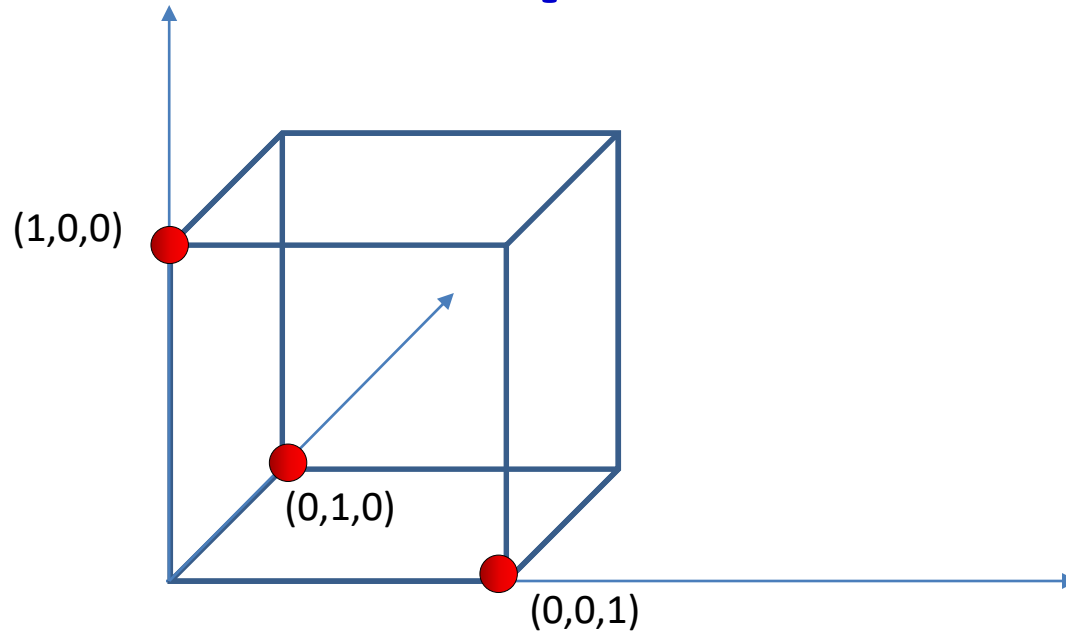
- The one hot representation uses only N corners of the 2^N corners of a unit cube
  - Actual volume of space used = 0
    - $(1, \varepsilon, \delta)$ has no meaning except for $\varepsilon = \delta = 0$
  - Density of points: $\mathcal{O}\left(\frac{N}{r^N}\right)$
- This is a tremendously inefficient use of dimensions

Why one-hot representation
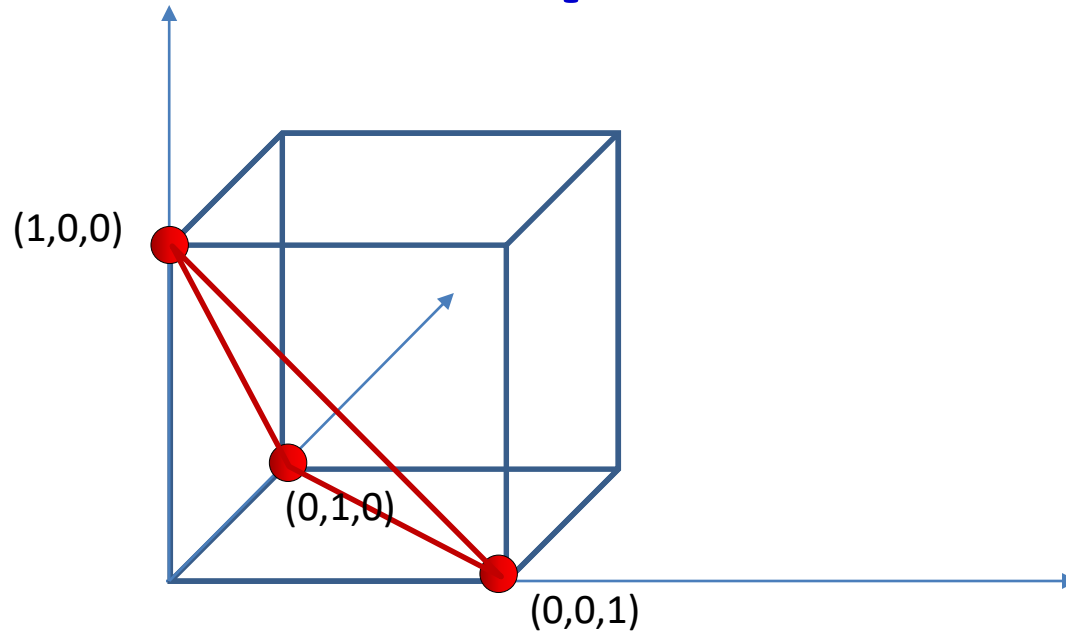
- The one-hot representation makes no assumptions about the relative importance of words
  - All word vectors are the same length
- It makes no assumptions about the relationships between words
  - The distance between every pair of words is the same

Solution to dimensionality problem
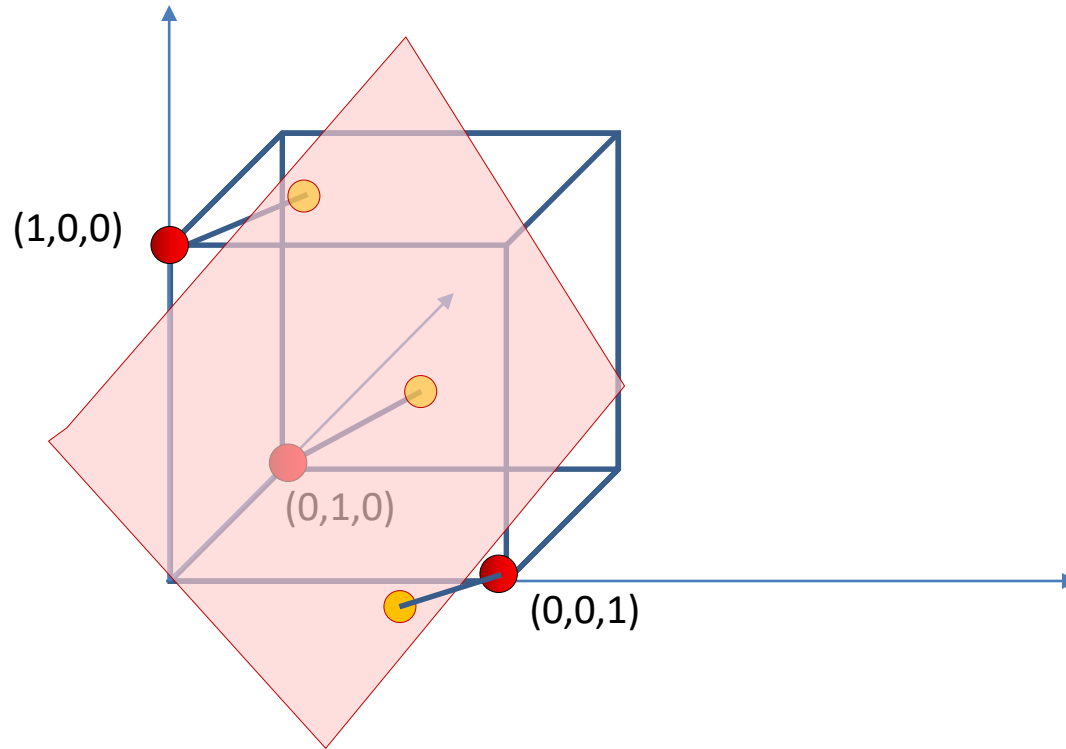
- Project the points onto a lower-dimensional subspace
  - The volume used is still 0, but density can go up by many orders of magnitude
    - Density of points: $\mathcal{O}\left(\frac{N}{r^M}\right)$

Solution to dimensionality problem
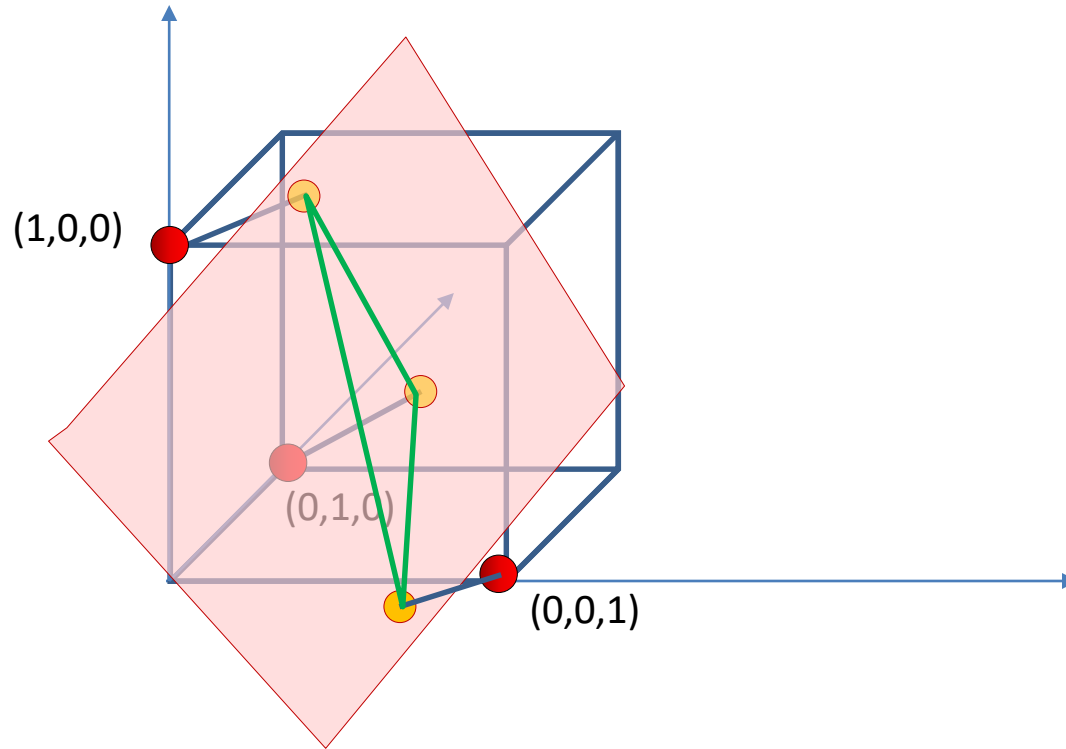
- Project the points onto a lower-dimensional subspace
  - The volume used is still 0, but density can go up by many orders of magnitude
    - Density of points: $\mathcal{O}\left(\frac{N}{r^M}\right)$
  - If properly learned, the distances between projected points will capture semantic relations between the words
    - This will also require linear transformation (stretching/shrinking/rotation) of the subspace

The *Projected* word vectors

Four score and seven years ???

$$W_n = f(PW_1, PW_2, \ldots, PW_{n-1})$$
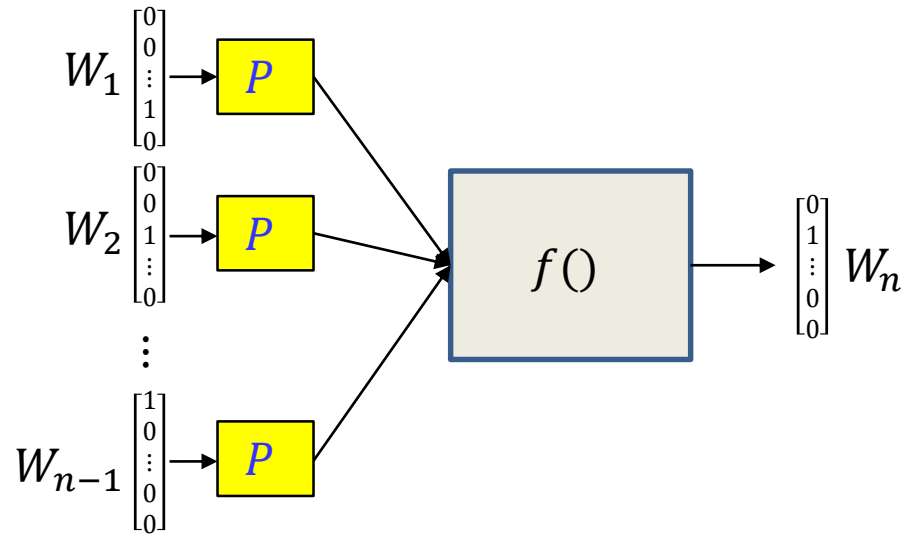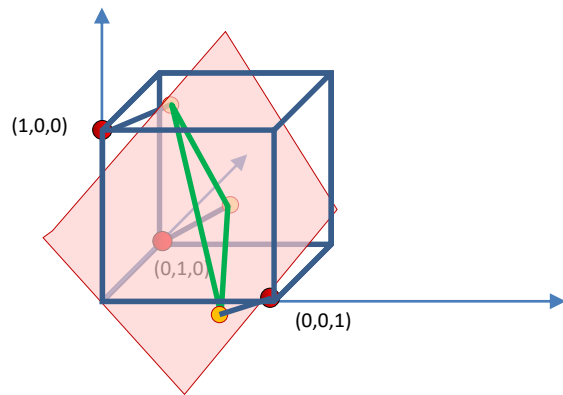
- *Project* the N-dimensional one-hot word vectors into a lower-dimensional space
  - Replace every one-hot vector W_i by PW_i
  - P is an $M \times N$ matrix
  - PW_i is now an M-dimensional vector
  - *Learn P* using an appropriate objective
    - Distances in the projected space will reflect relationships imposed by the objective

49

"Projection"

$$W_n = f(PW_1, PW_2, \ldots, PW_{n-1})$$
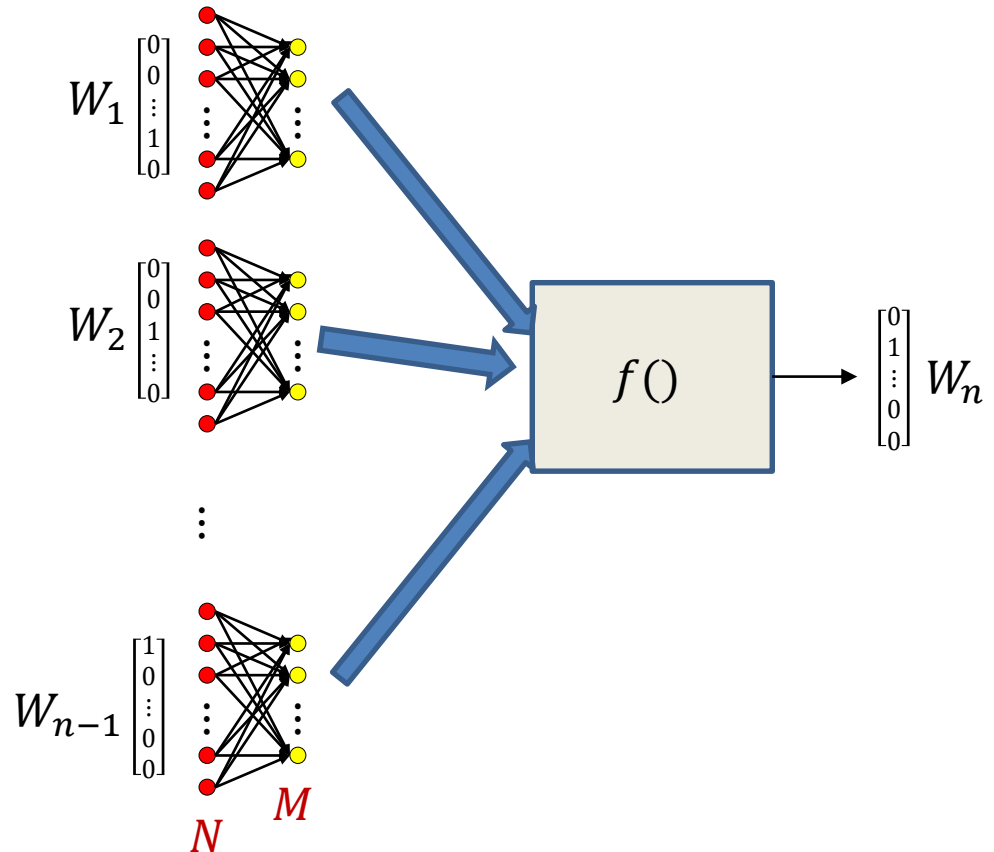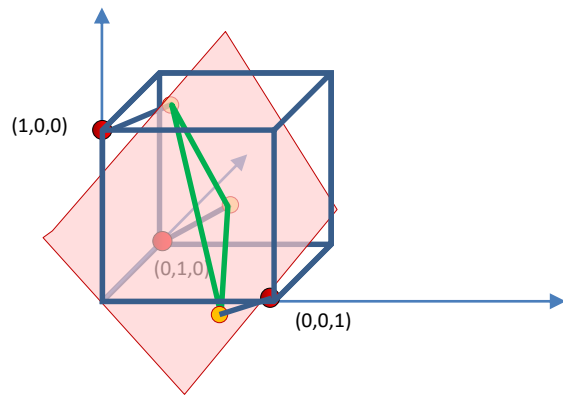
- *P* is a simple linear transform
- A single transform can be implemented as a layer of M neurons with linear activation
- The transforms that apply to the individual inputs are all M-neuron linear-activation subnets with tied weights

Predicting words: The TDNN model
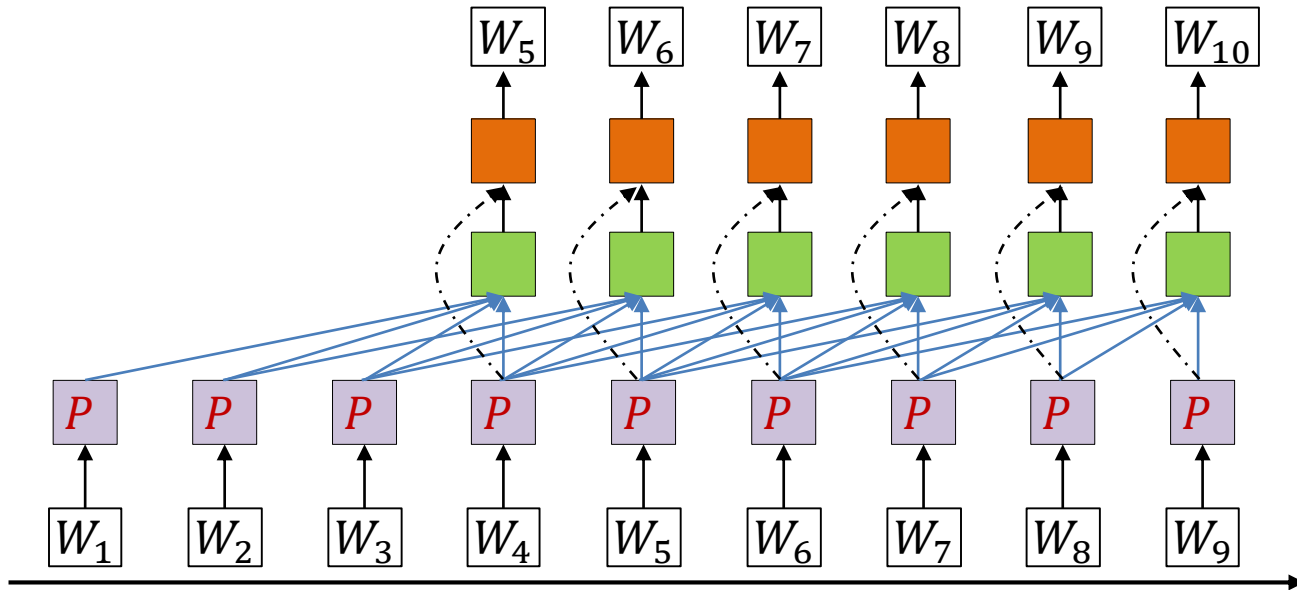
- Predict each word based on the past N words
  - "A neural probabilistic language model", Bengio et al. 2003
  - Hidden layer has Tanh() activation, output is softmax

- One of the outcomes of learning this model is that we also learn low-dimensional representations PW of words

Alternative models to learn projections
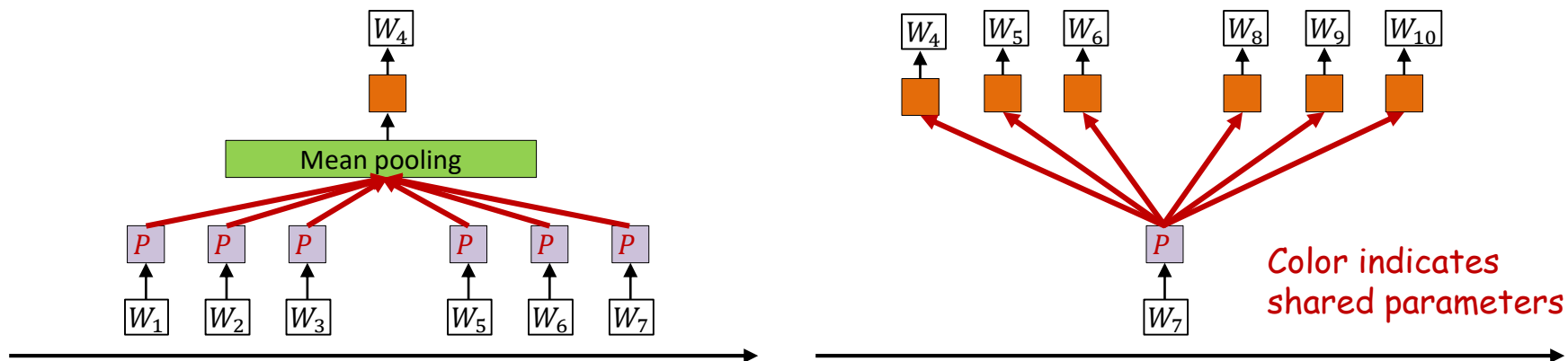
- Soft bag of words: Predict word based on words in immediate context
  - Without considering specific position
- Skip-grams: Predict adjacent words based on current word
- More on these in a future recitation

Embeddings: Examples

Country and Capital Vectors Projected by PCA
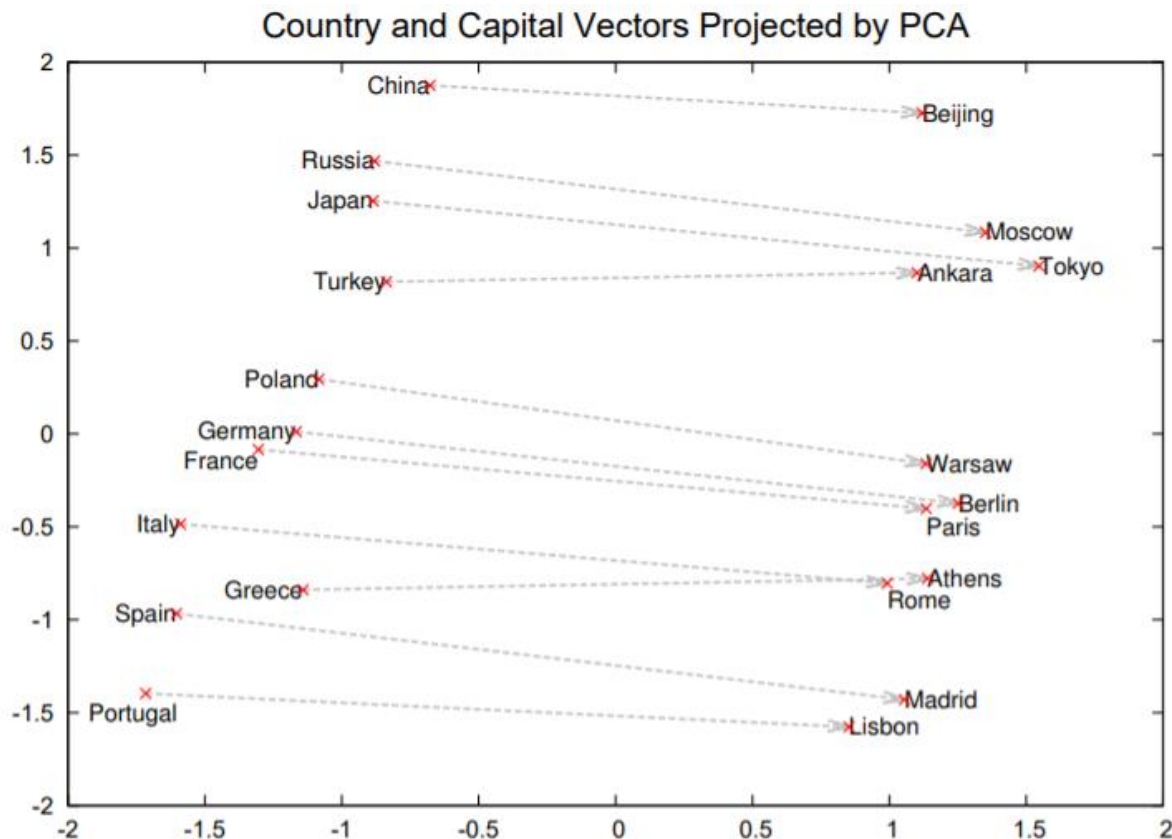
Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

- From Mikolov et al., 2013, "Distributed Representations of Words and Phrases and their Compositionality"

Generating Language: The model
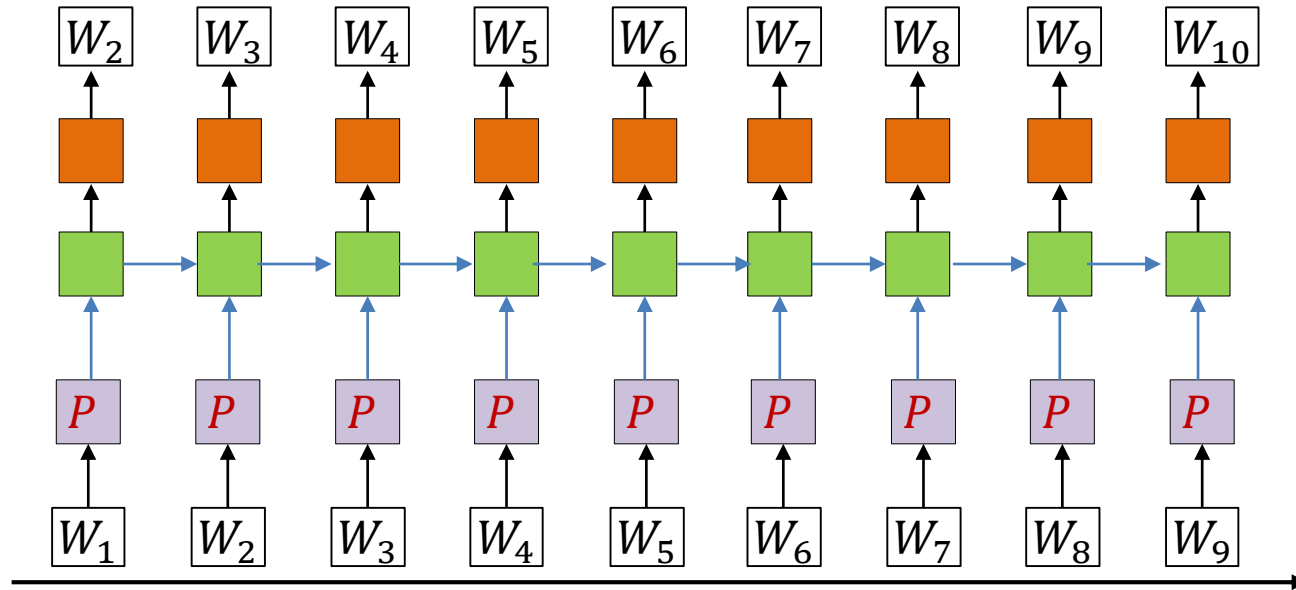
- The hidden units are (one or more layers of) LSTM units
- Trained via backpropagation from a lot of text

Generating Language: Synthesis
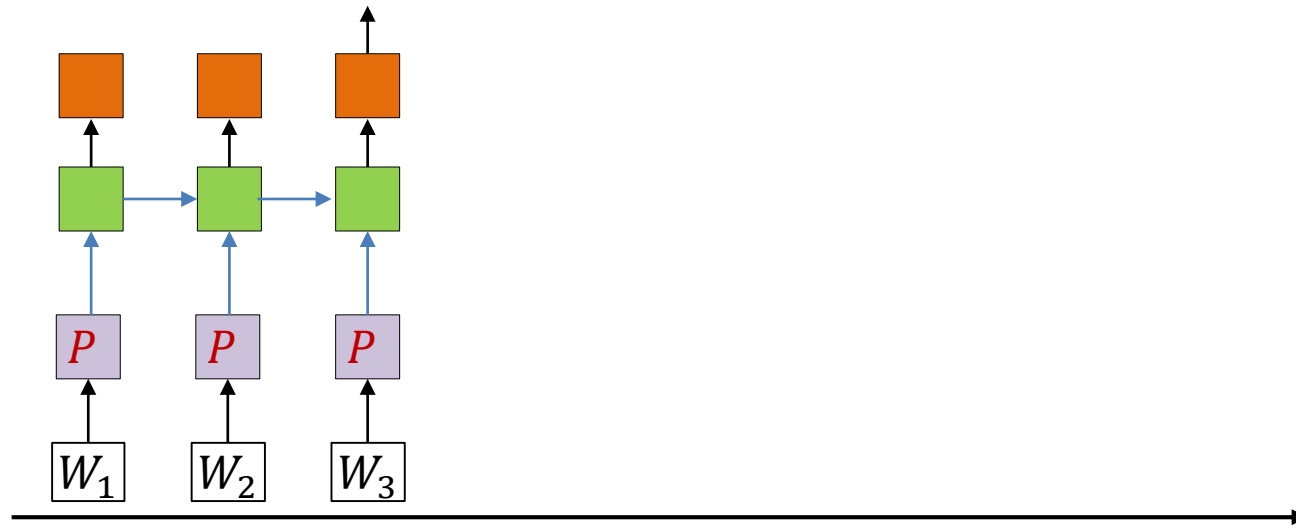
- On trained model : Provide the first few words
  - One-hot vectors
- After the last input word, the network generates a probability distribution over words
  - Outputs an N-valued probability distribution rather than a one-hot vector

Generating Language: Synthesis
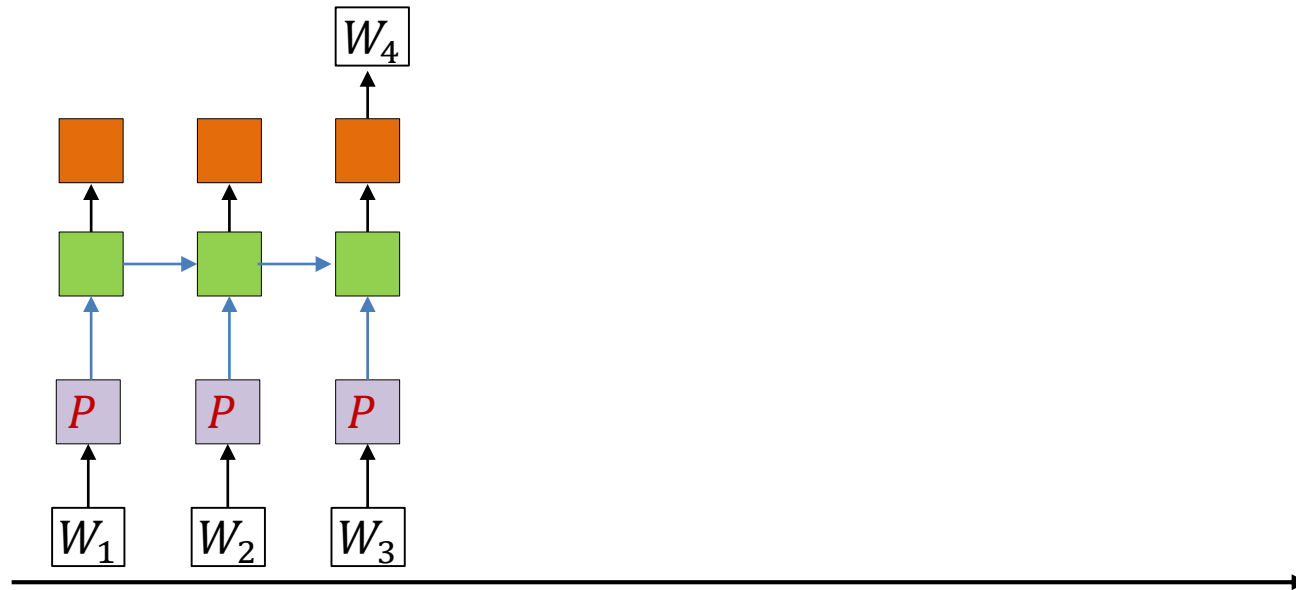
- On trained model : Provide the first few words
  - One-hot vectors
- After the last input word, the network generates a probability distribution over words
  - Outputs an N-valued probability distribution rather than a one-hot vector
- Draw a word from the distribution
  - And set it as the next word in the series

Generating Language: Synthesis
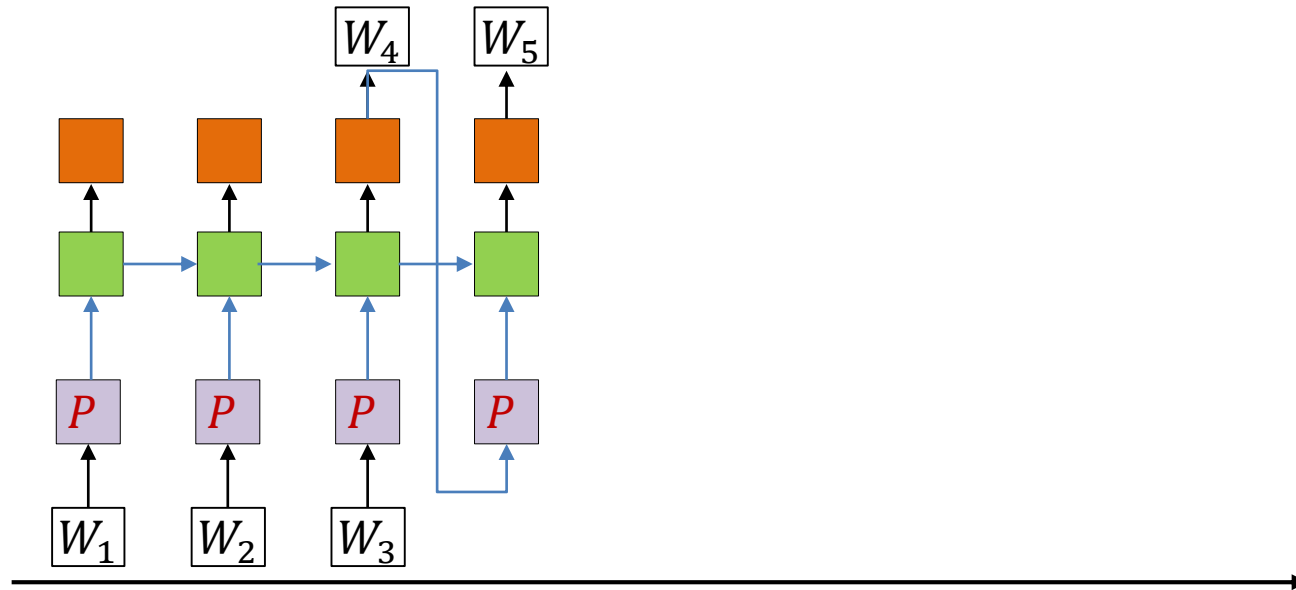
- Feed the drawn word as the next word in the series
  - And draw the next word from the output probability distribution

Generating Language: Synthesis
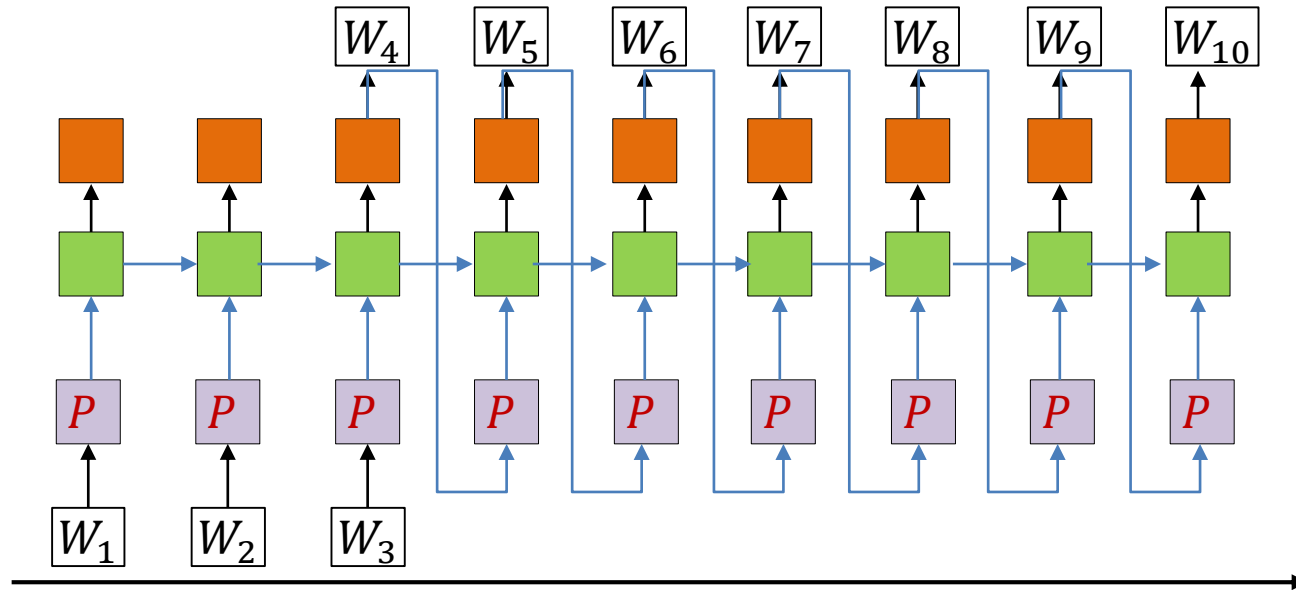
- Feed the drawn word as the next word in the series
  - And draw the next word from the output probability distribution
- Continue this process until we terminate generation
  - In some cases, e.g. generating programs, there may be a natural termination

58

Which open source project?

```c
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECON
  return segtable;
}
```

Trained on linux source code

Actually uses a *character-level* model (predicts character sequences)

59

Composing music with RNN
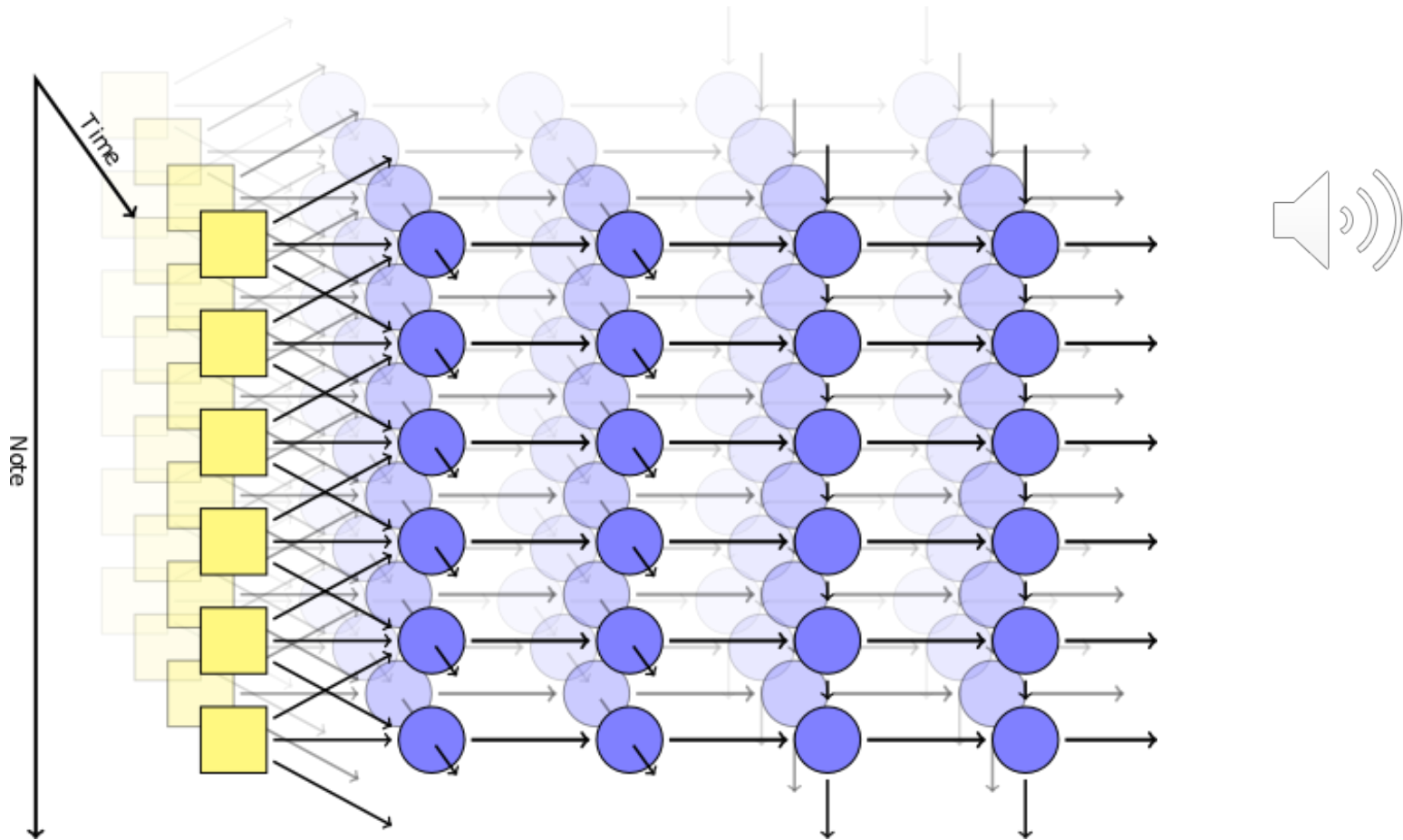
Returning to our problem

- Divergences are harder to define in other scenarios..

Variants on recurrent nets
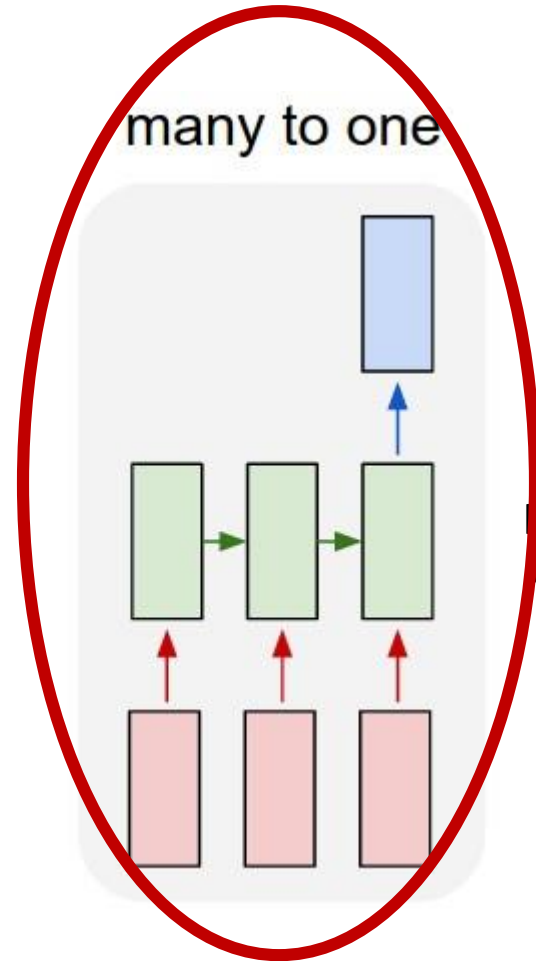
one to one

one to many

many to one

Images from Karpathy

- 1: Conventional MLP
- 2: Sequence *generation*, e.g. image to caption
- 3: Sequence based *prediction or classification,* e.g. Speech recognition, text classification

Example..
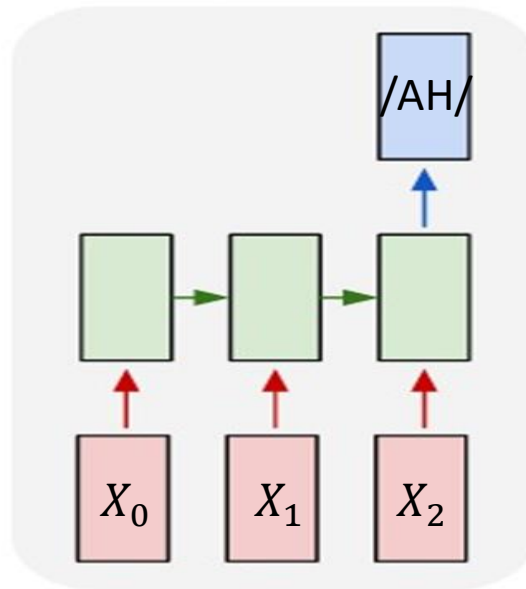
- Speech recognition
- Input : Sequence of feature vectors (e.g. Mel spectra)
- Output: Phoneme ID at the end of the sequence

Issues: Forward pass
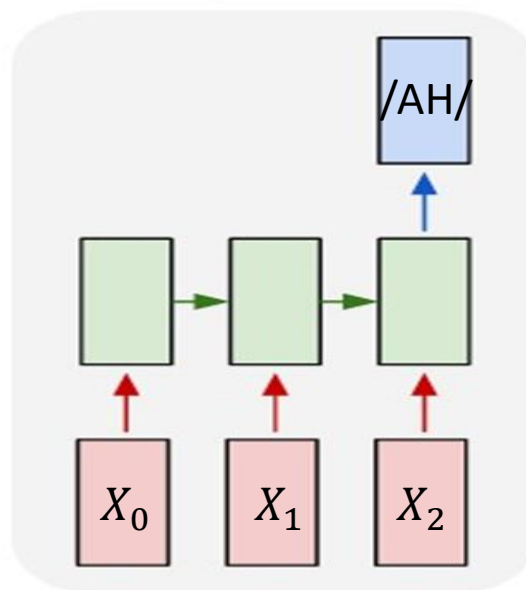
- Exact input sequence provided
  - Output generated when the last vector is processed
    - Output is a probability distribution over phonemes

- But what about at *intermediate stages?*

Forward pass
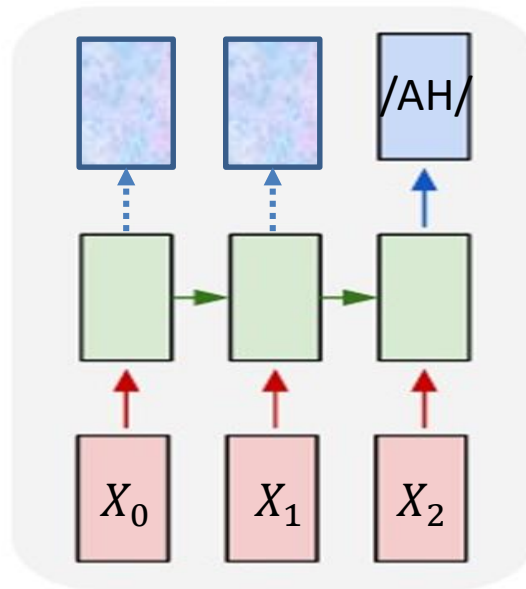
- Exact input sequence provided
  - Output generated when the last vector is processed
    - Output is a probability distribution over phonemes

- Outputs are actually produced for *every* input
  - We only *read* it at the end of the sequence

Training
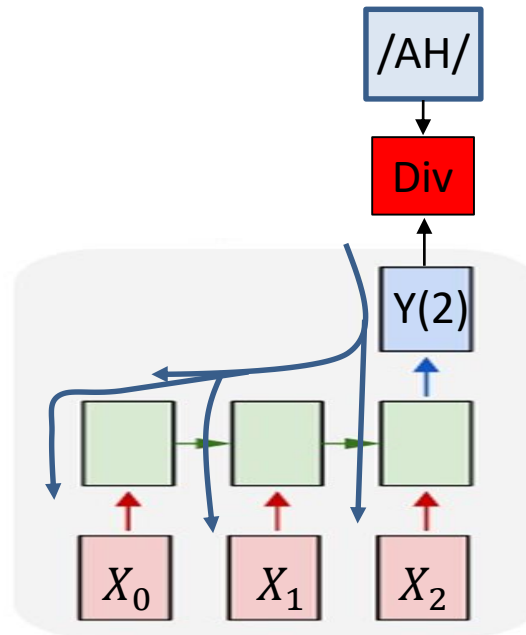
- The Divergence is only defined at the final input

$$- DIV\big(Y_{target}, Y\big) = Xent(Y(T), Phoneme)$$

- This divergence must propagate through the net to update all parameters

Training

/AH/

Div

Y(2)

X_0 X_1 X_2
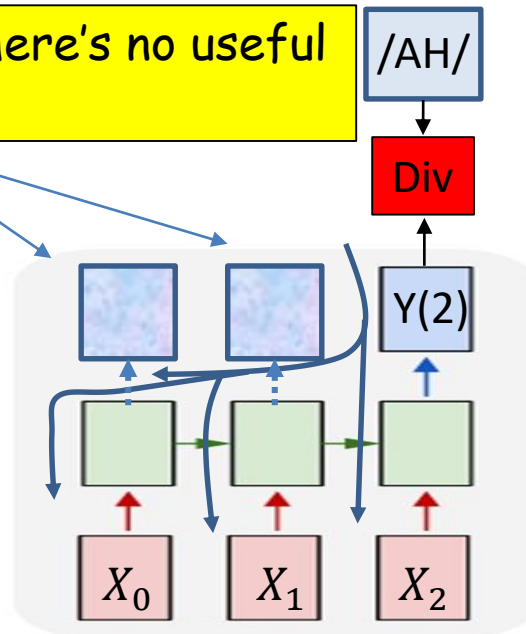
- The Divergence is only defined at the final input
  - $DIV\left(Y_{target}, Y\right) = Xent(Y(T), Phoneme)$
- This divergence must propagate through the net to update all parameters

Training

Fix: Use these outputs too.

These too must ideally point to the correct phoneme
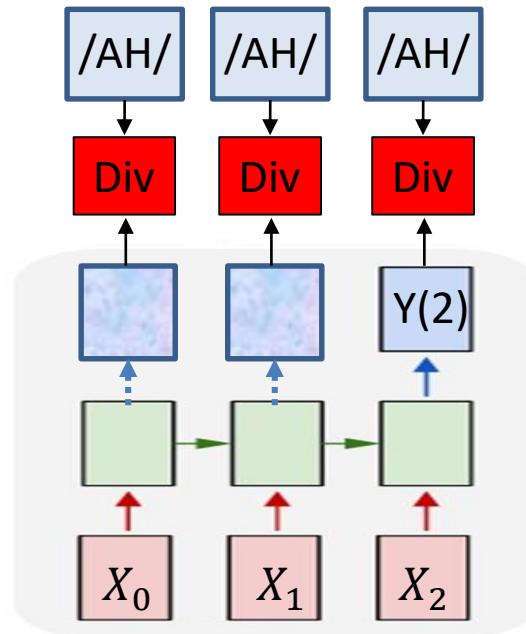
- Define the divergence everywhere

$$DIV\left(Y_{target}, Y\right) = \sum_t w_t Xent(Y(t), Phoneme)$$

  – Typical weighting scheme: all are equally important

- This reduces it to a previously seen problem for training

A more complex problem
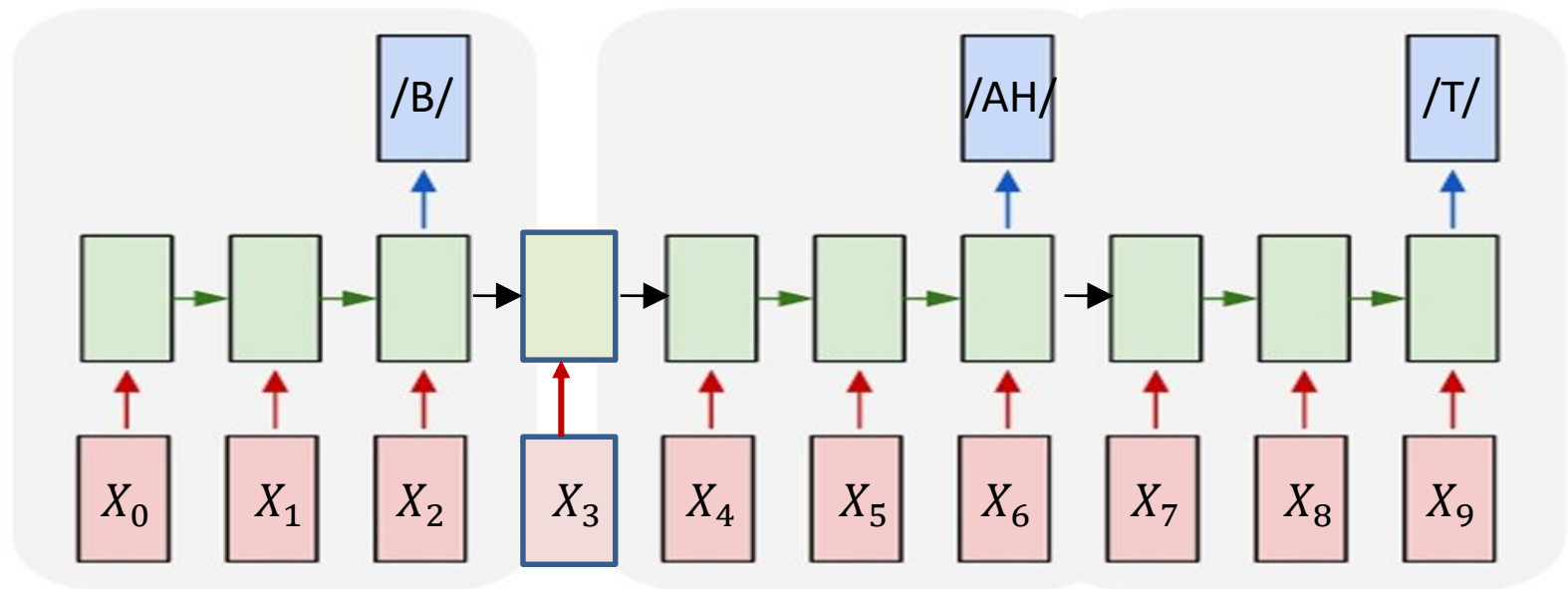
- Objective: Given a sequence of inputs, asynchronously output a sequence of symbols
  - This is just a simple concatenation of many copies of the simple "output at the end of the input sequence" model we just saw

- But this simple extension complicates matters..

The *sequence-to-sequence* problem
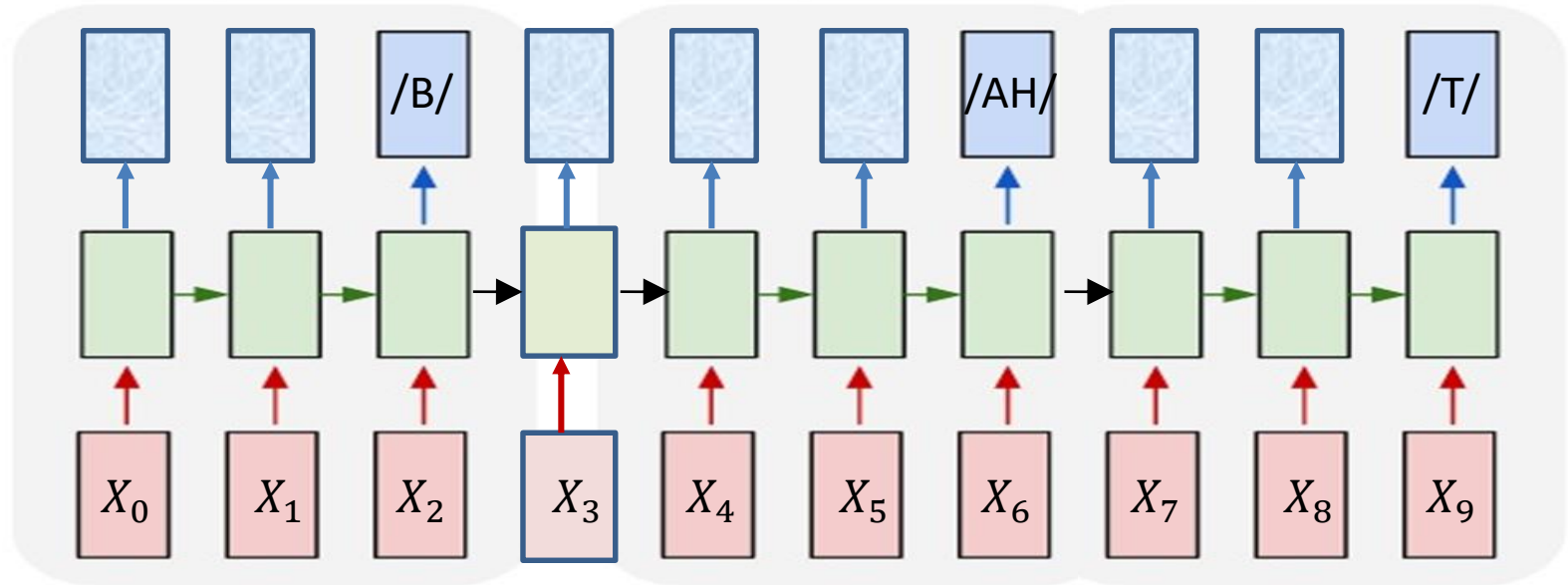
- How do we know *when* to output symbols
  - In fact, the network produces outputs at *every* time
  - *Which* of these are the *real* outputs

The actual output of the network

/AH/	y_0^1	y_1^1	y_2^1	y_3^1	y_4^1	y_5^1	y_6^1	y_7^1	y_8^1
/B/	y_0^2	y_1^2	y_2^2	y_3^2	y_4^2	y_5^2	y_6^2	y_7^2	y_8^2
/D/	y_0^3	y_1^3	y_2^3	y_3^3	y_4^3	y_5^3	y_6^3	y_7^3	y_8^3
/EH/	y_0^4	y_1^4	y_2^4	y_3^4	y_4^4	y_5^4	y_6^4	y_7^4	y_8^4
/IY/	y_0^5	y_1^5	y_2^5	y_3^5	y_4^5	y_5^5	y_6^5	y_7^5	y_8^5
/F/	y_0^6	y_1^6	y_2^6	y_3^6	y_4^6	y_5^6	y_6^6	y_7^6	y_8^6
/G/	y_0^7	y_1^7	y_2^7	y_3^7	y_4^7	y_5^7	y_6^7	y_7^7	y_8^7

X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
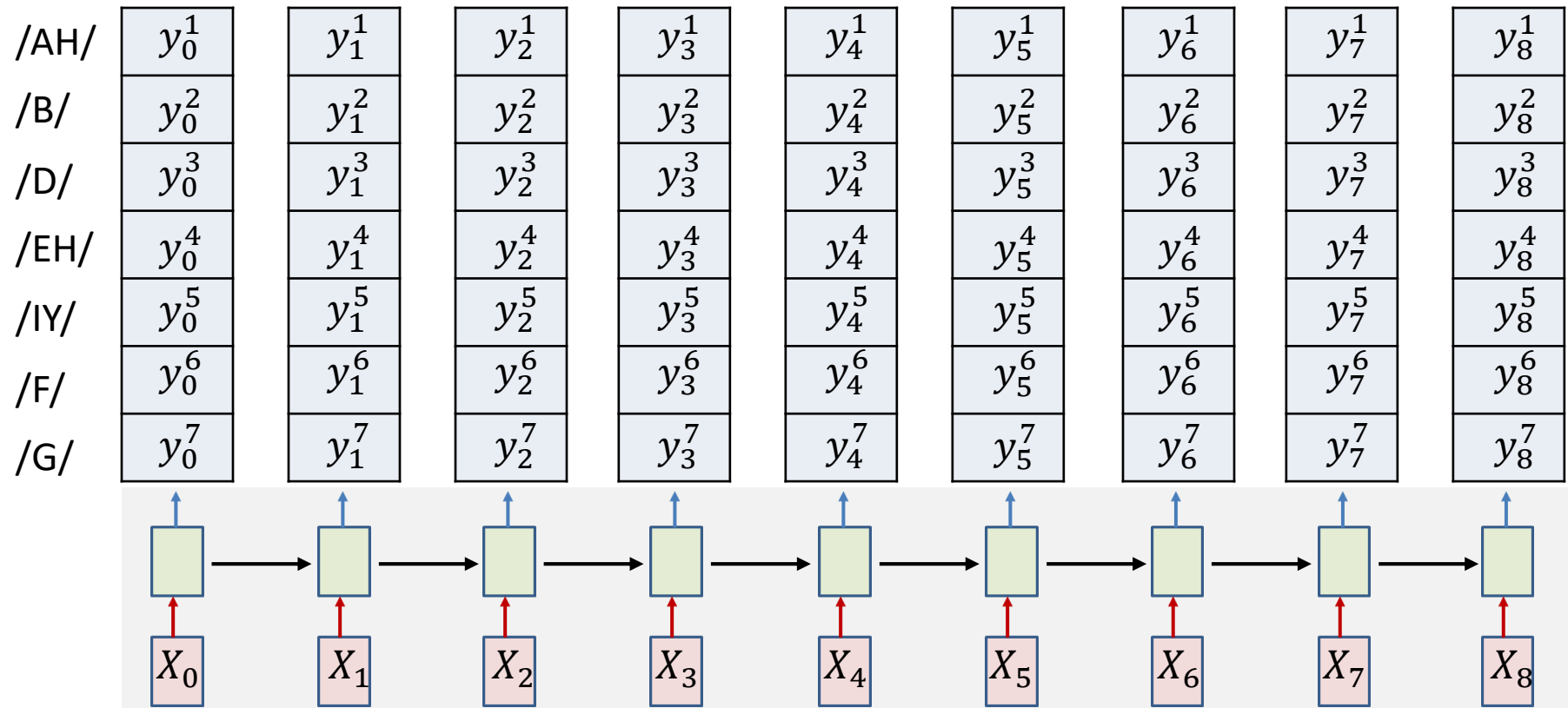
- At each time the network outputs a probability for *each* output symbol

The actual output of the network

	y_0^1	y_1^1	y_2^1	y_3^1	y_4^1	y_5^1	y_6^1	y_7^1	y_8^1
/AH/	y_0^1	y_1^1	y_2^1	y_3^1	y_4^1	y_5^1	y_6^1	y_7^1	y_8^1
/B/	y_0^2	y_1^2	y_2^2	y_3^2	y_4^2	y_5^2	y_6^2	y_7^2	y_8^2
/D/	y_0^3	y_1^3	y_2^3	y_3^3	y_4^3	y_5^3	y_6^3	y_7^3	y_8^3
/EH/	y_0^4	y_1^4	y_2^4	y_3^4	y_4^4	y_5^4	y_6^4	y_7^4	y_8^4
/IY/	y_0^5	y_1^5	y_2^5	y_3^5	y_4^5	y_5^5	y_6^5	y_7^5	y_8^5
/F/	y_0^6	y_1^6	y_2^6	y_3^6	y_4^6	y_5^6	y_6^6	y_7^6	y_8^6
/G/	y_0^7	y_1^7	y_2^7	y_3^7	y_4^7	y_5^7	y_6^7	y_7^7	y_8^7
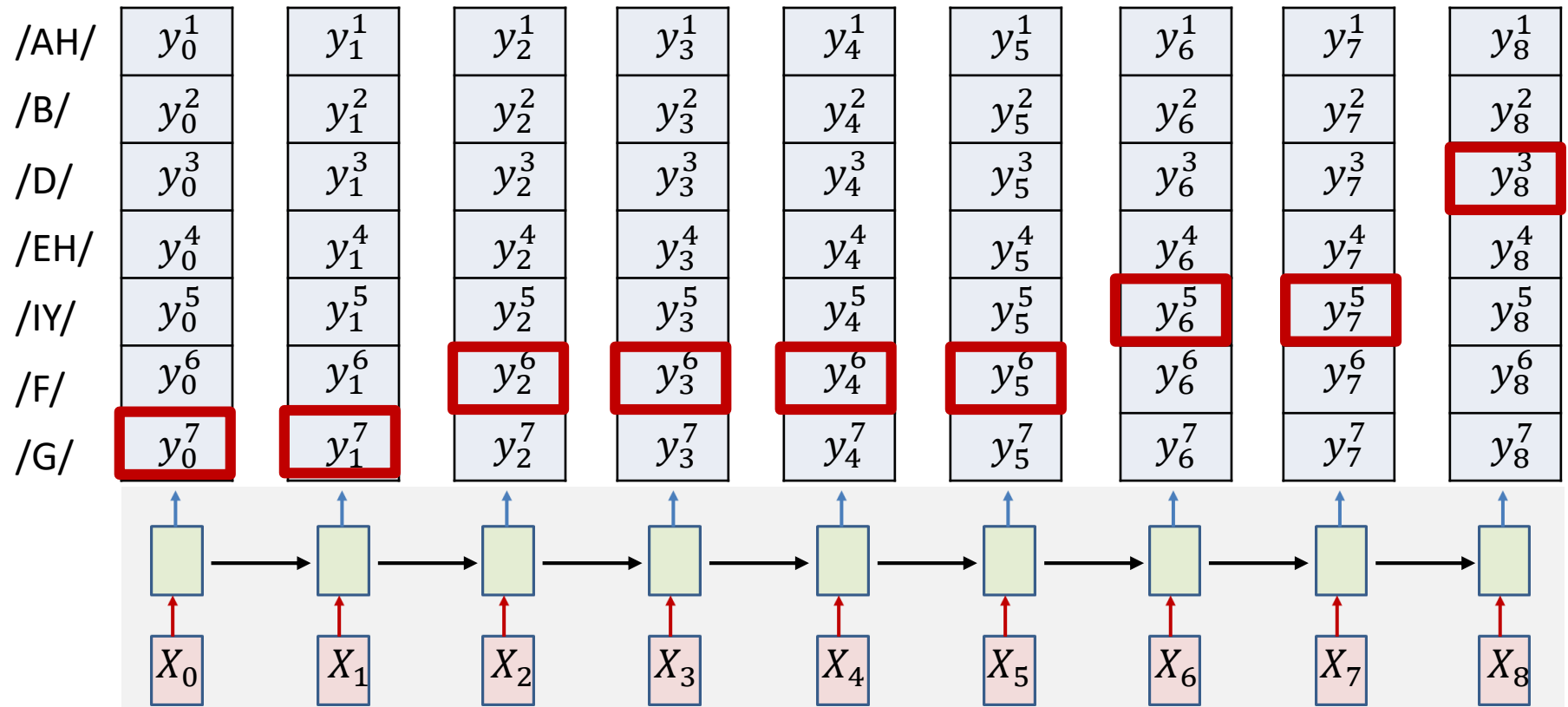
- Option 1: Simply select the most probable symbol at each time
  - *Merge* d
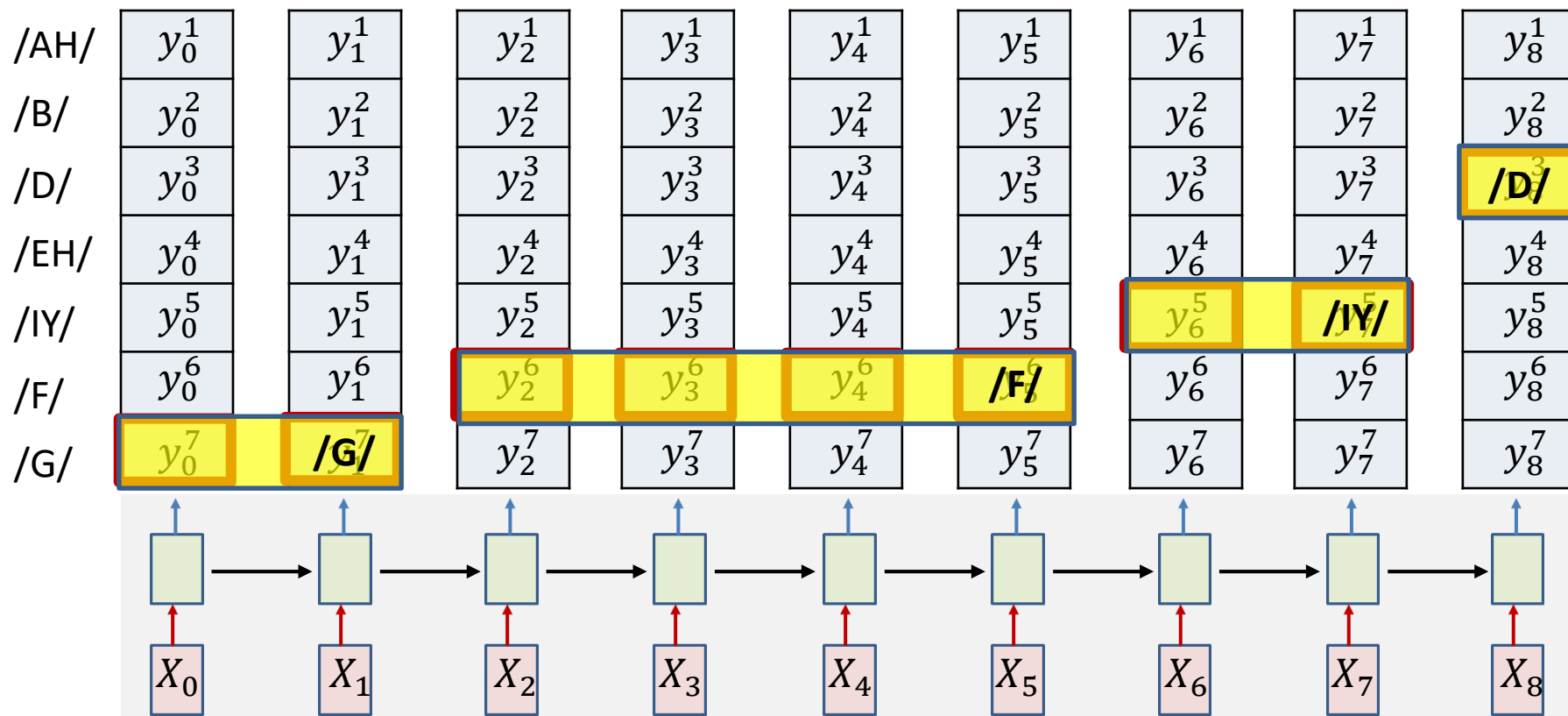
The actual output of the network

- Option 1: Simply select the most probable symbol at each time
  - *Merge* adjacent repeated symbols, and place the actual emission of the symbol in the final instant

The actual output of the network

/AH/	y_0^1	y_1^1	y_2^1	y_3^1	y_4^1	y_5^1	y_6^1	y_7^1	y_8^1
/B/	y_0^2	y_1^2	y_2^2	y_3^2	y_4^2	y_5^2	y_6^2	y_7^2	y_8^2
/D/	y_0^3	y_1^3	y_2^3	y_3^3	y_4^3	y_5^3	y_6^3	y_7^3	**/D/**
/EH/							y_6^4	y_7^4	y_8^4
/IY/							y_6^5	**/IY/**	y_8^5
/F/	y_0^6	y_1^6	y_2^6	**/F/**	y_4^6	**/F/**	y_6^6	y_7^6	y_8^6
/G/	y_0^7	**/G/**	y_2^7	y_3^7	y_4^7	y_5^7	y_6^7	y_7^7	y_8^7

Cannot distinguish between an extended symbol and repetitions of the symbol

- Option 1: Simply select the most probable symbol at each time
  - *Merge* adjacent repeated symbols, and place the actual emission of the symbol in the final instant

The actual output of the network

Resulting sequence may be meaningless (what word is "GFIYD"?)

Cannot distinguish between an extended symbol and repetitions of the symbol
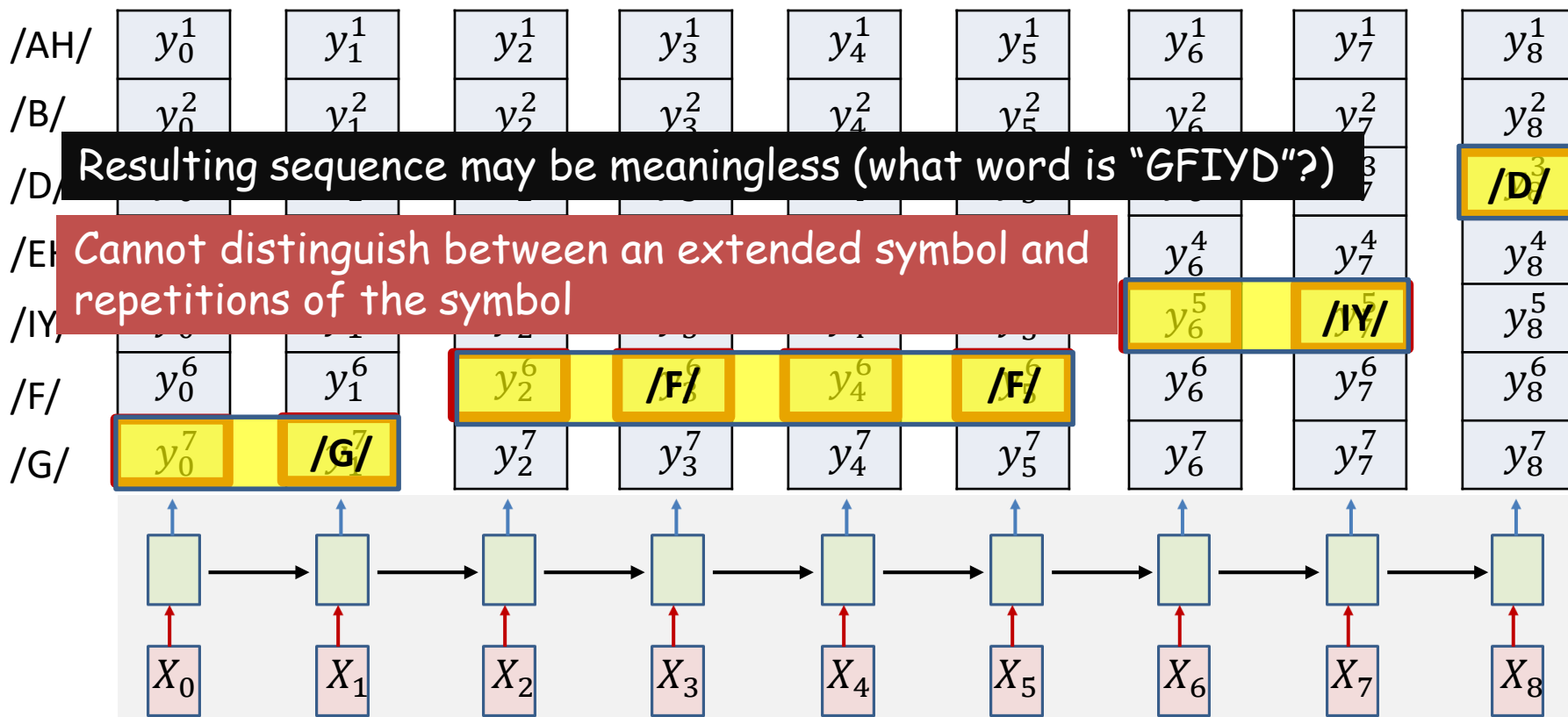
- Option 1: Simply select the most probable symbol at each time
  - *Merge* adjacent repeated symbols, and place the actual emission of the symbol in the final instant

The actual output of the network

- Option 2: Impose external constraints on what sequences are allowed
  - *E.g.* only allow sequences corresponding to dictionary words
  - *E.g. Sub-symbol* units (like in HW1 – what were they?)

The *sequence-to-sequence* problem
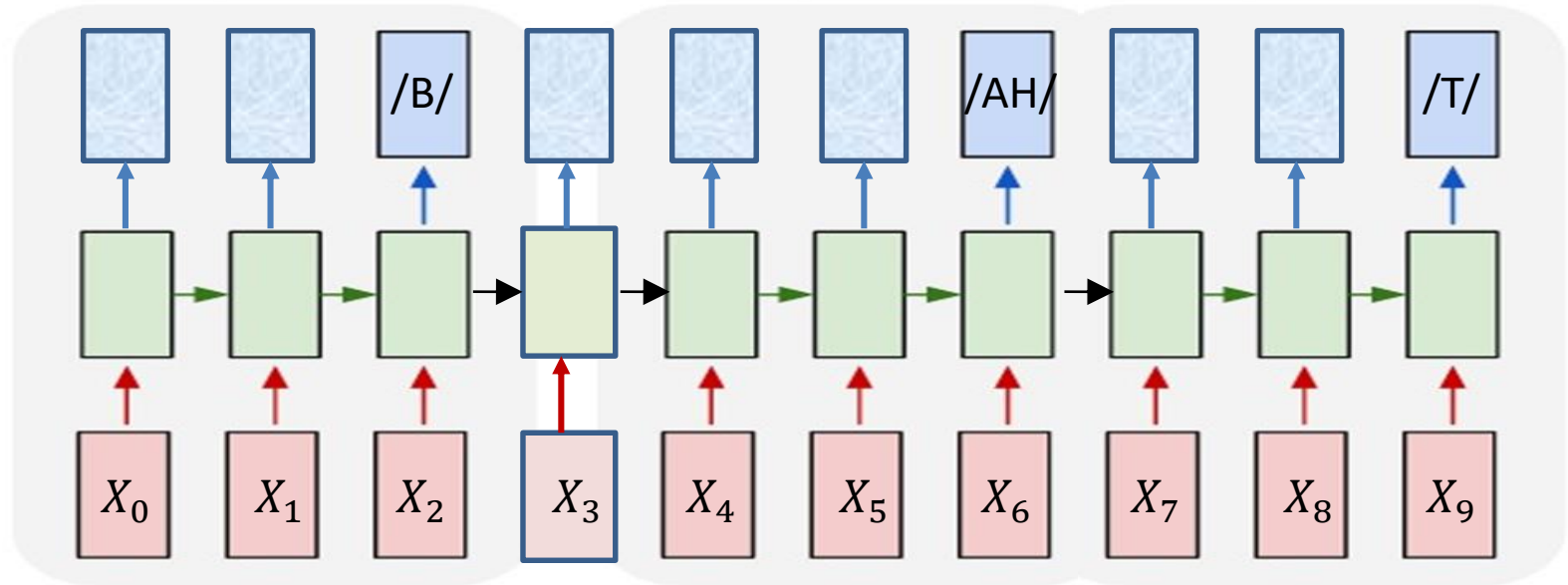
- How do we know *when* to output symbols
  - In fact, the net~~~~oduces outputs at *every* time

    Addressed
  - *Which* of these are the *real* outputs

- How do we *train* these models?

Training
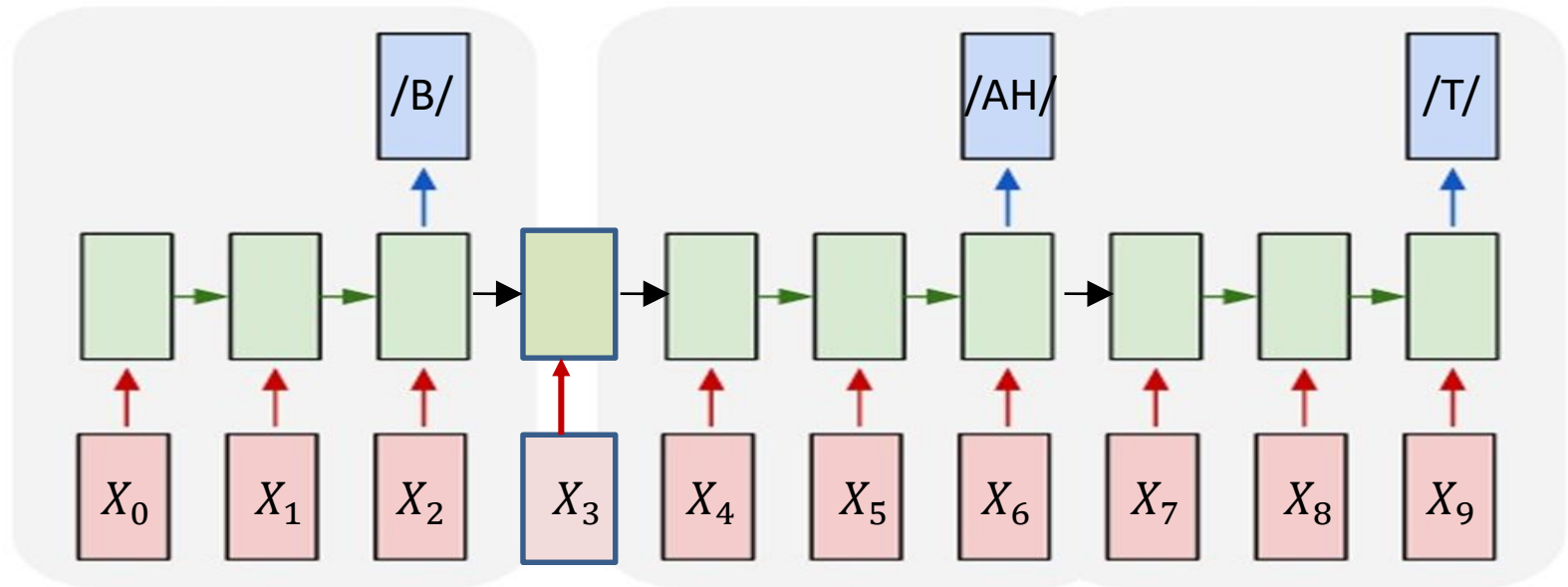
- Given output symbols *at the right locations*
  - The phoneme /B/ ends at X_2, /AH/ at X_6, /T/ at X_9
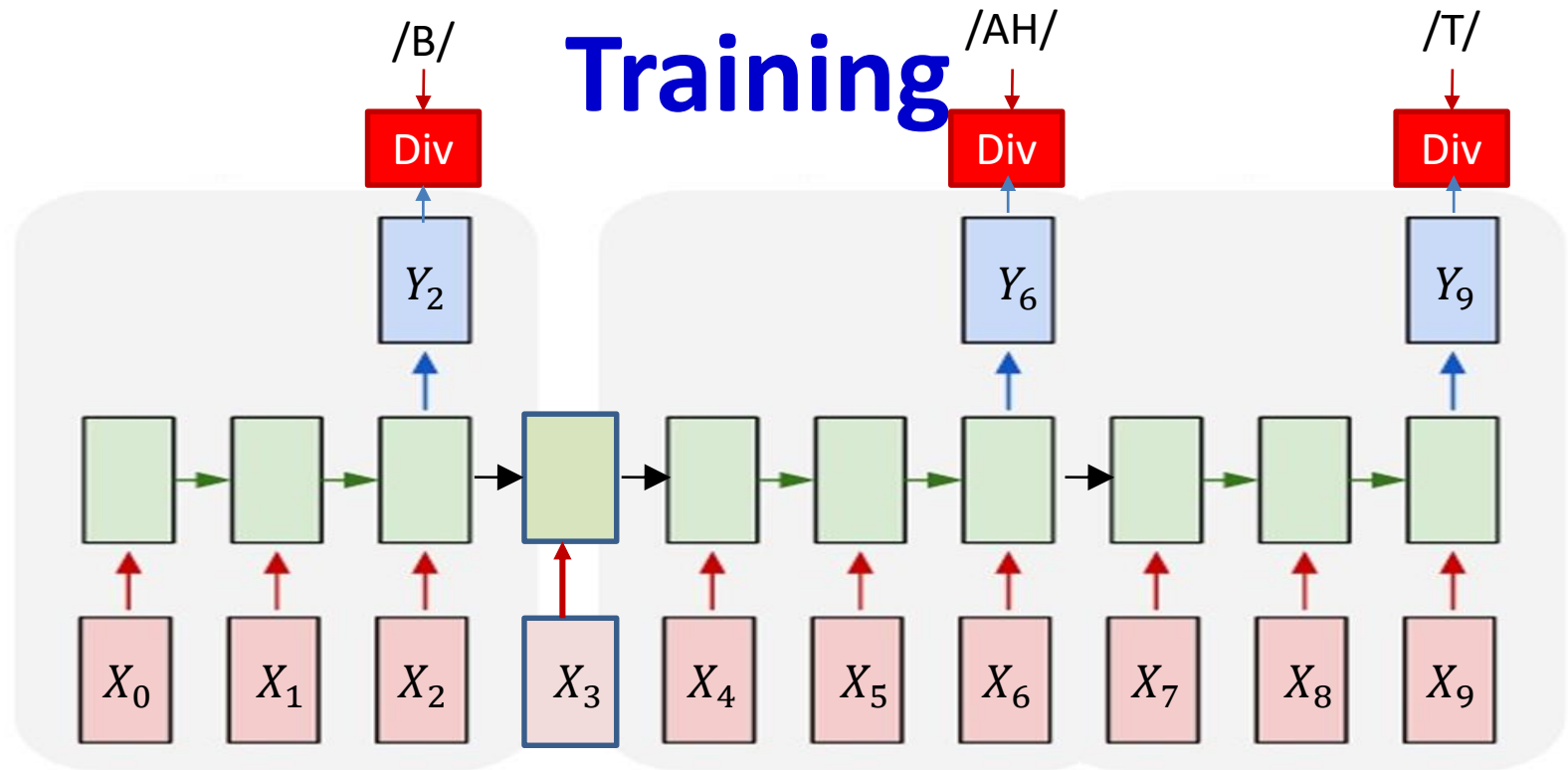
- Either just define Divergence as:
  $$DIV = Xent(Y_2, B) + Xent(Y_6, AH) + Xent(Y_9, T)$$
- Or..
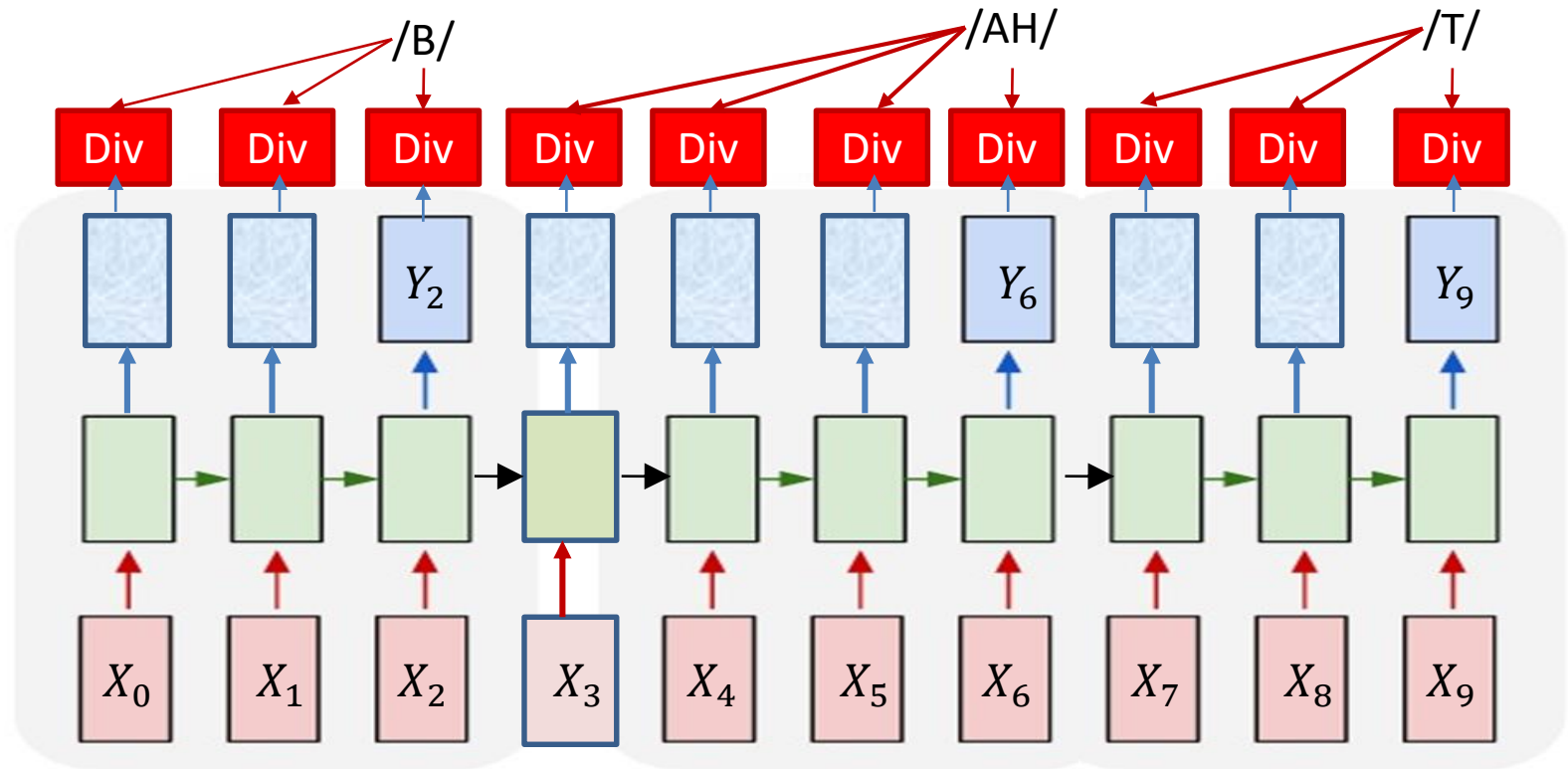
- Either just define Divergence as:
$$DIV = Xent(Y_2, B) + Xent(Y_6, AH) + Xent(Y_9, T)$$

- Or *repeat the symbols over their duration*

$$DIV = \sum_t Xent(Y_t, symbol_t) = -\sum_t \log Y(t, symbol_t)$$

Problem: No timing information provided

/B/   /AH/ /T/
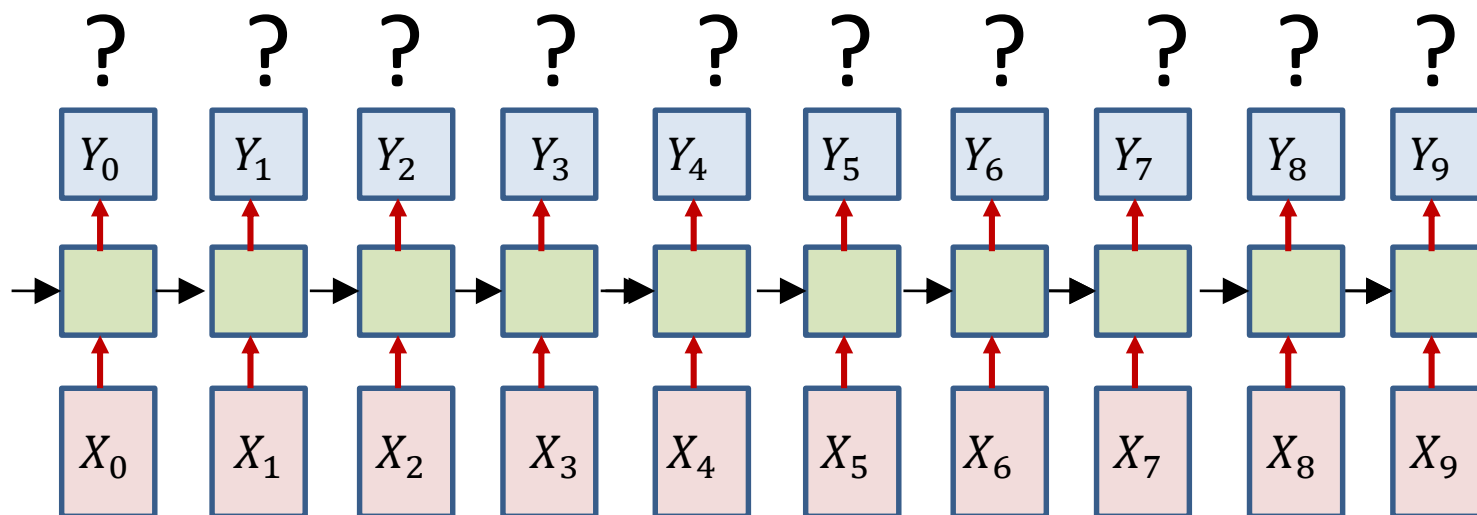
- Only the sequence of output symbols is provided for the training data
  - But no indication of which one occurs where
- How do we compute the divergence?
  - And how do we compute its gradient w.r.t. Y_t

Not given alignment

/B/	/AH/	/AH/	/AH/	/AH/	/AH/	/AH/	/AH/	/AH/	/T/
Div	Div	Div	Div	Div	Div	Div	Div	Div	Div
Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9
X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
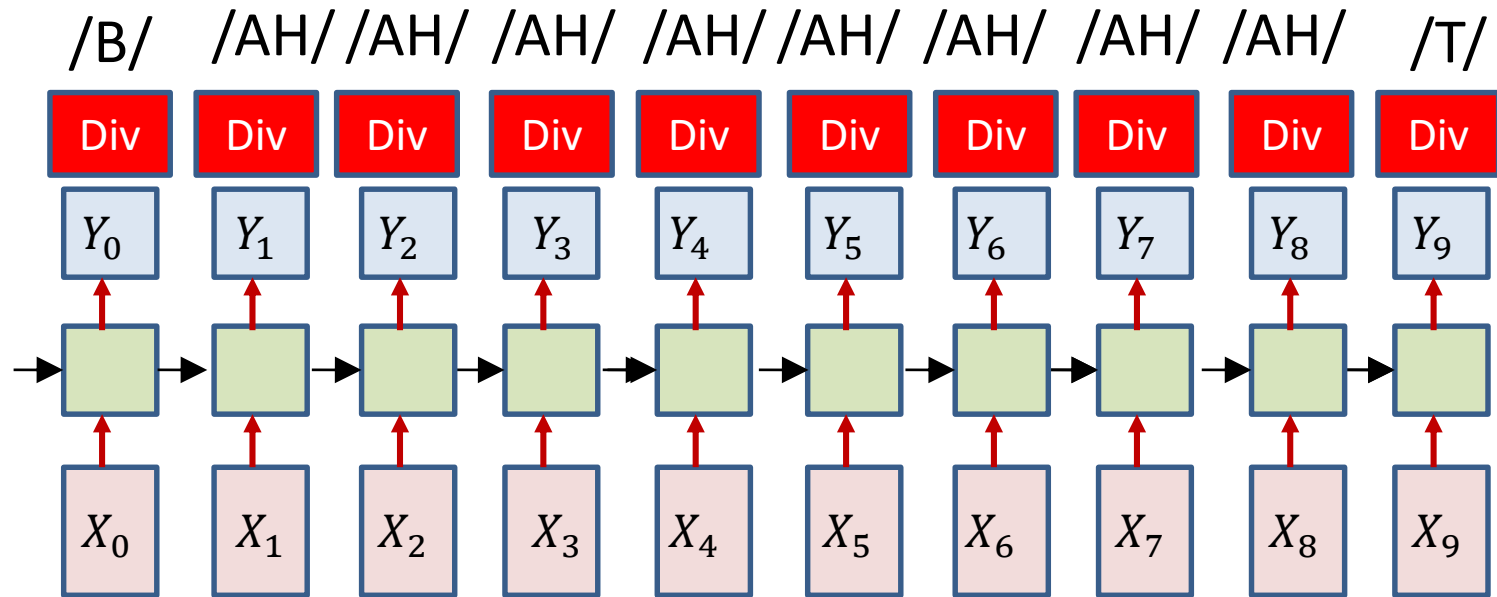
- There are many possible alignments
- Corresponding each there is a divergence

$$Div(B, AH \ldots, AH, T)$$

$$= -\sum_t \log Y(t, [B, AH, \ldots, AH, T]_t)$$

82

Not given alignment

/B/ /B/ /AH/ /AH/ /AH/ /AH/ /AH/ /AH/ /AH/ /T/

| Div | Div | Div | Div | Div | Div | Div | Div | Div | Div |

Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7 Y_8 Y_9

X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8 X_9
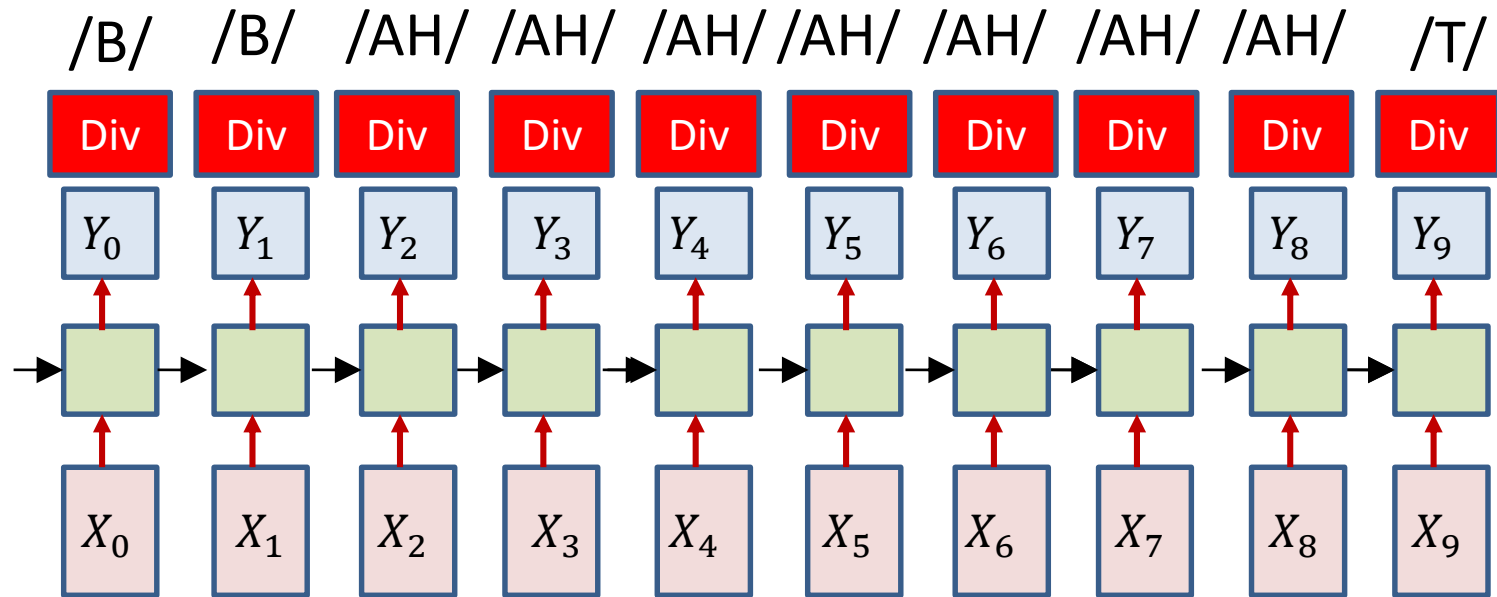
- There are many possible alignments
- Corresponding each there is a divergence

$$Div(B, B \ldots, AH, T)$$

$$= -\sum_t \log Y(t, [B, B, \ldots, AH, T]_t)$$

Not given alignment
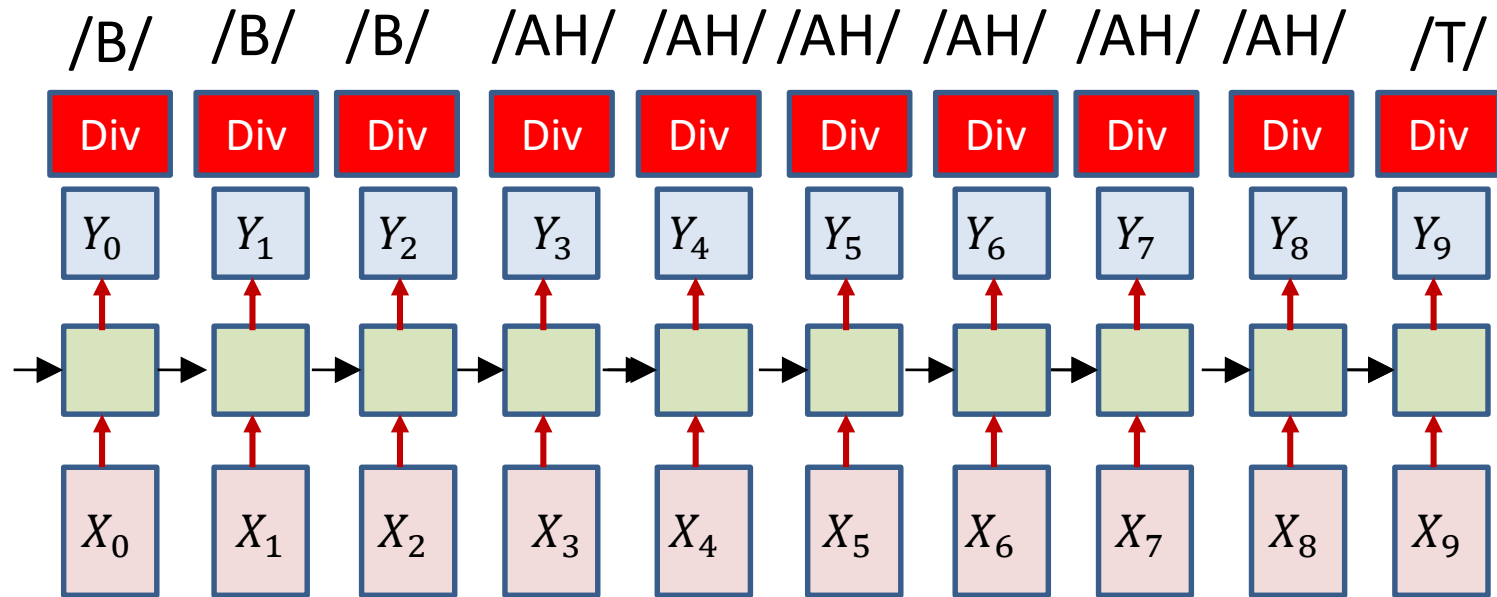
- There are many possible alignments
- Corresponding each there is a divergence

$$Div(B, B, B \dots, AH, T)$$

$$= - \sum_t \log Y(t, [B, B, B, \dots, AH, T]_t)$$

Not given alignment

/B/ /AH/ /AH/ /AH/ /AH/ /AH/ /AH/ /AH/ /T/ /T/

Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7 Y_8 Y_9

X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8 X_9
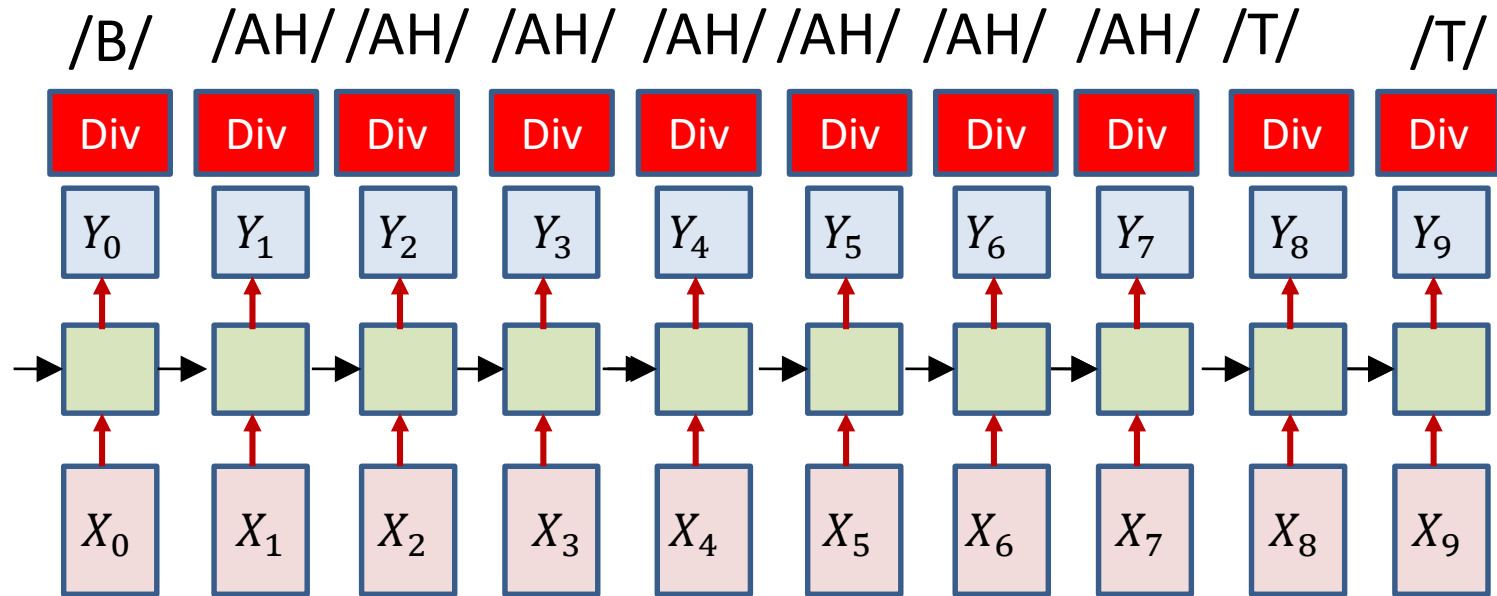
- There are many possible alignments
- Corresponding each there is a divergence

$$Div(B, AH \ldots, T, T)$$

$$= -\sum_t \log Y(t, [B, AH, \ldots, T, T]_t)$$

85

In the absence of alignment

- Any of these alignments is possible
- Simply consider *all* of these alignments
- The total Divergence is the weighted sum of the Divergences for the individual alignments
- $DIV =$
$w(B, AH \dots, AH, T)Div(B, AH \dots, AH, T) +$
$w(B, B \dots, AH, T)Div(B, B \dots, AH, T) +$
$w(B, AH \dots, T, T)Div(B, AH \dots, T, T) \dots$

In the absence of alignment

- Any of these alignments is possible
- Simply consider *all* of these alignments
- The total Divergence is the weighted sum of the Divergences for the individual alignments

$$DIV = \sum_{s_1 \dots s_N} w(s_1 \dots s_N) Div(s_1 \dots s_N)$$

  – Where $s_1 \dots s_N$ is the sequence of target symbols corresponding to an alignment of the target (e.g. phoneme) sequence $P_1 \dots P_K$
    - Note: $K \leq N$

In the absence of alignment

- Any of these alignments is possible
- Simply consider *all* of these alignments
- The total Divergence is the weighted sum of the Divergences for the individual alignments

$$DIV = \sum_{s_1 \ldots s_N} w(s_1 \ldots s_N) Div(s_1 \ldots s_N)$$

Give less weight to less *likely* alignments

In the absence of alignment

- Any of these alignments is possible
- Simply consider *all* of these alignments
- The total Divergence is the weighted sum of the Divergences for the individual alignments

$$DIV = \sum_{s_1 \ldots s_N} P(s_1 \ldots s_N | X, P_1 \ldots P_K) Div(s_1 \ldots s_N)$$

Weigh by the probability of the alignment

This is the *expected* divergence across all alignments

In the absence of alignment

- Any of these alignments is possible

- Simply consider *all* of these alignments

- The total Divergence is the weighted sum of the Divergences for the individual alignments

$$DIV = -\sum_{s_1 \ldots s_N} P(s_1 \ldots s_N | X, P_1 \ldots P_K) \sum_t \log Y(t, s_t)$$

Must compute this