Lecture 4

Language Modeling and RNNs Part 2

석사과정 장명준

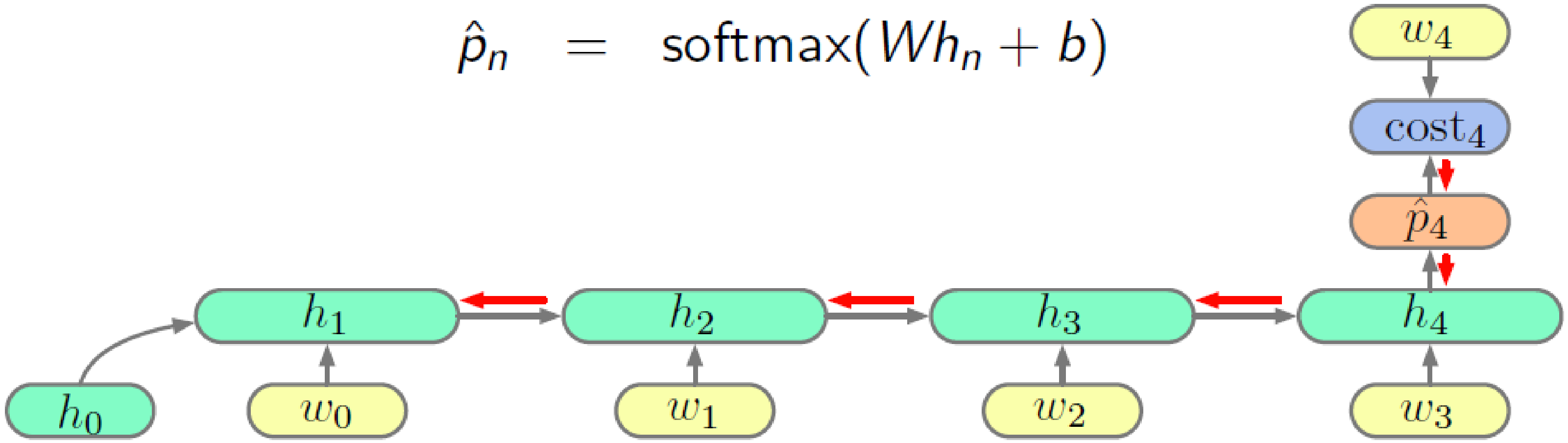
KOREA UNIVERSITY
Data Science & Business Analytics

CONTENTS

Vanishing/Exploding gradients

Vanishing/Exploding gradients

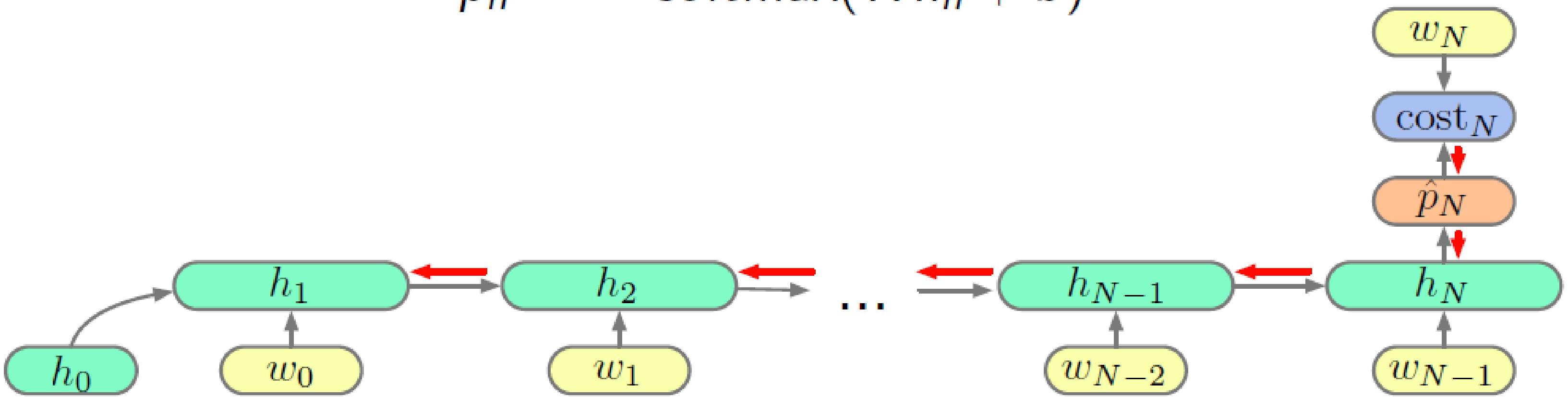
$$h_n = g(V[x_n; h_{n-1}] + c)$$
$$\hat{p}_n = \text{softmax}(Wh_n + b)$$



$$\frac{\partial \text{cost}_4}{\partial h_1} = \frac{\partial \text{cost}_4}{\partial \hat{p}_4} \frac{\partial \hat{p}_4}{\partial h_4} \frac{\partial h_4}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1}$$

Vanishing/Exploding gradients

$$\begin{aligned} h_n &= g(V[x_n; h_{n-1}] + c) \\ \hat{p}_n &= \mathrm{softmax}(W h_n + b) \end{aligned}$$



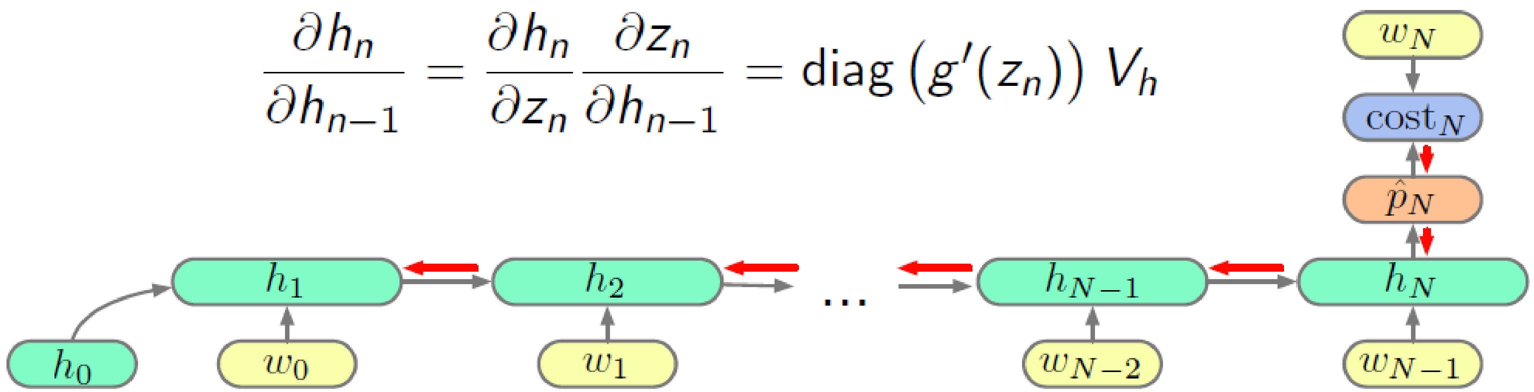
$$\frac{\partial \mathrm{cost}_N}{\partial h_1} = \frac{\partial \mathrm{cost}_N}{\partial \hat{p}_N} \frac{\partial \hat{p}_N}{\partial h_N} \left(\prod_{n \in \{N, \ldots, 2\}} \frac{\partial h_n}{\partial h_{n-1}} \right)$$

Vanishing/Exploding gradients

$$h_n = g(\underbrace{V_x x_n + V_h h_{n-1} + c}_{z_n}), \quad \frac{\partial \text{cost}_N}{\partial h_1} = \frac{\partial \text{cost}_N}{\partial \hat{p}_N} \frac{\partial \hat{p}_N}{\partial h_N} \left(\prod_{n \in \{N,\ldots,2\}} \frac{\partial h_n}{\partial z_n} \frac{\partial z_n}{\partial h_{n-1}} \right)$$

$$\frac{\partial h_n}{\partial z_n} = \mathrm{diag}\left(g'(z_n)\right) \qquad \frac{\partial z_n}{\partial h_{n-1}} = V_h$$

$$\frac{\partial h_n}{\partial h_{n-1}} = \frac{\partial h_n}{\partial z_n} \frac{\partial z_n}{\partial h_{n-1}} = \mathrm{diag}\left(g'(z_n)\right) V_h$$



Vanishing/Exploding gradients

$$\frac{\partial \text{cost}_N}{\partial h_1} = \frac{\partial \text{cost}_N}{\partial \hat{p}_N} \frac{\partial \hat{p}_N}{\partial h_N} \left(\prod_{n \in \{N,\ldots,2\}} \text{diag}\left(g'(z_n)\right) V_h \right)$$

$$\left\| \frac{\partial h_n}{\partial h_{n-1}} \right\| \leq \left\| diag\big(g'(z_n)\big) \right\| \|V_h\| \leq \beta_d \beta_h \qquad \blacktriangleright \qquad \left\| \frac{\partial h_n}{\partial h_1} \right\| = \left\| \prod_{j=2}^{n} \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta_d \beta_h)^{n-1}$$

If we use L2 norm

$$\left\| \frac{\partial h_n}{\partial h_1} \right\| = \left\| \prod_{j=2}^{n} \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq {\beta_d}^{n-1} \left(\sqrt{\lambda_{max}}\right)^{n-1}$$

Since $\|A\|_2 = \max_{\|x\|_2 = 1} \|Ax\|_2 = \sqrt{\lambda_{max}}$

If the largest eigenvalue of V_h is :
- 1, then gradient will propagate
- >1, exploding gradients
- <1, vanishing gradients

Vanishing/Exploding gradients

* Proof) $$\|A\|_2 = \max_{\|x\|_2=1} \|Ax\|_2 = \sqrt{\lambda_{max}}$$

Let's solve the problem below

$$Maximize\left(\|Ax\|_2{}^2 = x^T A^T Ax\right)$$
$$subject\ to\ x^T x = 1$$

By using the Lagrange multiplier method

$$\frac{\partial x^T A^T Ax}{\partial x} = \lambda \frac{\partial x^T x}{\partial x}$$

$$2A^T Ax = 2\lambda x$$

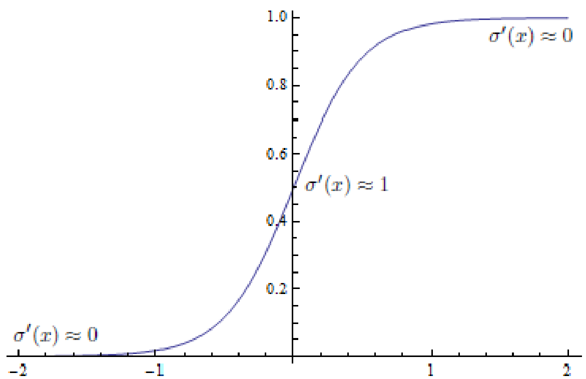
$$\therefore x^T A^T Ax = \lambda x^T x = \lambda$$

It follows that

$$\|A\|_2 = \max_{\|x\|_2=1} \|Ax\|_2 = \left(\max_{x^T x=1} x^T A^T Ax\right)^{1/2} = \sqrt{\lambda_{max}}$$

Vanishing/Exploding gradients

- Most of the times, the spectral radius (max eigenvalue) of V_h is small
- Also, many non-linearities($g(\cdot)$) can also shrink the gradient



Solutions:

1. **Second order optimizers**
2. **Careful initialization**
3. **Changing the network architecture**

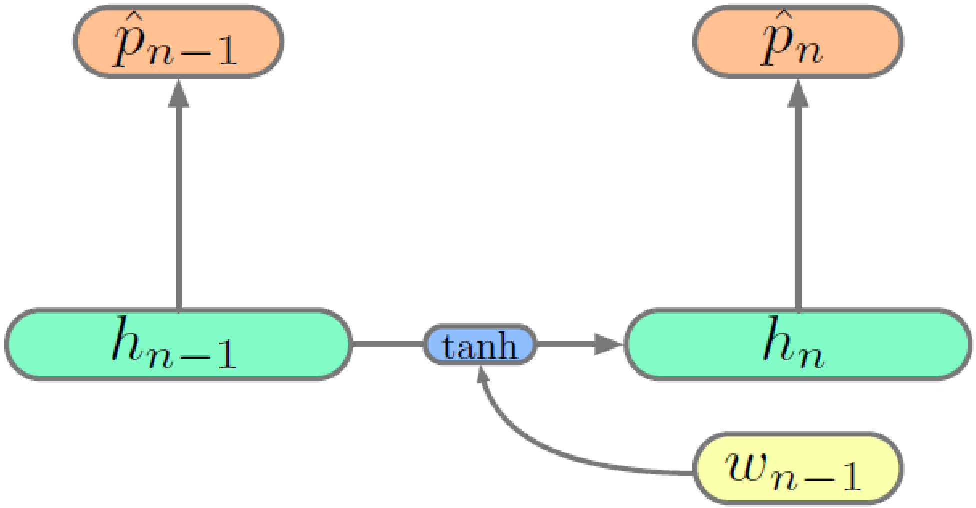
LSTM & Deep RNN

Long Short Term Memory

Simple Recurrent Neural Network

$$h_n = \tanh(V[w_{n-1}; h_{n-1}] + c)$$

Update by multiplying

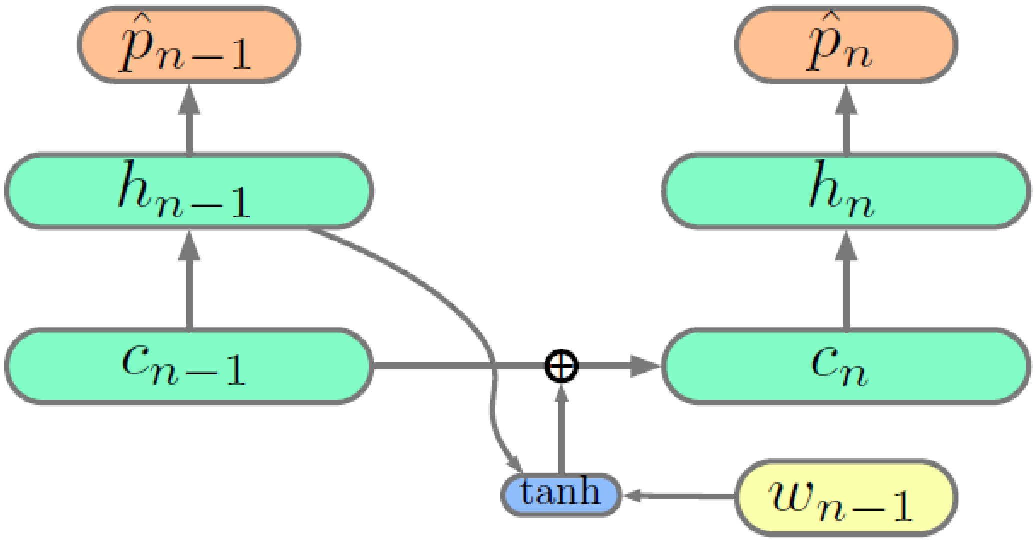


Long Short Term Memory

LSTM
- Introduce cell state : we can think this as memory

$$c_n = c_{n-1} + \tanh(V[w_{n-1}; h_{n-1}] + b_c)$$

Update by additive : gradient flow nicely



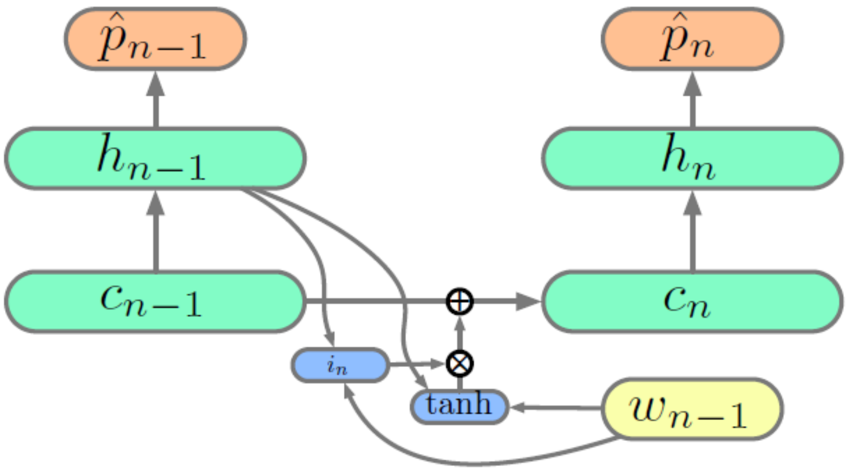
Need to balance : explode because of summation

Long Short Term Memory

LSTM

- Introduce gate for balancing : input gate

$$c_n = c_{n-1} + i_n \circ \tanh(V[w_{n-1}; h_{n-1}] + b_c)$$



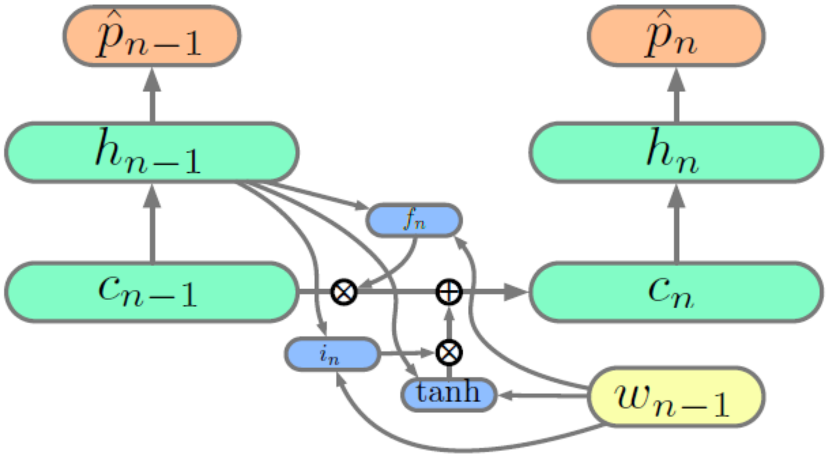
$$i_n = \sigma\left(W_i[w_{n-1}; h_{t-1}] + b_i\right).$$

→ continuous vector : different gate for every hidden unit

Long Short Term Memory

LSTM
- gate for c_n : forget gate

$$c_n = f_n \circ c_{n-1} + i_n \circ \tanh(V[w_{n-1}; h_{n-1}] + b_c)$$



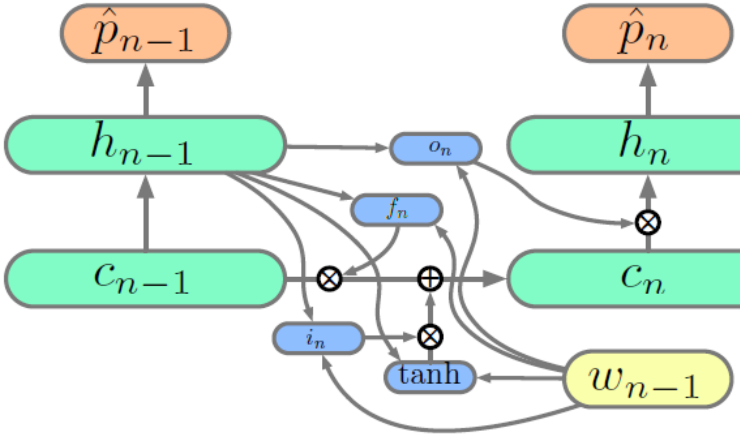
$$i_n = \sigma(W_i[w_{n-1}; h_{t-1}] + b_i),$$
$$f_n = \sigma(W_f[w_{n-1}; h_{t-1}] + b_f).$$

Long Short Term Memory

LSTM

- Why not put a gate on the output? : output gate

$$
\begin{aligned}
c_n &= f_n \circ c_{n-1} + i_n \circ \tanh(V[w_{n-1}; h_{n-1}] + b_c) \\
h_n &= o_n \circ \tanh\left(W_h c_n + b_h\right).
\end{aligned}
$$

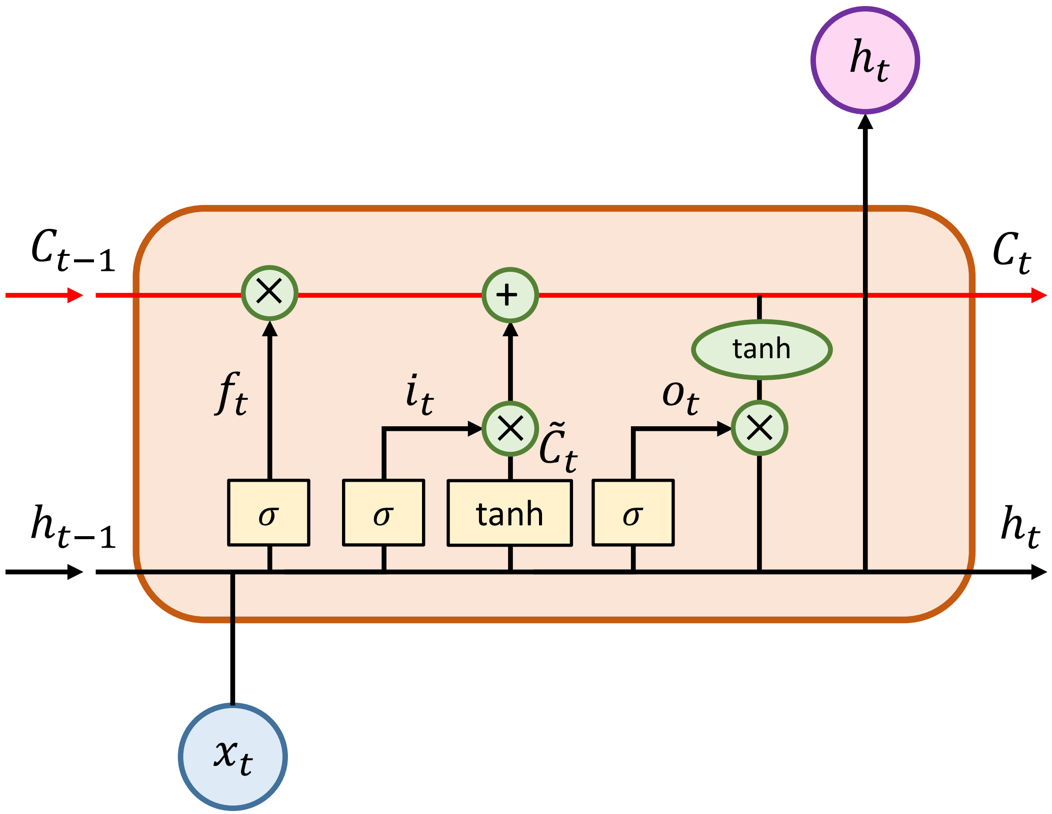


$$
\begin{aligned}
i_n &= \sigma\left(W_i[w_{n-1}; h_{t-1}] + b_i\right), \\
f_n &= \sigma\left(W_f[w_{n-1}; h_{t-1}] + b_f\right), \\
o_n &= \sigma\left(W_o[w_{n-1}; h_{t-1}] + b_o\right).
\end{aligned}
$$

- Sigmoid : good to use as gate (0 ~ 1)
- tanh : can be replaced to other non-linearity

Long Short Term Memory

LSTM



Long Short Term Memory

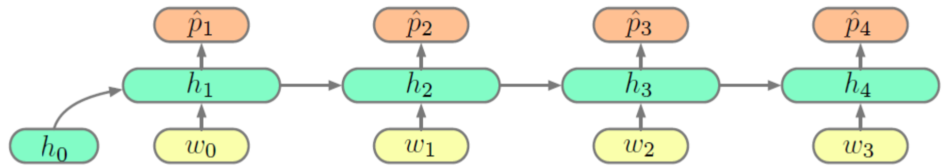
LSTM

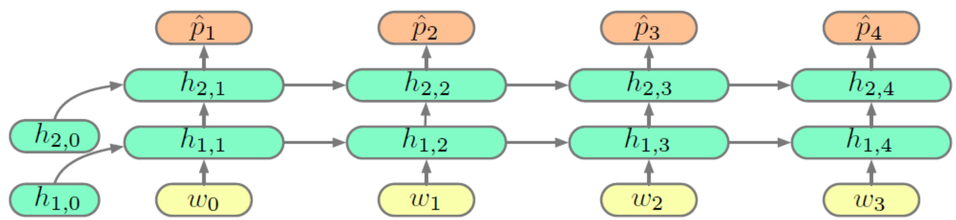
$$c_n = f_n \circ c_{n-1} + (1 - f_n) \circ \hat{c}_n$$

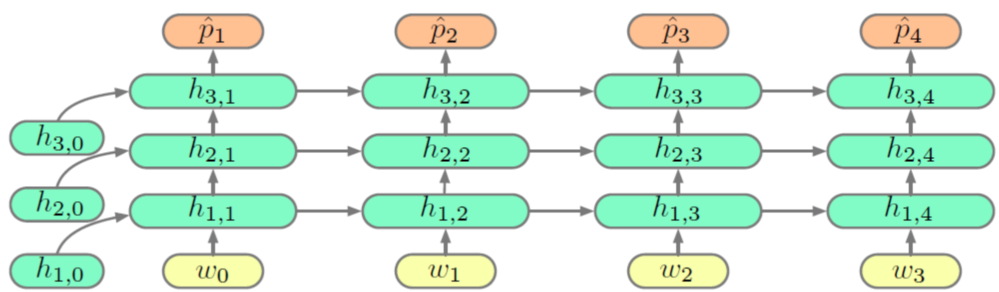
$$\hat{c}_n = \tanh(W_c[w_{n-1}; h_{t-1} + b_c])$$
$$h_n = o_n \circ \tanh(W_h c_n + b_h)$$
$$i_n = \sigma(W_i[w_{n-1}; h_{t-1} + b_i])$$
$$f_n = \sigma\left(W_f\left[w_{n-1}; h_{t-1} + b_f\right]\right)$$
$$o_n = \sigma(W_o[w_{n-1}; h_{t-1} + b_o])$$

Long Short Term Memory

Deep RNN



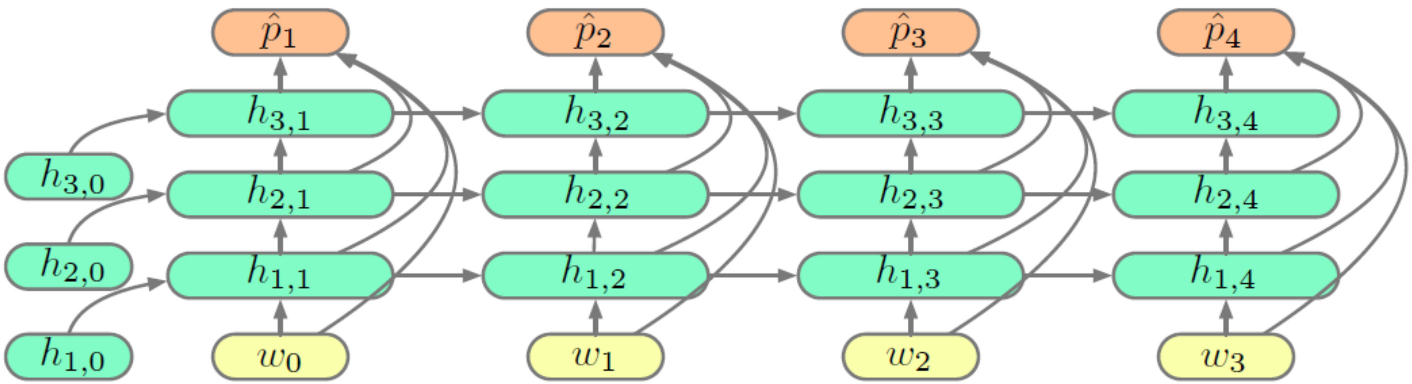




Deep RNN increases representational ability but also the memory

Long Short Term Memory

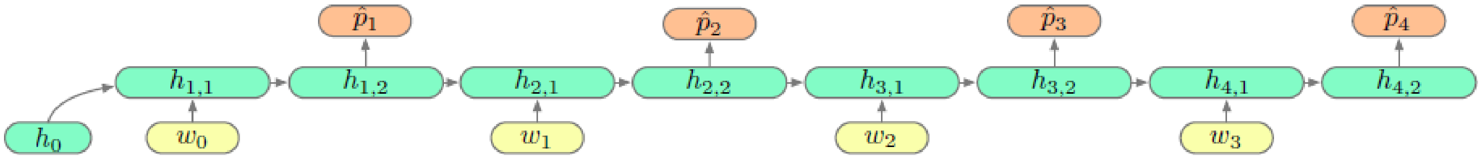
Deep RNN
- Skip connection : Deep network is hard to back propagate



Long Short Term Memory

Deep RNN

- Increase in the time dimension
- This improves the representational ability
- But not the memory capacity : we still have 1 recurrent sequence



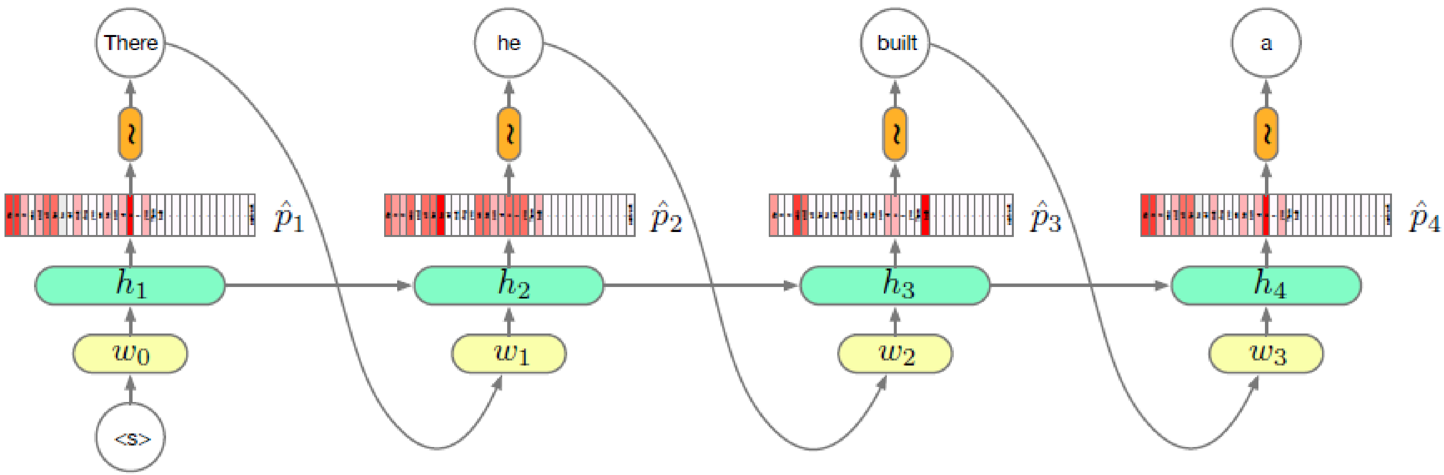
Scaling: Large Vocabularies

Scaling: Large Vocabularies

- Much of the computational cost is dominated by calculating:

$$\hat{p}_n = softmax(W_o h_n + b)$$

$$(|V| \times h) \times (h \times 1)$$



Solutions:

1. **Short-list :** use the RNN model for the most frequent words, and traditional N-gram model for the rest, easy to use but lose RNN's main advantage (generalization to rare events)

2. **Batch local short-lists**

Scaling: Large Vocabularies

Batch local short-lists

- Approximate full partition function from a subset of the vocabulary

Probability

$$p(y_t|y_{<t},x) = \frac{1}{Z}\exp(w_t^T \emptyset(y_{t-1}, z_t, c_t) + b_t)$$

$$z_t = g(y_{t-1}, z_{t-1}, c_t)$$
$$c_t = r(z_{t-1}, h_1, \dots, h_T)$$

$$Z = \sum_{k:y_k \in V} \exp\left(w_k^T \emptyset(y_{t-1}, z_t, c_t) + b_k\right) \text{ , } w_t^T: target\ word\ vector, b_t: target\ word\ bias$$

Calculate gradient

$$\nabla \log p(y_t|y_{<t},x) = \nabla\varepsilon(y_t) - \sum_{k:y_k\in V} p(y_k|y_{<t},x)\nabla\varepsilon(y_k) \Rightarrow E_P[\nabla\varepsilon(y)]$$

$$where\ \varepsilon(y_j) = w_j^T \emptyset(y_{t-1}, z_t, c_t) + b_j$$

$$where\ P\ denotes\ p(y|y_{<t},x)$$

Main idea

- Approximate expectation by sampling with a small number of samples

Given a predefined proposal distribution Q and a set V' of samples from Q

$$E_P[\nabla\varepsilon(y)] \approx \sum_{k:y_k\in V'} \frac{w_k}{\sum_{k':y_{k'}\in V'} w_{k'}} \nabla\varepsilon(y_k)$$

$$where\ w_k = \exp(\varepsilon(y_k) - \log Q(y_k))$$

On Using Very Large Target Vocabulary for Neural Machine Translation, Jean at al., ACL 2015

Scaling: Large Vocabularies

Batch local short-lists

In Practice

- Partition the training corpus and define a subset V' of the target vocabulary for each partition prior to training

Let V_i' refers to the subset of target words used for i^{th} partition and $Q_i(y_k) = \begin{cases} \frac{1}{|V_i'|} & if\ y_k \in V_i' \\ 0\ otherwise \end{cases}$

From equation

$$E_P[\nabla \varepsilon(y)] \approx \sum_{k:y_k \in V'} \frac{w_k}{\sum_{k':y_{k'} \in V'} w_{k'}} \nabla \varepsilon(y_k)\ \ where\ w_k = \exp(\varepsilon(y_k) - \log Q(y_k))$$

$$p(y_t|y_{<t}, x) \approx \frac{w_k}{\sum_{k':y_{k'} \in V'} w_{k'}} = \frac{\exp(\varepsilon(y_t) - \log Q(y_t))}{\sum_{k':y_{k'} \in V'} \exp(\varepsilon(y_{k'}) - \log Q(y_{k'}))}$$

$$= \frac{\exp(\varepsilon(y_t) + |V'|)}{\sum_{k':y_{k'} \in V'} \exp(\varepsilon(y_{k'}) + |V'|)}$$

$$= \frac{\exp(\varepsilon(y_t) exp(|V'|)}{\sum_{k':y_{k'} \in V'} \exp(\varepsilon(y_{k'}) exp(|V'|)}$$

$$= \frac{\boldsymbol{exp}(\boldsymbol{w}_t^T \emptyset(\boldsymbol{y}_{t-1}, \boldsymbol{z}_t, \boldsymbol{c}_t) + \boldsymbol{b}_t)}{\sum_{k':y_{k'} \in V'} \boldsymbol{exp}(\boldsymbol{w}_{k'}^T \emptyset(\boldsymbol{y}_{t-1}, \boldsymbol{z}_t, \boldsymbol{c}_t) + \boldsymbol{b}_{k'})}$$

On Using Very Large Target Vocabulary for Neural Machine Translation, Jean at al., ACL 2015

Scaling: Large Vocabularies

- Much of the computational cost is dominated by calculating:

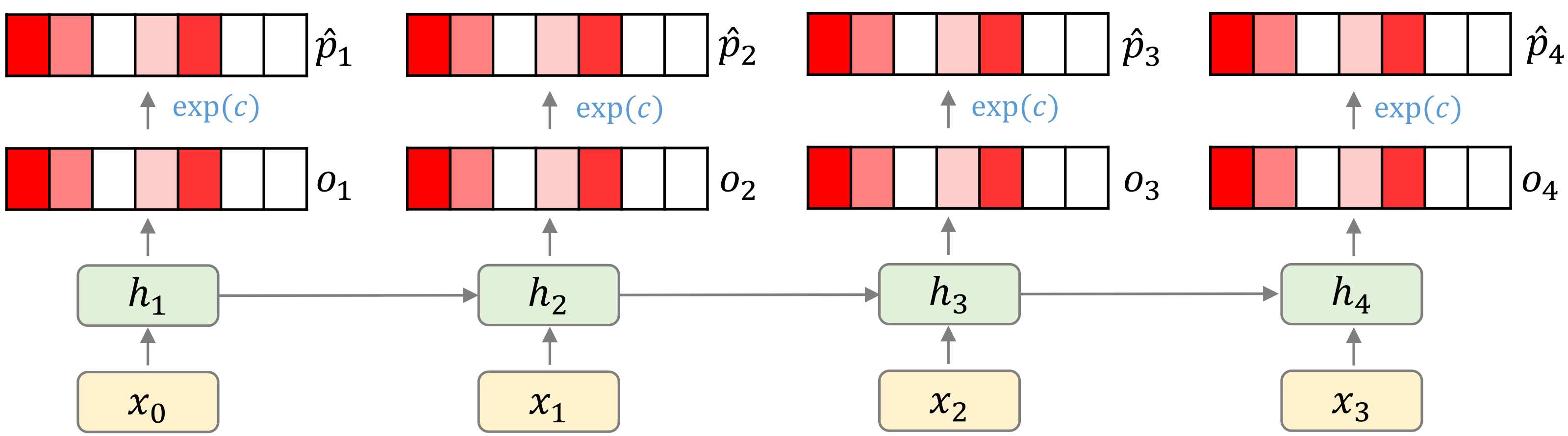
$$\hat{p}_n = softmax(W_o h_n + b)$$

Solutions:

3. Approximate the gradient/ change the objective :

consider maximizing likelihood by making the log partition function an independent parameter c

$$\hat{p}_n = exp(W_o h_n + b) \times \exp(c)$$

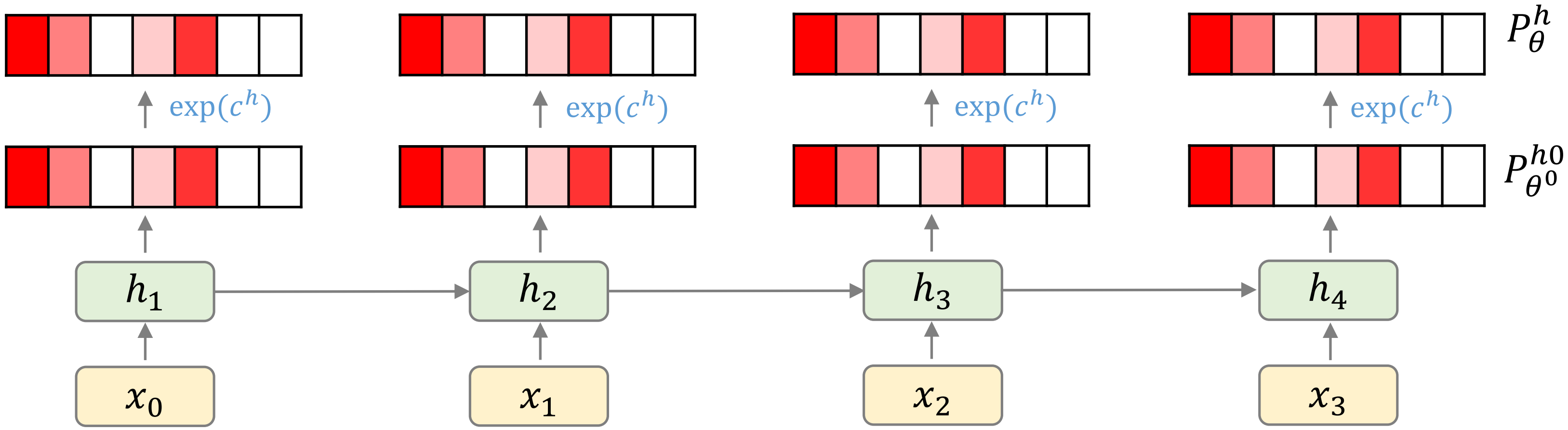


Leads to an ill defined objective

Scaling: Large Vocabularies

Noise-contrastive estimation

Reduce problem of softmax to that of binary classification discriminating of words that occur after a particular context h



$$P_\theta^h(w) = P_{\theta^0}^{h0}(w)\exp\left(c^h\right), \theta = \{\theta^0, c^h\}$$

$P_{\theta^0}^{h0}(w)$: *unnormalized distribution*, c^h: *learned parameter*

$P_d^h(w)$: *empirical distribution of words that occur after a particular context h*
$P_n(w)$: *noise distribution (unigram)*

Object

$$P_\theta^h(w) \approx P_d^h(w)$$

A fast and simple algorithm for training neural probabilistic language models, Mnih and Teh, 2012

Scaling: Large Vocabularies

Noise-contrastive estimation

Assume that noise samples are k times more frequent than data samples

$$p^h(d,w) = \begin{cases} \frac{k}{1+k} \times P_n(w)\ if\ d=0 \\ \frac{1}{1+k} \times P_d^h(w)\ if\ d=1 \end{cases}$$

$$P^h(D=0|w) = \frac{kP_n(w)}{P_d^h(w) + kP_n(w)}$$
$$P^h(D=1|w) = \frac{P_d^h(w)}{P_d^h(w) + kP_n(w)}$$

$$\mathit{Object}$$
$$P_\theta^h(w) \approx P_d^h(w)$$

$$P^h(D=0|w) = \frac{kP_n(w)}{P_d^h(w) + kP_n(w)}$$
$$P^h(D=1|w) = \frac{P_d^h(w)}{P_d^h(w) + kP_n(w)}$$

$$P^h(D=0|w,\theta) = \frac{kP_n(w)}{P_\theta^h(w) + kP_n(w)}$$
$$P^h(D=1|w,\theta) = \frac{P_\theta^h(w)}{P_\theta^h(w) + kP_n(w)}$$

A fast and simple algorithm for training neural probabilistic language models, Mnih and Teh, 2012

Scaling: Large Vocabularies

Noise-contrastive estimation

$$P^h(D=0|w,\theta) = \frac{kP_n(w)}{P_\theta^h(w) + kP_n(w)}$$
$$P^h(D=1|w,\theta) = \frac{P_\theta^h(w)}{P_\theta^h(w) + kP_n(w)}$$

Maximize the expectation of $\log P^h(D|w,\theta)$ under the mixture of the data and noise samples

$$J^h(\theta) = E_{P_d^h}\left[\log \frac{P_\theta^h(w)}{P_\theta^h(w) + kP_n(w)}\right] + kE_{P_n}\left[\frac{kP_n(w)}{P_\theta^h(w) + kP_n(w)}\right]$$

With the gradient

$$\frac{\partial}{\partial\theta}J^h(\theta) = E_{P_d^h}\left[\frac{kP_n(w)}{P_\theta^h(w) + kP_n(w)}\frac{\partial}{\partial\theta}\log P_\theta^h(w)\right] + kE_{P_n}\left[\frac{P_\theta^h(w)}{P_\theta^h(w) + kP_n(w)}\ \frac{\partial}{\partial\theta}\log P_\theta^h(w)\right]$$
$$= \sum_w \frac{kP_n(w)}{P_\theta^h(w) + kP_n(w)} \times (P_d^h(w) - P_\theta^h(w))\frac{\partial}{\partial\theta}\log P_\theta^h(w)$$

And that as $k \to \infty$

$$\frac{\partial}{\partial\theta}J^h(\theta) = \sum_w (P_d^h(w) - P_\theta^h(w))\frac{\partial}{\partial\theta}\log P_\theta^h(w)$$

A fast and simple algorithm for training neural probabilistic language models, Mnih and Teh, 2012

Scaling: Large Vocabularies

Noise-contrastive estimation

In practice, given a word w observed in context h, we compute its contribution to the gradient by generating k noise samples $x_1, \dots, x_k$ and using the formula

$$\frac{\partial}{\partial\theta} J^h(\theta) = E_{P_d^h}\left[\frac{kP_n(w)}{P_\theta^h(w) + kP_n(w)} \frac{\partial}{\partial\theta} \log P_\theta^h(w)\right] + kE_{P_n}\left[\frac{P_\theta^h(w)}{P_\theta^h(w) + kP_n(w)} \frac{\partial}{\partial\theta} \log P_\theta^h(w)\right]$$
$$= \frac{kP_n(w)}{P_\theta^h(w) + kP_n(w)} \frac{\partial}{\partial\theta} \log P_\theta^h(w) + k\sum_{i=1}^{k} \frac{1}{k} \frac{P_\theta^h(x_i)}{P_\theta^h(x_i) + kP_n(x_i)} \frac{\partial}{\partial\theta} \log P_\theta^h(x_i)$$

Since distributions for different contexts share parameters, we cannot learn these distributions independently

$$J(\theta) = \sum_h P(h) J^h(\theta)$$

$P(h)$: *empirical context porobabilities*

A fast and simple algorithm for training neural probabilistic language models, Mnih and Teh, 2012

Scaling: Large Vocabularies

- Much of the computational cost is dominated by calculating:

$$\hat{p}_n = softmax(W_o h_n + b)$$

Solutions:

4. Factorize the output vocabulary :

-Make assumption about distribution

-There are lots of algorithms for grouping vocabularies into classes

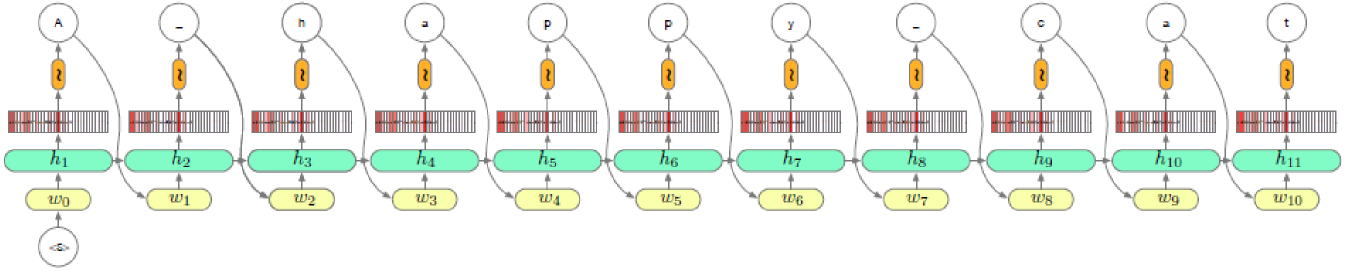
$$p\left(w_n \middle| \hat{p}_n^{class}, \hat{p}_n^{word}\right) = p(class(w_n)|\hat{p}_n^{class}) \times p(w_n|class(w_n), \hat{p}_n^{word})$$

-Assuming balanced classes, this gives a $\sqrt{V}$ speed up

Ex) V = 1,000,000, classes = 1,000

Sub-Word Level Language Modeling

- Result in much smaller softmax and no unknown words
- Model can capture subword structure and morphology : disunited, disinherited, disinterested
- Longer sequences, hard to back propagate
- A lot of structure in languages, bottom low is word and we want to learn correlation between words
- Characters cannot learn correlation between the words



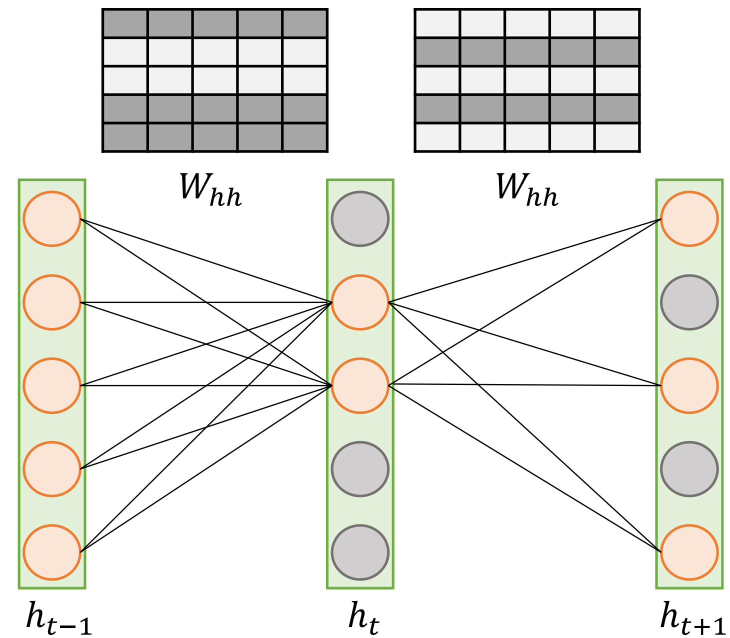
Regularization

Regularization: Drop out

- Dropout is ineffective when applied to recurrent connections
- If we repeat couple of times, in expectation, every hidden unit will be dropped out

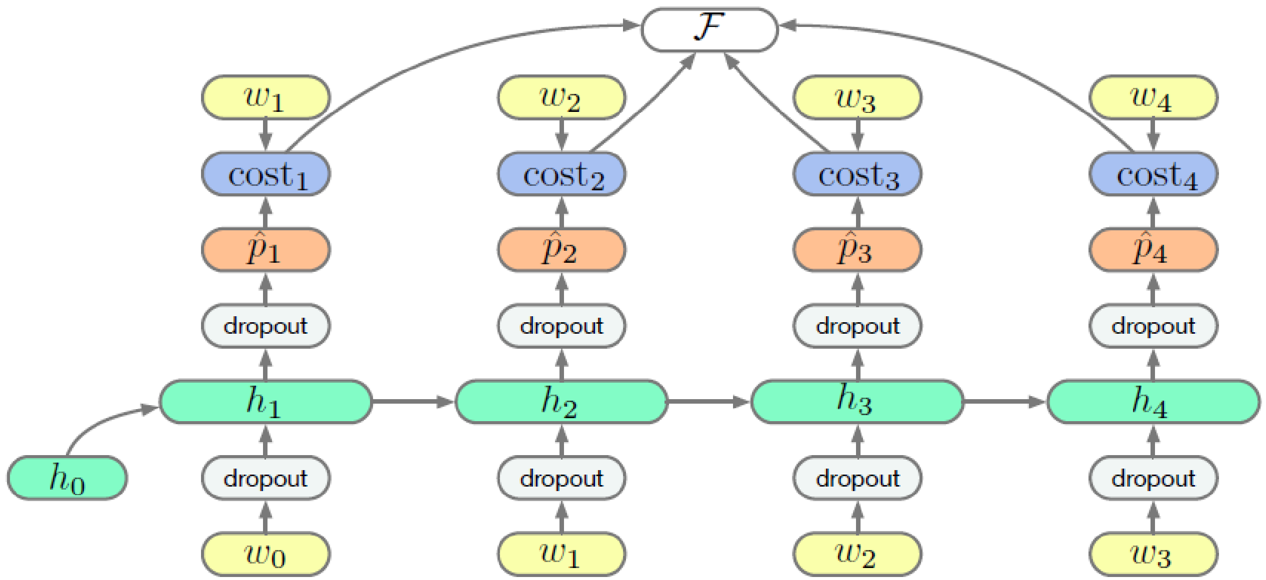
$$\underset{h\times 1}{h_t} = \underset{h\times h}{W_{hh}}\,\underset{h\times 1}{h_{t-1}}$$

$$\begin{bmatrix} h_{t,1} \\ h_{t,2} \\ h_{t,3} \\ \cdots \\ h_{t,h} \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,h} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,h} \\ \vdots & \cdots & \ddots & \vdots \\ w_{h,1} & w_{h,2} & \cdots & w_{h,h} \end{bmatrix} \begin{bmatrix} h_{t-1,1} \\ h_{t-1,2} \\ h_{t-1,3} \\ \cdots \\ h_{t-1,h} \end{bmatrix}$$



Regularization: Drop out

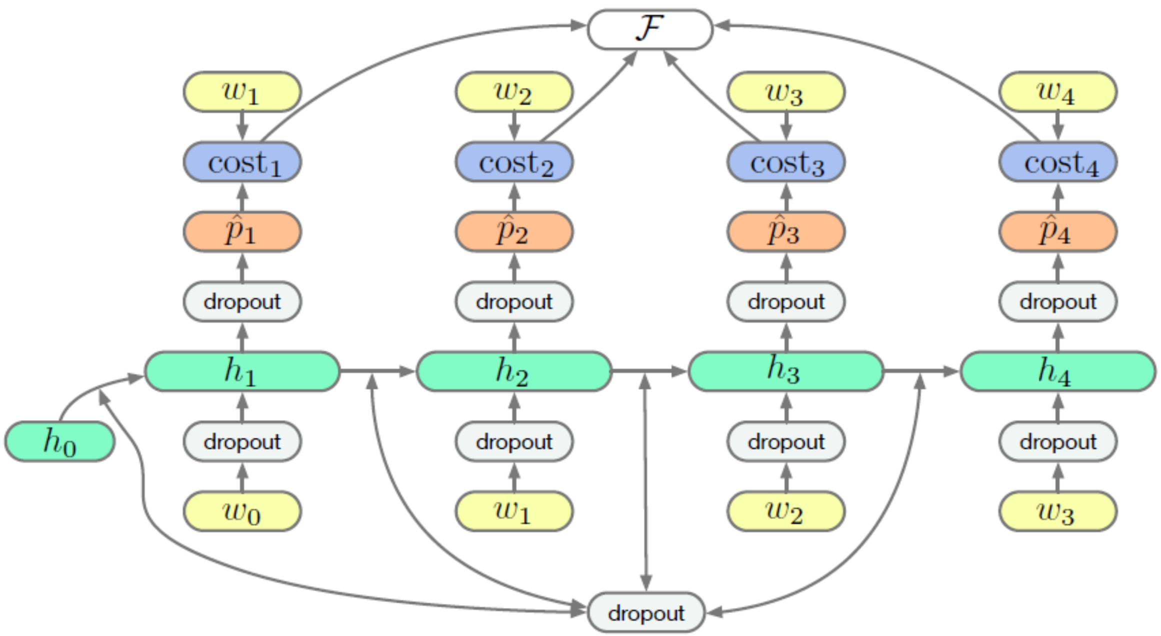
- Dropout is ineffective when applied to recurrent connections
- If we repeat couple of times, in expectation, every hidden unit will be dropped out
- Instead, put drop out everywhere else in the network(Input, Output layer)



- But recurrent connections are not regularized

Regularization: Drop out

- Rather than sampling a random mask for every connection in drop out, ***sample one random mask*** for the recurrent connection and use same mask of every time step




Q&A