MSA Scoring

Based on MSA Transformer

Group lianyh, tengyue & yangxch

MSA Input Preprocessing

- Read names, sequences and scores
  - shape of MSA input : `[num_sequence, sequence_len]`
- Remove insertions (lowercase letters) of sequences
- Subsampling MSAs
  - subsample `>256` sequences to `256` sequences per MSA
  - HH-Filter: `>256` to `≈256` sequences
    - `hhfilter -i input.a3m -o filtered.a3m -diff 256`
  - Diversity Maximizing: `≈256` to `256` sequences
    - greedily pick sequence with max hamming distance
  - result shape: `[256, sequence_len]`

MSA Embeddings from MSA Transformer

- Use `esm.pretrained.esm_msa1b_t12_100M_UR50S()`
- Convert MSA Input to Batch Input
  - convert letters to numeric tokens
  - Add `<begin_of_sequence>` token and paddings for each sequence
  - result shape: `[batch_size, 256, 1+sequence_len]`
- Obtain MSA embeddings from MSA Transformer
  - MSA Transformer uses `embedding_size = 768`
  - model output shape: `[batch_size, 256, 1+sequence_len, 768]`
  - we extract the query sequence and discard `<begin_of_sequence>`
  - result embedding shape: `[sequence_len, 768]` for each MSA

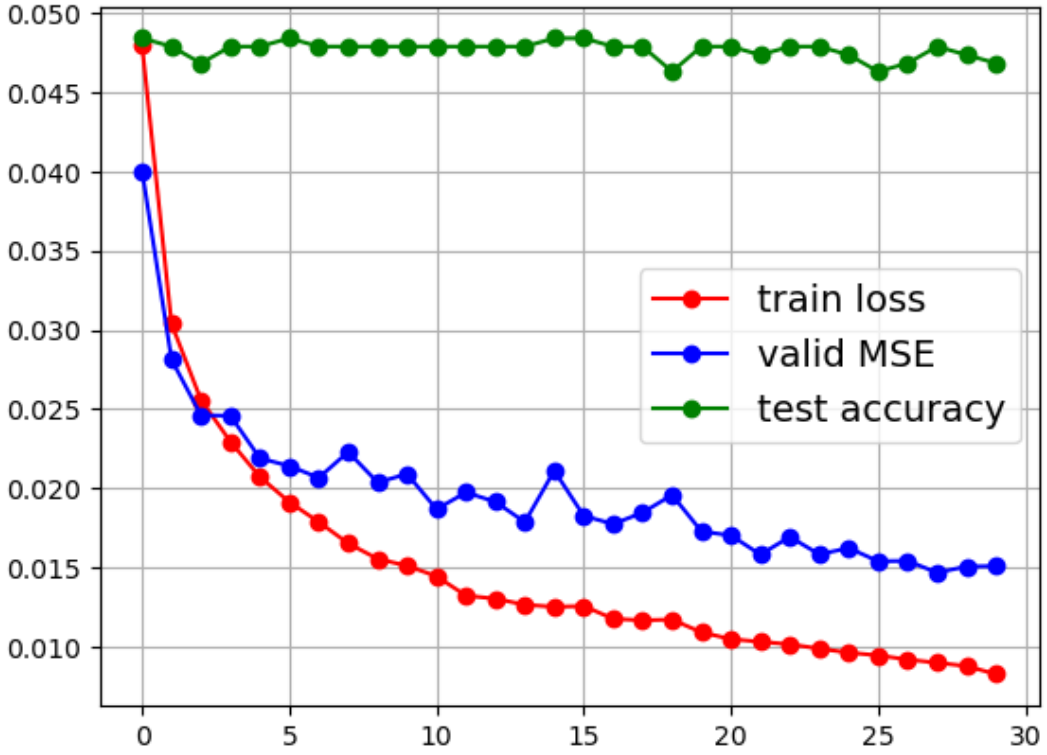
MSA Scoring Network

- Now we have `[sequence_len, 768]` embedding for each MSA
  - consider it as a feature map (image) and feed into CNN
  - however, for experiment, we average it over the `sequence_len` axis
  - result shape: a `768`-dim vector for each MSA

- For experiment, we simply train a MLP
  - input `768`-dim vector -> Fully-Connected Layers -> output score
  - reach `0.00826` train MSE loss after `30` epochs
  - prediction accuracy on test set: `0.9579` after `20` epochs

MSA Scoring Network

- Reach `0.00826` train MSE loss after `30` epochs
- Test accuracy: `0.9579`
  - `91` correct out of `95` pairs of MSA
  - high from the first training epoch
- Naïve baseline for comparison:
  - output MSA with more sequences
  - baseline test acc: `0.7789`
  - a great boost!



MSA Scoring Network

- Some correctly predicted pairs:

```
256x 194 y_gt:0.9624 ✓ y_pred:90.7269 ✓
206x 194 y_gt:0.5687   y_pred:62.0711

256x 465 y_gt:0.7737 ✓ y_pred:70.7444 ✓
256x 465 y_gt:0.5517   y_pred:49.1714
```

```
 50x 133 y_gt:0.2689   y_pred:40.9198
256x 133 y_gt:0.9678 ✓ y_pred:56.2354 ✓

 48x  93 y_gt:0.6630 ✓ y_pred:67.0188 ✓
256x  93 y_gt:0.2038   y_pred:32.6395
```

- Some wrongly predicted pairs:

```
256x 155 y_gt:0.8572 ✓ y_pred:79.3215
203x 155 y_gt:0.8149   y_pred:85.1131 ✗

200x  60 y_gt:0.9280 ✓ y_pred:82.2290
256x  60 y_gt:0.8941   y_pred:82.4489 ✗
```

```
 11x 101 y_gt:0.8525 ✓ y_pred:49.7983
256x 101 y_gt:0.3600   y_pred:54.4962 ✗

256x 350 y_gt:0.9542 ✓ y_pred:80.1949
256x 350 y_gt:0.7264   y_pred:82.2742 ✗
```

Future Work

- Try other (combinations of) pipelines:
  - other MSA embedding representations, like a feature map instead of vector
  - other network structures, like deep CNN instead of MLP

- Focus on comparing MSAs of the same query sequence
  - like `T1024-D1_xxx, T1011-D2_xxx` as we would like in the test set
  - might try pairwise loss or triplet loss for MSAs of same query sequence
  - Instead of blindly fitting arbitrary MSA with its score

- Improve explainability
  - try more weakly supervised model?
  - try to apply some XAI methods

Pairwise Cross-Entropy Loss

- `T1024-D1_base_fm` `0.96112` y_1
- `T1024-D1_aug_fm` `0.70208` y_2

$$\Delta = \Pr(y_1 > y_2) = \text{sigmoid}(y_1 - y_2)$$

$$\hat{\Delta} = \hat{\Pr}(y_1 > y_2) = \text{sigmoid}(\hat{y}_1 - \hat{y}_2)$$

$$L = -\Delta \log(\hat{\Delta}) - (1 - \Delta) \log(1 - \hat{\Delta})$$

Triplet Loss

- Current regression model cannot score MSAs >90 well
- Anchor: T1024-D1_cov50_fm 94.43
- Positive: T1024-D1_base_fm 96.112
- Negative: T1024-D1_rand9_fm 58.42

$$L = max(0, \text{margin} + d(x_{\text{anchor}}, x_{\text{positive}}) - d(x_{\text{anchor}}, x_{\text{negative}}))$$