AGGREGATION LEARNING

WHEN YOUR FEATURES ARE AT DIFFERENT GRANULARITY THAN YOUR TARGET

Frits Hermans

FRITS HERMANS

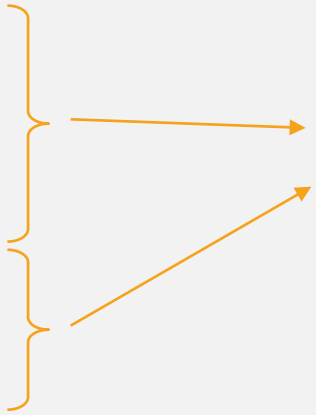
- Amsterdam, the Netherlands
- Physicist turned into data scientist
- ING Bank

INPUT-OUTPUT GRANULARITY MISMATCH

THE PROBLEM

- Granularity difference input vs output
  - E.g. transaction monitoring
    - Input are transactions; multiple rows per client
    - Output is on client level; single label per client

Client ID	Date	Amount	Debit/credit	Description	...
1	2023-01-01	9,95	Debit	Coffee	...
1	2023-01-01	1800,00	Credit	Salary	...
1	2023-01-02	21,99	Debit	Supermarket	...
...	...	...			...
2	2023-01-01	2,00	Debit	Soda machine	...
2	2023-01-02	800,00	Debit	Rent	...
...	...	...	...	...	...



Client ID	Target
1	0
2	1
...	...

TRADITIONAL SOLUTION

- Aggregate input level features to produce model features
  - Take min, max, mean, sum, count etc

Client ID	Date	Amount	Debit/credit	Description	...
1	2023-01-01	9,95	Debit	Coffee	...
1	2023-01-01	1800,00	Credit	Salary	...
1	2023-01-02	21,99	Debit	Supermarket	...
...	...	...			...
2	2023-01-01	2,00	Debit	Soda machine	...
2	2023-01-02	800,00	Debit	Rent	...
...	...	...	...	...	...

Client ID	sum credit	sum debit	count credit	...	Target
1	1800,00	1784,23	1	...	0
2	1939,00	1920,00	3	...	1
...	...	...	...	...	...

NO FEATURE INTERACTION ON INPUT LEVEL

feature	value
'fraction of transactions to tax havens'	80%
'total euro amount'	40.040
...	...

These aggregated features can come from...

...these transactions....

to tax haven	euro amount	description
1	10	Sandwich
1	10	Drinks
1	10	Sunbeds
1	10	Wine
0	40.000	Car

...or these

to tax haven	euro amount	description
1	10.000	As discussed
1	10.000	As discussed
1	10.000	As discussed
1	10.000	As discussed
0	40	Re[illegible]ran

EXPLAINABILITY

- Model explainability (e.g. Shap) on aggregated feature level can give counter intuitive results:
  - E.g.: 'fraction of transactions to low risk countries' drives score up, two possible explanations:

Explanation 1

Transactions

to low risk country	to medium risk country	to high risk country
0	1	
0		1
0		1
0	1	
0		1

All transactions go to countries with higher risk

Explanation 2

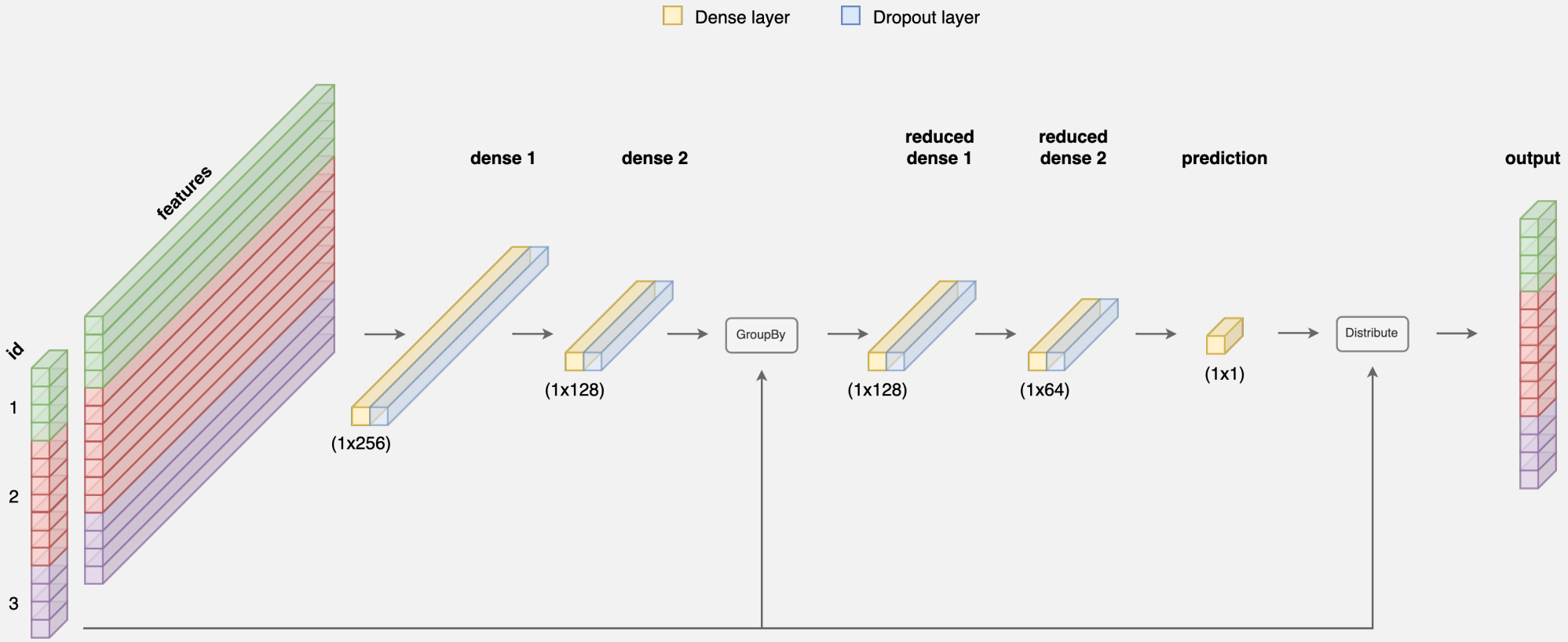
Transactions

from tax haven	to low risk country
1	1
1	1
1	1
1	1
1	1

Transactions go from tax haven to low risk countries: potential money laundering

AGGREGATION LEARNING

NETWORK ARCHITECTURE



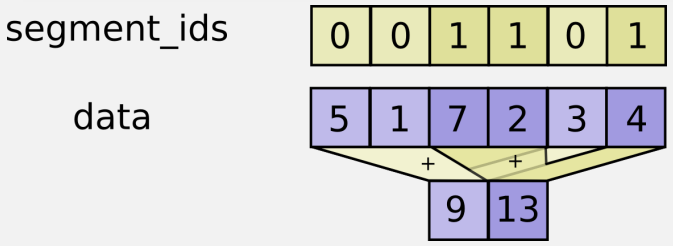
GROUPBY AND DISTRIBUTE

GROUPBY

- Tensorflow has a few GroupBy functions to be used in neural networks, e.g.: `tf.math.unsorted_segment_sum`

```
>>> unsorted_segment_sum([5, 1, 7, 2, 3, 4],
                         [0, 0, 1, 1, 0, 1],
                         num_segments=2).numpy()

array([9,13], dtype=int32)
```



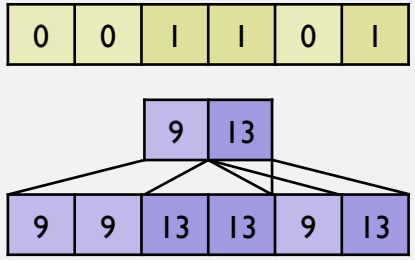
Source: tensorflow.org

DISTRIBUTE

- To distribute the predictions back to the granularity level of the input, Tensorflow offers the function `tf.gather`

```
>>> tf.gather([9, 13],
              [0, 0, 1, 1, 0, 1]).numpy()

array([9, 9, 13, 13, 9, 13], dtype=int32)
```



AGGREGATION LEARNING CLASSIFIER

```python
def create_model(n_features, n_ids):
    input_ = tf.keras.Input((n_features + 1,), name='input')
    cid = tf.keras.layers.Lambda(lambda x: tf.keras.backend.cast(x[:, 0], 'int64'), name='id')(input_)
    features_input = tf.keras.layers.Lambda(lambda x: x[:, 1:])(input_)

    dense = tf.keras.layers.Dense(256, activation='relu', name='dense_1')(features_input)
    dense = tf.keras.layers.Dense(128, activation='relu', name='dense_2')(dense)

    reduced = tf.math.unsorted_segment_sum(dense, tf.reshape(cid, (-1,)), num_segments=n_ids, name='segment_sum')

    reduced_dense = tf.keras.layers.Dense(128, activation='relu', name='reduced_dense_1')(reduced)
    reduced_dense = tf.keras.layers.Dense(64, activation='relu', name='reduced_dense_2')(reduced_dense)
    predicted = tf.keras.layers.Dense(1, activation='sigmoid', name='prediction')(reduced_dense)

    output = tf.gather(predicted, cid)

    model = tf.keras.Model(inputs=input_, outputs=output)
    model.compile(
        loss=...,
        optimizer=...,
        metrics=...
    )
    return model
```

AGGREGATION LEARNING CLASSIFIER

Split the input into ID and feature values

Dense layers on transaction level

The groupby operation

Dense layers on aggregated (ID) level

Distribute the predictions to input (transaction) level

```python
def create_model(n_features, n_ids):
    input_ = tf.keras.Input((n_features + 1,), name='input')
    cid = tf.keras.layers.Lambda(lambda x: tf.keras.backend.cast(x[:, 0], 'int64'), name='id')(input_)
    features_input = tf.keras.layers.Lambda(lambda x: x[:, 1:])(input_)

    dense = tf.keras.layers.Dense(256, activation='relu', name='dense_1')(features_input)
    dense = tf.keras.layers.Dense(128, activation='relu', name='dense_2')(dense)

    reduced = tf.math.unsorted_segment_sum(dense, tf.reshape(cid, (-1,)), num_segments=n_ids, name='segment_sum')

    reduced_dense = tf.keras.layers.Dense(128, activation='relu', name='reduced_dense_1')(reduced)
    reduced_dense = tf.keras.layers.Dense(64, activation='relu', name='reduced_dense_2')(reduced_dense)
    predicted = tf.keras.layers.Dense(1, activation='sigmoid', name='prediction')(reduced_dense)

    output = tf.gather(predicted, cid)

    model = tf.keras.Model(inputs=input_, outputs=output)
    model.compile(
        loss=...,
        optimizer=...,
        metrics=...
    )
    return model
```

LOSSES AND METRICS

- Losses and metrics need to be adjusted
  - Clients have different number of transactions
  - Classification problem might be imbalanced
- Sample weight:

$$\frac{\text{class weight for customer label}}{\text{n transactions for customer}}$$

client id	label	1/n txs	class weight	sample weight
1	0	1/3	0.05	0.017
1	0	1/3	0.05	0.017
1	0	1/3	0.05	0.017
2	1	1/2	0.95	0.475
2	1	1/2	0.95	0.475
...	...	...	...	...

BATCHES

- When training a neural network or when predicting, data are divided over batches
  - All rows in a batch pass through the network at once
  - Therefore all rows for the same ID should be in the same batch
    - Otherwise the groupby operation is incomplete
- Use binpacking algorithm to fill the batches
  - Batches are filled up to capacity
  - No client IDs are broken

Batch 1

client id	feature 1
1	...
1	...
2	...
2	...
3	...

Batches contain incomplete information

Batch 2

client id	feature 1
7	...
7	...
1	...
12	...
13	...

Batch 1

client id
1
1
1
17
17
300

Batch 2

client id
4
4
4
4
4
4

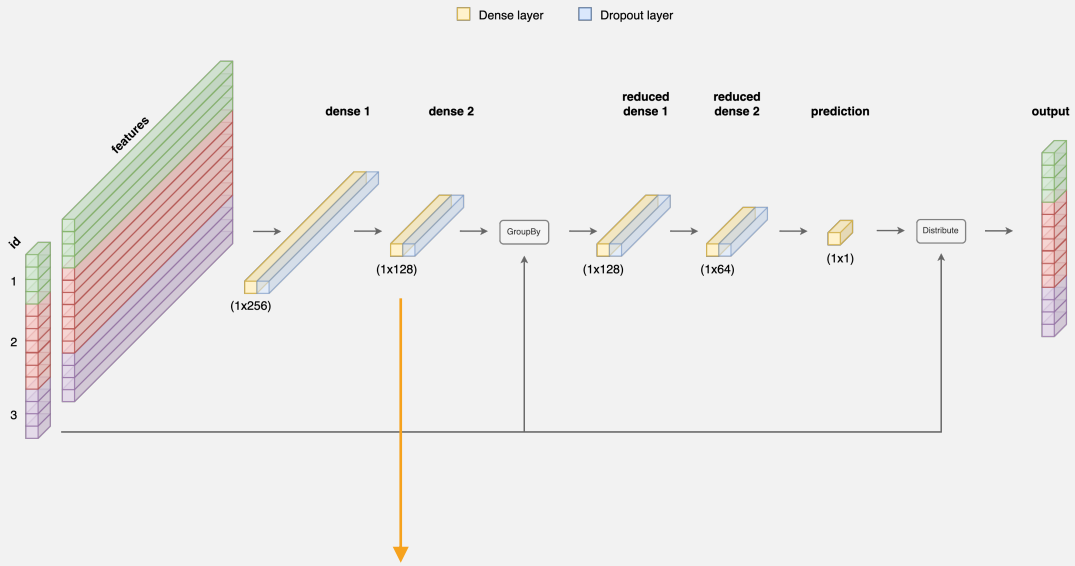
Batch 3

client id
5
5
314
314
314
314

EXPLAINABILITY

- Which input is driving the model output?
- Shapley doesn't work because it will treat the ID as a feature
- We can make use of characteristics of neural networks
  1. Simple: Extract output at intermediate levels to get importance of each row in the input
  2. Advanced: Use integrated gradients to get feature importance on input row level

Row level prediction importance




```
model_tx_importance = tf.keras.Model(inputs=model.input,
                                     outputs=model.get_layer('dense_2').output)

model_tx_importance.predict(df[['id']+features]).sum(axis=1)
```

EXAMPLE

- Goal is to show interaction...
  - ...between features *within* transactions
  - ...between *different* transactions
- Generate synthetic 'transaction monitoring' data
  - 5 binary features: x_1, ..., x_5
  - 4 transactions per ID
  - Risk type 1: transaction has x_1 and x_2 positive
  - Risk type 2: transaction has x_1 and x_4 positive
  - Risk type 1 is mitigated if another transaction has x_1 positive, x_2 negative and x_3 positive
  - Positive label if risk type 1 is identified and not mitigated and/or if risk type 2 is identified, otherwise negative
- Train the model on this synthetic data

- Model converges to perfect out-of-sample performance (trivially)
- Example shows...
  - ...how interaction between features and transactions are correctly included
  - ...transaction importance from last dense layer on transaction level

id	x_1	x_2	x_3	x_4	x_5	risk_type_1	risk_type_2	risk_reducing_1	target	proba	tx_importance
7043	1	0	1	0	0	0	0	1	0	0.000000	14.038346
7043	1	1	1	0	0	1	0	0	0	0.000000	19.631275
7043	1	0	0	0	1	0	0	0	0	0.000000	0.418377
7043	1	0	0	0	1	0	0	0	0	0.000000	0.418377

id	x_1	x_2	x_3	x_4	x_5	risk_type_1	risk_type_2	risk_reducing_1	target	proba	tx_importance
7160	0	0	0	0	1	0	0	0	1	1.000000	0.000000
7160	1	1	0	1	0	1	1	0	1	1.000000	23.912897
7160	1	0	1	0	1	0	0	1	1	1.000000	13.979871
7160	1	1	1	0	1	1	0	0	1	1.000000	19.371525

QUESTIONS