یادگیری عمیق

مدرس: محمدرضا محمدی

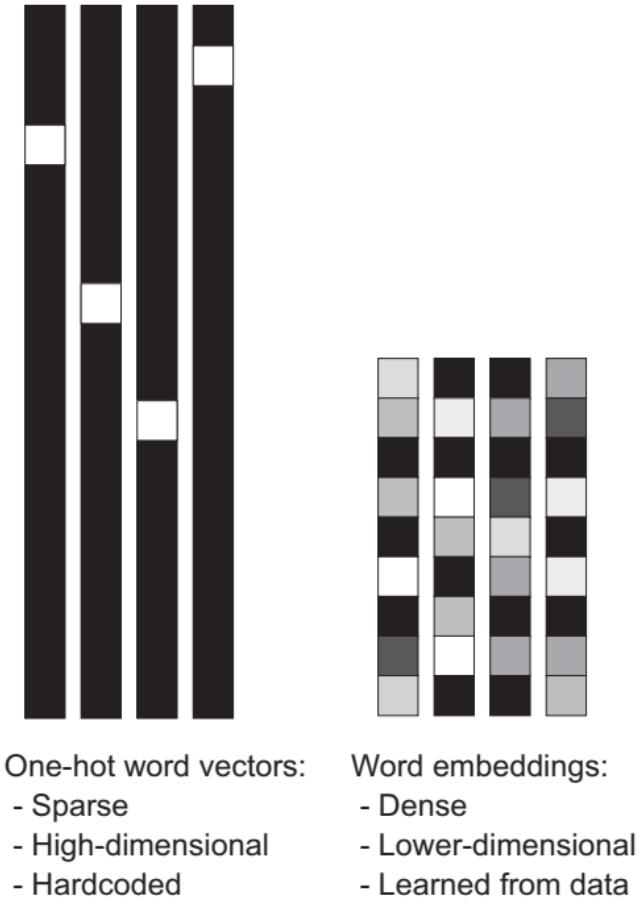
بهار ۱۴۰۲

شبکه‌های عصبی بازگشتی

Recurrent Neural Networks

جانمایی کلمات (Word embedding)

- جانمایی کلمات اطلاعات بیشتر را در ابعاد بسیار کمتری قرار می‌دهد
- این بردارها را می‌توان با استفاده از حجم زیادی از متن پیش‌آموزش داد و در مجموعه داده‌های کوچک از آنها استفاده کرد
- انتظار می‌رود فاصله هندسی بین هر دو بردار کلمه با فاصله معنایی بین کلمات مرتبط باشد
- همچنین، انتظار می‌رود جهت‌های مختلف در فضای آموخته شده معنادار باشند



One-hot word vectors:
- Sparse
- High-dimensional
- Hardcoded

Word embeddings:
- Dense
- Lower-dimensional
- Learned from data

آموزش جانمایی کلمات

- یک فضای مناسب تا حد زیادی به مسئله مورد نظر بستگی دارد
- فضای مناسب برای دسته‌بندی نقد فیلم ممکن است متفاوت از فضای مناسب برای دسته‌بندی اسناد حقوقی باشد، زیرا اهمیت برخی روابط معنایی متفاوت است
- آموزش یک فضای جانمایی جدید برای هر مسئله جدید منطقی است

CLASS torch.nn.Embedding(num_embeddings, embedding_dim, padding_idx=*None*, max_norm=*None*, norm_type=*2.0*, scale_grad_by_freq=*False*, sparse=*False*, _weight=*None*, device=*None*, dtype=*None*)

A simple lookup table that stores embeddings of a fixed dictionary and size.

This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.

Parameters

- **num_embeddings** (*int*) – size of the dictionary of embeddings
- **embedding_dim** (*int*) – the size of each embedding vector
- **padding_idx** (*int, optional*) – If specified, the entries at padding_idx do not contribute to the gradient; therefore, the embedding vector at padding_idx is not updated during training, i.e. it remains as a fixed “pad”. For a newly constructed Embedding, the embedding vector at padding_idx will default to all zeros, but can be updated to another value to be used as the padding vector.
- **max_norm** (*float, optional*) – If given, each embedding vector with norm larger than max_norm is renormalized to have norm max_norm.
- **norm_type** (*float, optional*) – The p of the p-norm to compute for the max_norm option. Default 2.
- **scale_grad_by_freq** (*boolean, optional*) – If given, this will scale gradients by the inverse of frequency of the words in the mini-batch. Default False.
- **sparse** (*bool, optional*) – If True, gradient w.r.t. weight matrix will be a sparse tensor. See Notes for more details regarding sparse gradients.

Variables

~Embedding.weight (*Tensor*) – the learnable weights of the module of shape (num_embeddings, embedding_dim) initialized from N(0,1)

Shape:

- Input: (∗), IntTensor or LongTensor of arbitrary shape containing the indices to extract
- Output: (∗,*H*), where * is the input shape and *H*=embedding_dim

لایه Embedding

- لغت‌نامه‌ای که اندیس صحیح مربوط به هر توکن را به یک بردار معنایی نگاشت می‌کند
- هنگامی که یک لایه Embedding را می‌سازیم، وزن‌های آن در ابتدا تصادفی است
- در طول آموزش، این بردارهای کلمات به تدریج از طریق پس‌انتشار تنظیم می‌شوند

```
# an Embedding module containing
# 10 tensors of size 3
embedding = nn.Embedding(10, 3)
# a batch of 2 samples of 4 indices each
input = torch.LongTensor([[1,2,4,5],[4,3,2,9]])
emb = embedding(input)
```

```
tensor([[[ 1.1838, -1.5785,  0.9585],
         [-0.7131, -1.3546, -0.3694],
         [-0.9007, -0.1768,  1.5017],
         [ 0.0806, -0.1731, -0.6342]],

        [[-0.9007, -0.1768,  1.5017],
         [-0.0437, -0.1967, -0.6747],
         [-0.7131, -1.3546, -0.3694],
         [-1.5153,  0.1286,  0.6137]]],
       grad_fn=<EmbeddingBackward0>)
```

جانمایی کلمات پیش‌آموخته

- مشابه با مفهوم شبکه‌های کانولوشنی پیش‌آموخته
  - زمانیکه داده‌های کافی برای یادگیری ویژگی‌های قدرتمند نداریم، اما انتظار داریم ویژگی‌هایی که به آن نیاز داریم نسبتاً عمومی باشند
- بجای یادگیری جانمایی کلمات به طور مشترک با مسئله مورد نظر، می‌توان بردارهای جانمایی آموخته شده برای حل یک مسئله دیگر را بارگذاری کنیم
- معمولاً با استفاده از اطلاعات آماری وقوع کلمات محاسبه می‌شود
  - با یا بدون استفاده از شبکه‌های عصبی
- دو مورد از معروف‌ترین و موفق‌ترین آنها Word2vec و GloVe هستند

شبکه‌های عصبی بازگشتی

- در مدل‌های Autoregressive دیدیم که به دنبال $P(x_t \mid x_{t-1}, \dots, x_1)$ هستیم
- در برخی مسائل می‌توان از تقریب زیر استفاده کرد:

$$P(x_t \mid x_{t-1}, \dots, x_1) \approx P(x_t \mid h_{t-1})$$

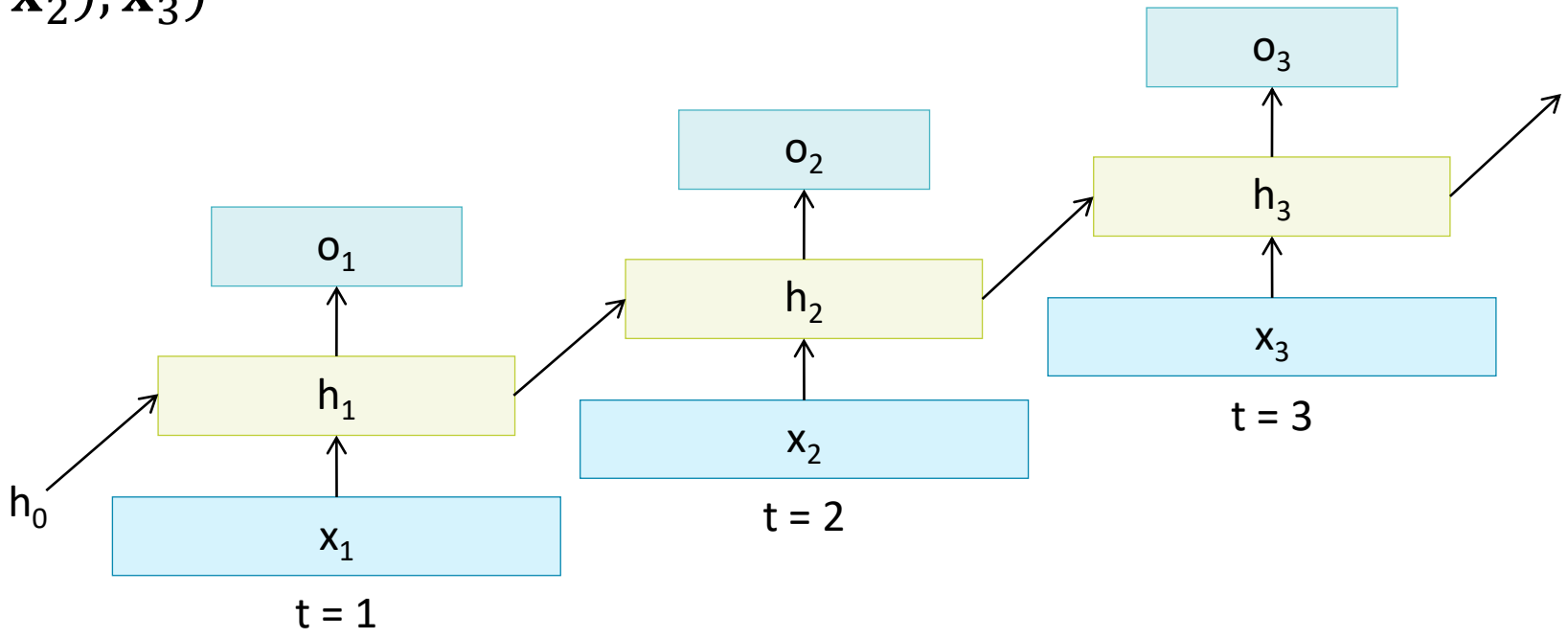
- که به h_{t-1} حالت پنهان (hidden state)یا متغیر پنهان (hidden variable) گفته می‌شود
  - اطلاعات دنباله از x_1 تا x_{t-1} در h_{t-1} ذخیره می‌شود
- حالت پنهان در هر گام با استفاده از مقدار x_t به‌روزرسانی می‌شود

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

- با یک تابع f به اندازه کافی قدرتمند، مدل متغیر پنهان تقریب خوبی خواهد بود
  - چنین تابعی ممکن است از لحاظ حافظه و محاسبات گران باشد

شبکه‌های عصبی بازگشتی

$$\mathbf{h}_3 = f_W(\mathbf{h}_2, \mathbf{x}_3)$$
$$= f_W(f_W(\mathbf{h}_1, \mathbf{x}_2), \mathbf{x}_3)$$
$$= f_W(f_W(f_W(\mathbf{h}_0, \mathbf{x}_1), \mathbf{x}_2), \mathbf{x}_3)$$
$$= g^{(3)}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$$



شبکه‌های عصبی بازگشتی

- یک دنباله از بردارهای $\boldsymbol{x}$ می‌تواند با استفاده از یک رابطه بازگشتی در هر زمان پردازش شود
- در این مدل، یک تابع یکسان با مجموعه پارامترهای یکسان در زمان‌های مختلف استفاده می‌شود

some function with parameters W — input vector at time t

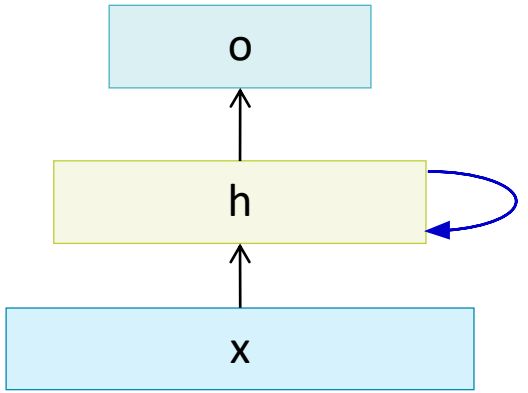
$$\mathbf{h}_t = f_W(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

new state — old state

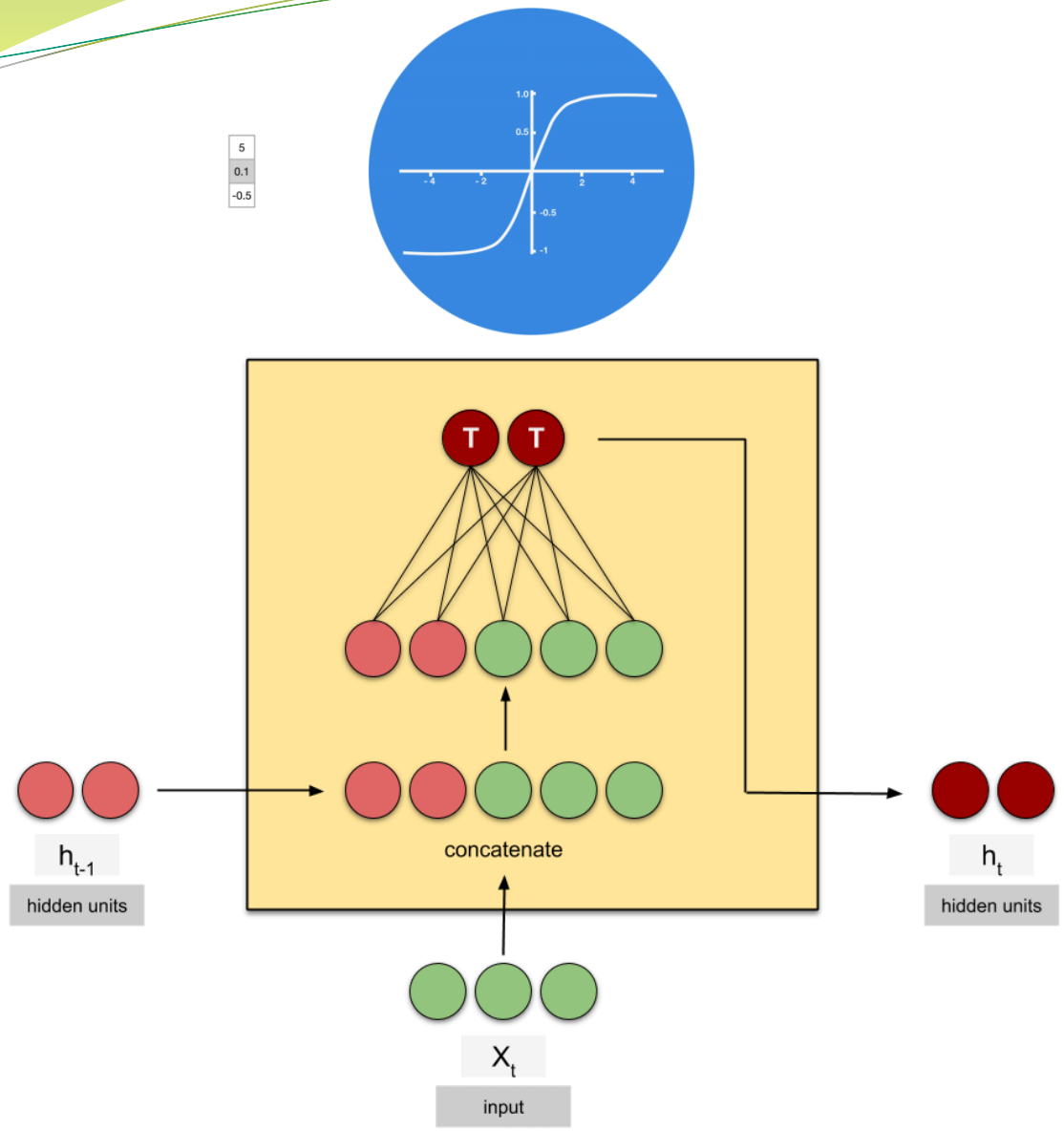
Simple RNN

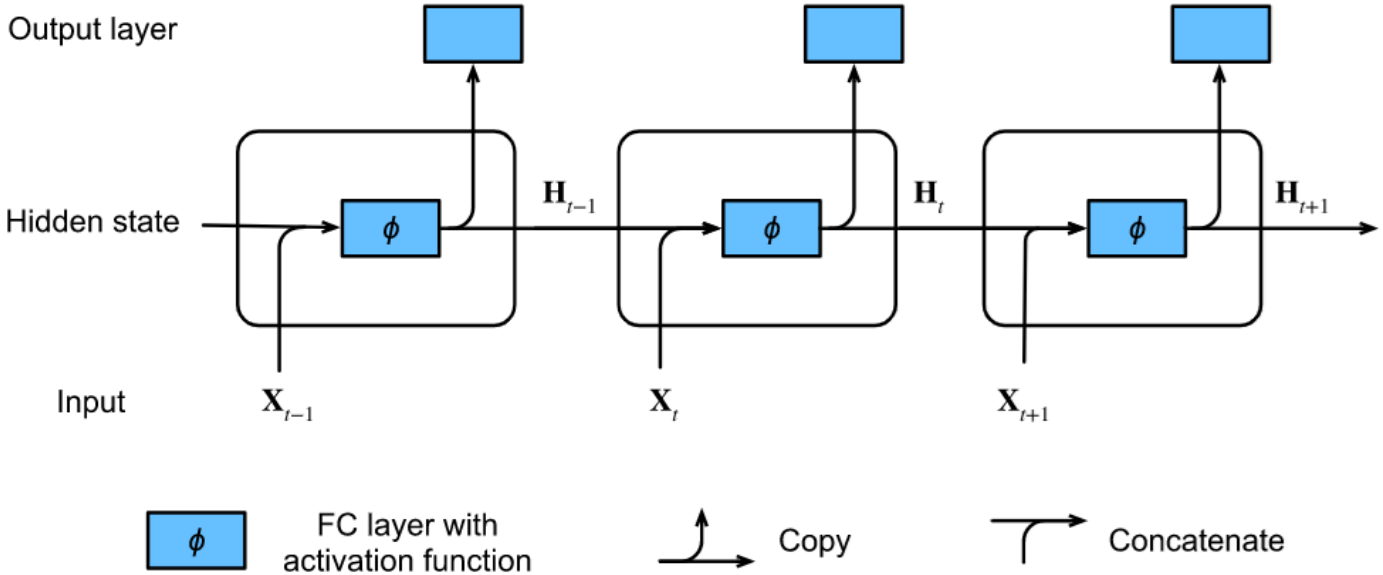
$$\mathbf{h}_t = \phi(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\mathbf{o}_t = \mathbf{W}_{hq}\mathbf{h}_t + \mathbf{b}_q$$



(Simple) RNN

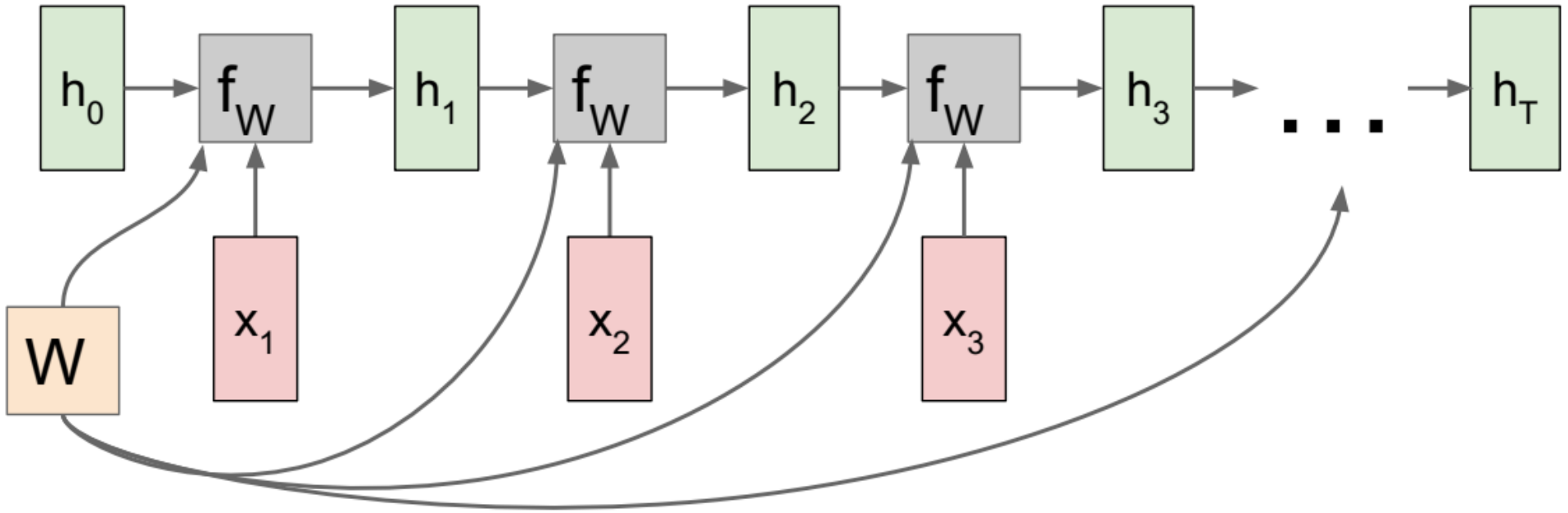




https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

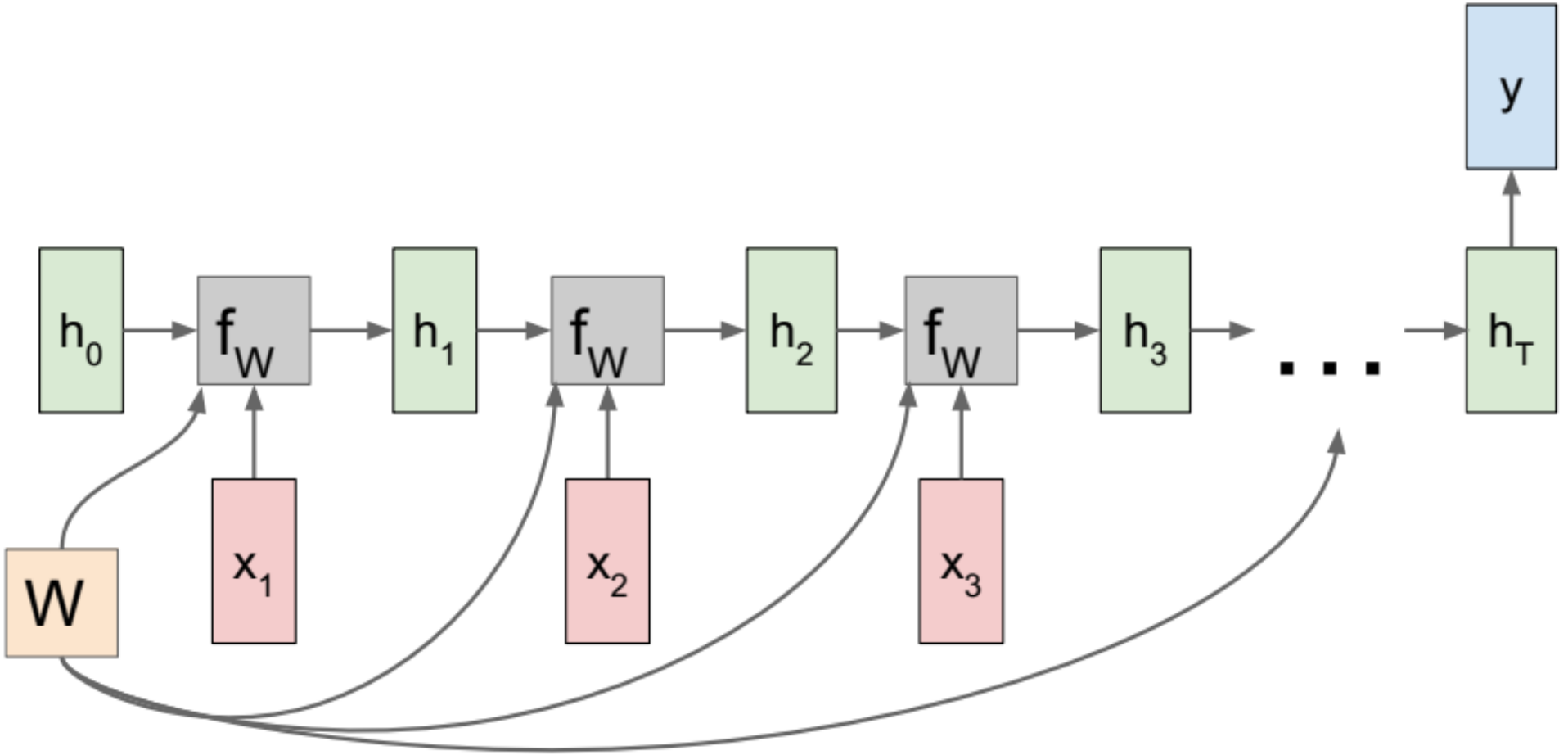
گراف محاسباتی RNN

- در هر مرحله زمانی از همان ماتریس وزن استفاده مجدد می‌شود



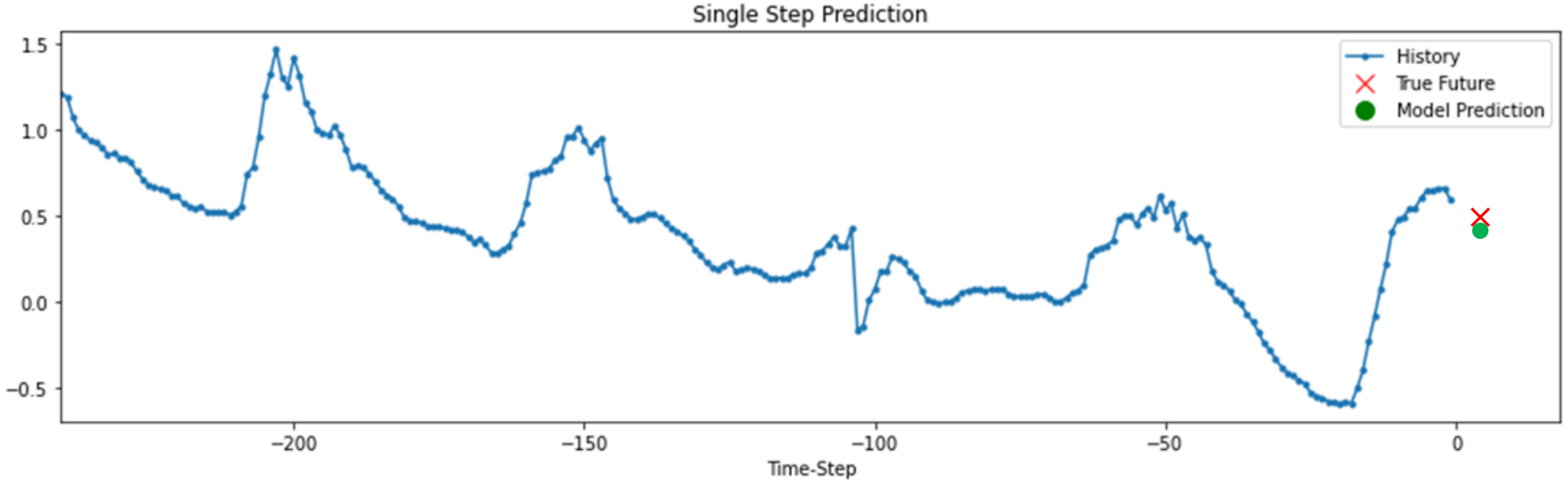
http://cs231n.stanford.edu/slides/2020/lecture_10.pdf

Many to One



مثال: پیش‌بینی سری زمانی

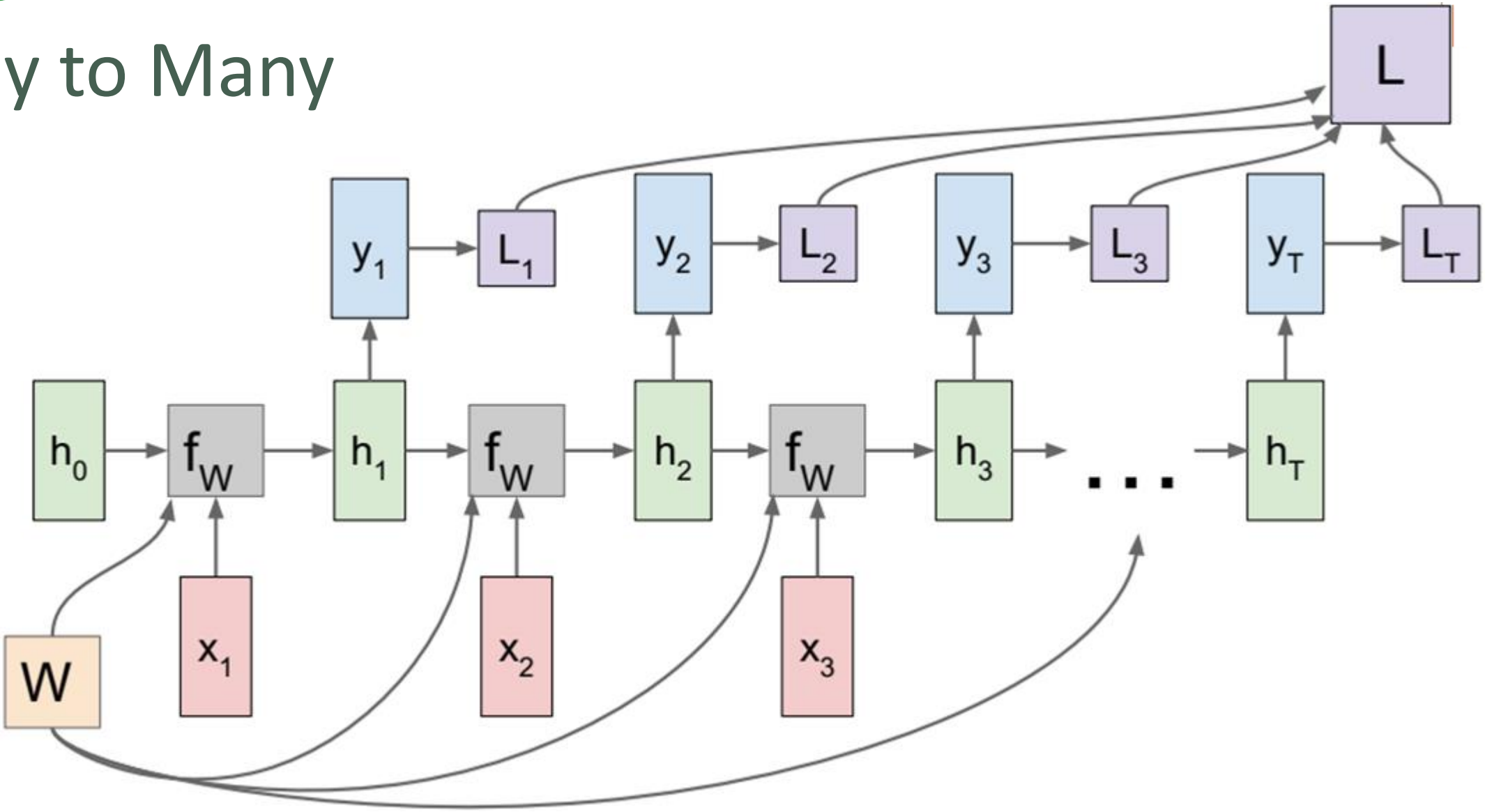
- از داده‌های ۷۲۰ زمان گذشته (۷۲۰/۶ = ۱۲۰ ساعت) با گام ۳ برای پیش‌بینی استفاده می‌شود
- این داده‌ها برای پیش‌بینی دما پس از N گام زمانی (N/6 ساعت) استفاده می‌شود
  - برای همگرایی بهتر، از نرمال‌سازی ویژگی‌ها استفاده می‌کنیم
- از ۳۰۰،۶۹۳ نمونه ابتدایی برای آموزش و از ۱۱۹،۱۲۶ نمونه انتهایی برای آزمون استفاده شده است



شبکه	تعداد پارامتر	زمان هر epoch	خطای آموزش	خطای آزمون
FC (64) + FC(1)	107,649	52s	0.1377	0.1413
SimpleRNN (64) + FC (1)	4,673	241s	0.0403	0.0599

زمان هدف	۱ ساعت		۶ ساعت		۱۲ ساعت	
شبکه	خطای آموزش	خطای آزمون	خطای آموزش	خطای آزمون	خطای آموزش	خطای آزمون
SimpleRNN (64) + FC (1)	0.0158	0.0311	0.2193	0.2516	0.2752	0.4135
GRU (64) + FC (1)	0.0127	0.0165	0.0662	0.0831	0.0986	0.1313

Many to Many



مدل زبان

- هدف از یک مدل زبان این است که احتمال یک دنباله از توکن‌ها را تخمین بزند

$$P(x_1, x_2, \dots, x_T)$$

- یک از کاربردهای مدل‌های ربان، تولید متن طبیعی است

$$x_t \sim P(x_t \mid x_{t-1}, \dots, x_1)$$

- علاوه بر این، برای ایجاد یک گفتگوی معنادار، می‌توان از شرطی کردن متن به قطعات گفتگوی قبلی استفاده کرد
- به عنوان مثال، عبارت‌های “to recognize speech” و “to wreck a nice beach” شبیه به هم تلفظ می‌شوند اما از لحاظ محتوایی و با استفاده از مدل زبان می‌توان عبارت نخست را انتخاب کرد

یادگیری یک مدل زبان

- طبق قاعده ضرب در تئوری احتمال:

$$P(x_1, x_2, \dots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{t-1}, \dots, x_1)$$

- به عنوان مثال:

$$P(\text{deep}, \text{learning}, \text{is}, \text{fun}) = P(\text{deep})P(\text{learning} \mid \text{deep})P(\text{is} \mid \text{deep}, \text{learning})P(\text{fun} \mid \text{deep}, \text{learning}, \text{is})$$

- نیاز است تا احتمال واژه‌ها و همچنین احتمال شرطی واژه‌ها را محاسبه کنیم
- اگر یک مجموعه داده بسیار بزرگ داشته باشیم، می‌توانیم به صورت آماری آنها را تخمین بزنیم
- برای کلمه "deep" می‌توان درصد جملاتی که با آن شروع شده‌اند را محاسبه کرد
  - یک روش با مقداری دقت کمتر این است که مجموع تمام رخدادهای "deep" را به تعداد کل واژگان تقسیم کنیم

یادگیری یک مدل زبان

- برای تخمین احتمال شرطی می‌توان به صورت زیر عمل کرد

$$\hat{P}(\text{learning} \mid \text{deep}) = \frac{n(\text{deep, learning})}{n(\text{deep})}$$

- تخمین احتمال شرطی دشوارتر است زیرا تعداد تکرار جفت واژه‌ها کمتر است و این پیچیدگی برای دنباله‌های طولانی‌تر بیشتر می‌شود
- به دلیل اینکه برخی ترکیب‌ها ممکن است در مجموعه داده ما اتفاق نیافتاده باشند، در بسیاری مواقع از تقریب‌هایی مانند Laplace smoothing استفاده می‌کنیم
  - اگر $\epsilon_1 = 0$ باشد همان حالت عادی است و اگر $\epsilon_1 \to \infty$ به توزیع یکنواخت همگرا می‌شود

$$\hat{P}(x) = \frac{n(x) + \epsilon_1/m}{n + \epsilon_1}$$

$$\hat{P}(x' \mid x) = \frac{n(x, x') + \epsilon_2 \hat{P}(x')}{n(x) + \epsilon_2}$$

$$\hat{P}(x'' \mid x, x') = \frac{n(x, x', x'') + \epsilon_3 \hat{P}(x'')}{n(x, x') + \epsilon_3}$$

مدل‌های مارکوف

- فرض می‌کنیم نمونه فعلی به تعداد کمی از نمونه‌های قبلی وابسته باشد

- مرتبه ۰ (مستقل):

$$P(x_1, x_2, x_3, x_4) = P(x_1)P(x_2)P(x_3)P(x_4)$$

- مرتبه ۱:

$$P(x_1, x_2, x_3, x_4) = P(x_1)P(x_2|x_1)P(x_3|x_2)P(x_4|x_3)$$

- مرتبه ۲:

$$P(x_1, x_2, x_3, x_4) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)P(x_4|x_2, x_3)$$

- به این مدل‌ها به ترتیب unigram، bigram و trigram گفته می‌شود