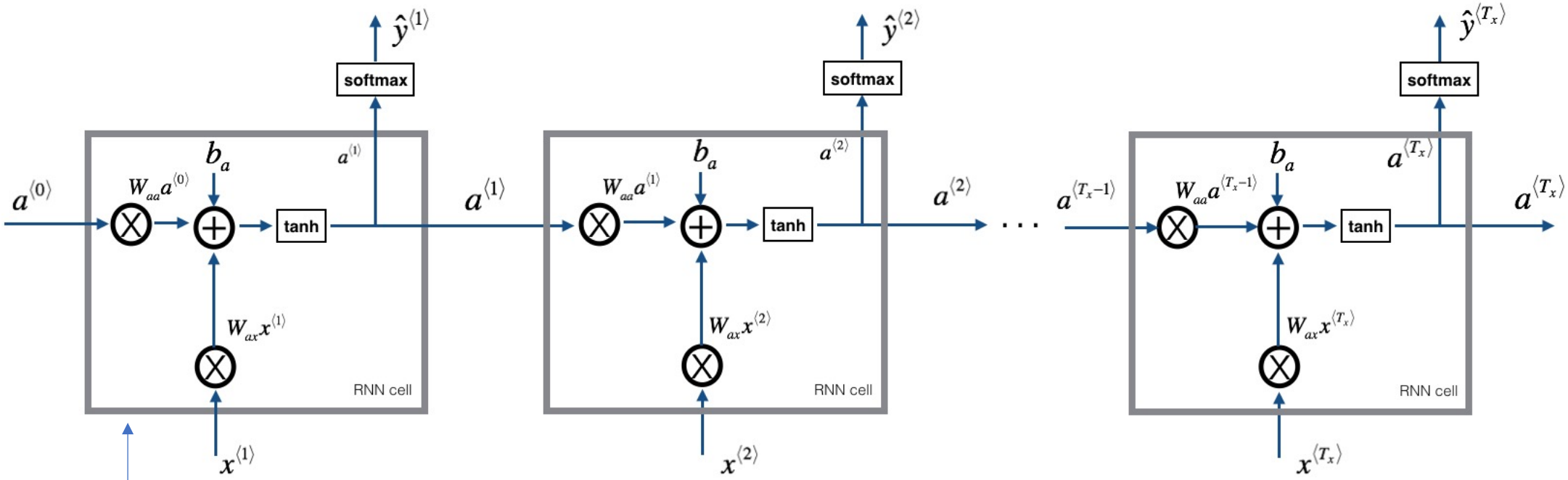
Building a RNN and LSTM with pytorch

Structure of RNN

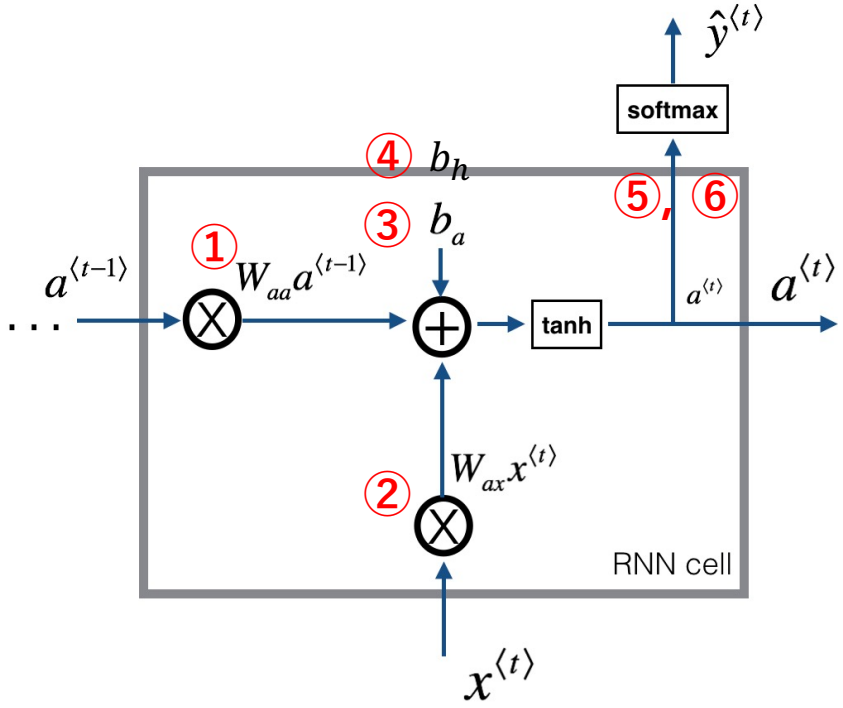


Cell

$x^{\langle t \rangle}: input$
$y^{\langle t \rangle}: output$
$a^{\langle t \rangle}: activation$

$W_{aa}: a \rightarrow a\ parameters$
$W_{ax}: x \rightarrow a\ parameters$
$W_{ya}: a \rightarrow y\ parameters$
$b_a: bias\ for\ tanh\ activation$
$b_y: bais\ for\ softmax\ activation$

RNN: Forward process (cell)



- **Given**
  $x^{\langle t \rangle}$: $input$
  $a^{\langle t-1 \rangle}$: $activation$

- **Parameters**
  W_{ax}: $x \rightarrow a$ $parameters$
  W_{aa}: $a \rightarrow a$ $parameters$
  W_{ya}: $a \rightarrow y$ $parameters$
  b_a: $bias\ for\ tanh\ activation$
  b_y: $bais\ for\ softmax\ activation$
  $\boldsymbol{b_h}$: $\boldsymbol{bais\ for\ hidden\ state}$

- **Computations**

$$a^{\langle t \rangle} = tanh\left(\overset{②}{W_{ax}x^{\langle t \rangle}} + \overset{①}{W_{aa}a^{\langle t-1 \rangle}} + \overset{③}{b_a} + \overset{④}{\boldsymbol{b_h}}\right)$$

$$\hat{y}^{\langle t \rangle} = soft\max(\underset{⑤}{W_{ya}a^{\langle t \rangle}} + \underset{⑥}{b_y})$$

- **Python code**

parameters

```
# Retrieve parameters from "parameters"
Wax = parameters["Wax"]
Waa = parameters["Waa"]
Wya = parameters["Wya"]
ba = parameters["ba"]
by = parameters["by"]
```

Computations

```
a_next = np.tanh(np.dot(Waa, a_prev) + np.dot(Wax, xt) + ba)
yt_pred = softmax(np.dot(Wya, a_next) + by)
```

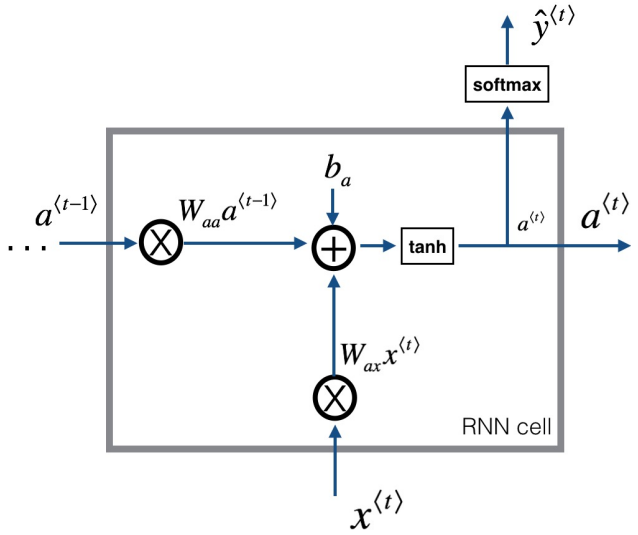
→

```
def rnn_cell_forward(xt, a_prev, parameters):
```

Define RNN with pytorch

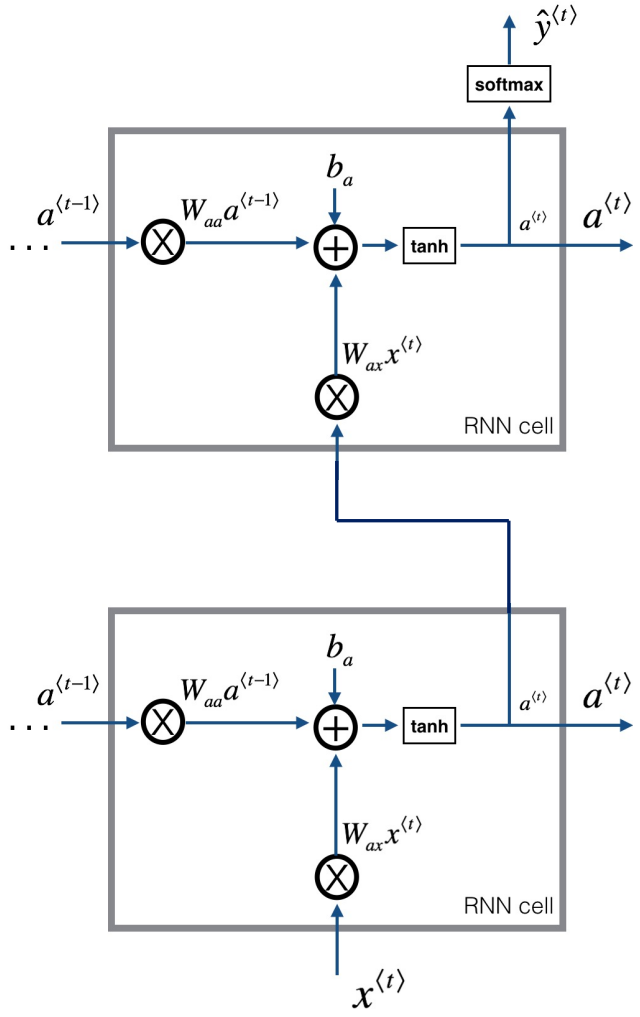
- Generate RNN model
  RNN_model = torch.nn.RNN(input_size, hidden_size, num_layers, bias, …)
  - input_size: 入力サイズ
  - hidden_size: hidden stateの数
  - num_layers: cellの数

- 例) input size = 2
  hidden_size = 5
  num_layers = 1



[0,1]

input size = 9
hidden_size = 5
num_layers = 2



[0, 0, 0, 0, 1, 0, 0, 0, 0]

Today's RNN

- 目標：次の単語を予測する
  ex) input: “I like” → output: “dogs” or “cats”

- コードの構造
  A. データの前処理及び関数、パラメータの宣言
  B. RNNの構築
  C. 学習
  D. 検証

Forward 計算
勾配計算
Backward 計算(パラメータ更新)

cf) Generate dino names from RNN with numpy

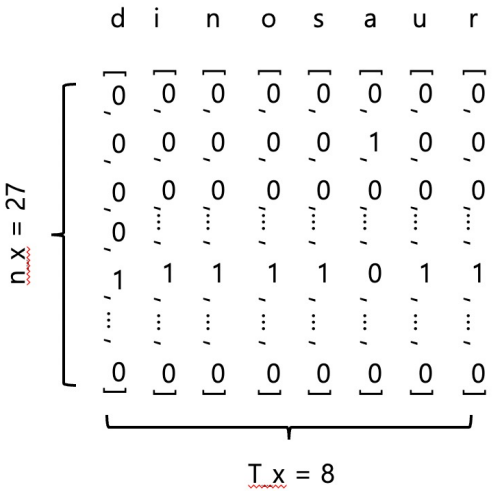
- Alphabet to number

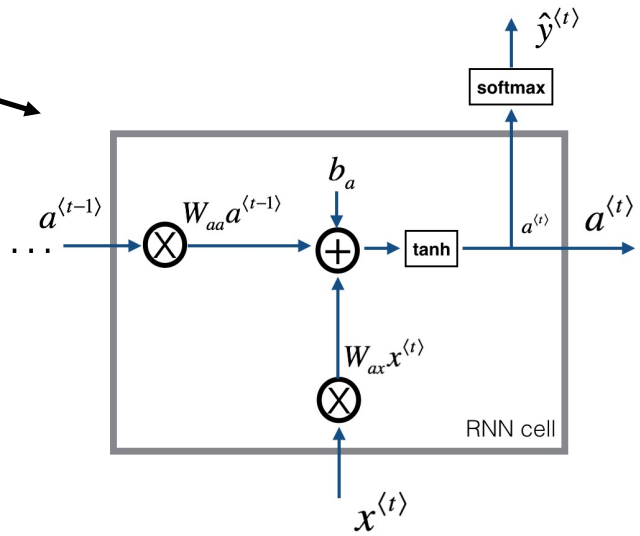
```
\n = 0
a = 1
b = 2
c = 3
d = 4

...

z = 26
```

- Number to one-hot vector

$\backslash n = [0, 0, 0, \dots, 0]^T$
$a = [0, 1, 0, \dots, 0]^T$
$b = [0, 0, 1, \dots, 0]^T$
...
$z = [0, 0, 0, \dots, 1]^T$

- Example of the word ‘dinosaur’





A. データの前処理及び変数、関数の宣言

- データセット（6文章）
  ["i like dogs", "i love coffee", "i hate milk", "you like cats", "you love milk", "you hate coffee"]

- One to hot vectorに変換

```
{'cats': 0,    ⟶ [1., 0., 0., 0., 0., 0., 0., 0., 0.]
 'coffee': 7,
 'dogs': 6,
 'hate': 1,
 'i': 5,
 'like': 2,
 'love': 3,
 'milk': 4,
 'you': 8}     ⟶ [0., 0., 0., 0., 0., 0., 0., 0., 1.]
```

  → 9 classes

- 必要な変数、関数の設定
  batch size = 6
  hidden_size = 5
  損失関数(loss function) = Cross entropy function
  勾配法 = Adam（学習率 = 0.01）

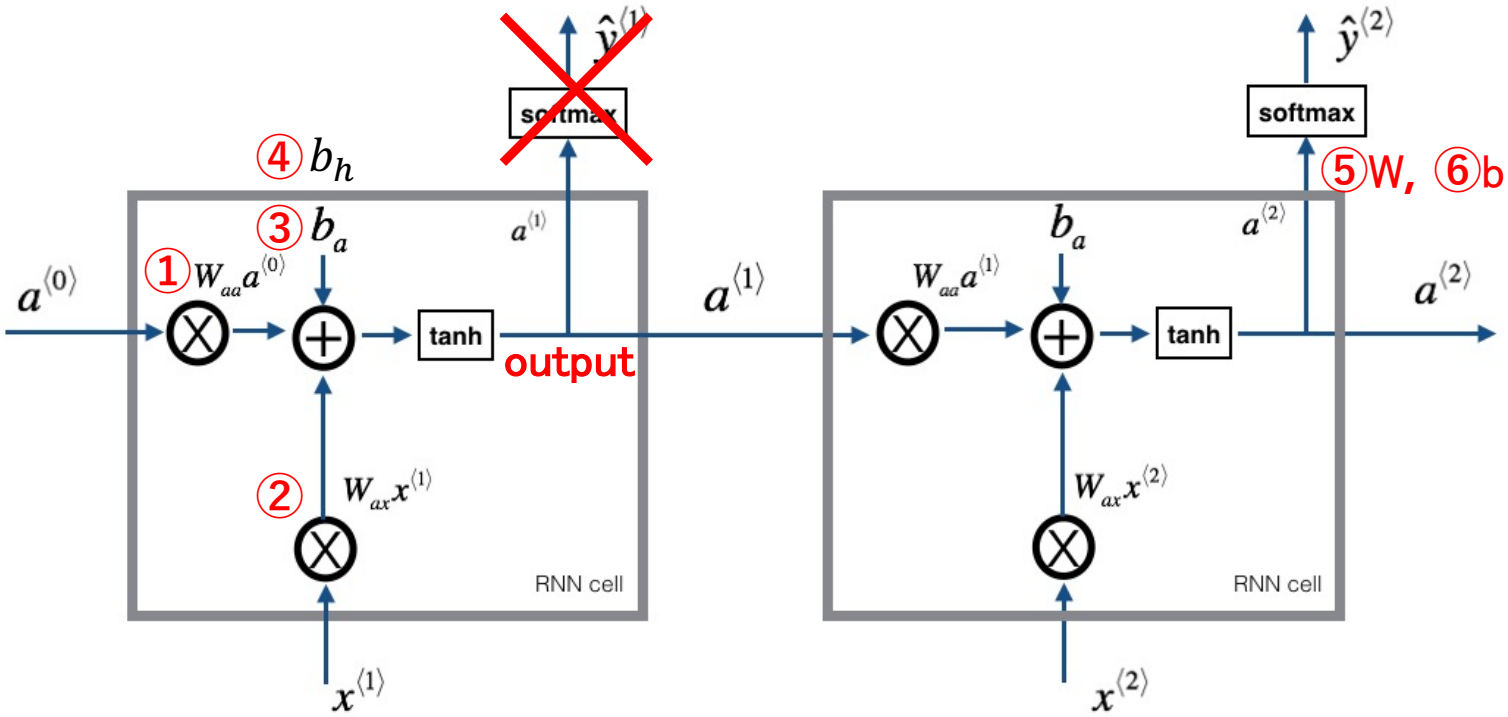
- 問題設定
  Two words are given → What comes next?
  ex) “I”, “like” → “cats”

  Input
  ”I”: [0., 0., 0., 0., 0., 1., 0., 0., 0.]
  “like”: [0., 0., 1., 0., 0., 0., 0., 0., 0.]

  ↓ RNN

  Target
  “cats”: [1., 0., 0., 0., 0., 0., 0., 0., 0.]

B. RNNの構築



”I”: [0., 0., 0., 0., 0., 1., 0., 0., 0.]

“like”: [0., 0., 1., 0., 0., 0., 0., 0., 0.]

- **Parameters**
  1,2,3,4: nn.RNNに内臓
  5,6: 別途宣言
  softmax: 別途宣言

- **Computations**

RNN内部

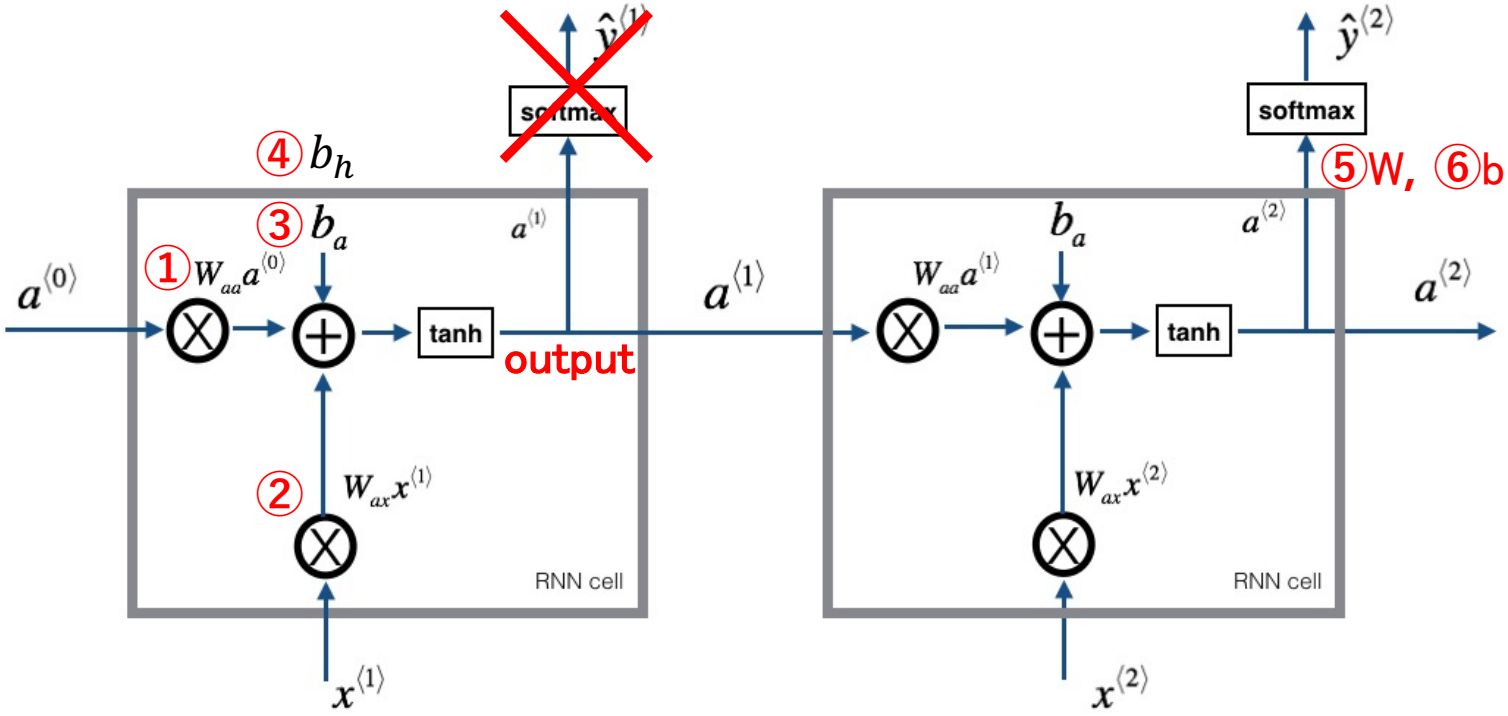
$$a^{\langle t\rangle} = tanh\left(\overset{②}{W_{ax}x^{\langle t\rangle}} + \overset{①}{W_{aa}a^{\langle t-1\rangle}} + \overset{③}{b_a} + \overset{④}{\boldsymbol{b_h}}\right)$$

softmax

$$\hat{y}^{\langle t\rangle} = soft\max\left(\overset{⑤}{W_{ya}a^{\langle t\rangle}} + \overset{⑥}{b_y}\right)$$

C. 学習

Target: [1., 0., 0., 0., 0., 0., 0., 0., 0.]
VS
Predict($y^{\langle 2 \rangle}$): [0., 1., 0., 0., 0., 0., 0., 0., 0.]



"I": [0., 0., 0., 0., 0., 1., 0., 0., 0.]

"like": [0., 0., 1., 0., 0., 0., 0., 0., 0.]

損失計算: Cross entropy function

↓

勾配計算: Adam optimizerを使用
loss.backward()

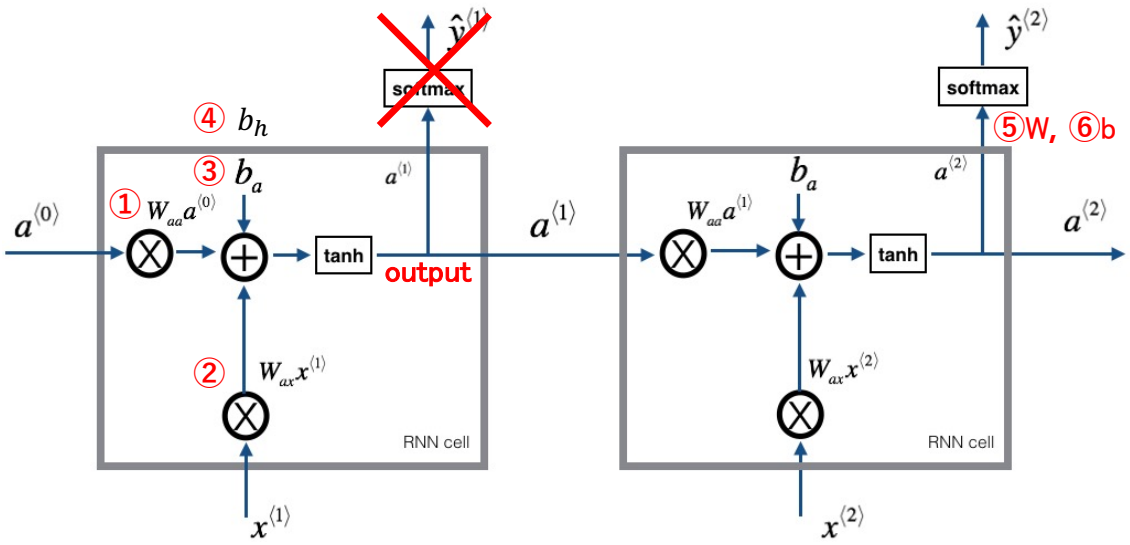
↓

パラメータ更新: optimizer.step()

6文章に対して500回繰り返す

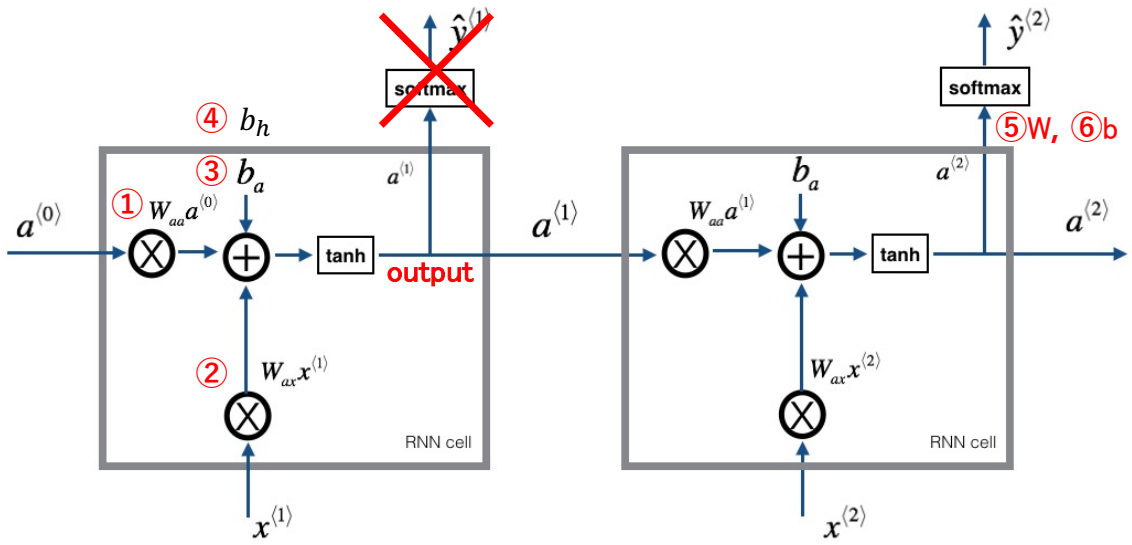
D. 検証

"dogs": [0., 0., 0., 0., 0., 0., 1., 0., 0.]



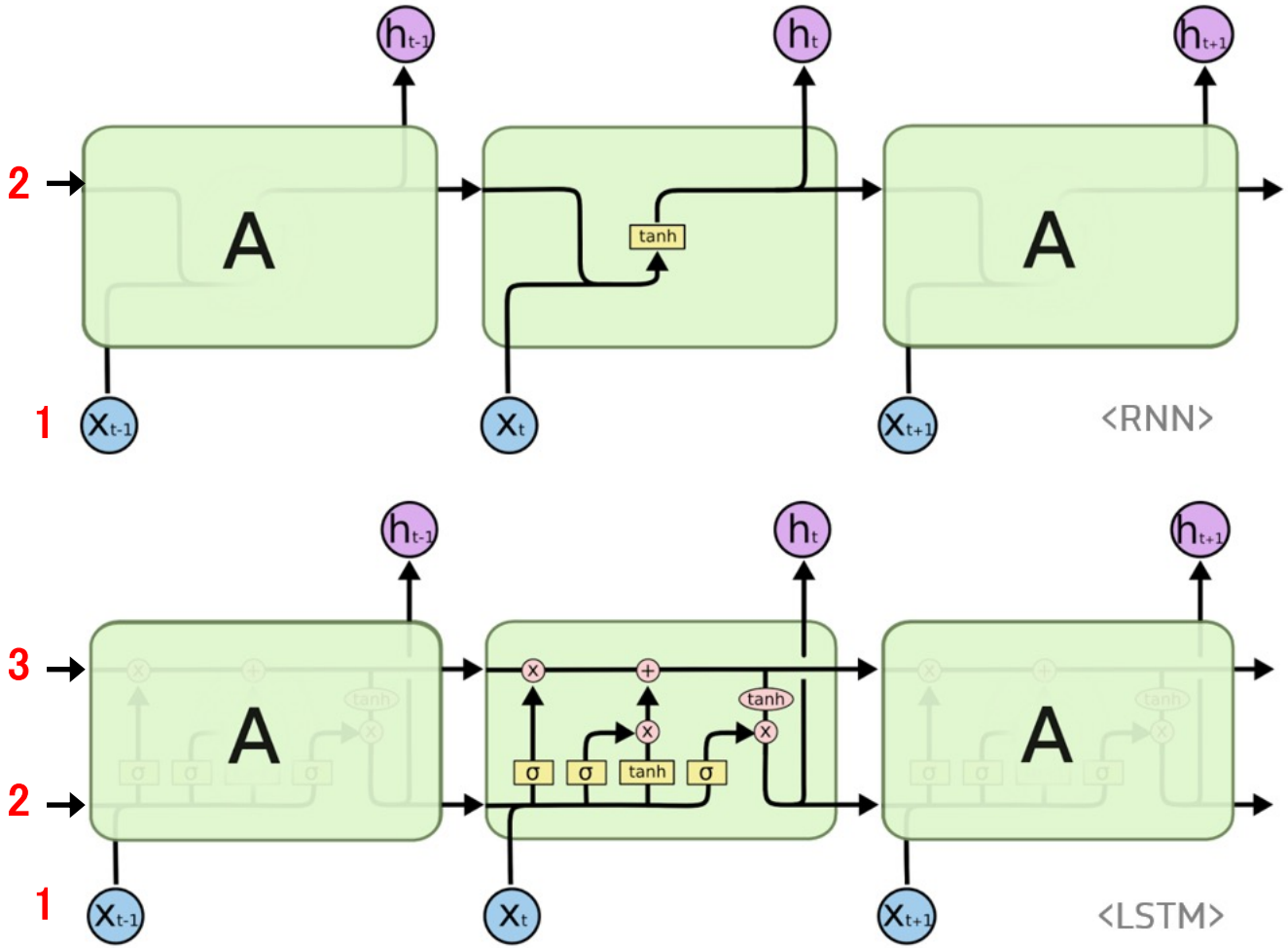
"I": [0., 0., 0., 0., 0., 1., 0., 0., 0.] "like": [0., 0., 1., 0., 0., 0., 0., 0., 0.]

"coffee": [0., 0., 0., 0., 0., 0., 0., 1., 0.]



"I": [0., 0., 0., 0., 0., 1., 0., 0., 0.] "love": [0., 0., 0., 1., 0., 0., 0., 0., 0.]

Structure of LSTM

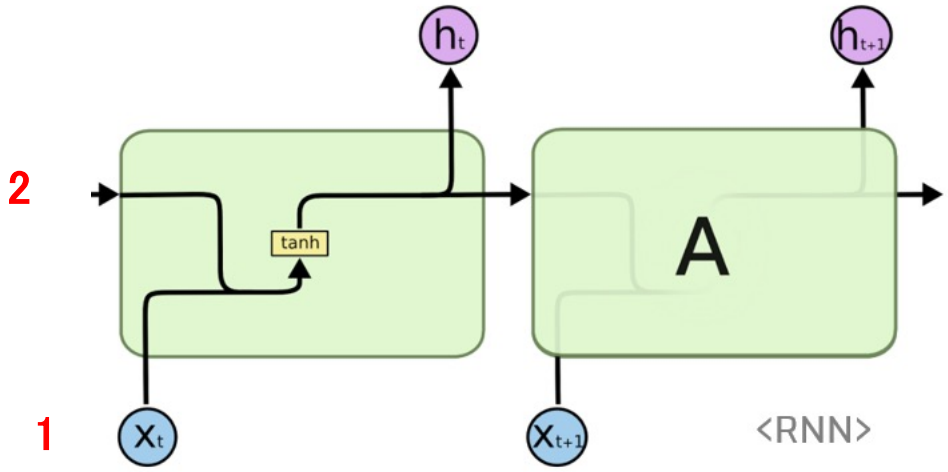


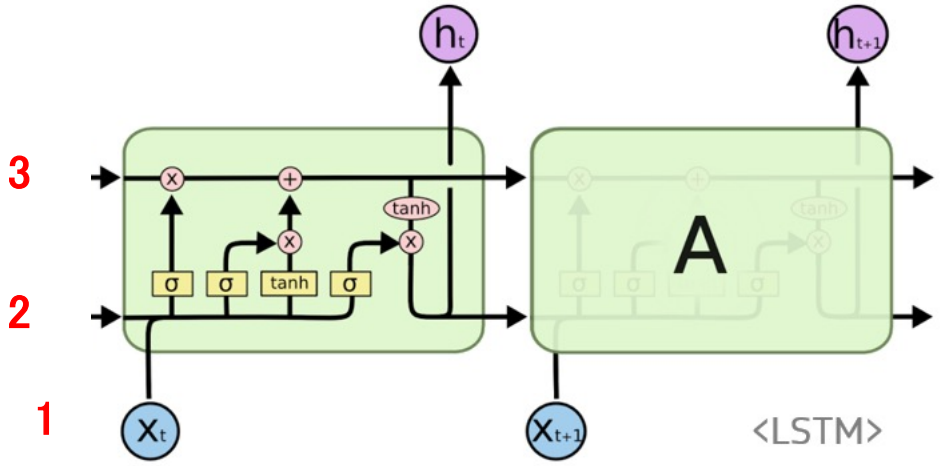
- 骨格は同じ
  input → Cells → output

- Cellの中が違う
  A. Input
     RNN: 2つ
     LSTM: 3つ

  B. 計算過程
     RNN: tanh
     LSTM: σs, tanh

Differences in the code (モデル定義)





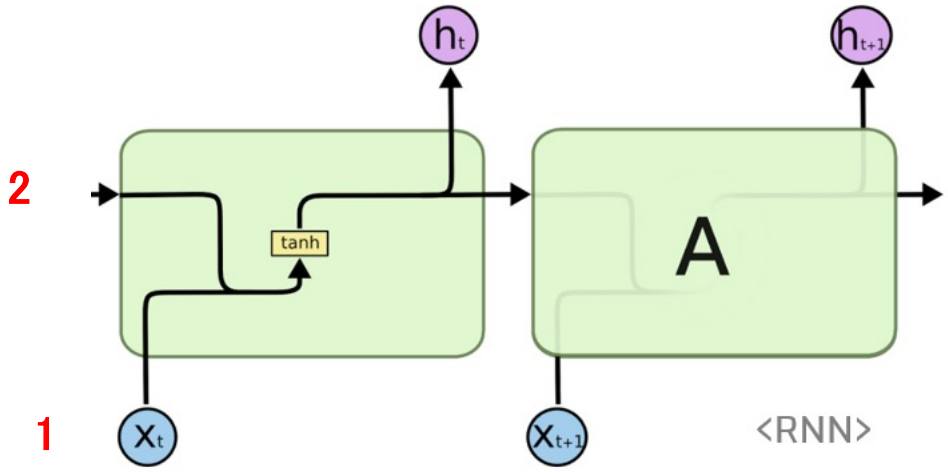

```python
class TextRNN(nn.Module):
    def __init__(self):
        super(TextRNN, self).__init__()

        self.rnn = nn.RNN(input_size=n_class, hidden_size=n_hidden, dropout=0.3)
        self.W = nn.Parameter(torch.randn([n_hidden, n_class]).type(dtype))
        self.b = nn.Parameter(torch.randn([n_class]).type(dtype))
        self.Softmax = nn.Softmax(dim=1)

    def forward(self, hidden, X):
        X = X.transpose(0, 1) →1
        outputs, hidden = self.rnn(X, hidden)
        outputs = outputs[-1]
        model = torch.mm(outputs, self.W) + self.b
        return model
```

→2

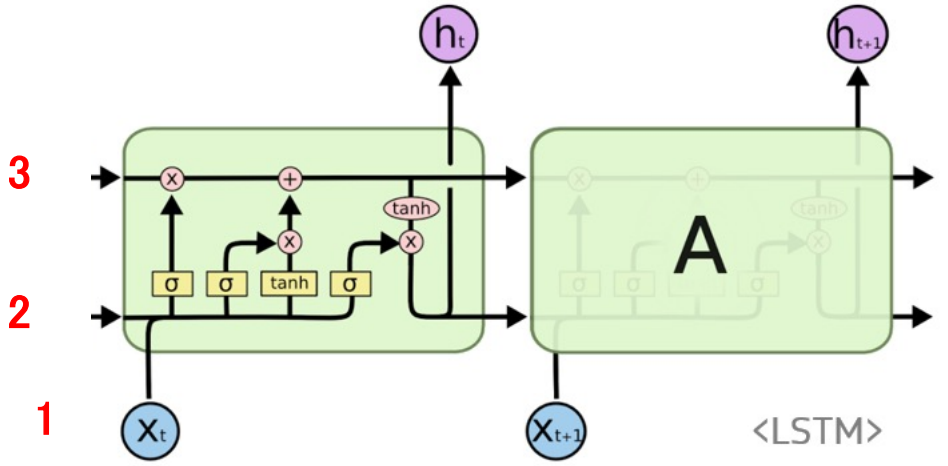
```python
class TextLSTM(nn.Module):
    def __init__(self):
        super(TextLSTM, self).__init__()

        self.lstm = nn.LSTM(input_size=n_class, hidden_size=n_hidden, dropout=0.3)
        self.W = nn.Parameter(torch.randn([n_hidden, n_class]).type(dtype))
        self.b = nn.Parameter(torch.randn([n_class]).type(dtype))
        self.Softmax = nn.Softmax(dim=1)

    def forward(self, hidden_and_cell, X):
        X = X.transpose(0, 1) →1
        outputs, hidden = self.lstm(X, hidden_and_cell)
        outputs = outputs[-1]
        model = torch.mm(outputs, self.W) + self.b
        return model
```

→2, 3

Differences in the code (学習)




```
model = TextRNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

for epoch in range(500):
    hidden = torch.zeros(1, batch_size, n_hidden, requires_grad=True)
    output = model(hidden, input_batch)
    loss = criterion(output, target_batch)

    if (epoch + 1) % 100 == 0:
        print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.6f}'.format(loss))

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```




```
model = TextLSTM()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

for epoch in range(500):
    hidden = torch.zeros(1, batch_size, n_hidden, requires_grad=True)
    cell = torch.zeros(1, batch_size, n_hidden, requires_grad=True)
    output = model((hidden, cell), input_batch)
    loss = criterion(output, target_batch)

    if (epoch + 1) % 100 == 0:
        print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.6f}'.format(loss))

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```