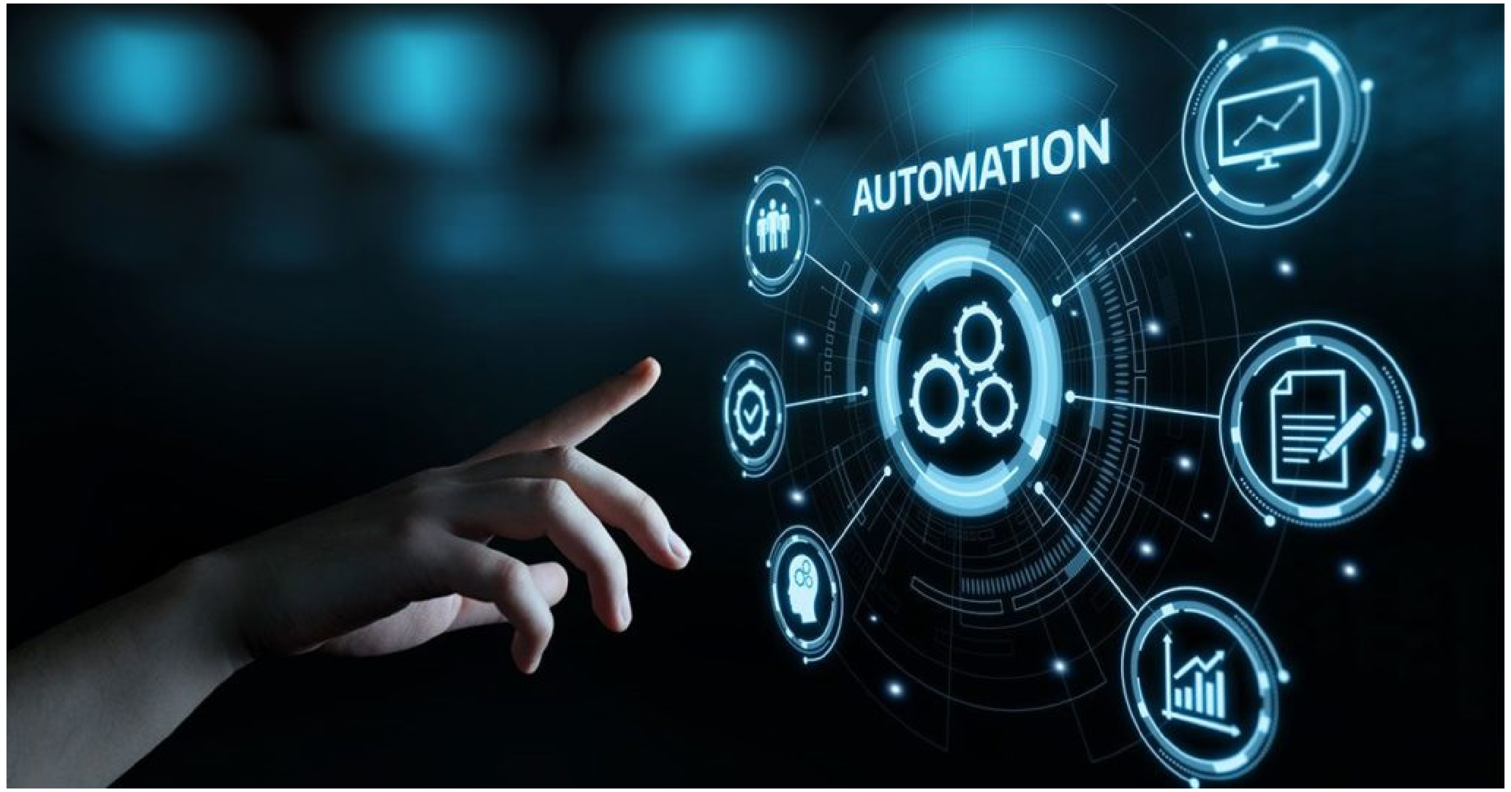
Production Planning

Production Planning

Let's consider this (simplified) decision problem



- We need to select from a set production jobs
- Performing a job provides a value
- We have a minimum value that we need reach
- Every job has an associated cost

Problem Formulation

We provide a mathematical formulation for this problem

$$z^* = \operatorname{argmin}_z \sum_{j=1}^{n} y_j z_j$$

$$\text{subject to:} \sum_{j=1}^{n} v_j z_j \geq v_{min}$$

$$z_j \in \{0, 1\} \qquad \forall j = 1..n$$

Where:

- $z = 1$ iff we pick the j-th job, and 0 otherwise
- v_j is the value for the j-th job
- v_{min} is the minimum required value
- y_j is the cost for the j-th job

Solution Approaches

Optimization problems can be address via several approaches

Some come from th AI fields, other from Mathematics

- Mathematical Programming (e.g. MILP)
- Constraint Programming
- SAT Modulo Theories
- ...

Problems in this class can be very challenging

...But mature tools to address them exist, e.g.:

- The Gurobi solver
- Google or-tools
- The Z3 theorem prover by Microsoft
- ...

Uncertainty

In some real world scenarios the costs would be unknown

...Typically, beucase they are subject to uncertainty

- In other words, we know how much we gain
- ...But we don't know how much we pay

We will assume that at least some information is available

- In particular that y is not observable
- ...But also correlated with some quantity x that we can observe

E.g. some kind of economic indicator

We will also assume that we have historical data

- In particular, by keeping track of what happened in the past
- We collected some examples (x_i, y_i)

Adjusted Formulation and Solution Strategy

How does this change our formulation?

Under uncertainty, we'll want to maximize the expected profit:

$$z^* = \text{argmin}_z \sum_{j=1}^{n} \mathbb{E}_{y \sim P(Y)}[y_j]\, z_j$$

$$\text{subject to: } \sum_{j=1}^{n} v_j z_j \geq v_{min}$$

$$z_j \in \{0, 1\} \qquad \forall j = 1..n$$

- The notation $\mathbb{E}$ represents an expectation
- I.e. a mean over all possible outcomes
- Basically, we want to do well on average

More risk-averse formulations are possible, but we'll not cover them

Predict, the Optimize

A possible solution strategy consists in:

- Based on x, obtaining an estimate for the expected costs
- ...Then we can solve the decision problem as before

This strategy is known as "predict, then optimize"

- We can use Machine Learning to train an estimator
- With the estimator, we can predict the expected costs
- ...Then we can use optimization to find the best decisions

We can view the estimation as a regression problem

...And this is a sound strategy

- In fact, under the right conditions
- A regressor actually tries to estimate a mean

Data Generation

We will use synthetic data for this use case

First, we generate the know parameters for the knapsack problem

```
In [2]: nitems, rel_req, seed = 20, 0.5, 42
        prb = util.generate_problem(nitems=nitems, rel_req=rel_req, seed=seed)
        display(prb)
```

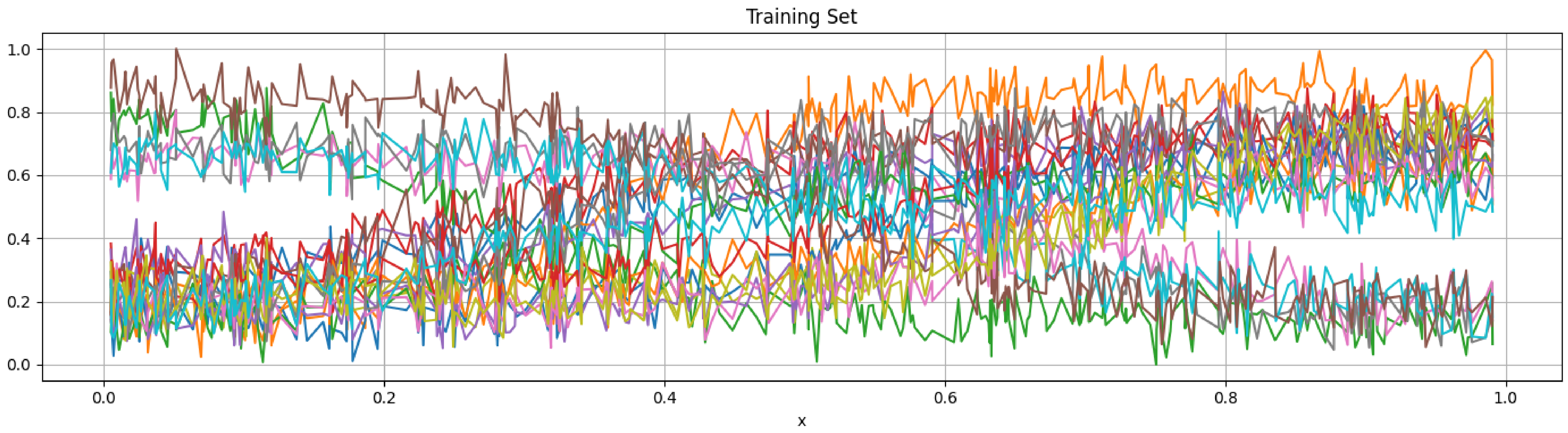
```
ProductionProblem(values=[1.14981605 1.38028572 1.29279758 1.23946339 1.06240746 1.06239781
 1.02323344 1.34647046 1.240446   1.28322903 1.0082338  1.38796394
 1.33297706 1.08493564 1.07272999 1.0733618  1.1216969  1.20990257
 1.17277801 1.11649166], requirement=11.830809153591138)
```

- The jobs
- The values
- The minimum requirement on the total value

Data Generation

The we generate a collection of historical data

```
In [3]: data_tr = util.generate_costs(nsamples=350, nitems=nitems, seed=seed, noise_scale=.2, noise_type
        util.plot_df_cols(data_tr, figsize=figsize, title='Training Set', scatter=False)
```

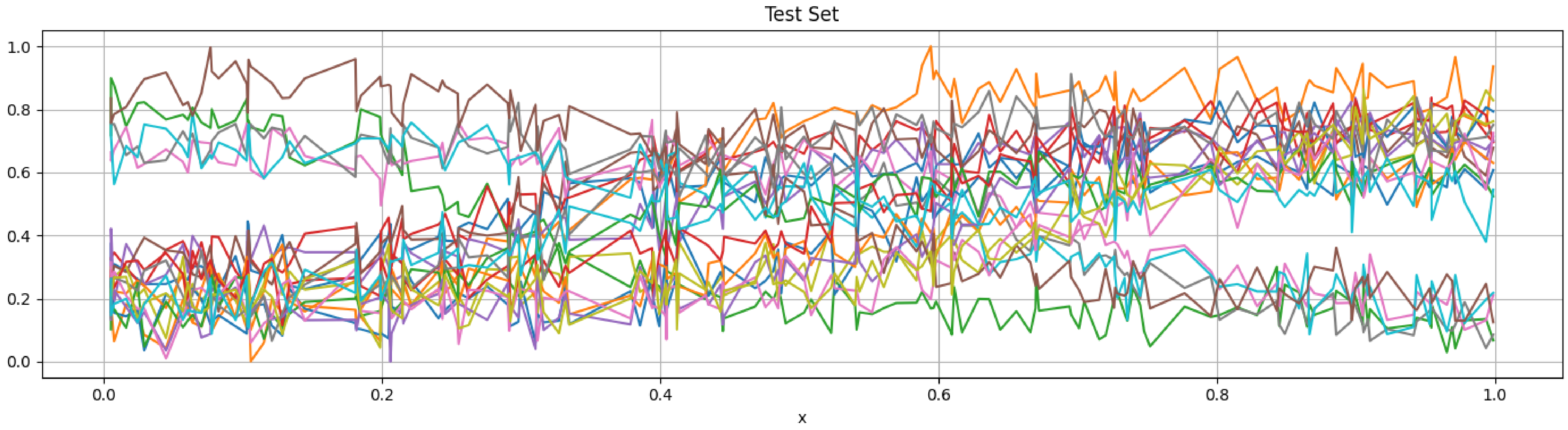


- On the x axis we have the market indicator
- Every color corresponds to the cost for a different investment option

Data Generation

We also generate a second collection of data for testing

In [4]:
```
data_ts = util.generate_costs(nsamples=150, nitems=nitems, seed=seed, sampling_seed=seed+1, nois
util.plot_df_cols(data_ts, figsize=figsize, title='Test Set', scatter=False)
```



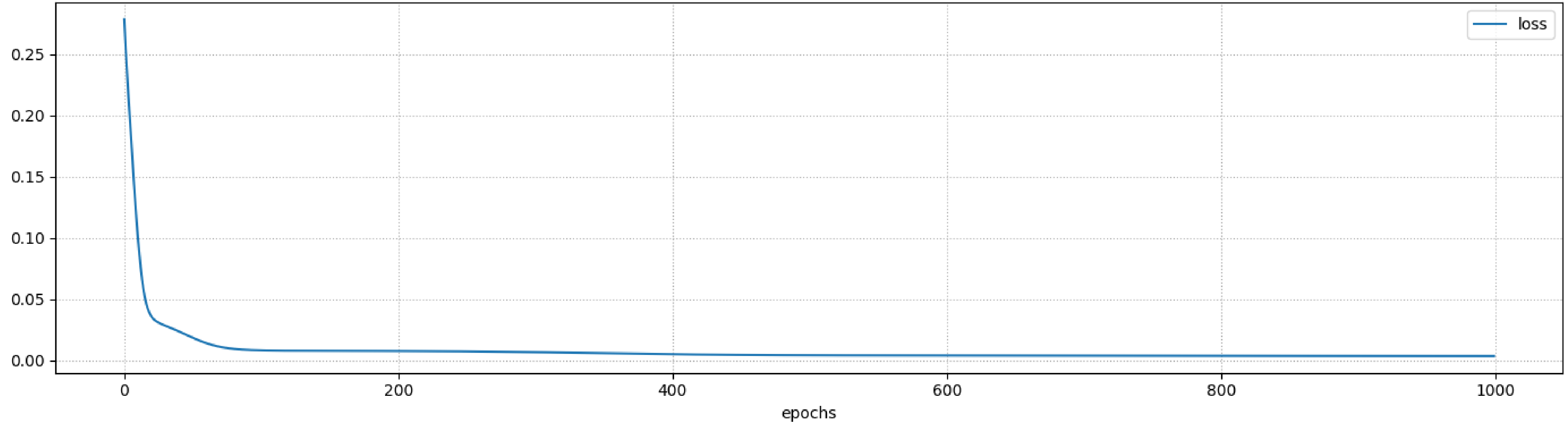
- We will not use this data for training
- So we can use to assess the performance on unseen examples

Training the Estimator

We can not train our estimator

In [5]:
```
pfl_1s = util.build_nn_model(input_shape=1, output_shape=nitems, hidden=[16], name='pfl_1s', out
%time history = util.train_nn_model(pfl_1s, data_tr.index.values, data_tr.values, epochs=1000, 
util.plot_training_history(history, figsize=figsize, print_final_scores=False)
util.print_ml_metrics(pfl_1s, data_tr.index.values, data_tr.values, label='training')
util.print_ml_metrics(pfl_1s, data_ts.index.values, data_ts.values, label='test')
```

```
CPU times: user 9.19 s, sys: 339 ms, total: 9.53 s
Wall time: 7.13 s
```



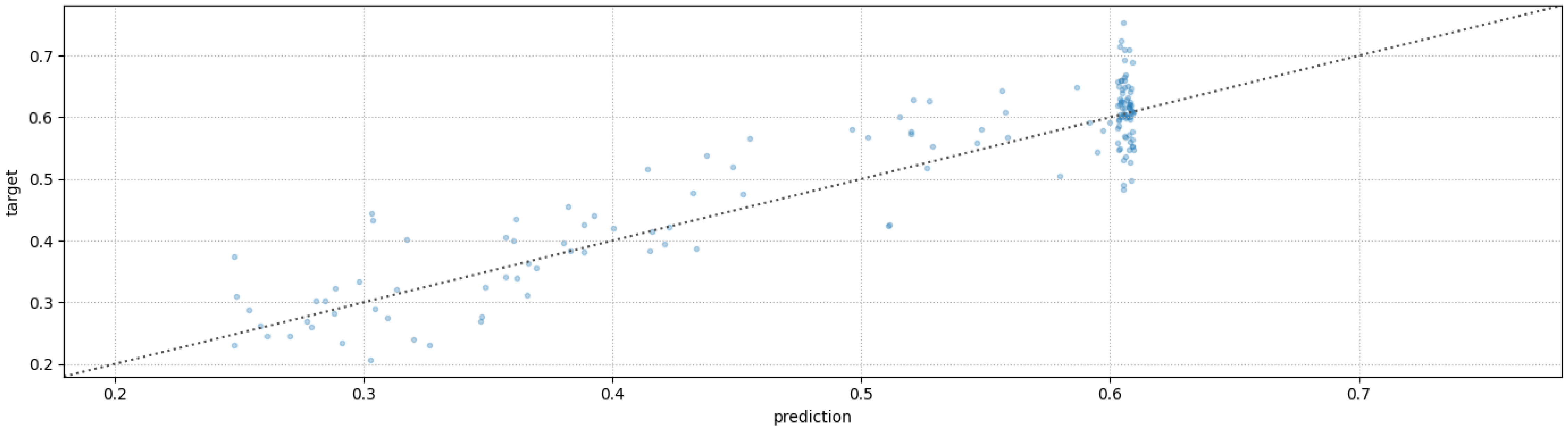

```
R2: 0.90, MAE: 0.049, RMSE: 0.06 (training)
R2: 0.90, MAE: 0.05, RMSE: 0.06 (test)
```

Estimation Accuracy

Our estimator is quite accurate, despite the presence of uncertainty

In [6]:
```
pfl_1s_pred_ts = pfl_1s.predict(data_ts.index.values, verbose=0)
util.plot_pred_scatter(data_ts.values[:, 0], pfl_1s_pred_ts[:, 0], figsize=figsize, alpha=0.3)
```

```
R2: 0.83
MAE: 0.04
```



- Here we see the prediction/ground truth correlation for the 1st item

Evaluating the Decisions

Our final evaluation, however, should be in terms of cost

We can do it like this:

- Let $z^*(\hat{y})$ be the solution we compute based on the cost estimates $\hat{y}$
- Let $z^*(y)$ be the solution we compute based on the true costs y
- Then, we can measure how well we are doing by computing:

$$\frac{\text{cost}(z^*(\hat{y})) - \text{cost}(z^*(y))}{\text{cost}(z^*(y))}$$

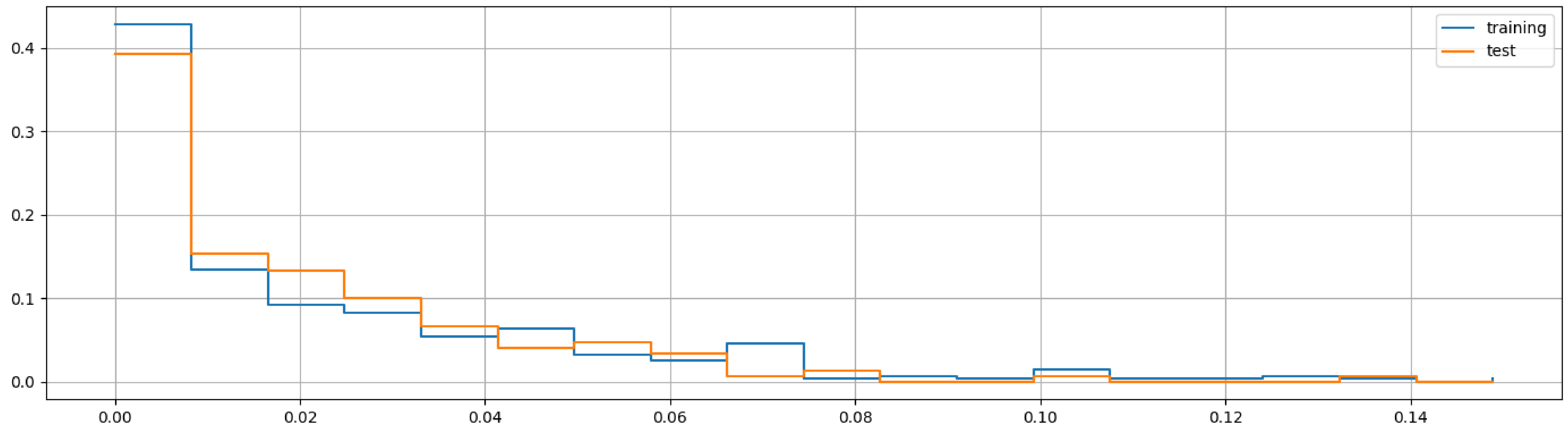
This is called relative (post-hoc) regret

- It's the different between the cost that we actually payed
- ...And the cost we could have paid with the benefit of hindsight

Post-hoc Regret Evaluation

Let's check how well we're doing in terms of post-hoc regret

```
In [7]: r_tr_1s = util.compute_regret(prb, pfl_1s, data_tr.index.values, data_tr.values)
        r_ts_1s = util.compute_regret(prb, pfl_1s, data_ts.index.values, data_ts.values)
        util.plot_histogram(r_tr_1s, figsize=figsize, label='training', data2=r_ts_1s, label2='test', pr
```




```
Mean: 0.023 (training), 0.020 (test)
```

The average is roughly ~2%

Limits of this Approach

The "predict, then optimize" approach work well on one condition

Our ML model should effective at estimating the expected costs

- However, this is not always possible
- Sometimes you lack clear statistical information on the data
- ...Or sometimes you are forced to use a simpler model
- ...For example to provide explainability

We will run an experiment to test this condition

We will restrict ourselves to using a Linear Regressor

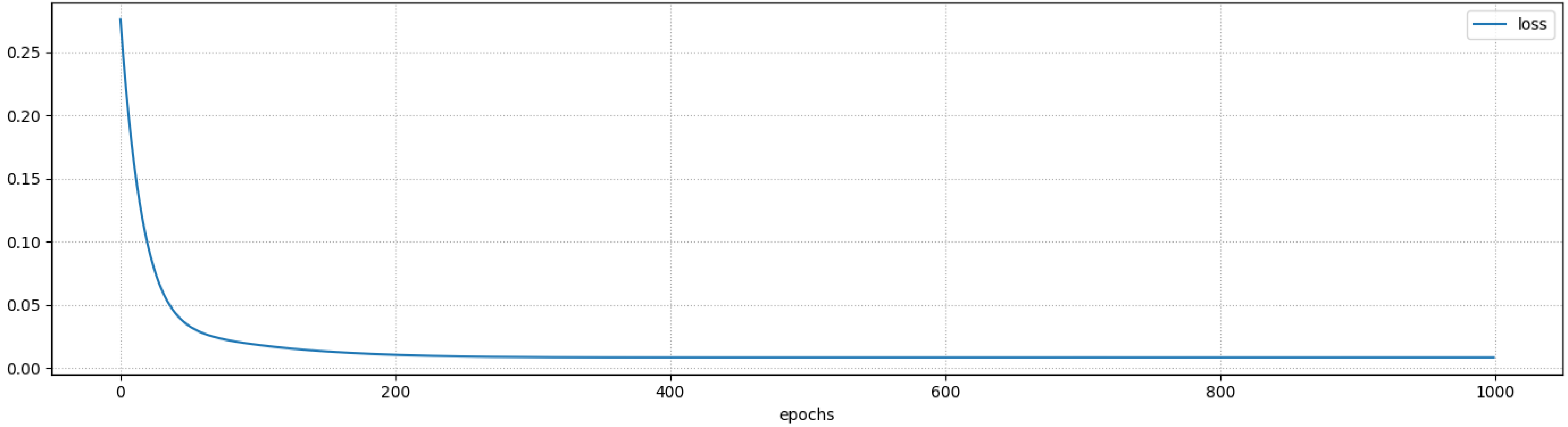
- As an ML model, this is much easier to interpret
- ...But also considerably less accurate in many cases

Training a Linear Regressor

Let's try our new, simpler, model

```
In [8]: pfl_1s_l = util.build_nn_model(input_shape=1, output_shape=nitems, hidden=[], name='pfl_1s_l', c
        %time history = util.train_nn_model(pfl_1s_l, data_tr.index.values, data_tr.values, epochs=1000,
        util.plot_training_history(history, figsize=figsize, print_final_scores=False)
        util.print_ml_metrics(pfl_1s_l, data_tr.index.values, data_tr.values, label='training')
        util.print_ml_metrics(pfl_1s_l, data_ts.index.values, data_ts.values, label='test')
```

```
CPU times: user 8.08 s, sys: 278 ms, total: 8.35 s
Wall time: 6.71 s
```

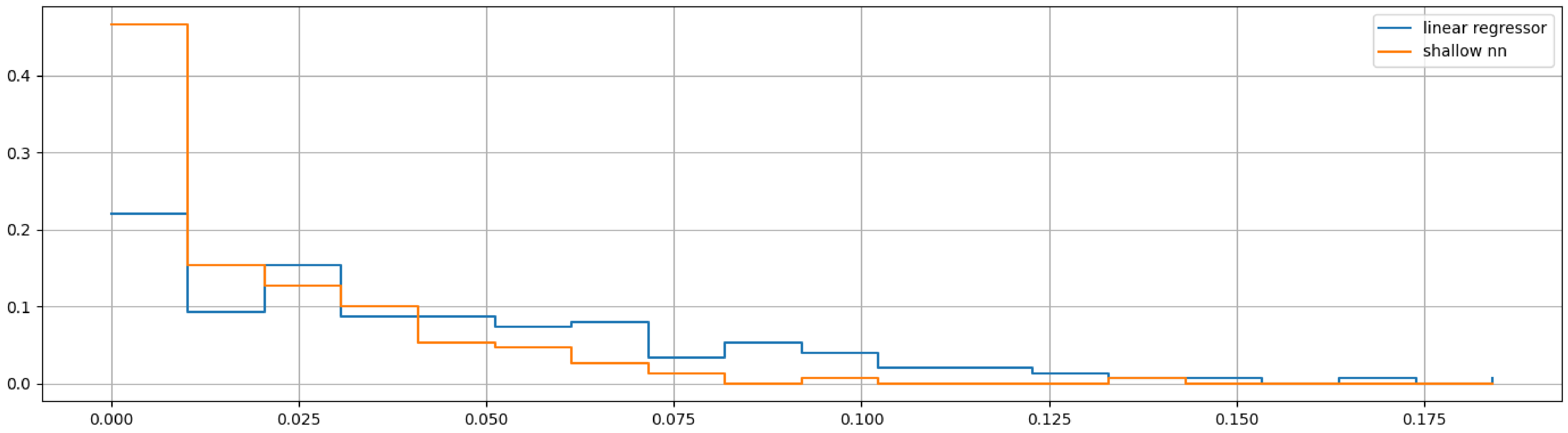



```
R2: 0.79, MAE: 0.074, RMSE: 0.09 (training)
R2: 0.78, MAE: 0.076, RMSE: 0.09 (test)
```

Regret Evaluation

Let's check how the model fares in terms of (post-hoc) regret

```
In [9]: r_tr_1s_l = util.compute_regret(prb, pfl_1s_l, data_tr.index.values, data_tr.values)
        r_ts_1s_l = util.compute_regret(prb, pfl_1s_l, data_ts.index.values, data_ts.values)
        util.plot_histogram(r_ts_1s_l, figsize=figsize, label='linear regressor', data2=r_ts_1s, label2=
```




```
Mean: 0.044 (linear regressor), 0.020 (shallow nn)
```

- The simpler model leads to twice the regret

A Hybrid Method

In this kind of situation, using a hybrid method can close the gap

A hybrid method is one that meld elements to different fields

- We are already using learning and optimization in sequence
- ...But hybrid methods do it at the same time

The method we will use is called Decision Focused Learning

...And consists in minimizing regret at training time:

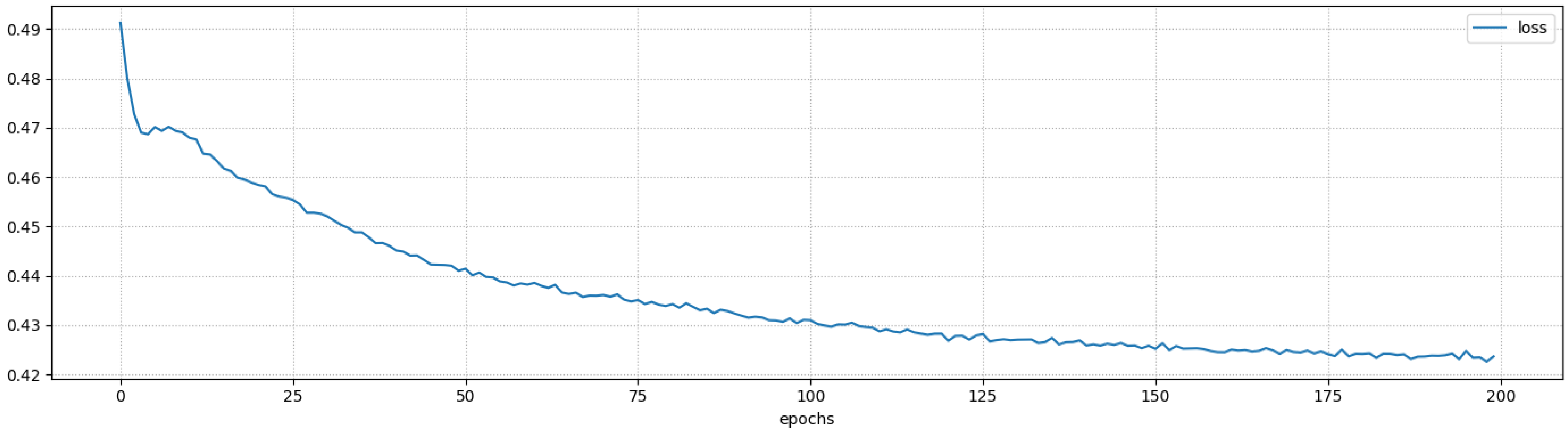
- Given the current network parameters θ
- We solve one optimization problem for every training example
- We identify an improvement direction for the parameters
- ...And we adjust the parameters in that direction

Training a DFL Approach

Let's train a DFL approach

```
In [10]: spo_1s = util.build_dfl_ml_model(input_size=1, output_size=nitems, problem=prb, hidden=[], name=
         %time history = util.train_dfl_model(spo_1s, data_tr.index.values, data_tr.values, epochs=200, v
         util.plot_training_history(history, figsize=figsize, print_final_scores=False)
         util.print_ml_metrics(spo_1s, data_tr.index.values, data_tr.values, label='training')
         util.print_ml_metrics(spo_1s, data_ts.index.values, data_ts.values, label='test')
```

```
CPU times: user 2min 48s, sys: 11min 9s, total: 13min 57s
Wall time: 1min 23s
```



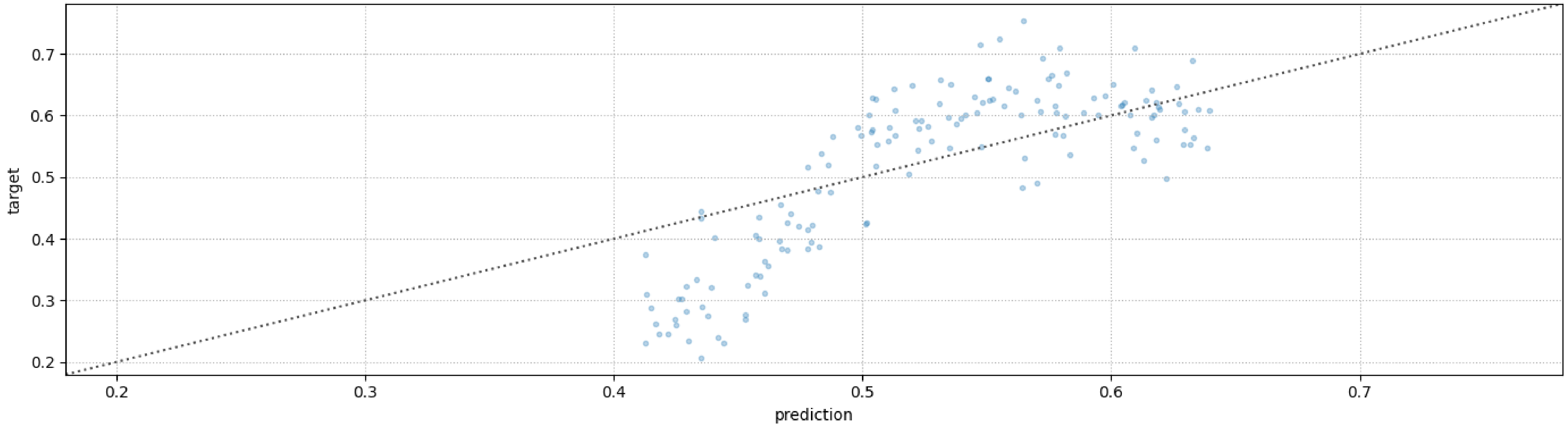

```
R2: 0.66, MAE: 0.089, RMSE: 0.12 (training)
R2: 0.67, MAE: 0.089, RMSE: 0.12 (test)
```

Estimation Accuracy

In terms of prediction quality, DFL approaches are considerably worse

```
In [12]: spo_1s_pred_ts = spo_1s.predict(data_ts.index.values, verbose=0)
         util.plot_pred_scatter(data_ts.values[:, 0], spo_1s_pred_ts[:, 0], figsize=figsize, alpha=0.3)
```

```
R2: 0.54
MAE: 0.08
```

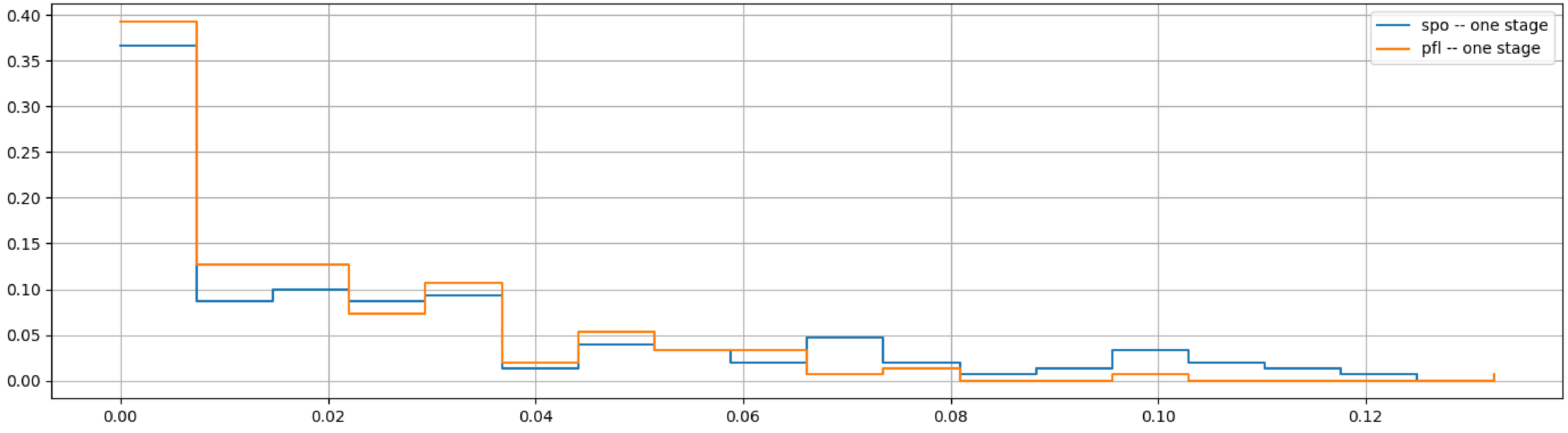


...Since their goal is not accuracy, but cost minimization

Regret Evaluation

We get a main advantage in terms of regret

In [11]:
```
r_ts_spo_1s = util.compute_regret(prb, spo_1s, data_ts.index.values, data_ts.values)
util.plot_histogram(r_ts_spo_1s, figsize=figsize, label='spo -- one stage', data2=r_ts_1s, label
```




```
Mean: 0.028 (spo -- one stage), 0.020 (pfl -- one stage)
```

Despite the model simplicity, we are almost matching our original results