Prediction of hashtag flow in Twitter

Avirup Saha, Soumyadeep Roy, Srinidhi Moodalagiri, Kushal Gaikwad, Abhay Shukla

Introduction

In general, the popularity of a hashtag depends on the two primary factors (i) hashtag tweet reinforcement and (ii) hashtag-hashtag competition. We wish to predict the popularity of a hashtag using these features. Towards this end we propose two approaches:

- Linear memoryless model (LMM) which is a novel approach that relies on hashtag-tweet reinforcement.
- SeqGAN model (as described in (Yu et al. 2017)) which explicitly models inter-hashtag competition.

Problem formulation for linear memoryless model (LMM)

- We model the conditional intensity, $\lambda(t)$ of the temporal point process in terms of the tweet-popularity index k (a measure of the number of re-tweets of a given tweet) and the event history of the point process. We use the tweet-popularity index to model the phenomenon of hashtag-tweet reinforcement. That is, if a tweet becomes is popular (i.e. re-tweeted many times) then all the hashtags present in the tweet also become popular i.e. are used more frequently in other tweets.
- We use this conditional intensity to predict the expected number of tweets bearing the same hashtag posted in a subsequent time interval.
- We evaluate the accuracy of the prediction by comparing our predicted value with the ground truth.

We make the following assumptions as mentioned in (Samanta et al.):

- There are a finite set of hashtags $H = \{H_1, H_2, ... H_p\}$
- Each tweet $tweet_j$ is associated with a time instant t_j and a hashtag set $HS_j = \{H_n \mid H_n \text{ is contained in } tweet_j\} \subseteq H$. Thus there is a many-to-many relationship between the set of hashtags and the set of tweets.
- Only one tweet is generated at a time instant t_j and there exists a total ordering $<$ over the set of tweets based on their timestamps.

Definitions

- Formally, a tweet-chain C_i, corresponding to tweet i, can be written as $C_i = \{t_j | \text{tweet } i \text{ is (re)tweeted at time } t_j\}$.
- Similarly, a hashtag $\boldsymbol{H}$ can be expressed in terms of the tweet-chains, $\boldsymbol{H} = \{C_i | C_i \text{ is a tweet-chain bearing } \boldsymbol{H}\}$.
- Finally, the *history* of the hashtag $\boldsymbol{H}$ until and excluding time t, $\mathcal{H}_{\boldsymbol{H}}(t)$ can be represented as the union of the posting times of the corresponding (re)tweets posted before t.

$$\mathcal{H}_{\boldsymbol{H}}(t) = \cup_{C_i \in \boldsymbol{H}} \{t_j | t_j \in C_i \text{ and } t_j < t\}$$

- Given a hashtag $\boldsymbol{H}$, we define the counting variable as $N_{\boldsymbol{H}}(t)$, where $N_{\boldsymbol{H}}(t) \in \{0\} \cup \mathbb{Z}^+$ counts the number of (re)tweets posted until and excluding time t.
- The conditional probability of observing an event in infinitesimal time interval $[t, t + dt)]$ is characterized as

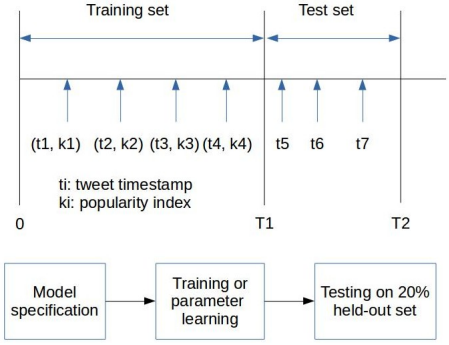
$$\mathbb{P}(\text{An event triggers in } [t, t + dt) | \mathcal{H}_{\boldsymbol{H}}(t)) = \lambda_{\boldsymbol{H}}(t)\, dt \tag{1}$$

$$\text{i.e., } \mathbb{E}_{dN_{\boldsymbol{H}}(t) \sim \{0,1\}}[dN_{\boldsymbol{H}}(t) | \mathcal{H}_{\boldsymbol{H}}(t)] = \lambda_{\boldsymbol{H}}(t) dt \tag{2}$$

- Here $dN_{\boldsymbol{H}}(t)$ indicates the number of (re)tweets in the infinitesimal time-window $[t, t+dt)$ and $\lambda_{\boldsymbol{H}}(t)$ stands for the associated hashtag intensity, which further depends on the history $\mathcal{H}_{\boldsymbol{H}}(t)$.
- We estimate $\lambda_{\boldsymbol{H}}(t)$ using parameters learned from a given dataset using a neural network model.

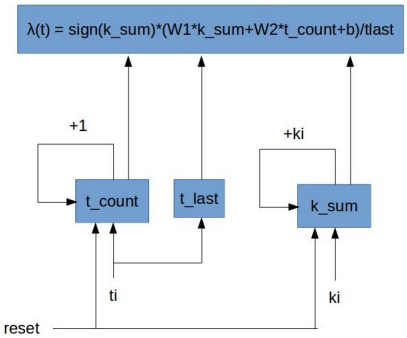
Model

Overview



We divide both training and test sets into several time intervals of equal width.

Training model



Here we have

$$\lambda(t) = sign(k\_sum)*(W1*k\_sum+W2*t\_count+b)/tlast$$

W1, W2 and b are learned by minimizing the SSE in training intervals. Thus for a hashtag H, if $X_{H,n}$ and $Y_{H,n}$ are the observed and predicted counts in the nth training interval, the loss function is given by

$$\mathbb{L} = \sum_{n=1}^{N} \mid X_{H,n} - Y_{H,n} \mid^2 \tag{3}$$

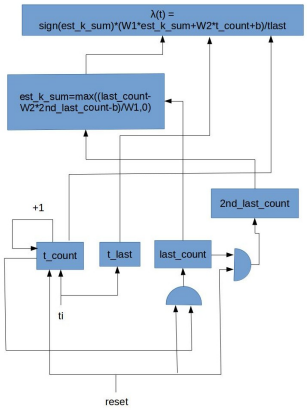
where N is the number of training intervals and is considered here as a hyper-parameter. We use the following features:

- **t_count:** Count of the tweets observed so far bearing a given hashtag. This value is reset to zero at the beginning of every interval.
- **k_sum:** Sum of the popularity indices of all tweets observed so far in the current interval bearing a given hashtag. This feature indicates the popularity of the hashtag in the current interval.
- We use a modified linear combination of these two features with a bias.
- sign(k_sum) = 0 if k_sum = 0 and 1 if k_sum > 0. This avoids false predictions in empty intervals.

Intuition behind Test model

- We don't know the values of **k_sum** for the test intervals.
- If the intervals are of sufficiently small width, the value of **k_sum** for the current interval will be roughly the same as the value for the last interval.
- We have to estimate the value of **k_sum** for the last interval from the given data.
- We perform this estimation by using the learned values of the parameters together with the values of **t_count** observed for the last and 2nd last intervals.

Test model

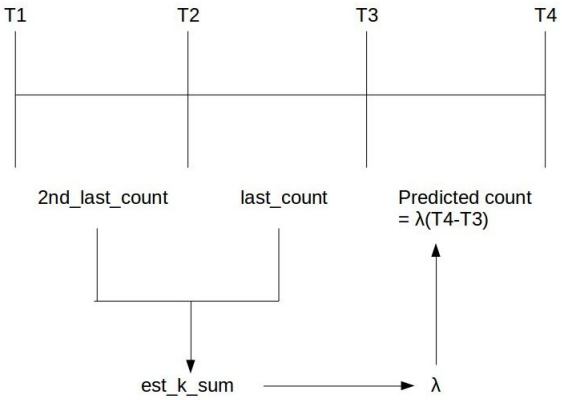


Here we use the equations

$$est\_k\_sum = max((last\_count - W2 * 2nd\_last\_count - b)/W1, 0)$$
$$\lambda(t) = sign(est\_k\_sum) * (W1 * est\_k\_sum + W2 * t\_count + b)/tlast$$

Here W1, W2 and b are fixed to the values learned by the training model.

Prediction in the test set



Results

Evaluation Metrics

- **Mean Absolute Percentage Error (MAPE):** It captures the mean deviation between the observed and the predicted popularity for a hahstag up to time t. It is defined by the formula

$$\text{MAPE}(\boldsymbol{H}) = \frac{1}{M_{\boldsymbol{H}}} \sum_{i=0}^{M_{\boldsymbol{H}}-1} \left| \frac{\hat{N}_{\boldsymbol{H}}(t_i) - N_{\boldsymbol{H}}(t_i)}{N_{\boldsymbol{H}}(t_i)} \right|. \tag{4}$$

  Here $\hat{N}_{\boldsymbol{H}}(t)$ and $N_{\boldsymbol{H}}(t)$ are the estimated and actual number of retweets for a hashtag $\boldsymbol{H}$ respectively, at time t. $M_{\boldsymbol{H}}$ denotes the total number messages of hashtag $\boldsymbol{H}$ in the test-set.

- **Spearman's Rank Correlation Coefficient (SRCC):** The Spearman's Rank Correlation Coefficient between predicted rank-list $\hat{R}_{\mathbb{H}}$, and actual rank-list $R_{\mathbb{H}}$ of a hashtag set $\mathbb{H}$ can be defined as,

$$\rho(\hat{R}_{\mathbb{H}}, R_{\mathbb{H}}) = \frac{\mathbf{Cov}(\hat{R}_{\mathbb{H}}, R_{\mathbb{H}})}{\sqrt{\mathbf{Var}(\hat{R}_{\mathbb{H}})\mathbf{Var}(R_{\mathbb{H}})}}. \tag{5}$$

Here **Cov**(.) defines covariance of the two variables, and **Var**(.) denotes the variance.

- **Avg. Recall (AvRe) and Avg. Precision (AvPr) (for jump detection):** If in two consecutive time-intervals there is a sudden change in the rank of a hashtag by more than half the total no. of competing hashtags, we call it a *jump*.
- Let a hashtag $\boldsymbol{H} \in \mathbb{H}$ have a rank $\text{rank}_{\boldsymbol{H},[t_i,t_{i+1})}$ in the time-interval $[t_i, t_{i+1})$. Then, if $|\text{rank}_{\boldsymbol{H},[t_i,t_{i+1})} - \text{rank}_{\boldsymbol{H},[t_{i-1},t_i)}| \geq |\mathbb{H}|/2$, it is considered a jump.
- Recall measures the fraction of cases in which an algorithm rightly identifies a jump, while precision measures the fraction of cases where a real jump has occurred when the algorithm predicts a jump.

Datasets

- As explained in (Samanta et al.), seven datasets were created by carefully selecting a few hashtags from raw data obtained using Twitter search API to collect all the tweets related to the following events (corresponding to a 2-3 weeks period around the event date):
- Events: (i) *The Academy Awards 2016* (Oscars), (ii) *MTV Awards 2016* (MTV), (iii) *Earthquake in Nepal 2015* (Nepal-Earthquake), (iv) *Democratic Primaries for US Presidential Election 2016* (Dem-Primary), (v) *Big Billion Day sale of e-commerce site Flipkart 2014* (BBD), (vi) *Copa America Football Tournament 2016* (Copa), and (vii) *T20 Cricket World Cup 2016* (T20WC).

Evaluation Protocol

Training and testing: Given a stream of temporal data $\mathcal{H}_{\boldsymbol{H}}$ for hashtags $\boldsymbol{H} \in \mathbb{H}$, we first split it into training and test set where training comprises of the first 80% of the total number of messages ($|\mathcal{H}|$). We use this 80% messages as input to train our model for estimating the parameters. We divide the training-time in several ten-hour intervals and compare the popularity of the competing hashtags in each of them. The estimated model is thereafter used to predict the popularity dynamics of the hashtags in the test set. In order to determine the predictive prowess of LMPP (and the baselines), we follow two different evaluation approaches.
(i) Forecasting hashtag popularity: In this approach, the predictor aims to forecast the hashtag popularity by computing the expected number of message-posts in the test set.
(ii) Rank prediction of competing hashtags: As mentioned earlier, hashtag competition usually exhibits variations in the popularity rankings of the corresponding hashtags. Here, we attempt to retrieve the popularity rankings of competing hashtags in the test-set, which in turn indicates how efficiently an algorithm captures the phenomenon of hashtag competition.

Performance comparison (MAPE)

Table 1 presents a comparative sketch in terms of MAPE on the 20% held-out set. We observe that for all datasets, the linear memoryless model (LMM) achieves lower MAPE compared to all other baselines. It is second only to the SeqGAN model (described later).

Performance comparison (SRCC)

To evaluate the ability of the algorithms to detect variations in the popularity ranking of the competing hashtags, we divide the test time into several intervals of same duration. At each of them, we obtain a ranked list of the participating hashtags, based on the predicted popularity using LMPP as well as the baselines. On the other hand, from the original trace, we derive the ground-truth of these ranked lists. To measure how closely a predicted order matches the actual one, we compute Spearman's rank correlation coefficient (SRCC) between them. Table 1 depicts a comparative analysis in terms of SRCC for all the proposals. We observe that the linear memoryless model performs significantly better than the baselines across all datasets.

Detection of Sudden Changes in Ranking

Besides evaluating the accuracy of prediction of general rank order, the efficacy of a system can be measured in terms of its success in 'catching' the instances where a particular hashtag suddenly becomes popular (or suddenly loses popularity). We extracted such instances from the datasets (around 9% hashtags in the test-set), and checked how accurately the competing algorithms identify those instances. Table 2 reports the average precision and recall of jump detection for algorithms across all the datasets. In both precision and recall, LMM significantly outperforms the other algorithms (except SeqGAN) in most cases. The only exceptions are the *Oscars* and *T20WC* datasets, where LMPP performs better than LMM in terms of precision. This is because LMM tends to make false predictions and its precision suffers as a result.

Experiment on varying the size of the training set

In addition to the above experiments we investigated the behaviour of our model by varying the proportion of the training and test sets and computing the MAPE and SRCC in each of these cases. Figures 1 through 7 depict the results (MAPE and SRCC) obtained by varying the proportion of the training set from 40% to 80% in steps of 10% for each of the seven datasets (equivalently, the proportion of the test set was varied from 60% to 20% in steps of 10%). We observe that MAPE tends to increase as the proportion of the training set increases (or as the proportion of the test set decreases) whereas the SRCC shows a random variation. This surprising behaviour of the MAPE can be explained as follows: In our model the predicted value of the counting variable remains fairly close to the actual value in each test interval no matter how far away it is from the last training interval. Thus the numerator $|\ \hat{N}_{\boldsymbol{H}}(t_i) - N_{\boldsymbol{H}}(t_i)\ |$ remains fairly constant whereas the denominator $N_{\boldsymbol{H}}(t_i)$ keeps increasing as the test interval goes further away from the last training interval. Consequently, the MAPE being an average measure decreases as the number of test intervals increases. Hence as the proportion of the training set increases, the MAPE will also increase as observed in the graphs.

	MAPE(%)								SRCC							
Datasets	**SeqGAN**	**LMM**	**LMPP**	**HTR**	**RPP**	**Hawkes**	**SEISMIC**	**SpikeM**	**SeqGAN**	**LMM**	**LMPP**	**HTR**	**RPP**	**Hawkes**	**SEISMIC**	**SpikeM**
Oscars	**14.33**	**15.90**	18.90	21.78	24.30	19.23	24.79	27.11	**0.89**	**0.86**	0.85	0.68	0.80	0.52	0.10	0.74
MTV-Awards	**4.04**	**4.46**	05.14	06.76	15.37	13.57	19.09	24.45	**0.98**	**0.97**	0.87	0.86	0.81	0.75	0.70	0.82
Nepal-Earthquake	**5.2**	**6.72**	07.50	08.54	22.28	15.42	13.73	17.95	**0.93**	**0.92**	0.91	0.87	0.63	0.32	0.63	0.75
Dem-Primary	**5.12**	**6.68**	08.33	10.50	11.33	11.62	26.09	19.12	**0.93**	**0.92**	0.86	0.68	0.80	0.51	0.10	0.73
BBD	**12.65**	**12.86**	15.40	17.94	19.09	15.94	18.03	20.89	**0.97**	**0.96**	0.95	0.79	0.90	0.91	0.43	0.79
Copa	**10.82**	**11.59**	17.67	20.07	17.75	19.18	23.32	22.44	**0.96**	**0.96**	**0.91**	0.42	0.75	0.88	0.42	0.64
T20WC	**9.4**	**9.85**	10.25	11.90	13.10	15.08	25.55	41.74	**0.98**	**0.98**	**0.87**	0.58	0.83	0.31	0.56	0.47

Table 1: MAPE (%) and SRCC of proposed and baseline algorithms on all datasets with 20% held-out set. The cells with light orange (blue) color indicates the best (second best) predictor.

	Avg. Precision								Avg. Recall							
Datasets	**SeqGAN**	**LMM**	**LMPP**	**HTR**	**RPP**	**Hawkes**	**SEISMIC**	**SpikeM**	**SeqGAN**	**LMM**	**LMPP**	**HTR**	**RPP**	**Hawkes**	**SEISMIC**	**SpikeM**
Oscars	**0.76**	0.62	**0.74**	0.54	0.32	0.38	0.31	0.33	**0.83**	**0.80**	0.75	0.45	0.32	0.37	0.33	0.34
MTV-Awards	**0.88**	**0.86**	0.31	0.30	0.30	0.31	0.31	0.30	**0.83**	**0.75**	0.33	0.32	0.33	0.33	0.30	0.32
Nepal-Earthquake	**0.81**	**0.78**	0.61	0.60	0.37	0.28	0.40	0.44	**0.96**	**0.92**	0.70	0.54	0.37	0.33	0.52	0.57
Dem-Primary	**0.77**	**0.73**	0.69	0.56	0.48	0.30	0.45	0.34	**1.0**	**1.0**	**0.72**	0.49	0.48	0.29	0.57	0.36
BBD	**0.92**	**0.89**	0.66	0.48	0.32	0.55	0.31	0.43	**1.0**	**1.0**	**0.67**	0.48	0.32	0.64	0.28	0.40
Copa	**1.0**	**1.0**	**0.72**	0.29	0.42	0.60	0.29	0.34	**0.9**	**0.88**	0.59	0.33	0.42	0.53	0.33	0.42
T20WC	**0.8**	0.75	**1.0**	0.32	0.64	0.10	0.29	0.65	**1.0**	**1.0**	**0.67**	0.32	0.64	0.10	0.21	0.54

Table 2: Average precision and recall in jump detection for all algorithms. Orange (blue) indicates best (second best) predictor. In general LMM has high recall but lower precision since it tends to make false jump predictions. SeqGAN does not suffer from this problem.

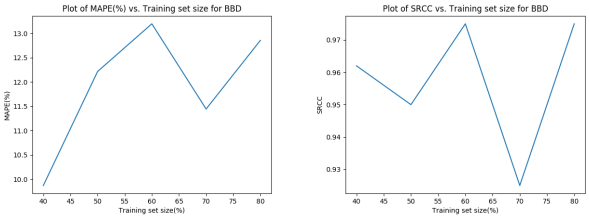


Figure 1: MAPE and SRCC for varying training set size (BBD)

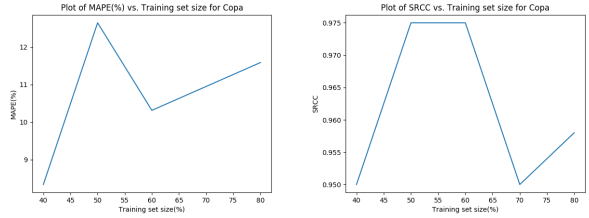


Figure 2: MAPE and SRCC for varying training set size (Copa)

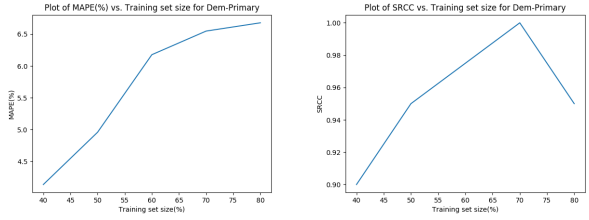


Figure 3: MAPE and SRCC for varying training set size (Dem-Primary)

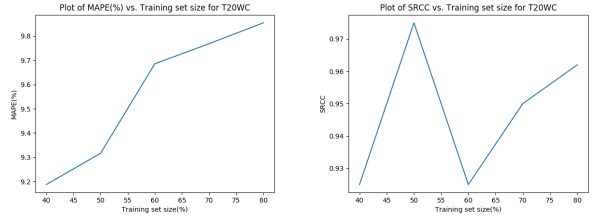


Figure 4: MAPE and SRCC for varying training set size (T20WC)

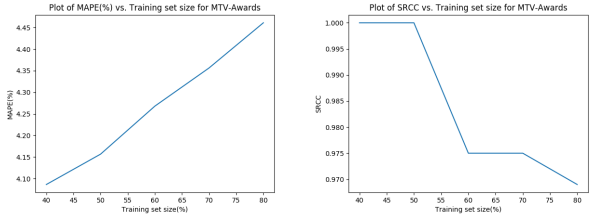


Figure 5: MAPE and SRCC for varying training set size (MTV-Awards)

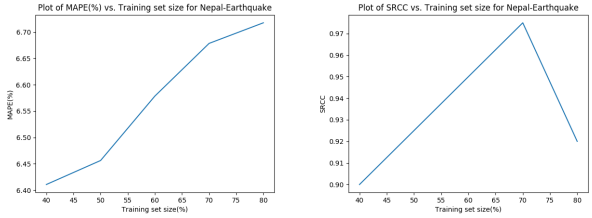


Figure 6: MAPE and SRCC for varying training set size (Nepal-Earthquake)

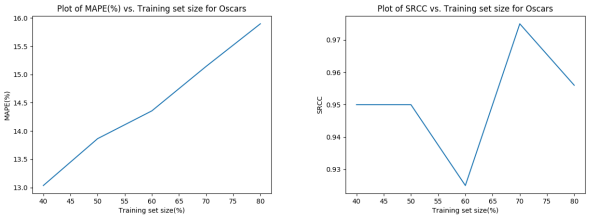


Figure 7: MAPE and SRCC for varying training set size (Oscars)

Conclusion

- We have proposed a linear memoryless neural-network based framework (LMM) to model the hashtag flows as a temporal point process.
- This framework explicitly models hashtag-tweet reinforcement but does not consider inter-hashtag competition.
- We achieve better results than all baselines in almost all datasets. However, LMM is inferior to the SeqGAN approach which we describe next.
- This model can perhaps be improved further by introducing nonlinearity and explicitly modeling the interactions between hashtags.

Problem formulation for SeqGAN model

- In the context of SeqGAN (proposed in (Yu et al. 2017)) we define a new version of the proposed problem as follows:
- **Given data:** For a particular hashtag H we are given a series of timestamps $\{t_{H,j} \mid j = 1, 2, ...f\}$ where f is the total number of observed timestamps. Let (0,T) be the total time window of observation of the temporal point process. Then $0 <= t_{H,1}$ and $t_{H,f} <= T$
- We convert this data to intervalwise agglomerated data. That is, we define a set of intervals $\{I_n \mid n = 1, 2, \ldots, N\}$ where N is the total number of intervals and $I_n = ((n-1) \times T/N, n \times T/N)$.
- We then get a discrete integer sequence $\{X_{H,n} \mid n = 1, 2, \ldots, N\}$ where $X_{H,n} =\mid \{t_{H,j} \mid t_{H,j} \in I_n\} \mid$. This integer sequence is of fixed length N (a hyperparameter) which is the number of time intervals.
- This integer sequence forms the groundtruth data for the SeqGAN model (described below)
- Using this model we tackle the sequence generation problem as detailed in (Yu et al. 2017).
- We train the model using a combination of groundtruth and generated data.
- We model inter-hashtag competition as follows: Let $\mathbb{H}$ be a set of hashtags. We train the discriminator for a given hashtag H$\in \mathbb{H}$ by feeding the groundtruth data for other hashtags H$\prime \in \mathbb{H}$, H$\prime \neq$H along with generated samples as negative examples and only the groundtruth data for the hashtag H as positive examples. This will force the generator for H to avoid generating data similar to other hashtags.
- We evaluate the performance of the system by generating a probable sequence for the test data using only the pretrained generator and comparing the generated sequence with the groundtruth using metrics such as MAPE and SRCC. We also estimate the average precision and recall for jump prediction as done for the linear memoryless model.

SeqGAN overview

As in (Yu et al. 2017) we define the sequence generation problem as follows: Given the groundtruth sequence $\{X_{H,n}\}$, for a hashtag H, we train a θ-parameterized generative model $G_{H,\theta}$ to produce a sequence $\{Y_{H,n} \mid n = 1, 2, \ldots, N\}$ where $Y_{H,n} \in \mathbb{N}$. We interpret this problem based on reinforcement learning. In interval n, the state s is the current produced tokens $(Y_{H,1}, \ldots, Y_{H,n-1}$ and the action a is the next token $Y_{H,n}$ to select.

We also train a ϕ-parameterized discriminative model D_ϕ (Goodfellow et al. 2014) to provide a guidance for improving generator G_θ. $D_\phi(Y_{H,1:N})$ is a probability indicating how likely a sequence $Y_{H,1:N}$ is from real sequence data or not.

The discriminative model D_ϕ is trained by providing positive examples from the real sequence data for a given hashtag H and negative examples from the synthetic sequences generated from the generative model G_θ as well as from the real sequence data for all other hashtags H$\prime \neq$H. At the same time, the generative model G_θ is updated by employing a policy gradient and MC search on the basis of the expected end reward received from the discriminative model D_ϕ . The reward is estimated by the likelihood that it would fool the discriminative model D_ϕ.

SeqGAN via Policy Gradient

Following (Sutton et al. 2000), when there is no intermediate reward, the objective of the generator model (policy) $G_\theta(Y_{H,n}|Y_{H,1:n-1})$ is to generate a sequence from the start state s_0 to maximize its expected end reward:

$$J(\theta) = E\left[R_T \mid s_\theta, \theta\right] = \sum_{n=1}^{N} G_\theta(Y_{H,n} \mid s_0).Q_{D_\phi}^{G_\theta}(s_\theta, Y_{H,n}) \tag{6}$$

where R_T is the reward for a complete sequence and comes from the discriminator D_ϕ. $Q_{D_\phi}^{G_\theta}(s_\theta, a)$ is the action-value function of a sequence, i.e. the expected accumulative reward starting from state s, taking action a, and then following policy G_θ. As per (Yu et al. 2017), we apply Monte Carlo search with a roll-out policy G_β to sample the unknown last N − n counts. We represent a K-time Monte Carlo search as

$$\{Y_{H,1:N}^1, \ldots, Y_{H,1:N}^K\} = MC^{G_\beta}(Y_{H,1:n}; K) \tag{7}$$

Following (Yu et al. 2017) we set G_β the same as the generator. The action-value function is

$$Q_{D_\phi}^{G_\theta}(s = Y_{H,1:n-1}, a = Y_{H,n}) =$$

$$\begin{cases} \frac{1}{K}\sum_{k=1}^{K} D_\phi(Y_{H,1:N}^k), Y_{H,1:N}^k \in MC^{G_\beta}(Y_{H,1:n}; K) & \text{for } n < N \\ D_\phi(Y_{H,1:n}) & \text{for } n = N \end{cases} \quad (8)$$

As in (Yu et al. 2017) we re-train the discriminator as follows:

$$min_\phi - E_{Y\sim p_{data}}\left[logD_\phi(Y)\right] - E_{Y\sim G_\theta}\left[log(1 - D_\phi(Y))\right] \quad (9)$$

We update the generator's parameters as

$$\theta \leftarrow \theta + \alpha_h \nabla_\theta J(\theta) \quad (10)$$

where $\alpha_h \in \mathbb{R}^+$ is the learning rate at the h-th step and $\nabla_\theta J(\theta)$ is estimated as in (Yu et al. 2017). The following algorithm shows the details of our approach:

Algorithm 1 Sequence Generative Adversarial Nets

Require: generator policy G_θ; roll-out policy G_β ; discriminator D_ϕ ; a sequence dataset for a set of hashtags $\mathbb{H}$ given by $S = \{X_{H,1:N}\}$, $H \in \mathbb{H}$; a particular hashtag H under consideration.
Initialize G_θ, D_ϕ with random weights θ, ϕ.
Pre-train G_θ using MLE on S
$\beta \leftarrow \theta$
Generate negative samples using G_θ and groundtruth data for other hashtags $H\prime \in \mathbb{H}$, $H\prime \neq H$ for training D_ϕ
Pre-train D_ϕ via minimizing the cross entropy
repeat
  for g˙steps **do**
    Generate a sequence $Y_{H,1:N} = (Y_{H,1}, \ldots, Y_{H,N}) \sim G_\theta$
    for n in 1:N **do**
      Compute $Q_{D_\phi}^{G_\theta}(s = Y_{H,1:n-1}, a = Y_{H,n})$ by Eq. 8
    end for
    Update generator parameters via policy gradient Eq. 10
  end for
  for d˙steps **do**
    Use current G_θ and groundtruth data for other hashtags $H\prime \in \mathbb{H}$, $H\prime \neq H$ to generate negative examples and combine with given positive examples S
    Train discriminator D_ϕ for k epochs by Eq. 9
  end for
  $\beta \leftarrow \theta$
until SeqGAN converges

Generator

As in (Yu et al. 2017) we use Long Short Term Memory(LSTM) (Hochreiter and Schmidhuber 1997) as the generator model with the update equations:

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f.[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f), \quad (11)$$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i.[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i), \quad (12)$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o.[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o), \quad (13)$$

$$\boldsymbol{s}_t = \boldsymbol{f}_t \odot s_{t-1} + i_t \odot tanh(W_s.[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_s), \quad (14)$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot tanh(\boldsymbol{s}_t), \quad (15)$$

where $[h, x]$ is the vector concatenation and $\odot$ is the element-wise product. For training the RNN we use the maximum likelihood estimation (MLE). The loss function for the generator is the negative log-likelihood $-\sum_{n=1}^{N} logG_\theta(Y_{H,n} = X_{H,n} \mid X_{H,1:N})$ for a generated sequence $Y_{H,1:N}$ given an input sequence $X_{H,1:N}$ for a hashtag H.

Discriminator

As in (Yu et al. 2017) we choose the convolutional neural network (CNN) as the discriminator. We first represent the input sequence $X_{H,1:N}$ as

$$\boldsymbol{\varepsilon}_{1:N} = \boldsymbol{x}_1 \oplus \boldsymbol{x}_2 \oplus \ldots \oplus \boldsymbol{x}_N, \quad (16)$$

where $\boldsymbol{x}_n \in \mathbb{R}^k$ is the k-dimensional token embedding and $\oplus$ is the vertical concatenation operator to build the matrix $\boldsymbol{\varepsilon}_{1:N} \in \mathbb{R}^{T\times k}$. Then a kernel $\boldsymbol{w} \in \mathbb{R}^{l\times k}$ applies a convolutional operation to a window size of l words to produce a new feature map

$$c_i = \rho(\boldsymbol{w} \otimes \boldsymbol{\varepsilon}_{i:i+l-1} + b), \quad (17)$$

where $\oplus$ operator is the summation of element-wise production, b is a bias term and ρ is a non-linear function. A kernel $\boldsymbol{w}$ with window size l applied to the concatenated embeddings of input sequence will produce a feature map

$$\boldsymbol{c} = [c_1, \ldots, c_{T-l+1}]. \quad (18)$$

Finally we apply a max-over-time pooling operation over the feature map $\tilde{c} = max\{c\}$ and pass all pooled features from different kernels to a fully connected softmax layer to get the probability that a given sequence is real. To enhance the performance, we also add the highway architecture (Srivastava, Greff, and Schmidhuber 2015) before the final fully connected layer:

$$\boldsymbol{\tau} = \sigma(\boldsymbol{W}_T.\tilde{\boldsymbol{c}} + \boldsymbol{b}_T), \quad (19)$$

$$\tilde{\boldsymbol{C}} = \boldsymbol{\tau}.H(\tilde{\boldsymbol{c}}, \boldsymbol{W}_H) + (\boldsymbol{1} - \boldsymbol{\tau}).\tilde{\boldsymbol{c}}, \quad (20)$$

$$(21)$$

where $\boldsymbol{W}_T$, $\boldsymbol{b}_T$ and $\boldsymbol{W}_H$ are highway layer weights, H denotes an affine transform followed by a non-linear activation function such as a rectified linear unit (ReLU) and $\boldsymbol{\tau}$ is the "transform gate" with the same dimensionality as $H(\tilde{\boldsymbol{c}}, \boldsymbol{W}_H)$ and $\tilde{\boldsymbol{c}}$. Finally, we apply a sigmoid transformation to get the probability that a given sequence is real:

$$\hat{y} = \sigma(\boldsymbol{W}_o.\tilde{\boldsymbol{C}} + b_o) \quad (22)$$

where $\boldsymbol{W}_o$ and b_o are the output layer weight and bias.

When optimizing discriminative models, supervised training is applied to minimize the cross entropy, which is widely

used as the objective function for classification and prediction tasks:

$$\mathbb{L}(y, \hat{y}) = -ylog\hat{y} - (1-y)log(1-\hat{y}) \qquad (23)$$

where y is the ground truth label of the input sequence and $\hat{y}$ is the predicted probability from the discriminative models.

Results

We have tested the new model (SeqGAN) on the same dataset used to evaluate the Linear Memoryless Model (LMM). As in case of LMM we use the metrics MAPE and SRCC which were defined when we described the experiments on LMM. As before, we also record the average precision and recall for jump detection.

Performance comparison (MAPE)

Table 1 presents a comparative sketch for SeqGAN in terms of MAPE on the 20% held-out set. We observe that for all datasets, SeqGAN performs best by achieving lowest MAPE compared to all other baselines as well as LMM.

Performance comparison (SRCC)

Table 1 depicts a comparative analysis in terms of SRCC for all the proposals. We observe that SeqGAN performs significantly better than the baselines as well as LMM across all datasets.

Detection of Sudden Changes in Ranking

Table 2 reports the average precision and recall of jump detection for algorithms across all the datasets. Here we see that SeqGAN is superior to LMM as well as other baselines in terms of precision and recall in almost all datasets. The only exception is the T20WC dataset, where LMPP outperforms SeqGAN in terms of precision. However in this case the precision achieved by SeqGAN (0.8) is fairly good and hence we can conclude that unlike LMM, SeqGAN does not suffer very much from the problem of false predictions.

Conclusion

- We have proposed SeqGAN (described in (Yu et al. 2017) as a solution to the problem of hashtag popularity prediction.
- Unlike LMM, SeqGAN explicitly models competition between hashtags. However it does not explicitly model hashtag-tweet reinforcement.
- This model is superior to LMM as well as other baselines in almost all cases.

References

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, 2672–2680.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Samanta, B.; De, A.; Chakraborty, A.; and Ganguly, N. Lmpp: A large margin point process combining reinforcement and competition for modeling hashtag popularity.

Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, 1057–1063.

Yu, L.; Zhang, W.; Wang, J.; and Yu, Y. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, 2852–2858.