

Mixed-mode Federated Learning Cloud-based System

Partha Pratim Saha

ID No. : 2019AD04100

Email Address : 2019AD04100@wilp.bits-pilani.ac.in

Federated Learning

- A machine learning technique that collaboratively train a learning model centrally by keeping data decentralized.
- Enables multiple participants to build a common global model without sharing their entire datasets, thus addressing data privacy, security, and access rights to their data.

Problem Statement

Background: Many participants are involved in a traditional Federated Learning model

Option 1: Share and pool all the data in a central location (Centralized Learning)

Problem 1: Participants don't trust each other, to share their data sets

Option 2: Data stays local with each participant, and code travels (Decentralized Learning)

Problem 2: Significant performance impact

Our Proposal: Partition data in 2 parts: keep private data as local and share the rest as public

Benefit: Optimize based on security and performance trade-offs

Centralized Learning

- The central server receives the dataset contributions from all clients
- Here one time data copy, multiple executions on those data to prepare the global model at server side
- Performance is limited to one-time movement of the data
- Subsequent multiple iterations done centrally to perform computational operations on those data
- Better performance but has security concerns

Decentralized/Distributed Learning

- Program has to travel to central server once for each iteration
- Performance is limited by multiple movements of the program code
- Lower performance due to high time complexity
- Better data sharing security

Example - 3 hospitals with single central server

t_{da} = data copying delays from Hospital A to central server

t_{db} = data copying delays from Hospital B to central server

t_{dc} = data copying delays from Hospital C to central server

t_{pa} = time for code and weights of Neural Network to travel from central server to hospital A

t_{pb} = time for code and weights of Neural Network to travel from central server to hospital B

t_{pc} = time for code and weights of Neural Network to travel from central server to hospital C

t_{px} = program execution time

So, total data copy time to central database is:

$$t_{da} + t_{db} + t_{dc}$$

and in a completely centralized model, total run time will be:

$$T1 = t_{da} + t_{db} + t_{dc} + n * t_{px}$$

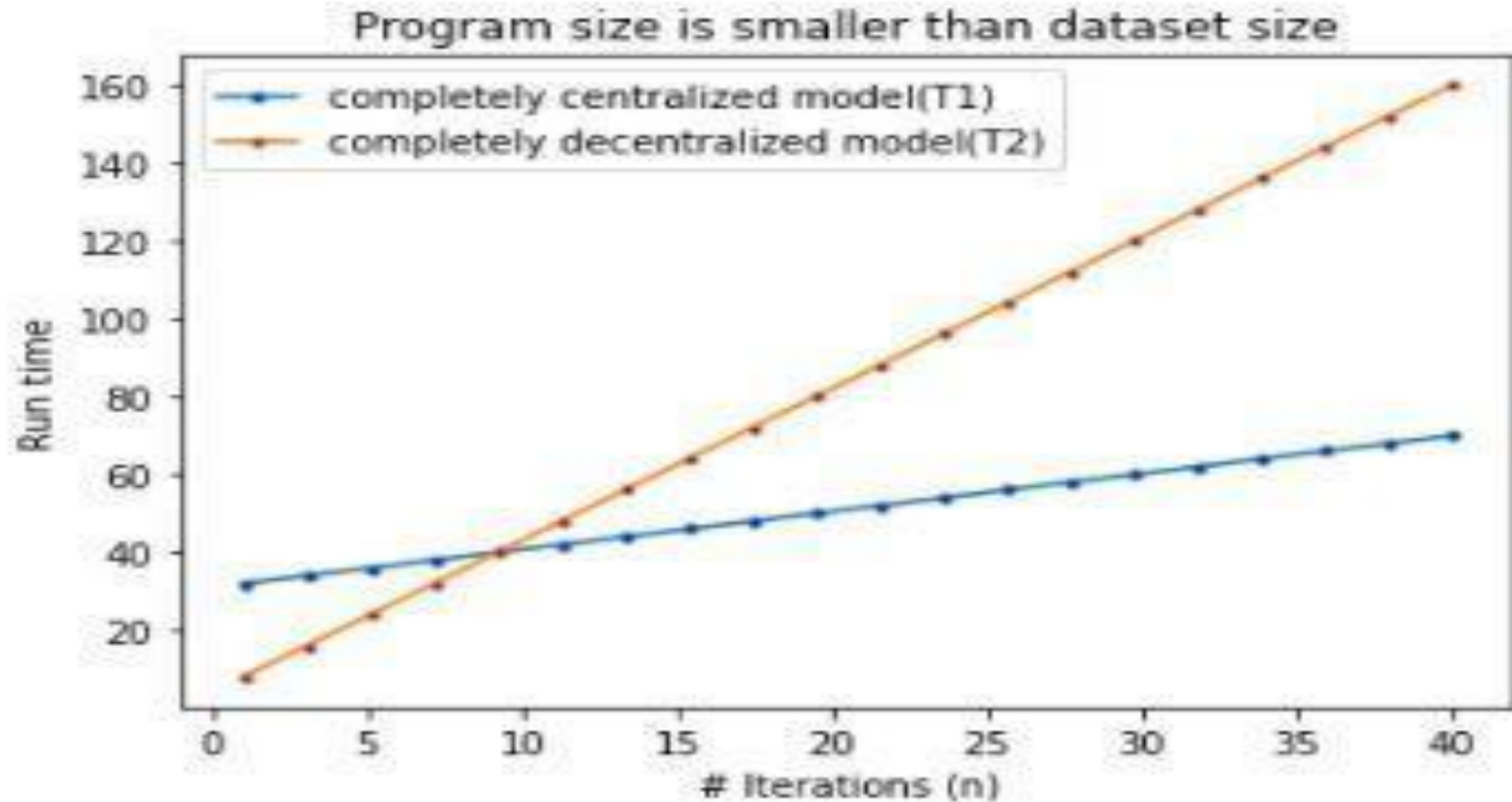
For a fully decentralized Federated learning system, total run time will be:

$$T2 = n * (t_{pa} + t_{pb} + t_{pc} + t_{px})$$

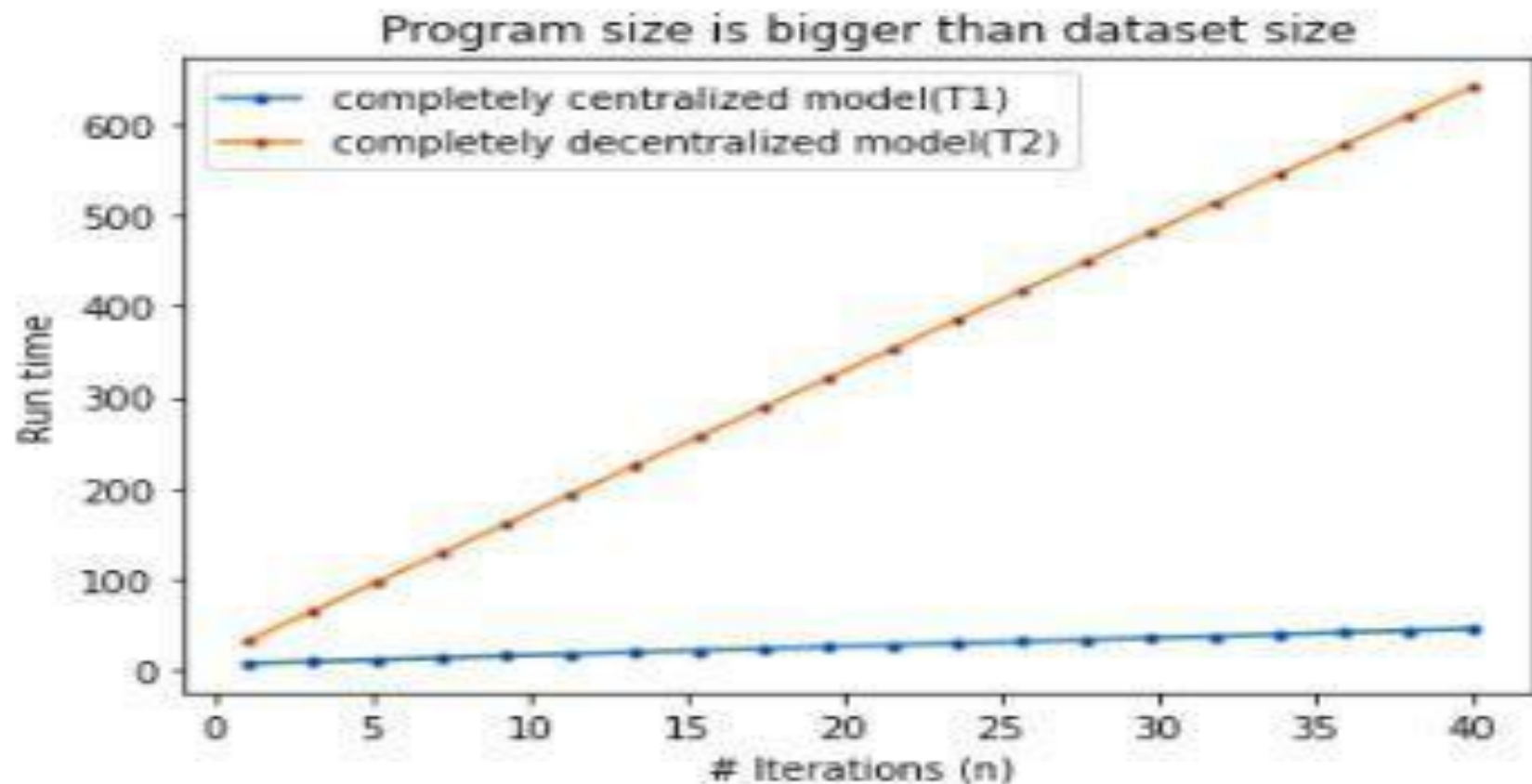
Contd..

- For larger values of n , $T_2 \gg T_1$, because in the first model we copy data only once, whereas in the second case, the code travel once for each iteration.
- In a single iteration for a large program, if $tpa > tda$, and $tpb > tdb$, and $tpc > tdc$ then $T_2 > T_1$
- For smaller programs(small tp 's) and larger dataset(large td 's), since model training is done iteratively, n may be 10 or higher, then also $T_2 > T_1$
- If the program binary is only $1/10$ -th size of the dataset. When very few iterations are required for the training program, then $T_1 > T_2$

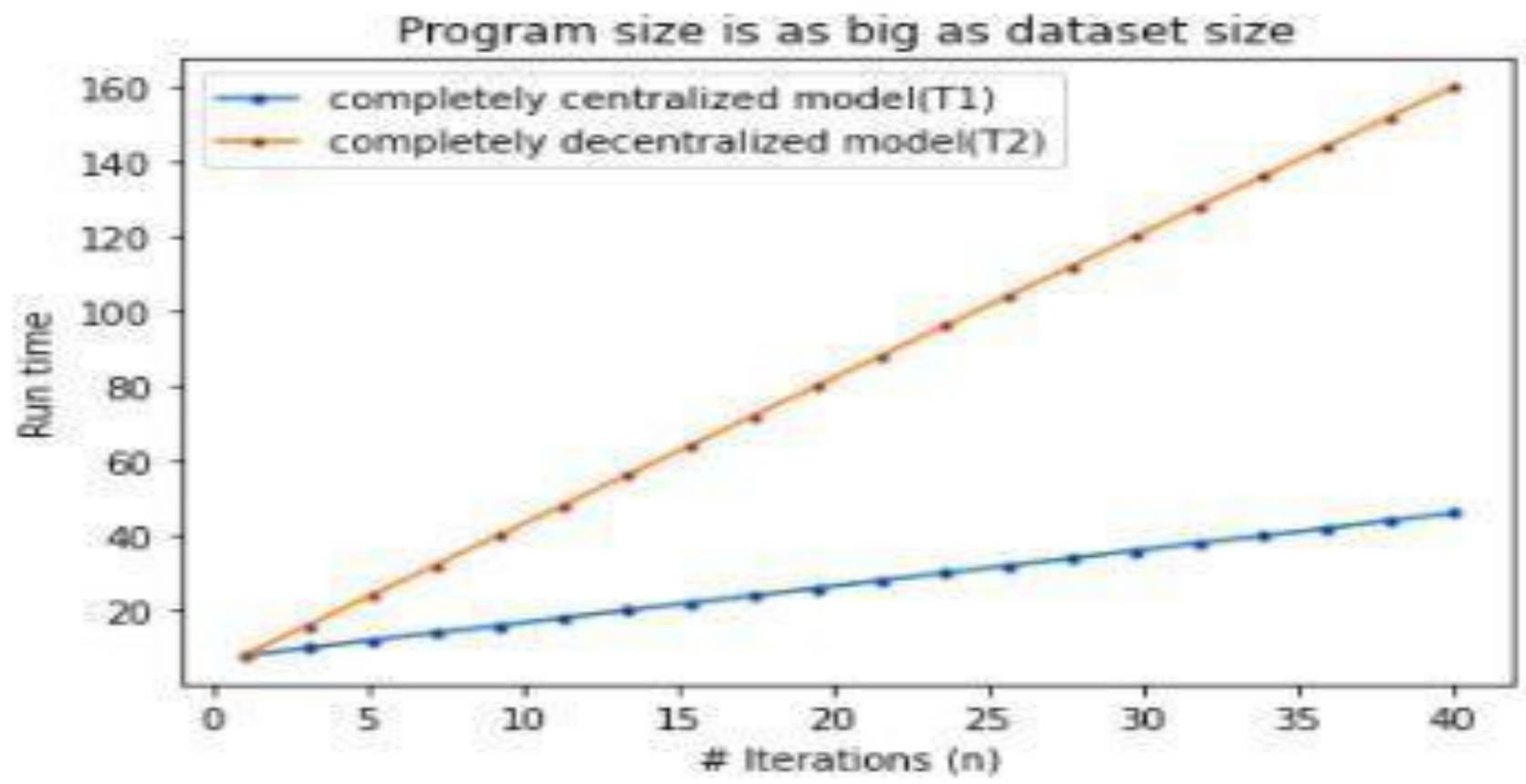
Smaller program than dataset - decentralized dominates($n \geq 10$)



Bigger program than dataset - decentralized dominates

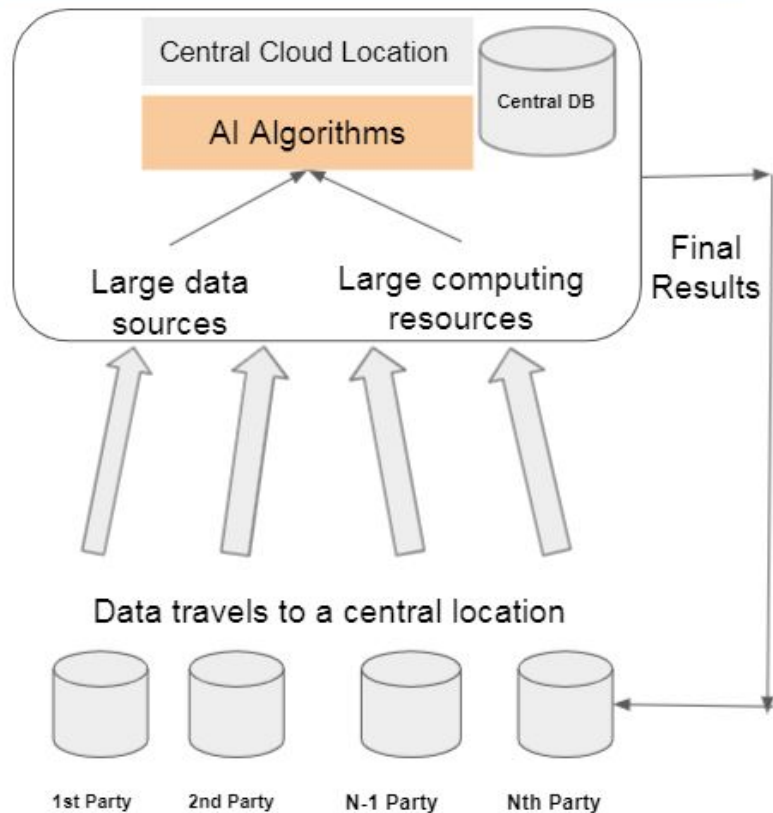


Equal size of program and dataset - decentralized dominates

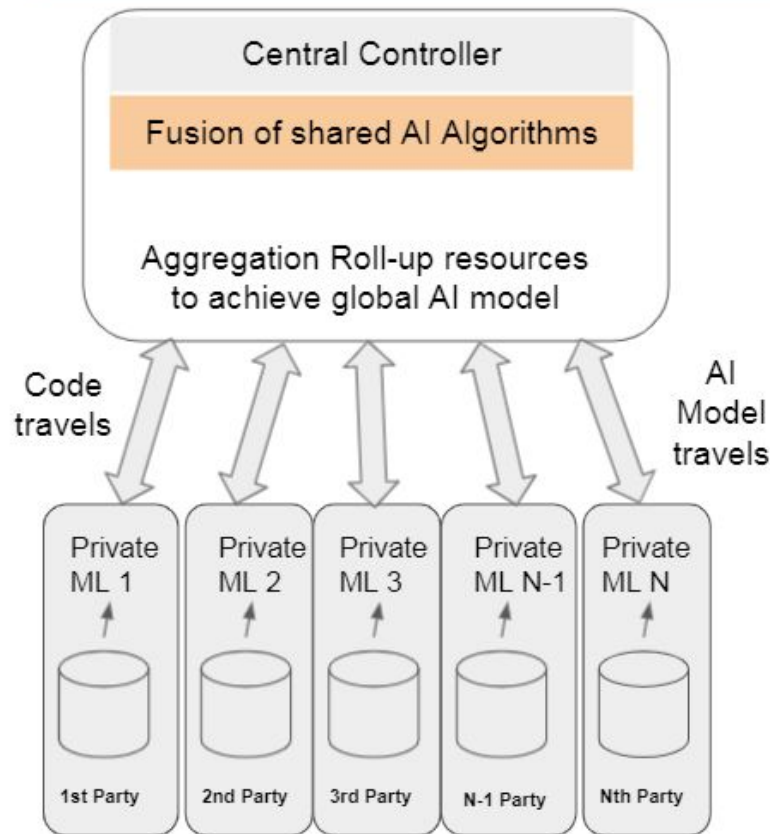


Centralized vs Decentralized Machine Learning Architecture

Centralized machine learning architecture



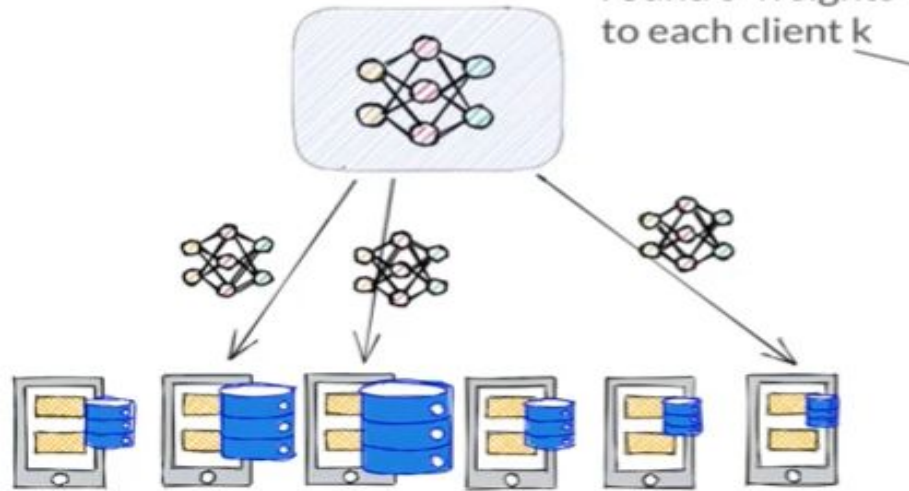
Decentralized machine learning architecture



Literature Reviews and recent works - FedAvg algorithm

so for the teeth round the weight is denoted by

FedAvg



Algorithm 1 FederatedAveraging. The K clients are indexed by k ; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:

```
initialize  $w_0$ 
for each round  $t = 1, 2, \dots$  do
   $m \leftarrow \max(C \cdot K, 1)$ 
   $S_t \leftarrow$  (random set of  $m$  clients)
  for each client  $k \in S_t$  in parallel do
     $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$ 
   $w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k$ 
```

ClientUpdate(k, w): // Run on client k
 $B \leftarrow$ (split \mathcal{P}_k into batches of size B)
 for each local epoch i from 1 to E do
 for batch $b \in B$ do
 $w \leftarrow w - \eta \nabla \ell(w; b)$
 return w to server

Before start, algo started to sample C clients from all K clients. In this case, there are t rounds, in each round $K=6$, $C=0.5$, it send current round weights to all client. For t -th round, the weight is $w_{\{t\}}$ and then client start SGD on their local data and return to server. Once the server receives, then securely aggregate them and take weighted average and repeat. FedAvg gives weight to devices by the amount of data each device own, so this FedAvg favours clients with more data.

The Federated Averaging algorithm

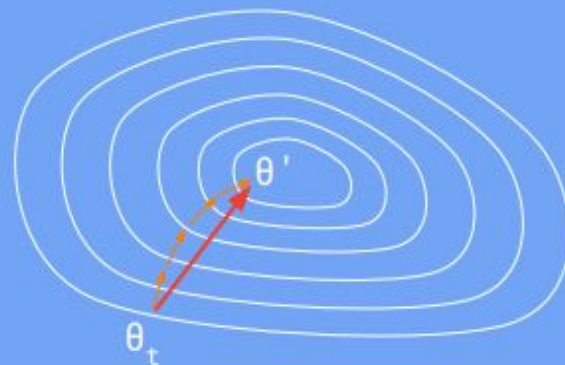
Server

Until Converged:

1. Select a random subset of clients
2. In parallel, send current parameters θ_t to those clients

Selected Client k

1. Receive θ_t from server.
2. Run some number of minibatch SGD steps, producing θ'
3. Return $\theta' - \theta_t$ to server.



3. $\theta_{t+1} = \theta_t + \text{data-weighted average of client updates}$

Literature Reviews and recent works - FedReconstruction algo on local param

Algorithm 1 Federated Reconstruction Training

Input: set of global parameters \mathcal{G} , set of local parameters \mathcal{L} , dataset split function S , reconstruction algorithm R , client update algorithm U

Server executes:

$g^{(0)} \leftarrow (\text{initialize } \mathcal{G})$

for each round t **do**

$\mathcal{S}^{(t)} \leftarrow (\text{randomly sample } m \text{ clients})$

for each client $i \in \mathcal{S}^{(t)}$ **in parallel do**

$(\Delta_i^{(t)}, n_i) \leftarrow \text{ClientUpdate}(i, g^{(t)})$

end for

$n = \sum_{i \in \mathcal{S}^{(t)}} n_i$

$g^{(t+1)} \leftarrow g^{(t)} + \eta_s \sum_{i \in \mathcal{S}^{(t)}} \frac{n_i}{n} \Delta_i^{(t)}$

end for

ClientUpdate:

$(\mathcal{D}_{i,s}, \mathcal{D}_{i,q}) \leftarrow S(\mathcal{D}_i)$

$l_i^{(t)} \leftarrow R(\mathcal{D}_{i,s}, \mathcal{L}, g^{(t)})$

$g_i^{(t)} \leftarrow U(\mathcal{D}_{i,q}, l_i^{(t)}, g^{(t)})$

$\Delta_i^{(t)} \leftarrow g_i^{(t)} - g^{(t)}$

$n_i \leftarrow |\mathcal{D}_{i,q}|$

return $\Delta_i^{(t)}, n_i$ to the server

Contd... Recent works : FedProx algorithm

or proximal term that
penalizes large changes in weights

FedProx

Replace each client's loss $F_k(w)$ with the following objective instead

$$\min_w h_k(w; w^t) = F_k(w) + \frac{\mu}{2} \|w - w^t\|^2$$

Proximal term
(hence FedProx)

It penalizes large change of weights that helps convergence for highly heterogeneous data. This proximal term, hence FedProximal or in short FedProx, as penalizes the model from changing too much on one single device. We can control the amount of penalization by the hyper param μ .

Contd... Recent works : q-FedAvg algorithm

we penalize **worse performing devices**
more



q-FedAvg

$$\min_w f_q(w) = \sum_{k=1}^m \frac{p_k}{q+1} F_k^{q+1}(w),$$

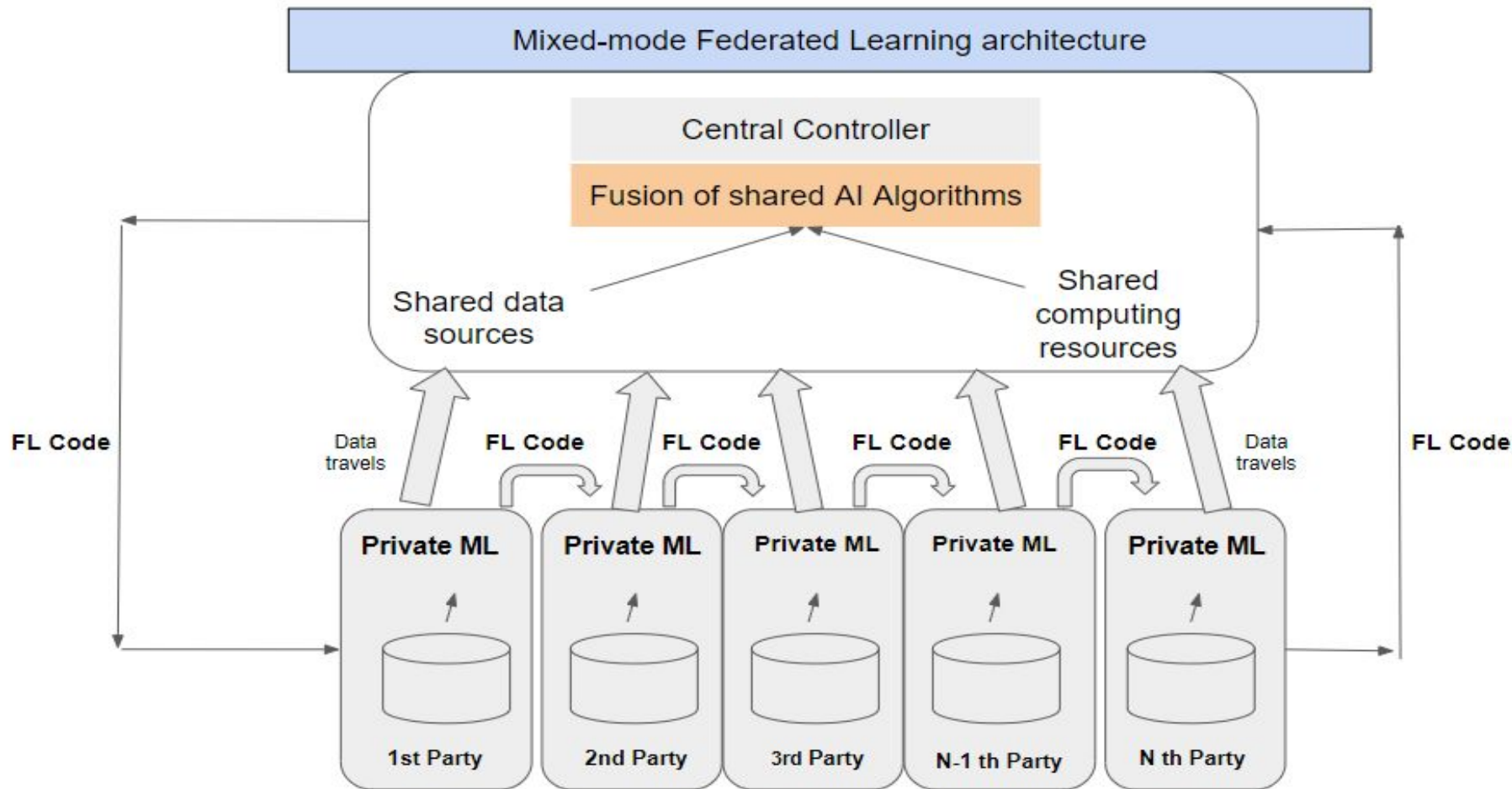
q= penalizing factor

larger q ,more the worst performing device dominate the overall loss so more FAIR it becomes. Thus, the federated learning become keeping fairness during training

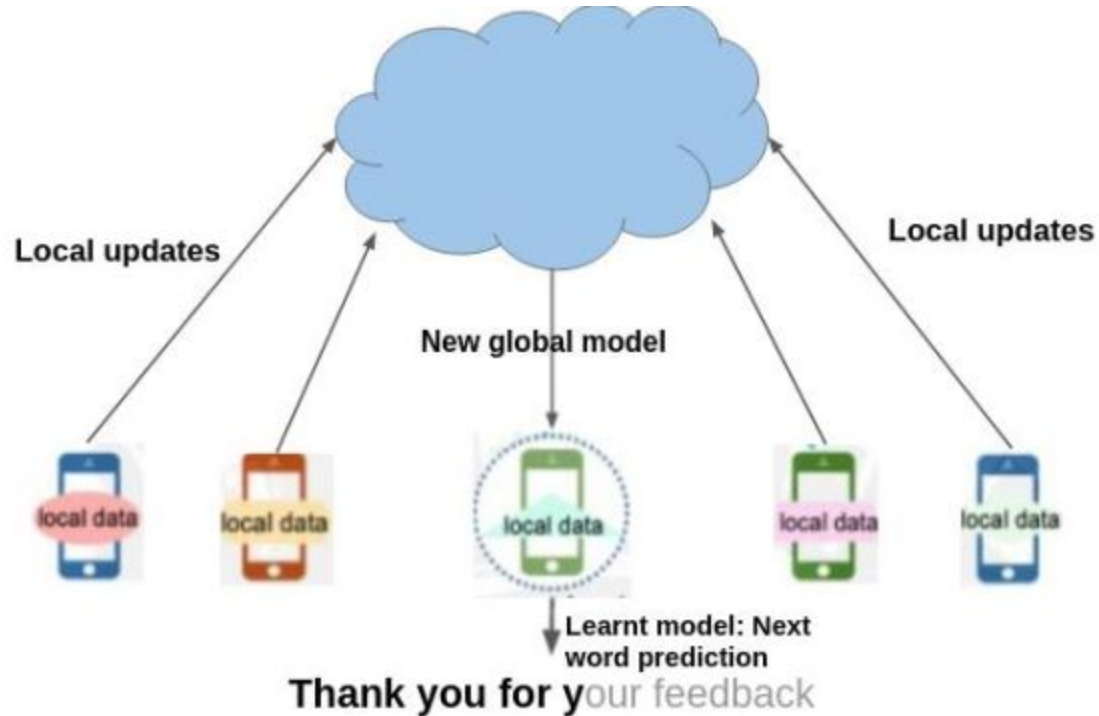
Mixed-mode Federated Learning

- Combination of centralized and decentralized machine learning
- **On Security:** only sharable data will be available to others for the model execution
- **On Performance:** Model becomes more robust by visiting on multi-parties datasets thus creating global model

Mixed-model FL = Centralized + Decentralized Learning

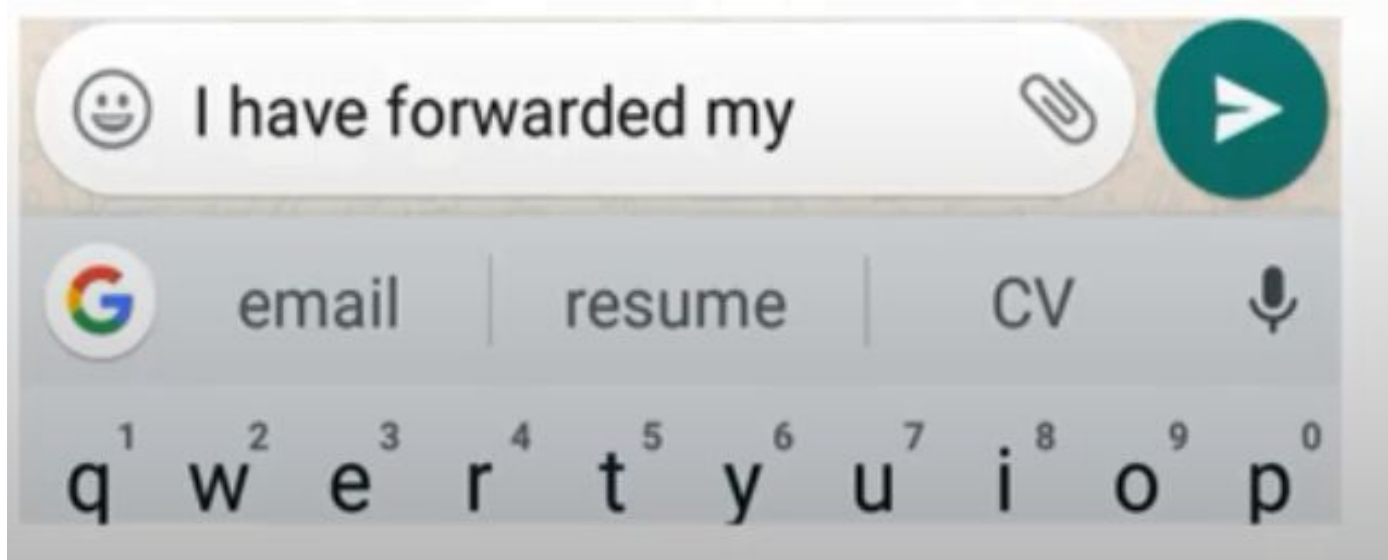


Application: Next word prediction

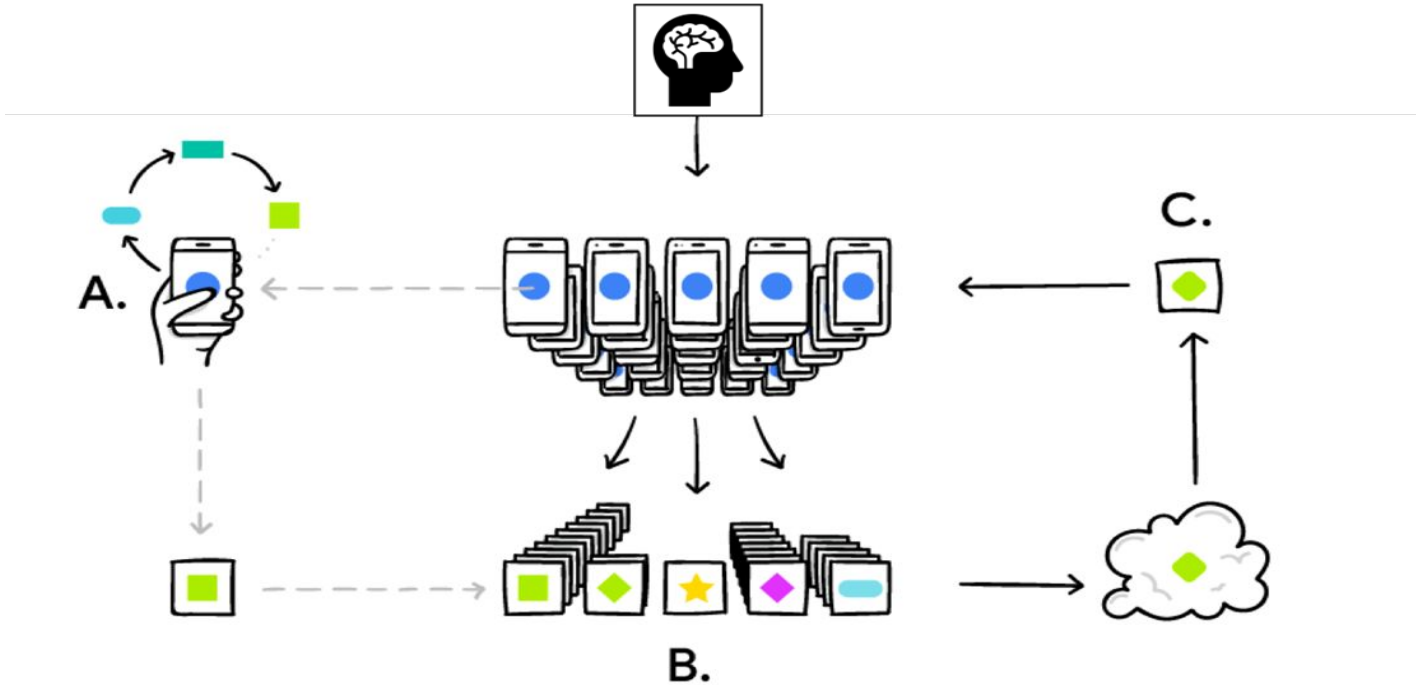


3 words suggestions in mobile

The center position in the suggestion strip is reserved for highest-probability, candidate, while the second and third most likely candidates occupy the left and right positions, respectively.

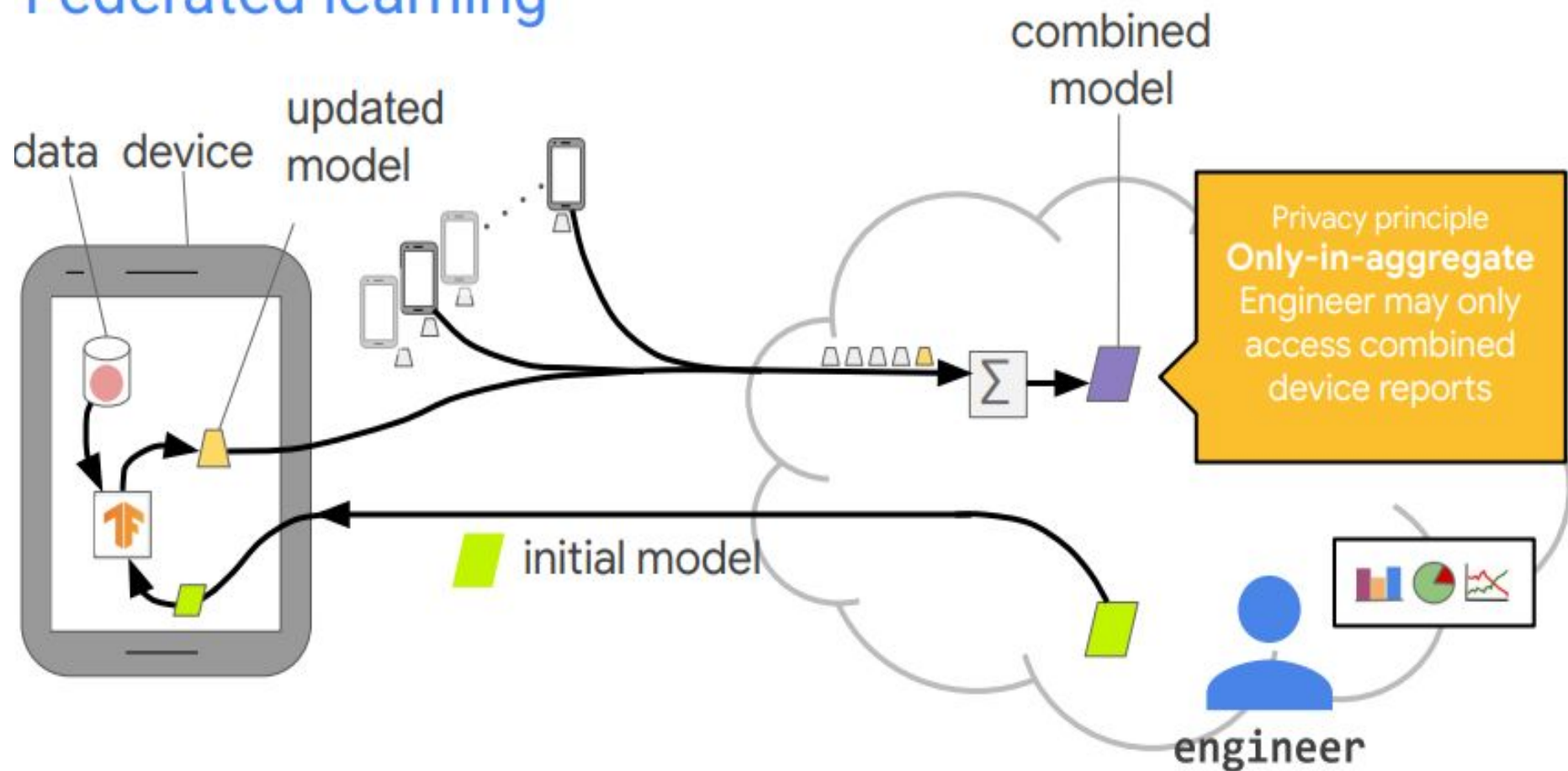


Mobile Device Personalization

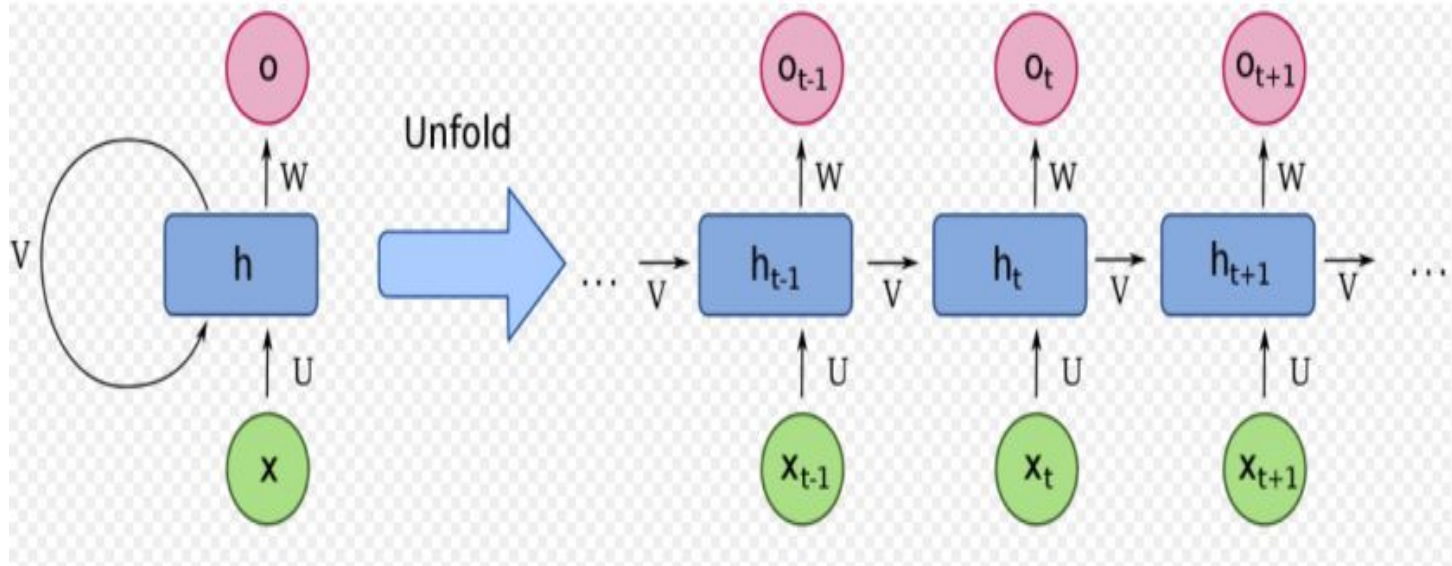


Mobile phone personalizes the model locally, based on the usage (A). Many users' updates are aggregated (B) to form a consensus change (C) to the shared model, after which the procedure is repeated.

Federated learning



Models used in Mixed mode FL project

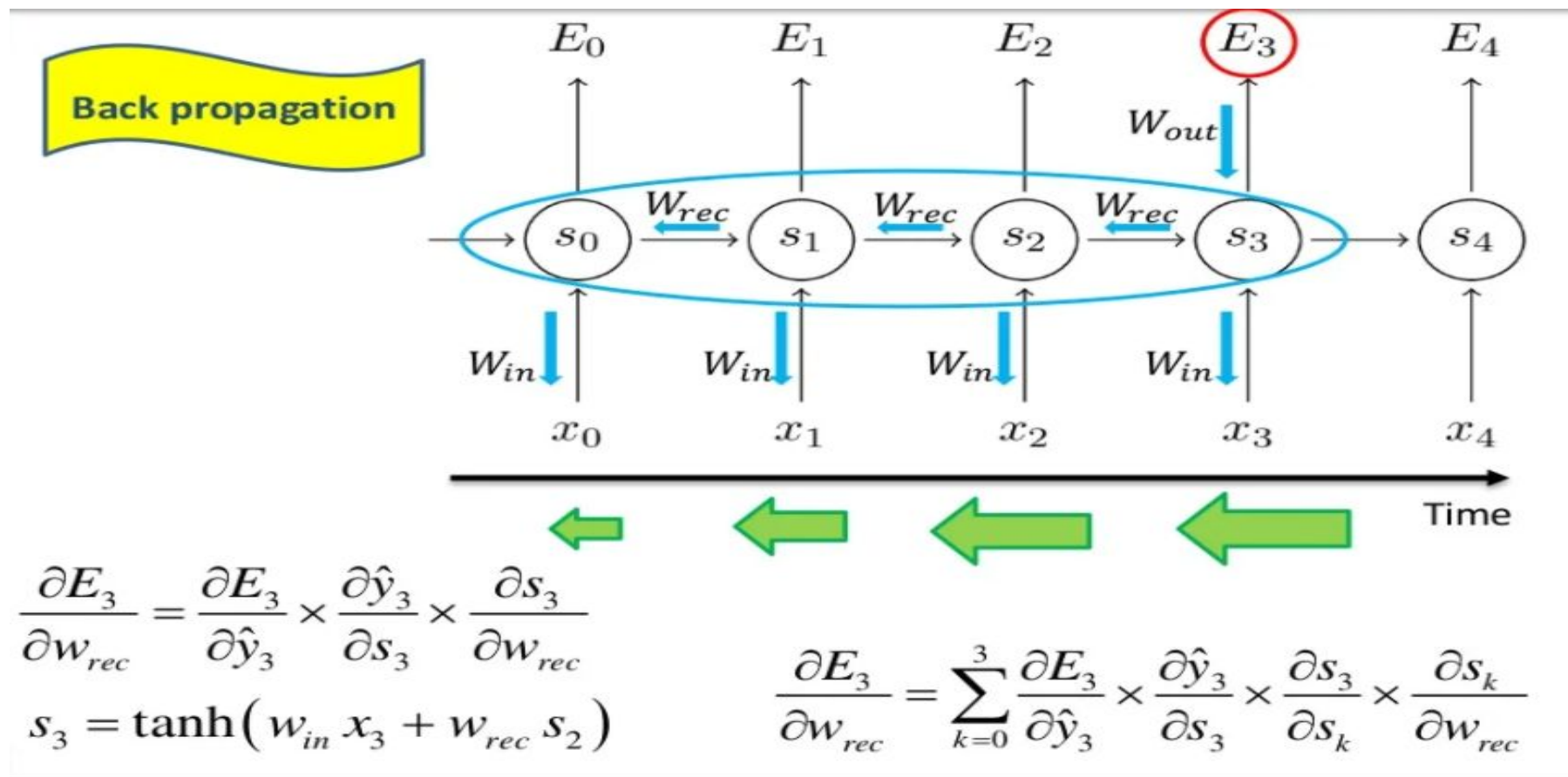


- **RNN**
- **LSTM**
- **BiLSTM**
- **Word2Vec**

-> In sequence to sequence learning, when we unroll a sequence through a timed network, it will look like the same structure will follow in multiple levels of the network. So it is called the Recurrent structure of neural Networks

Issue in RNN: Long Range Dependencies, Vanishing Gradient and Exploding Gradient

Vanishing Gradient in RNN



Vanishing Gradient contd...

$$s_3 = \tanh(w_{in} x_3 + w_{rec} s_2) \quad (\tanh)' \in [0,1]$$

$$\frac{\partial E_3}{\partial w_{rec}} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \times \frac{\partial \hat{y}_3}{\partial s_3} \times \frac{\partial s_3}{\partial s_k} \times \frac{\partial s_k}{\partial w_{rec}}$$

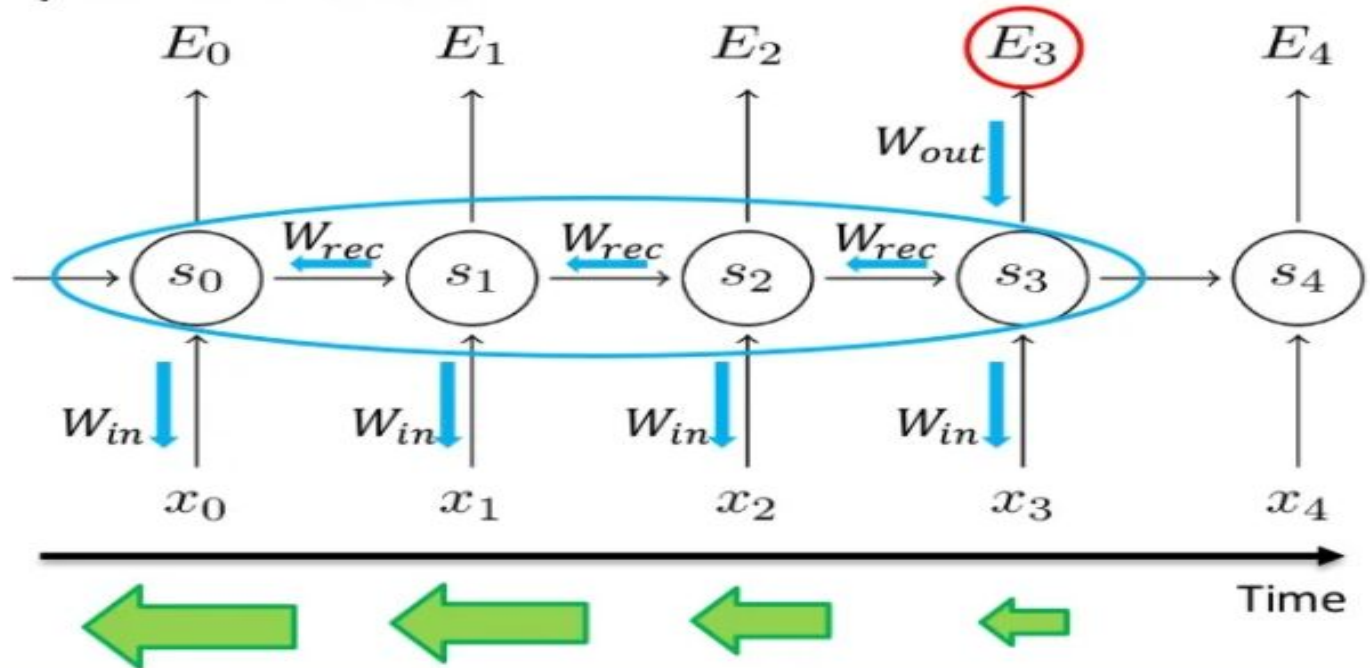
$$\frac{\partial E_3}{\partial w_{rec}} = \frac{\partial E_3}{\partial \hat{y}_3} \times \frac{\partial \hat{y}_3}{\partial s_3} \times \frac{\partial s_3}{\partial s_2} \times \frac{\partial s_2}{\partial w_{rec}} + \frac{\partial E_3}{\partial \hat{y}_3} \times \frac{\partial \hat{y}_3}{\partial s_3} \times \frac{\partial s_3}{\partial s_1} \times \frac{\partial s_1}{\partial w_{rec}} + \frac{\partial E_3}{\partial \hat{y}_3} \times \frac{\partial \hat{y}_3}{\partial s_3} \times \frac{\partial s_3}{\partial s_0} \times \frac{\partial s_0}{\partial w_{rec}}$$

$\frac{\partial s_3}{\partial s_2} \times \frac{\partial s_2}{\partial s_1}$
 $\frac{\partial s_3}{\partial s_2} \times \frac{\partial s_2}{\partial s_1} \times \frac{\partial s_1}{\partial s_0}$

$$\frac{\partial E_3}{\partial w_{rec}} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \times \frac{\partial \hat{y}_3}{\partial s_3} \times \left(\prod_{j=k+1}^3 \frac{\partial s_j}{\partial s_{j-1}} \right) \times \frac{\partial s_k}{\partial w_{rec}}$$

Exploding Gradient in RNN

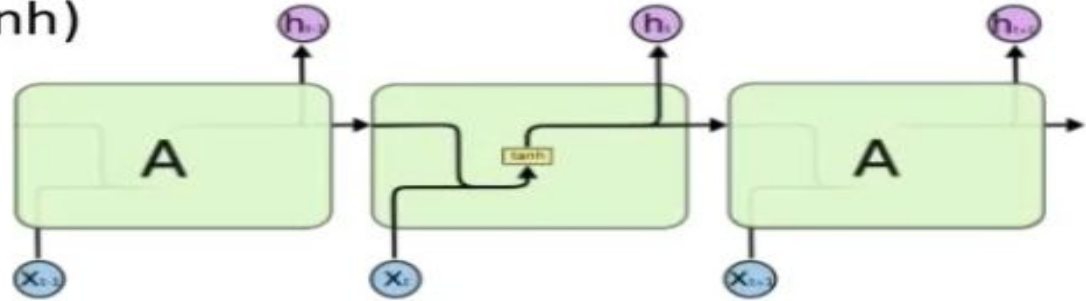
- Increasing weights
- Large errors
- Unstability in the network



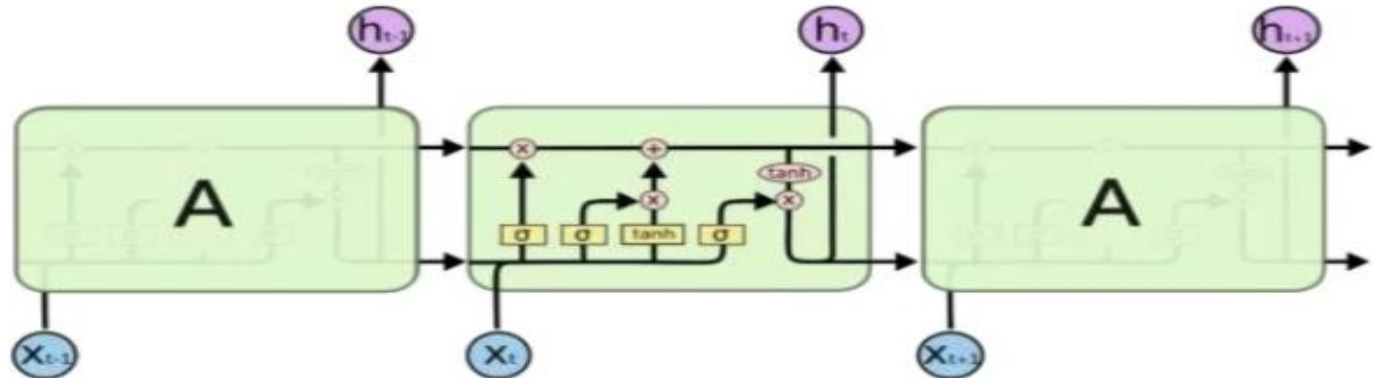
LSTM VS RNN

- **Difference between RNN and LSTM:**

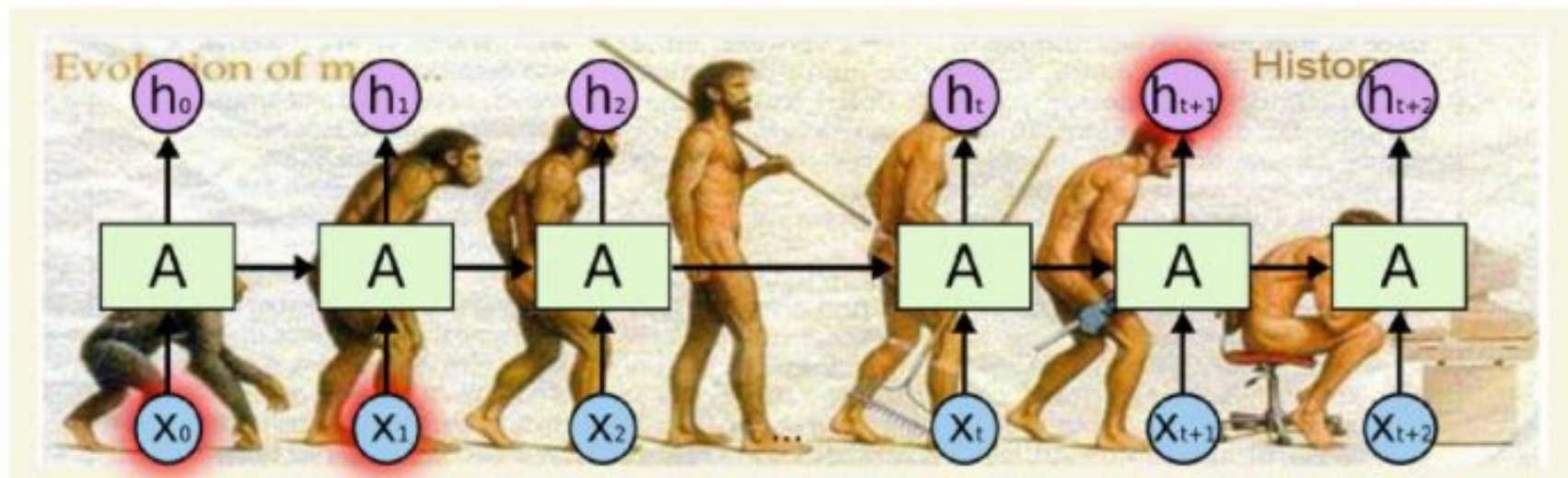
- **RNN:** single layer (tanh)






- **LSTM:** 4 interactive layers



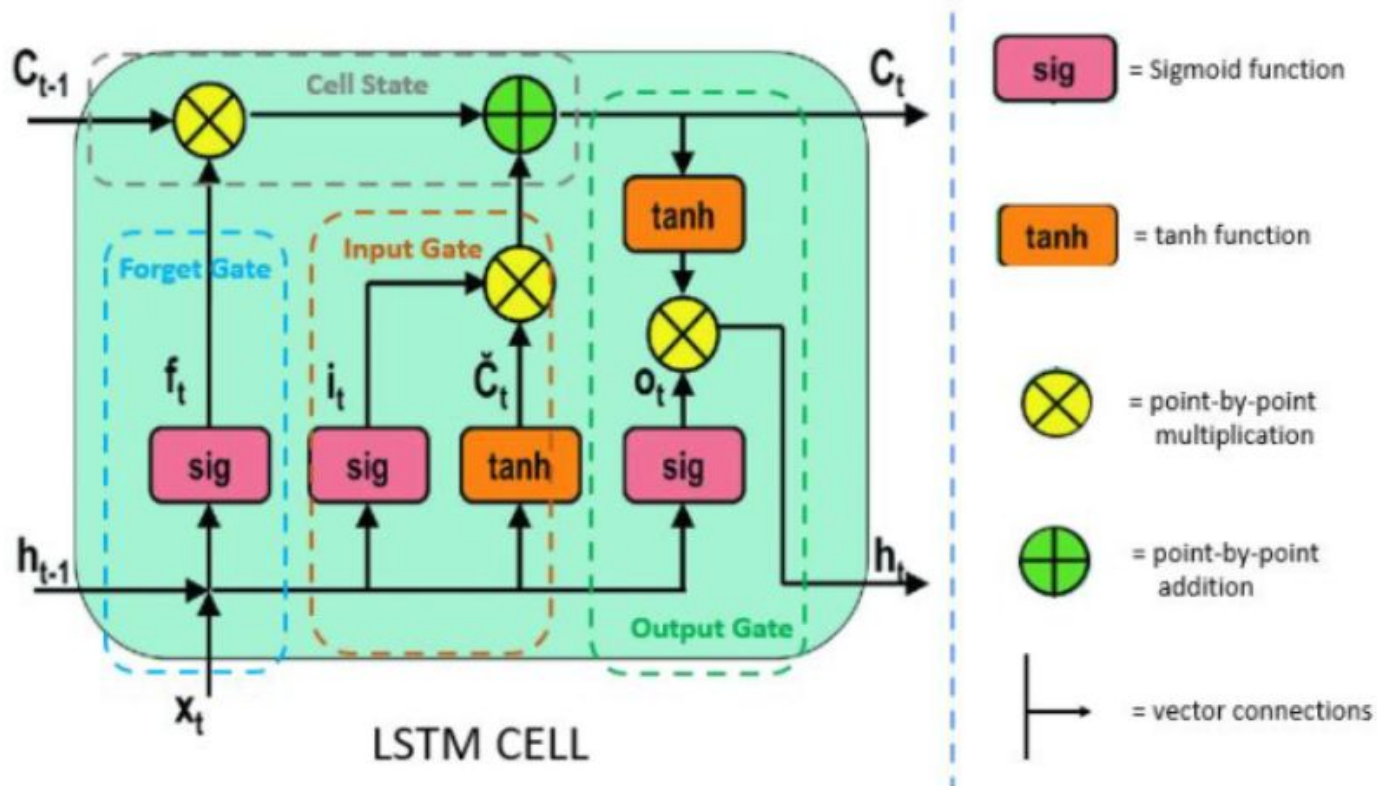
Longer-Term Dependencies



Meaning of Each Gate in LSTM Network

- An LSTM has three of these gates, to protect and control the cell state:
 - Forget gate layer  Keep gate
 - Input gate layer  Write gate
 - Output gate layer  Read gate

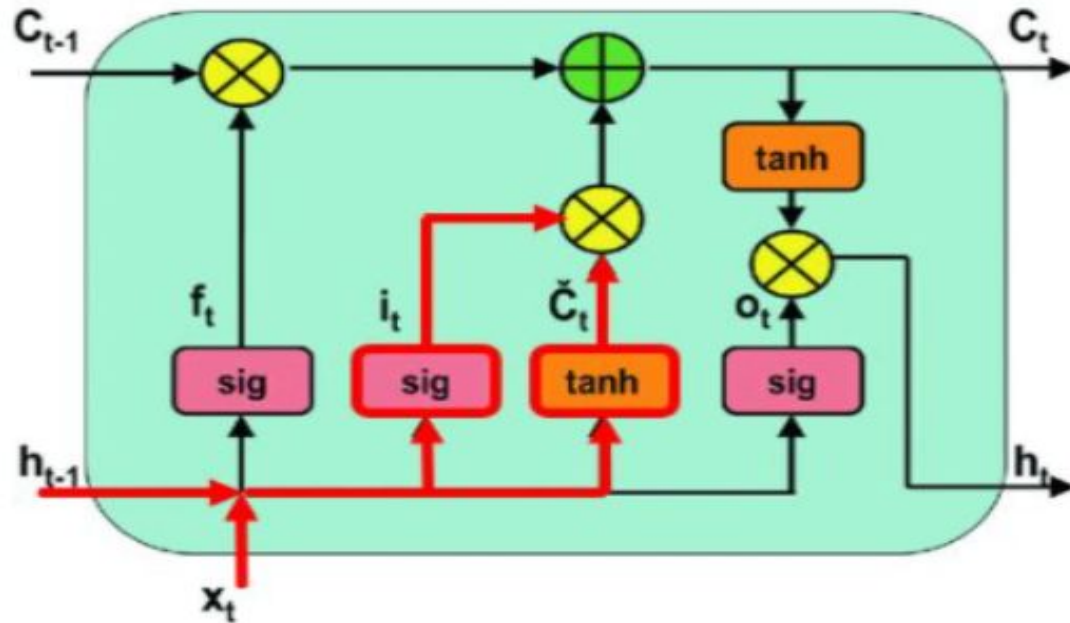
LSTM Networks Architecture



Long Short Term Memory networks that are able to learn long-term dependencies. It is capable of remembering information for long periods of time.

LSTM Input Gate

Input Gate



Input Gate Operation

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

t = timestep

i_t = input gate at t

W_i = Weight matrix of sigmoid operator between input gate and output gate

b_t = bias vector at t

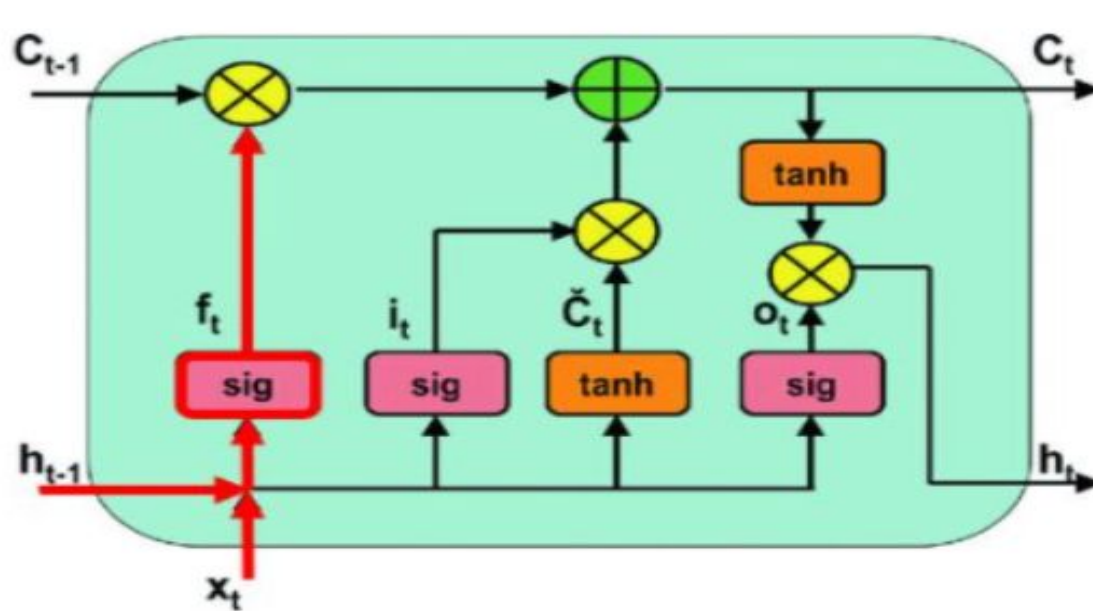
\tilde{C}_t = value generated by tanh

W_C = Weight matrix of tanh operator between cell state information and network output

b_C = bias vector at t , w.r.t W_C

Forget Gate in LSTM

Forget Gate



Forget Gate Operation

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

t = timestep

f_t = forget gate at t

x_t = input

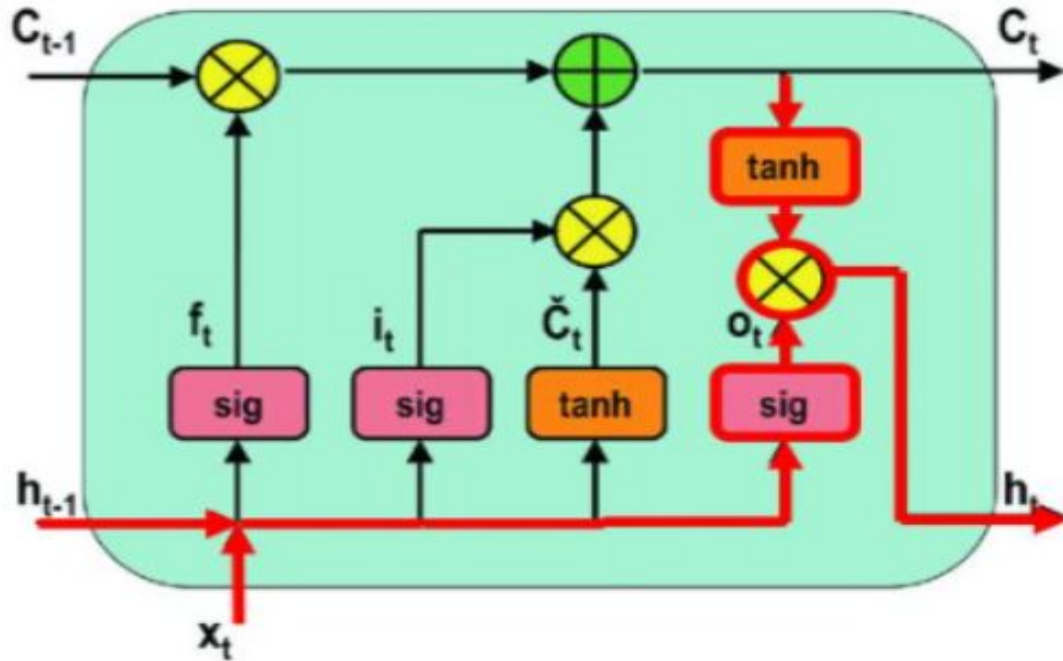
h_{t-1} = Previous hidden state

W_f = Weight matrix between
forget gate and input gate

b_t = connection bias at t

Output Gate in LSTM

Output Gate



Output Gate Operation

$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

t = timestep

O_t = output gate at t

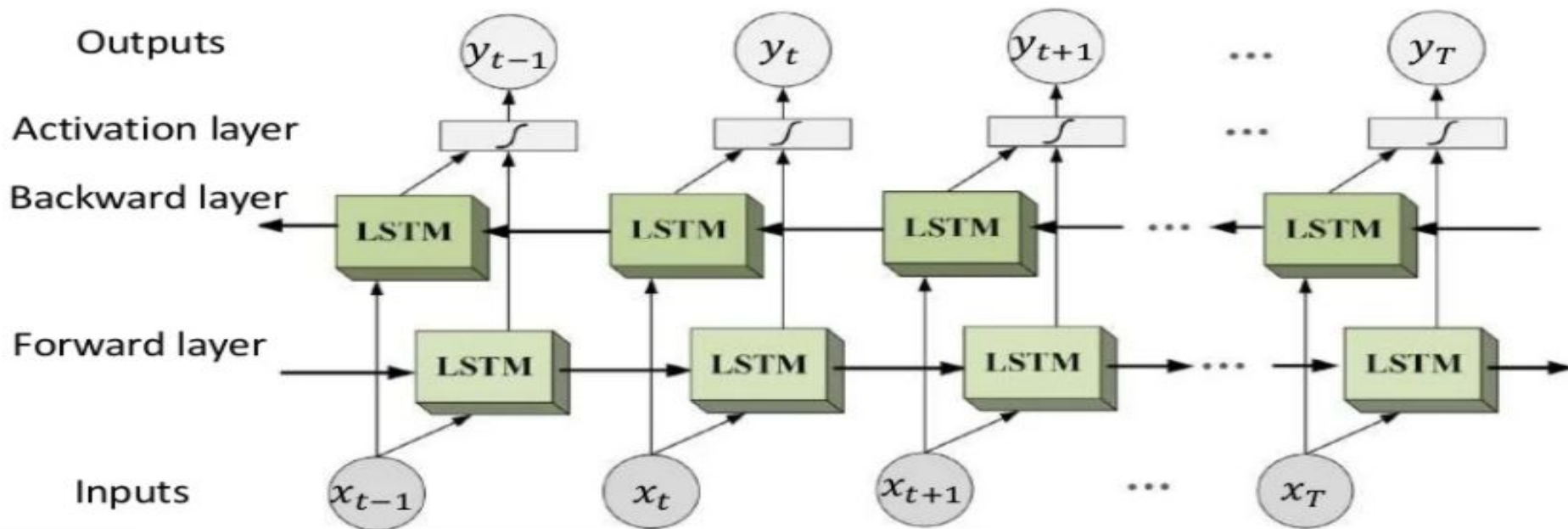
W_o = Weight matrix of output gate

b_o = bias vector, w.r.t W_o

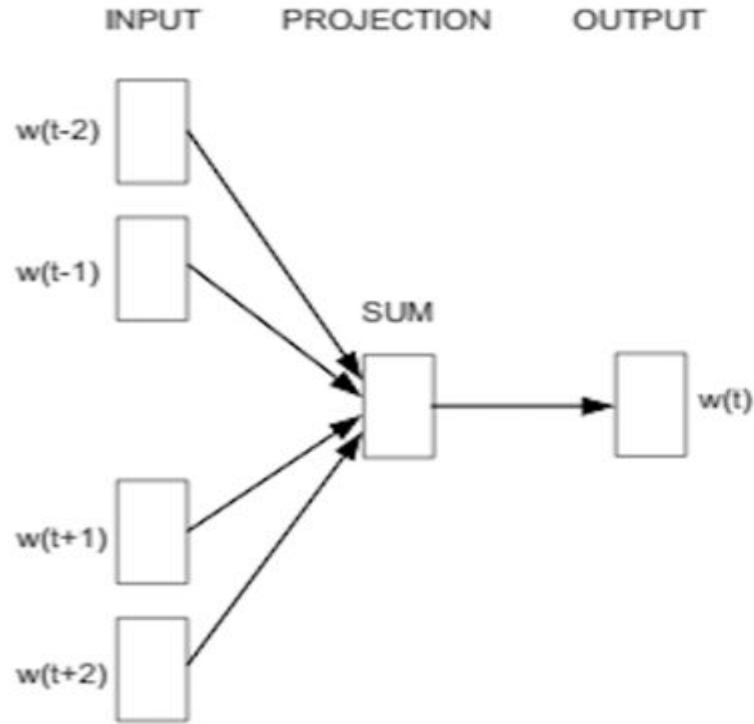
h_t = LSTM output

Bi-LSTM

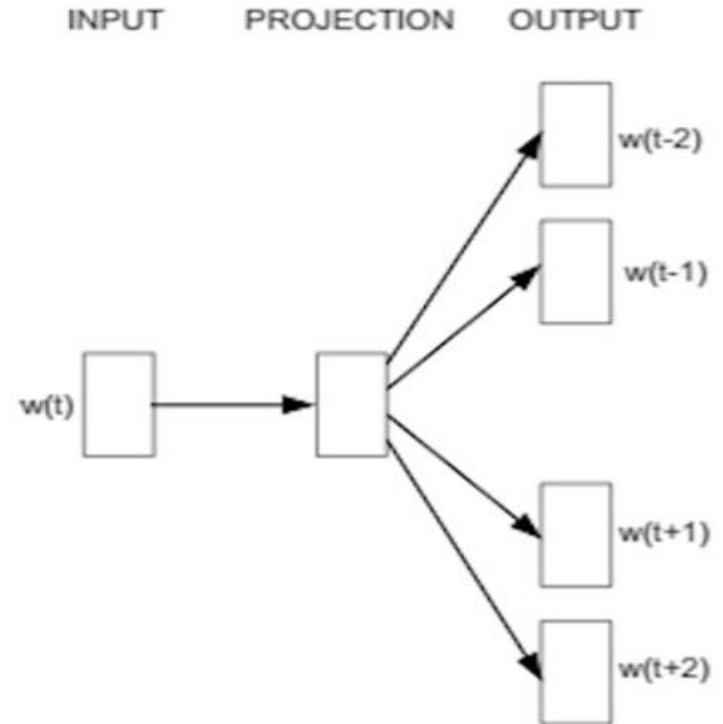
- Training Information travels in both forward and backward directions
- Remembers complex long term dependencies better.
- **Using BiLSTM:**



Word2Vec - Predict next word using Context



CBOW



Skip-gram

CBOW - Continuous Bag of Words

Bag of words

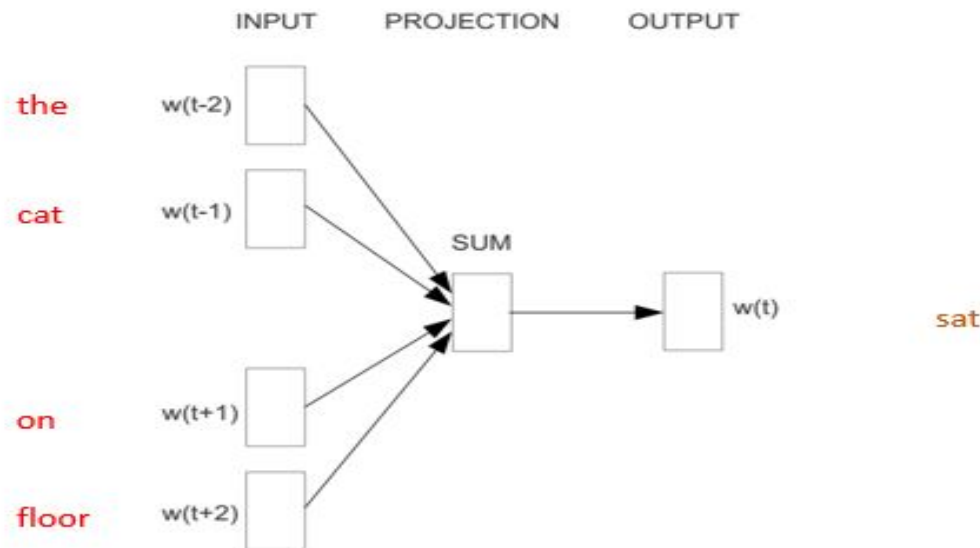
- Gets rid of word order. Used in discrete case using counts of words that appear.

CBOW

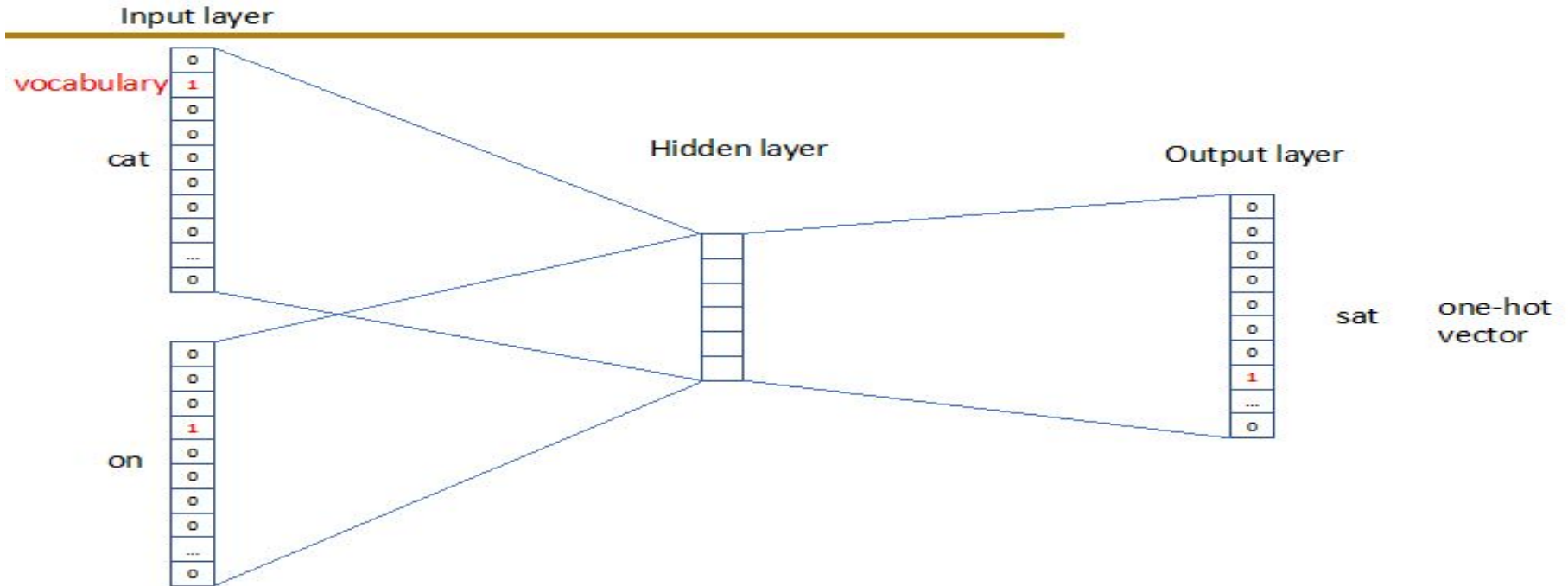
- Takes vector embeddings of n words before target and n words after and adds them (as vectors).
- Also removes word order, but the vector sum is meaningful enough to deduce missing word.

Word2vec – Continuous Bag of Word

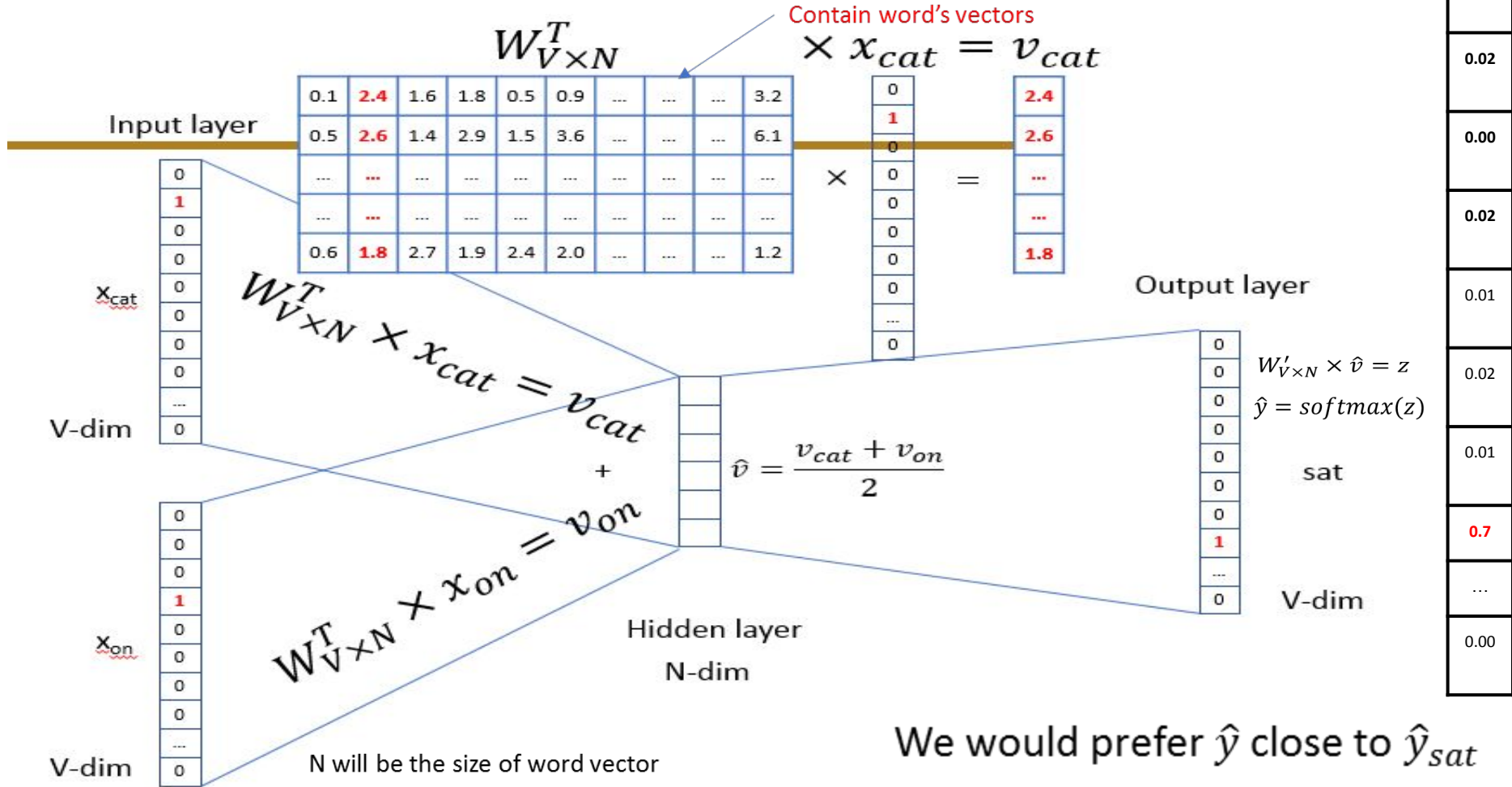
- E.g. “The cat sat on floor”
 - Window size = 2



One Hot encoding in CBOW

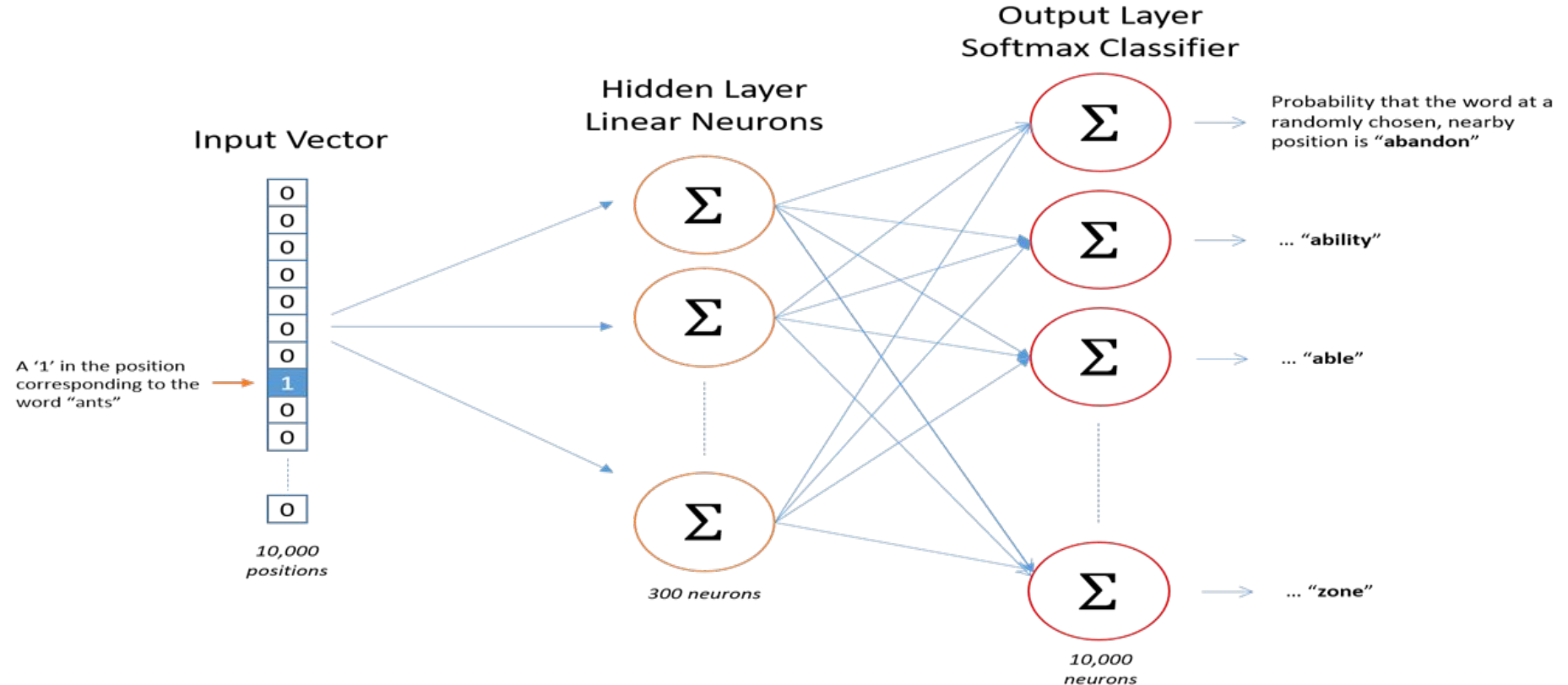


Word2Vec Model - How prediction happens



Skip gram

- Map from center word to the probabilities of the surrounding words



Embedding Matrix code

FL_dataset - models.py

File Edit View Navigate Code Refactor Run Tools Git Window Help

FL_dataset > top3next > utils > models.py

main.py x utils.py x local.py x federated.py x keyedvectors.py x constants.py x models.py x

```
209
210 def embed_vocab(word2idx, V, D): D: 300 V: 45539 word2idx: {'lol': 1, 'like': 2, 'get': 3, 'good': 4, 'go': 5, 'u': 6, 'know': 7, 'really': 8, 'im': 9
211 word2vec_model = KeyedVectors.load_word2vec_format(GOOGLE_W2V, binary = True) word2vec_model: <gensim.models.keyedvectors.KeyedVectors object at 0x600
212 #print(word2vec)
213 embedding_matrix = np.zeros((V + 1, D)) embedding_matrix: [[ 0. 0. 0. ... 0. 0., 0. ], [-0.23535156 -0.82
214 regexp = re.compile(r'[0-9]') regexp: re.compile('[0-9]')
215 for word, i in word2idx.items(): i: 5017 word: 'nooooooooooooo'
216 print(word)
217 if i >= V:
218     continue
219 try:
220     if word == 'didn't':
221         word = 'didn't'
222     elif regexp.search(word):
223         #print("**** >>>>>> " + word)
embed_vocab() > for word, i in word2idx.items()
```

Debug: main x

Debugger Console

Frames

- MainThread
- embed_vocab, models.py:38
- _build, models.py:38
- _init_, models.py:34
- train_local_model, lo
- <module>, main.py:

Switch frames from anyw

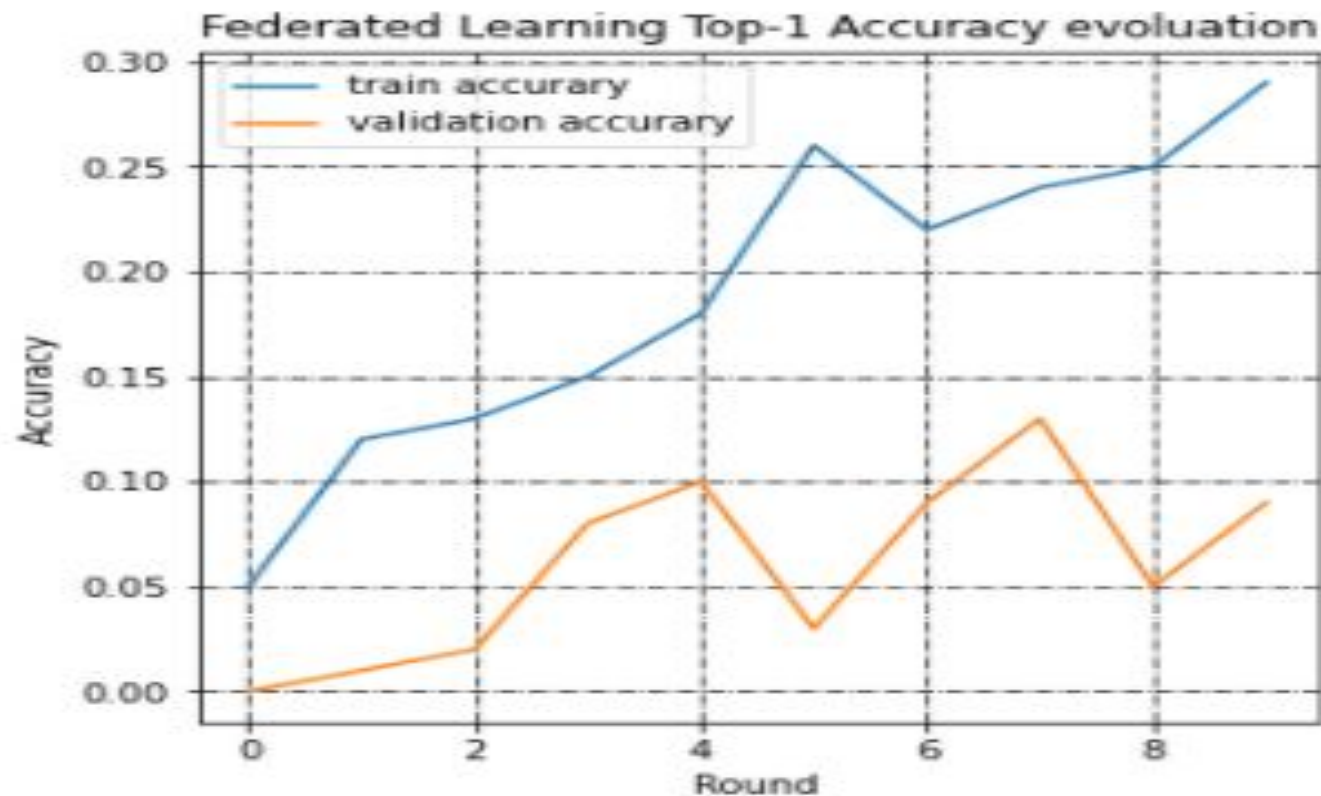
Git TQDO Prob

embedding_matrix

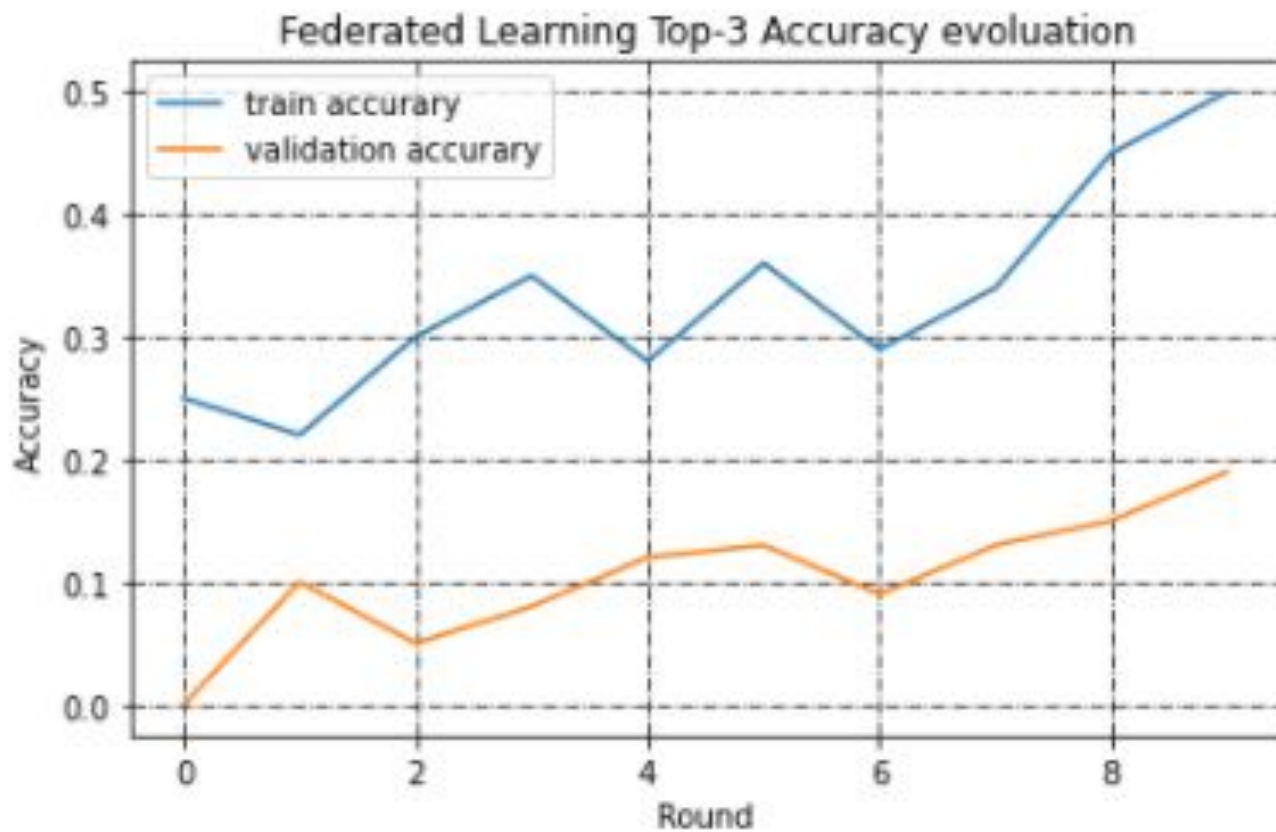
word Embedding Vectors

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	
0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1	-0.23535	-0.04956	0.08203	0.34180	-0.14355	0.14062	-0.09180	-0.12354	0.14160	0.07275	-0.19727	-0.39062	-0.32227	-0.15137	-0.01
2	0.10352	0.13770	-0.00298	0.18164	-0.00024	0.10693	0.19727	0.00751	-0.08447	0.13672	-0.00244	-0.05298	-0.02246	-0.04175	-0.15
3	0.03320	-0.08984	-0.29492	0.11523	-0.07129	-0.05396	0.01050	0.11914	0.02454	0.08008	-0.15039	-0.19043	0.04785	0.17090	-0.12
4	0.04053	0.06250	-0.01746	0.07861	0.03271	-0.01263	0.00964	0.12354	-0.02148	0.15234	-0.05835	-0.10645	0.02124	0.13574	-0.13
5	-0.02637	0.06836	-0.03113	0.21973	0.00342	-0.00903	0.10791	-0.17480	0.07715	0.00038	-0.10254	-0.01733	-0.03088	0.05762	-0.10
6	-0.25391	0.04663	0.16406	-0.01831	-0.28320	-0.12695	-0.21387	-0.28125	-0.05908	0.10107	-0.00793	-0.24609	-0.34375	0.13965	-0.23
7	-0.05493	-0.11719	0.02783	0.07471	-0.19727	0.09326	0.18164	-0.06177	0.10400	-0.06885	-0.02307	-0.15039	-0.07959	0.02979	-0.17
8	0.09619	-0.02869	-0.10840	0.14551	-0.09180	0.04150	0.03687	-0.15527	0.01227	-0.03247	-0.07861	-0.19922	-0.03101	-0.21387	-0.25
9	-0.03662	0.01453	0.03516	0.23047	-0.20801	0.26172	-0.13184	-0.08740	0.07520	0.03882	-0.19727	-0.37109	-0.22461	-0.05029	0.14
10	-0.12891	-0.15625	-0.26953	0.14453	0.03931	-0.07227	-0.23145	-0.06104	-0.01324	0.14551	-0.18652	-0.03735	0.02527	-0.03857	-0.21

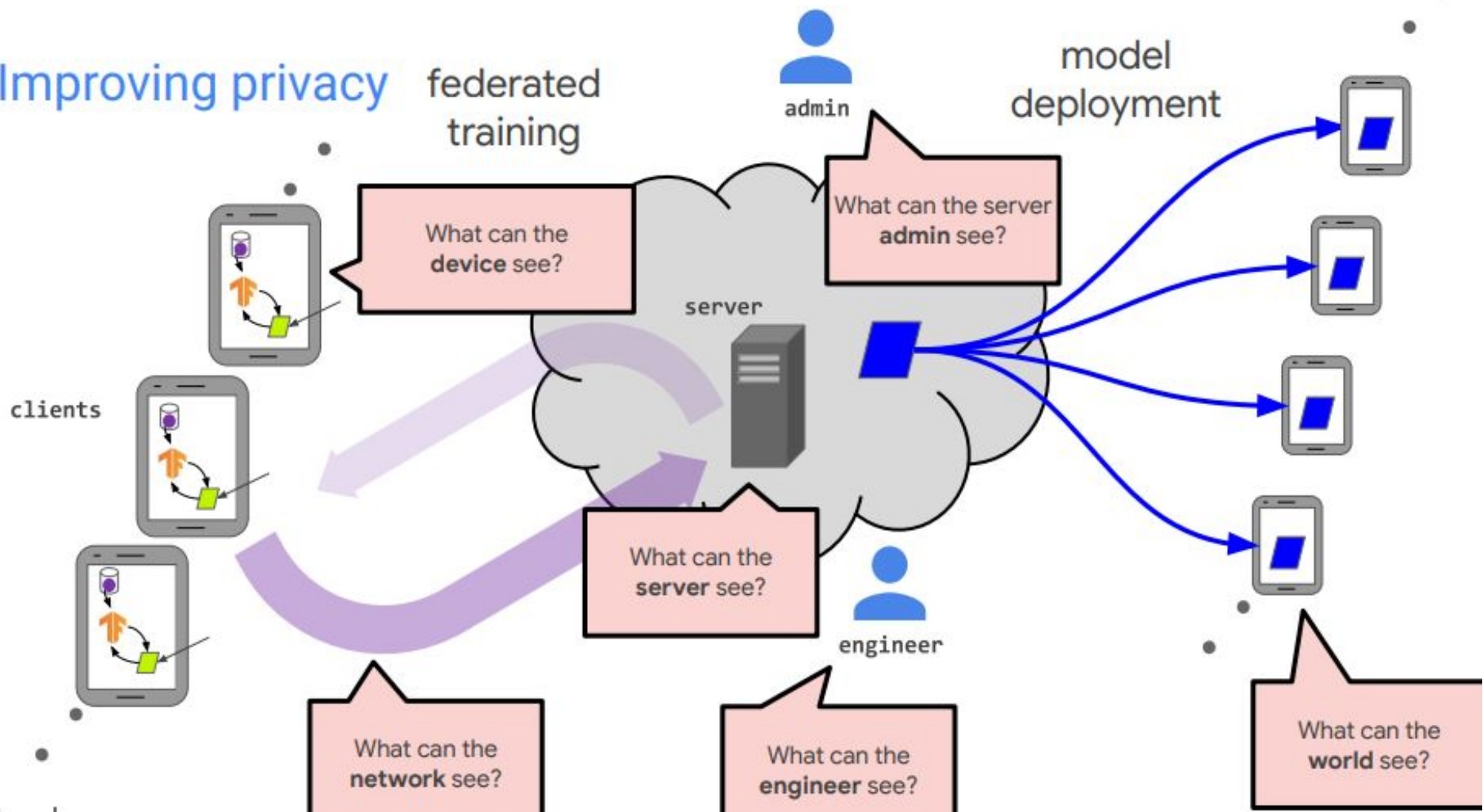
Top 1 Accuracy



Top 3 Accuracy



Improving privacy



Challenges in Mixed-mode Federated Learning

- **Local and global models:** Server is required to maintain a global model for its clients' inferences
 - **Limited data sharing:** Training method can run without requiring the server to access any data, including the target labels, only program needs to travel
 - **Sample synchronization:** Not all the clients have all the samples, in those cases model miss learning from those data distributions
- **No existing FL methods can meet all these requirements better than proposed Mixed-mode FL scheme**

Thank

You