Recurrent tweets

Project presentation

Mathias Berglund, Petri Kyröläinen, Yu Shen

December 9, 2013

Agenda

Project background – tweet sentiment classification

Model part 1: Learning word representations

Model part 2: Classification with recurrent neural network

Goal to classify whether tweet has positive, negative or neutral sentiment

Goal to create model for classifying tweets

- Tweets can have multiple sentiments
- Model is to classify tweets into positive, neutral or negative sentiment

“Pretty Little Liars was the shit ! I can't wait til tomorrow ! I wanna see who all innocent & who got something to do with Allison dying !”

Positive tweet

“@Duffy_Louise Nooooooo this Sunday is the last episode of Downton Abbey . :(There's a Christmas special coming but that's AGES away .”

Negative tweet

“Manchester United will try to return to winning ways when they face Arsenal in the Premier League at Old Trafford on Saturday .”

Neutral tweet

We use data from ”SemEval-2013: Sentiment Analysis in Twitter” with annotated tweets

SemEval-2013 a sentiment analysis challenge in summer 2013

SemEval-2013 had multiple challenges
- Multiple challenges in SemEval-2013
- Data included Tweets and text messages
- Tasks included message and word classification

SemEval-2013 workshop co-located with NAACL
- Organized with “The 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies” in summer of 2013

Twitter data includes 7 485 annotated tweets

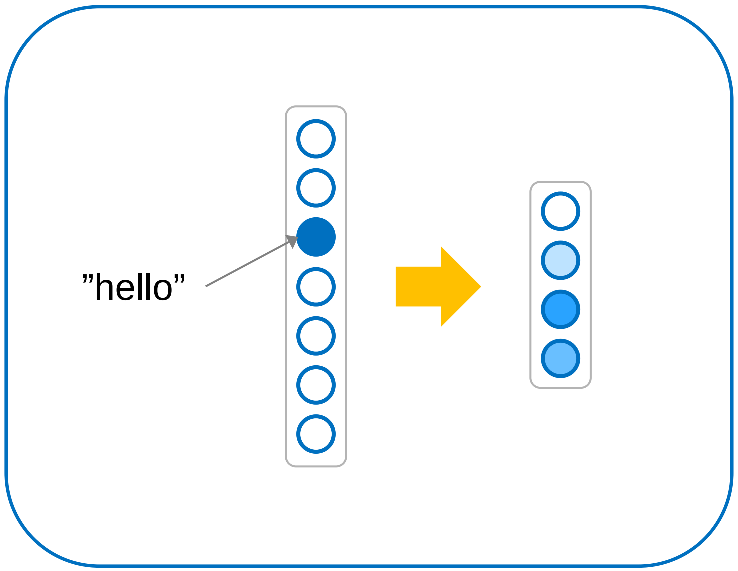
Twitter data set includes 7 463 annotated tweets
- Hand annotated tweets classified into “positive”, “negative”, “objective” or “neutral”
- Split into 6 448 training set and 1 017 “development set” (used as test set)

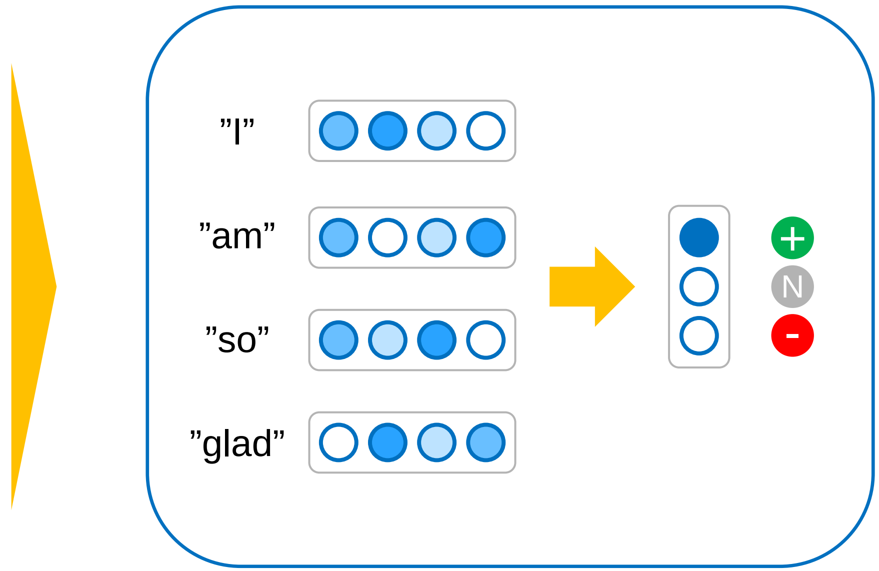
Vocabulary size 23 123

Our approach: two-stage training where semantics of tokens learned in first part, and second part used for classification

<u>Part 1</u>: Map each token (word) to a continuous-valued vector with semantic meaning

<u>Part 2</u>: Map a stream of words (i.e. a tweet) into sentiment

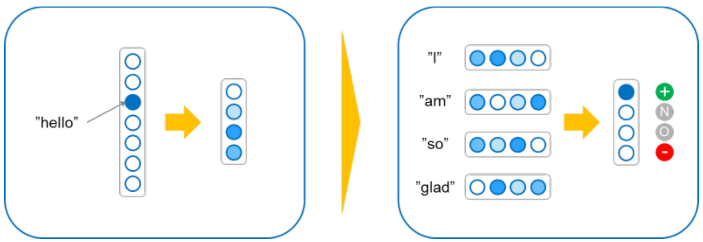




Motivation: Unsupervised training for first part enables use of unlabeled data

Results: We get an average F1-score of 42, which is still behind state-of-the-art 69 with handcrafted features

Our approach reached an F1-score of 41.75



F1: 41.75

Training first step with large data set still under development

State-of-the-art reached an F1-score of 69.02

NRC-Canada SVM

A multitude of hand-crafted features used in conjunction with SVM classification

F1: 69.02

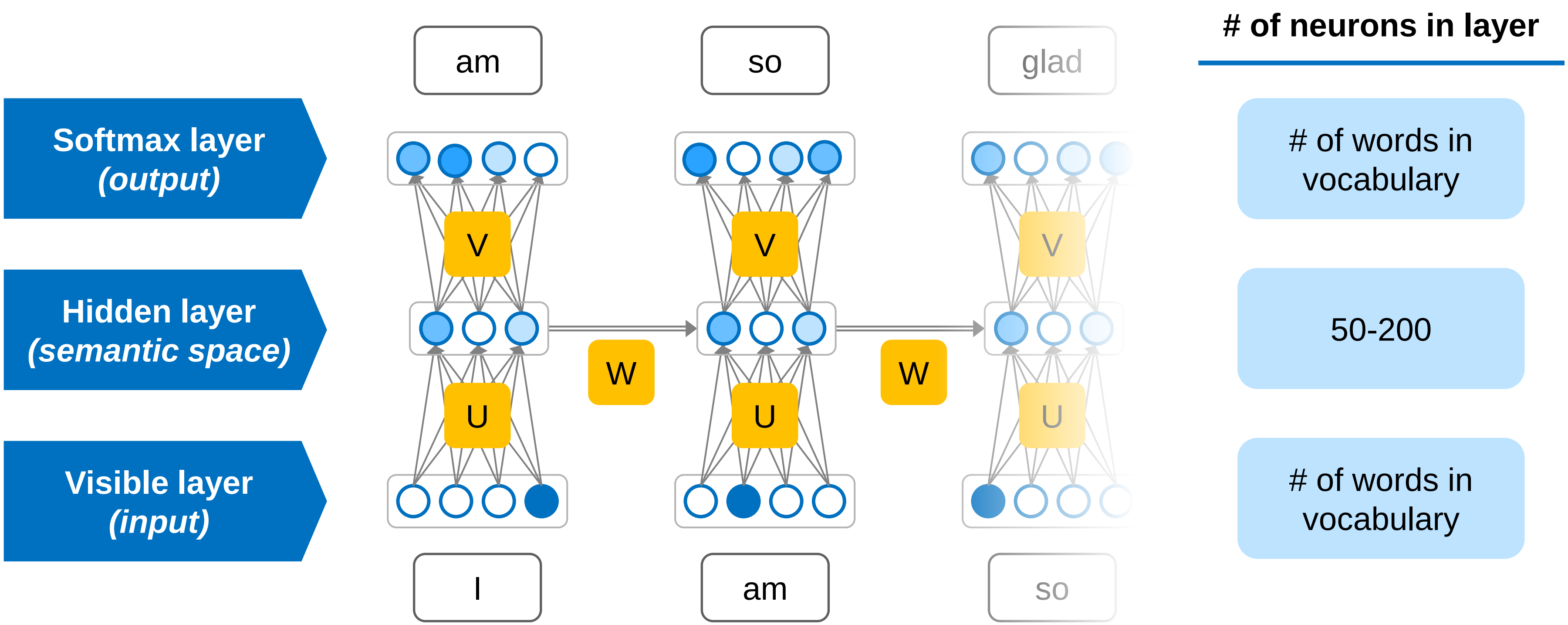
Agenda

Project background – tweet sentiment classification

Model part 1: Learning word representations

Model part 2: Classification with recurrent neural network

Mikolov’s Recurrent Neural Network Language Model



Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig. "Linguistic regularities in continuous space word representations." *Proceedings of NAACL-HLT*. 2013.
https://www.aclweb.org/anthology/N/N13/N13-1090.pdf

Recurrent neural networks in mathematical notation

Computation of input, hidden and output layer activations (forward pass)

$$\mathbf{x}(t) = [\mathbf{w}(t)^T \mathbf{s}(t-1)^T]^T$$

$$s_j(t) = f\left(\sum_i x_i(t) u_{ji}\right)$$

$$y_k(t) = g\left(\sum_j s_j(t) v_{kj}\right)$$

$$f(z) = \frac{1}{1+e^{-z}}, \quad g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$

Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig. "Linguistic regularities in continuous space word representations." *Proceedings of NAACL-HLT*. 2013.
https://www.aclweb.org/anthology/N/N13/N13-1090.pdf

Weight updates in Backprogation Through Time algorithm

Weight updates (backward pass)

$$\mathbf{V}(t{+}1) = \mathbf{V}(t) + \mathbf{s}(t)\mathbf{e}_o(t)^T\alpha - \mathbf{V}(t)\beta$$

$$\mathbf{U}(t{+}1) = \mathbf{U}(t) + \sum_{z=0}^{T}\mathbf{w}(t{-}z)\mathbf{e}_h(t{-}z)^T\alpha - \mathbf{U}(t)\beta$$

$$\mathbf{W}(t{+}1) = \mathbf{W}(t) + \sum_{z=0}^{T}\mathbf{s}(t{-}z{-}1)\mathbf{e}_h(t{-}z)^T\alpha - \mathbf{W}(t)\beta$$

α = learning rate
β = regularization parameter

Error functions

$$\mathbf{e}_o(t) = \mathbf{d}(t) - \mathbf{y}(t)$$

$$\mathbf{e}_h(t{-}\tau{-}1) = d_h\left(\mathbf{e}_h(t{-}\tau)^T\mathbf{W}, t{-}\tau{-}1\right)$$

$$d_{hj}(x,t) = x s_j(t)(1 - s_j(t))$$

Mikolov, Tomas, "Statistical Language Models Based on Neural Networks", PHD thesis. Brno University of Technology, 2012.

Time complexity of RNNLM is fairly high

$$O = E \times T \times [(H + 1) \times H \times \tau + H \times V]$$

E = epochs

T = tokens (or minipatches) in training set

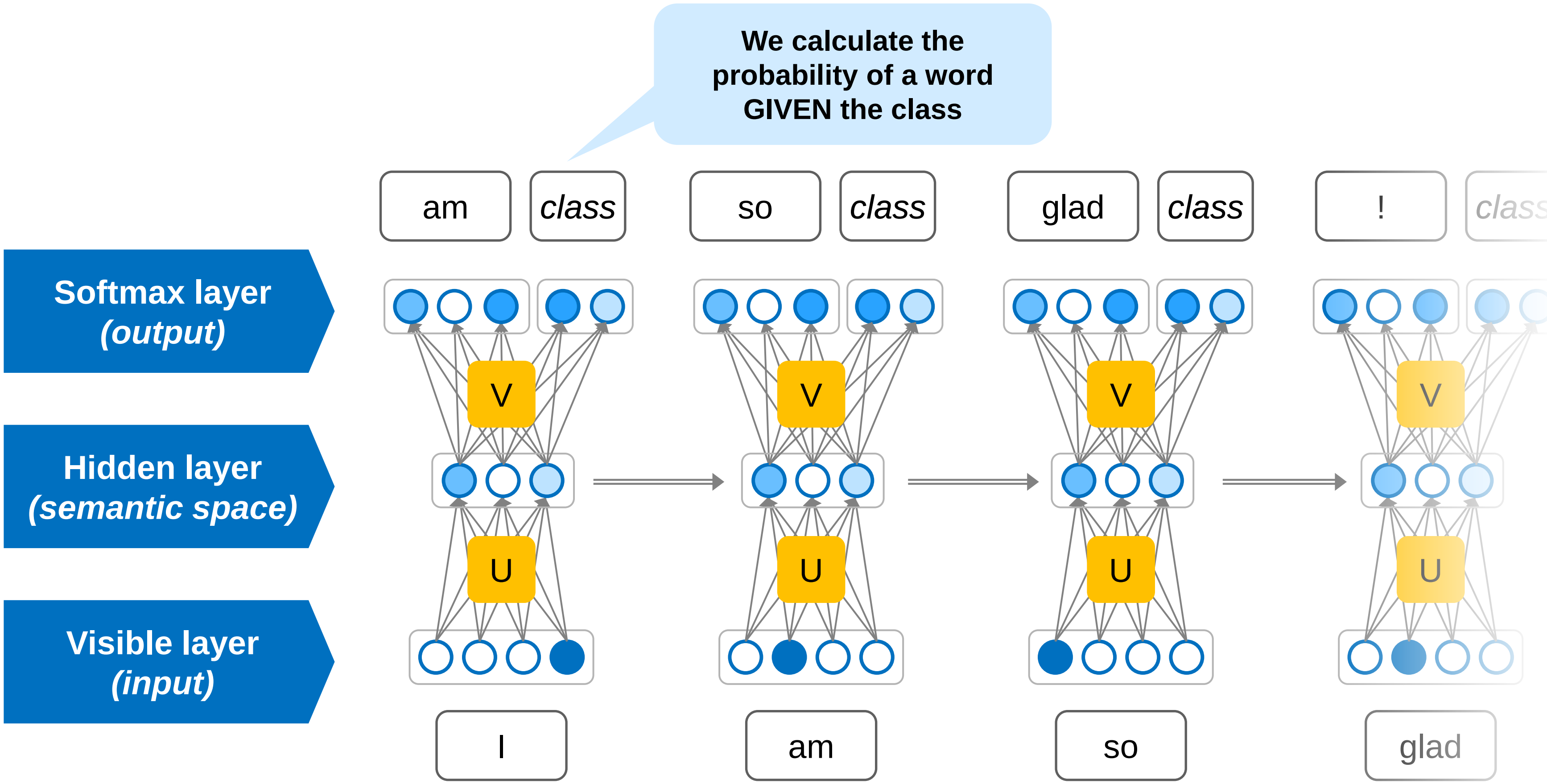
H = hidden neurons

V = size of vocabulary

τ = time steps in back propagation through time algorithm

T. Mikolov et al., "Extensions of recurrent neural network language model", In Proceedings of the 2011 IEEE ICASSP, Prague

RNN language model with output layer factorized by classes



T. Mikolov et al., "Extensions of recurrent neural network language model", In Proceedings of the 2011 IEEE ICASSP, Prague

RNN language model with classes in mathematical notation

Conditional probability of word can be factorized

$$P(w_i|history) = P(c_i|\mathbf{s}(t))P(w_i|c_i, \mathbf{s}(t))$$

The two factors are computed as

$$c_l(t) = g\left(\sum_j s_j(t) w_{lj}\right)$$

$$y_c(t) = g\left(\sum_j s_j(t) v_{cj}\right)$$

T. Mikolov et al., "Extensions of recurrent neural network language model", In Proceedings of the 2011 IEEE ICASSP, Prague

Classes reduce time complexity of RNNLM considerably

Standard

$$O = E \times T \times [(H+1) \times H \times \tau + H \times V]$$

Factorized by class

$$O = E \times T \times [(H+1) \times H \times \tau + H \times (C + V_C)]$$

H = hidden neurons

V = size of vocabulary

C = classes

V_C = expected number of word types in the class

T. Mikolov et al., "Extensions of recurrent neural network language model", In Proceedings of the 2011 IEEE ICASSP, Prague

Preliminary results using RNN language model

Hidden layer size	Classes	Log prob of valid data	Validation perplexity	Epochs
100	500	-55118.9	488.0	6
100	100	-53708.3	416.5	13
100	50	-53731.1	417.6	13
50	50	-53827.0	422.1	13
200	**50**	**-53644.3**	**413.6**	**12**

Perplexity = average branching factor

Software: Mikolov, Tomas, Kombrink Stefan, Deoras Anoop, Burget Lukas, and Cernocky Jan. "RNNLM – Recurrent Neural Network Language Modeling Kit" *In: ASRU 2011 Demo Session.*

Learning reduces perplexity a lot in the word prediction task

RNNLM with hidden layer size of 200 and 50 classes

Iteration	Valid. Perplexity
0	**729.2**
1	631.9
2	576.5
3	550.6
4	540.9
5	516.5
6	492.4
7	473.8
8	456.3
9	440.7
10	429.7
11	419.7
12	**413.6**

Software: Mikolov, Tomas, Kombrink Stefan, Deoras Anoop, Burget Lukas, and Cernocky Jan. "RNNLM – Recurrent Neural Network Language Modeling Kit" *In: ASRU 2011 Demo Session.*

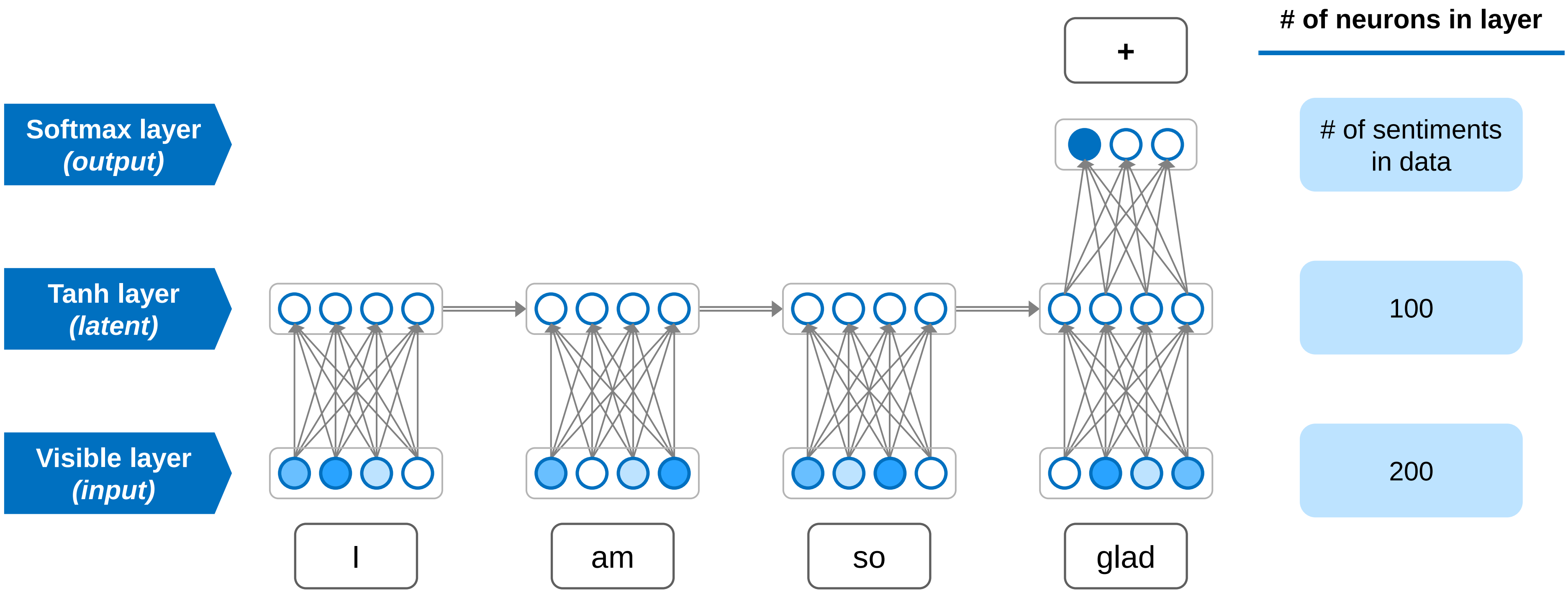
Agenda

Project background – tweet sentiment classification

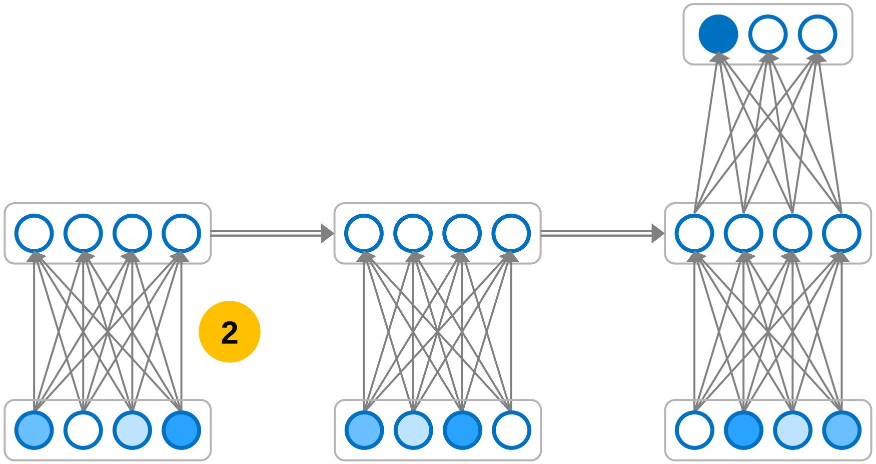
Model part 1: Learning word representations

Model part 2: Classification with recurrent neural network

In step 2, we train recurrent neural network in supervised fashion using only the labeled tweets



We train the network using stochastic gradient descent and Nesterov-type momentum



Training details

1 BPTT with Nesterov momentum

- Network trained with stochastic gradient descent
- Learning rate set with ADADELTA
- Nesterov-type momentum used with mom=0.99

2 Weights pretrained as a language model

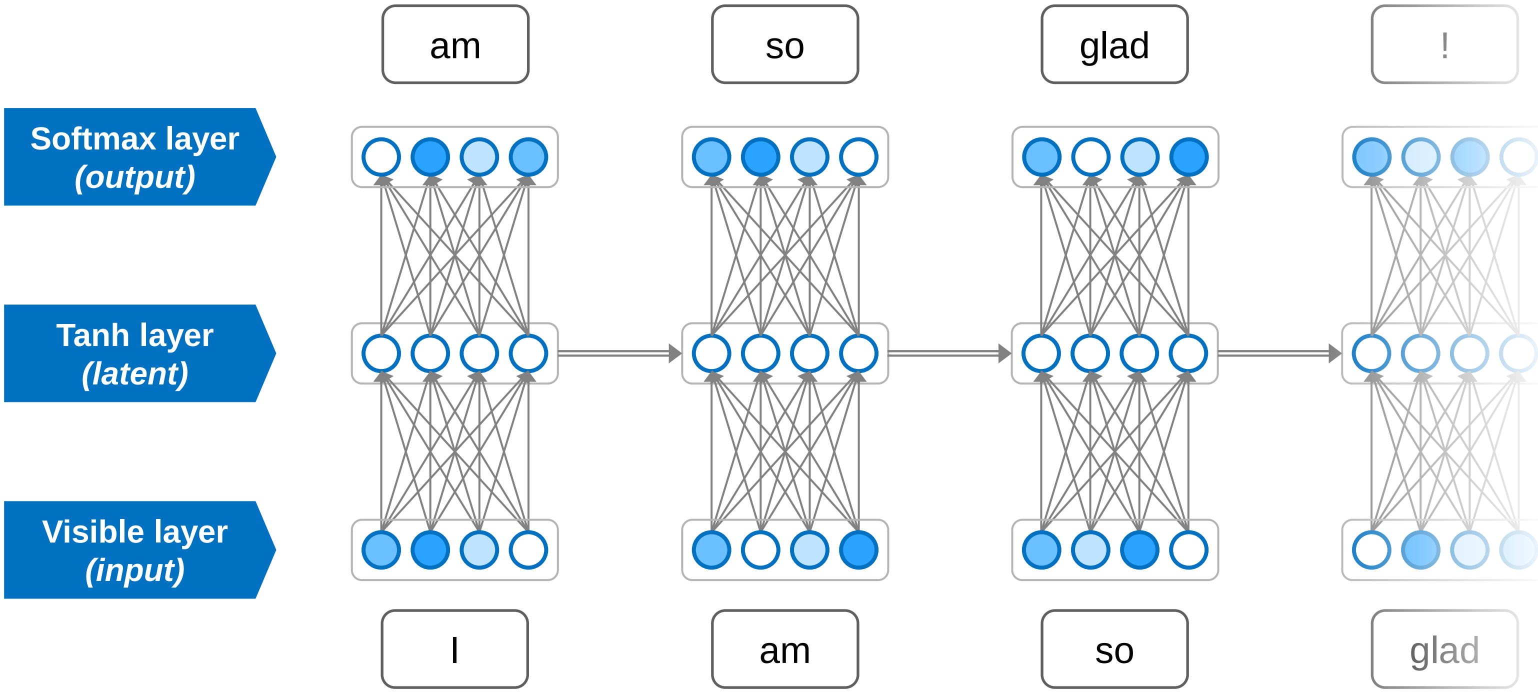
- Pretraining as a regularization tool
- Weights initialized by predicting the next word in semantic spae

3 Training for max 100 epochs in minibatches

- Training in minibatches of 10
- Training done for 100 epochs or until early-stop criterion with error on 20% validation set rising

2

Regularization by language model that predicts the next word, but this time in the semantic space provided by step 1



1

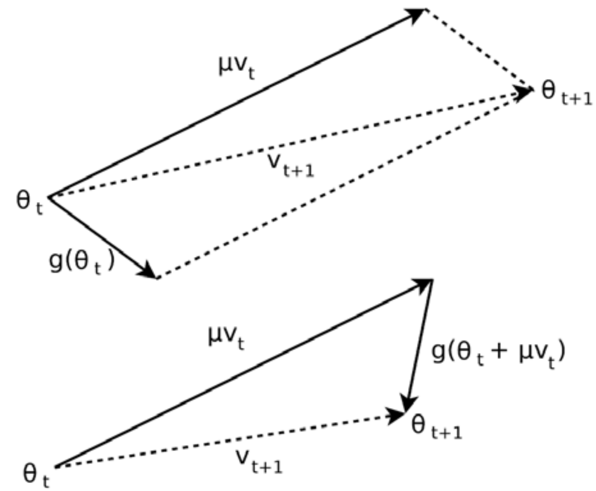
Nesterov-type momentum was shown to improve very deep network learning considerably

Traditional momentum is "slow" to react

$$v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

Nesterov-type momentum reacts faster to gradient change

$$v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t + \boldsymbol{\mu v_t})$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$



Sutskever, Ilya, et al. "On the importance of initialization and momentum in deep learning."

When evaluating the test set, we insert a tweet, and classify it based on the largest value in the output layer