how to solve time-series
classification problem? → recurrent neural network   RNN
↳ sequentially partitioned data, where segments distributed amongst clients

use SL approach
1D convolutional neural networks
1DCNN

Context-aware applications
multi-dimensional
time-series data
Sequence of data points indexed in time order

"stay usable"

challenges
• maintaining utility during privacy preserving data processing

first few LSTM layers,
∴ reduce computation burden compared to centralized learning scheme

SL

long-short term memory
LSTM

selectively remember patterns for long durations

LSTMSPLIT NOTES

how provide data privacy protection while using ML?

SOLVED: part client, part server

split learning (SL)

federated learning (FL)
enables client-side train models w/ own private data
→ NO disclosure data

PRO

CON  high computational cost client-side when dataset large

alleviate computational burden w/ still data privacy

LSTMSPLIT

• uses differential privacy (DP) to add noise

• type of RNN,
• effective in processing TS-data
• deals w/ vanishing & exploding gradient problem when training RNNs
• feedback connections — can process point data & sequential data w/o relation to points in previous steps
(sequential → take into account prev. steps)
points from

ADDING NOISE

randomized algorithm, sensitivity:
larger sensitivity of func g, easier for adversary
solve by adding sufficient noise

only good for $l_1$ sens,
but we have $l_2$,
need large ε value

① Laplace Mechanism — sensitivity mentioned in LSTMSPLIT

**Definition 3.2** (The Laplace Distribution). The Laplace Distribution (centered at 0) with scale b is the distribution with probability density function

$$Lap(x|b) = \frac{1}{2b} \exp\left(-\frac{|x|}{b}\right)$$

preserves $(\varepsilon, 0)$-DP

The Laplace distribution is a symmetric version of the exponential distribution. Adds noise from a symmetric continuous distribution to true answer.

random. laplace
↳ numpy pkg

**Definition 3.3** (The Laplace Mechanism). Given any function $f :$ $\mathbb{N}^{|\chi|} \to \mathbb{R}^k$, the Laplace mechanism is defined as:

$$M_L(x, f(\cdot), \varepsilon) = f(x) + (Y_1, \dots, Y_k)$$

where $Y_i$ are i.i.d. random variables drawn from $Lap(\Delta f/\varepsilon)$.

② Staircase Mechanism ↓

HAR dataset
Split 1DCNN better

CLIENT
updates weights of network
↳ centralized node OR peer-to-peer node
request to train network
↓
download weights from server
↓
send addr - last trained client curr
↓
curr client update model by connecting to last trained
↓
downloads latest trained weights
↓
updates own weights
↓
calculate
↓
send output to server, waits for gradient from server
↓
back propagate

can't use SL w/ LSTM arch @ only 1 layer — split into $C^{th}$ layers

multiple clients:



Figure 3: SL architecture of the LSTM network.



Figure 4: An example of SL training structure with multiple clients.

**Algorithm 1: LSTMSPLIT Client**
**Require:** Training data $D^{c_i}_{train}$ of the client $c_i \in \{1, \dots, c\}$.
1: Initialize weight $W^{c_i} \leftarrow 0$.
2: **for** each client $c_i \in \{1, \dots, c\}$ **do**
3:  Set $W^{c_i} \leftarrow W^x$
4:  **for** each epoch, $e_j \in \{1, \dots, e\}$ **do**
5:   Feedforward propagation $W^{c_i}_e$ with $D^{c_i}_{train}$.
6:   Calculate activation function $A^{c_i}_{c_j}$ at its cut layer.
7:   Send $A^{c_i}_{c_j}$ with their labels $Y^{c_i}$ to the server $x$.
8:   Receive gradient $d^{c_i}_{A_{c_j}} = \nabla l \left( A^{c_i}_{c_j}; W^{c_i}_{c_j} \right)$
9:   Backward propagation with $d^{c_i}_{A_{c_j}}$.
10:   Update the weight $W^{c_i}_{c_{j-1}} \leftarrow W^{c_i}_{c_j} - \eta d^{c_i}_{A_{c_j}}$
11:  **end for**
12: **end for**

**Algorithm 2: LSTMSPLIT Server**
**Require:** The activation function $A^{c_i}$ of the cut layer and the labels $Y^{c_i}$ of the client $c_i$.
1: Set weight $W^x \leftarrow W^{c_i}$.
2: **for** each epoch, $e_j \in \{1, \dots, e\}$ **do**
3:  Feedforward propagation $W^x$ with $A^{c_i}$.
4:  Calculate loss with labels $Y^{c_i}$ and predictions $Y^{c_i}_p$.
5:  Backpropagation.
6:  $W^x_{c_j+1} \leftarrow W^x_{c_j} - \eta \nabla l \left( A^{c_i}_{c_j}, W^x_{c_j} \right)$.
7:  Send $d^{c_i}_{A_{c_j}} = \nabla l \left( A^{c_i}_{c_j}, W^x_{c_j} \right)$ to client $c_i$.
8: **end for**
9: Update weight $W^{x_0} \leftarrow W^{x_0} + W^x$.

```
shape[0] + np.random.laplace(loc=0, scale=sensitivity/epsilon)
```

by running the proceeding cell multiple times. Each time, the output changes, but most of the time,
ue answer (14,235) to be useful.