Ch07. Recurrent Neural Network (RNN)

Hwanjo Yu

POSTECH

http://hwanjoyu.org

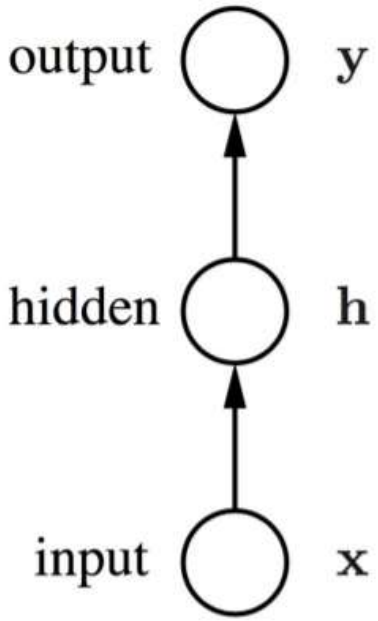
Sequence Modeling

Wanted: Probability over sequences, $x \sim p(x_1, x_2, \ldots, x_T)$.

- Speech recognition: Audio to text
- Machine translation: Text to text
- Sentiment analysis: Text to labels (positive, negative, neutral)
- Music generation: Initials to music
- Video classification: Video to labels
- Name entity recognition: text to labels (name entity)

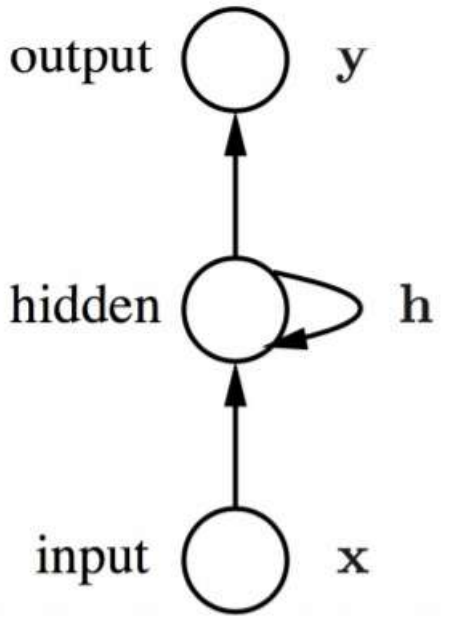
Feedforward Net

- $\mathbf{y}_t = \varphi\left(\mathbf{W}_{yh}\mathbf{h}_t + \mathbf{b}_y\right)$,
- $\mathbf{h}_t = \varphi(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{b}_h)$.



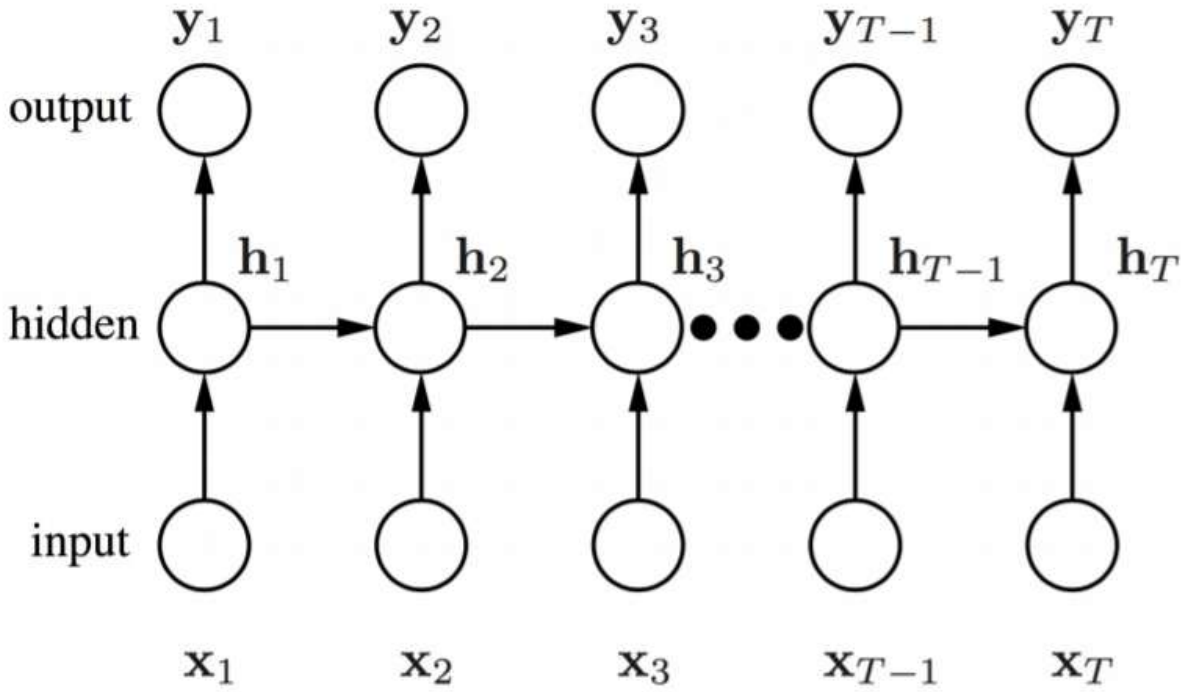
Vanilla RNN

- $\mathbf{y}_t = \varphi\left(\mathbf{W}_{yh}\mathbf{h}_t + \mathbf{b}_y\right)$,
- $\mathbf{h}_t = \varphi(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$.

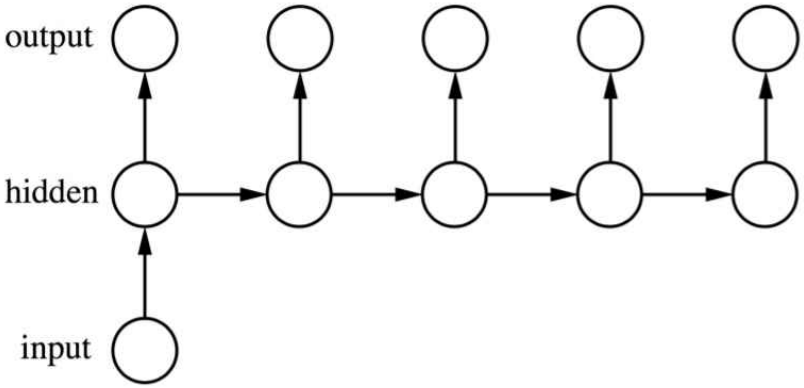


Vanilla RNN: Unfolding Computational Graph

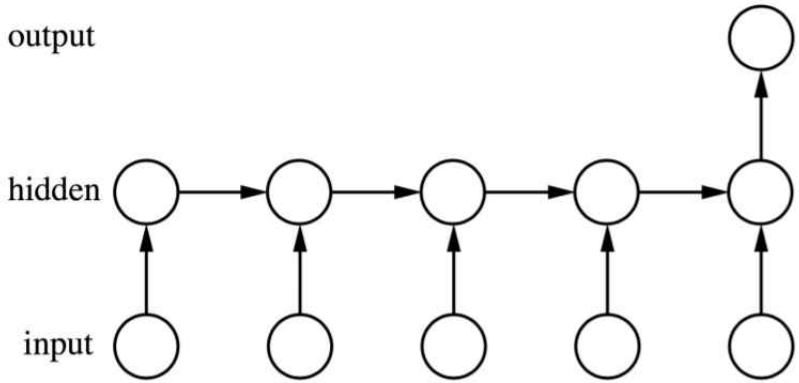
- $\mathbf{y}_t = \varphi\left(\mathbf{W}_{yh}\mathbf{h}_t + \mathbf{b}_y\right)$,
- $\mathbf{h}_t = \varphi(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$.



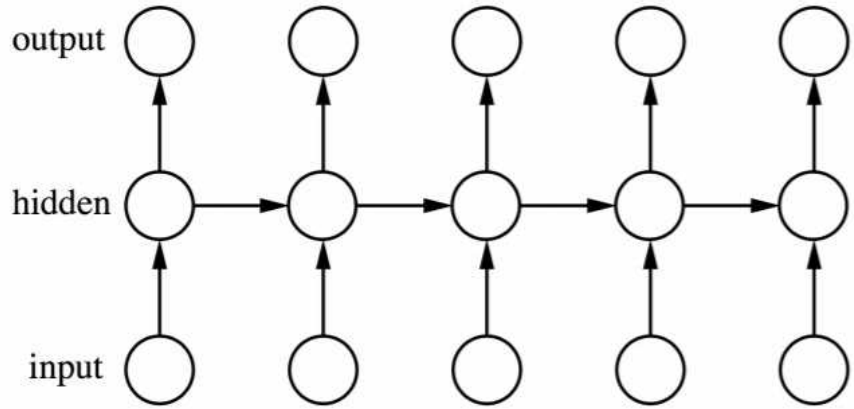
One-to-Many



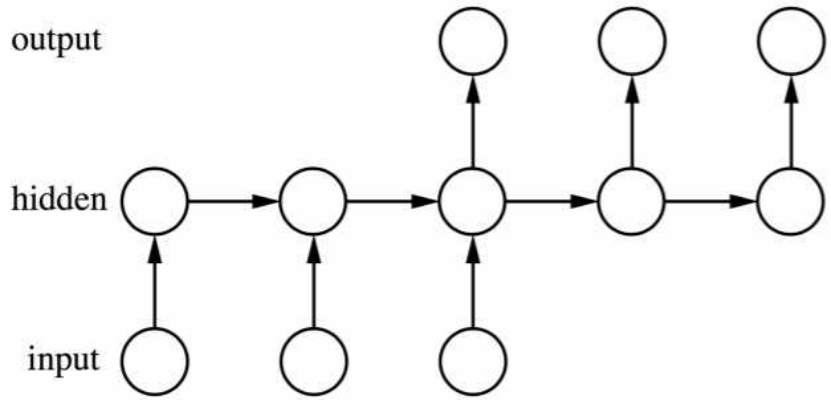
Many-to-One



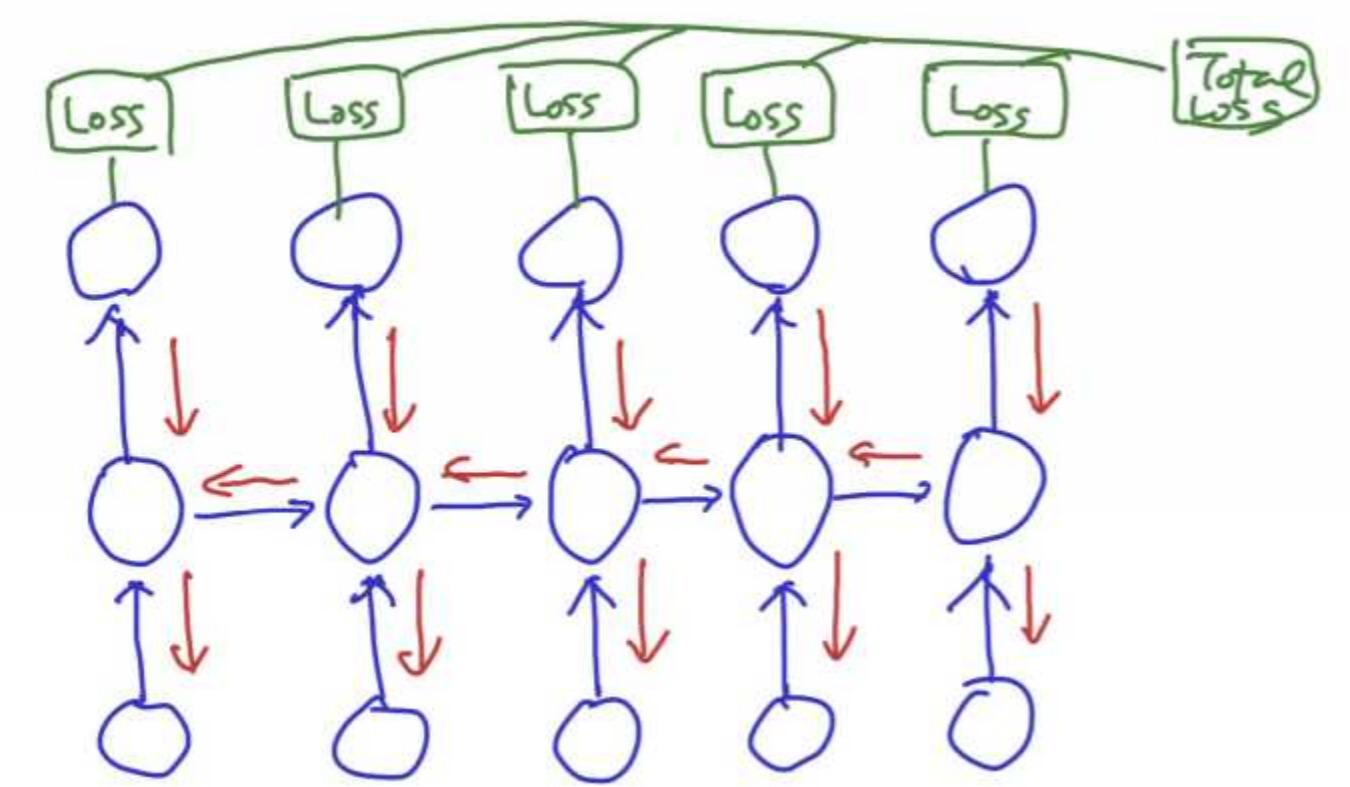
Many-to-Many



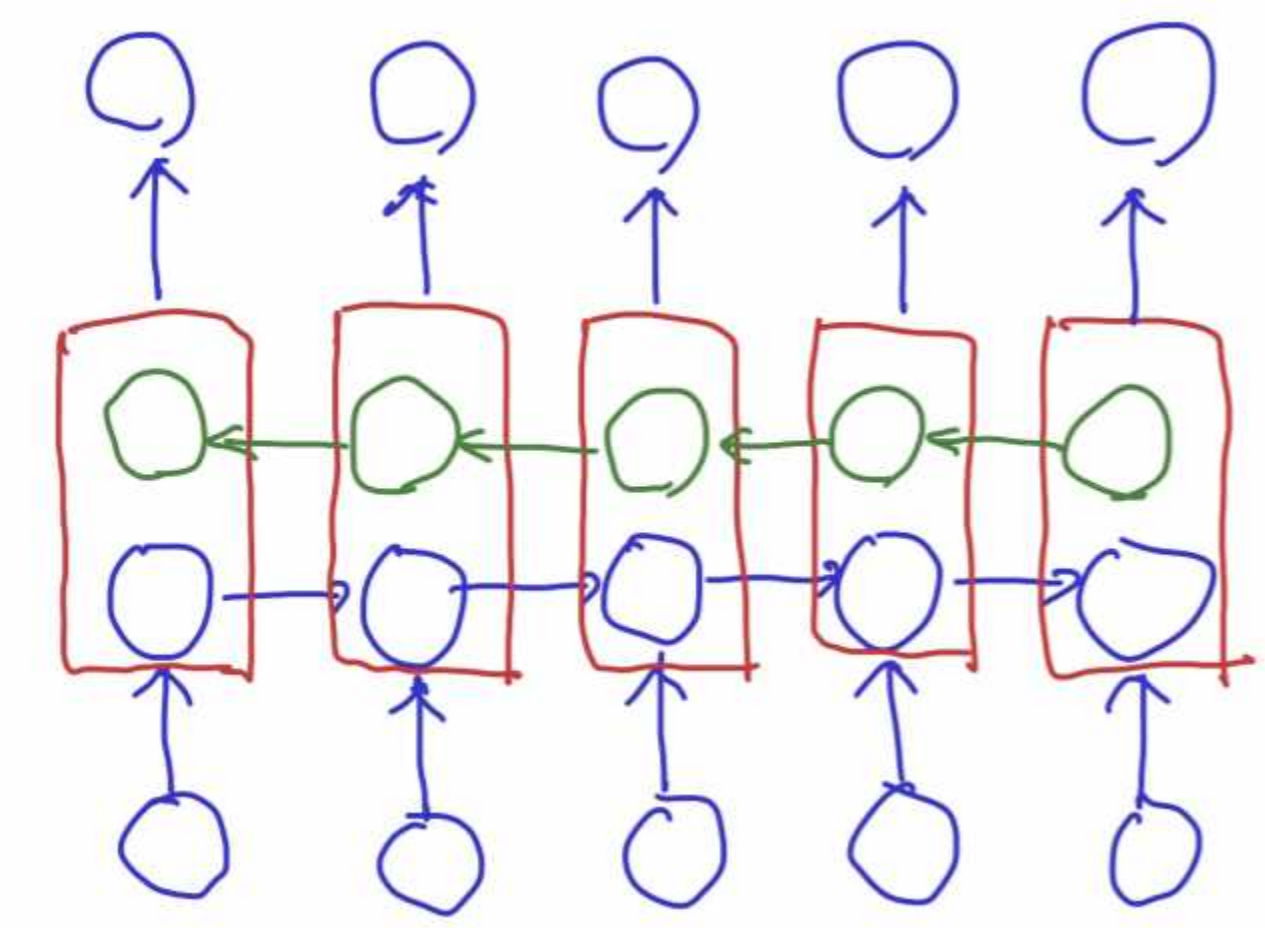
Many-to-Many: Encoder-Decoder



Backprop through Time



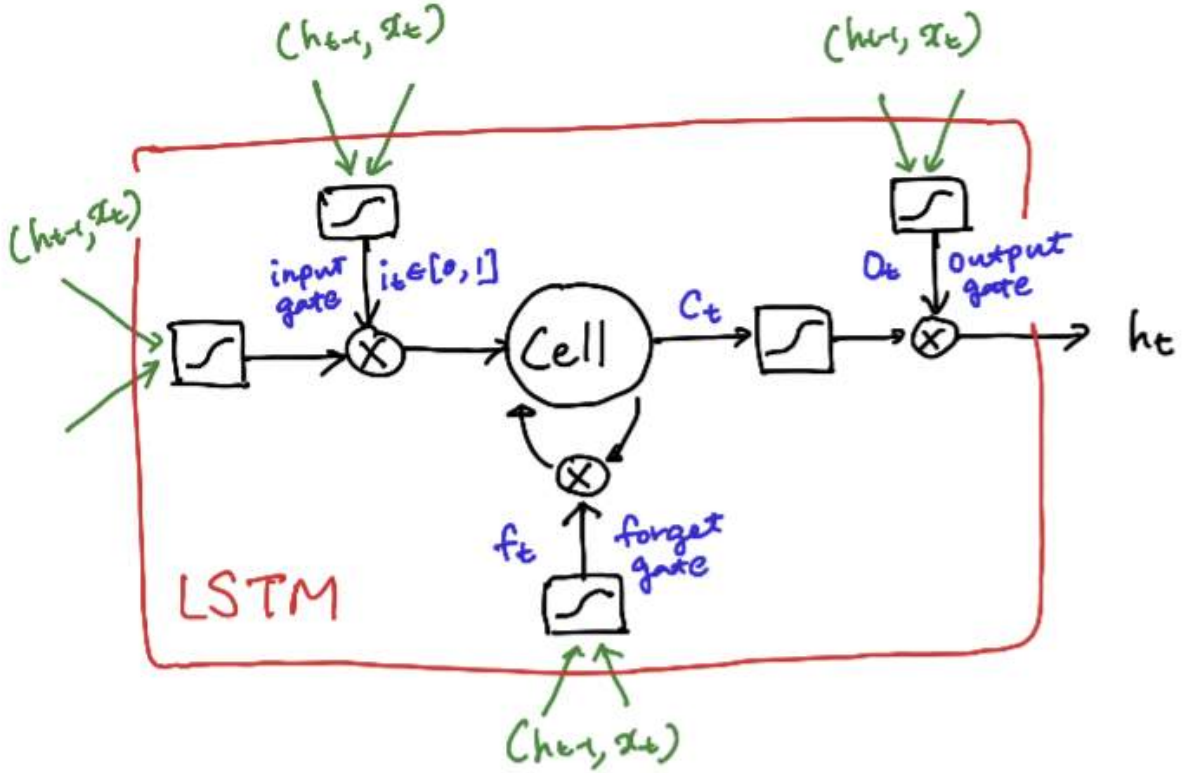
Bidirectional RNN



Long Short Term Memory (LSTM)

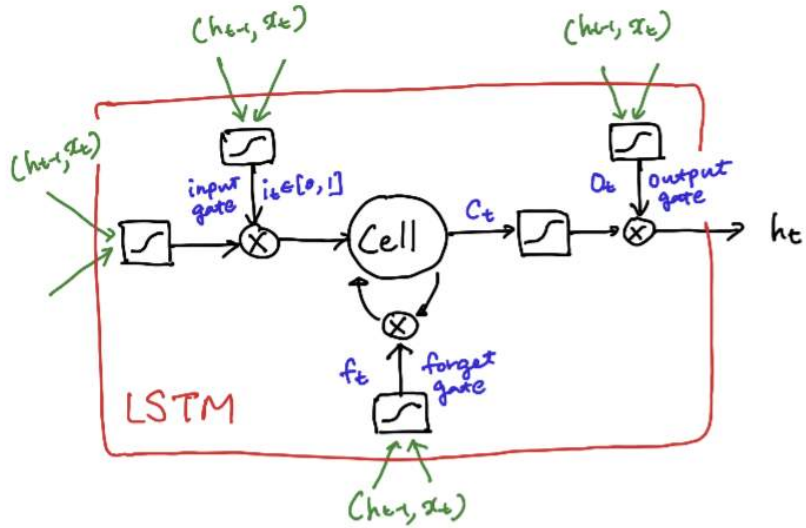
S. Hochreiter and J. Schmidhuber (1997), ”Long short-term memory,” Neural Computation.

- $\mathbf{h}_t = \mathrm{LSTM}(\mathbf{x}_t, \mathbf{h}_{t-1})$
  - Input gate: Scales input to cell (write)
  - Output gate: Scales output from cell (read)
  - Forget gate: Scales old cell value (reset)



LSTM Cell Updates

- $\mathbf{f}_t = \text{sigmoid}\left(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{b}_f\right)$
- $\mathbf{i}_t = \text{sigmoid}(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{b}_i)$
- $\mathbf{o}_t = \text{sigmoid}(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{b}_o)$



- $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c)$
- $\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$

Gated Recurrent Unit (GRU)

K. Cho et al. (2014), ”Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” EMNLP.

I J. Chung et al. (2014), ”Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” Preprint arXiv:1412.3555?.

- $\mathbf{h}_t = \mathrm{GRU}(\mathbf{x}_t, \mathbf{h}_{t-1})$
- Two gates: update gate $\mathbf{z}_t$ and reset gate $\mathbf{r}_t$

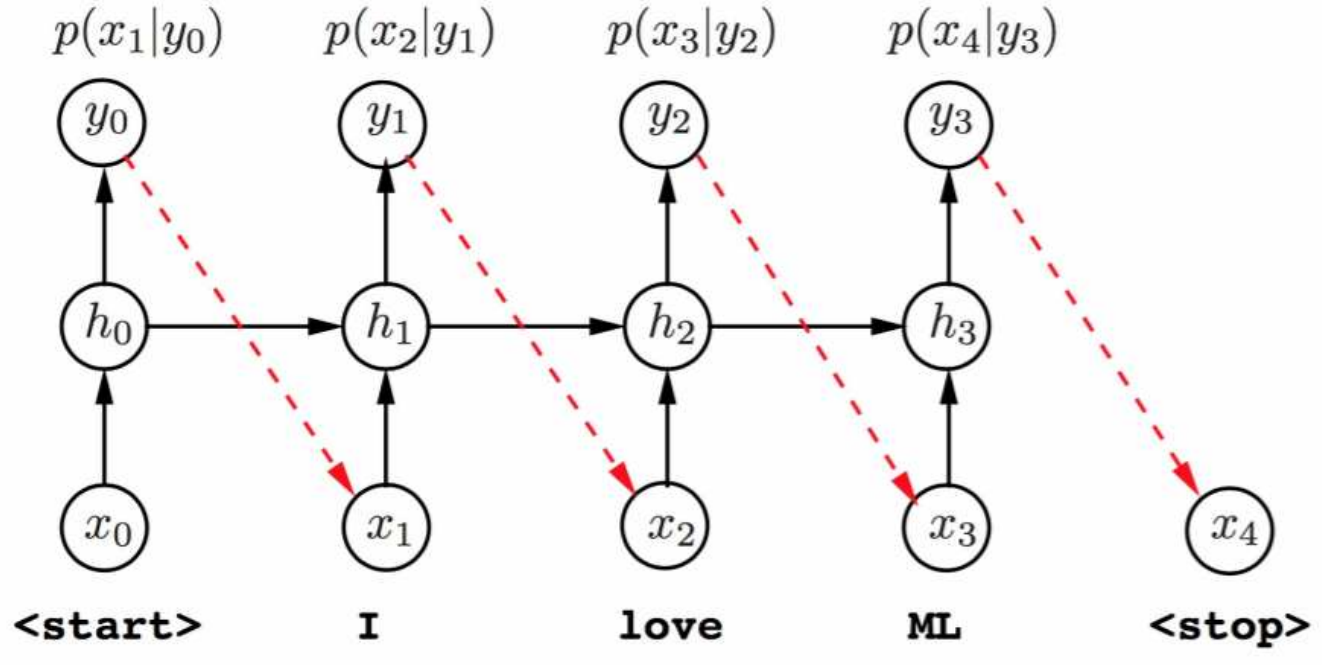
- $\mathbf{z}_t = \mathrm{sigmoid}(\mathbf{W}_{zx}\mathbf{x}_t + \mathbf{W}_{zh}\mathbf{h}_{t-1} + \mathbf{b}_z)$
- $\mathbf{r}_t = \mathrm{sigmoid}(\mathbf{W}_{rx}\mathbf{x}_t + \mathbf{W}_{h}\mathbf{h}_{t-1} + \mathbf{b}_r)$

- $\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tanh(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h)$

Generative RNN for language model

Alex Graves (2013), ”Generating Sequences With Recurrent Neural Networks,” Preprint arXiv:1308.0850.

- Parameterize $p(\mathbf{x}_{t+1}|\mathbf{y}_t)$ where $\mathbf{y}_t = \sigma(\mathbf{W}_y\mathbf{h}_t + \mathbf{b}_y)$: $\mathbf{x}$ is sampled from $p(\mathbf{x}_{t+1}|\mathbf{y}_t)$
- Likelihood $p(x_{1:T+1}) = \prod_{t=1}^{T} p(\mathbf{x}_{t+1}|\mathbf{y}_t)$
- Loss function $\mathcal{J} = -\sum_{t=1}^{T} \log p(\mathbf{x}_{t+1}|\mathbf{y}_t)$



Training RNNs for Sequence Prediction

Teacher-forcing: R. J. Williams and D. Zipser (1989), ”A learning algorithm for continually running fully recurrent neural networks,” Neural computation.

Scheduled sampling: S. Bengio, O. Vinyals, N. Jaitly, N. Shazeer (2015), ”Scheduled sampling for sequence prediction with recurrent neural networks,” NeurIPS.

Professor-forcing: Alex Lamb, Anirudh Goyal, Aaron Courville, Ying Zhang, Saizheng Zhang, Yoshua Bengio (2016), ”Professor Forcing: A New Algorithm for Training Recurrent Networks,” NeurIPS.

Model

- **Supervised learning:** Given a pair of input/output sequences, $\{(\mathbf{X}^n, \mathbf{Y}^n)|n = 1, \dots, N\}$, where the input could be static or dynamic and the target is a sequence belonging to a fixed known dictionary.
- Model an output sequence $\mathbf{Y} = (\mathbf{y}_1, \dots, \mathbf{y}_T)$, given an input sequence $\mathbf{X} = \mathbf{x}_{1:T_X}$

$$p(\mathbf{y}_1, \dots, \mathbf{y}_T | \mathbf{x}_1, \dots, \mathbf{x}_{T_X}) = \prod_{t=1}^{T} p(\mathbf{y}_t | \mathbf{y}_{1:t-1}, \mathbf{x}_{1:T_X}) = \prod_{t=1}^{T} p(\mathbf{y}_t | \mathbf{h}_t)\,.$$

- Hidden states $\mathbf{h}_t$ are computed by an RNN:

$$\mathbf{h}_t = \begin{cases} f(\mathbf{x}_{1:T_X}) & \text{if } t = 1, \\ f(\mathbf{h}_{t-1}, \mathbf{y}_{t-1}) & \text{otherwise,} \end{cases}$$

- where $f(\cdot)$ could be the vanilla RNN, LSTM, or GRU.

Training RNNs for Sequence Prediction

Training

- Determine a set of model parameters θ that maximizes the log-likelihood of producing the correct target sequence $\mathbf{Y}^n$ given the input $\mathbf{X}^n$ for all training pairs $\{(\mathbf{X}^n, \mathbf{Y}^n) | n = 1, \ldots, N\}$:

$$\theta_{ML} = \underset{\theta}{\operatorname{argmax}} \sum_{(\mathbf{X}^n, \mathbf{Y}^n) \in \mathcal{D}} \log p(\mathbf{Y}^n | \mathbf{X}^n; \theta)$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{(\mathbf{X}^n, \mathbf{Y}^n) \in \mathcal{D}} \sum_{t} \log p(\mathbf{y}_t^n | \mathbf{y}_{1:t-1}^n, \mathbf{X}^n; \theta)$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{(\mathbf{X}^n, \mathbf{Y}^n) \in \mathcal{D}} \sum_{t} \log p(\mathbf{y}_t^n | \mathbf{h}_t^n; \theta),$$

- where

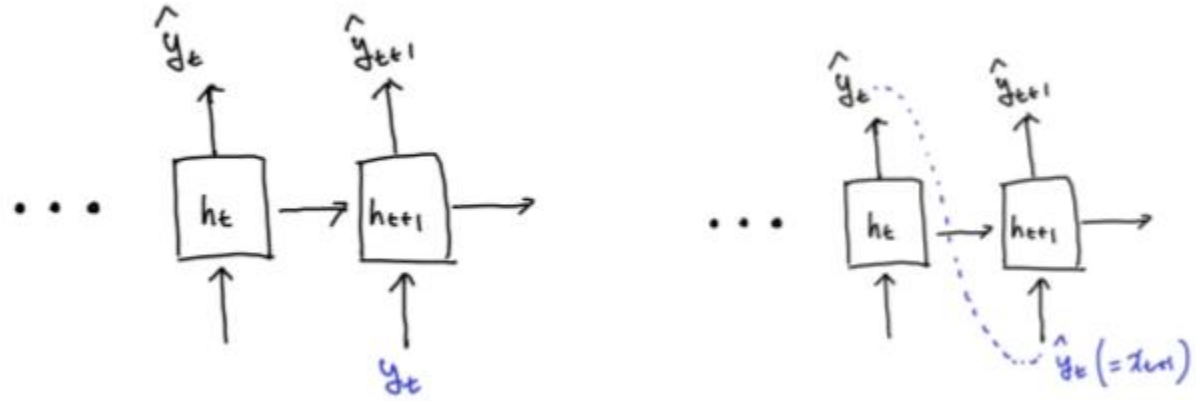
$$\mathbf{h}_t^n = \begin{cases} f(\mathbf{X}^n) & \text{if } t = 1, \\ f(\mathbf{h}_{t-1}^n, \mathbf{y}_{t-1}^n) & \text{otherwise,} \end{cases}$$

- Prediction of token $\mathbf{y}_t$ requires either the true previous token $\mathbf{y}_{t-1}$ or an estimate $\hat{\mathbf{y}}_{t-1}$ coming from the model itself.

Teacher-Forcing

R. J. Williams and D. Zipser (1989), ”A learning algorithm for continually running fully recurrent neural networks,” Neural computation.

- **Training:** The model receives the ground truth output $\mathbf{y}_t$ (instead of the generated one $\hat{\mathbf{y}}_t$) as input in the next time step $\mathbf{x}_{t+1}$.



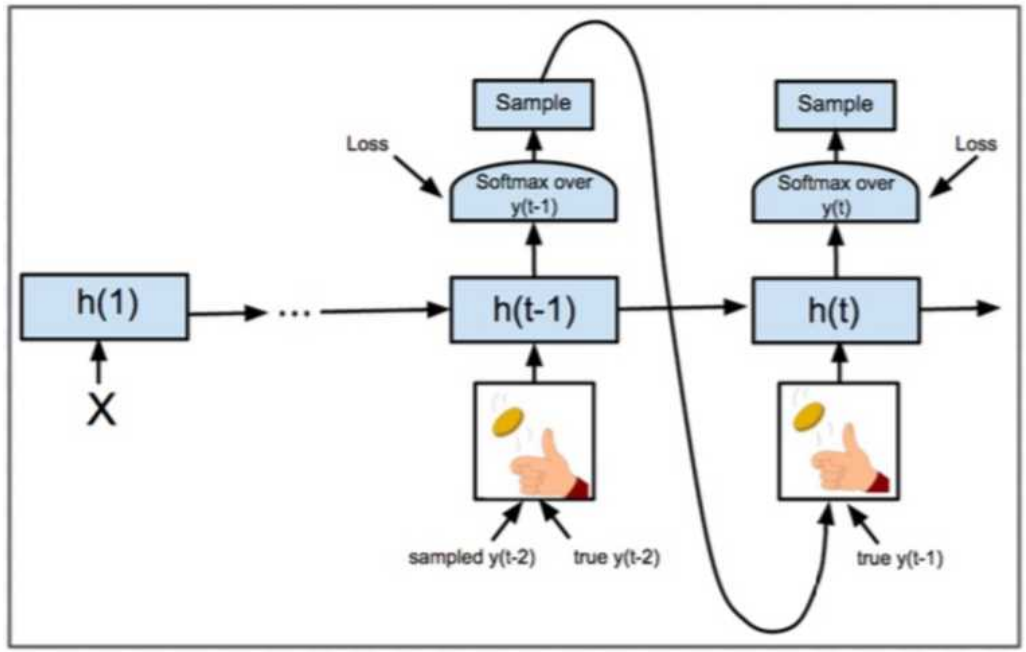
(a) TF (b) Without TF

- **Inference (test):** Open loop mode with network outputs fed back as inputs.
- Inputs that the model will see during training time could be quite different from that it will see at test time

Scheduled Sampling

S. Bengio, O. Vinyals, N. Jaitly, N. Shazeer (2015), "Scheduled sampling for sequence prediction with recurrent neural networks," NeurIPS.

- Train the model with both teacher-forced inputs $\mathbf{y}_t$ and free-running inputs $\hat{\mathbf{y}}_t$.

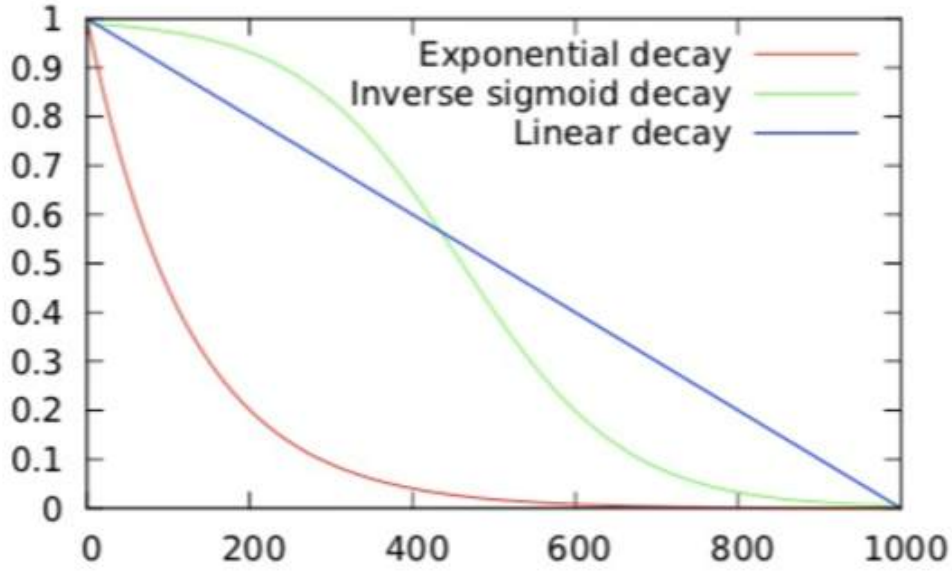


Mini-batch i:

- Use the true previous token $\mathbf{y}_t$ with probability ϵ_i;
- Use the model's own prediction $\hat{\mathbf{y}}_t$ with probability $1 - \epsilon_i$.

Decay Schedule

$\epsilon_i = 1$ (teacher forcing mode) and $\epsilon_i = 0$ (free running mode)



- **Linear decay:** $\epsilon_i = \max(\epsilon, k - ci)$ where $0 \leq \epsilon < 1$ is the minimum amount of truth to be given to the model and k and c provide the offset and slope of the decay, which depend on the expected speed of convergence.
- **Exponential decay:** $\epsilon_i = k^i$ where $k < 1$ is a constant that depends on the expected speed of convergence.
- **Inverse sigmoid decay:** $\epsilon_i = \frac{k}{k + \exp(i/k)}$ where $k \geq 1$ depends on the expected speed of convergence.

Experiments on Image Captioning

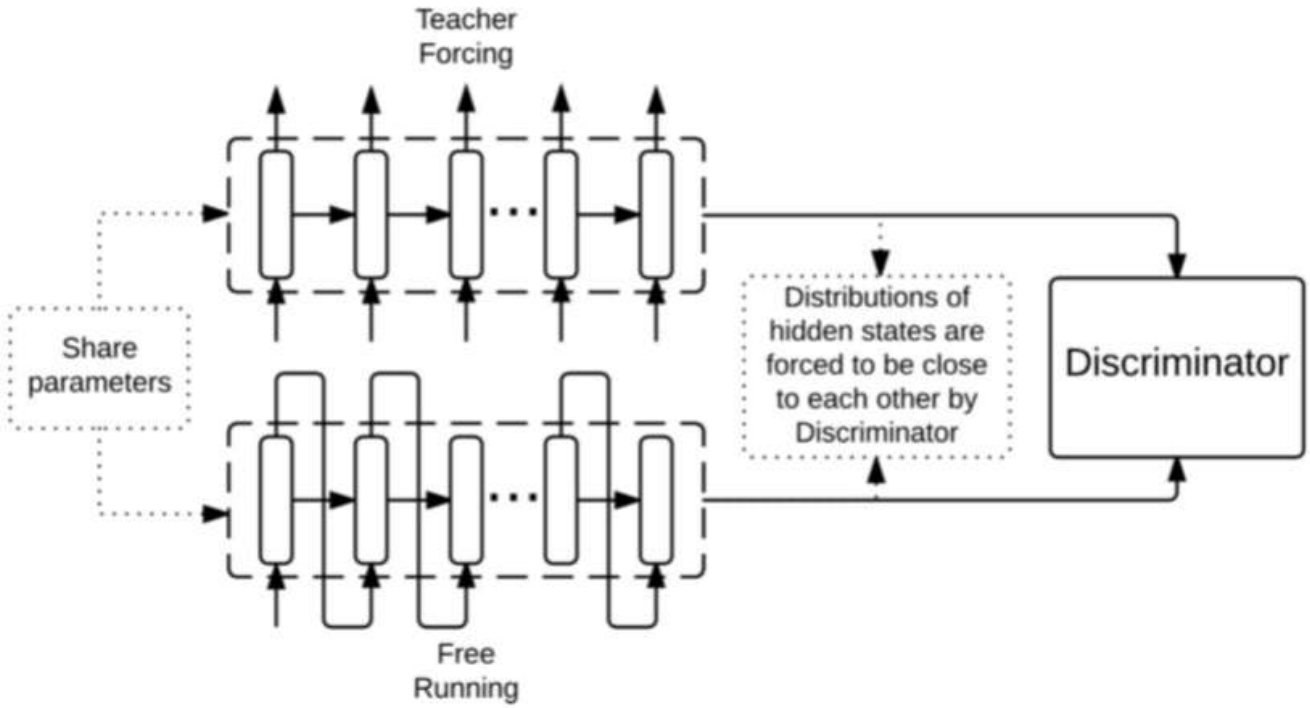
Table 1: Various metrics (the higher the better) on the MSCOCO development set for the image captioning task.

Approach vs Metric	BLEU-4	METEOR	CIDER
Baseline	28.8	24.2	89.5
Baseline with Dropout	28.1	23.9	87.0
Always Sampling	11.2	15.7	49.7
Scheduled Sampling	**30.6**	**24.3**	**92.1**
Uniform Scheduled Sampling	29.2	24.2	90.9
Baseline ensemble of 10	30.7	25.1	95.7
Scheduled Sampling ensemble of 5	**32.3**	**25.4**	**98.7**

It's worth noting that we used our scheduled sampling approach to participate in the 2015 MSCOCO image captioning challenge [21] and ranked first in the final leaderboard.

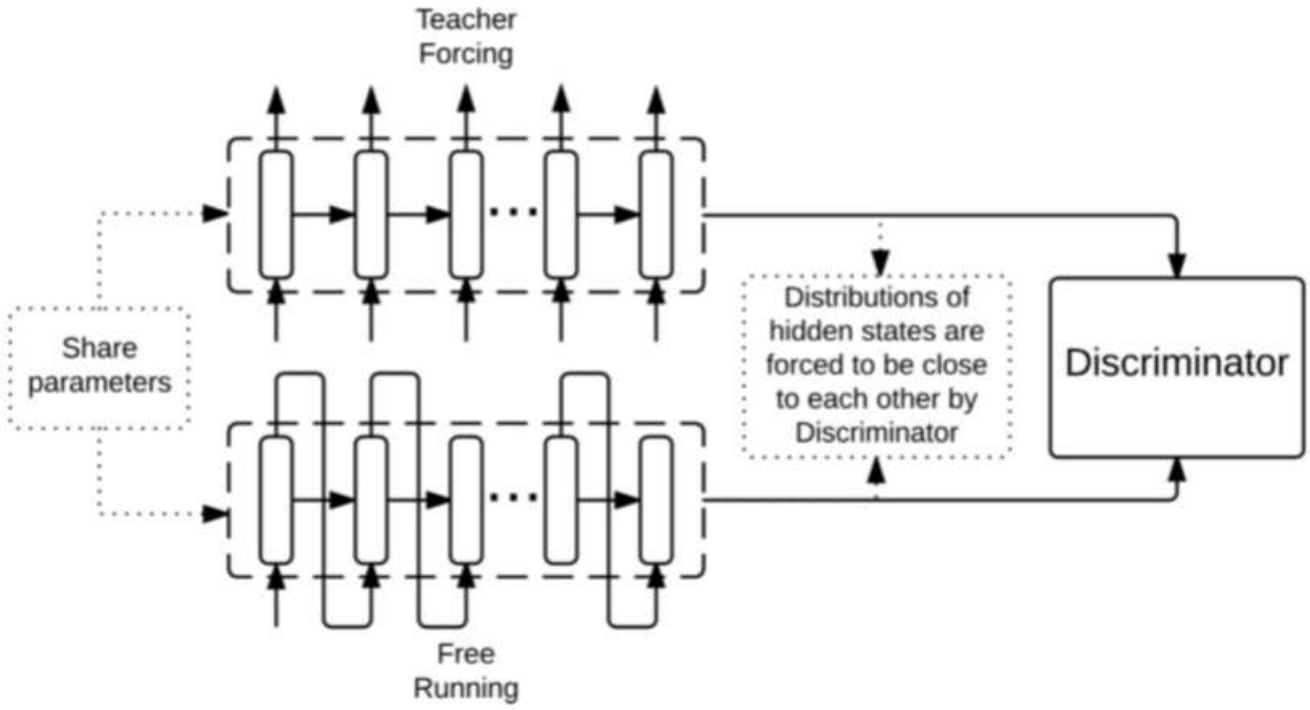
Professor-Forcing via Adversarial Training

Alex Lamb, Anirudh Goyal, Aaron Courville, Ying Zhang, Saizheng Zhang, Yoshua Bengio (2016), ”Professor Forcing: A New Algorithm for Training Recurrent Networks,” NeurIPS.



- Want the RNN to match the training data.
- Want the behavior of the network (both in its outputs and in the dynamics of its hidden states) to be *indistinguishable* whether the network is trained with its inputs clamped to a training sequence (teacher forcing mode) or whether its inputs are self-generated (free-running generative mode).

Professor-Forcing via Adversarial Training



- θ_g: Parameters of the generative RNN.
- θ_d: Parameters of the discriminator.
- $B(\mathbf{x}, \mathbf{y}, \theta_g)$: Outputs the behavior sequence (chosen hidden states and output values). Note that $\mathbf{x}$ comes always from the training data, but $\mathbf{y}$ comes from the training data or is self-generated.
- $D(B(\mathbf{x}, \mathbf{y}, \theta_g); \theta_d)$: Discriminator which scores the probability that the behavior sequence was produced in teacher-forcing mode.

Professor-Forcing: Training

- **Discriminator:** Trained to maximize the likelihood of correctly classifying a behavior sequence:

$$\max_{\theta_d} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim \text{data}} \left[\log D\left(B\left(\mathbf{x},\mathbf{y},\theta_g\right);\theta_d\right)\right] + \mathbb{E}_{\mathbf{y}\sim p_{\theta_g}(\mathbf{y}|\mathbf{x})} \left[\log\left(1 - D\left(B\left(\mathbf{x},\mathbf{y},\theta_g\right);\theta_d\right)\right)\right].$$

- **Generator:** Trained (a) to maximize the likelihood of the data:

$$\mathcal{L}_1\left(\theta_g\right) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim \text{data}} \left[\log p_{\theta_g}(\mathbf{y}|\mathbf{x})\right],$$

- and (b) to fool the discriminator.

$$\mathcal{L}_2\left(\theta_g|\theta_d\right) = \mathbb{E}_{\mathbf{x}\sim \text{data}\ ,\ \mathbf{y}\sim p_{\theta_g}(\mathbf{y}|\mathbf{x})}\left[\log D\left(B\left(\mathbf{x},\mathbf{y},\theta_g\right);\theta_d\right)\right],$$

$$\mathcal{L}_3\left(\theta_g|\theta_d\right) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim \text{data}} \left[\log(1 - D\left(B\left(\mathbf{x},\mathbf{y},\theta_g\right);\theta_d\right))\right].$$

- Training the generator involves:

$$\max_{\theta_g} \mathcal{L}_1\left(\theta_g\right) + \mathcal{L}_2\left(\theta_g|\theta_d\right)$$

or

$$\text{or } \max_{\theta_g} \mathcal{L}_1\left(\theta_g\right) + \mathcal{L}_2\left(\theta_g|\theta_d\right) + \mathcal{L}_3\left(\theta_g|\theta_d\right)$$