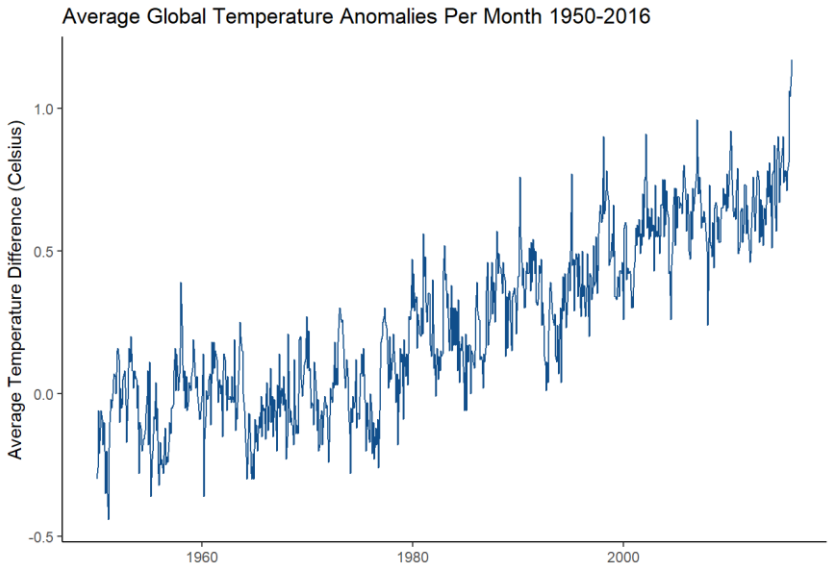
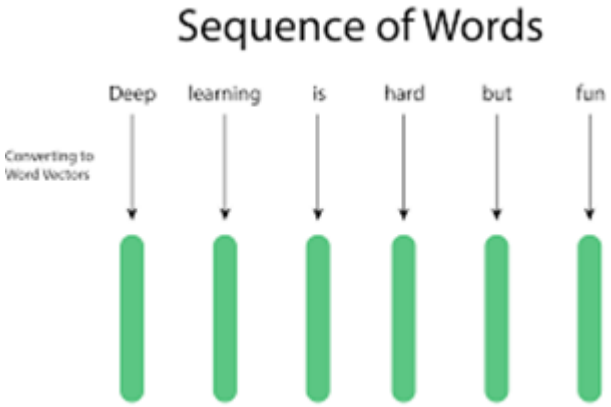
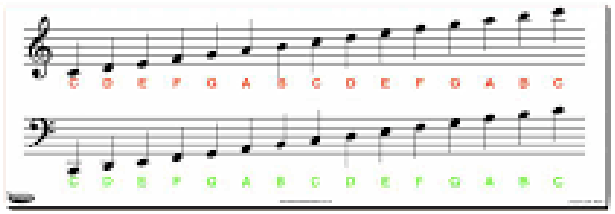
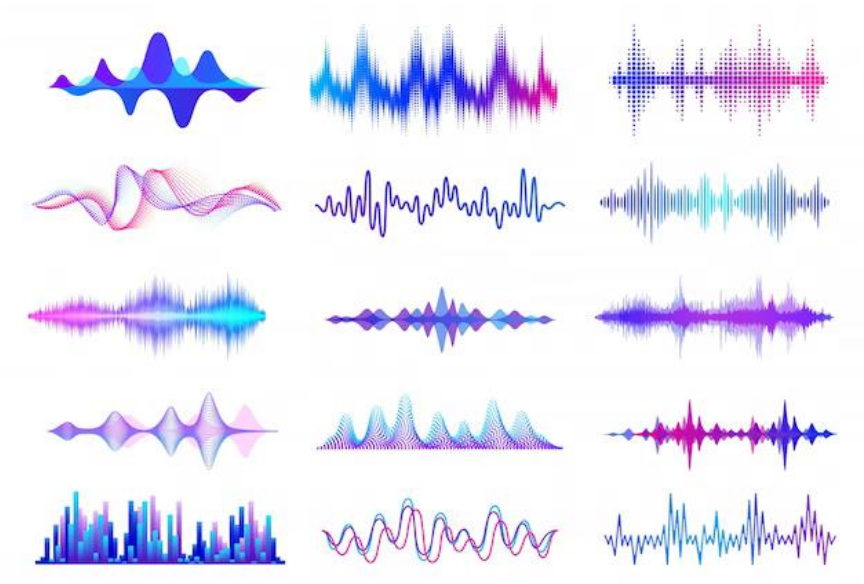
Recurrent Neural Networks (RNN)

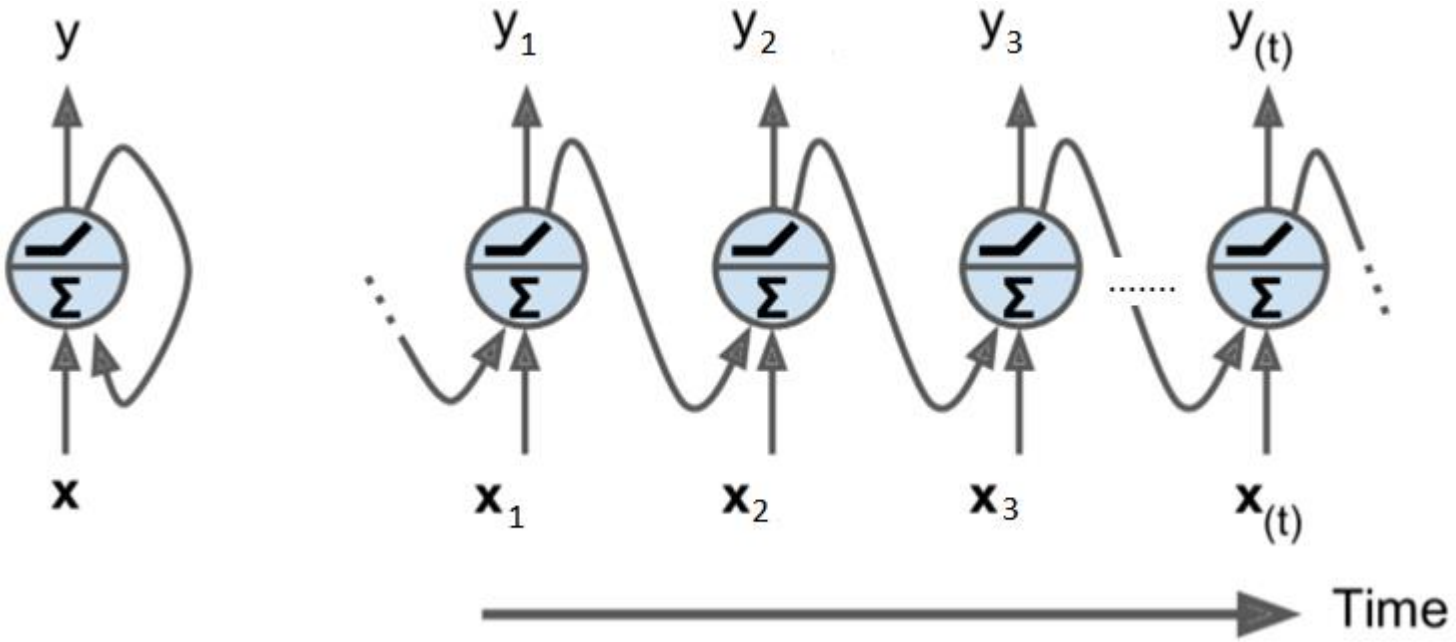
Sequential data

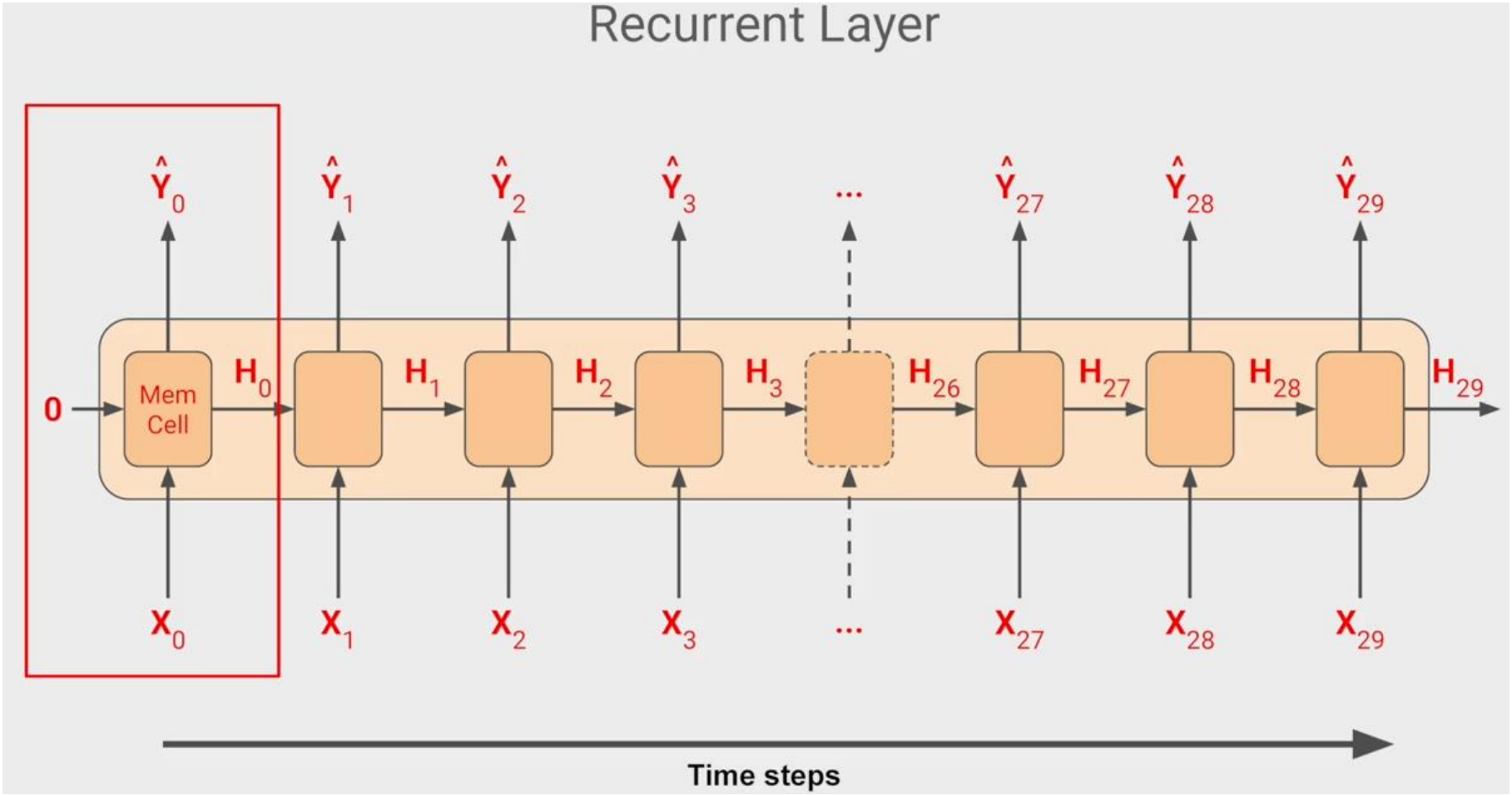









y
x
y1
y2
y3
y(t)
x1
x2
x3
x(t)
Σ
Time



Input windows **X** shape=[batch size, # time steps, # dims]

https://www.coursera.org/learn/tensorflow-sequences-time-series-and-prediction/home/info

Equation 14-2. Outputs of a layer of recurrent neurons for all instances in a mini-batch

$$\mathbf{Y}_{(t)} = \phi\left(\mathbf{X}_{(t)} \cdot \mathbf{W}_x + \mathbf{Y}_{(t-1)} \cdot \mathbf{W}_y + \mathbf{b}\right)$$

$$= \phi\left(\begin{bmatrix}\mathbf{X}_{(t)} & \mathbf{Y}_{(t-1)}\end{bmatrix} \cdot \mathbf{W} + \mathbf{b}\right) \text{ with } \mathbf{W} = \begin{bmatrix}\mathbf{W}_x \\ \mathbf{W}_y\end{bmatrix}$$

- $\mathbf{Y}_{(t)}$ is an $m \times n_{\text{neurons}}$ matrix containing the layer's outputs at time step t for each instance in the mini-batch (m is the number of instances in the mini-batch and n_{neurons} is the number of neurons).
- $\mathbf{X}_{(t)}$ is an $m \times n_{\text{inputs}}$ matrix containing the inputs for all instances (n_{inputs} is the number of input features).
- $\mathbf{W}_x$ is an $n_{\text{inputs}} \times n_{\text{neurons}}$ matrix containing the connection weights for the inputs of the current time step.
- $\mathbf{W}_y$ is an $n_{\text{neurons}} \times n_{\text{neurons}}$ matrix containing the connection weights for the outputs of the previous time step.
- $\mathbf{b}$ is a vector of size n_{neurons} containing each neuron's bias term.
- The weight matrices $\mathbf{W}_x$ and $\mathbf{W}_y$ are often concatenated vertically into a single weight matrix $\mathbf{W}$ of shape $(n_{\text{inputs}} + n_{\text{neurons}}) \times n_{\text{neurons}}$ (see the second line of Equation 14-2).

Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

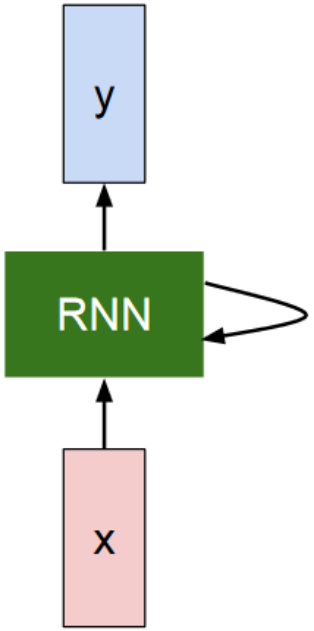
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

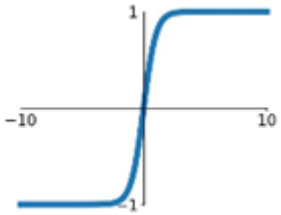
input vector at some time step

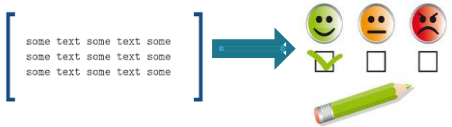
some function with parameters W

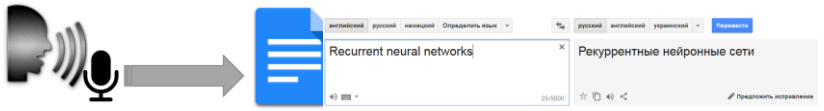


tanh

$\tanh(x)$



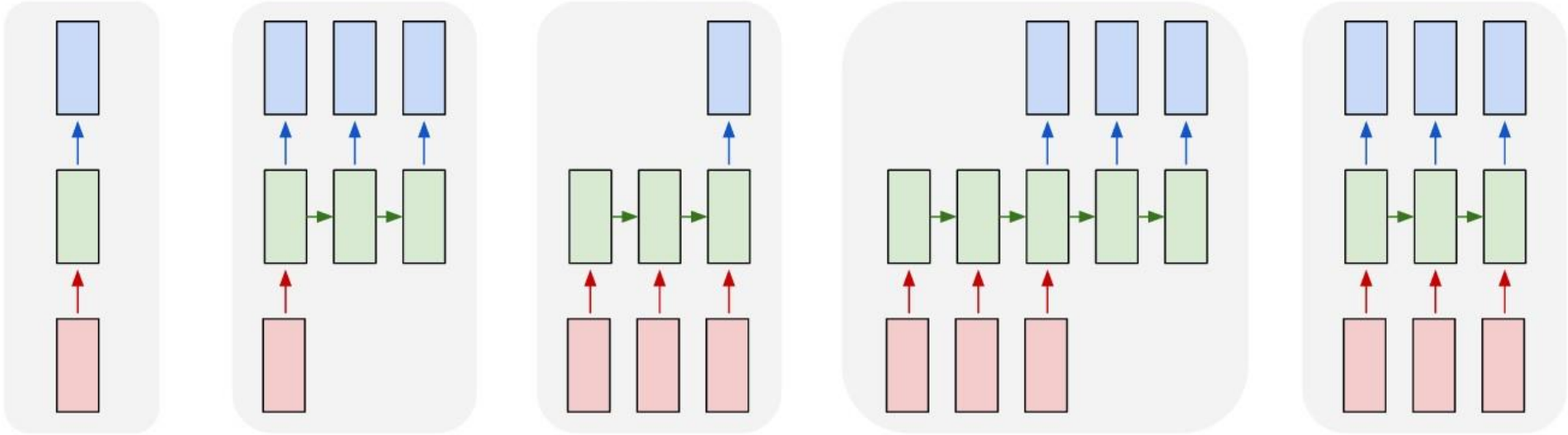




e.g. **Sentiment Classification**
sequence of words -> sentiment

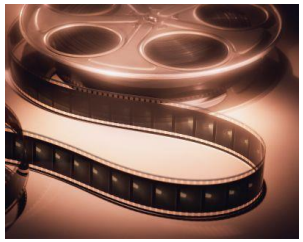
e.g. **Machine Translation**
seq of words -> seq of words

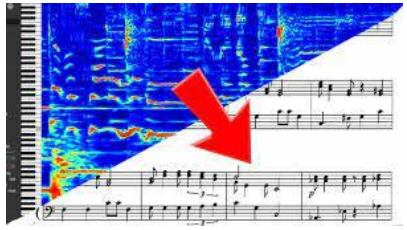
one to one | one to many | many to one | many to many | many to many

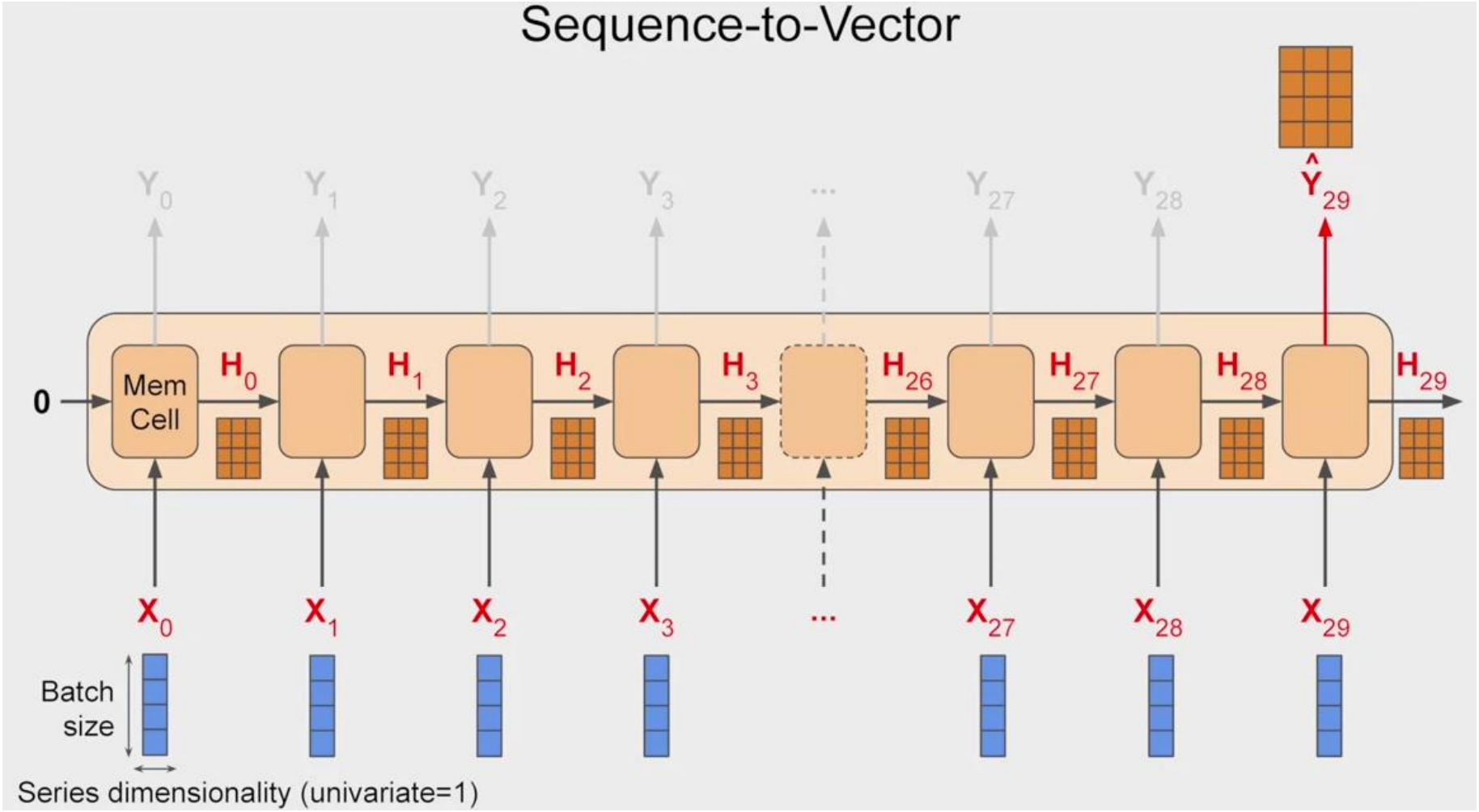


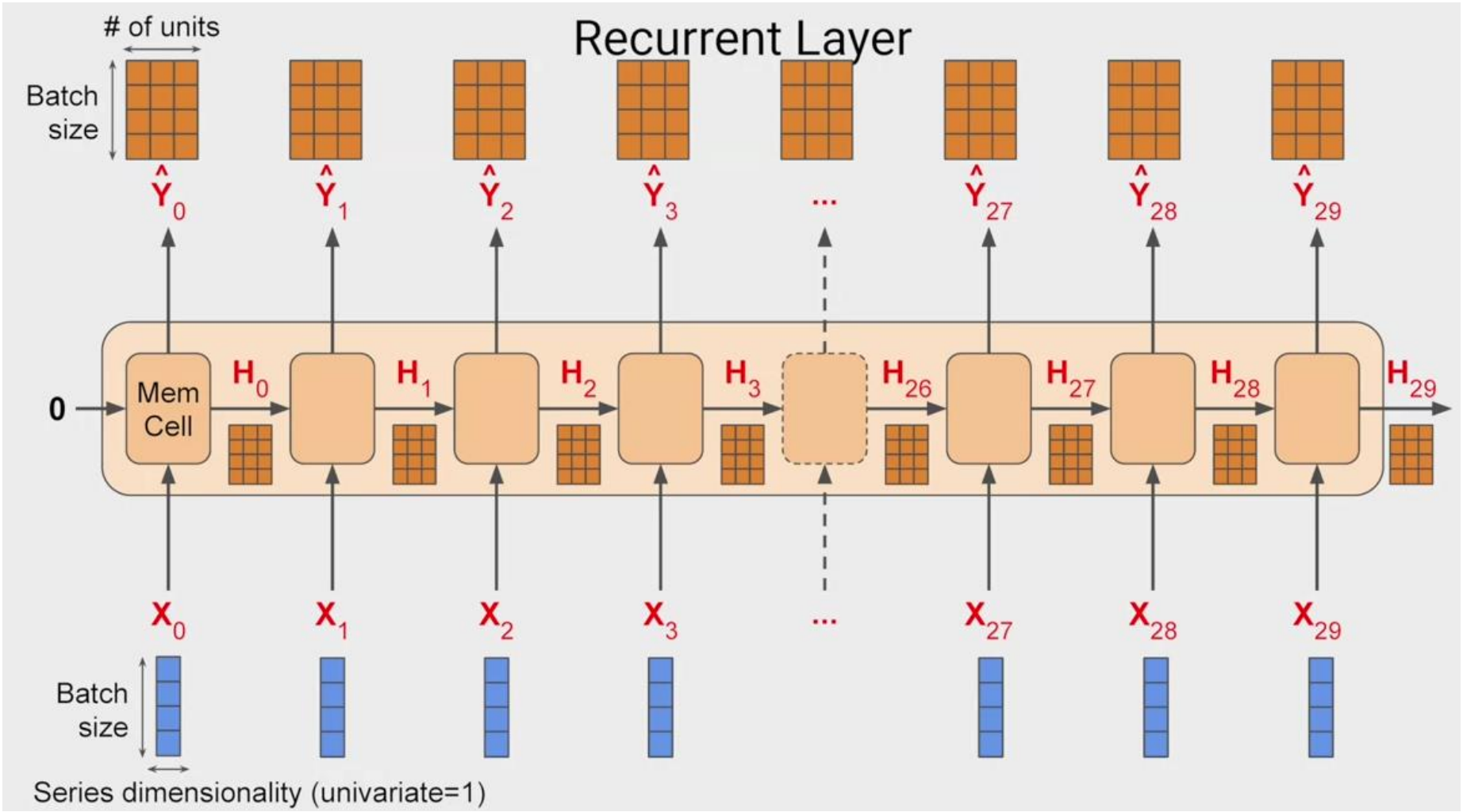
e.g. **Image Captioning**
image -> sequence of words

e.g. **Video classification on frame level**



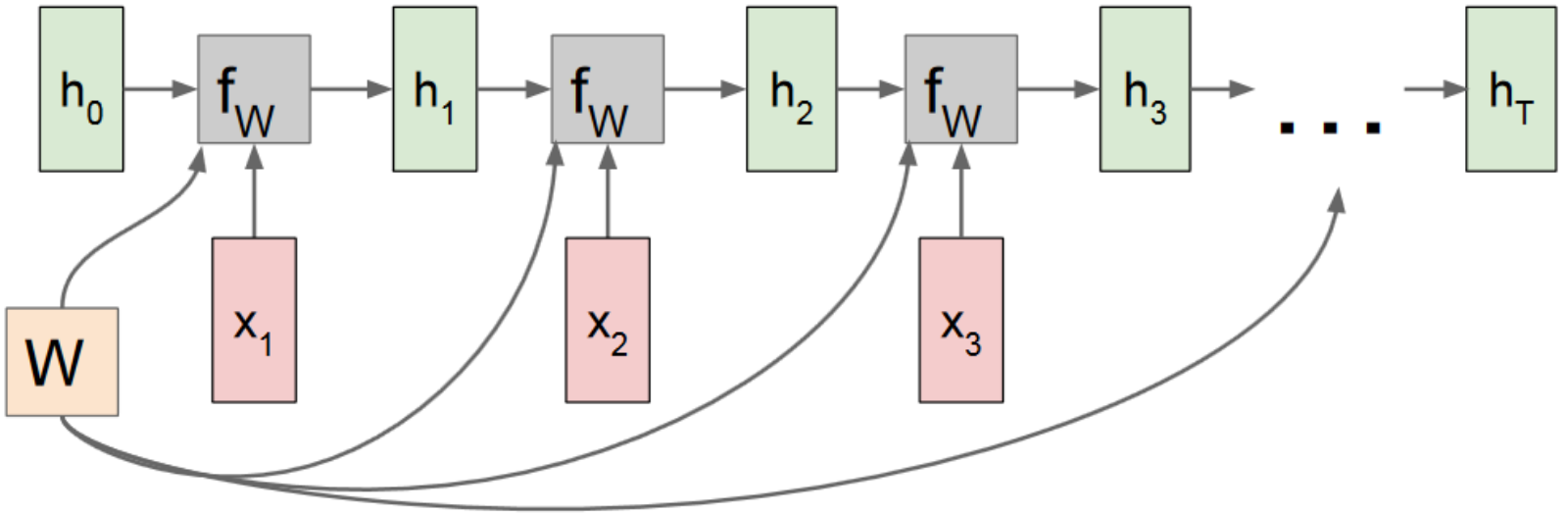



Sequence-to-Vector
Y0
Y1
Y2
Y3
...
Y27
Y28
Ŷ29
0
Mem Cell
H0
H1
H2
H3
H26
H27
H28
H29
X0
X1
X2
X3
...
X27
X28
X29
Batch size
Series dimensionality (univariate=1)



https://www.coursera.org/learn/tensorflow-sequences-time-series-and-prediction/home/info

Notice: the same function and the same set of parameters are used at every time step.



Reminder: Backpropagation with weight constraints

- It is easy to modify the backprop algorithm to incorporate linear constraints between the weights.
- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.
  - So if the weights started off satisfying the constraints, they will continue to satisfy them.

$$To\ constrain: \quad w_1 = w_2$$

$$we\ need: \quad \Delta w_1 = \Delta w_2$$

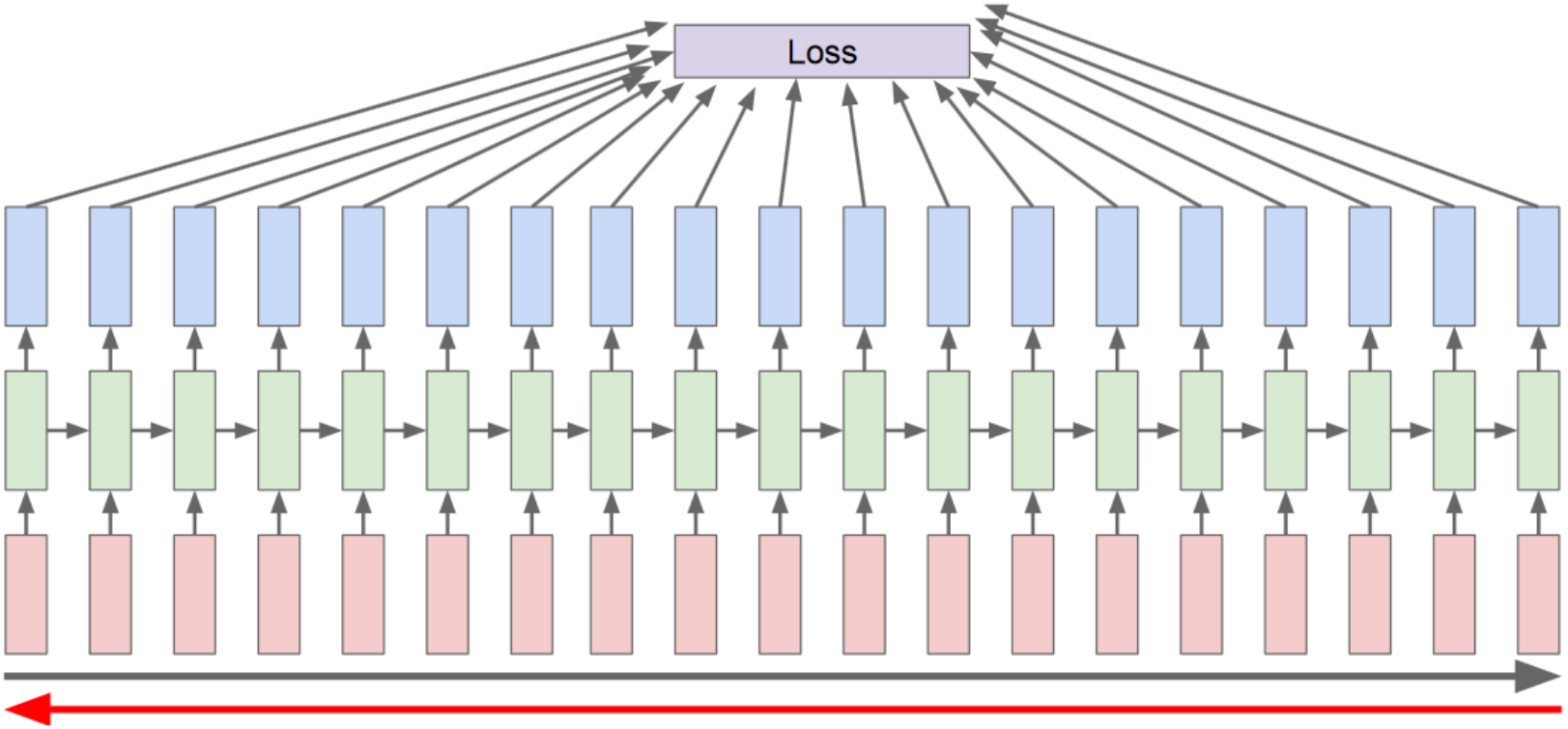
$$compute: \quad \frac{\partial E}{\partial w_1} \quad and \quad \frac{\partial E}{\partial w_2}$$

$$use \quad \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \quad for\ w_1\ and\ w_2$$

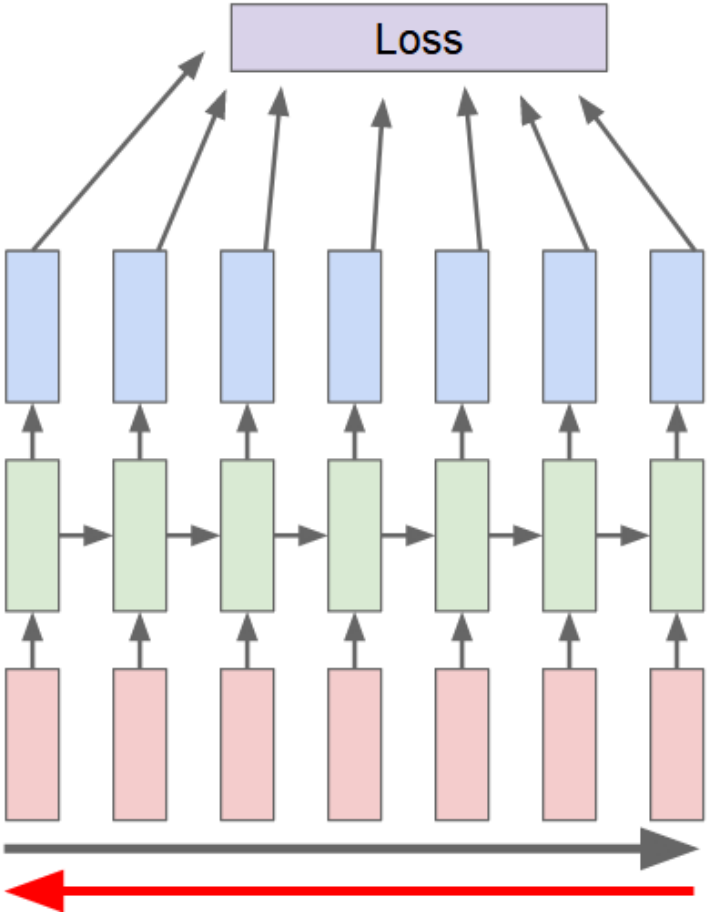
https://www.coursera.org/learn/neural-networks/home/week/7

Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

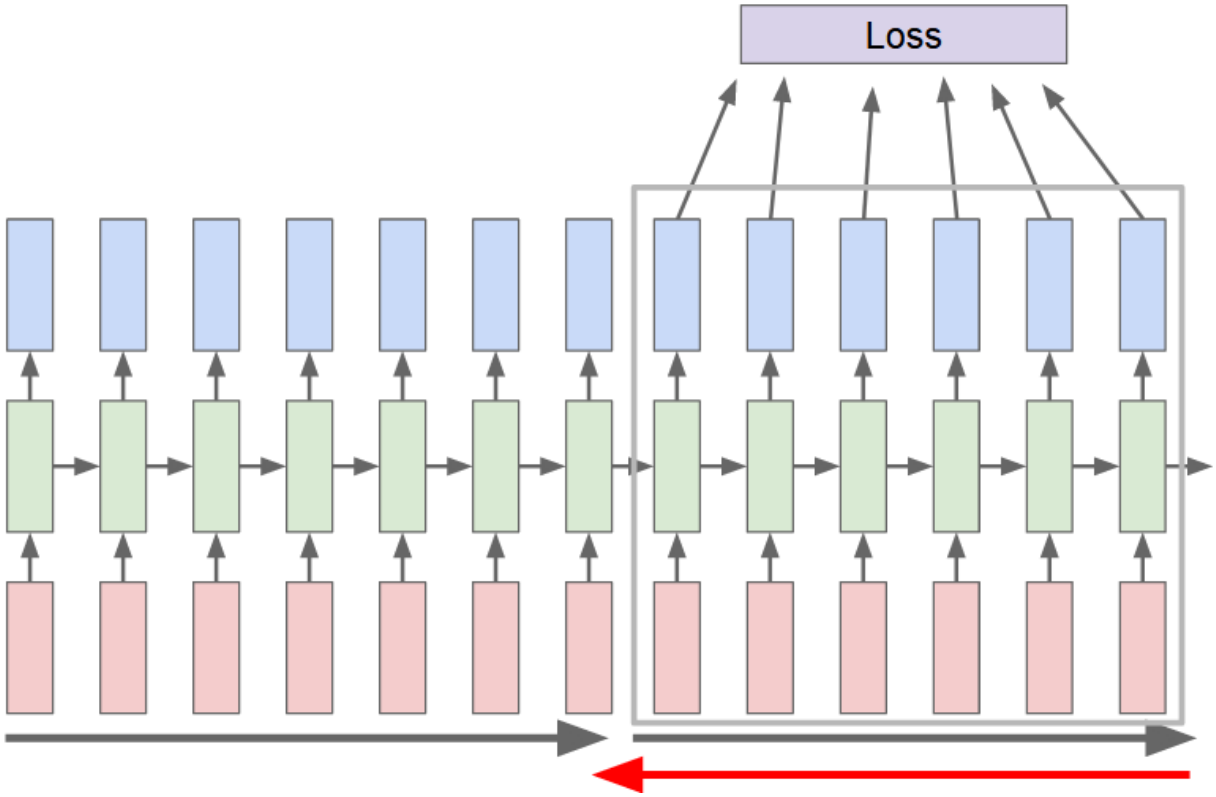


Truncated Backpropagation through time



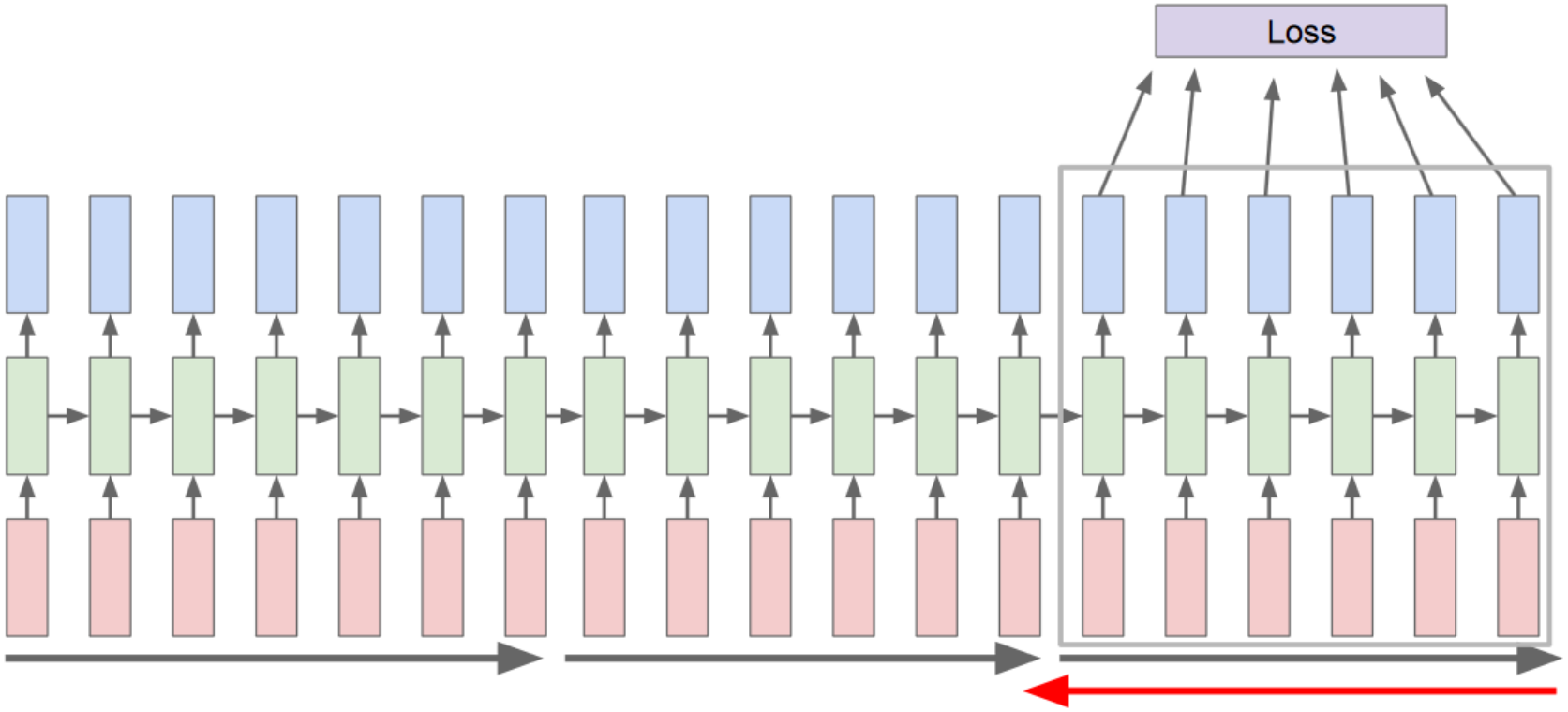
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

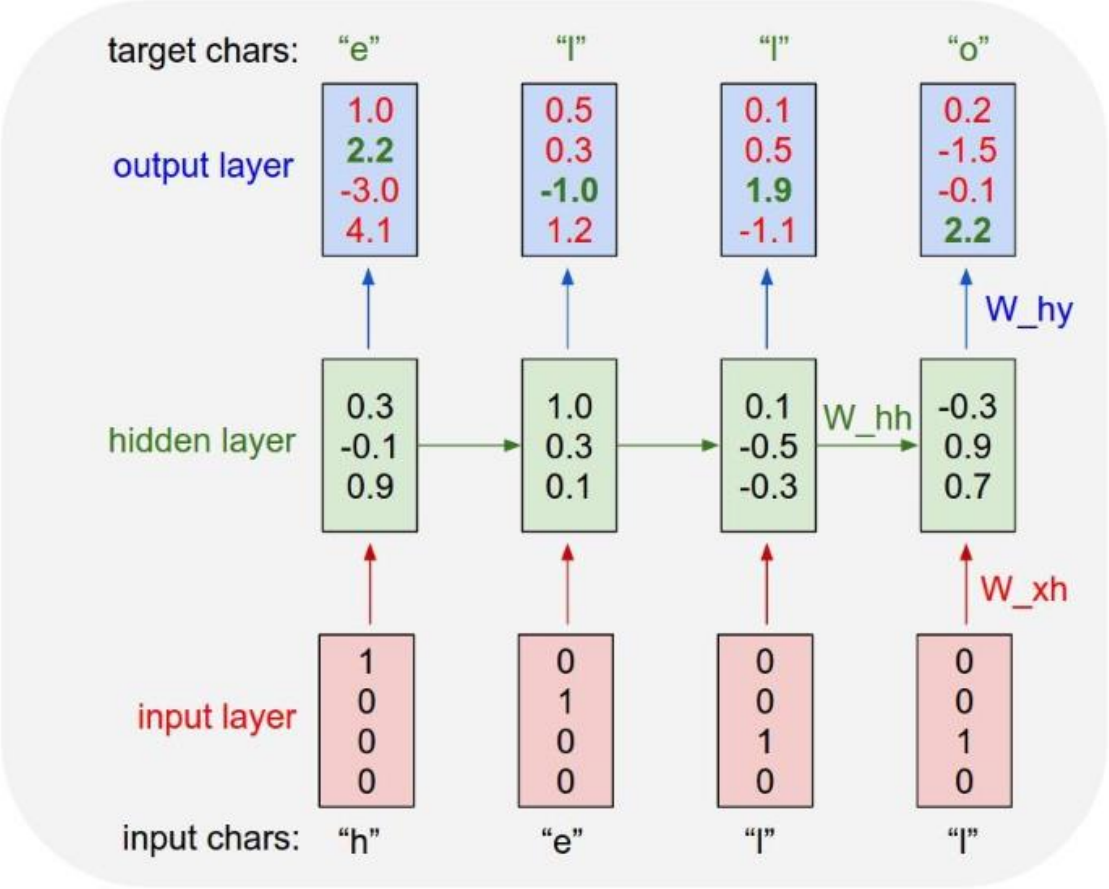
Truncated Backpropagation through time



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

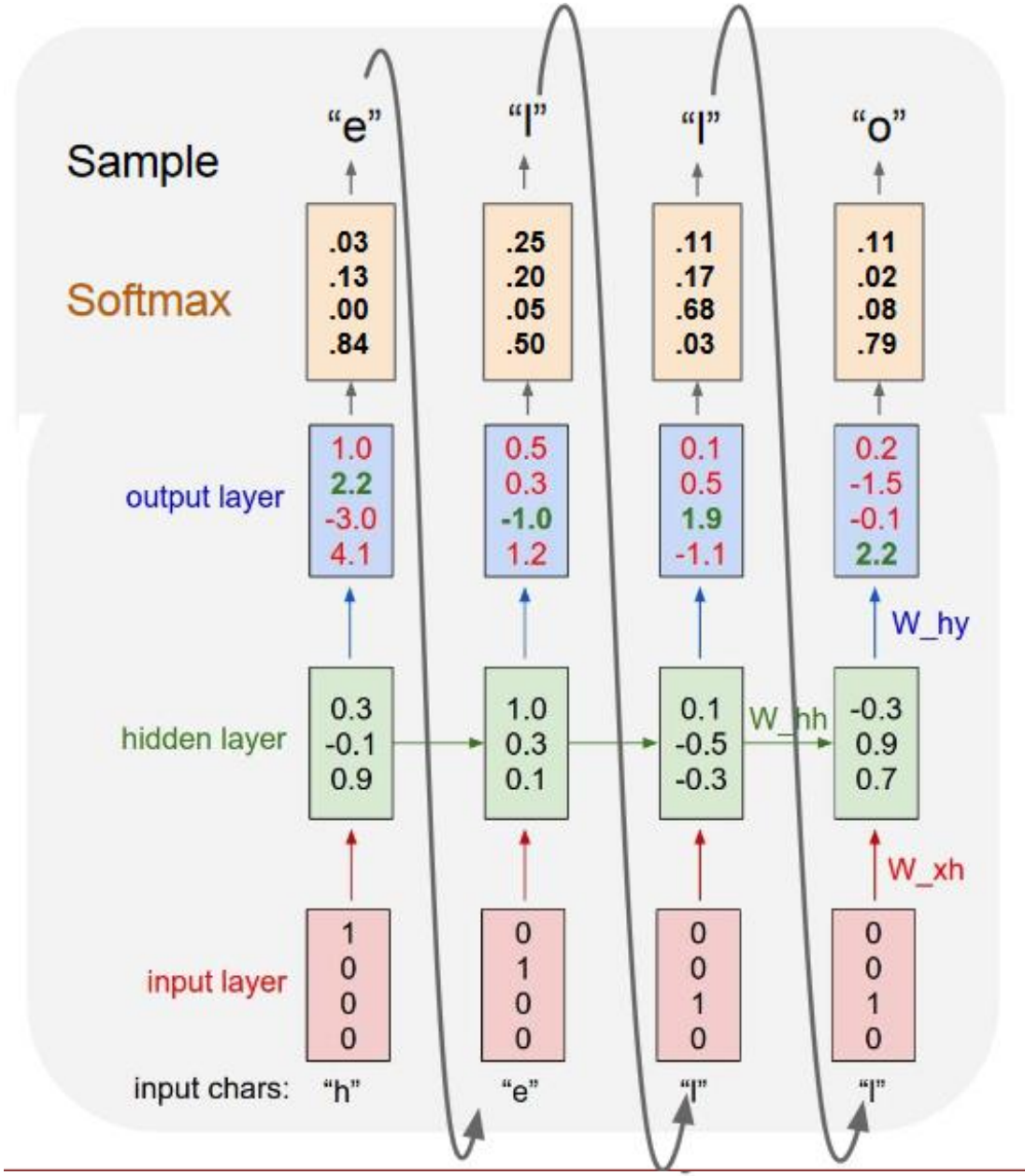


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Example: Character-level Language Model Sampling

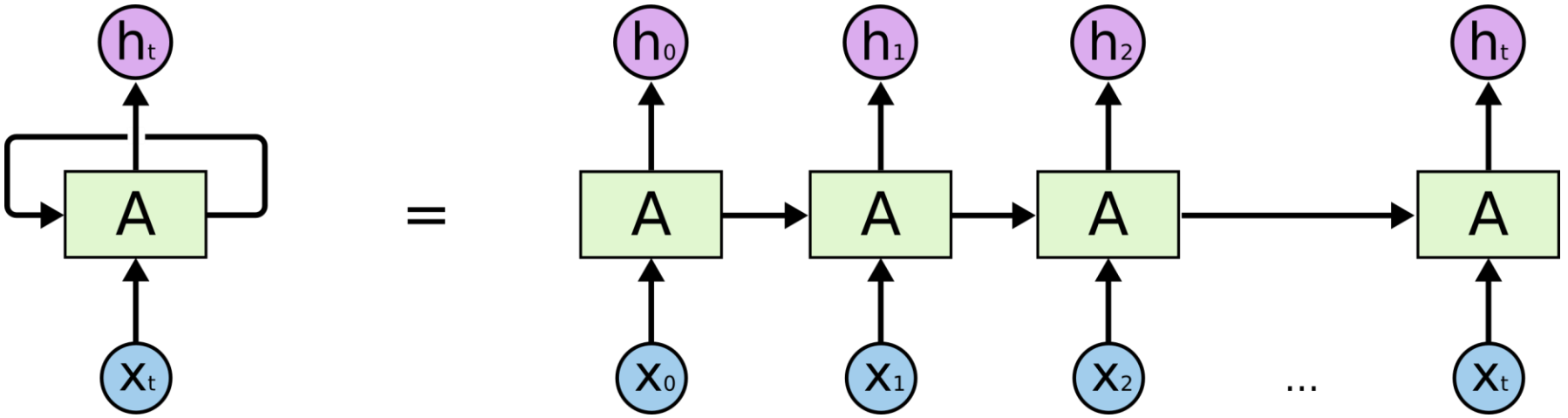
Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

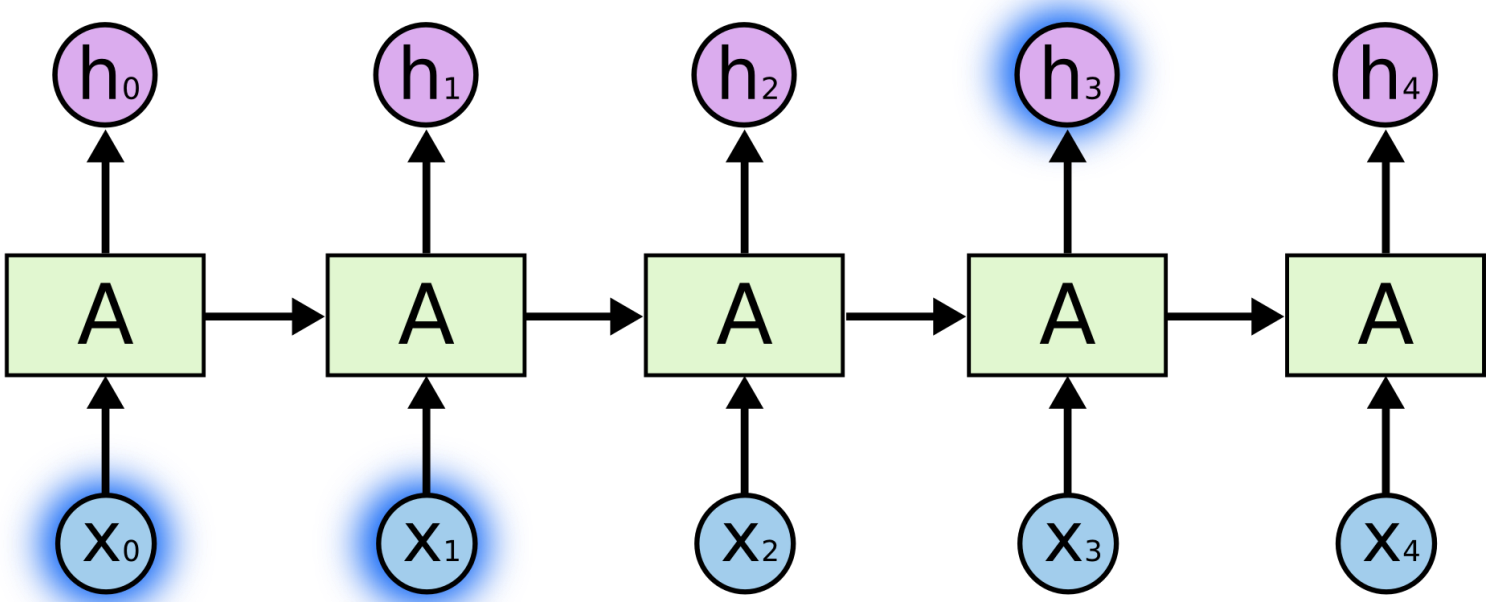


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

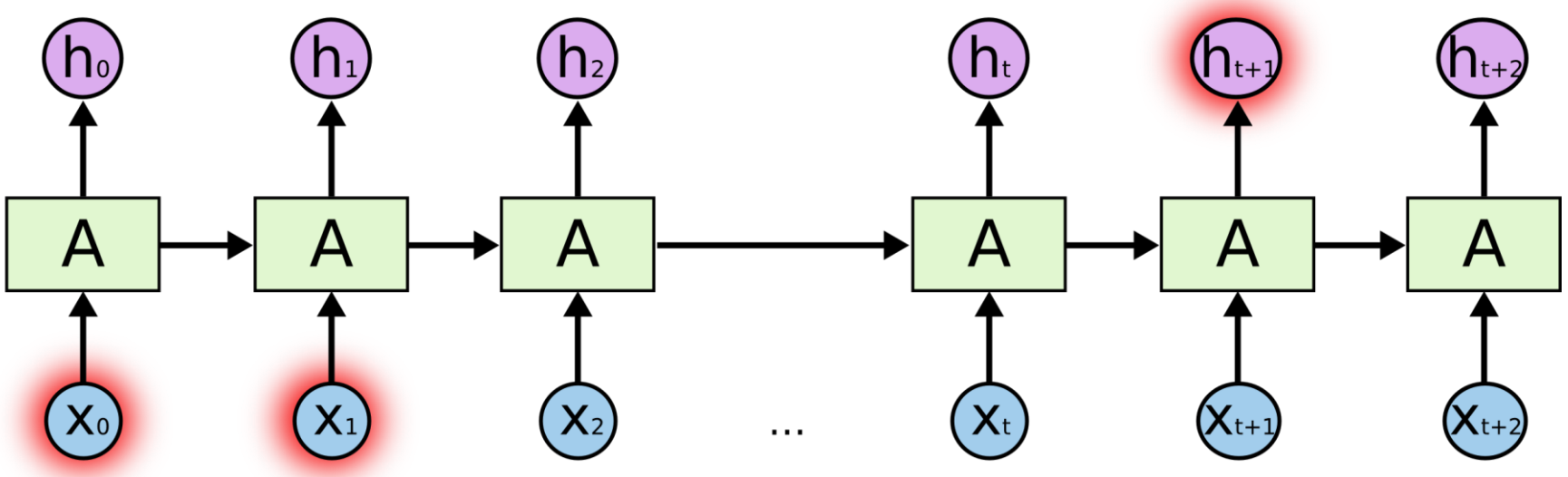
RNN
Long Short Term Memory (LSTM)



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

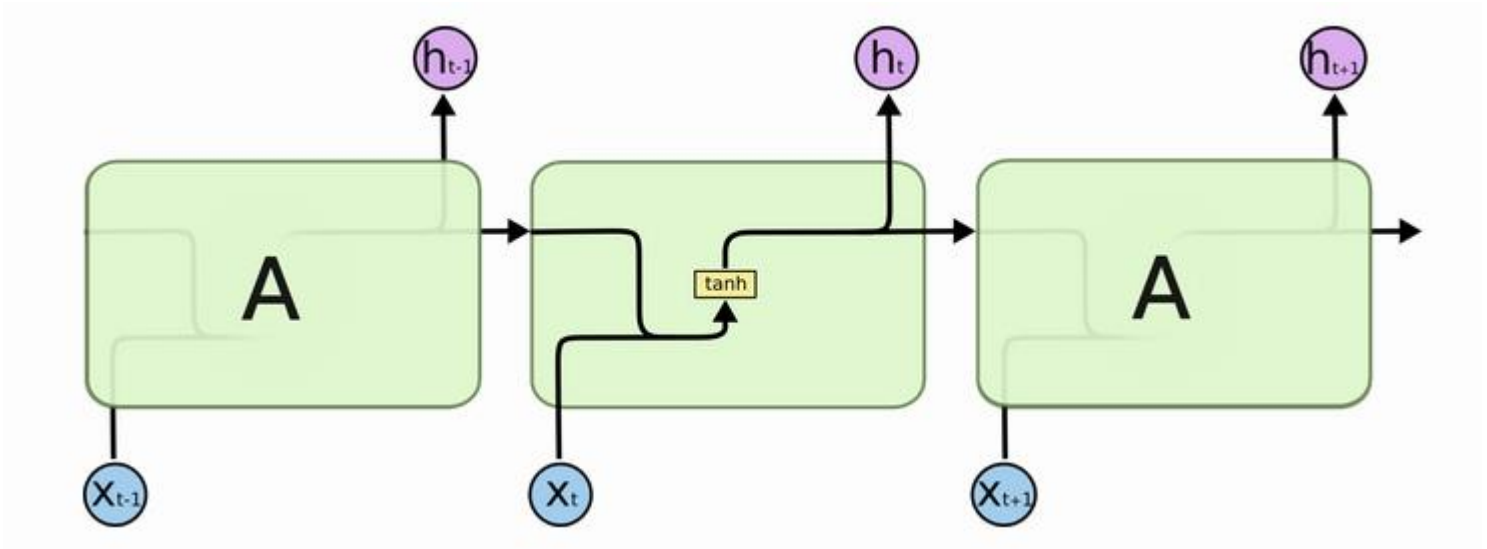


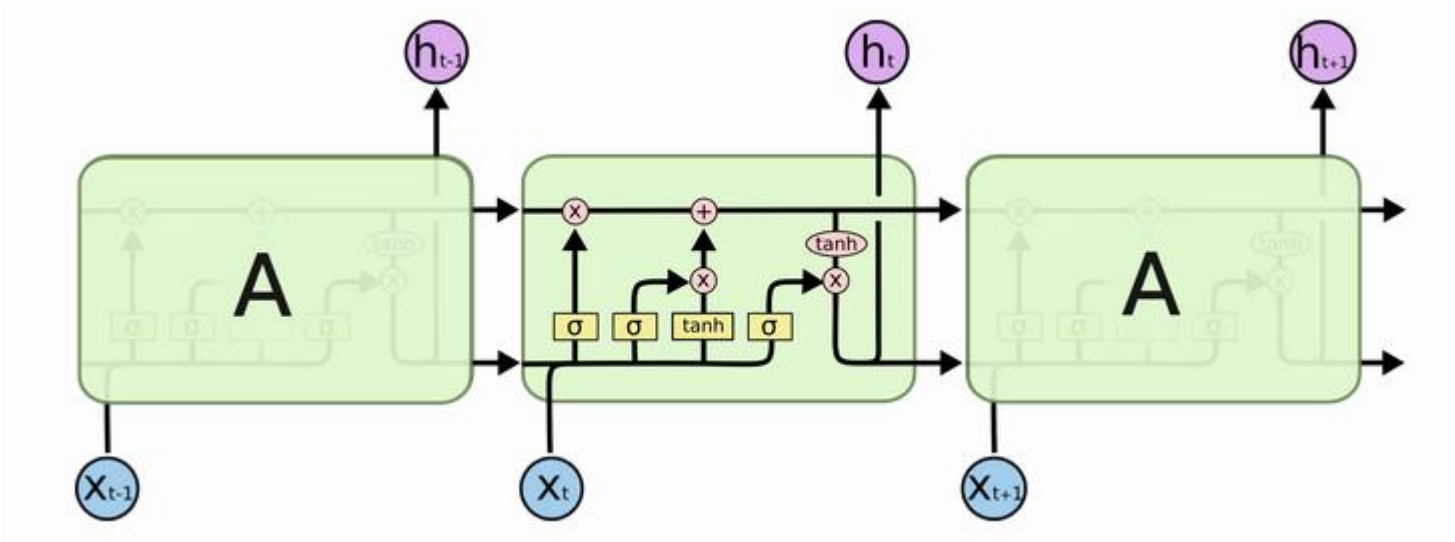
http://colah.github.io/posts/2015-08-Understanding-LSTMs/



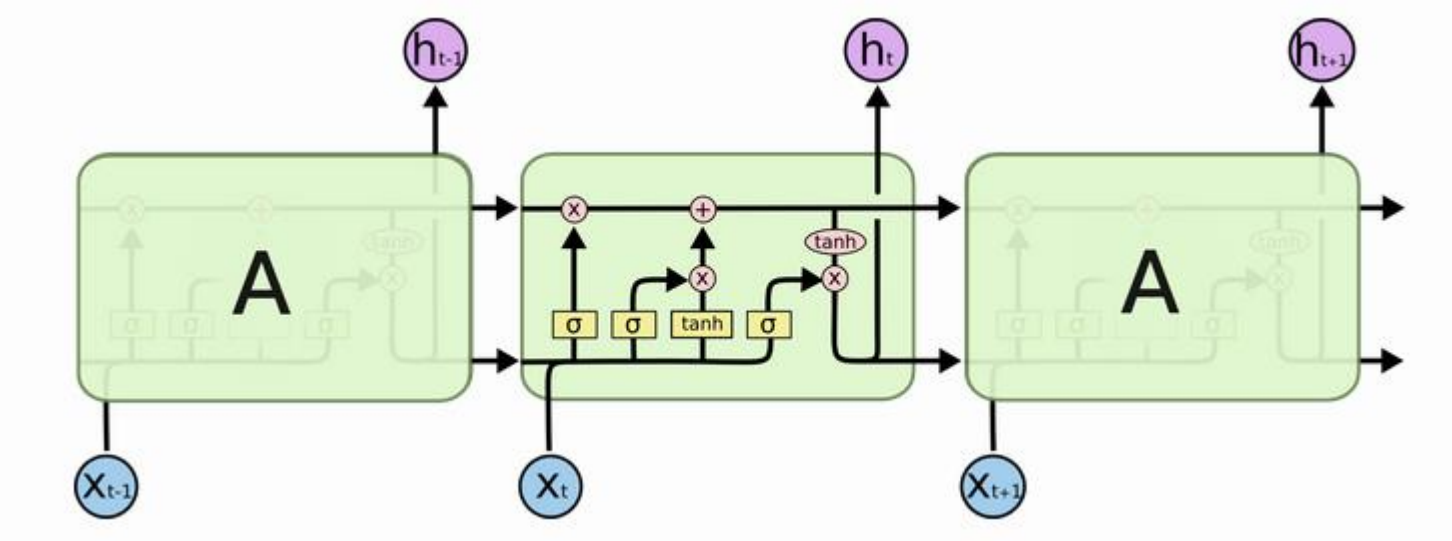
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM

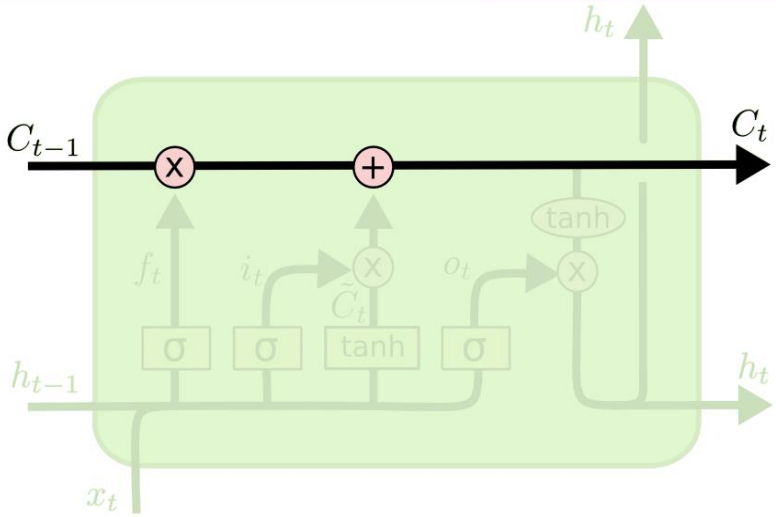




http://colah.github.io/posts/2015-08-Understanding-LSTMs/

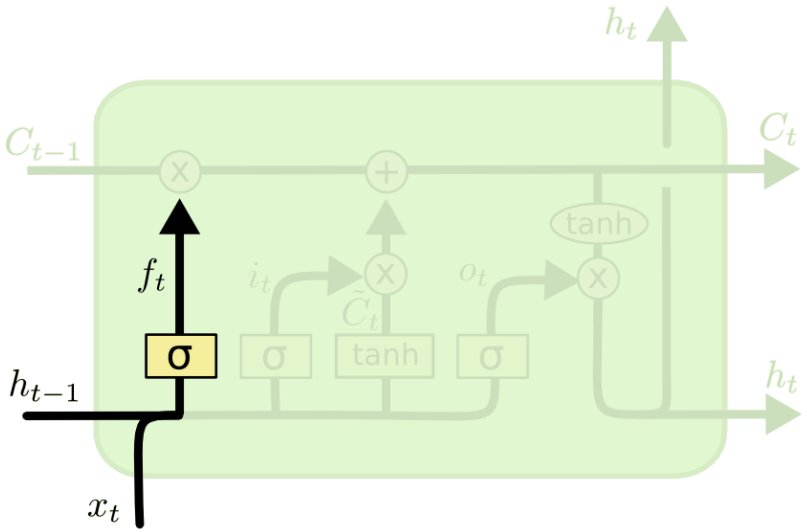


Uninterrupted gradient flow!



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

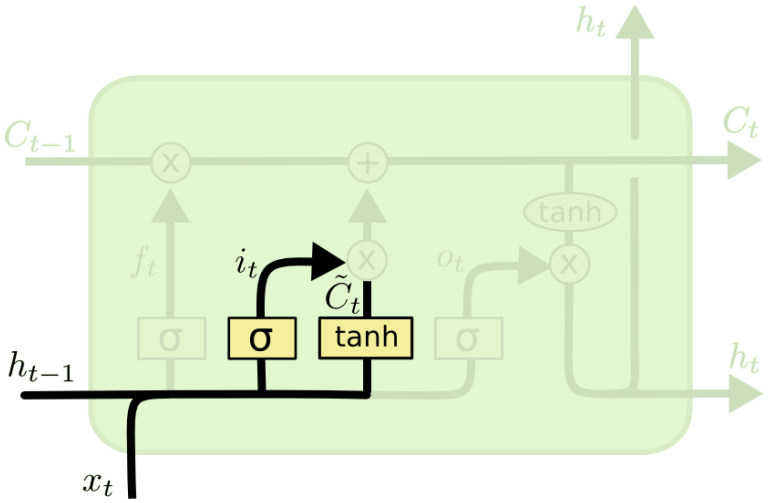
«Забування» попередньої інформації



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

«Вибір» нової інформації

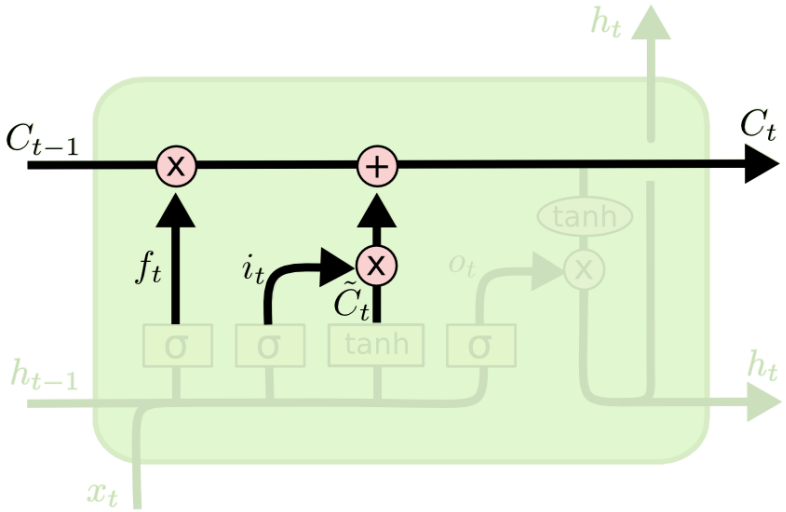


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

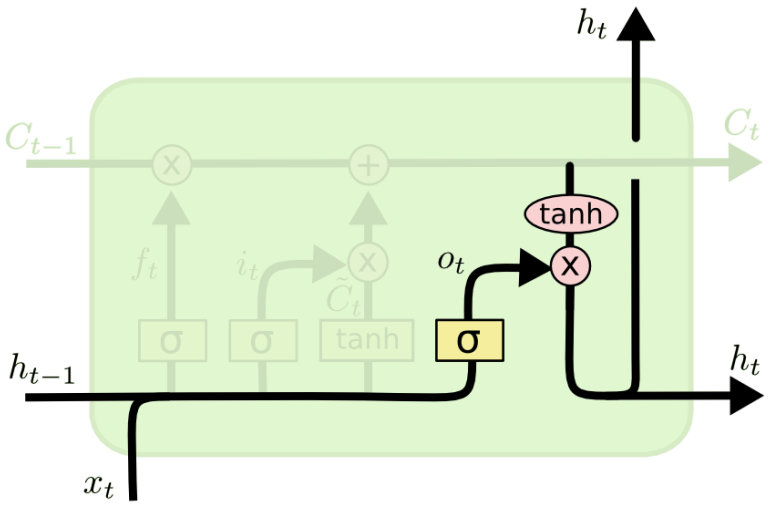
Формування поточного стану комірки



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Формування вихідної інформації

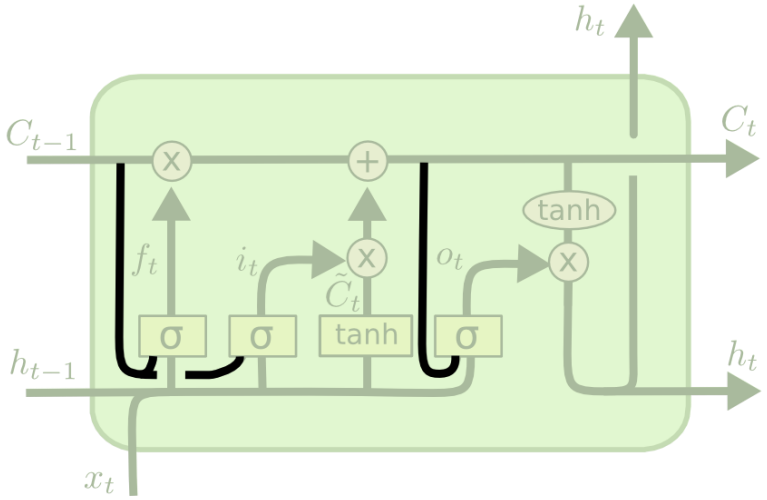


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Деякі різновиди LSTM

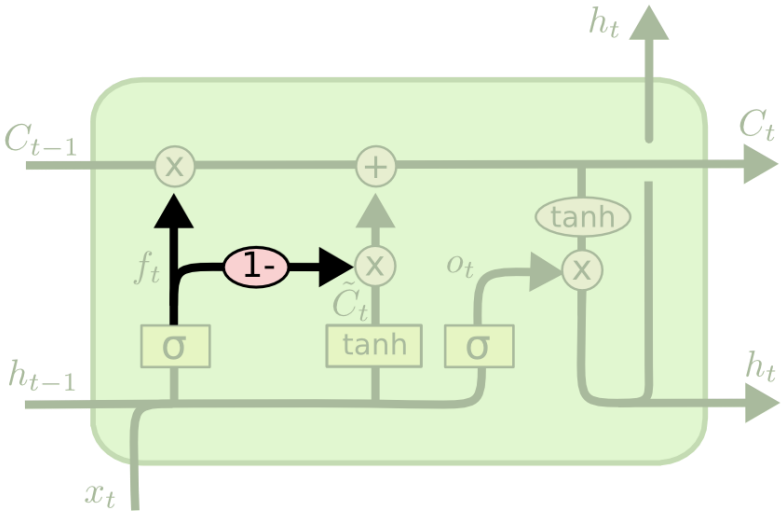


“peephole connections”

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$

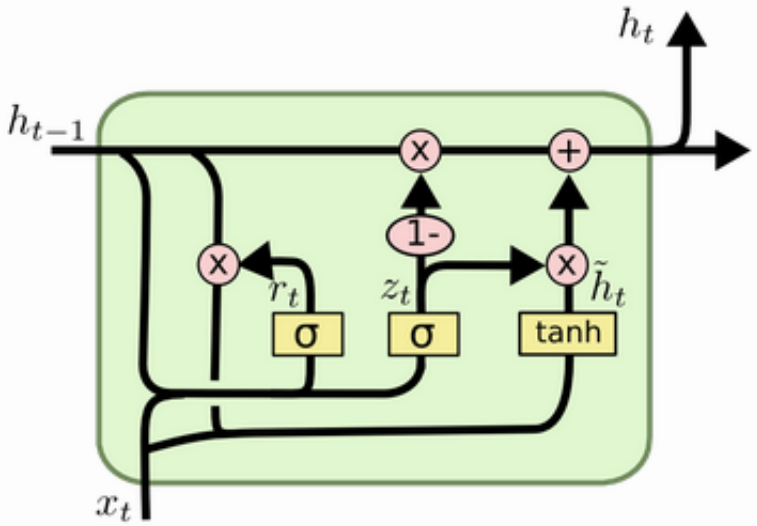
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$



$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit (GRU)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Leo Tolstoy's "War and Peace"

100 iterations

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

300 iterations

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

700 iterations

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

2000 iterations

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Shakespeare

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fe
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Algebraic Geometry (Latex)

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariciely in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S,} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ *(by the formal open covering* X *and a single map* $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ *over* U *compatible with the complex*

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If* T *is surjective we may assume that* T *is connected with residue fields of* S*. Moreover there exists a closed subspace* $Z \subset X$ *of* X *where* U *in* X' *is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) f *is locally of finite type. Since* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\ldots,0}$.

Lemma 0.2. *Let* X *be a locally Noetherian scheme over* S, $E = \mathcal{F}_{X/S}$. *Set* $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. *Since* $\mathcal{I}^n \subset \mathcal{I}^n$ *are nonzero over* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \overline{A}_2$ *works.*

Lemma 0.3. *In Situation ??. Hence we may assume* $\mathfrak{q}' = 0$.

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Linux Source Code

```
/*
 * 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_SECONDS << 12;
  return segtable;
}
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Generating Baby Names

Rudi Levette Berice Lussa Hany Mareanne Chrestina Carissy Marylen Hammine Janye Marlise Jacacrie Hendred Romand Charienna Nenotto Ette Dorane Wallen Marly Darine Salina Elvyn Ersia Maralena Minoria Ellia Charmin Antley Nerille Chelon Walmor Evena Jeryly Stachon Charisa Allisa Anatha Cathanie Geetra Alexie Jerin Cassen Herbett Cossie Velen Daurenge Robester Shermond Terisa Licia Roselen Ferine Jayn Lusine Charyanne Sales Sanny Resa Wallon Martine Merus Jelen Candica Wallin Tel Rachene Tarine Ozila Ketia Shanne Arnande Karella Roselina Alessia Chasty Deland Berther Geamar Jackein Mellisand Sagdy Nenc Lessie Rasemy Guen Gavi Milea Anneda Margoris Janin Rodelin Zeanna Elyne Janah Ferzina Susta Pey Castina

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Cell sensitive to position in line:

```
The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and did not,
surrender.
```

Cell that turns on inside quotes:

```
"You mean to imply that I have nothing to eat out of.... On the
contrary, I can supply you with everything even if you want to give
dinner parties," warmly replied Chichagov, who tried by every word he
spoke to prove his own rectitude and therefore imagined Kutuzov to be
animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating
smile: "I meant merely to say what I said."
```

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

A large portion of cells are not easily interpretable. Here is a typical example:

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /* Of the currently implemented string fields, PATH_MAX
   * defines the longest valid length.
   */
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Cell that turns on inside comments and quotes:

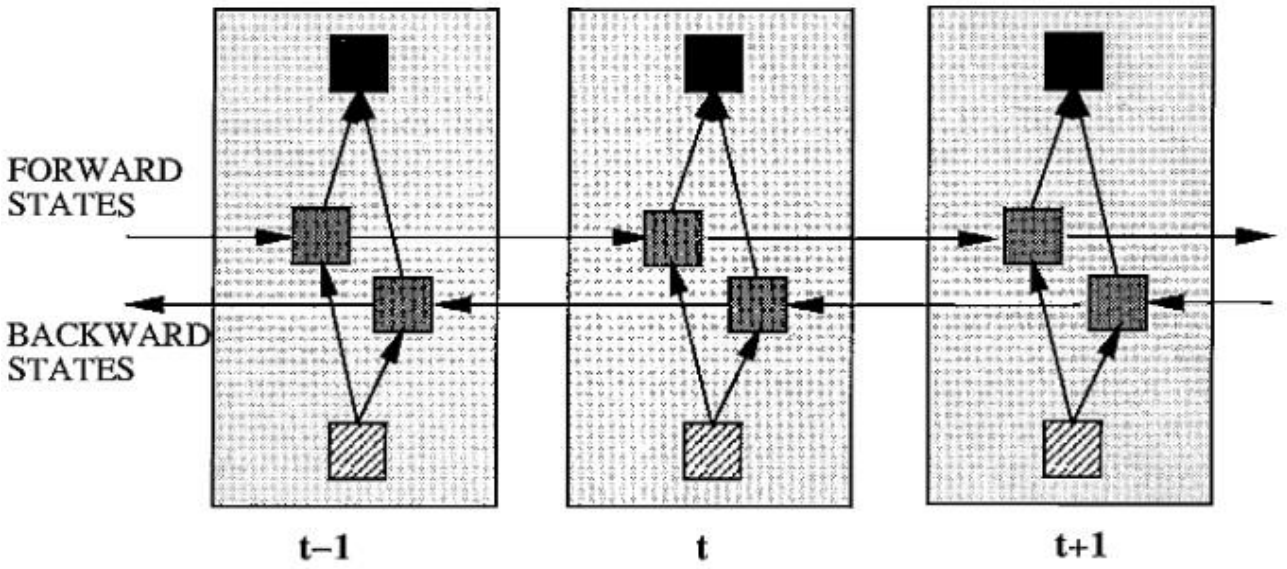
```
/* Duplicate LSM field information.  The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
           struct audit_field *sf)
{
  int ret = 0;
  char *lsm_str;
  /* our own copy of lsm_str */
  lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
  if (unlikely(!lsm_str))
    return -ENOMEM;
  df->lsm_str = lsm_str;
  /* our own (refreshed) copy of lsm_rule */
  ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                 (void **)&df->lsm_rule);
  /* Keep currently invalid fields around in case they
   * become valid after a policy reload. */
  if (ret == -EINVAL) {
    pr_warn("audit rule for LSM \'%s\' is invalid\n",
      df->lsm_str);
    ret = 0;
  }
  return ret;
}
```

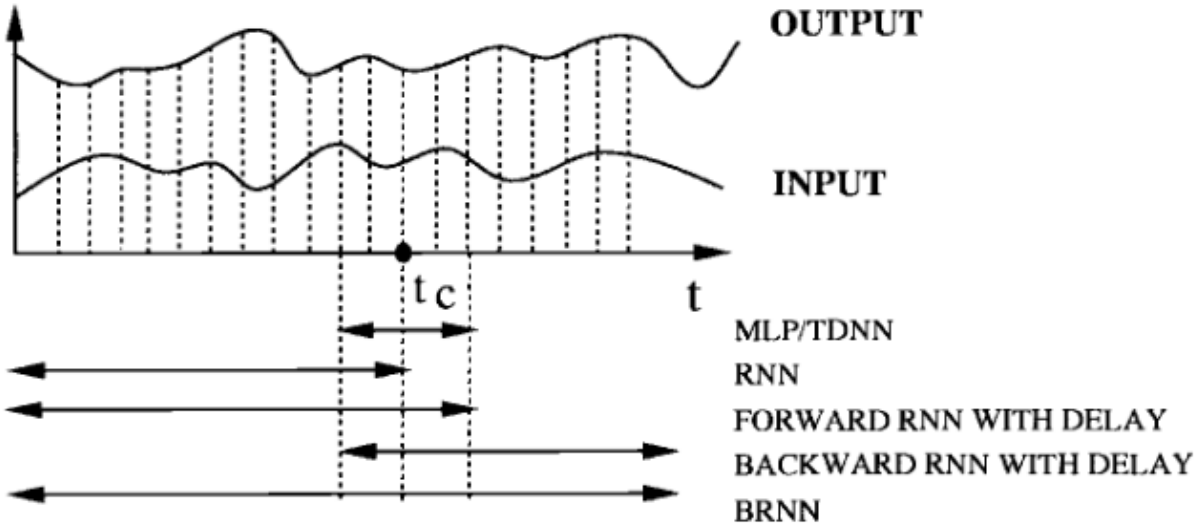
Cell that is sensitive to the depth of an expression:

```
#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;
}
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Bidirectional Recurrent Neural Networks





https://[redacted]semanticscholar.org/4b80/89bc9b49f84de43acc2eb8900035f7d492b2.pdf

RNN with TensorFlow

Module: tf.keras.layers

See Stable | See Nightly

Keras layers API.

Modules

`experimental` module: Public API for tf.keras.layers.experimental namespace.

Classes

`class AbstractRNNCell`: Abstract object representing an RNN cell.

`class Activation`: Applies an activation function to an output.

`class ActivityRegularization`: Layer that applies an update to the cost function based inpu

`class Add`: Layer that adds a list of inputs.

`class AdditiveAttention`: Additive attention layer, a.k.a. Bahdanau-style attention.

`class AlphaDropout`: Applies Alpha Dropout to the input.

`class Attention`: Dot-product attention layer, a.k.a. Luong-style attention.

`class Average`: Layer that averages a list of inputs element-wise.

`class AveragePooling1D`: Average pooling for temporal data.

`class AveragePooling2D`: Average pooling operation for spatial data.

`class AveragePooling3D`: Average pooling operation for 3D data (spatial or spatio-temporal).

`class AvgPool1D`: Average pooling for temporal data.

`class AvgPool2D`: Average pooling operation for spatial data.

`class AvgPool3D`: Average pooling operation for 3D data (spatial or spatio-temporal).

`class BatchNormalization`: Layer that normalizes its inputs.

`class Bidirectional`: Bidirectional wrapper for RNNs.

`class CategoryEncoding`: A preprocessing layer which encodes integer features.

`class CenterCrop`: A preprocessing layer which crops images.

`class Concatenate`: Layer that concatenates a list of inputs.

`class Conv1D`: 1D convolution layer (e.g. temporal convolution).

`class Conv1D`: 1D convolution layer (e.g. temporal convolution).

`class Conv1DTranspose`: Transposed convolution layer (sometimes called Deconvolution).

`class Conv2D`: 2D convolution layer (e.g. spatial convolution over images).

`class Conv2DTranspose`: Transposed convolution layer (sometimes called Deconvolution).

`class Conv3D`: 3D convolution layer (e.g. spatial convolution over volumes).

`class Conv3DTranspose`: Transposed convolution layer (sometimes called Deconvolution).

`class ConvLSTM1D`: 1D Convolutional LSTM.

`class ConvLSTM2D`: 2D Convolutional LSTM.

`class ConvLSTM3D`: 3D Convolutional LSTM.

`class Convolution1D`: 1D convolution layer (e.g. temporal convolution).

`class Convolution1DTranspose`: Transposed convolution layer (sometimes called Deconvolution).

`class Convolution2D`: 2D convolution layer (e.g. spatial convolution over images).

`class Convolution2DTranspose`: Transposed convolution layer (sometimes called Deconvolution).

`class Convolution3D`: 3D convolution layer (e.g. spatial convolution over volumes).

`class Convolution3DTranspose`: Transposed convolution layer (sometimes called Deconvolution).

`class Cropping1D`: Cropping layer for 1D input (e.g. temporal sequence).

`class Cropping2D`: Cropping layer for 2D input (e.g. picture).

`class Cropping3D`: Cropping layer for 3D data (e.g. spatial or spatio-temporal).

`class Dense`: Just your regular densely-connected NN layer.

`class DenseFeatures`: A layer that produces a dense `Tensor` based on given `feature_columns`.

`class DepthwiseConv1D`: Depthwise 1D convolution.

`class DepthwiseConv2D`: Depthwise 2D convolution.

`class Discretization`: A preprocessing layer which buckets continuous features by ranges.

`class Dot`: Layer that computes a dot product between samples in two tensors.

`class Dropout`: Applies Dropout to the input.

`class ELU`: Exponential Linear Unit.

`class EinsumDense`: A layer that uses `tf.einsum` as the backing computation.

`class Embedding`: Turns positive integers (indexes) into dense vectors of fixed size.

`class Flatten`: Flattens the input. Does not affect the batch size.

https://www.tensorflow.org/api_docs/python/tf/keras/layers

tf.keras.layers.SimpleRNN

View source on GitHub

Fully-connected RNN where the output is to be fed back to input.

Inherits From: `RNN`, `Layer`, `Module`

View aliases

```
tf.keras.layers.SimpleRNN(
    units,
    activation='tanh',
    use_bias=True,
    kernel_initializer='glorot_uniform',
    recurrent_initializer='orthogonal',
    bias_initializer='zeros',
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    stateful=False,
    unroll=False,
    **kwargs
)
```

tf.keras.layers.LSTM

View source on GitHub

Long Short-Term Memory layer - Hochreiter 1997.

Inherits From: `RNN`, `Layer`, `Module`

```
tf.keras.layers.LSTM(
    units,
    activation='tanh',
    recurrent_activation='sigmoid',
    use_bias=True,
    kernel_initializer='glorot_uniform',
    recurrent_initializer='orthogonal',
    bias_initializer='zeros',
    unit_forget_bias=True,
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    stateful=False,
    time_major=False,
    unroll=False,
    **kwargs
)
```

tf.keras.layers.GRU

View source on GitHub

Gated Recurrent Unit - Cho et al. 2014.

Inherits From: `RNN`, `Layer`, `Module`

```
tf.keras.layers.GRU(
    units,
    activation='tanh',
    recurrent_activation='sigmoid',
    use_bias=True,
    kernel_initializer='glorot_uniform',
    recurrent_initializer='orthogonal',
    bias_initializer='zeros',
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    stateful=False,
    unroll=False,
    time_major=False,
    reset_after=True,
    **kwargs
)
```

tf.keras.layers.Bidirectional

View source on GitHub

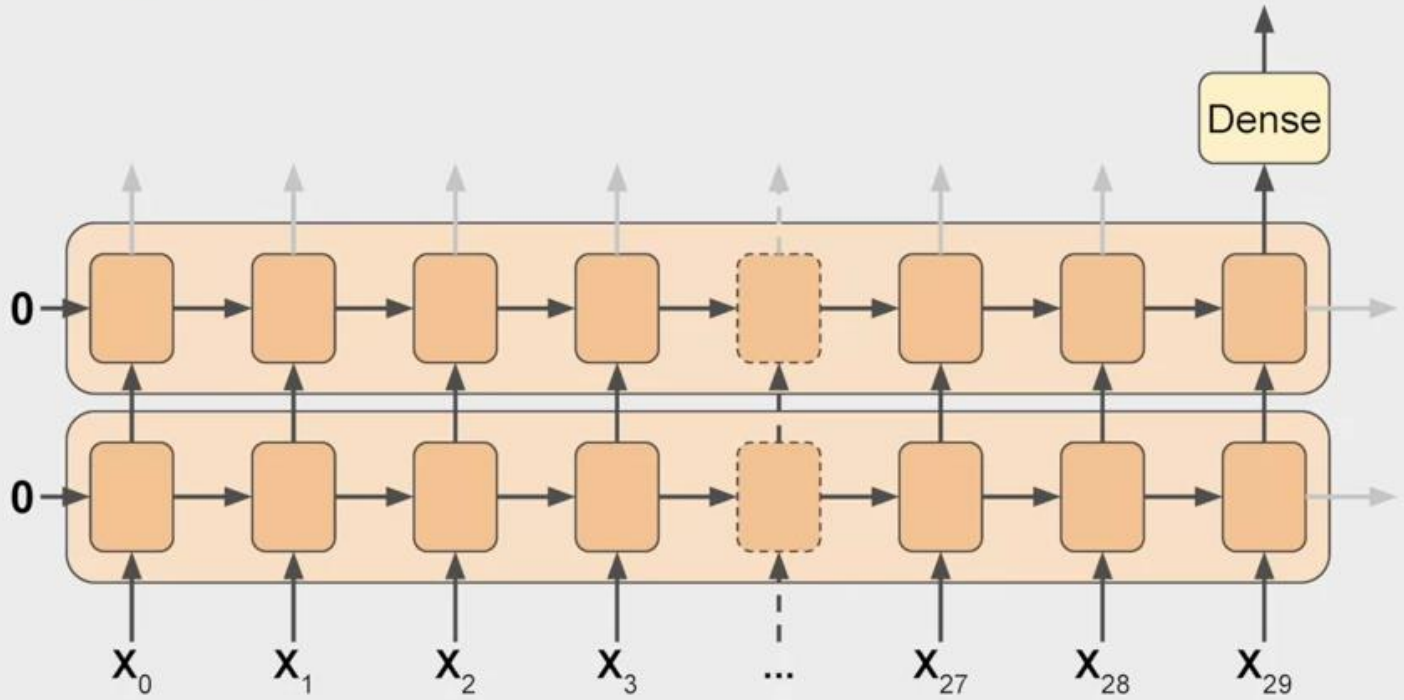
Bidirectional wrapper for RNNs.

Inherits From: `Wrapper`, `Layer`, `Module`

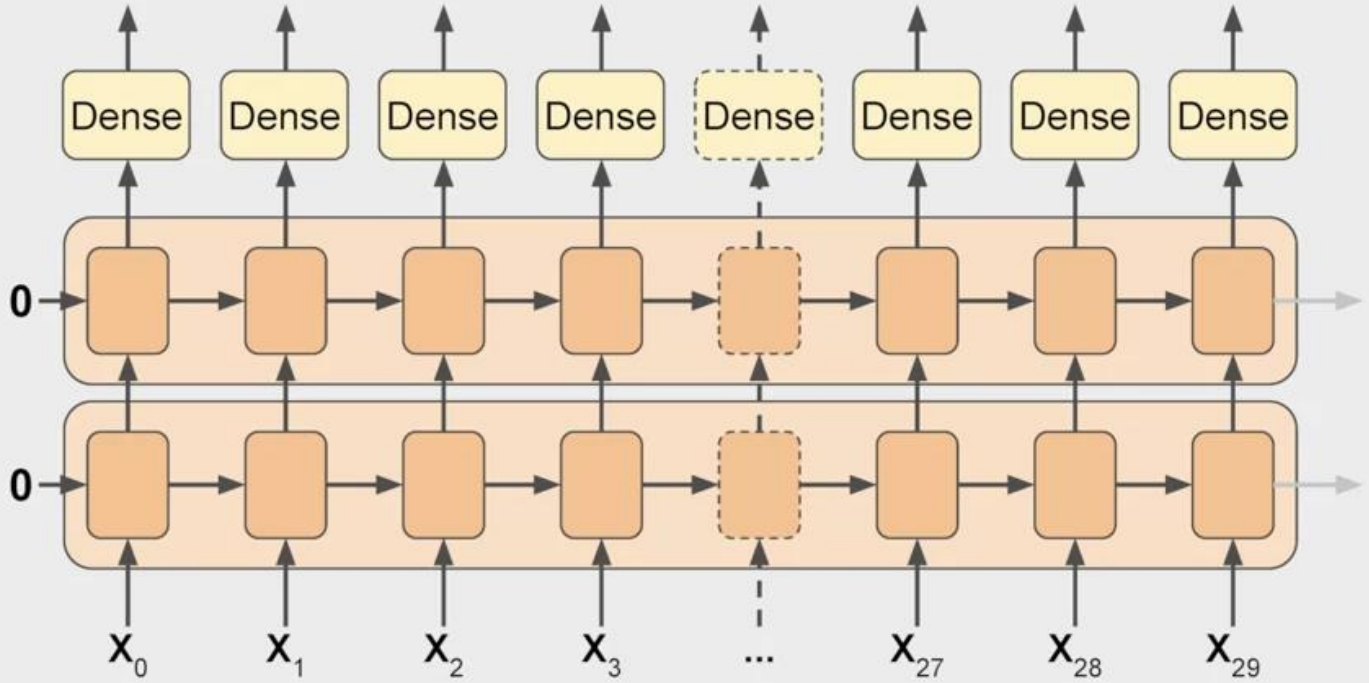
View aliases

```
tf.keras.layers.Bidirectional(
    layer,
    merge_mode='concat',
    weights=None,
    backward_layer=None,
    **kwargs
)
```

```
model = keras.models.Sequential([
  keras.layers.SimpleRNN(20, return_sequences=True,
                         input_shape=[None, 1]),
  keras.layers.SimpleRNN(20),
  keras.layers.Dense(1)
])
```



```
model = keras.models.Sequential([
  keras.layers.SimpleRNN(20, return_sequences=True,
                         input_shape=[None, 1]),
  keras.layers.SimpleRNN(20, return_sequences=True),
  keras.layers.Dense(1)
])
```



https://www.coursera.org/learn/tensorflow-sequences-time-series-and-prediction/home/info

Was this helpful?

tf.keras.layers.TimeDistributed

View source on GitHub

This wrapper allows to apply a layer to every temporal slice of an input.

Inherits From: `Wrapper`, `Layer`, `Module`

View aliases

```
tf.keras.layers.TimeDistributed(
    layer, **kwargs
)
```

Used in the notebooks

Used in the tutorials

- Load video data

Every input should be at least 3D, and the dimension of index one of the first input will be considered to be the temporal dimension.

Активация
Чтобы активи
"Параметры".

```
tf.keras.layers.SimpleRNN(
    units, activation='tanh', use_bias=True,
    kernel_initializer='glorot_uniform',
    recurrent_initializer='orthogonal',
    bias_initializer='zeros', kernel_regularizer=None,
    recurrent_regularizer=None, bias_regularizer=None, activity_regularizer=None,
    kernel_constraint=None, recurrent_constraint=None, bias_constraint=None,
    dropout=0.0, recurrent_dropout=0.0, return_sequences=False, return_state=False
    go_backwards=False, stateful=False, unroll=False, **kwargs
)
```

`return_sequences`	Boolean. Whether to return the last output in the output sequence, or the full sequence. Default: `False`.
`return_state`	Boolean. Whether to return the last state in addition to the output. Default: `False`

```
inputs = np.random.random([32, 10, 8]).astype(np.float32)
simple_rnn = tf.keras.layers.SimpleRNN(4)

output = simple_rnn(inputs)  # The output has shape `[32, 4]`.

simple_rnn = tf.keras.layers.SimpleRNN(
    4, return_sequences=True, return_state=True)

# whole_sequence_output has shape `[32, 10, 4]`.
# final_state has shape `[32, 4]`.
whole_sequence_output, final_state = simple_rnn(inputs)
```

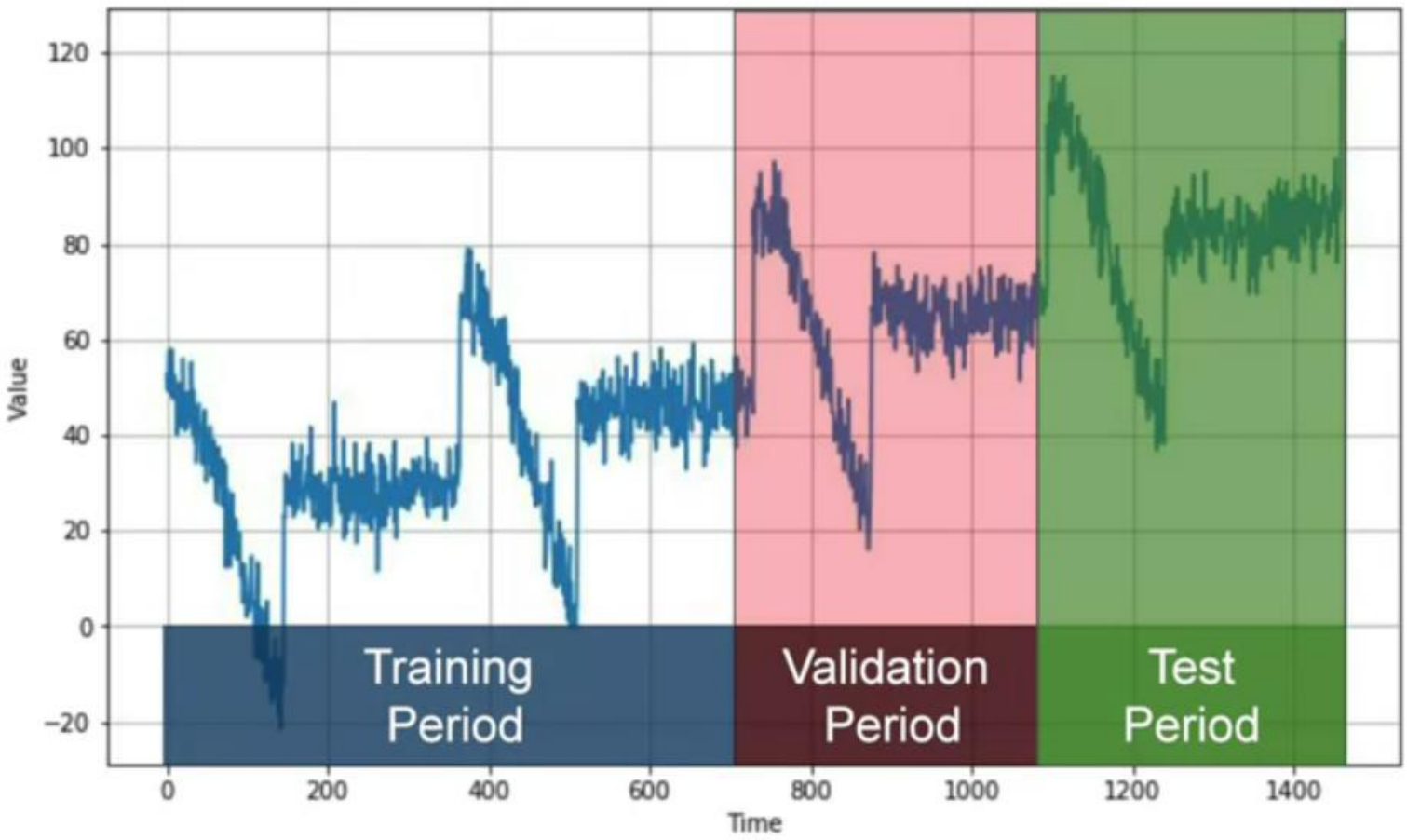
https://www.tensorflow.org/api_docs/python/tf/keras/layers/SimpleRNN

```
tf.keras.layers.Bidirectional(
    layer, merge_mode='concat', weights=None, backward_layer=None,
    **kwargs
)
```

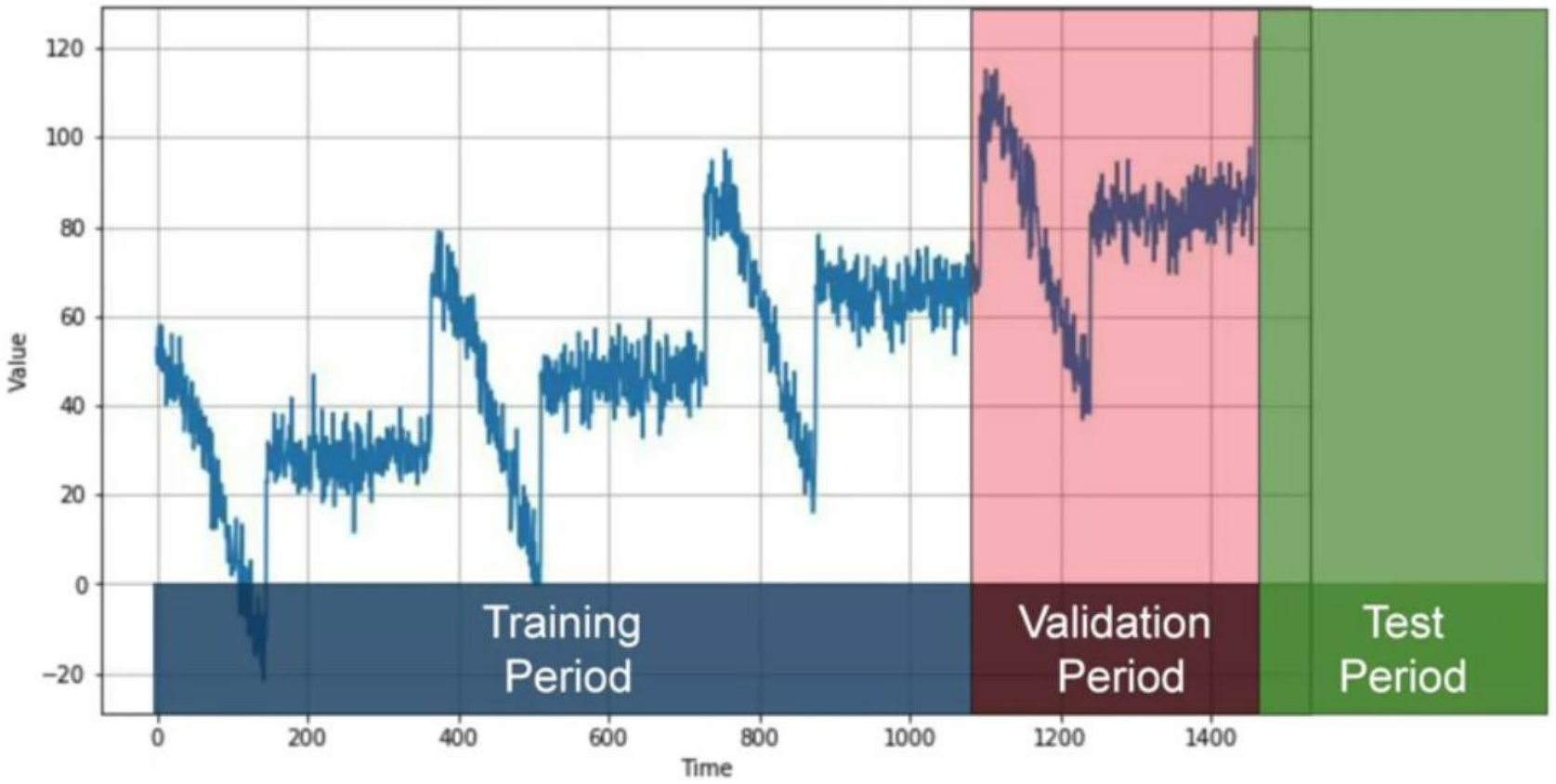
```
model = Sequential()
model.add(Bidirectional(LSTM(10, return_sequences=True), input_shape=(5, 10)))
model.add(Bidirectional(LSTM(10)))
model.add(Dense(5))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='rmsprop')

 # With custom backward layer
 model = Sequential()
 forward_layer = LSTM(10, return_sequences=True)
 backward_layer = LSTM(10, activation='relu', return_sequences=True,
                       go_backwards=True)
 model.add(Bidirectional(forward_layer, backward_layer=backward_layer,
                         input_shape=(5, 10)))
 model.add(Dense(5))
 model.add(Activation('softmax'))
 model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
```

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Bidirectional

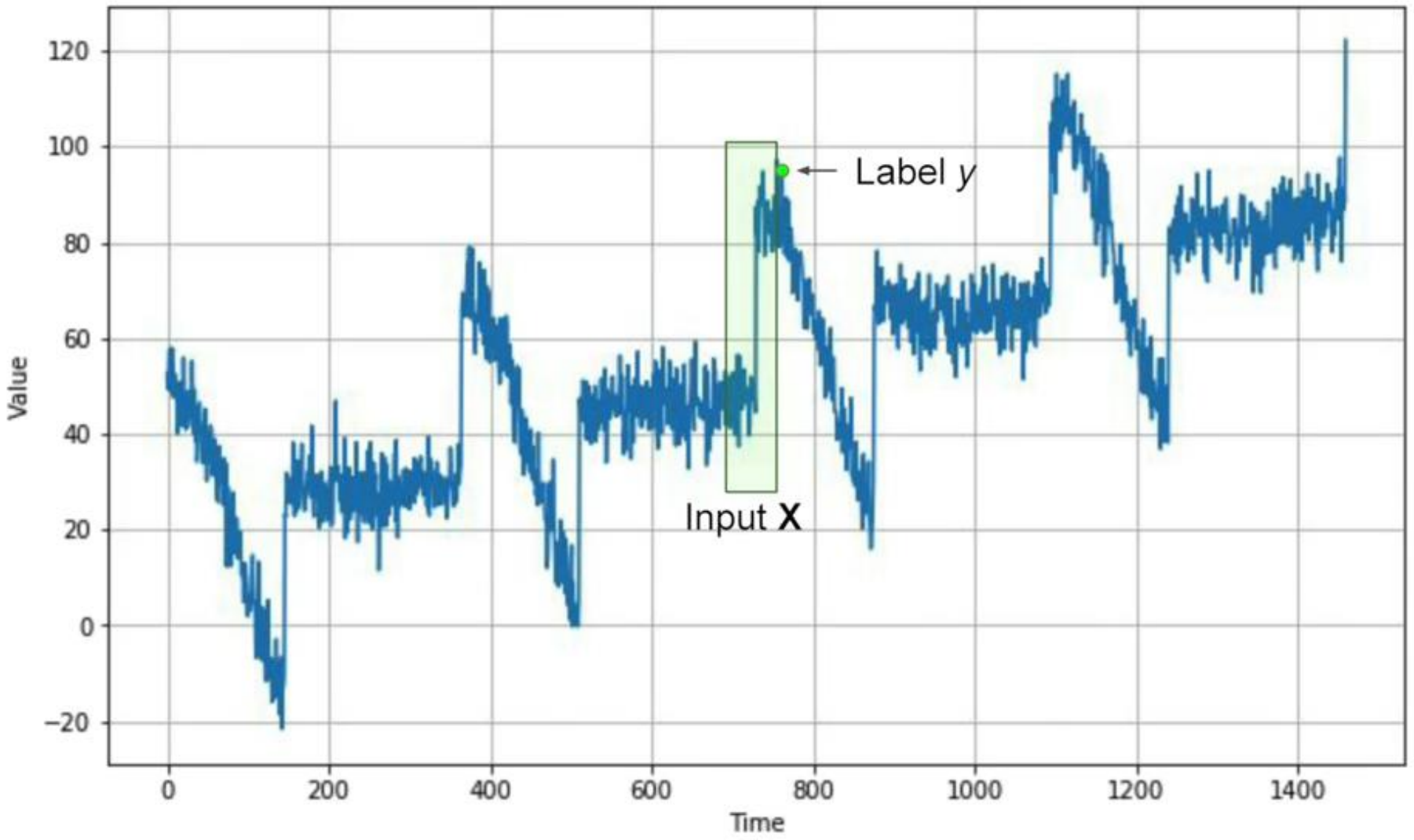
RNN for time series forecasting



https://www.co[illegible]org/learn/tensorflow-sequences-time-series-and-prediction/home/info



https://www.co[illegible]org/learn/tensorflow-sequences-time-series-and-prediction/home/info

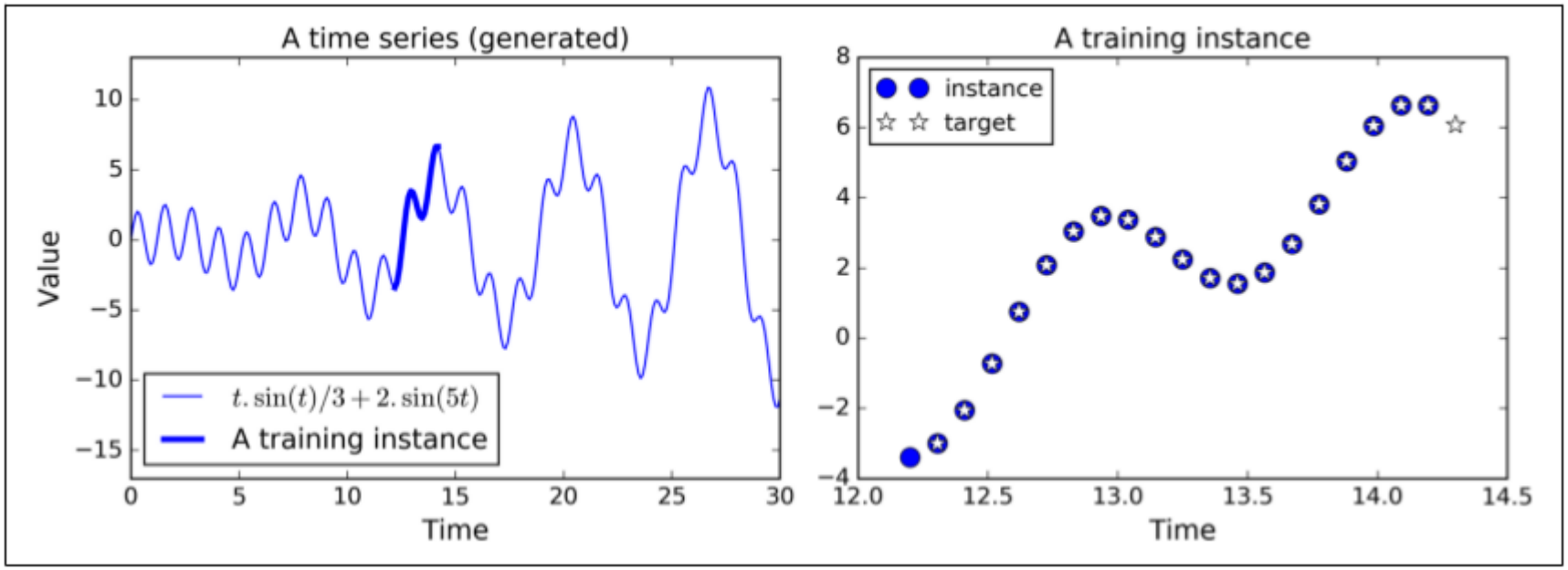


https://www.co[illegible]org/learn/tensorflow-sequences-time-series-and-prediction/home/info

$$x_1, x_2, \ldots, x_{n-2}, x_{n-1}$$

$$x_n = f(x_{n-1}, \ldots, x_{n-l})$$

$$y_4 = f(x_1, x_2, x_3)$$
$$y_5 = f(x_2, x_3, x_4)$$
$$y_6 = f(x_3, x_4, x_5)$$



```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  dataset = tf.data.Dataset.from_tensor_slices(series)
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  dataset = dataset.shuffle(shuffle_buffer)
                   .map(lambda window: (window[:-1], window[-1]))
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset
```

https://www.co[illegible]org/learn/tensorflow-sequences-time-series-and-prediction/home/info

```python
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  dataset = tf.data.Dataset.from_tensor_slices(series)
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  dataset = dataset.shuffle(shuffle_buffer)
                   .map(lambda window: (window[:-1], window[-1]))
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset
```

```python
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1)
for window_dataset in dataset:
  for val in window_dataset:
    print(val.numpy(), end=" ")
  print()
```

```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
6 7 8 9
7 8 9
8 9
9
```

```python
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
for window_dataset in dataset:
  for val in window_dataset:
    print(val.numpy(), end=" ")
  print()
```

```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
```

https://www.co[illegible]org/learn/tensorflow-sequences-time-series-and-prediction/home/info

```python
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  dataset = tf.data.Dataset.from_tensor_slices(series)
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  dataset = dataset.shuffle(shuffle_buffer)
                   .map(lambda window: (window[:-1], window[-1]))
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset
```

```python
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1:]))
for x,y in dataset:
  print(x.numpy(), y.numpy())
```

```
[0 1 2 3] [4]
[1 2 3 4] [5]
[2 3 4 5] [6]
[3 4 5 6] [7]
[4 5 6 7] [8]
[5 6 7 8] [9]
```

https://www.co[illegible]org/learn/tensorflow-sequences-time-series-and-prediction/home/info

```
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
  tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                      input_shape=[None]),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True)),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True)),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])


model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning_rate=1e-6,
momentum=0.9))
model.fit(dataset,epochs=100)
```

https://www.coursera.org/learn/[illegible]flow-sequences-time-series-and-prediction/lecture/8u3wq/coding-lstms

tf.expand_dims

For example:

If you have a single image of shape `[height, width, channels]`:

```
>>> image = tf.zeros([10,10,3])
```

You can add an outer `batch` axis by passing `axis=0`:

```
>>> tf.expand_dims(image, axis=0).shape.as_list()
[1, 10, 10, 3]
```

The new axis location matches Python `list.insert(axis, 1)`:

```
>>> tf.expand_dims(image, axis=1).shape.as_list()
[10, 1, 10, 3]
```

Following standard Python indexing rules, a negative `axis` counts from the end so `axis=-1` adds an inner most dimension:

```
>>> tf.expand_dims(image, -1).shape.as_list()
[10, 10, 3, 1]
```