A code-free deep learning toolbox

Piero Molino // piero.molino@gmail.com

Deep Learning experimentation toolbox based on TensorFlow

No coding required

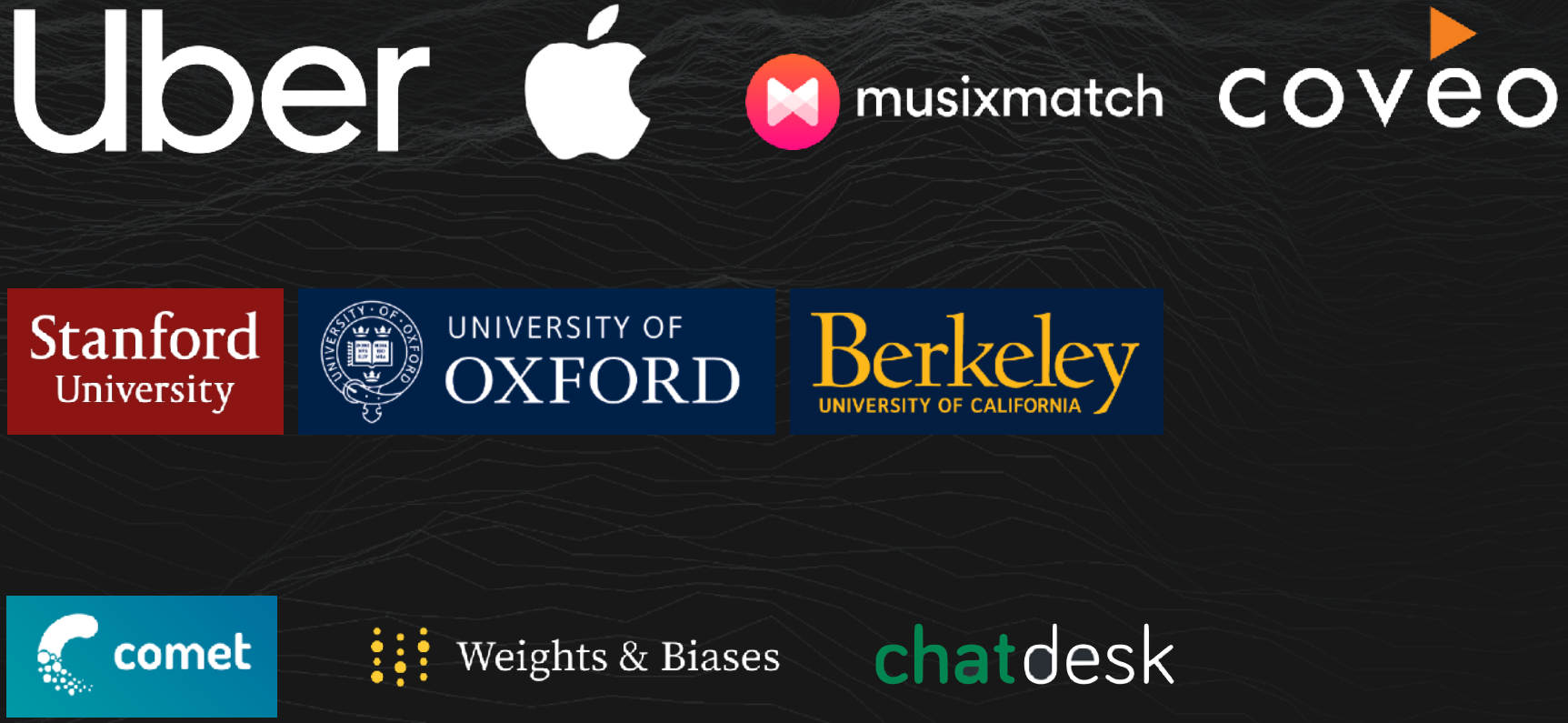
Statistics

7000+ Stars on GitHub

2400+ downloads/month

60+ Contributors

~80 commits/month


Uber
musixmatch
coveo
Stanford University
UNIVERSITY OF OXFORD
Berkeley
UNIVERSITY OF CALIFORNIA
comet
Weights & Biases
chatdesk

GENERAL

A new data type-based approach to deep learning model design that makes the tool suited for many different applications.

FLEXIBLE

Experienced users have deep control over model building and training, while newcomers will find it easy to use.

EXTENSIBLE

Easy to add new model architecture and new feature data-types.

UNDERSTANDABLE

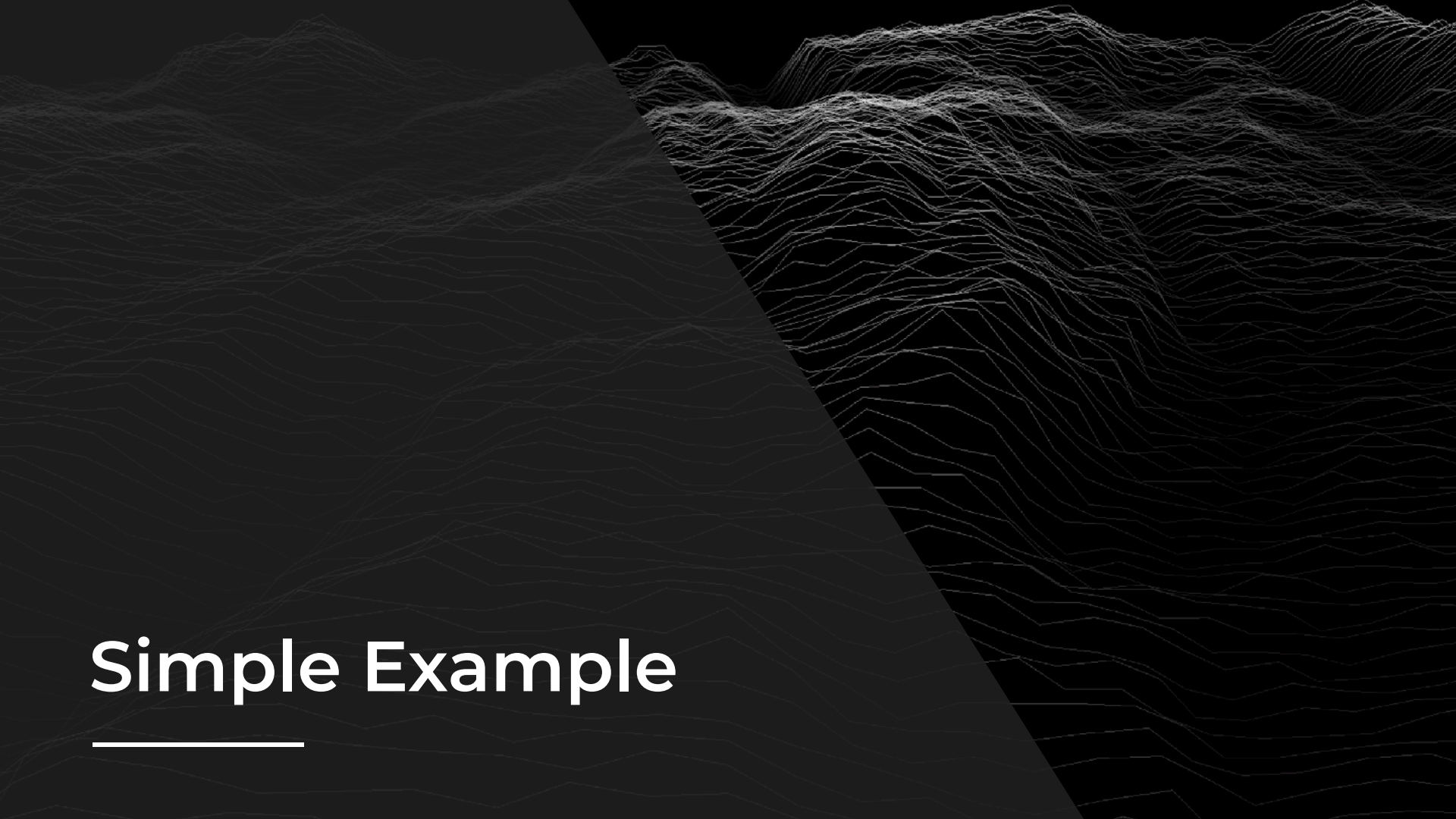
We provide standard visualizations to understand their performances and compare their predictions.

EASY

No coding skills are required to train a model and use it for obtaining predictions.

OPEN

Released under the open source Apache License 2.0.



Simple Example

Example

NEWS	CLASS
Toronto Feb 26 - Standard Trustco said it expects earnings in 1987 to increase at least 15 to 20 pct from the 9 140 000 dlrs or 252 dlrs...	earnings
New York Feb 26 - American Express Co remained silent on market rumors it would spinoff all or part of its Shearson Lehman Brothers Inc...	acquisition
BANGKOK March 25 - Vietnam will resettle 300 000 people on state farms known as new economic zones in 1987 to create jobs and grow more high-value export crops...	coffe

Example experiment command

```
ludwig experiment
--dataset my_dataset.csv
--config "{input_features: [{name: news, type: text}], output_features: [{name:
class, type: category}]}"

# you can specify a --config_file file_path argument instead
# that reads a YAML file
```

The Experiment command:

1. splits the data in training, validation and test sets
2. trains a model on the training set
3. validates on the validation set (early stopping)
4. predicts on the test set

```
Epoch  1
Training: 100%|████████████████████████| 23/23 [00:02<00:00,  9.86it/s]
Evaluation train: 100%|████████████████| 23/23 [00:00<00:00, 46.39it/s]
Evaluation vali : 100%|████████████████| 4/4 [00:00<00:00, 29.05it/s]
Evaluation test : 100%|████████████████| 7/7 [00:00<00:00, 43.06it/s]
Took 2.8826s
```

class	loss	accuracy	hits@3
train	0.2604	0.9306	0.9899
vali	0.3678	0.8792	0.9769
test	0.3532	0.8929	0.9903

```
Validation accuracy on class improved, model saved
```

First epoch of training

```
Epoch  4
Training: 100%|████████████████████████| 23/23 [00:01<00:00, 14.87it/s]
Evaluation train: 100%|████████████████| 23/23 [00:00<00:00, 48.87it/s]
Evaluation vali : 100%|████████████████| 4/4 [00:00<00:00, 59.50it/s]
Evaluation test : 100%|████████████████| 7/7 [00:00<00:00, 51.70it/s]
Took 1.7578s
```

class	loss	accuracy	hits@3
train	0.0048	0.9997	1.0000
vali	0.2560	0.9177	0.9949
test	0.1677	0.9440	0.9964

```
Validation accuracy on class improved, model saved
```

Fourth epoch of training, validation accuracy improved

```
Epoch 14
Training: 100%|████████████████████████| 23/23 [00:01<00:00, 14.69it/s]
Evaluation train: 100%|████████████████| 23/23 [00:00<00:00, 48.67it/s]
Evaluation vali : 100%|████████████████| 4/4 [00:00<00:00, 61.29it/s]
Evaluation test : 100%|████████████████| 7/7 [00:00<00:00, 50.45it/s]
Took 1.7749s
```

class	loss	accuracy	hits@3
train	0.0001	1.0000	1.0000
vali	0.4208	0.9100	0.9974
test	0.2081	0.9404	0.9988

```
Last improvement of accuracy on class happened 10 epochs ago
```

Fourteenth epoch of training, validation accuracy has not improved in a while, let's EARLY STOP

```
===== class =====
accuracy: 0.94403892944
hits_at_k: 0.996350364964
Confusion Matrix
( { 'avg_f1_score_macro': 0.6440659449587669,
    'avg_f1_score_micro': 0.94403892944038914,
    'avg_f1_score_weighted': 0.94233823910531345,
    'avg_precision_macro': 0.71699990923354218,
    'avg_precision_micro': 0.94403892944038925,
    'avg_precision_weighted': 0.94403892944038925,
    'avg_recall_macro': 0.60875299574797059,
    'avg_recall_micro': 0.94403892944038925,
    'avg_recall_weighted': 0.94403892944038925,
    'kappa_score': 0.91202440199068868,
    'overall_accuracy': 0.94403892944038925},)
```

Test statistics overall

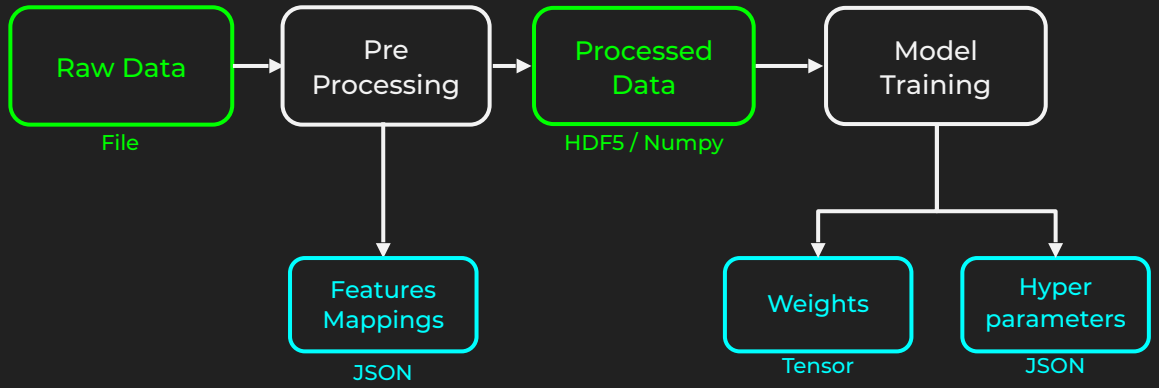
```
{sport: {'accuracy': 0.95377128953771284,
         'f1_score': 0.95214105793450876,
         'fall_out': 0.046948356807511749,
         'false_discovery_rate': 0.050251256281407031,
         'false_negative_rate': 0.045454545454545414,
         'false_negatives': 18,
         'false_omission_rate': 0.042452830188679291,
         'false_positive_rate': 0.046948356807511749,
         'false_positives': 20,
         'hit_rate': 0.95454545454545459,
         'informedness': 0.90759709773794284,
         'markedness': 0.90729591352991368,
         'matthews_correlation_coefficient': 0.90744649313843584,
         'miss_rate': 0.045454545454545414,
         'negative_predictive_value': 0.95754716981132071,
         'positive_predictive_value': 0.94974874371859297,
         'precision': 0.94974874371859297,
         'recall': 0.95454545454545459,
         'sensitivity': 0.95454545454545459,
         'specificity': 0.95305164319248825,
         'true_negative_rate': 0.95305164319248825,
         'true_negatives': 406,
         'true_positive_rate': 0.95454545454545459,
         'true_positives': 378},
```

Test statistics per class

How does it work?

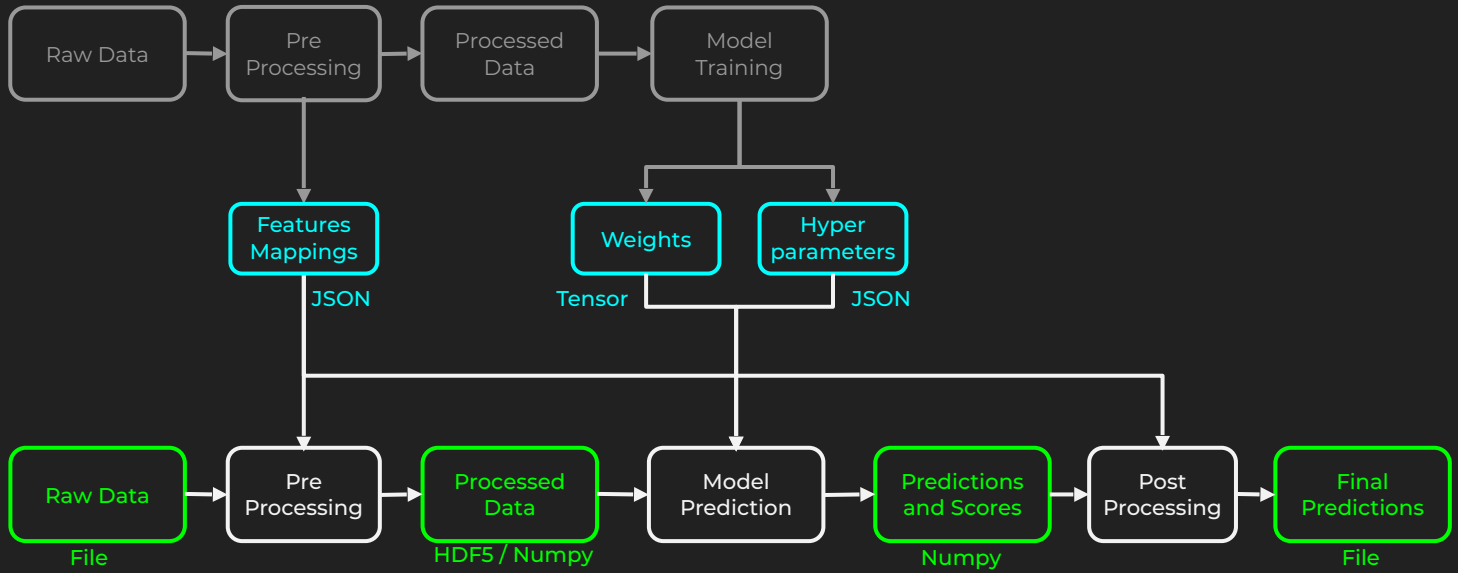
Training

Fields mappings, model hyper-parameters
and weights are saved during training



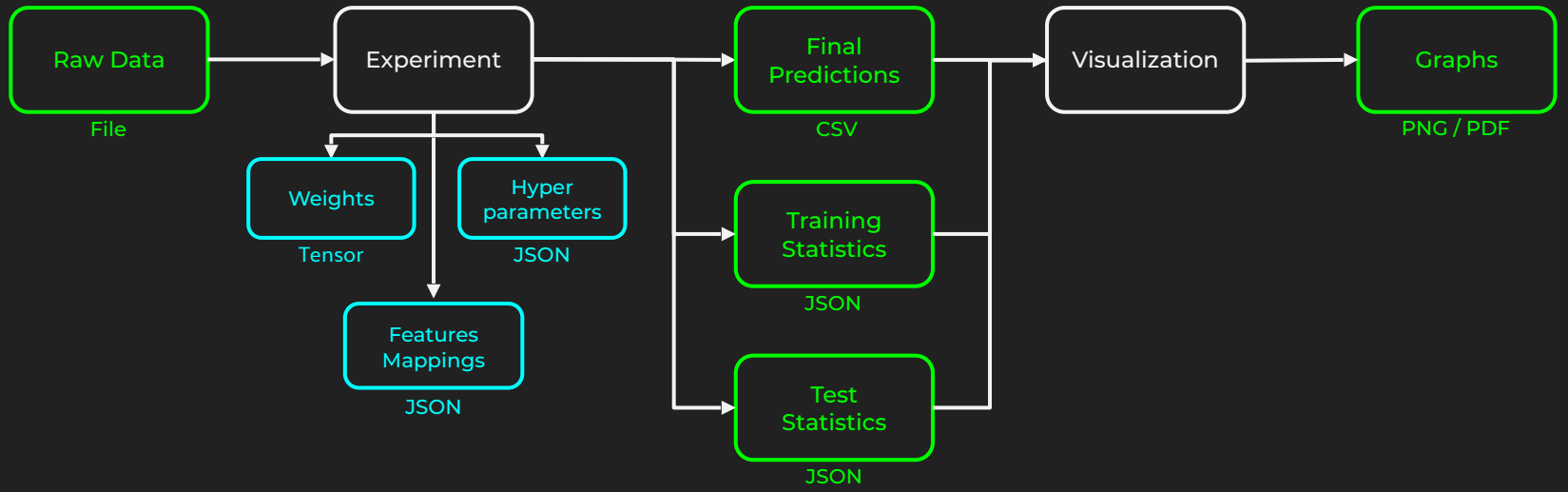
Prediction

The same fields mappings obtained during training are used to pre-process to each datapoint and to post-process each prediction of the model in order map back to labels



Running Experiments

The Experiment trains a model on the training set and predicts on the test set. It outputs predictions and training and test statistics that can be used by the Visualization component to obtain graphs



Under the hood

The magic lies in:

Data type abstraction

Declarative configuration

Smart use of `**kwargs`

Back to the example

```
ludwig experiment
--dataset my_dataset.csv
--config "{input_features: [{name: news, type: text}], output_features: [{name:
class, type: category}]}"
```

Configuration

```
ludwig experiment
--dataset my_dataset.csv
--config "{input_features: [{name: news, type: text}], output_features: [{name:
class, type: category}]}"
```

Merging with defaults it gets resolved to

```
{
input_features: [
  { encoder: parallel_cnn,
    level: word,
    name: news,
    tied_weights: None,
    type: text}],
combiner: {type: concat},
output_features: [
   { dependencies: [],
     loss: {   class_distance_temperature: 0,
               class_weights: 1,
               confidence_penalty: 0,
               distortion: 1,
               labels_smoothing: 0,
               negative_samples: 0,
               robust_lambda: 0,
               sampler: None,
               type: softmax_cross_entropy,
               unique: False,
               weight: 1},
     name: is_name,
     reduce_dependencies: sum,
     reduce_input: sum,
     top_k: 3,
     type: category}],
```

```
training: {
    batch_size: 128,
    bucketing_field: None,
    decay: False,
    decay_rate: 0.96,
    decay_steps: 10000,
    dropout_rate: 0.0,
    early_stop: 3,
    epochs: 20,
    gradient_clipping: None,
    increase_batch_size_on_plateau: 0,
    increase_batch_size_on_plateau_max: 512,
    increase_batch_size_on_plateau_patience: 5,
    increase_batch_size_on_plateau_rate: 2,
    learning_rate: 0.001,
    learning_rate_warmup_epochs: 5,
    optimizer: {   beta1: 0.9,
                   beta2: 0.999,
                   epsilon: 1e-08,
                   type: adam},
    reduce_learning_rate_on_plateau: 0,
    reduce_learning_rate_on_plateau_patience: 5,
    reduce_learning_rate_on_plateau_rate: 0.5,
    regularization_lambda: 0,
    regularizer: l2,
    staircase: False,
    validation_field: combined,
    validation_measure: loss},
```

```
preprocessing: {
  text: {
    char_format: characters,
    char_most_common: 70,
    char_sequence_length_limit: 1024,
    fill_value: ,
    lowercase: True,
    missing_value_strategy: fill_with_const,
    padding: right,
    padding_symbol: <PAD>,
    unknown_symbol: <UNK>,
    word_format: space_punct,
    word_most_common: 20000,
    word_sequence_length_limit: 256},
  category: {
    fill_value: <UNK>,
    lowercase: False,
    missing_value_strategy: fill_with_const,
    most_common: 10000},
  force_split: False,
  split_probabilities: (0.7, 0.1, 0.2),
  stratify: None
}
```

Merging with defaults it gets resolved to

```
{
input_features: [
  { encoder: parallel_cnn,
    level: word,
    name: news,
    tied_weights: None,
    type: text}],
combiner: {type: concat},
output_features: [
   { dependencies: [],
     loss: {   class_distance_temperature: 0,
               class_weights: 1,
               confidence_penalty: 0,
               distortion: 1,
               labels_smoothing: 0,
               negative_samples: 0,
               robust_lambda: 0,
               sampler: None,
               type: softmax_cross_entropy,
               unique: False,
               weight: 1},
     name: is_name,
     reduce_dependencies: sum,
     reduce_input: sum,
     top_k: 3,
     type: category}],
```

```
training: {
    batch_size: 128,
    bucketing_field: None,
    decay: False,
    decay_rate: 0.96,
    decay_steps: 10000,
    dropout_rate: 0.0,
    early_stop: 3,
    epochs: 20,
    gradient_clipping: None,
    increase_batch_size_on_plateau: 0,
    increase_batch_size_on_plateau_max: 512,
    increase_batch_size_on_plateau_patience: 5,
    increase_batch_size_on_plateau_rate: 2,
    learning_rate: 0.001,
    learning_rate_warmup_epochs: 5,
    optimizer: {   beta1: 0.9,
                   beta2: 0.999,
                   epsilon: 1e-08,
                   type: adam},
    reduce_learning_rate_on_plateau: 0,
    reduce_learning_rate_on_plateau_patience: 5,
    reduce_learning_rate_on_plateau_rate: 0.5,
    regularization_lambda: 0,
    regularizer: l2,
    staircase: False,
    validation_field: combined,
    validation_measure: loss},
```

```
preprocessing: {
  text: {
    char_format: characters,
    char_most_common: 70,
    char_sequence_length_limit: 1024,
    fill_value: ,
    lowercase: True,
    missing_value_strategy: fill_with_const,
    padding: right,
    padding_symbol: <PAD>,
    unknown_symbol: <UNK>,
    word_format: space_punct,
    word_most_common: 20000,
    word_sequence_length_limit: 256},
  category: {
    fill_value: <UNK>,
    lowercase: False,
    missing_value_strategy: fill_with_const,
    most_common: 10000},
  force_split: False,
  split_probabilities: (0.7, 0.1, 0.2),
  stratify: None
}
```

Merging with defaults it gets resolved to

```
{
input_features: [
combiner: {type: concat},
output_features: [
  { dependencies: [],
    loss: {   class_distance_temperature: 0,
              class_weights: 1,
              sampler: None,
              type: softmax_cross_entropy,
              unique: False,
              weight: 1},
    name: is_name,
```

INPUT FEATURES

COMBINER

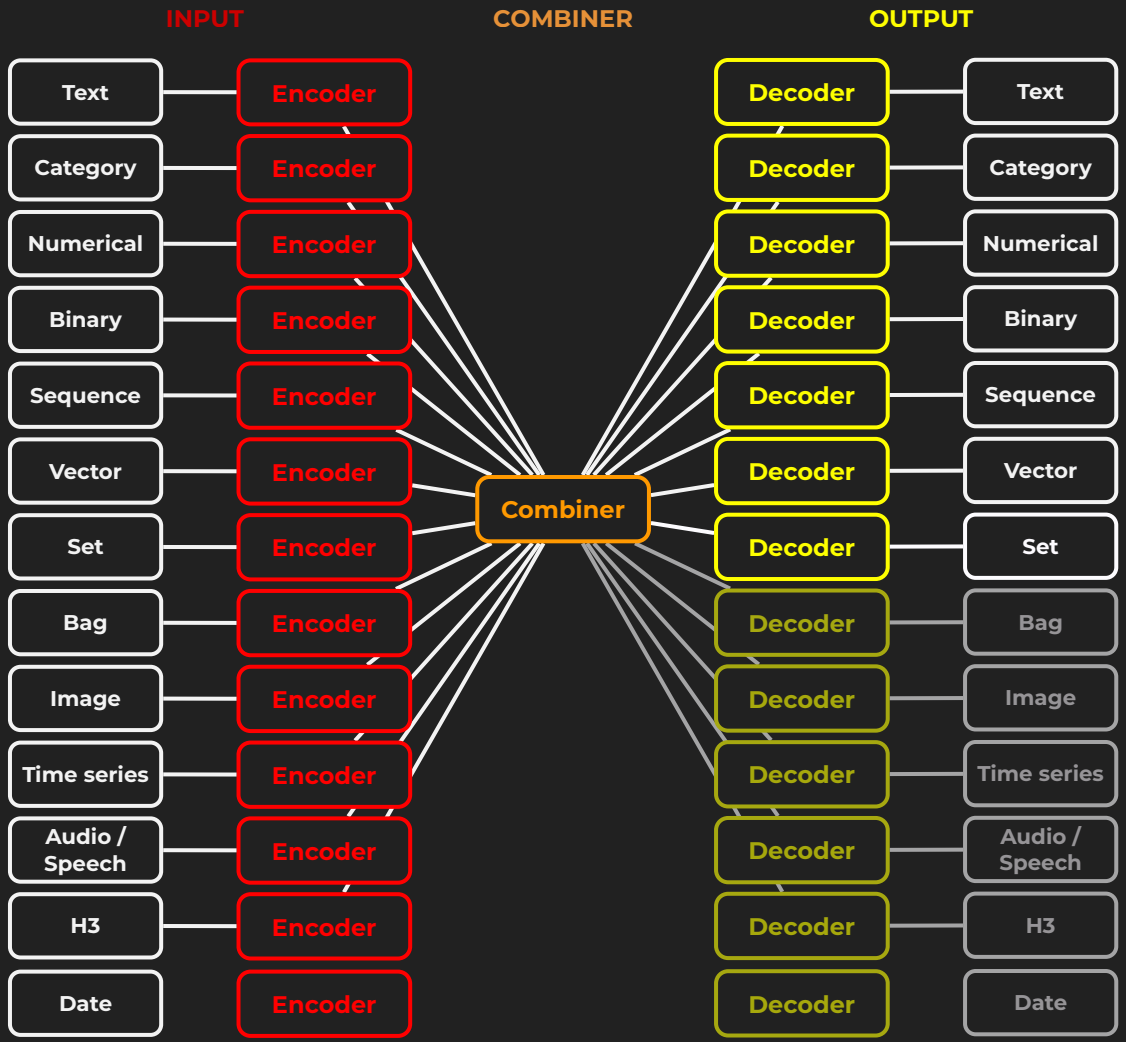
OUTPUT FEATURES

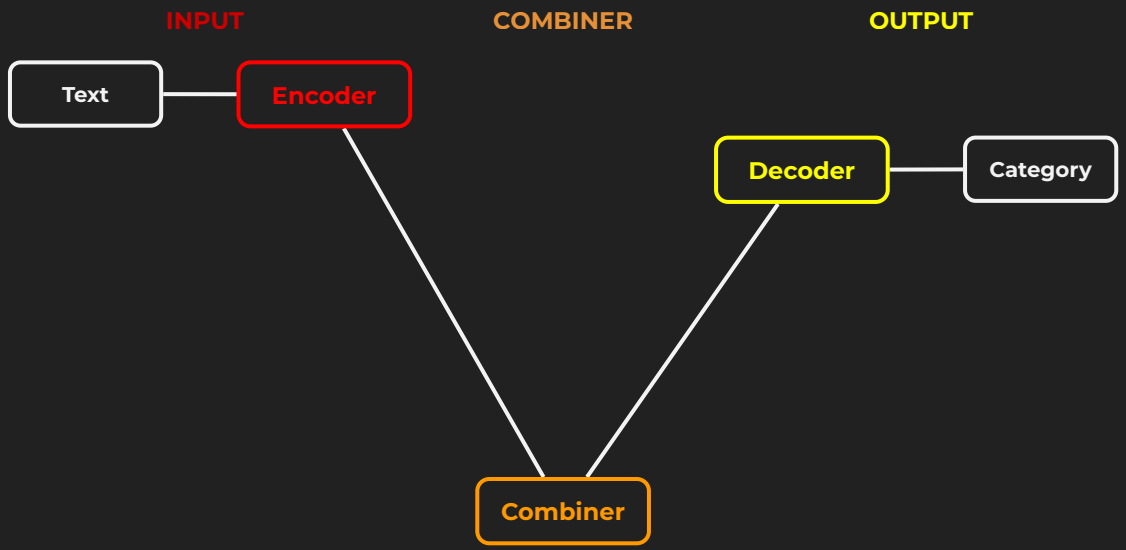
```
training: {
    batch_size: 128,
    bucketing_field: None,
    decay: False,
    decay_rate: 0.96,
    decay_steps: 10000,
    dropout_rate: 0.0,
    early_stop: 3,
    epochs: 20,
    gradient_clipping: None,
    increase_batch_size_on_plateau: 0,
    increase_batch_size_on_plateau_max: 512,
    optimizer: {
                beta2: 0.999,
                epsilon: 1e-08,
                type: adam},
    reduce_learning_rate_on_plateau: 0,
    reduce_learning_rate_on_plateau_patience: 5,
    reduce_learning_rate_on_plateau_rate: 0.5,
    regularization_lambda: 0,
    regularizer: l2,
    staircase: False,
    validation_field: combined,
    validation_measure: loss},
```

TRAINING

```
preprocessing: {
  text: {
    char_format: characters,
    char_most_common: 70,
    char_sequence_length_limit: 1024,
    fill_value: ,
    lowercase: True,
    missing_value_strategy: fill_with_const,
    padding: right,
    padding_symbol: <PAD>,
    unknown_symbol: <UNK>,
    word_format: space_punct,
    missing_value_strategy: fill_with_const,
    most_common: 10000},
  force_split: False,
  split_probabilities: (0.7, 0.1, 0.2),
  stratify: None
}
```

PREPROCESSING


INPUT
COMBINER
OUTPUT
Text
Category
Numerical
Binary
Sequence
Vector
Set
Bag
Image
Time series
Audio / Speech
H3
Date
Encoder
Encoder
Encoder
Encoder
Encoder
Encoder
Encoder
Encoder
Encoder
Encoder
Encoder
Encoder
Encoder
Combiner
Decoder
Decoder
Decoder
Decoder
Decoder
Decoder
Decoder
Decoder
Decoder
Decoder
Decoder
Decoder
Decoder
Text
Category
Numerical
Binary
Sequence
Vector
Set
Bag
Image
Time series
Audio / Speech
H3
Date



Text Classifier

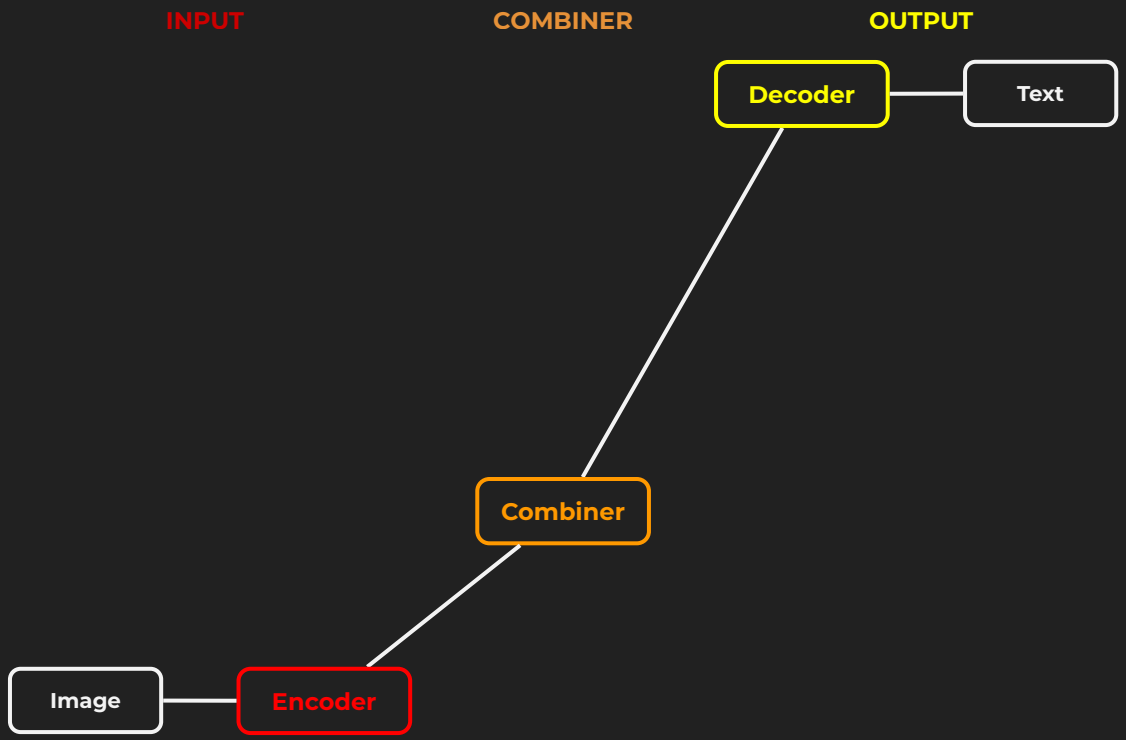
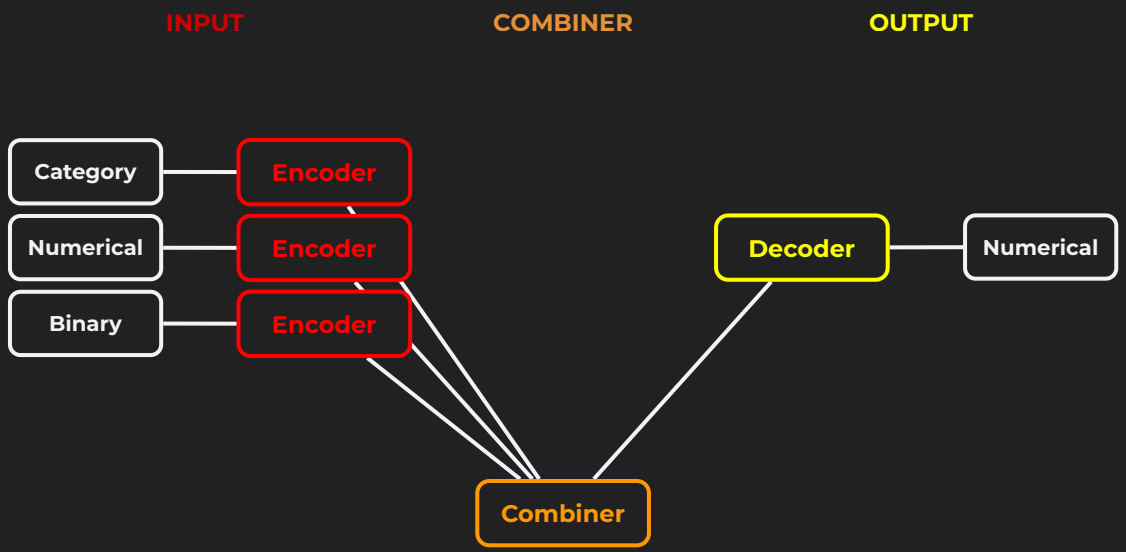
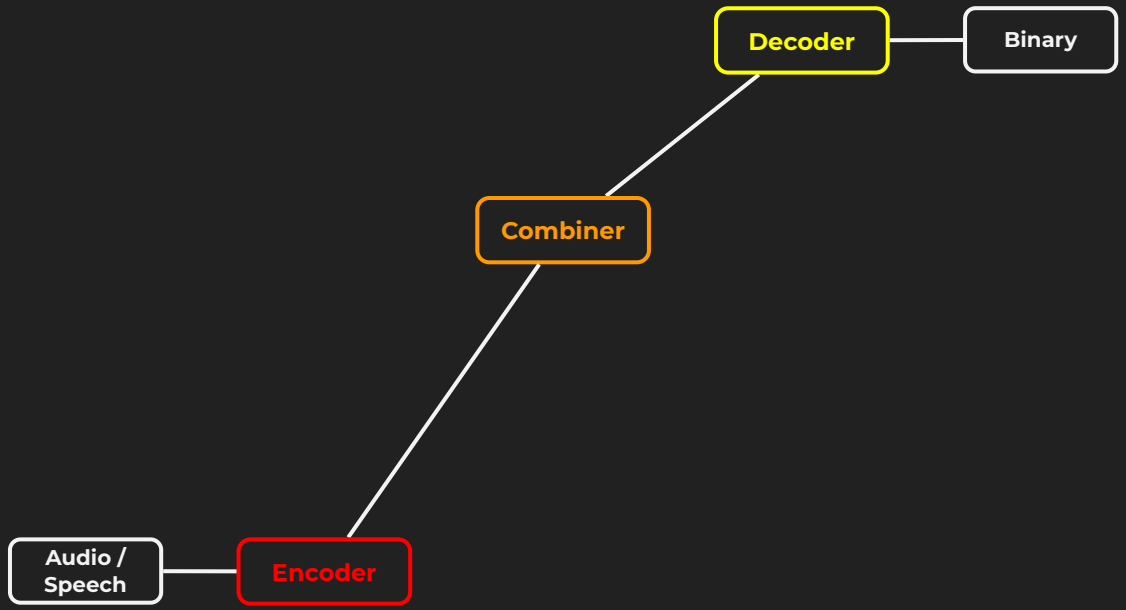


Image Captioning

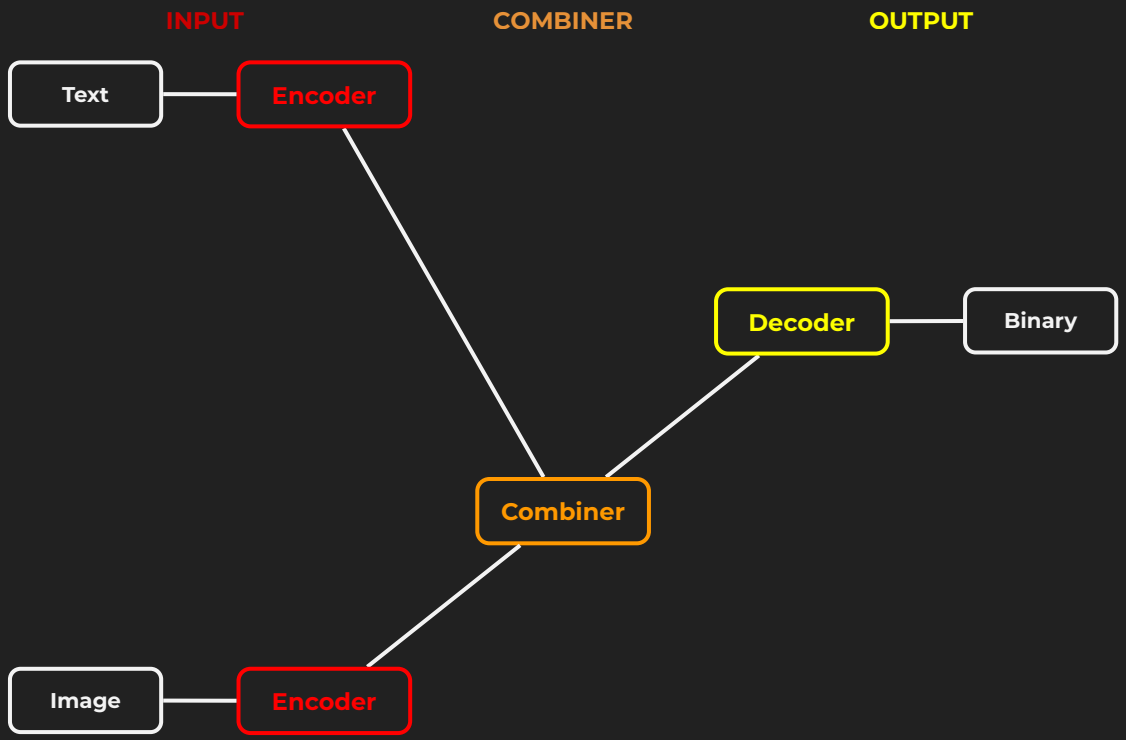


Classic Regression

INPUT COMBINER OUTPUT



Speaker Verification

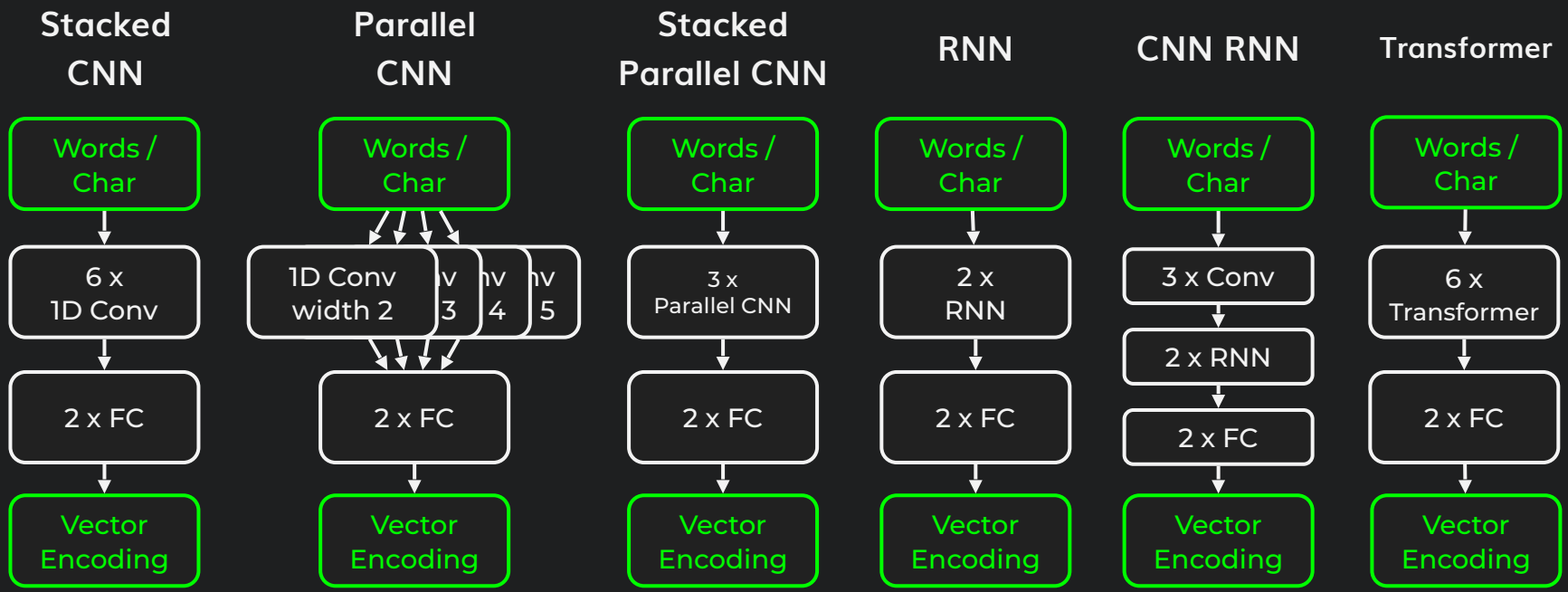


Visual Question Answering

Each input type can have multiple encoders to choose from

Each output type can have multiple decoders to choose from

Text input features encoders



ParallelCNN encoding

```
name: text_csv_column_name
type: text
encoder: parallel_cnn
level: word / char
tied_weights: null
representation: dense / sparse
embedding_size: 256
embeddings_on_cpu: false
pretrained_embeddings: null
embeddings_trainable: true
conv_layers: null
num_conv_layers: null
filter_size: 3
num_filters: 256
pool_size: null
fc_layers: null
num_fc_layers: null
fc_size: 256
activation: relu
norm: null
dropout: false
regularize: true
reduce_output: sum
```

Grey parameters are optional

ParallelCNN encoding

```
name: text_csv_col
type: text,
encoder: parallel_
level: word / char
tied_weights: null
representation: de
embedding_size: 25
embeddings_on_cpu:
pretrained_embeddi
embeddings_trainab
conv_layers: null
num_conv_layers: n
filter_size: 3
```

```python
class ParallelCNN(object):

    def __init__(
            self,
            should_embed=True,
            vocab=None,
            representation='dense',
            embedding_size=256,
            embeddings_trainable=True,
            pretrained_embeddings=None,
            embeddings_on_cpu=False,
            conv_layers=None,
            num_conv_layers=None,
            filter_size=3,
            num_filters=256,
            pool_size=None,
            fc_layers=None,
            num_fc_layers=None,
            fc_size=256,
            norm=None,
            activation='relu',
            dropout=False,
            initializer=None,
            regularize=True,
            reduce_output='max',
            **kwargs):
```

Code that builds a parallel_cnn

StackedCNN encoding

```
name: text_csv_column_name
type: text
encoder: stacked_cnn
level: word / char
tied_weights: null
representation: dense / sparse
embedding_size: 256
embeddings_on_cpu: false
pretrained_embeddings: null
embeddings_trainable: true
conv_layers: null
num_conv_layers: null
filter_size: 3
num_filters: 256
pool_size: null
fc_layers: null
num_fc_layers: null
fc_size: 256
activation: relu
norm: null
dropout: false
initializer: null
regularize: true
reduce_output: max
```

Grey parameters are optional

StackedParallelCNN encoding

```
name: text_csv_column_name
type: text
encoder: stacked_parallel_cnn
level: word / char
tied_weights: null
representation: dense / sparse
embedding_size: 256
embeddings_on_cpu: false
pretrained_embeddings: null
embeddings_trainable: true
stacked_layers: null
num_stacked_layers: null
filter_size: 3
num_filters: 256
pool_size: null
fc_layers: null
num_fc_layers: null
fc_size: 256
norm: null
activation: relu
regularize: true
reduce_output: max
```

Grey parameters are optional

RNN encoding

```
name: text_csv_column_name
type: text
encoder: rnn
level: word / char
tied_weights: null
representation: dense / sparse
embedding_size: 256
embeddings_on_cpu: false
pretrained_embeddings: null
embeddings_trainable: true
num_layers: 1
cell_type: rnn
state_size: 256
bidirectional: false
dropout: false
initializer: null
regularize: true
reduce_output: sum
```

Grey parameters are optional

CNN-RNN encoding

```
name: text_csv_column_name
type: text
encoder: cnn_rnn
level: word / char
tied_weights: null
representation: dense / sparse
embedding_size: 256
embeddings_on_cpu: false
pretrained_embeddings: null
embeddings_trainable: true
conv_layers: null
num_conv_layers: null
filter_size: 3
num_filters: 256
pool_size: null
norm: null
activation: relu
num_rec_layers: 1
cell_type: rnn
state_size: 256
bidirectional: false
dropout: false
initializer: null
regularize: true
reduce_output: last
```

Grey parameters are optional

Transformer encoding

```
name: text_csv_column_name
type: text
encoder: transformer
level: word / char
representation: dense
embedding_size: 256
embeddings_trainable: True
pretrained_embeddings: None
embeddings_on_cpu: False
num_layers: 1
hidden_size: 256
num_heads: 8
transformer_fc_size: 256
dropout: 0.1
```

```
fc_layers: None
num_fc_layers: 0
fc_size: 256
use_bias: True
weights_initializer: glorot_uniform
bias_initializer: zeros
weights_regularizer: None
bias_regularizer: None
activity_regularizer: None
norm: None
norm_params: None
fc_activation: relu
fc_dropout: 0
reduce_output: last
```

Grey parameters are optional

Sequence, Time Series and Audio / Speech encoders

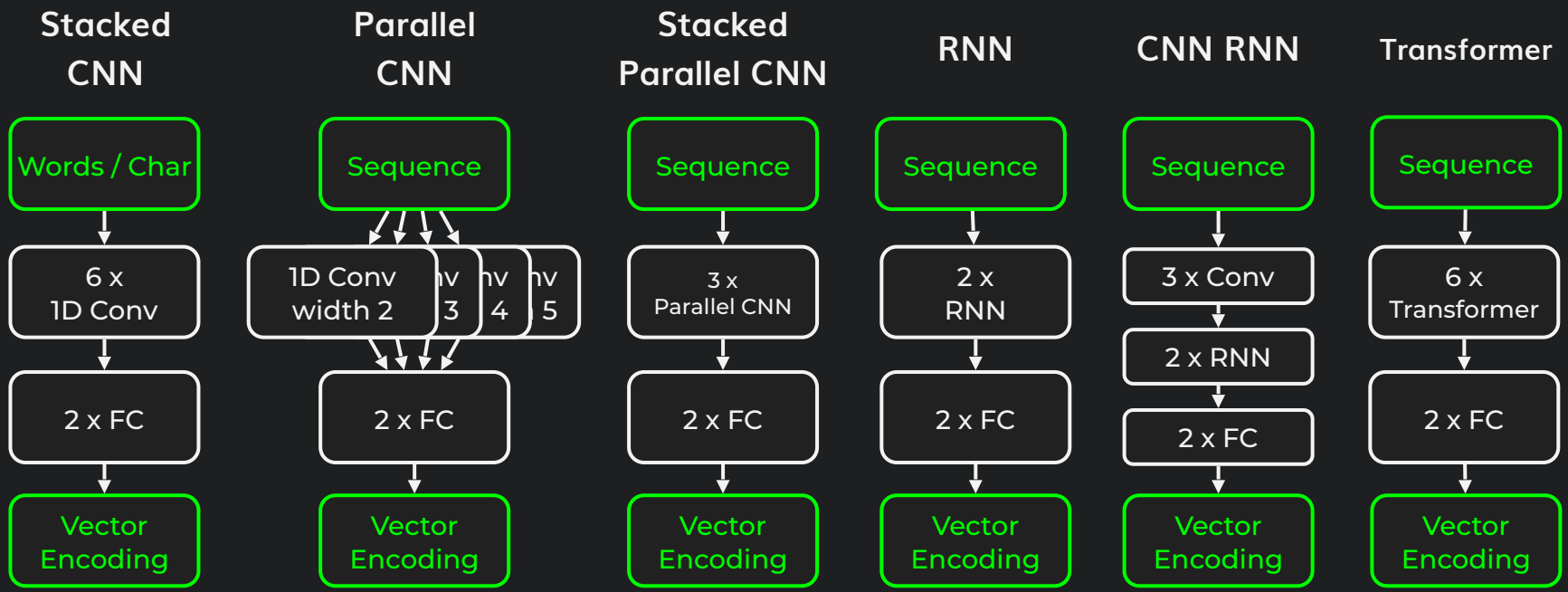
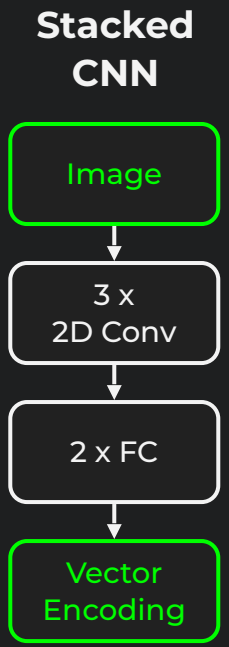
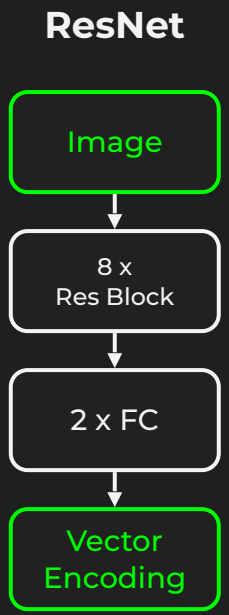


Image feature encoders

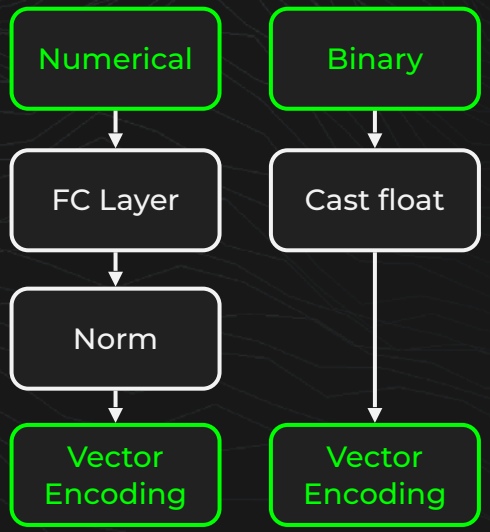




Numerical and binary features encoders

Numerical features are encoded with one FC layer + Norm for scaling purposes or are used as they are

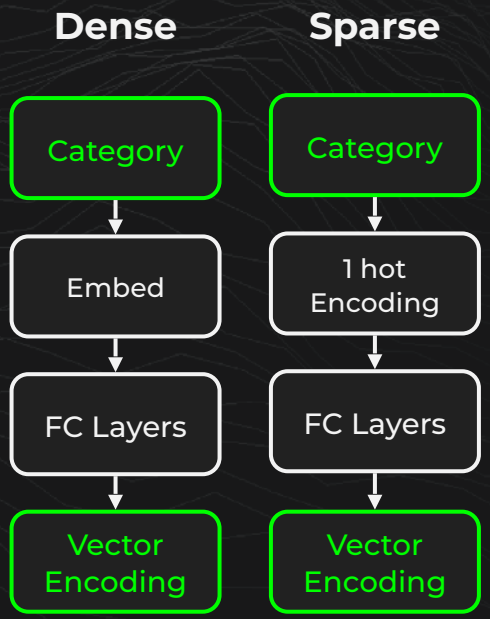
Binary features are just casted to floats



Category features encoders

If the dense encoder is specified, the embedding dimension cal also be provided

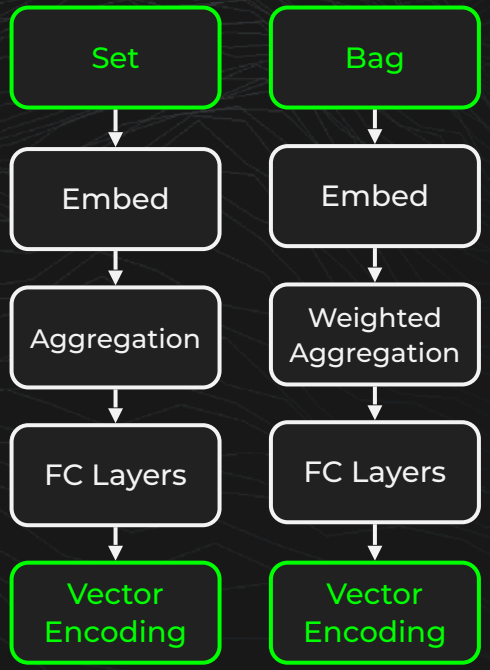
Alternatively the sparse 1 hot encoder can be used



Set and bag features encoders

They are both encoded by embedding the items, aggregating them, and passing them through FC layers

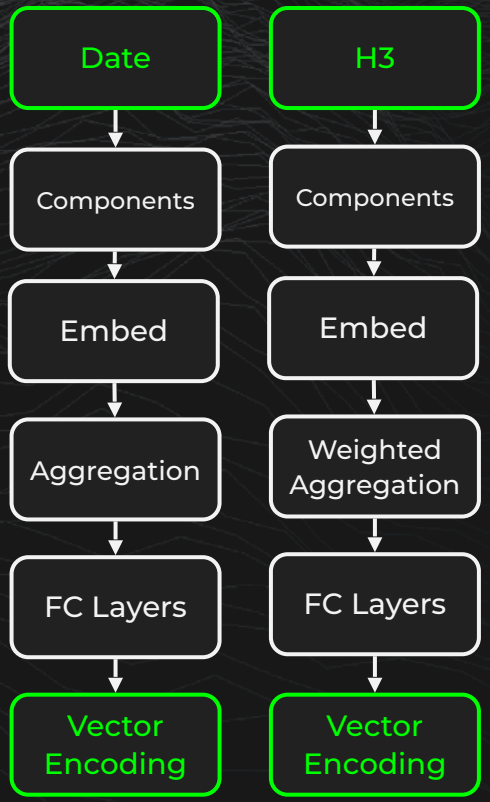
In the case of bags, the aggregation is a weighted sum



Date and H3 features encoders

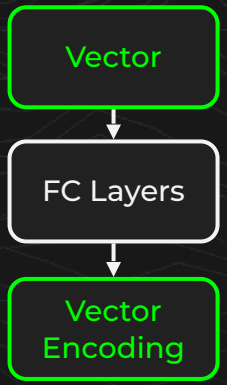
Dates are divided in components, components are embedded and aggregated

H3s (spatial indexing system) are divided in hierarchical components, embedded and aggregated

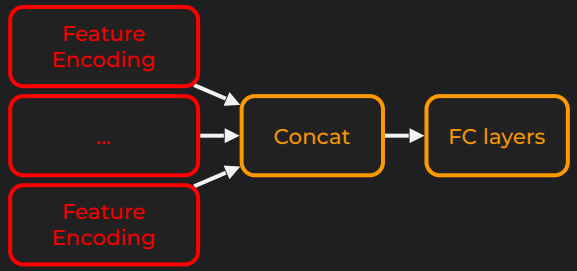


Vector features encoder

Vectors a simply passed through FC layers



Concat combiner



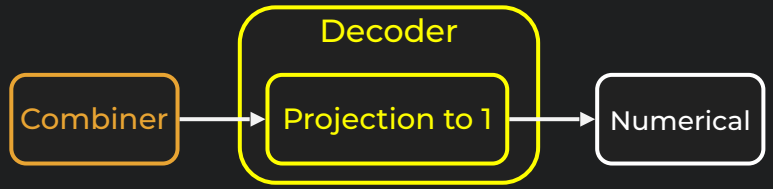

```
combiner:
  type: concat
  fc_layers: null
  num_fc_layers: 0
  fc_size: 256
  activation: relu
  norm: null
  dropout: false
  initializer: null
  regularize: true
```

Grey parameters are optional

Numerical features decoding

```
name: numerical_csv_column_name
type: numerical
reduce_inputs: sum
loss:
    type: mean_squared_error
fc_layers: null
num_fc_layers: 0
fc_size: 256
activation: relu
norm: null
dropout: false
initializer: null
regularize: true
```

Grey parameters are optional

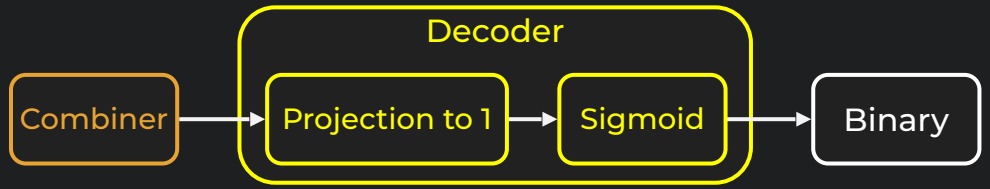


Binary features decoding

```
name: binary_csv_column_name
type: binary
reduce_inputs: sum
loss:
    type: cross_entropy
    confidence_penalty: 0
    robust_lambda: 0
fc_layers: null
num_fc_layers: 0
```

```
fc_size: 256
activation: relu
norm: null
dropout: false
initializer: null
regularize: true
threshold: 0.5
```

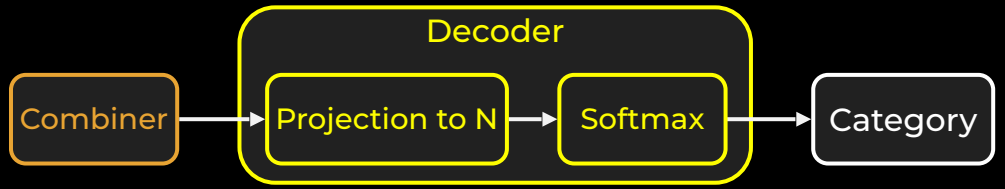
Grey parameters are optional



Category features decoding

```
name: category_csv_column_name
type: category
reduce_inputs: sum
loss:
  type: softmax_cross_entropy
  confidence_penalty: 0
  robust_lambda: 0
  class_weights: 1
  class_distances: null
  class_distance_temperature: 0
  labels_smoothing: 0
  negative_samples: 0
  sampler: null
  distortion: 1
  unique: false
fc_layers: null
num_fc_layers: 0
fc_size: 256
activation: relu
norm: null
dropout: false
initializer: null
regularize: true
top_k: 3
```

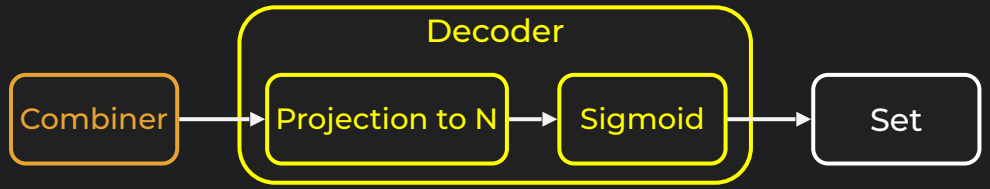
Grey parameters are optional



Set features decoding

```
name: set_csv_column_name
type: set
reduce_inputs: sum
loss:
    type: sigmoid_cross_entropy
fc_layers: null
num_fc_layers: 0
fc_size: 256
activation: relu
norm: null
dropout: false
initializer: null
regularize: true
threshold: 0.5
```

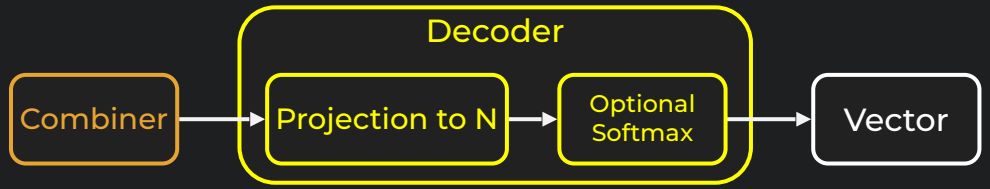
Grey parameters are optional



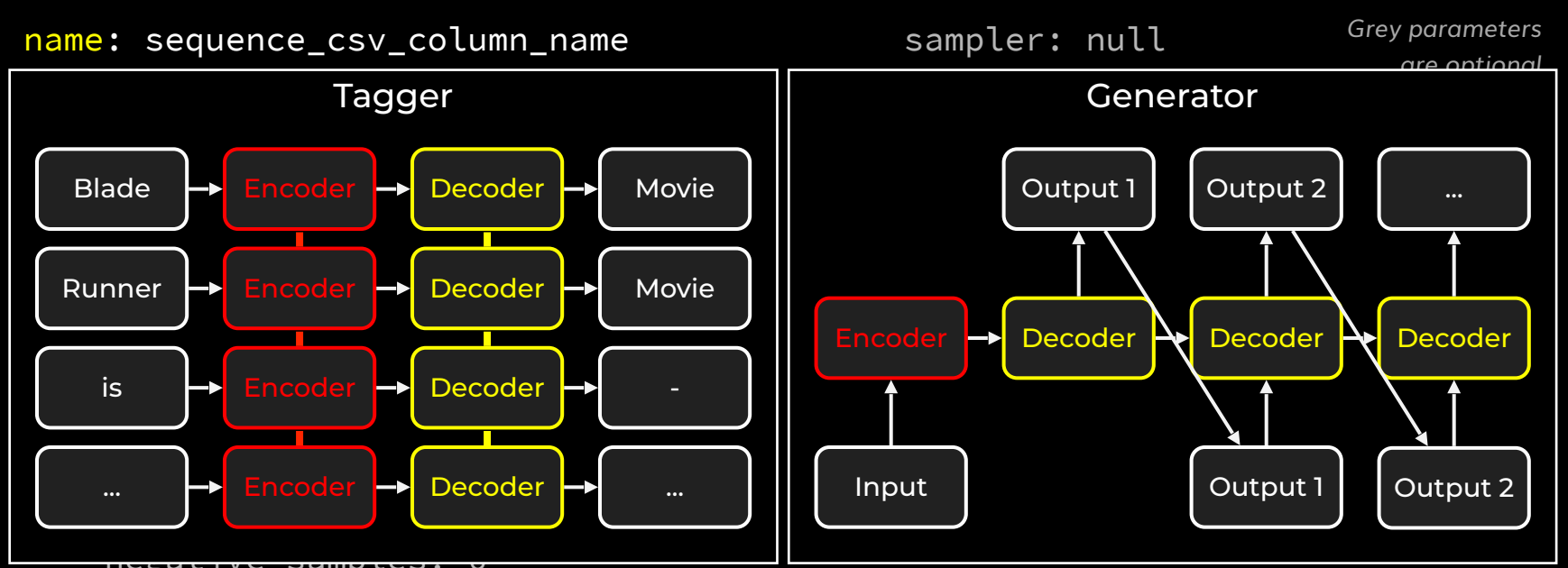
Vector features decoding

```
name: vector_csv_column_name
type: vector
reduce_inputs: sum
loss:
    type: mse / cross_entropy
fc_layers: null
num_fc_layers: 0
fc_size: 256
activation: relu
norm: null
dropout: false
initializer: null
regularize: true
```

Grey parameters are optional

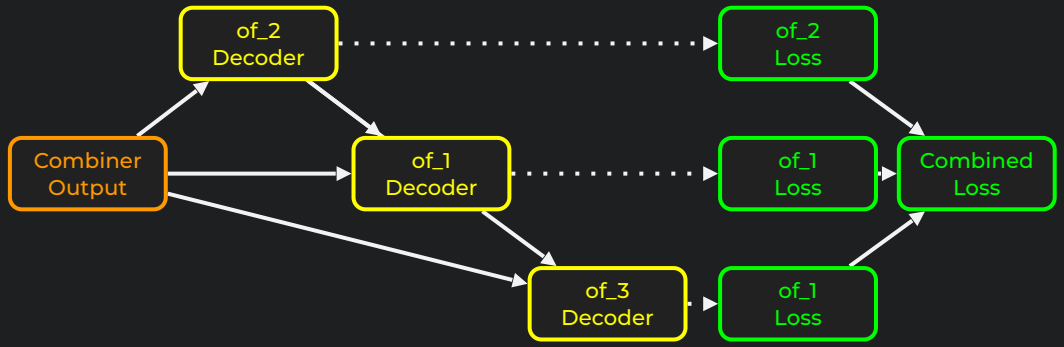


Sequence features decoding



Output features dependencies DAG

```
[{name: of_1,
  type: any_type,
  dependencies: [of_2]},
 {name: of_2,
  type: any_type,
  dependencies: []},
 {name: of_3,
  type: any_type,
  dependencies: [of_2, of_1]}]
```



Training parameters

```
training:
  batch_size: 128
  epochs: 100
  early_stop: 5
  optimizer: {type: adam, beta1: 0.9, beta2: 0.999, epsilon: 1e-08}
  learning_rate: 0.001
  decay: false
  decay_rate: 0.96
  decay_steps: 10000
  staircase: false
  regularization_lambda: 0
  reduce_learning_rate_on_plateau: 0
  reduce_learning_rate_on_plateau_patience: 5
  reduce_learning_rate_on_plateau_rate: 0.5
  increase_batch_size_on_plateau: 0
  increase_batch_size_on_plateau_patience: 5
  increase_batch_size_on_plateau_rate: 2
  increase_batch_size_on_plateau_max: 512
  validation_field: combined
  validation_measure: accuracy
  bucketing_field: null
```

Everything is optional

Preprocessing parameters

```
preprocessing:
  force_split: False,
  split_probabilities: (0.7, 0.1, 0.2),
  stratify: None
  text:
    char_format: characters,
    char_most_common: 70,
    char_sequence_length_limit: 1024,
    fill_value: ,
    lowercase: True,
    missing_value_strategy: fill_with_const,
    padding: right,
    padding_symbol: <PAD>,
    unknown_symbol: <UNK>,
    word_format: space_punct,
    word_most_common: 20000,
    word_sequence_length_limit: 256,
  category:
    fill_value: <UNK>,
    lowercase: False,
    missing_value_strategy: fill_with_const,
    most_common: 10000
sequence:
  ...
set:
  ...
...
```

Everything is optional

Supports text preprocessing in English, Italian, Spanish, German, French, Portuguese, Chinese, Japanese, Dutch, Greek, Polish, Romanian, Danish, Norwegian, Lithuanian and Multi-language

Example Models

Text Classification

NEWS	CLASS
Toronto Feb 26 - Standard Trustco said it expects earnings in 1987 to increase at least 15...	earnings
New York Feb 26 - American Express Co remained silent on market rumors...	acquisition
BANGKOK March 25 - Vietnam will resettle 300 000 people on state farms known as new economic...	coffe

```
ludwig experiment
--dataset reuters-allcats.csv
--config "{input_features: [{name: news,
type: text, encoder: parallel_cnn, level:
word}], output_features: [{name: class,
type: category}]}"
```

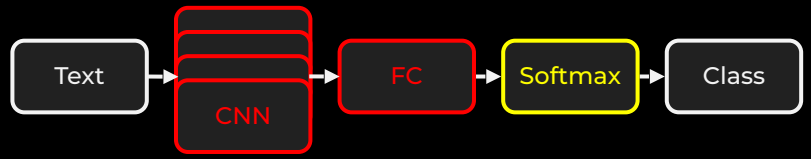


Image Object Classification (MNIST, ImageNet, ..)

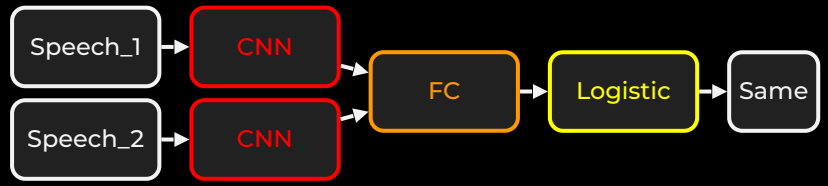
IMAGE_PATH	CLASS
imagenet/image_000001.jpg	car
imagenet/image_000002.jpg	dog
imagenet/image_000003.jpg	boat

```
ludwig experiment
--dataset imagenet.csv
--config "{input_features: [{name:
image_path, type: image, encoder:
stacked_cnn}], output_features: [{name:
class, type: category}]}"
```

Speaker Verification

SPEECH_PATH_1	SPEECH_PATH_2	SAME
speech_01_01.wav	speech_01_02.wav	True
speech_01_02.wav	speech_02_01.wav	False
speech_02_01.wav	speech_02_02.wav	False

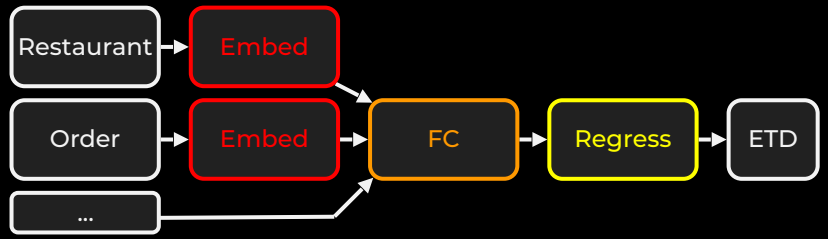
```
ludwig experiment
--dataset speakers.csv
--config "{input_features: [{name:
speech_path_1, type: audio}, {name:
speech_path_2, type: audio, tied_weights:
speech_path_1}], output_features: [{name:
same, type: binary}]}"
```



Expected Time of Delivery

RESTAURANT	ORDER	HOUR	MIN	ETD
Halal Guys	Chicken kebab, Lamb kebab	20	00	20
Il Casaro	Margherita	18	53	15
CurryUp	Chicken Tikka Masala, Veg Curry	21	10	35

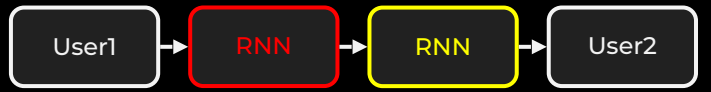
```
ludwig experiment
--dataset eats_etd.csv
--config "{input_features: [{name: restaurant,
type: category, encoder: embed}, {name: order,
type: bag}, {name: hour, type: numerical},
{name: min, type: numerical}],
output_features: [{name: etd, type: numerical,
loss: {type: mean_absolute_error}}]}"
```



Sequence2Sequence Chit-Chat Dialogue Model

USER1	USER2
Hello! How are you doing?	Doing well, thanks!
I got promoted today	Congratulations!
Not doing well today	I'm sorry, can I do something to help you?

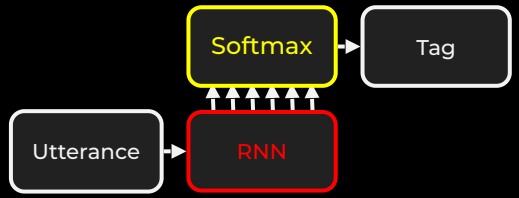
```
ludwig experiment
--dataset chitchat.csv
--config "{input_features: [{name: user1,
type: text, encoder: rnn, cell_type:
lstm}], output_features: [{name: user2,
type: text, decoder: generator, cell_type:
lstm, attention: bahdanau}]}"
```



Sequence Tagging for sequences or time series

UTTERANCE	TAG
Blade Runner is a 1982 neo-noir science fiction film directed by Ridley Scott	Movie Movie - - Date - - - - - - Person Person
Harrison Ford and Rutger Hauer starred in it	Person Person - Person Person - - -
Philip Dick 's novel Do Androids Dream of Electric Sheep ? was published in 1968	Person Person - - Book Book Book Book Book Book - - - Date

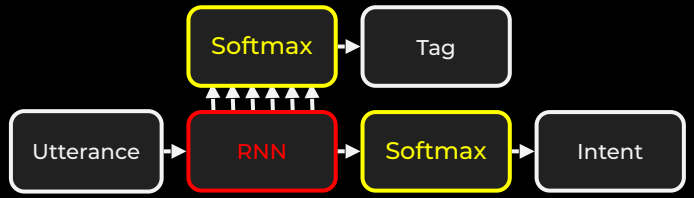
```
ludwig experiment
--dataset sequence_tags.csv
--config "{input_features: [{name:
utterance, type: text, encoder: rnn,
cell_type: lstm}], output_features: [{name:
tag, type: sequence, decoder: tagger}]}"
```



Natural Language Understanding

UTTERANCE	INTENT	TAG
I want to fly from Boston at 838 am	Flight	----- CITY - TIME TIME
What flights are available from NYC to Boston ?	Flight	----- CITY - CITY -
Cheapest airfare from LA to NYC	Airfare	COST_REL - - CITY - CITY

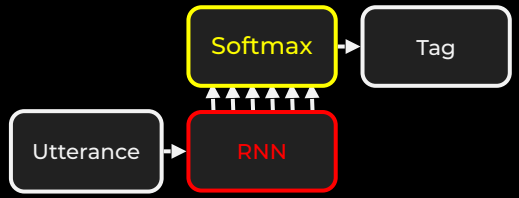
```
ludwig experiment
--dataset nlu.csv
--config "{input_features: [{name: utterance,
type: text, encoder: rnn, cell_type: lstm}],
output_features: [{name: intent, type:
category}, {name: tag, type: sequence,
decoder: tagger}]}"
```



Summarization

UTTERANCE	KEEP
Blade Runner is a 1982 neo-noir science fiction film directed by Ridley Scott	1 1 1 1 0 0 1 1 1 0 0 0 0
Harrison Ford and Rutger Hauer starred in it	1 1 0 0 0 0 1 1
Philip Dick 's novel Do Androids Dream of Electric Sheep ? was published in 1968	0 0 0 0 1 1 1 1 1 1 1 1 1 1

```
ludwig experiment
--dataset summarization.csv
--config "{input_features: [{name:
utterance, type: text, encoder: rnn,
cell_type: lstm}], output_features: [{name:
keep, type: sequence, decoder: tagger}]}"
```



Programmatic API

Easy to install:

```
pip install ludwig
```

Programmatic API:

```
from ludwig.api import LudwigModel
model = LudwigModel(config)
model.train(train_data)
model.save(path)
model = LudwigModel.load(path)
predictions = model.predict(other_data)
```

Model serving:

```
ludwig serve --model_path <model_path>
```


Visualizations

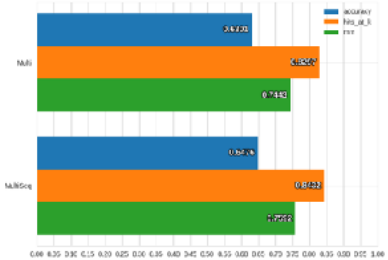
Outputs

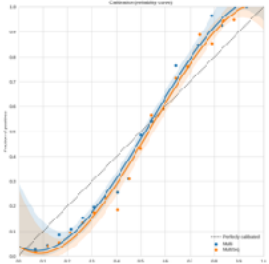
A directory containing:

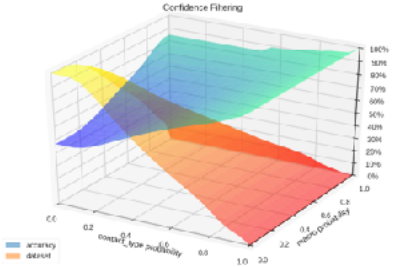
- trained model
- predictions CSV file for each output feature
- probabilities NPY file for each output feature
- training statistics JSON file
- test statistics JSON file

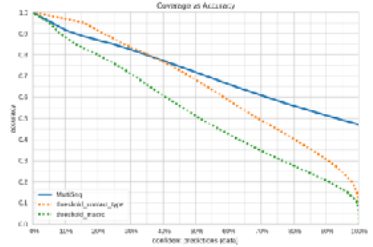
Monitoring during training with TensorBoard

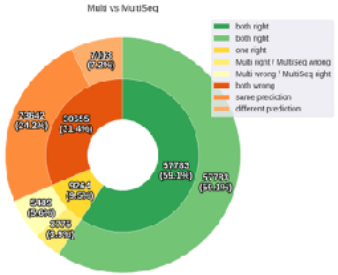
Visualizations are one command away

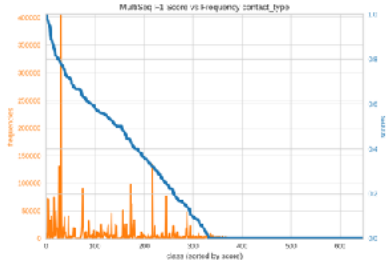


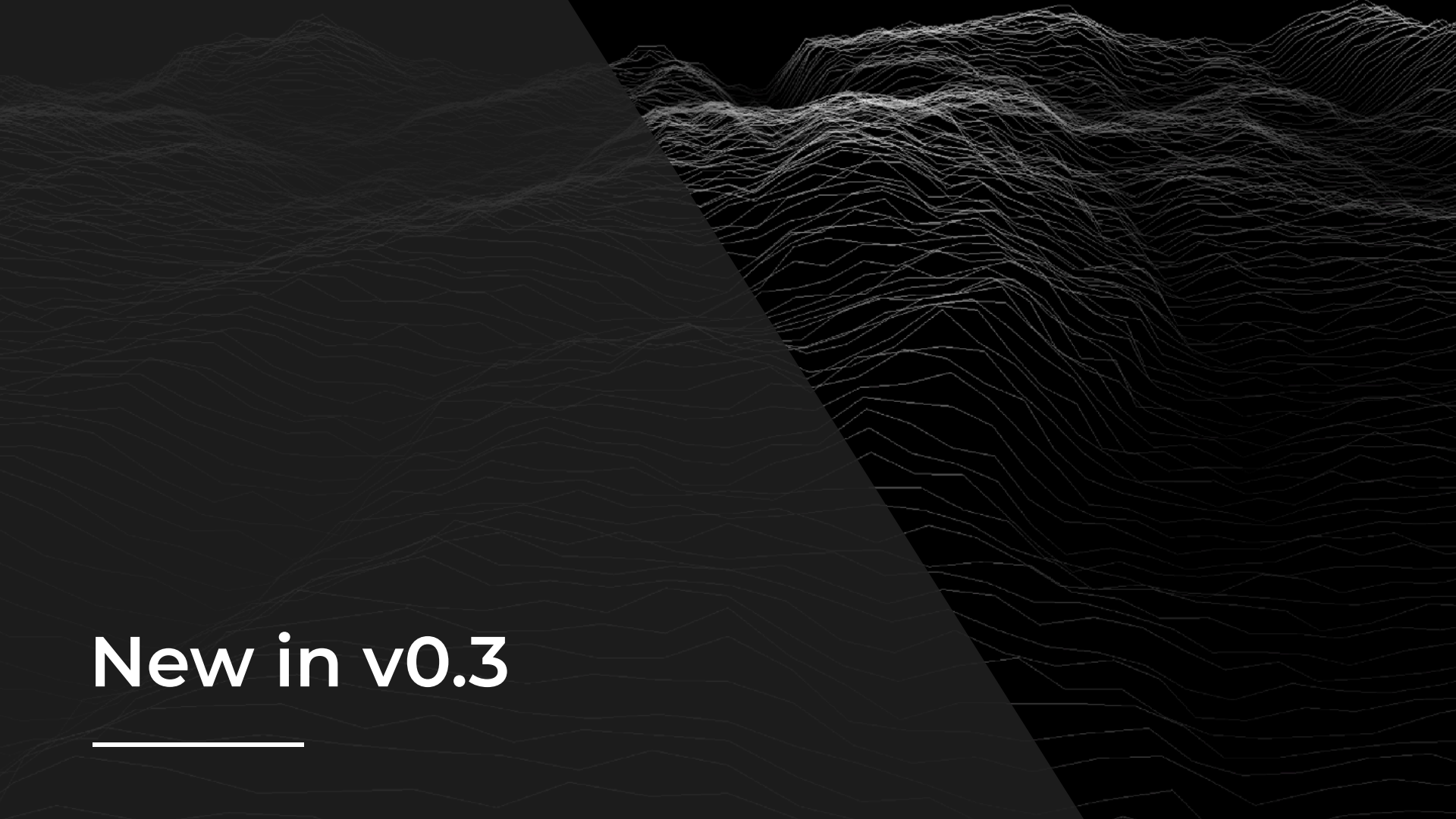












New in v0.3

Overview

Features

- Hyper-parameters optimization using **PySOT / Fiber**
- **Many new supported data formats**
- **Vector type (useful for noisy / weak supervision)**
- **K-fold cross validation**
- Deploy models seamlessly using **SavedModel / Neuropod**

Improvements and Integrations

- **Porting to TensorFlow 2 object-oriented API**
- **Integration of Hugging Face Transformers pre-trained models**
- Train models distributedly using **Horovod**
- **Weights and Biases integration (Comet.ml was added in v0.2)**
- **Custom model interface**

Hyper-parameter optimization

```yaml
input_features:
  -
    name: utterance
    type: text
    encoder: rnn
    cell_type: lstm
    num_layers: 2
output_features:
  -
    name: class
    type: category
training:
  learning_rate: 0.001
  optimizer:
    type: adam
```

```yaml
hyperopt:
 metric: accuracy
 parameters:
    utterance.num_layers:
      type: int
      low: 1
      high: 5
    training.learning_rate:
      type: float
      low: 0.0001
      high: 0.1
      scale: log
    training.optimizer.type:
      type: category
      values: [sgd, adam, adagrad]
  sampler:
    type: random / grid / pysot
    num_samples: 12
  executor:
    type: parallel / serial / fiber
    num_workers: 4
```

Hyper-parameter optimization

```
hyperopt:
 output_feature: class
 metric: accuracy
 goal: maximize
 split: validation
```

Hyper-parameter optimization

```
parameters:
    utterance.num_layers:
      type: int
      low: 1
      high: 5
    training.learning_rate:
      type: float
      low: 0.0001
      high: 0.1
      scale: log
    training.optimizer.type:
      type: category
      values: [sgd, adam, adagrad]
```

Hyper-parameter optimization

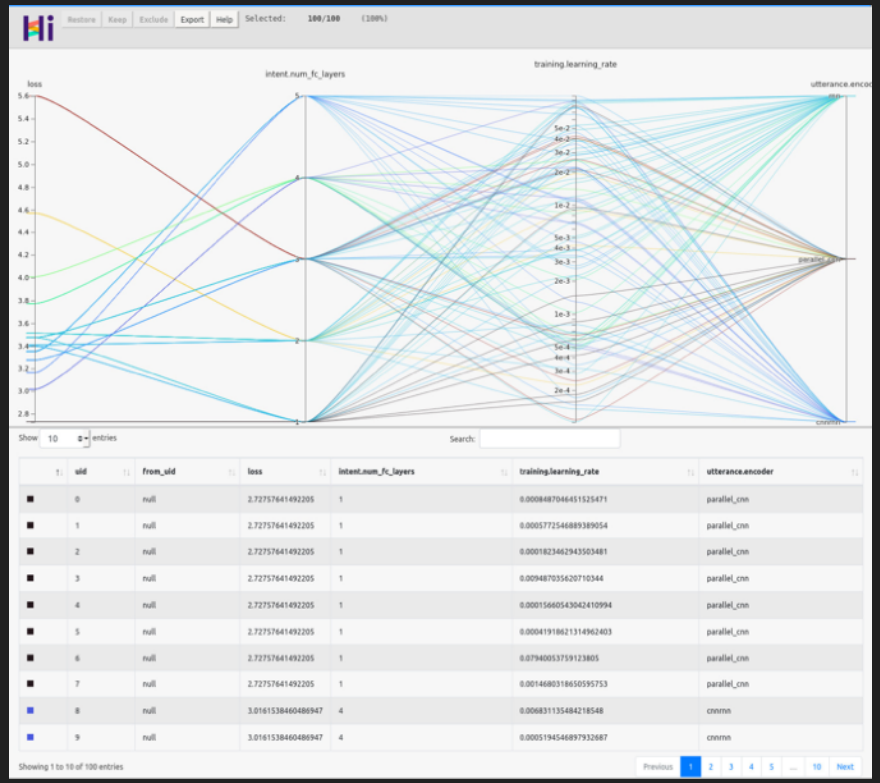
```
sampler:
  type: random / grid / pysot
  num_samples: 12
    ... sampler parameters ...
```

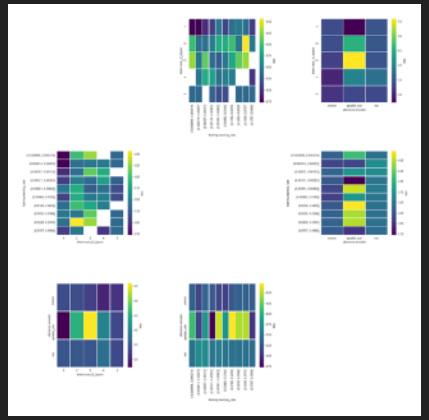
Hyper-parameter optimization

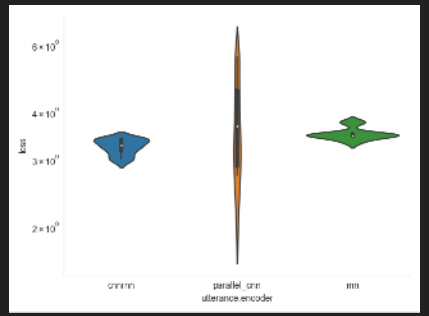
```
executor:
  type: parallel / serial / fiber
  num_workers: 4
  ... executor parameters ...
```

Ray & RayTune integration coming soon

Hyperopt Visualization







Transformers

```
input_features:
  -
    name: utterance
    type: text
    encoder: bert / distilbert / t5 / electra / gpt-2 / ...
```

Training distributedly with Horovod

```
horovodrun -np 4 ...other Horovod paramters...
ludwig experiment
--dataset my_dataset.csv
--config "{input_features: [{name: news, type: text}], output_features: [{name:
class, type: category}]}"
--use_horovod
```

New data formats

```
ludwig experiment
--dataset my_dataset
--data_format auto / csv / tsv / excel / parquet / json / jsonl / hdf5 / ...
```

Neuropod and SavedModel export

```
ludwig export_neuropod / export_savedmodel
--model_path my_model
--output_path output_model
```

K-fold cross validation

```
ludwig experiment
--dataset reuters-allcats.csv
--config "{input_features: [{name: news, type: text}], output_features: [{name:
class, type: category}]}"
--k_fold 10
```

Weights and Biases / Comet

```
ludwig experiment
--dataset reuters-allcats.csv
--config "{input_features: [{name: news, type: text}], output_features: [{name:
class, type: category}]}"
--wandb / comet
```

Custom Encoder TensorFlow 2 Example

```
class MyImageEncoder(Layer):

  def __init__(
    self,
    my_param=whatever,
    **kwargs
  ):
    self.my_param = my_param
    … other init …

  def call(
    self,
    inputs,
    is_training=None,
    mask=None
  ):
    # inputs size depends
    # on data type
    # for images it is
    # [batch, h, w, c]

    … TF2 code of my encoder …

    # hidden shape has to be
    # [batch, new_h, new_w, new_c]
    return hidden
```

Custom model interface

```
class MyModel(Model):

  def __init__(
    self,
    my_param=whatever,
    **kwargs
  ):
    self.my_param = my_param
    self.loss_f = …
    … other init …

  def call(
    self,
    inputs,
    is_training=None,
    mask=None
  ):
    … TF2 code of my encoder …
    return hidden

def train_step(self, batch):
  x, y = batch
  loss = self.loss_f(self(x), y)
  return {'loss': loss, ...}
```

How can you help?

- PyTorch backend
- Remote data streaming
- RayTune integration
- Autoencoders support
- Test battery with SOTA
- Data augmentation
- Model hub for trained models / encoders / decoders
- Tutorials and training material
- Complex combiners
- More features: nested lists, point clouds, videos
- More pre-trained inputs encoders
- More image encoders (ResNext, DenseNet, MobileNet)
- Adding missing decoders

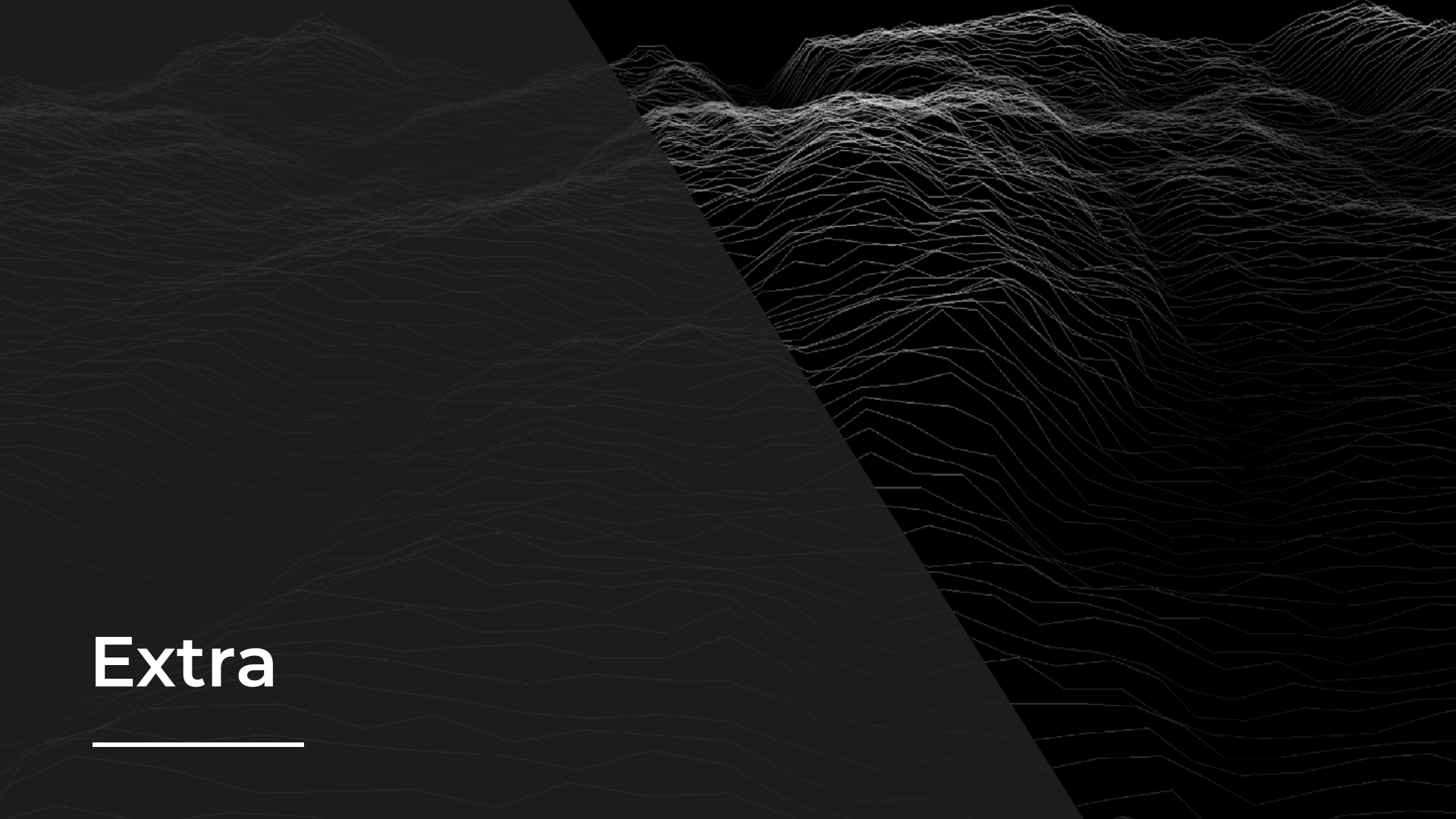
Documentation
http://ludwig.ai

Repository
http://github.com/uber/ludwig

Blogpost
http://eng.uber.com/introducing-ludwig
http://eng.uber.com/ludwig-v0-2/
http://eng.uber.com/ludwig-v0-3/

White paper
https://arxiv.org/abs/1909.07930

Key contributors
Travis Addair
Yaroslav Dudin
Sai Sumanth Miryala
Jim Thompson
Avanika Narayan
Ivaylo Stefanov
John Wahba
Doug Blank
Patrick von Platen
Carlo Grisetti
Chris Van Pelt
Boris Dayma



Extra

Machine Learning democratization

Non-expert users

From **idea** to **model** training in **minutes**

No need for **coding and deep knowledge**

Easy **declarative** model construction hides **complexity**

Expert users

Efficient exploration of model space through **quick iteration**

Tweak **every aspect** of preprocessing, modeling and training

Extend with your models