Showerflow

07/11/2024

Intro

Thanks Duncan for fixing a bug in `generate.py`. This is now merged.

Following from last time;

1) Plotted variants to be compared on same axis.
2) Tried training base distribution.
3) Tried only one block.
4) Tip from Thorsten – tried working with positive-only values in log space.
5) Investigated cause of increasing gradient norms.

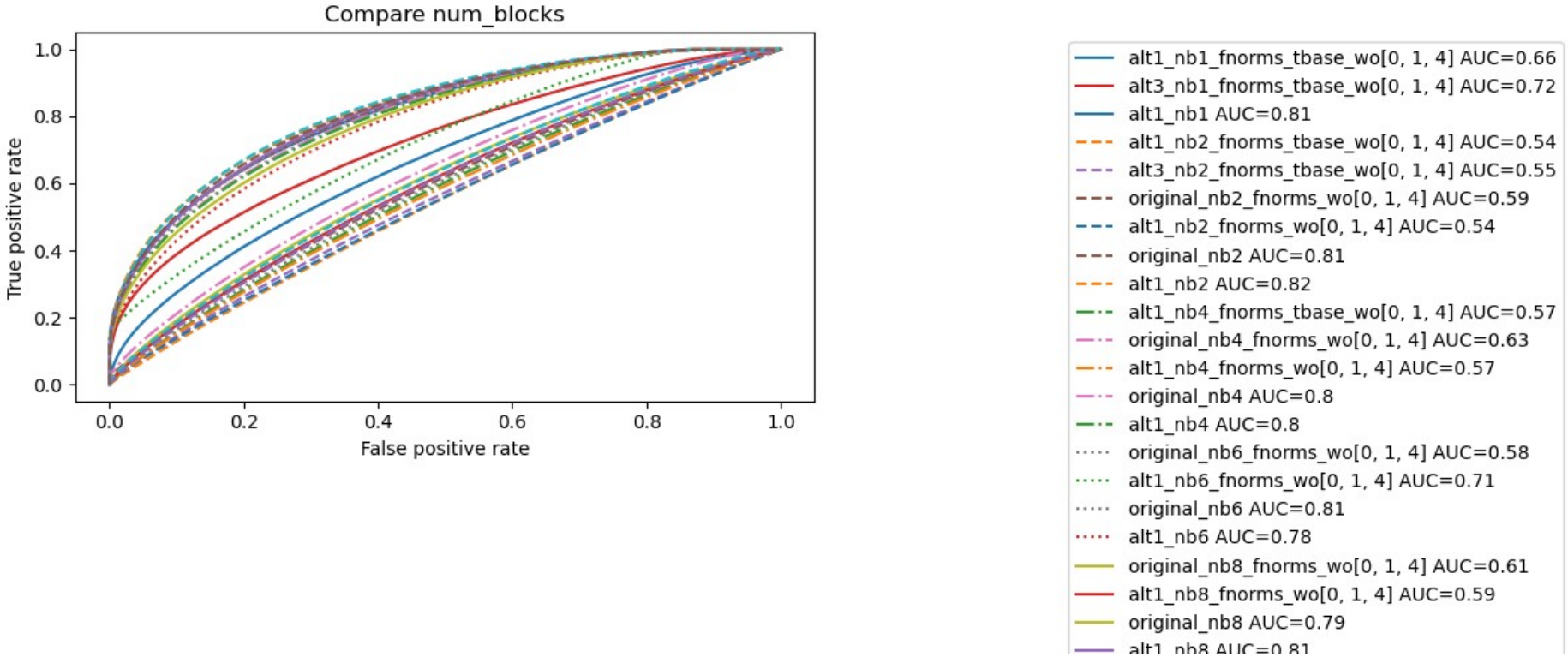
Naming conventions

- original/alt1/alt3 → transforms used in a block
- nb1/nb2/nb4… → Number of blocks
- Fnorms → Don’t predict totals and normalise by constant factor
- Tbase → train the base distirbution too

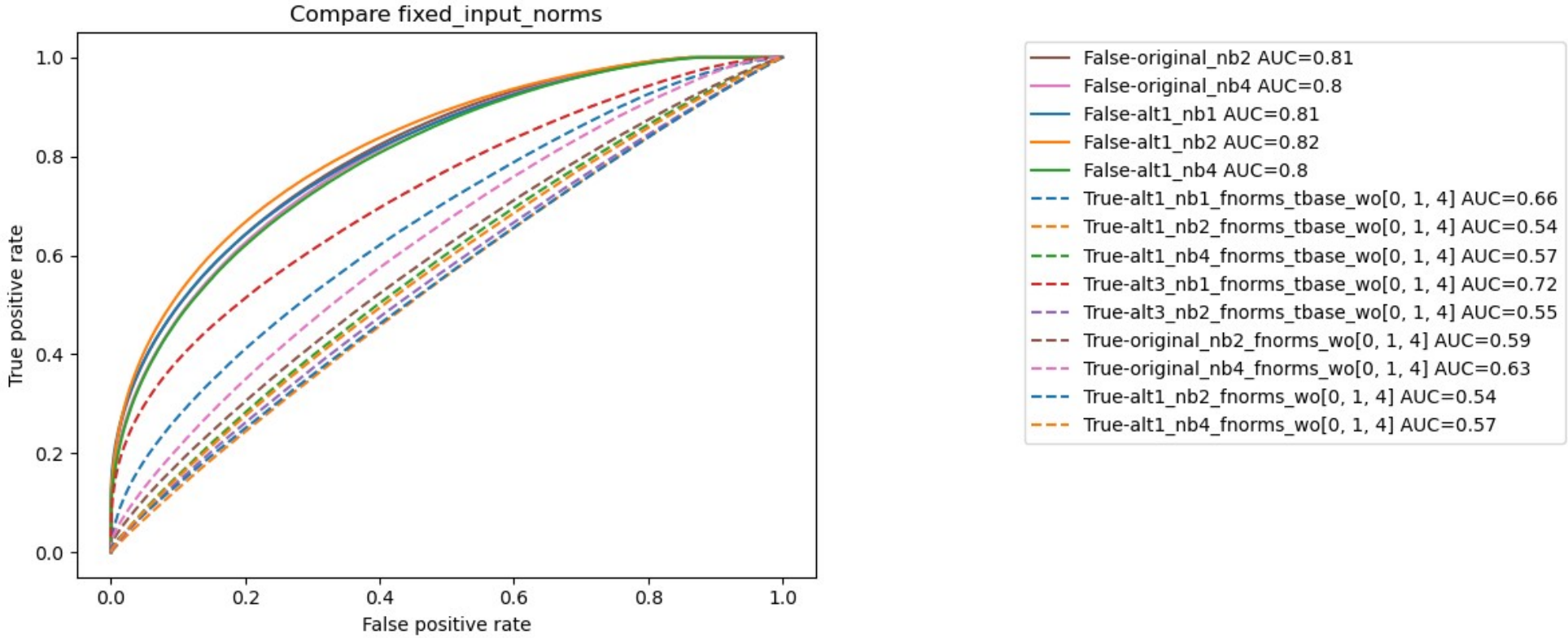
Caveat emptor

- Alt3 is discriminator may be insufficiently trained.
- Alt1 with tbase and 1 block may be insufficiently trained.

Num blocks


Compare num_blocks
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0
alt1_nb1_fnorms_tbase_wo[0, 1, 4] AUC=0.66
alt3_nb1_fnorms_tbase_wo[0, 1, 4] AUC=0.72
alt1_nb1 AUC=0.81
alt1_nb2_fnorms_tbase_wo[0, 1, 4] AUC=0.54
alt3_nb2_fnorms_tbase_wo[0, 1, 4] AUC=0.55
original_nb2_fnorms_wo[0, 1, 4] AUC=0.59
alt1_nb2_fnorms_wo[0, 1, 4] AUC=0.54
original_nb2 AUC=0.81
alt1_nb2 AUC=0.82
alt1_nb4_fnorms_tbase_wo[0, 1, 4] AUC=0.57
original_nb4_fnorms_wo[0, 1, 4] AUC=0.63
alt1_nb4_fnorms_wo[0, 1, 4] AUC=0.57
original_nb4 AUC=0.8
alt1_nb4 AUC=0.8
original_nb6_fnorms_wo[0, 1, 4] AUC=0.58
alt1_nb6_fnorms_wo[0, 1, 4] AUC=0.71
original_nb6 AUC=0.81
alt1_nb6 AUC=0.78
original_nb8_fnorms_wo[0, 1, 4] AUC=0.61
alt1_nb8_fnorms_wo[0, 1, 4] AUC=0.59
original_nb8 AUC=0.79

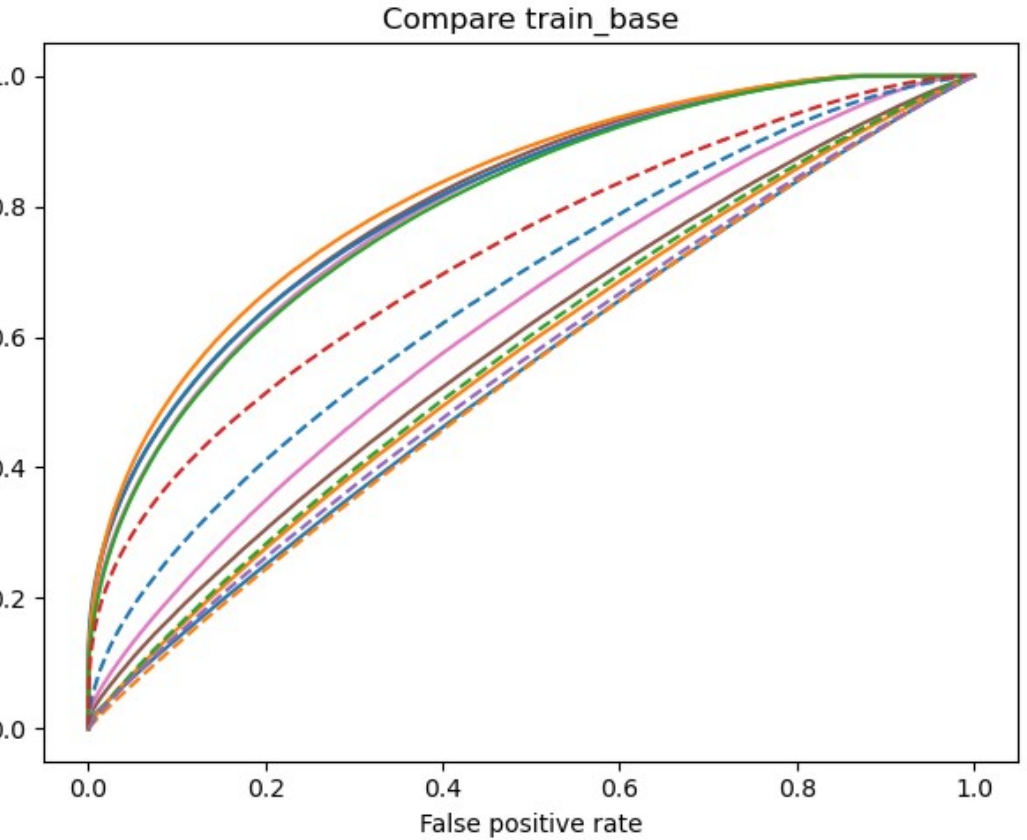
Fixed input norms



Tbase; train the base distribution

```
 7 class GaussModule(nn.Module):
 8     """
 9     Create a simple, flexible module with a set of independent normal distributions.
10     In addition to the usual torch.nn.Module behavior it has a `distribution` attribute
11     that is a torch.Distribution.
12     """
13     def __init__(self, num_inputs, device, **kwargs):
14         """
15         Constructor.
16 
17         Parameters
18         ----------
19         num_inputs : int
20             The dimension required for a single event (set of independent normals).
21         device : str
22             "cpu" or "gpu", the device to compute on.
23 
24         """
25         super().__init__()
26         self.loc = nn.Parameter(torch.zeros(num_inputs).to(device))
27         self.scale = nn.Parameter(torch.ones(num_inputs).to(device))
28         self.distribution = dist.Normal(self.loc, self.scale)
```

Tbase; train the base distribution



- False-original_nb2_fnorms_wo[0, 1, 4] AUC=0.59
- False-original_nb4_fnorms_wo[0, 1, 4] AUC=0.63
- False-alt1_nb2_fnorms_wo[0, 1, 4] AUC=0.54
- False-alt1_nb4_fnorms_wo[0, 1, 4] AUC=0.57
- False-original_nb2 AUC=0.81
- False-original_nb4 AUC=0.8
- False-alt1_nb1 AUC=0.81
- False-alt1_nb2 AUC=0.82
- False-alt1_nb4 AUC=0.8
- True-alt1_nb1_fnorms_tbase_wo[0, 1, 4] AUC=0.66
- True-alt1_nb2_fnorms_tbase_wo[0, 1, 4] AUC=0.54
- True-alt1_nb4_fnorms_tbase_wo[0, 1, 4] AUC=0.57
- True-alt3_nb1_fnorms_tbase_wo[0, 1, 4] AUC=0.72
- True-alt3_nb2_fnorms_tbase_wo[0, 1, 4] AUC=0.55

Alt3; Positive only values in log space.

```python
def compile_HybridTanH_alt3(
    num_blocks, num_inputs, num_cond_inputs, device, train_base=False
):
    factory = HybridTanH_factory(num_inputs, num_cond_inputs, device)

    transform_pattern = [
        "affine_coupling",
        "permutation",
        "affine_coupling",
        "permutation",
        "affine_coupling",
        "permutation",
        "spline_coupling",
        "permutation",
        "affine_coupling",
        "permutation",
        "affine_coupling",
        "permutation",
        "affine_coupling",
        "permutation",
    ]
    transform_pattern = sum([transform_pattern] * num_blocks, [])
    # if we predict the totals, make them positive
    exponential_ranges = [] if num_inputs < 65 else [(0, 2)]
    # the last 60 layers should be always positive
    exponential_ranges += [(num_inputs - 60, num_inputs)]
    transform_pattern.append("partial_exponential")

    model, flow_dist, transforms = factory.create(
        1,  # just one, becuase we have repeated inside our pattern
        transform_pattern,
        count_bins=8,
        train_base=train_base,
        exponential_ranges=exponential_ranges,
    )
    return model, flow_dist, transforms
```

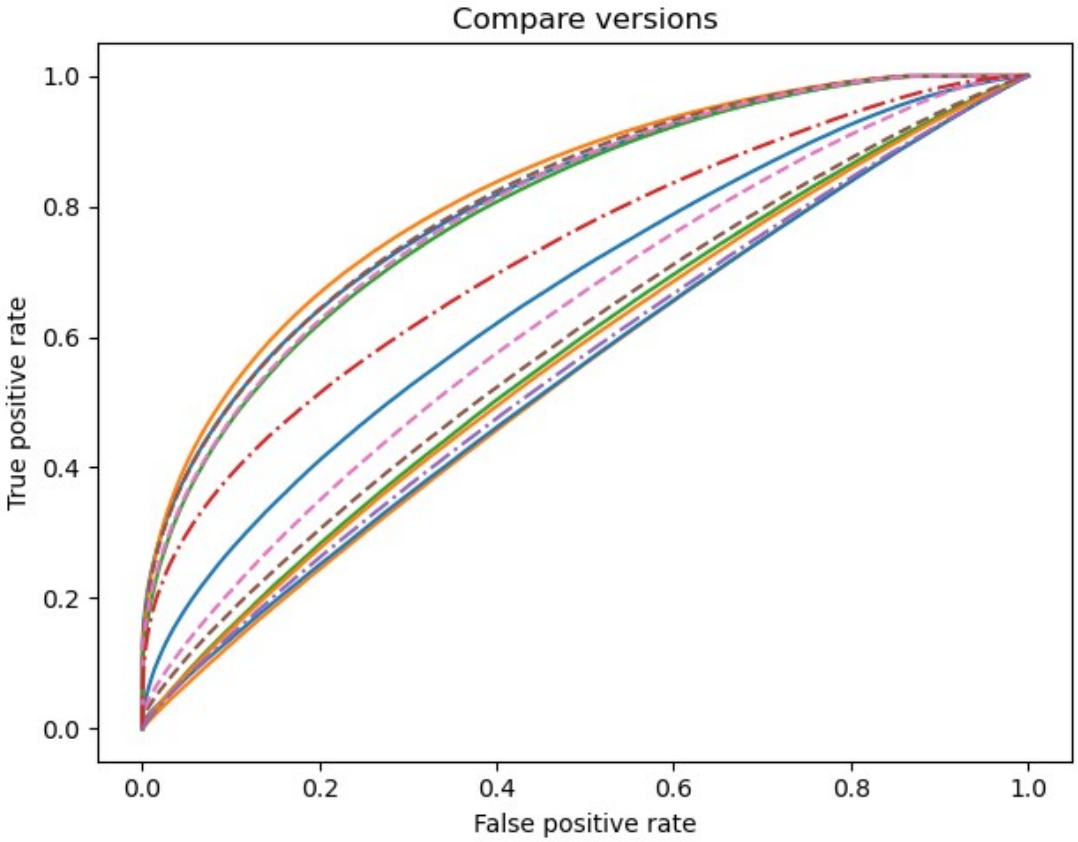
```python
    def add_partial_exponential(self, exponential_ranges, **kwargs):
        """
        Add one exponential transformation to the flow being constructed.
        Probably won't work unless it's only the last transform used.

        Under the hood, the distribution is shifted into the positive by
        a small amount, after the exponential transformation is applied.
        This avoids issues with the log of 0 when moving back to the original space.
        """
        identity = torch.distributions.transforms.identity_transform
        exponential = torch.distributions.transforms.ExpTransform()
        tiny_shift = torch.distributions.transforms.AffineTransform(
            loc=-torch.tensor(0.1).to(self.device),
            scale=torch.tensor(1.0).to(self.device),
        )
        change_points = np.array(exponential_ranges).flatten()

        lengths = np.diff(change_points).tolist()
        transforms = [exponential, identity] * len(exponential_ranges)
        shifts = [tiny_shift, identity] * len(exponential_ranges)
        if change_points[0] != 0:
            transforms = [identity] + transforms
            shifts = [identity] + shifts
            lengths = [change_points[0]] + lengths
        if change_points[-1] < self.num_inputs:
            lengths.append(self.num_inputs - change_points[-1])
        else:
            transforms = transforms[:-1]
            shifts = shifts[:-1]

        partial = torch.distributions.transforms.CatTransform(
            transforms, dim=-1, lengths=lengths
        )
        self.transforms.append(partial)
        shift = torch.distributions.transforms.CatTransform(
            shifts, dim=-1, lengths=lengths
        )
        self.transforms.append(shift)
```

Alt3; Positive only values in log space.

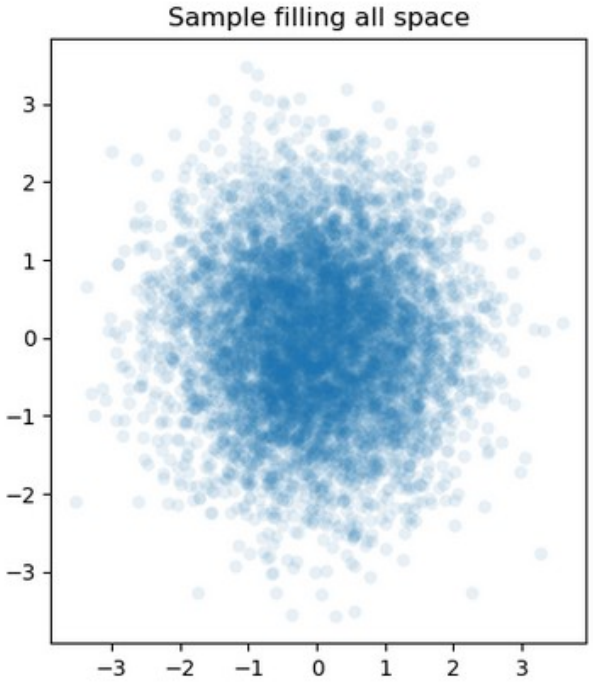


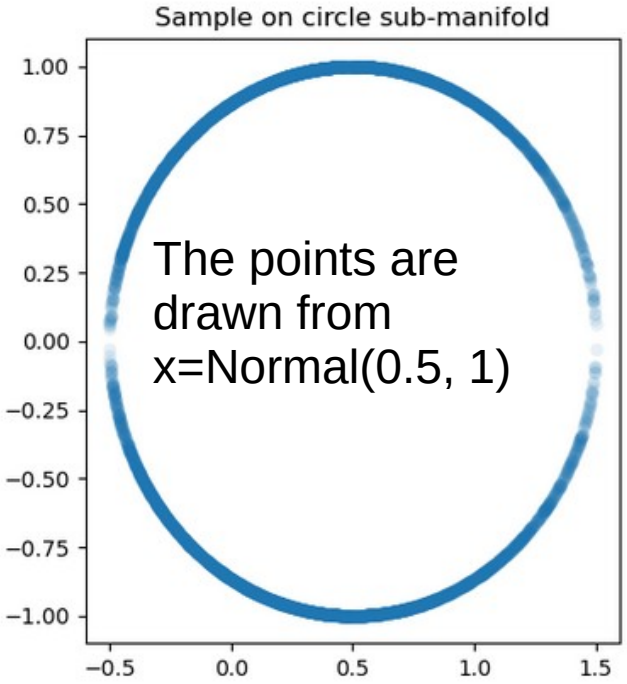
Take away

- 1 block is slightly worse than 2 blocks, but still better than 8 blocks.
- Fixed input norms/removing the totals is the clearest improvement.
- Training the base distribution doesn’t seem to do much, but a bit early to tell.
- Taking the positive-only values into log space isn’t a good solution. Possible loss of precision in high values? Not sure.

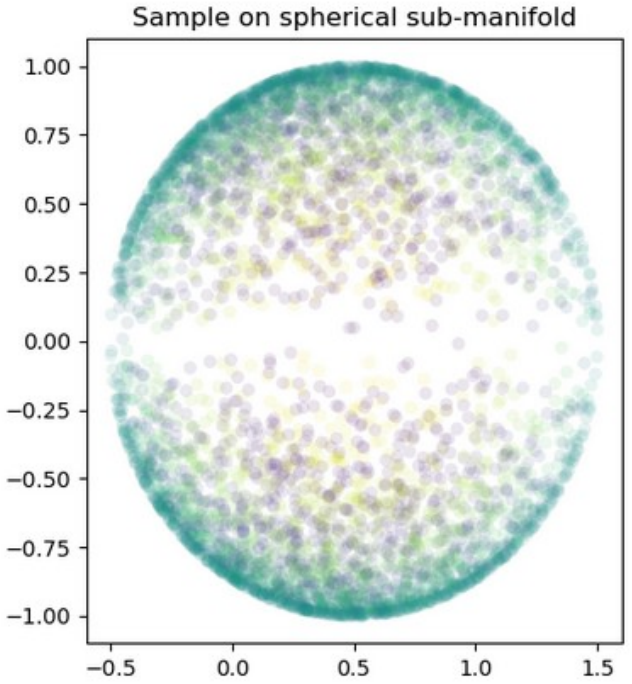
Now – some analysis on a toy model.

Three distributions

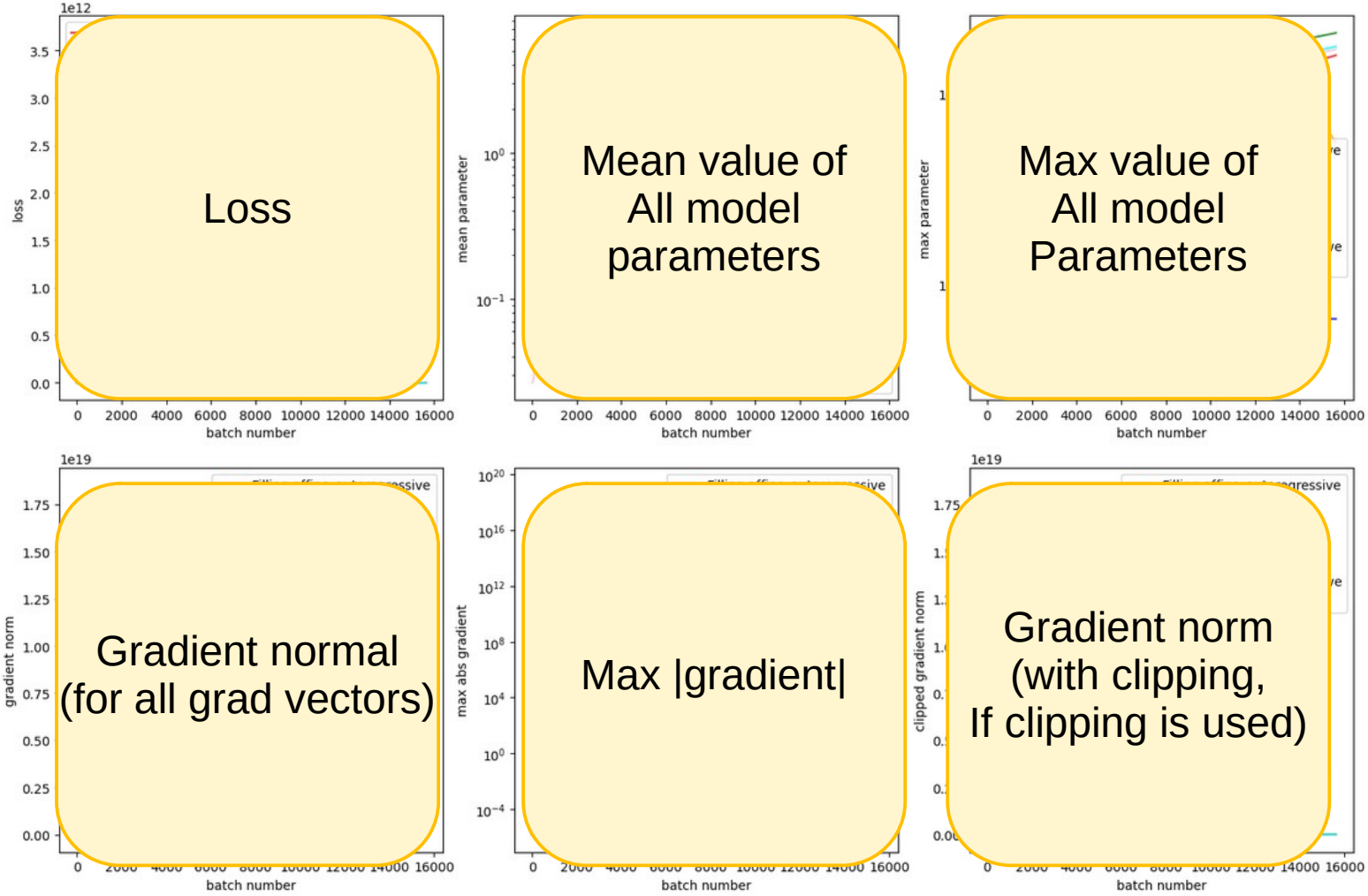




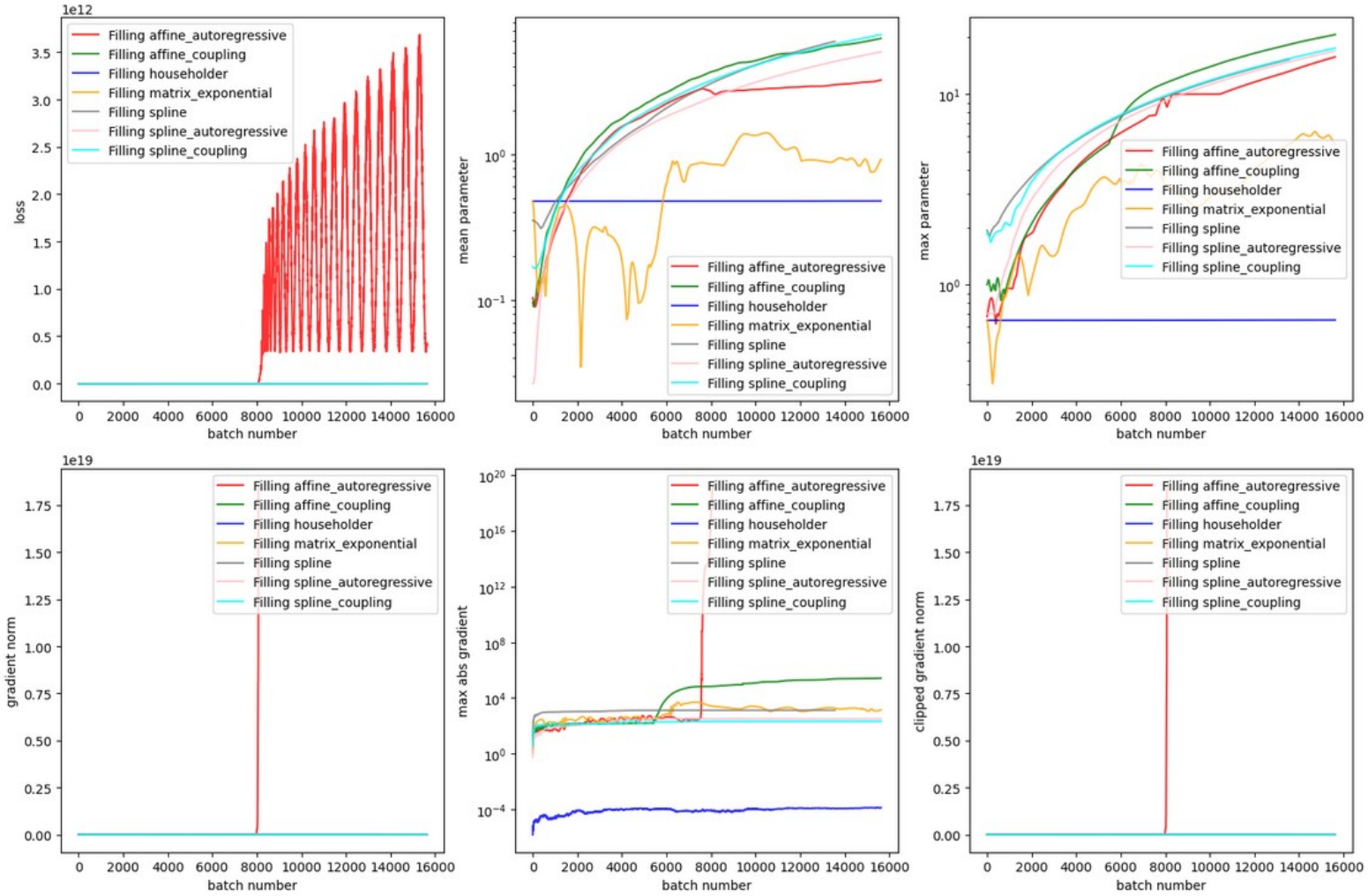
The points are drawn from x=Normal(0.5, 1)



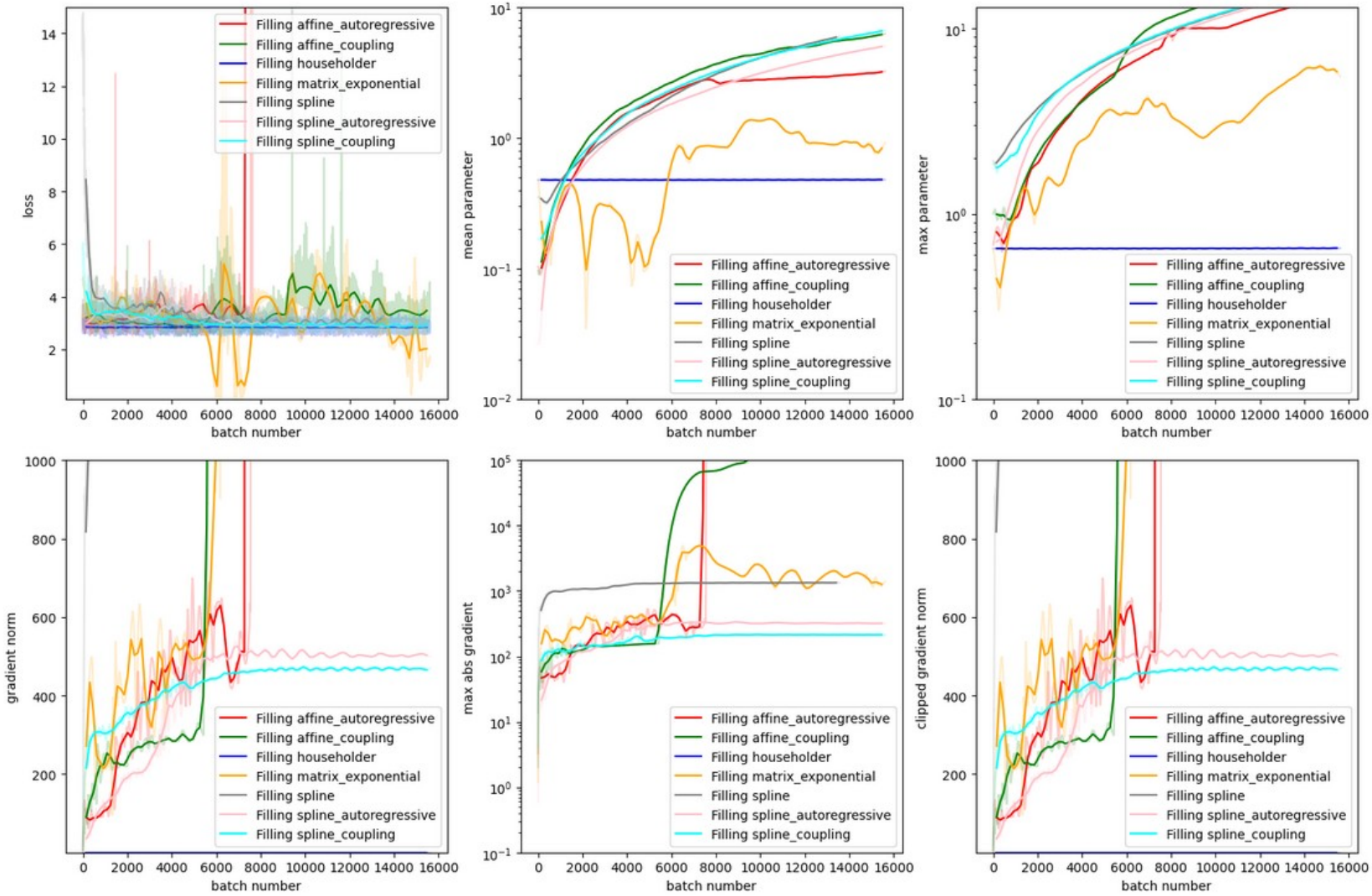
Train while tracking values



Train while tracking values



Train while tracking values



Gave up copying from notbook…. sorry