# COMP5212: Machine Learning

## Lecture 14

Minhao Cheng

# Logistics

## Term project

- 5 mins Presentation
  - Time: Last two lectures (Nov 28, Nov 30)
- Project report
  - ~4 pages (could put additional results in supplement)

# Programming HW2

## MLP & CNN

| Layer Number | MLP (in dim->out dim) | CNN |
|---|---|---|
| 1 | Linear 3072->1024 | Conv2D, in channel: 3, out channel: 64, kernel: $3 \times 3$, stride: 1, padding: 1 |
| 2 | Linear 1024->512 | Conv2D, in channel: 64, out channel: 128, kernel: $3 \times 3$, stride: 2, padding: 1 |
| 3 | Linear 512->256 | Conv2D, in channel: 128, out channel: 256, kernel: $3 \times 3$, stride: 2, padding: 1 |
| 4 | Linear 256->128 | Conv2D, in channel: 256, out channel: 256, kernel: $3 \times 3$, stride: 2, padding: 1 |
| 5 | Linear 128->64 | Linear 4096 -> 1024 |
| 6 | Linear 63->32 | Linear 1024 -> 1024 |
| 7 | Linear 32->10 | Linear 1024 -> 10 |

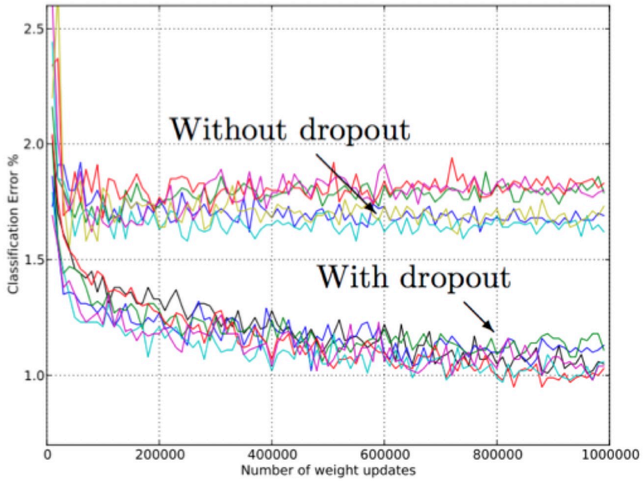
# Convolutional Neural Network

## Dropout

- One of the most effective regularization for deep neural networks

| Method | CIFAR-10 | CIFAR-100 |
|---|---|---|
| Conv Net + max pooling (hand tuned) | 15.60 | 43.48 |
| Conv Net + stochastic pooling (Zeiler and Fergus, 2013) | 15.13 | 42.51 |
| Conv Net + max pooling (Snoek et al., 2012) | 14.98 | - |
| Conv Net + max pooling + dropout fully connected layers | 14.32 | 41.26 |
| Conv Net + max pooling + dropout in all layers | 12.61 | **37.20** |
| Conv Net + maxout (Goodfellow et al., 2013) | **11.68** | 38.57 |

Table 4: Error rates on CIFAR-10 and CIFAR-100.

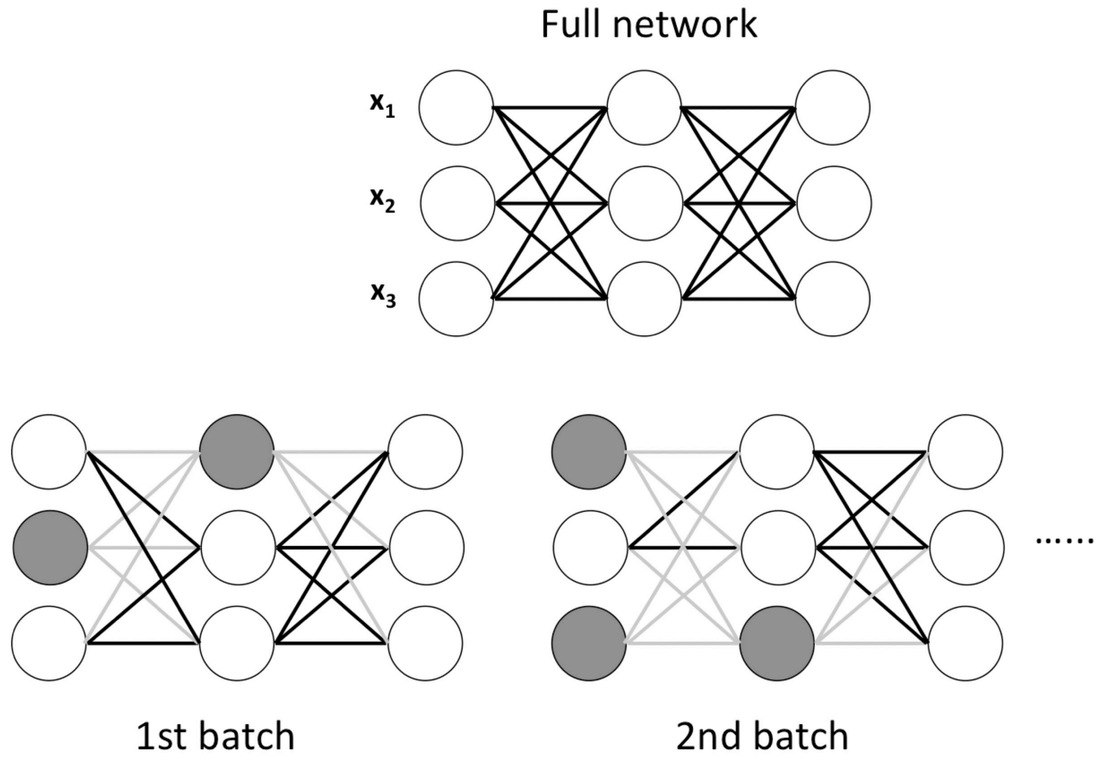



Srivastava et al, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, 2014.

# Convolutional Neural Network

## Dropout(training)

- Dropout in the **training** phase:
  - For each batch, turn off each neuron (including inputs) with a probability $1 - \alpha$
  - Zero out the removed nodes/edges and do backpropogation



# Convolutional Neural Network

## Dropout(test)

- The model is different from the full model:
- Each neuron computes
  - $x_i^{(l)} = B\sigma(\sum_j W_{ij}^{(l)} x_j^{(l-1)} + b_i^{(l)})$
    - Where B is Bernoulli variable that takes 1 with probability $\alpha$
- The expected output of the neuron:
  - $E[x_i^{(l)}] = \alpha\sigma(\sum_j W_{ij}^{(l)} x_j^{(l-1)} + b_i^{(l)})$
- Use the expected output at test time $\Rightarrow$ multiply all the weights by $\alpha$

# Convolutional Neural Network

## Explanations of dropout

- For a network with $n$ neurons, there are $2^n$ possible sub-networks
- Dropout: randomly sample over all $2^n$ possibilities
- Can be viewed as a way to learn Ensemble of $2^n$ models

# Convolutional Neural Network

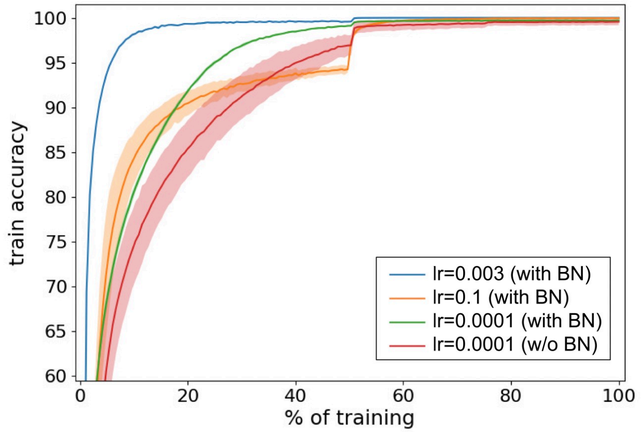
## Batch Normalization

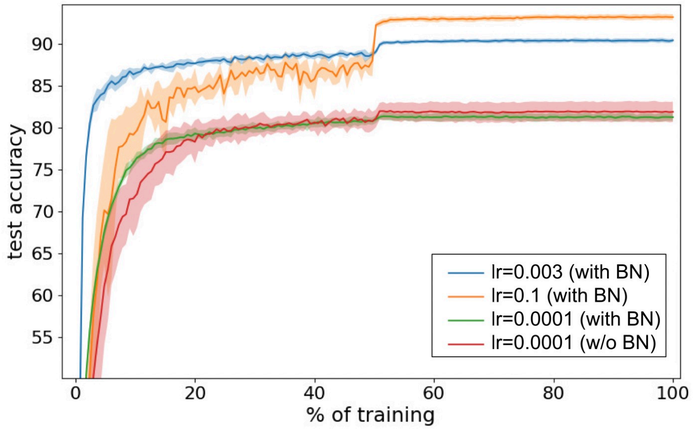
- Initially proposed to reduce co-variate shift
- $$O_{b,c,x,y} \leftarrow \gamma \frac{I_{b,c,x,y} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} + \beta \quad \forall b, c, x, y,$$
- $\mu_c = \frac{1}{|B|} \sum_{b,x,y} I_{b,c,x,y}$: the mean for channel $c$, and $\sigma_c$ standard deviation.
- $\gamma$ and $\beta$: two learnable parameters

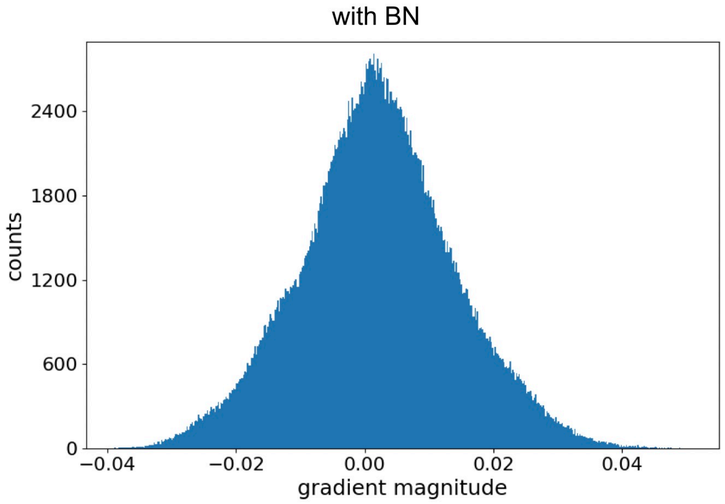
# Convolutional Neural Network

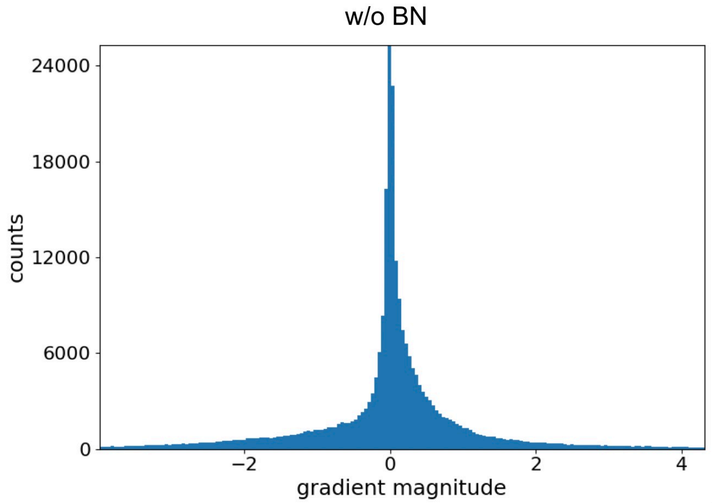
## Batch Normalization

- Couldn’t reduce covariate shift (Ilyas et al 2018)
- Allow larger learning rate
  - Constraint the gradient norm



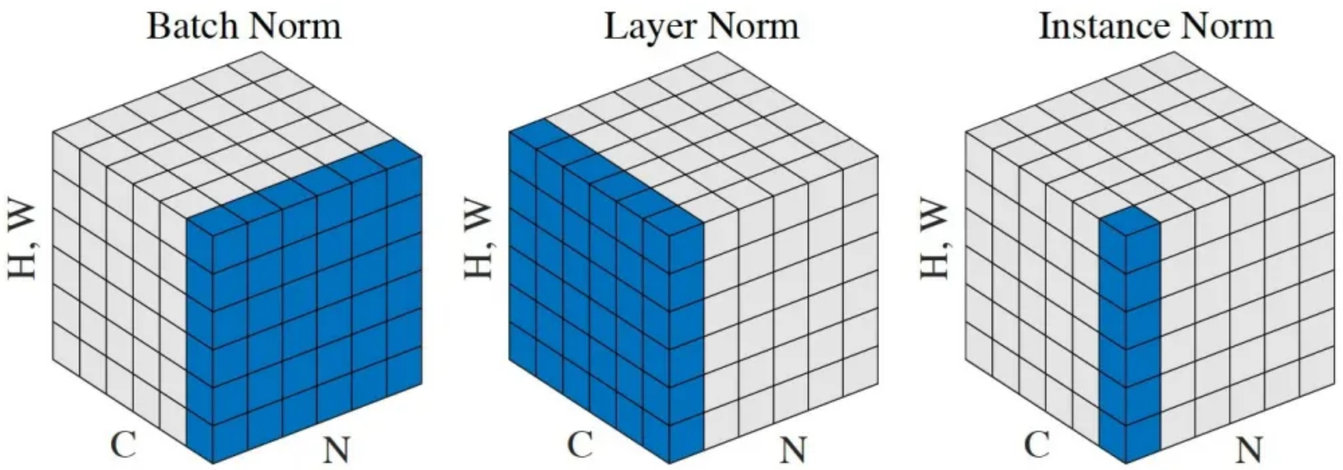



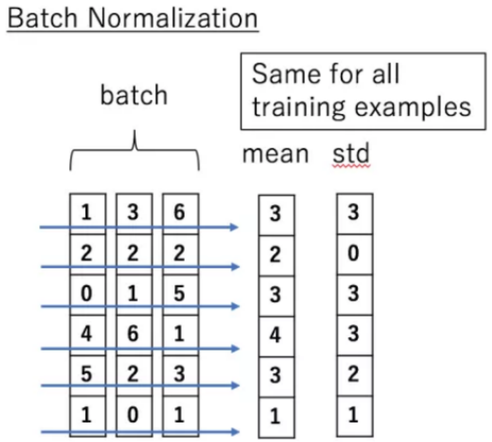



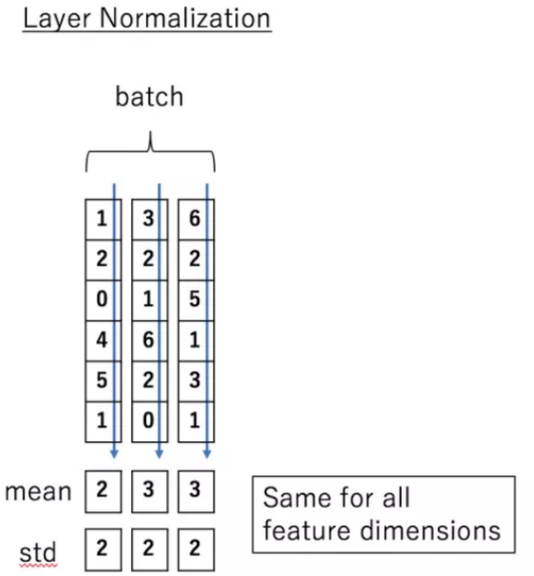



# Convolutional Neural Network

## Other normalization



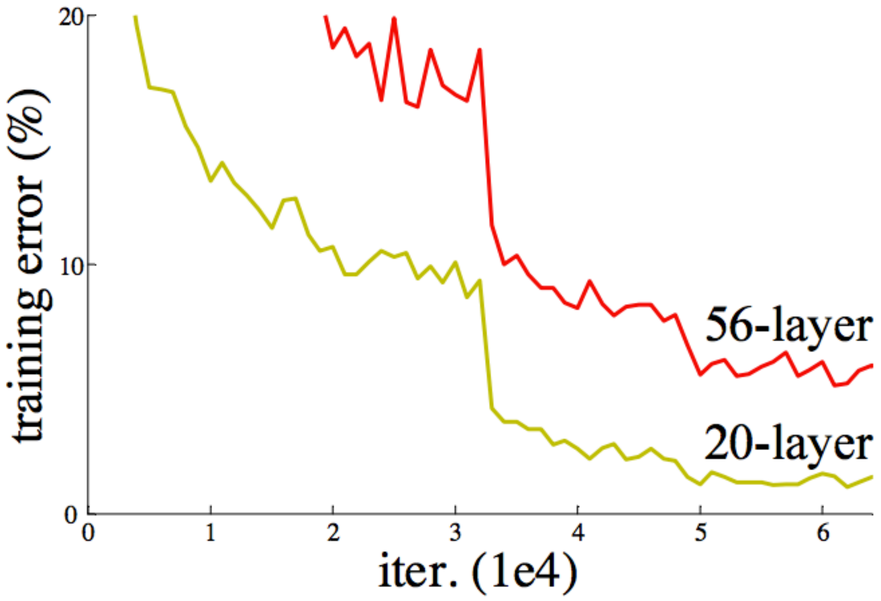



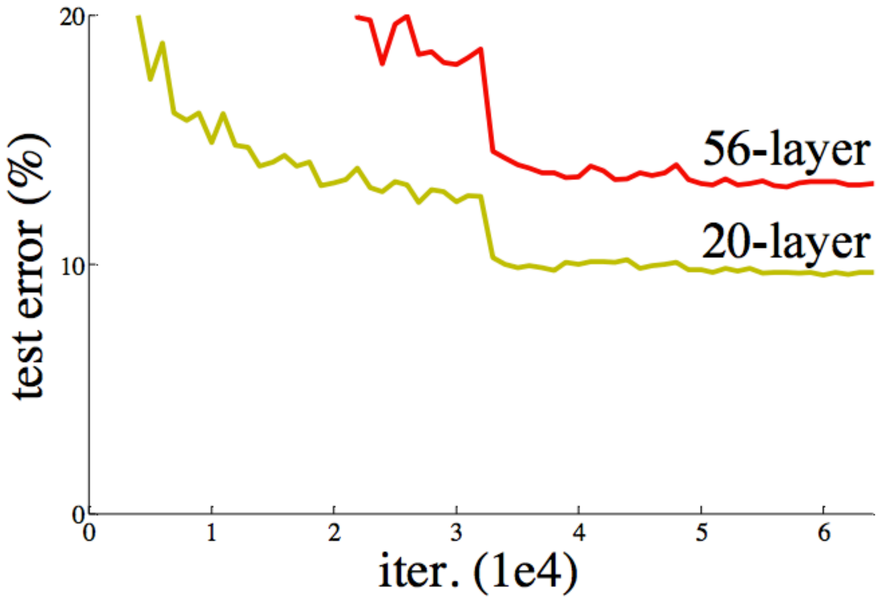



# Convolutional Neural Network

## Residual Networks

- Very deep convnets do not train well —vanishing gradient problem

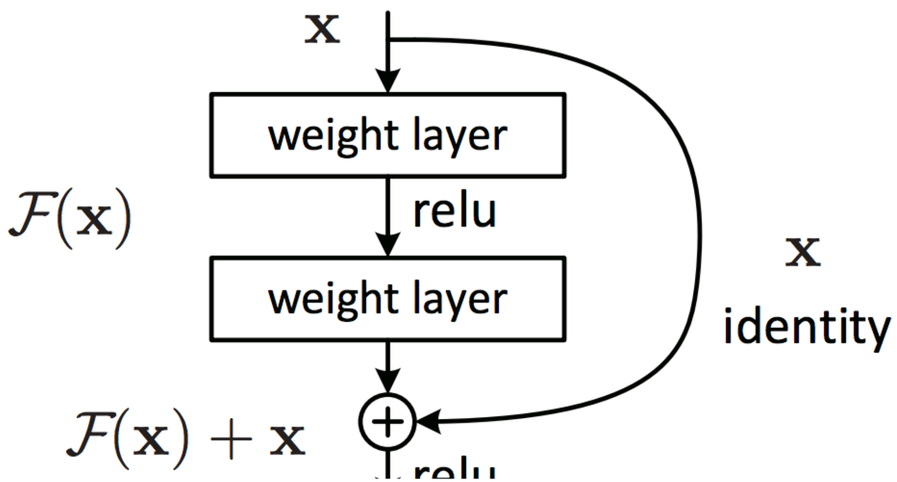


# Convolutional Neural Network

## Residual Networks

- Key idea: introduce ``pass through'' into each layer



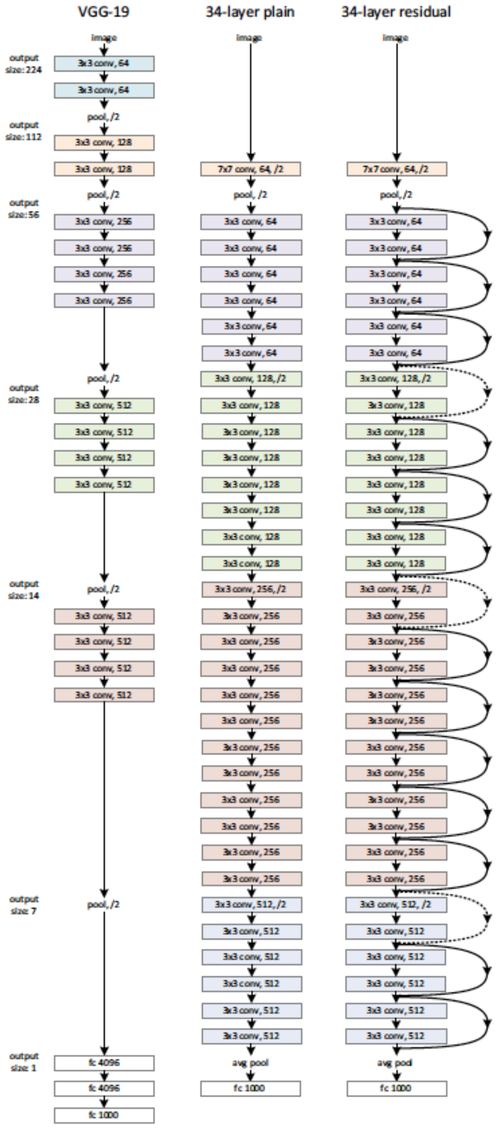

- Thus, only residual needs to be learned

# Convolutional Neural Network

## Residual Networks

| method | top-1 err. | top-5 err. |
|---|---|---|
| VGG [41] (ILSVRC'14) | - | 8.43† |
| GoogLeNet [44] (ILSVRC'14) | - | 7.89 |
| VGG [41] (v5) | 24.4 | 7.1 |
| PReLU-net [13] | 21.59 | 5.71 |
| BN-inception [16] | 21.99 | 5.81 |
| ResNet-34 B | 21.84 | 5.71 |
| ResNet-34 C | 21.53 | 5.60 |
| ResNet-50 | 20.74 | 5.25 |
| ResNet-101 | 19.87 | 4.60 |
| ResNet-152 | **19.38** | **4.49** |

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

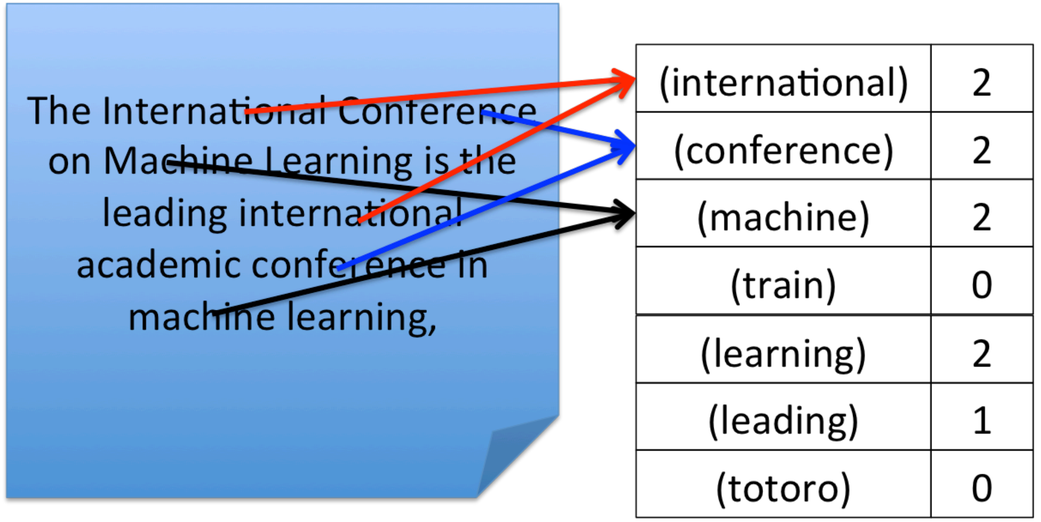


# Neural Networks for NLP

# Representation for sentence/document

## Bag of word

- A classical way to represent NLP data
- Each sentence (or document) is represented by a $d$-dimensional vector $\mathbf{x}$, where $x_i$ is number of occurrences of word $i$
- number of features = number of potential words (very large)



# Representation for sentence/document

## Feature generation for documents

- Bag of $n$-gram features ($n = 2$):
  - 10,000 words $\Rightarrow 10000^2$ potential features

The International Conference on Machine Learning is the leading international academic conference in machine learning,

| | |
|---|---|
| (international) | 2 |
| (conference) | 2 |
| (machine) | 2 |
| (train) | 0 |
| (learning) | 2 |
| (leading) | 1 |
| (totoro) | 0 |

| | |
|---|---|
| (international conference) | 1 |
| (machine learning) | 2 |
| (leading international) | 1 |
| (totoro tiger) | 0 |
| (tiger woods) | 0 |
| (international academic) | 1 |
| (international academic) | 1 |

# Representation for sentence/document

## Data Matrix (document)

- Use the bag-of-word matrix or the normalized version (TF-IDF) for a dataset (denoted by $D$):
  - tfidf(doc, word, D) $= tf$(doc, word) $\cdot\ idf$(word, D)
    - tf (doc, word): term frequency (word count in the document)/(total number of terms in the document)
    - idf (word, Dataset): inverse document frequency $\log$((Number of documents)/(Number of documents with this word))
  -

| | angeles | los | new | post | times | york |
|---|---|---|---|---|---|---|
| d1 | 0 | 0 | 1 | 0 | 1 | 1 |
| d2 | 0 | 0 | 1 | 1 | 0 | 1 |
| d3 | 1 | 1 | 0 | 0 | 1 | 0 |

tf-idf

| | angeles | los | new | post | times | york |
|---|---|---|---|---|---|---|
| d1 | 0 | 0 | 0.584 | 0 | 0.584 | 0.584 |
| d2 | 0 | 0 | 0.584 | 1.584 | 0 | 0.584 |
| d3 | 1.584 | 1.584 | 0 | 0 | 0.584 | 0 |

# Representation for sentence/document

## Bag of word + linear model

- Example: text classification (e.g., sentiment prediction, review score prediction)
- Linear model: $y \approx \text{sign}(w^T x)$ (e.g., by linear SVM/logistic regression)
- $w_i$: the ``contribution'' of each word

# Representation for sentence/document

## Bag of word + Fully connected network

- $f(x) = W_L\sigma(W_{L-1}\cdots\sigma(W_0x))$
- The first layer $W_0$ is a $d_1$ by $d$ matrix:
  - Each column $w_i$ is a $d_1$ dimensional representation of $i$-th word （word embedding )
  - $W_0x = x_1w_1 + x_2w_2 + \cdots + x_dw_d$ is a linear combination of these vectors
  - $W_0$ is also called the word embedding matrix
  - Final prediction can be viewed as an $L-1$ layer network on $W_0x$ (average of word embeddings)
- Not capturing the sequential information

# Recurrent Neural Network

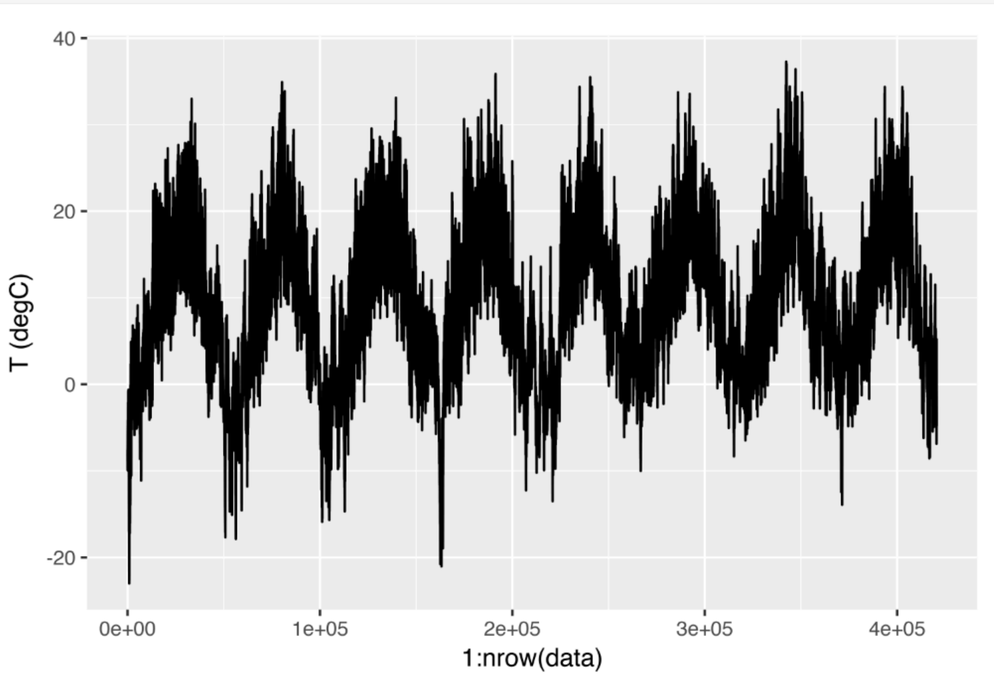
## Time series/Sequence data

- Input: $\{x_1, x_2, \cdots, x_T\}$
  - Each $x_t$ is the feature at time step $t$
  - Each $x_t$ can be a $d$-dimensional vector
- Output: $\{y_1, y_2, \cdots, y_T\}$
  - Each $y_t$ is the output at step $t$
  - Multi-class output or Regression output:
    - $y_t \in \{1,2,\cdots, L\}$ or $y_t \in \mathbb{R}$

# Recurrent Neural Network

## Example: Time Series Prediction

- Climate Data:
  - $x_t$: temperature at time $t$
  - $y_t$: temperature (or temperature change) at time $t + 1$
- Stock Price: Predicting stock price



# Recurrent Neural Network

## Example: Language Modeling

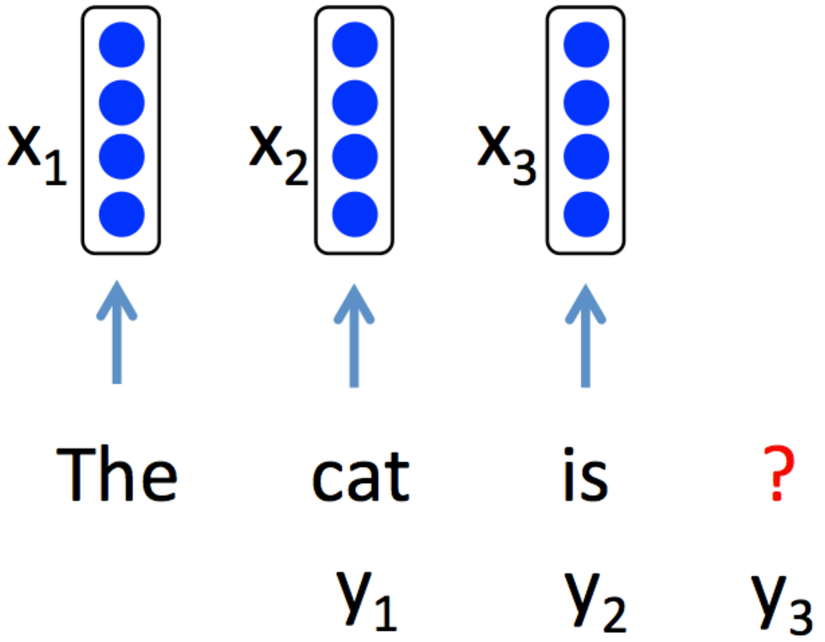
The cat is ?

# Recurrent Neural Network

## Example: Language Modeling

The cat is ?

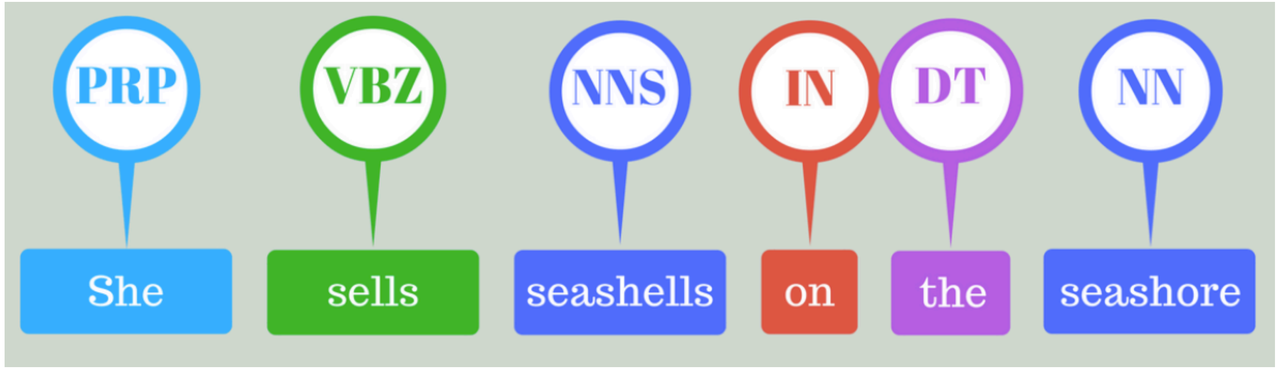
- $x_t$: one-hot encoding to represent the word at step $t$ ($[0,\ldots,0,1,0,\ldots,0]$)
- $y_t$: the next word
  - $y_t \in \{1,\cdots,V\}$ V: Vocabulary size



# Recurrent Neural Network

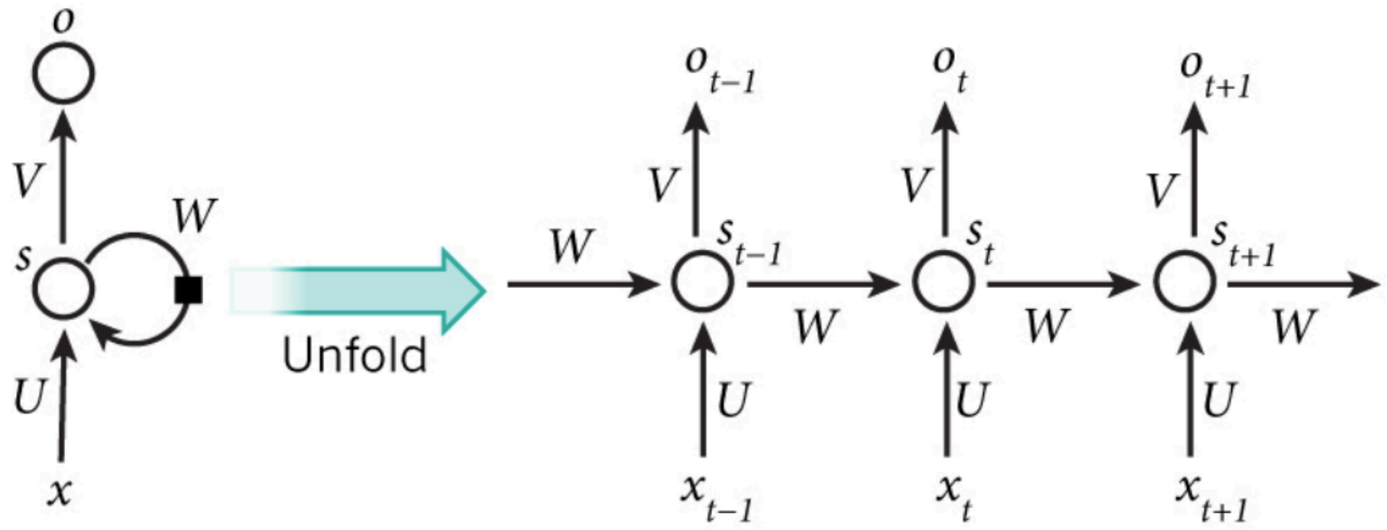
## Example: POS Tagging

- Part of Speech Tagging:
  - Labeling words with their Part-Of-Speech (Noun, Verb, Adjective, …)
  - $x_t$: a vector to represent the word at step $t$
  - $y_t$: label of word $t$



# Recurrent Neural Network

## Example: POS Tagging




- $x_t$: $t$-th input
- $s_t$: hidden state at time $t$ (`memory'' of the network)
  - $s_t = f(Ux_t + Ws_{t-1})$
  - $W$: transition matrix, $U$: word embedding matrix, $s_0$ usually set to be 0
- Predicted output at time $t$:
  - $o_t = \arg\max_i (Vs_t)_i$

# Recurrent Neural Network

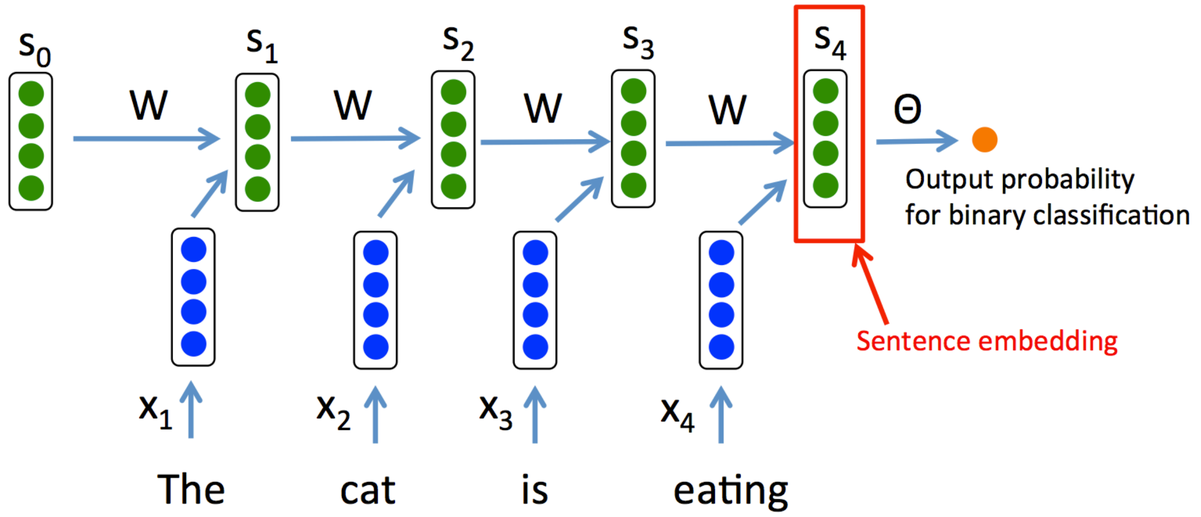
## Recurrent Neural Network (RNN)

- Training: Find $U, W, V$ to minimize empirical loss:
- Loss of a sequence:
  - $\sum_{t=1}^{T} \text{loss}(Vs_t, y_t)$
  - ($s_t$ is a function of $U, W, V$)
- Loss on the whole dataset:
  - Average loss over all sequences
- Solved by SGD/Adam

# Recurrent Neural Network

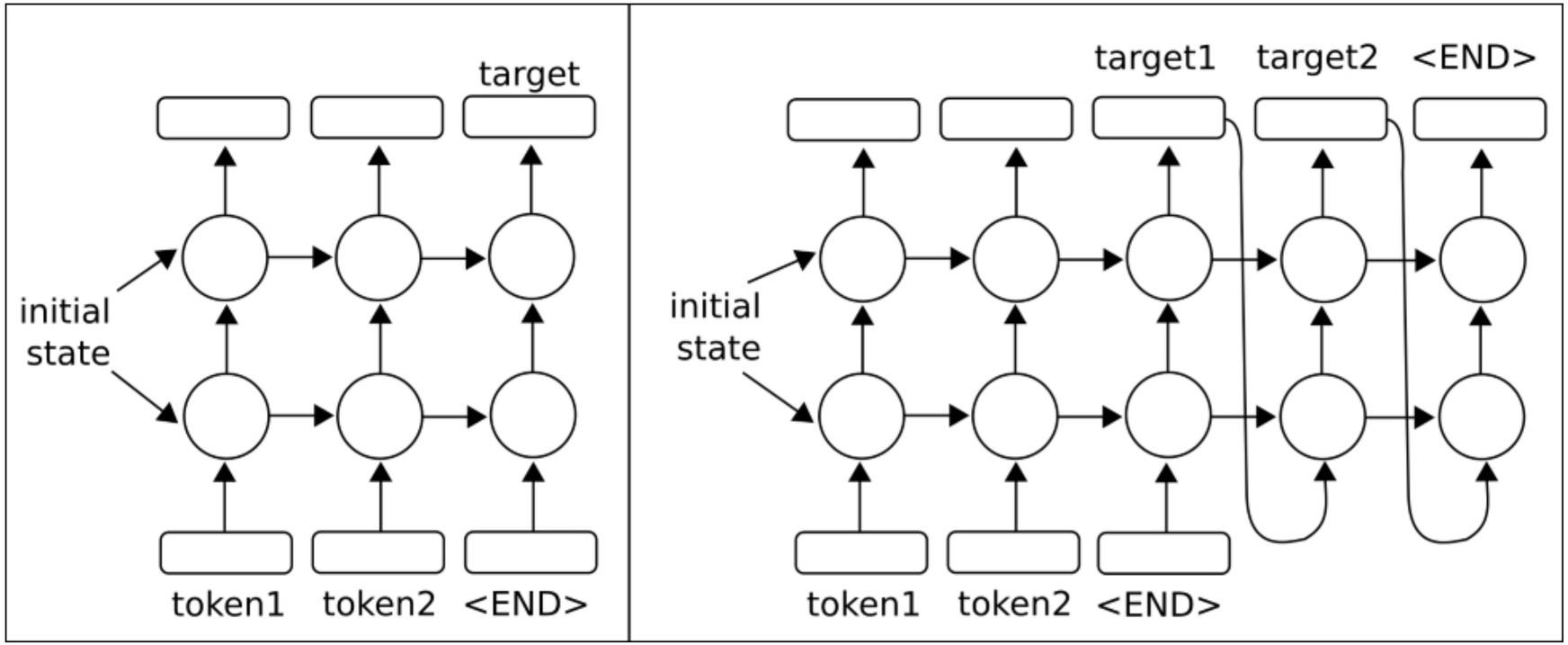
## RNN: Text Classification

- Not necessary to output at each step
- Text Classification:
  - sentence → category
  - Output only at the final step
- Model: add a fully connected network to the final embedding



# Recurrent Neural Network

## Multi-layer RNN



# Recurrent Neural Network

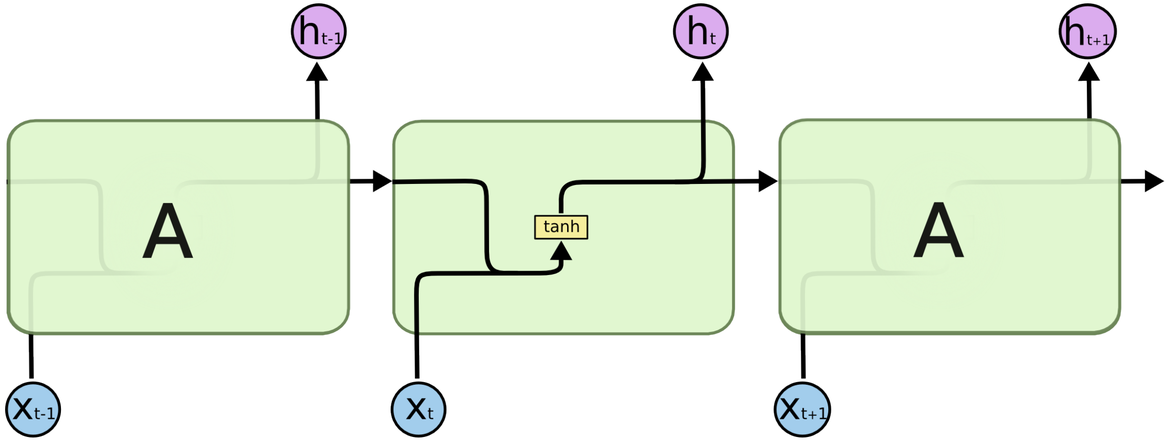
## Problems of Classical RNN

- Hard to capture long-term dependencies
- Hard to solve (vanishing gradient problem)
- Solution:
  - LSTM (Long Short Term Memory networks)
  - GRU (Gated Recurrent Unit)
  - …

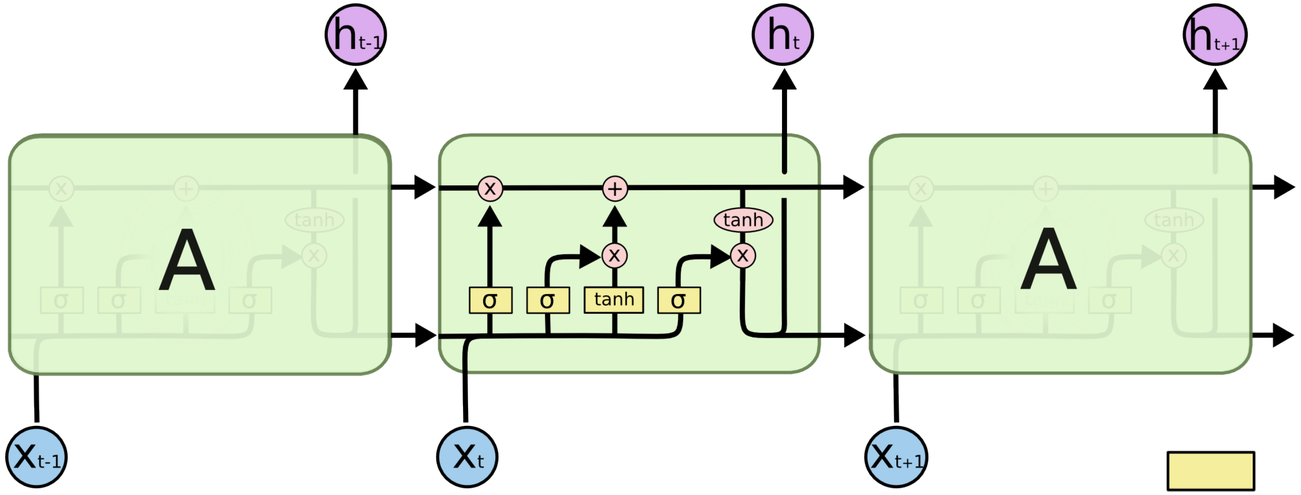
# Recurrent Neural Network

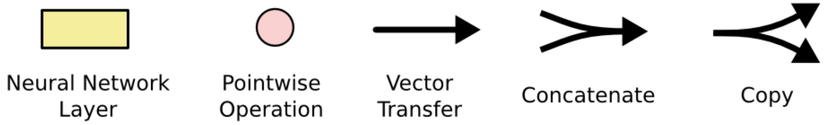
## LSTM

- RNN:




- LSTM:

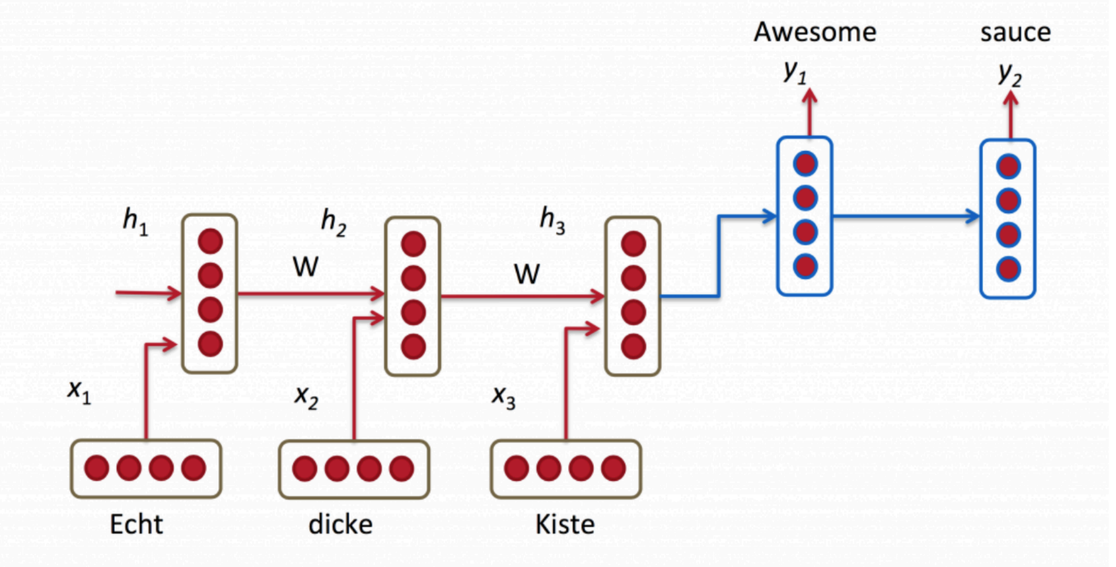


# Recurrent Neural Network

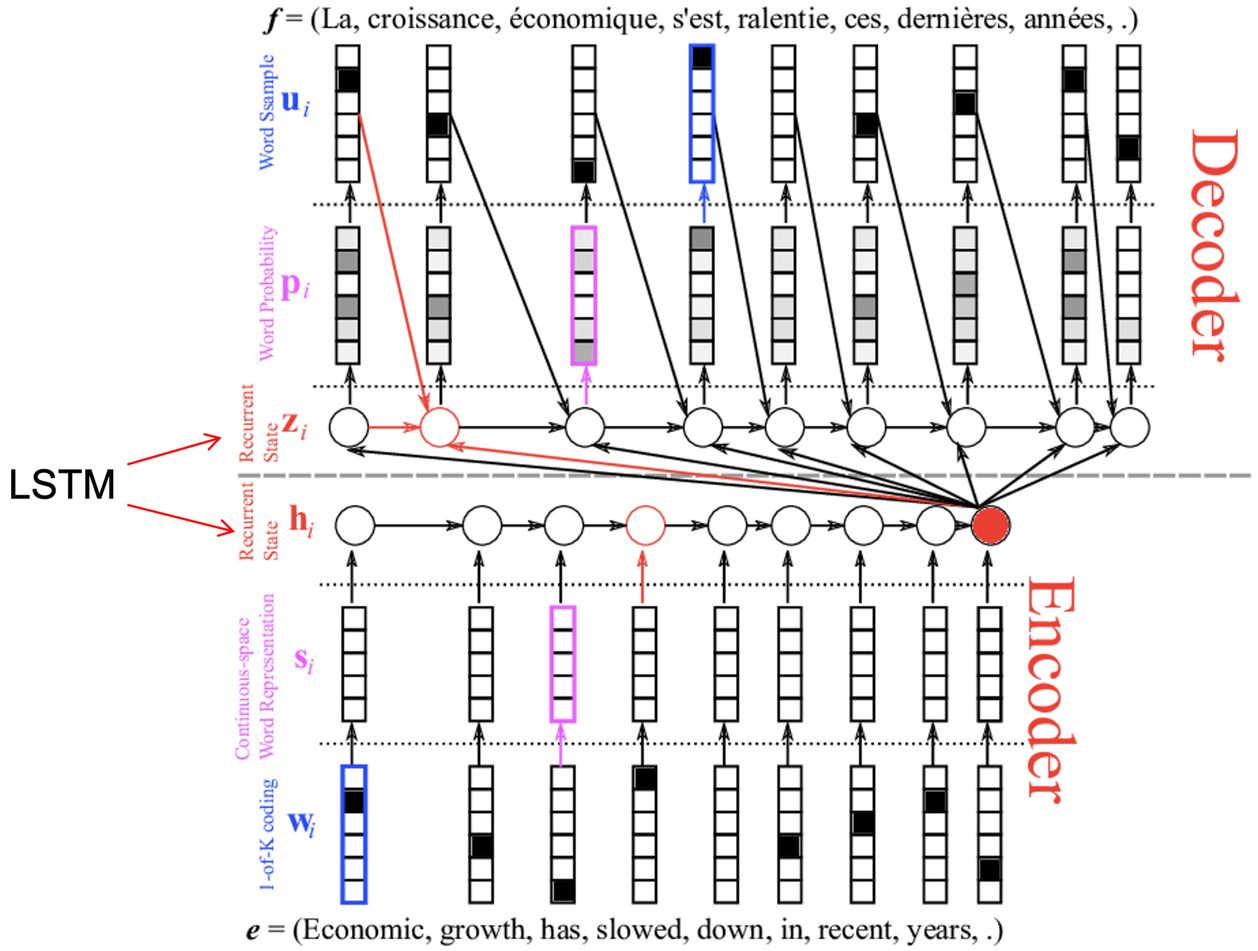
## Neural Machine Translation (NMT)

- Out the translated sentence from an input sentence
- Training data: a set of input-output pairs (supervised setting)
- Encoder-decoder approach:
  - Encoder: Use (RNN/LSTM) to encode the input sentence input a latent vector
  - Decoder: Use (RNN/LSTM) to generate a sentence based on the latent vector



# Recurrent Neural Network

## Neural Machine Translation



# Recurrent Neural Network

## Attention in NMT

- Usually, each output word is only related to a subset of input words (e.g., for machine translation)
- Let $u$ be the current decoder latent state, $v_1, \ldots, v_n$ be the latent sate for each input word
- Compute the weight of each state by
  - $p = \text{Softmax}(u^T v_1, \ldots, u^T v_n)$
- Compute the context vector by $Vp = p_1 v_1 + \ldots + p_n v_n$

# Recurrent Neural Network

## Attention in NMT