Applicability of DFL

'Cause the Devil is in the Details

Let's Second-Guess Ourselvers

However, let's not discount the prediction-focused approach yet

In fact, it's easy to see that:

$$\mathbb{E}[\text{regret}(y, \hat{y})] \xrightarrow[\mathbb{E}[L(y,\hat{y})] \to 0]{} 0$$

Intuitively:

- The more accurate we can be, the lower the regret
- Eventually, perfect predictions will result in 0 regret

Let's Second-Guess Ourselvers

However, let's not discount the prediction-focused approach yet

In fact, it's easy to see that:

$$\mathbb{E}[\text{regret}(y, \hat{y})] \xrightarrow[\mathbb{E}[L(y,\hat{y})]\to 0]{} 0$$

Intuitively:

- The more accurate we can be, the lower the regret
- Eventually, perfect predictions will result in 0 regret

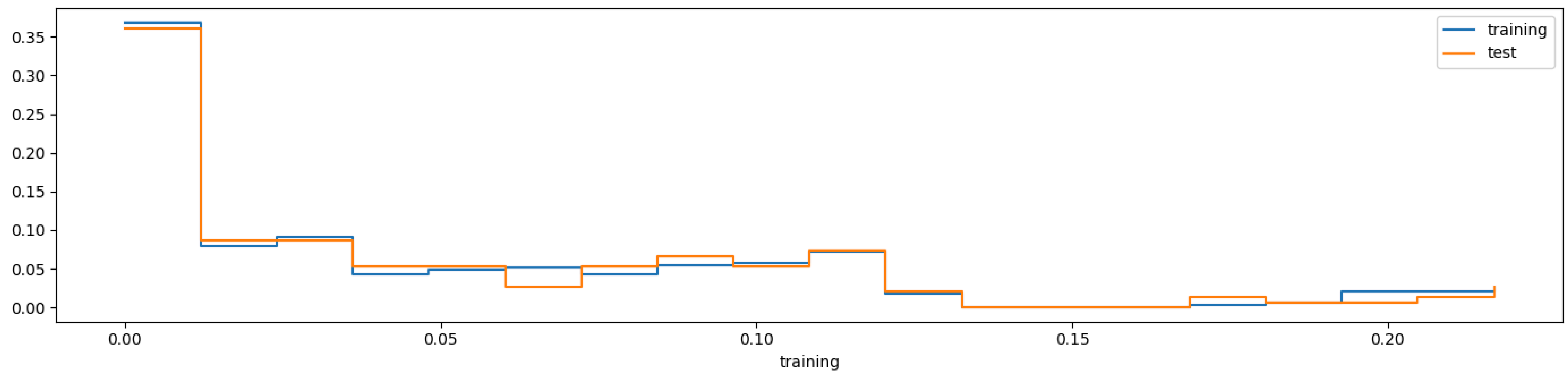
But then... What if we make our model bigger?

- We could get good predictions and good regret
- ...And training would be much faster

Our Baseline

Let's check again the results for our PFL linear regressor

In [4]:
```
pfl = util.build_ml_model(input_size=1, output_size=nitems, hidden=[], name='pfl_det', outpu
history = util.train_ml_model(pfl, data_tr.index.values, data_tr.values, epochs=1000, loss=
r_tr = util.compute_regret(prb, pfl, data_tr.index.values, data_tr.values)
r_ts = util.compute_regret(prb, pfl, data_ts.index.values, data_ts.values)
util.plot_histogram(r_tr, figsize=figsize, label='training', data2=r_ts, label2='test', pri
```



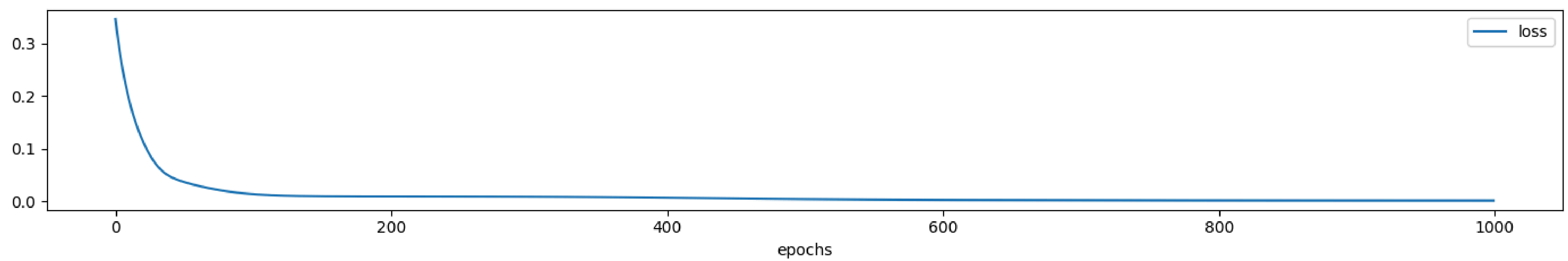

```
Mean: 0.052 (training), 0.052 (test)
```

This will be our main baseline

PFL Strikes Back

Let's try to use a non-linear model

In [5]:
```
pfl_acc = util.build_ml_model(input_size=1, output_size=nitems, hidden=[8], name='pfl_det_ac
history = util.train_ml_model(pfl_acc, data_tr.index.values, data_tr.values, epochs=1000, lo
util.plot_training_history(history, figsize=figsize_narrow, print_scores=False, print_time=T
util.print_ml_metrics(pfl_acc, data_tr.index.values, data_tr.values, label='training')
util.print_ml_metrics(pfl_acc, data_ts.index.values, data_ts.values, label='test')
```



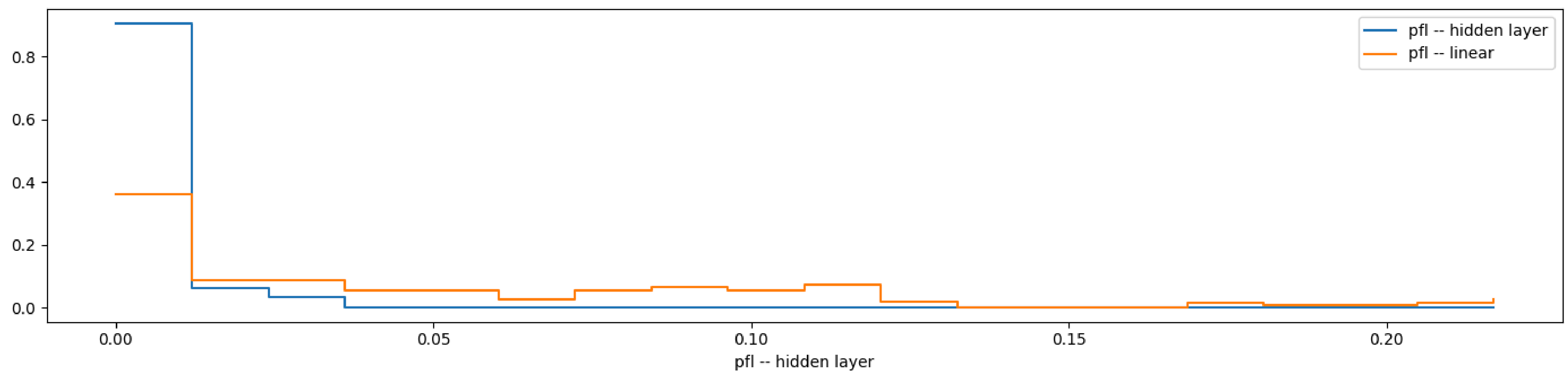

```
Training time: 7.3678 sec
R2: 0.98, MAE: 0.029, RMSE: 0.04 (training)
R2: 0.98, MAE: 0.029, RMSE: 0.04 (test)
```

More accurate, it is!

PFL Strikes Back

...And the improvement in terms of regret is remarkable

```
In [6]: r_ts_acc = util.compute_regret(prb, pfl_acc, data_ts.index.values, data_ts.values)
        util.plot_histogram(r_ts_acc, figsize=figsize, label='pfl -- hidden layer', data2=r_ts, labe
```




```
Mean: 0.003 (pfl -- hidden layer), 0.052 (pfl -- linear)
```

- DFL might do better with the same model complexity
- ...But the return would be diminished

Evening the Field

Can't we do anything about it?

- DFL predictions will always be off (more or less)
- ...But there are ways to make the approach faster

For example:

- You can use a problem relaxation, as in [1]
- You can limit recomputation by caching past solutions, as in [2]
- You can warm start the DFL approach with the PFL weights

Let's see the last two tricks in deeper detail

[1] Mandi, Jayanta, and Tias Guns. "Interior point solving for lp-based prediction+ optimisation." Advances in Neural Information Processing Systems 33 (2020): 7272-7282.
[2] Maxime Mulamba, Jayanta Mandi, Michelangelo Diligenti, Michele Lombardi, Victor Bucarey, Tias Guns: Contrastive Losses and Solution Caching for Predict-and-Optimize. IJCAI 2021: 2833-2840

Warm Starting and Solution Caching

Warm starting simple consists in using the PFL weights to initialize θ

Since accuracy is correlated with regret, this might accelerate convergence

Warm Starting and Solution Caching

Warm starting simple consists in using the PFL weights to initialize θ

Since accuracy is correlated with regret, this might accelerate convergence

Solution caching is applicable if the feasible space is fixed

I.e. to problems in the form:

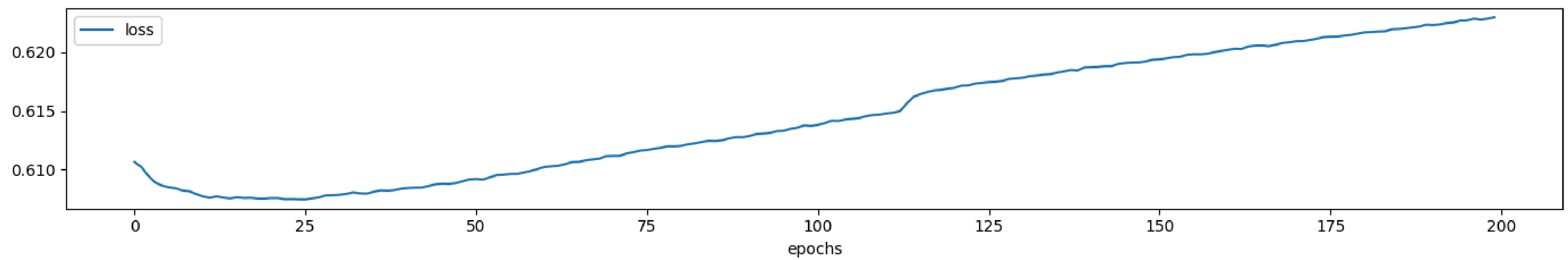
$$z^*(y) = \text{argmin}_z \{ f(z) \mid z \in F \}$$

- During training, we maintain a solution cache S
- Initially, we populate S with the true optimal solutions $z^*(y_i)$ for all examples
- Before computing $z^*(\hat{y})$ for the current prediction we flip a coin
- With probability p, we run the computation (and store any new solution in S)
- With probability $1 - p$, we solve instead $\hat{z}^*(\hat{y}) = \text{argmin}_z \{ f(z) \mid z \in S \}$

Speeding Up DFL

Let's use DFL with linear regression, a warm start, and a solution cache

```
In [7]: spo = util.build_dfl_ml_model(input_size=1, output_size=nitems, problem=prb, hidden=[], name
        history = util.train_dfl_model(spo, data_tr.index.values, data_tr.values, epochs=200, verbos
        util.plot_training_history(history, figsize=figsize_narrow, print_scores=False, print_time=T
        util.print_ml_metrics(spo, data_tr.index.values, data_tr.values, label='training')
        util.print_ml_metrics(spo, data_ts.index.values, data_ts.values, label='test')
```



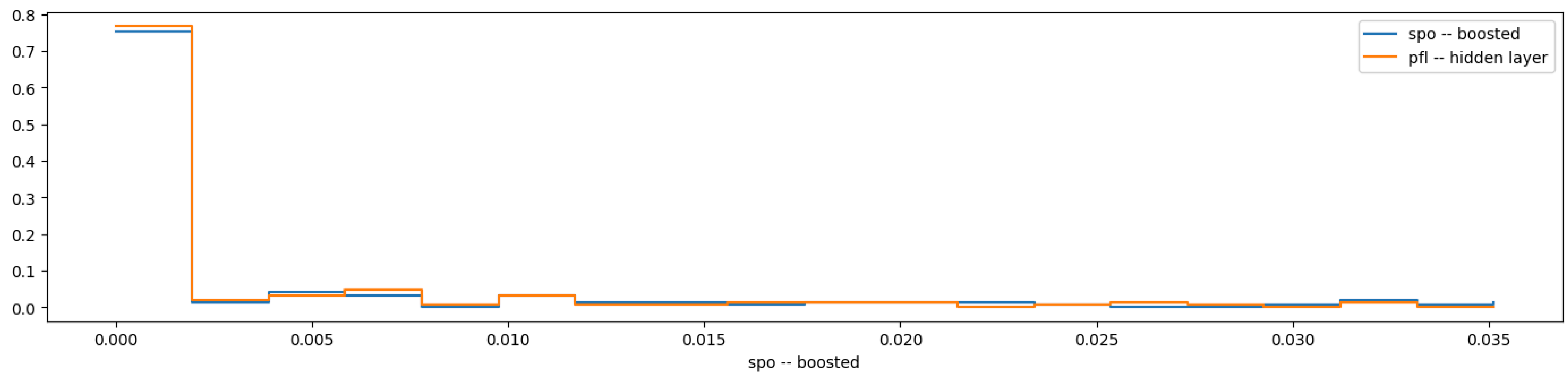

```
Training time: 21.0472 sec
R2: 0.65, MAE: 0.12, RMSE: 0.16 (training)
R2: 0.65, MAE: 0.12, RMSE: 0.16 (test)
```

The training time is still large, but much lower than our earlier DFL attempt

Speeding Up DFL

And the regret is even better!

In [12]:
```
r_ts_spo = util.compute_regret(prb, spo, data_ts.index.values, data_ts.values)
util.plot_histogram(r_ts_spo, figsize=figsize, label='spo -- boosted', data2=r_ts_acc, labe
```




```
Mean: 0.004 (spo -- boosted), 0.003 (pfl -- hidden layer)
```

We are matching the more complex PFL model with a simple linear regressor

Reflecting on What we Have

Therefore, DFL gives us at least two benefits

First, it can lead to lower regret compared to a prediction-focused appraoch

- As the models become more complex we have diminishing returns
- ...But for some applications every little bit counts

Second, it may allow using simpler ML models

- Simple models are faster to evaluate
- ...But more importantly they are easier to explain
- E.g. we can easily perform feature importance analysis

Reflecting on What we Have

Therefore, DFL gives us at least two benefits

First, it can lead to lower regret compared to a prediction-focused appraoch

- As the models become more complex we have diminishing returns
- ...But for some applications every little bit counts

Second, it may allow using simpler ML models

- Simple models are faster to evaluate
- ...But more importantly they are easier to explain
- E.g. we can easily perform feature importance analysis

Intuitively, DFL works best where PFL has estimation issues

Can we exploit this fact to maximize our advantage?

Maximizing Results

There's a simple case where PFL cannot make perfect predictions

You just need need to target a stochastic problem!

- E.g. you can usually tell the traffic situation based on (e.g.) time and weather
- ...But there still a lot of variability

Adjusting Goals

But with a stochastic problem, what is our real objective?

For a given x (i.e. a given context), we can formalize it like this:

$$\text{argmin}_z \left\{ \mathbb{E}_{y \sim P(Y|x)} \left[y^T z \right] \mid z \in F \right\}$$

- Given a value for the observable x
- We want to find a single decision vector z
- Such that z is feasible
- ...And z minimizes the expected cost over the distribution $P(Y \mid x)$

This is called a one-stage stochastic optimization problem

...And Keeping the Setup

Let's look again at the DFL training problem

$$\theta^* = \operatorname{argmin}_\theta \{\mathbb{E}_{(x,y)\sim P(X,Y)} [\text{regret}(y, \hat{y})] \mid \hat{y} = h(x; \theta)\}$$

With:

$$\text{regret}(y, \hat{y}) = y^T z^*(\hat{y}) - y^T z^*(y)$$

Since $y^T z^*(y)$ is independent on θ, this is equivalent to:

$$\theta^* = \operatorname{argmin}_\theta \{\mathbb{E}_{(x,y)\sim P(X,Y)} [y^T z^*(\hat{y})] \mid \hat{y} = h(x; \theta)\}$$

Which can be rewritten as:

$$\theta^* = \operatorname{argmin}_\theta \{\mathbb{E}_{x\sim P(X), y\sim P(Y|x)} [y^T z^*(\hat{y})] \mid \hat{y} = h(x; \theta)\}$$

...And Keeping the Setup

Now, let's restrict to the case where x is fixed

$$\theta^* = \text{argmin}_\theta \{\mathbb{E}_{y \sim P(Y|x)}[y^T z^*(\hat{y})] \mid \hat{y} = h(x;\theta)\}$$

Finally, by definition of $z^*(\cdot)$ we have:

$$\theta^* = \text{argmin}_\theta \{\mathbb{E}_{y \sim P(Y|x)}[y^T z^*(\hat{y})] \mid \hat{y} = h(x;\theta), z^*(\hat{y}) \in F\}$$

In other words:

- We are choosing θ
- So that $z^*(\hat{y})$ minimizes $\mathbb{E}_{y \sim P(Y|x)}[y^T z^*(\hat{y})]$

This is almost identical to one-stage stochastic optimization!

DFL For One-Stage Stochastic Optimization

DFL can address 1s-SOPs, with one restriction and two "superpowers":

The restriction is that we control z only through θ

- Therefore, depending on the chosen ML model architecture
- ...Obtaining some solutions might be impossible
- This issue can be sidestepped with a careful model choice

DFL For One-Stage Stochastic Optimization

DFL can address 1s-SOPs, with one restriction and two "superpowers":

The restriction is that we control z only through θ

- Therefore, depending on the chosen ML model architecture
- ...Obtaining some solutions might be impossible
- This issue can be sidestepped with a careful model choice

The first superpower is that we are not restricted to a single x value

- Given a new value for x, we just need to evaluate $h(x, \theta^*)$
- ...And then solve the usual optimization problem
- Many approaches do not deal with the estimation of the y distribution

DFL For One-Stage Stochastic Optimization

DFL can address 1s-SOPs, with one restriction and two "superpowers":

The restriction is that we control z only through θ

- Therefore, depending on the chosen ML model architecture
- ...Obtaining some solutions might be impossible
- This issue can be sidestepped with a careful model choice

The first superpower is that we are not restricted to a single x value

- Given a new value for x, we just need to evaluate $h(x, \theta^*)$
- ...And then solve the usual optimization problem
- Many approaches do not deal with the estimation of the y distribution

For the second superpower, we need to investigate a bit more

Classical Solution Approach

What would be the classical solution approach?

Starting from:

$$\operatorname{argmin}_z \left\{ \mathbb{E}_{y \sim P(Y|x)} [y^T z] \mid z \in F \right\}$$

We can use linearity to obtain:

$$\operatorname{argmin}_z \left\{ \mathbb{E}_{y \sim P(Y|x)} [y]^T z \mid z \in F \right\}$$

- So, we would first need to estimate the expected costs
- ...Then we could solve a deterministic problem

Classical Solution Approach

What would be the classical solution approach?

Starting from:

$$\operatorname{argmin}_z \left\{ \mathbb{E}_{y \sim P(Y|x)}[y^T z] \mid z \in F \right\}$$

We can use linearity to obtain:

$$\operatorname{argmin}_z \left\{ \mathbb{E}_{y \sim P(Y|x)}[y]^T z \mid z \in F \right\}$$

- So, we would first need to estimate the expected costs
- ...Then we could solve a deterministic problem

But isn't this what PFL is doing?

Regression and Expectation

(Stochastic) Regression is often presented as learning an expectation

...But it's trickier than that

- Using an MSE loss is equivalent to trying to learn $\mathbb{E}_{y \sim P(Y|x)}[y]$
- ...But only assuming that $P(Y \mid x)$ is Normally distributed
- ...And that it has the same variance everywhere

It is possible to do the same under more general conditions

...But it is much more complex

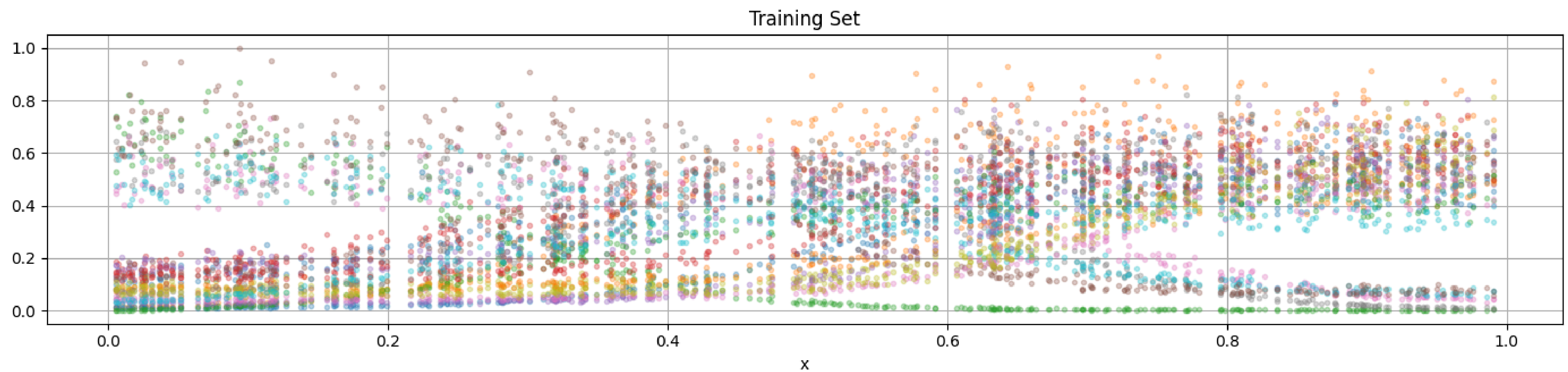
- If we know the distribution type, we can use a neuro-probabilistic model
- Otherwise, we need a fully fledged contextual generative model

In DFL, we can address this problem with 0 added effort!

A Simple Stress Test

We can test this idea by generating a stochastic dataset

In [15]:
```
data_tr = util.generate_costs(nsamples=350, nitems=nitems, seed=seed, noise_scale=.2, noise_
data_ts = util.generate_costs(nsamples=150, nitems=nitems, seed=seed, sampling_seed=seed+1,
util.plot_df_cols(data_tr, figsize=figsize, title='Training Set', scatter=True)
```

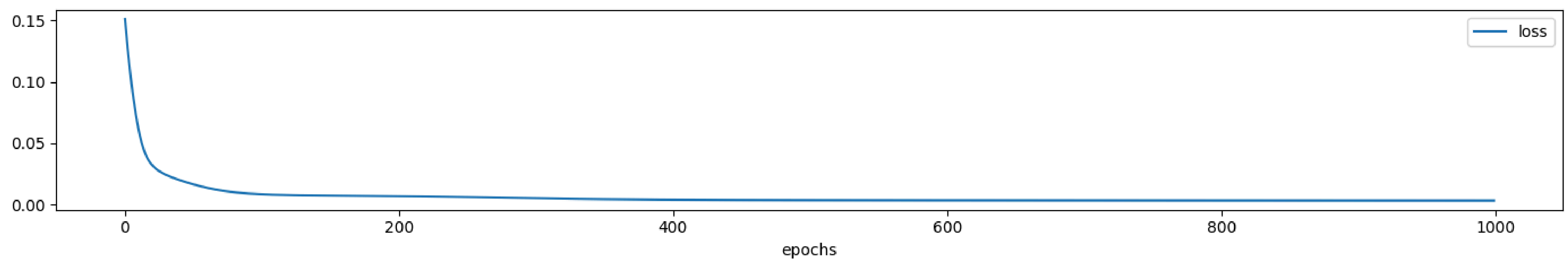


- ...And scaling the variance with y
- Which is also a very common setting in practice

Training a PFL Approach

We will train again a non-linear prediction focused approach

In [16]:
```
pfl_1s = util.build_ml_model(input_size=1, output_size=nitems, hidden=[8], name='pfl_1s', ou
history = util.train_ml_model(pfl_1s, data_tr.index.values, data_tr.values, epochs=1000, los
util.plot_training_history(history, figsize=figsize_narrow, print_scores=False, print_time=T
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')
```



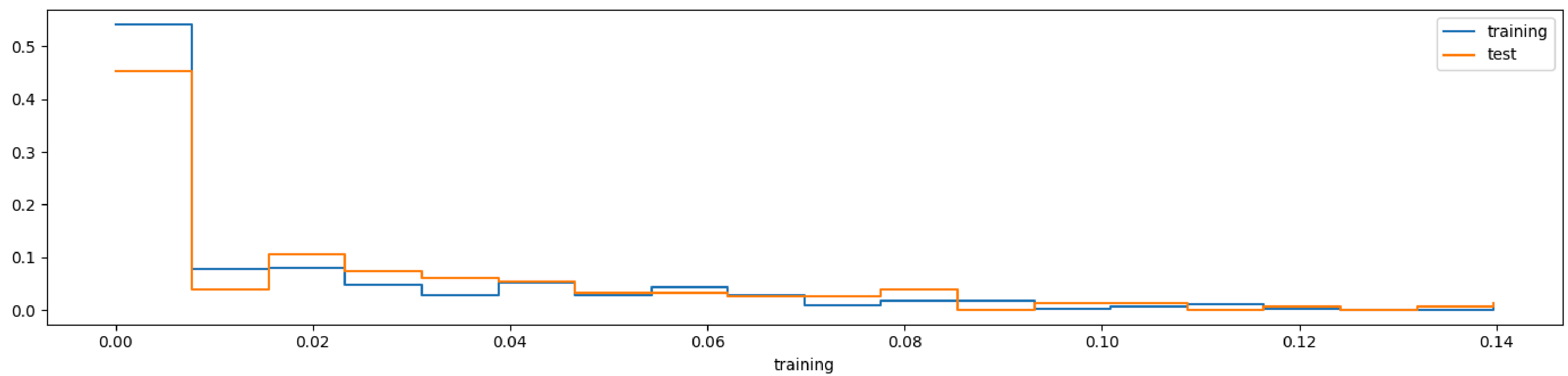

```
Training time: 7.3259 sec
R2: 0.91, MAE: 0.042, RMSE: 0.06 (training)
R2: 0.91, MAE: 0.042, RMSE: 0.06 (test)
```

The accuracy is (inevitably) worse, but still pretty good

PFL Regret

Let's evaluate the regret of the PFL approach

```
In [17]: 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
```




```
Mean: 0.020 (training), 0.026 (test)
```

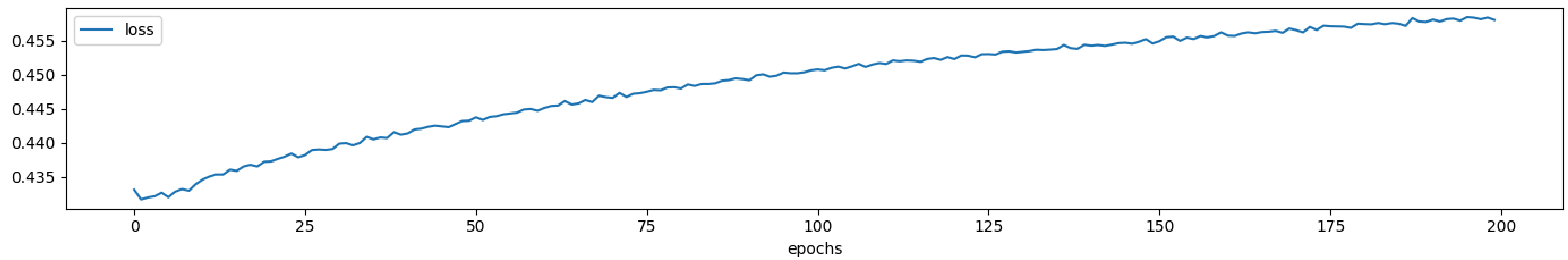
The regret has slighly worsened, due to the effect of uncertainty

Training a DFL Approach

We also a DFL approach with the same non-linear model

In [18]:

```
spo_1s = util.build_dfl_ml_model(input_size=1, output_size=nitems, problem=prb, hidden=[8],
history = util.train_dfl_model(spo_1s, data_tr.index.values, data_tr.values, epochs=200, ve
util.plot_training_history(history, figsize=figsize_narrow, print_scores=False, print_time=
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')
```

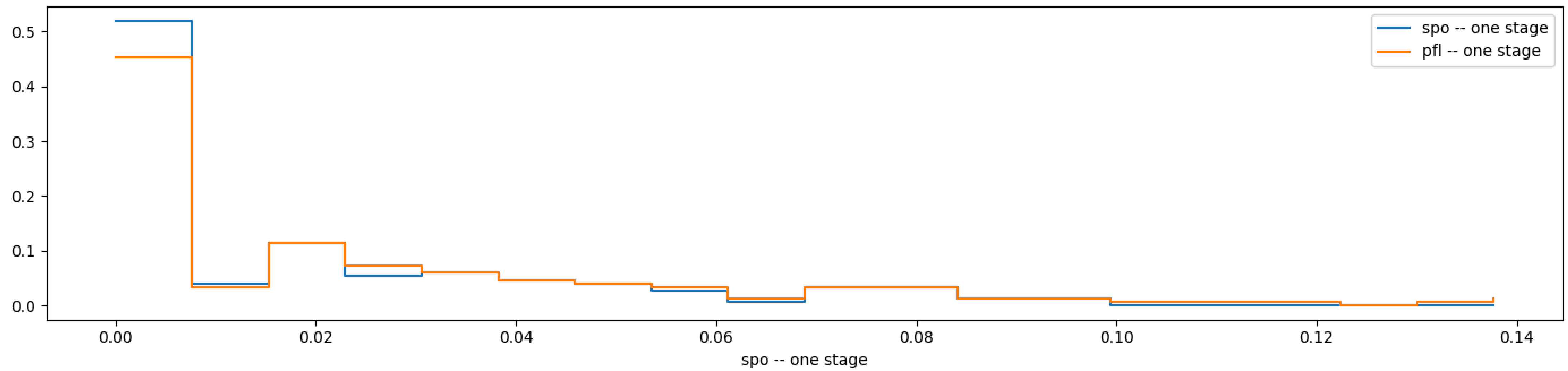



```
Training time: 20.4491 sec
R2: 0.82, MAE: 0.063, RMSE: 0.09 (training)
R2: 0.81, MAE: 0.063, RMSE: 0.09 (test)
```

DFL Regret

Now we can compare the regret for both approaches

In [19]:
```
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,
```




```
Mean: 0.020 (spo -- one stage), 0.026 (pfl -- one stage)
```

- There is a significant gap again
- Since the PFL approach is operating on an incorrect semantic

Considerations

DFL can be thought of as a one-stage stochastic optimization approach

In this setting:

- In particular, using a more accurate PFL model might still have poor regret
- ...Unless we know a lot about the distribution
- ...or we use a very complex estimator
- Conversely, DFL has not such issues

The gap becomes wider in case of non-linear cost functions:

- In this case the expected cost would not be equivalent to a sum of expectations
- But a DFL approach would have no such issues
- ...Provided it could deal with with non-linear functions