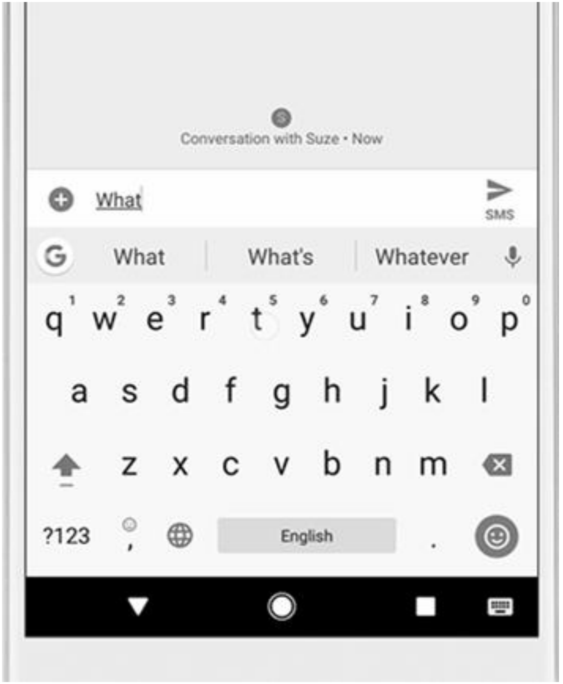
Federated Learning For Mobile Keyboard Prediction

Katie Anders
Archana Subba

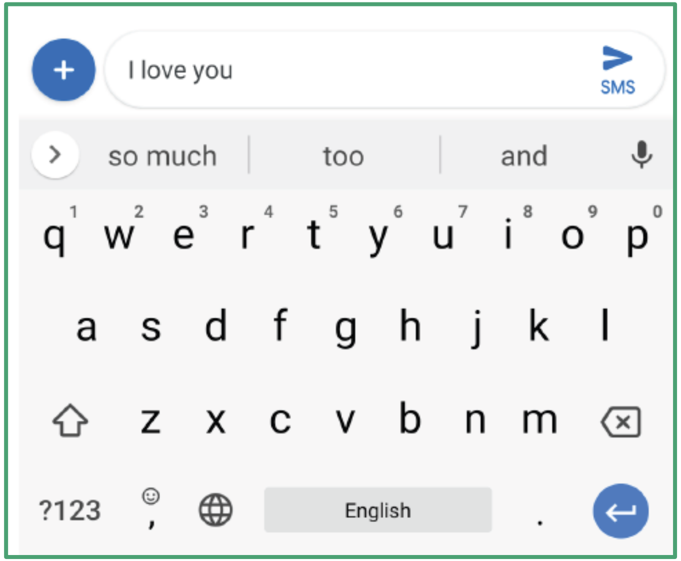
Table of Contents

Introduction

- Gboard — the Google keyboard
  - decoding noisy signals from input modalities including tap and word-gesture typing, provides autocorrection, word completion, and next-word prediction features.
- n-gram finite state transducer (FST) - former approach
  - High latency - slow performance
  - CPU consumption drained client device batteries.



Federated Learning

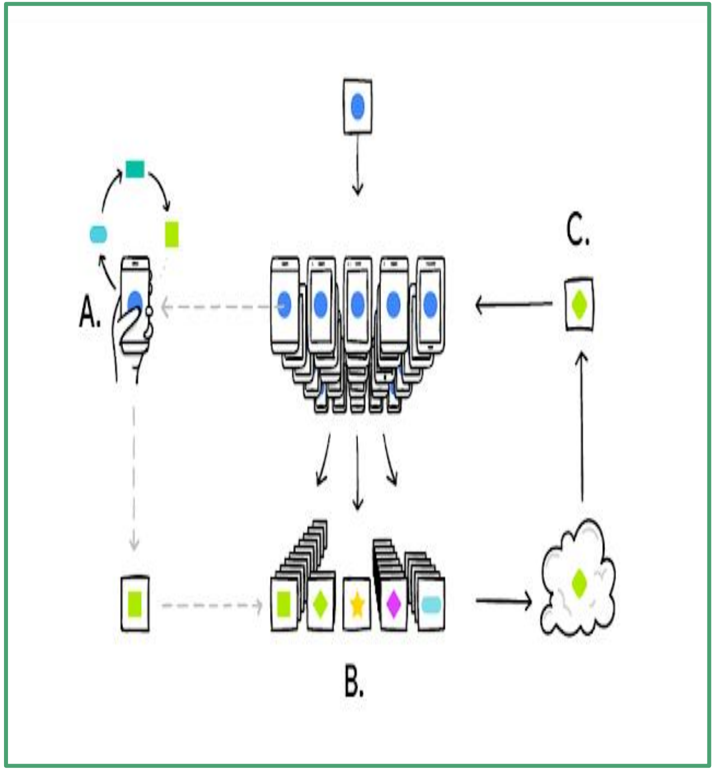


- Distributed, on-device learning framework
- Coupled Input and Forget Gate (CIFG), Long Short-Term Memory (LSTM) recurrent neural network (RNN)
- Also known as Collaborative Learning
    - Smarter models, lower latency, and less power consumption, all while ensuring privacy.
- Centralized model on decentralized data
    - Provides a decentralized computation strategy that can be employed to train a neural model.

The Approach - Federated Learning

“Learn from everyone, without learning about anyone.”

“Learn from everyone, without learning about anyone.”

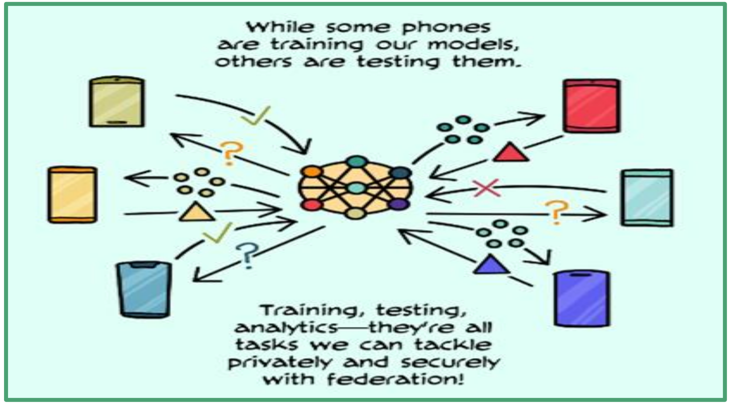


- Client devices compute SGD updates on locally-stored data
- A server aggregates the client updates to build a new global model
- The new model is sent back to clients, and the process is repeated

Features of Federated Learning

- Collaborative Learning - Trained on-device models are trained in the hand of other users.

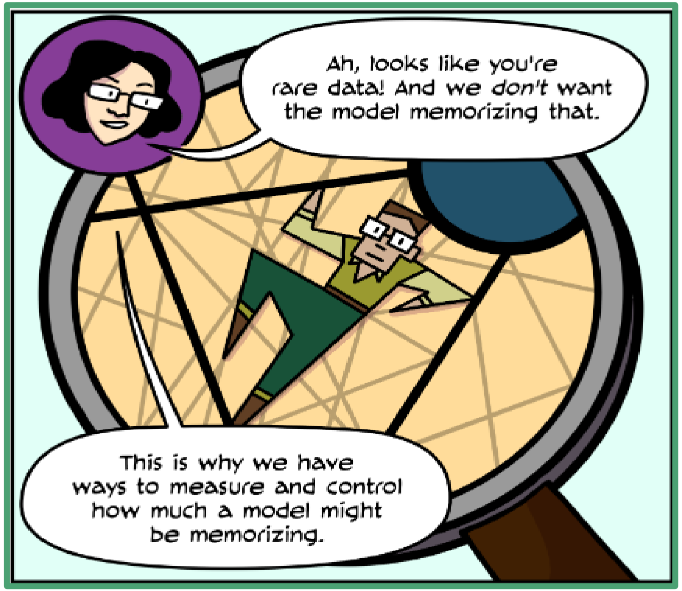
- Low Latency - No need to spend time collecting and aggregating data from diverse sources each time.



- Secure Aggregation - Enables the server to combine encrypted results and only decrypt the aggregate

Features of Federated Learning

- Differential Privacy for model memorization where a shared model's parameters might be too influenced by a single contributor.



Coupled Input and Forget Gate (CIFG)

- Variant of the Long Short-Term Memory (LSTM) recurrent neural network(RNN).
- Uses a single gate to control both the input and recurrent cell self-connections, reducing the number of parameters per cell by 25%.
- For timestep t, the input gate i_t and forget gate f_t have the relation

$$f_t = 1 - i_t.$$

- The number of computations and the parameter set size are reduced with no impact on model performance.
- Uses TensorFlow

FederatedAveraging

- The FederatedAveraging algorithm is used on the server to combine client updates and produce a new global model.
- Every client computes the average gradient, g_k, on its local data with the current model w_t using one or more steps of stochastic gradient descent (SGD)
- For a client learning rate, the local client update is given by: $w_t - \epsilon g_k \rightarrow w_{t+1}^k.$
- The server then does a weighted aggregation of the client models to obtain a new global model, w_{t+1}:

$$\sum_{k=1}^{K} \frac{n_k}{N} w_{t+1}^k \rightarrow w_{t+1},$$

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 $t = 1, 2, \ldots$ **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**
      $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
    $m_t \leftarrow \sum_{k \in S_t} n_k$
    $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$ *// Erratum*[4]

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

Implementation

```python
# Split the data into 3 parts
split1 = int(len(X) / 3)
split2 = int(2 * len(X) / 3)
X_train1, y_train1 = X[:split1], y[:split1]
X_train2, y_train2 = X[split1:split2], y[split1:split2]
X_test, y_test = X[split2:], y[split2:]

# Global LSTM RNN model
global_model = Sequential()
global_model.add(LSTM(32, input_shape=(X.shape[1], 1)))
global_model.add(Dense(1, activation='sigmoid'))

# Compile the global model
global_model.compile(loss='binary_crossentropy', optimizer=SGD(learning_rate=0.01), metrics=['accuracy'])

# Let's train the global model on all data
global_model.fit(X.reshape(-1, X.shape[1], 1), y, epochs=10, batch_size=32, verbose=0)

# Define the local LSTM RNN model1
local_model1 = Sequential()
local_model1.add(LSTM(32, input_shape=(X.shape[1], 1)))
local_model1.add(Dense(1, activation='sigmoid'))

# Compile the local model1
local_model1.compile(loss='binary_crossentropy', optimizer=SGD(learning_rate=0.01), metrics=['accuracy'])

# Define the local LSTM RNN model2
local_model2 = Sequential()
local_model2.add(LSTM(32, input_shape=(X.shape[1], 1)))
local_model2.add(Dense(1, activation='sigmoid'))

# Compile the local model2
local_model2.compile(loss='binary_crossentropy', optimizer=SGD(learning_rate=0.01), metrics=['accuracy'])

# Simulate local training and updating of global model
num_rounds = 10
for i in range(num_rounds):
    # Get local model weights and send to global model
    local_weights1 = local_model1.get_weights()
    local_weights2 = local_model2.get_weights()

    ## Get weighted aggregation
    agg_weights = (0.4 * np.array(local_weights1) + 0.6 * np.array(local_weights2))

    global_model.set_weights(agg_weights)

    # Evaluate global model on test dataset
    loss, acc = global_model.evaluate(X_test.reshape(-1, X_test.shape[1], 1), y_test, verbose=0)
    print(f'Round {i+1} - Global Model: Accuracy={acc:.4f}')

    # Train local model1 on local dataset
    local_model1.fit(X_train1.reshape(-1, X_train1.shape[1], 1), y_train1, epochs=3, batch_size=32, verbose=0)

    # Train local model2 on local dataset
    local_model2.fit(X_train2.reshape(-1, X_train2.shape[1], 1), y_train2, epochs=3, batch_size=32, verbose=0)
```

Results

- Results display the accuracy of the global model.
- Accuracy in round one is 41.84% and is 78.31% in round ten.
    - This displays an improvement in the global model’s accuracy throughout time.

```
Round 1 - Global Model: Accuracy=0.4184
Round 2 - Global Model: Accuracy=0.6533
Round 3 - Global Model: Accuracy=0.6896
Round 4 - Global Model: Accuracy=0.7148
Round 5 - Global Model: Accuracy=0.7316
Round 6 - Global Model: Accuracy=0.7561
Round 7 - Global Model: Accuracy=0.7642
Round 8 - Global Model: Accuracy=0.7687
Round 9 - Global Model: Accuracy=0.7789
Round 10 - Global Model: Accuracy=0.7831
```

Disadvantages of Federated Learning

- Heterogeneity - Statistical
  - Data is not identically generated and collected.
  - For example, users all use their phones and language in a variety of contextual ways.
  - This creates a challenge of complexity for next word predictions.
    - Issues arise with modeling, analysis, and evaluation.

Disadvantages of Federated Learning

- Heterogeneity - Systems
  - Hardware, network, and power limitations may cause a variation in the computational and physical capabilities of devices.
  - Active network sizes may also differ from the intended network size.
  - Issues like this can lead to significant fault tolerance and outliers that affect data.

Conclusion

- Trained a CIFG language model from scratch using federated learning to compete against a server-trained CIFG model and baseline n-gram model.
  - For keyboard next-word prediction tasks, the federated learning model outperformed the server and baseline models.
- The federated learning model is one of the first applications of it's kind in a commercial setting.
- Federated learning security and privacy advantages:
  - Improves language model quality while simultaneously training highly distributed computing devices.

References

Hard, Andrew, et al. “Federated Learning for Mobile Keyboard Prediction.” *CoRR*, vol. abs/1811.03604, 2018, http://arxiv.org/abs/1811.03604.

Li, Tian, et al. “Federated Learning: Challenges, Methods, and Future Directions.” *CoRR*, vol. abs/1908.07873, 2019, http://arxiv.org/abs/1908.07873.

McMahan, H. Brendan, et al. “Federated Learning of Deep Networks Using Model Averaging.” *CoRR*, vol. abs/1602.05629, 2016, http://arxiv.org/abs/1602.05629.

“Tensorflow: An Open Source Machine Learning Framework for Everyone.” *GitHub*, https://github.com/tensorflow/tensorflow.

Questions?

Thank you for your time!