...And Optimize

'Cause we are not just dealing with ML, ain't we?

Our Current Situation

The results so far are not comforting

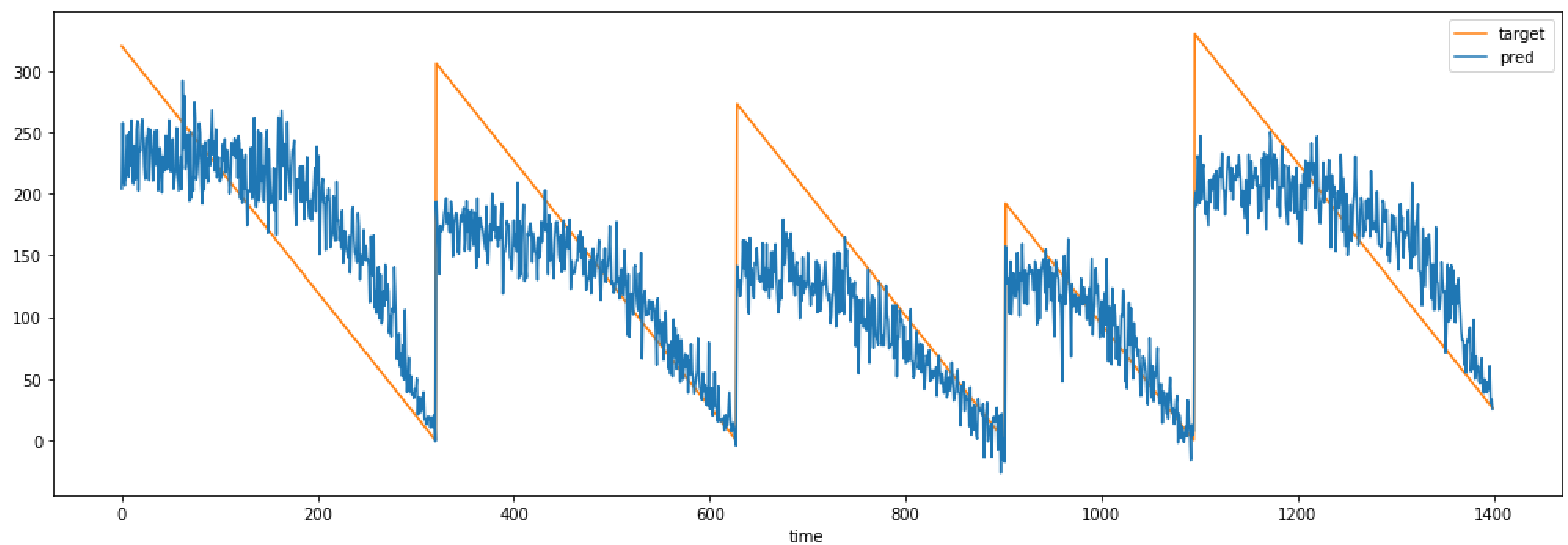
...But it's worth seeing what is going on over time

Our Current Situation

The results so far are not comforting

...But it's worth seeing what is going on over time

In [2]:
```
stop = 1400
util.plot_rul(tr_pred[:stop], tr['rul'][:stop], figsize=figsize)
```



...And we get the same shapes also on the validation and test set

...And Two Observations about It

- All sequeces of predicted RULs start with a "flat" section
- ...And after a while they bend and start decreasing

First: why is this happening? And why is is so consistent?

...And Two Observations about It

- All sequeces of predicted RULs start with a "flat" section
- ...And after a while they bend and start decreasing

First: why is this happening? And why is is so consistent?

One reason is that large RUL values are under-represented

...Since not all machines run for the same time

- As a result we will have larger noise
- ...And it may be impossible to predict RULs larger than in the training set

However, the curve bend relatively late

- ...And therefore there must be something more

...And Two Observations about It

- All sequeces of predicted RULs start with a "flat" section
- ...And after a while they bend and start decreasing

First: why is this happening? And why is is so consistent?

...And Two Observations about It

- All sequeces of predicted RULs start with a "flat" section
- ...And after a while they bend and start decreasing

First: why is this happening? And why is is so consistent?

The main reason is that degradation does not start immediately

...But typically only when microscopic defects grow to become perceivable

- As a result, early on the NN will be "see" examples with comparable input
- ...But different target values

When an MSE loss, the optimal choice in this case is to predict the average

...And Two Observations about It

- All sequeces of predicted RULs start with a "flat" section
- ...And after a while they bend and start decreasing

Second: is this pattern good or bad news for us?

Our goal is not to regress RUL values with high accuracy

...But rather to define a maintenance policy in the form:

$$f(x, \omega) < \theta \quad \Rightarrow \quad \text{trigger maintenance}$$

- For this, we just need to stop at the right time
- ...And our model may be accurate enough in the region that matters

Threshold Calibration as an Optimization Problem

Given a RUL estimator

...We can choose when to trigger maintenance by calibrating θ

- This is in fact an(ohter) optimization problem
- ...And to formulate it we need a cost function

Our cost function will rely on this simplified cost model:

- Whenever a turbine operates for a time step, we gain a profit of 1 unit
- A failure costs C units (i.e. the equivalent of C operation days)
- We never trigger maintenance before s time steps

Some comments:

- C is actually an offset over the cost of maintenance
- The last rule mimics using preventive maintenance as a fail-safe mechanism

The Cost Function

Let x_k be the times series for machine k (out of n_r), and n_t its length

With our RUL based policy:

- Given a cost function $cost(f(x_k), x_k, \theta)$ for one machine, the total cost is:

$$\sum_{k=1}^{n_r} cost(f(x_k), x_k, \theta)$$

- The time step when we trigger maintenance is given by:

$$\min\{i = 1..n_t \mid f(x_{ki}) < \theta\}$$

- A failure occurs if:

$$f(x_{ki}) \geq \theta \quad \forall i = 1..n_t$$

The Cost Function

The cost formula for a single machine will be:

$$cost(f(x_k), x_k, \theta) = op\_profit(f(x_k), \theta) + fail\_cost(f(x_k), \theta)$$

Where:

$$op\_profit(f(x_k), \theta) = -\max(0, \min\{i \in I_k \mid f(x_{ki}) < \theta\} - s)$$

$$fail\_cost(f(x_k), \theta) = \begin{cases} C \text{ if } f(x_{ki}) \geq \theta & \forall i \in I_k \\ 0 \text{ otherwise} \end{cases}$$

- s units of machine operation are guaranteed
- ...So we gain over the default policy only if we stop after that
- Profit is modeled as a negative cost

The Cost Function

Normally, we would determine s and C by talking to a domain expert

...In our case wi well pick reasonable values based on our data

- First, we collect all failure times:

```
In [3]: tr_failtimes = tr.groupby('machine')['cycle'].max()
```

- Then, we define s and C based on statistics:

```
In [4]: safe_interval = tr_failtimes.min()
        maintenance_cost = tr_failtimes.max()
```

- For the safe interval s, we choose the minimum failure time
- For the maintenance cost C we choose the largest failure time

We are taling about jet engines, so failing is BAD

Calibration and Policy Definition Problem

Our calibration problem is then in the form:

$$\text{argmin}_\theta \sum_{k=1}^{n_r} cost(f(x_k), x_k, \theta)$$

- If we pair it with our previous training step
- ...We obtain a formulation for the entire policy definition problem:

$$\text{argmin}_\theta \sum_{k=1}^{n_r} cost(f(x_k, \omega^*), x_k, \theta)$$

$$\text{where: } \omega^* = \text{argmin}_\omega \{ L(y, \hat{y}) \mid y = f(x, \omega) \}$$

This is how we should have started in the first place

Solving the Calibrarion Problem

Solving the calibration problem is very easy:

$$\text{argmin}_\theta \sum_{k=1}^{n_r} cost(f(x_k, \omega^*), x_k, \theta)$$

$$\text{where: } \omega^* = \text{argmin}_\omega \{ L(y, \hat{y}) \mid y = f(x, \omega) \}$$

- We need to optimize a single (scalar) variable, i.e. θ
- ...And changing θ does not impact the optimal ω

This is a univariate optimization problem

- The cost function is non-differentiable
- ...But the problem is so simple that even grid search will work very well

Solving the Calibration Problem

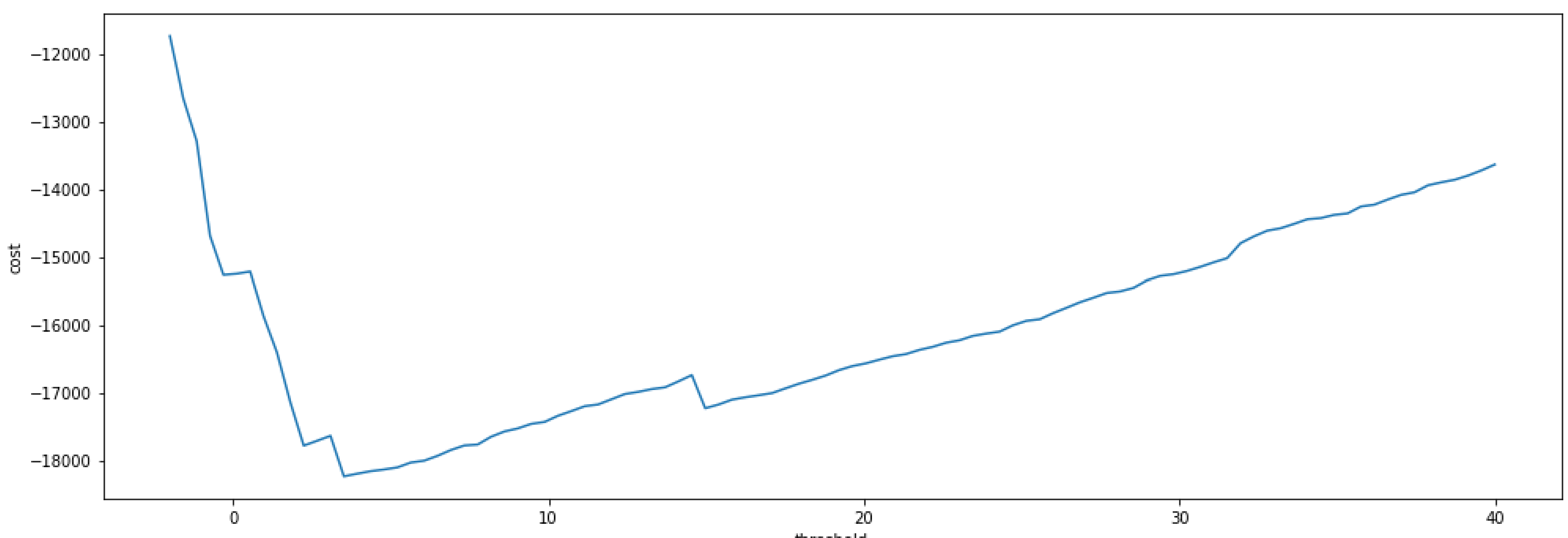
We can sample a range of values for the θ parameter

...Then simply pick the value with the smallest cost

- The code in `optimize_threshold` can also plot the corresponding cost surface

```
In [6]: cmodel = util.RULCostModel(maintenance_cost=maintenance_cost, safe_interval=safe_interval)
        th_range = np.linspace(-2, 40, 100)
        tr_thr = util.optimize_threshold(tr['machine'].values, tr_pred, th_range, cmodel, plot=True, fig
        print(f'Optimal threshold for the training set: {tr_thr:.2f}')
```

```
Optimal threshold for the training set: 3.52
```



Evaluation

Finally, we can check how we are doing on the test set:

```
In [7]: tr_c, tr_f, tr_sl = cmodel.cost(tr['machine'].values, tr_pred, tr_thr, return_margin=True)
        ts_c, ts_f, ts_sl = cmodel.cost(ts['machine'].values, ts_pred, tr_thr, return_margin=True)
        print(f'Cost: {tr_c} (training), {ts_c} (test)')

Cost: -18238 (training), -7075 (test)
```

We can also evaluate the margin for improvement:

```
In [8]: print(f'Avg. fails: {tr_f/len(tr_mcn):.2f} (training), {ts_f/len(ts_mcn):.2f} (test)')
        print(f'Avg. slack: {tr_sl/len(tr_mcn):.2f} (training), {ts_sl/len(ts_mcn):.2f} (test)')

Avg. fails: 0.01 (training), 0.00 (test)
Avg. slack: 15.06 (training), 11.63 (test)
```

- Slack = distance between when we stop and the failure
- The results are actually quite good!
- ...And we also generalize fairly well

Some Considerations

In principle, RUL regression is a very hard problem

- Our linearly decreasing RUL assumption is just a rough oversimplification
- ...RUL is inherently subject to stochastisticy
- ...And depends on the how the machine will be used

But we don't care, since RUL prediction was not our true problem

The real problem involved both prediction and optimization

- We had to optimize the NN parameters (to obtain good predictions)
- We had to optimize the threshold

The ultimate goal was to reduce maintenance cost

Keep in mind the big picture

- In a "predict, then optimize" setting
- ...Quality should be judged on the final cost