Autoencoders & Attention. Applications

Sourangshu Bhattacharya

Autoencoders

Autoencoders

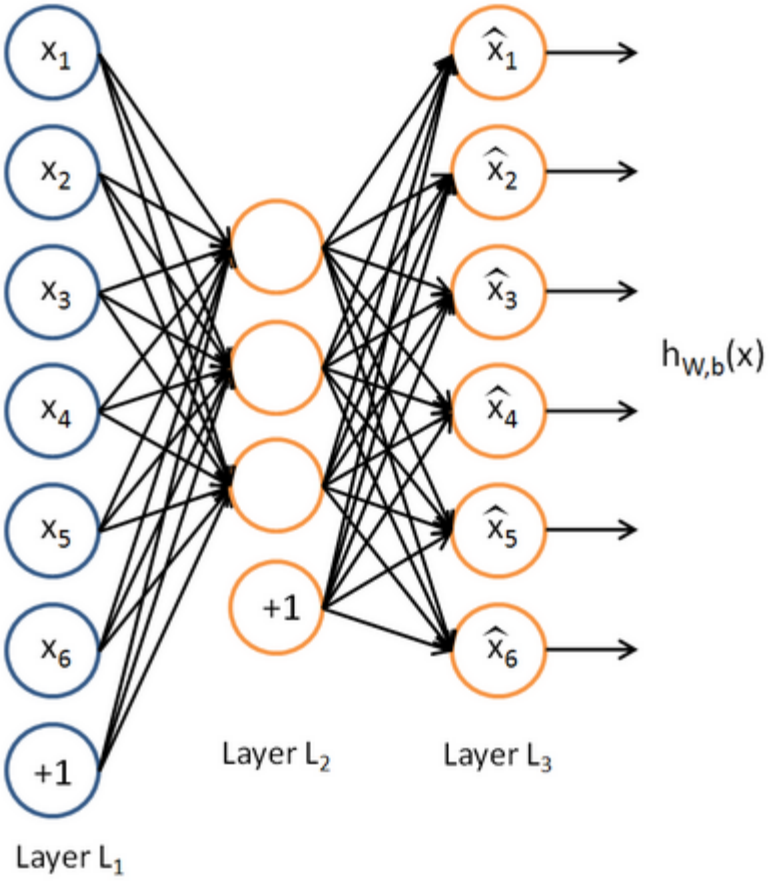
- Unsupervised Learning Algorithm
- Given an input *x*, we learn a *compressed* representation of the input, which we then try to reconstruct
- In the simpliest form: Feed forward network with hidden size < input size.
- We then search for parameters such that:

$$\hat{x} \approx x$$

  for all training examples
- The error function is:

$$E(x, W, b) = ||\hat{x} - x||_2$$

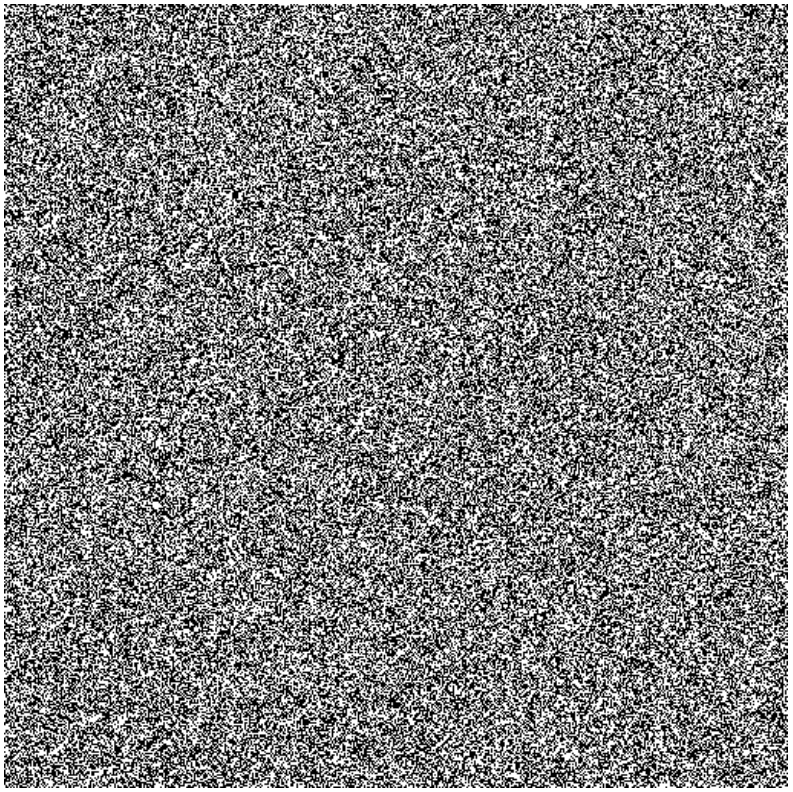
- Once we finished training, we are interested in the compressed representation, i.e. the values of the hidden units



Source: http://ufldl.stanford.edu/wiki/index.php/Autoencoders_and_Sparsity

Why would we use autoencoders?

- How does a randomly generated image look like?



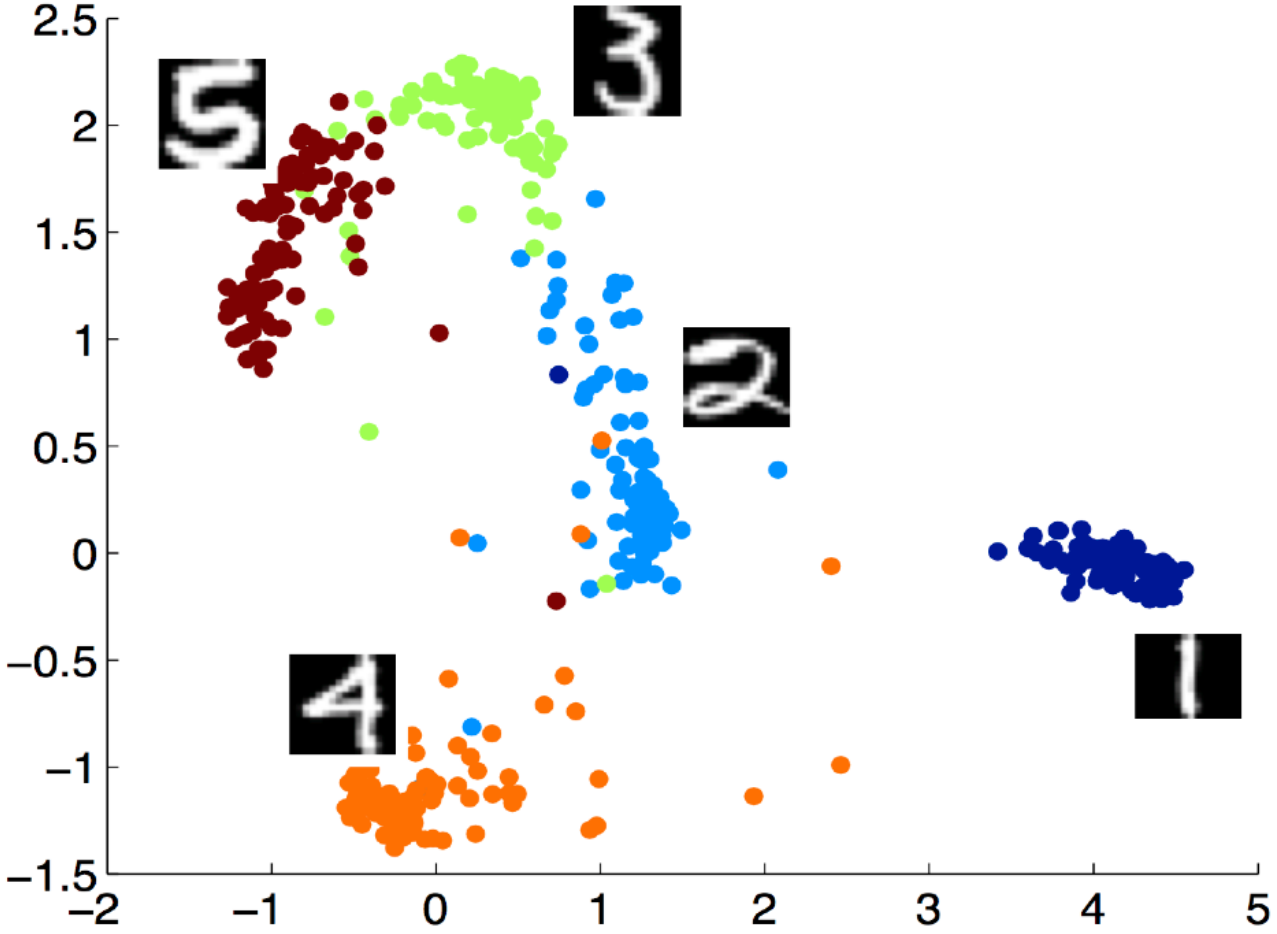
Why would we use autoencoders?

- What would be the probability to get an image like this from random sampling?

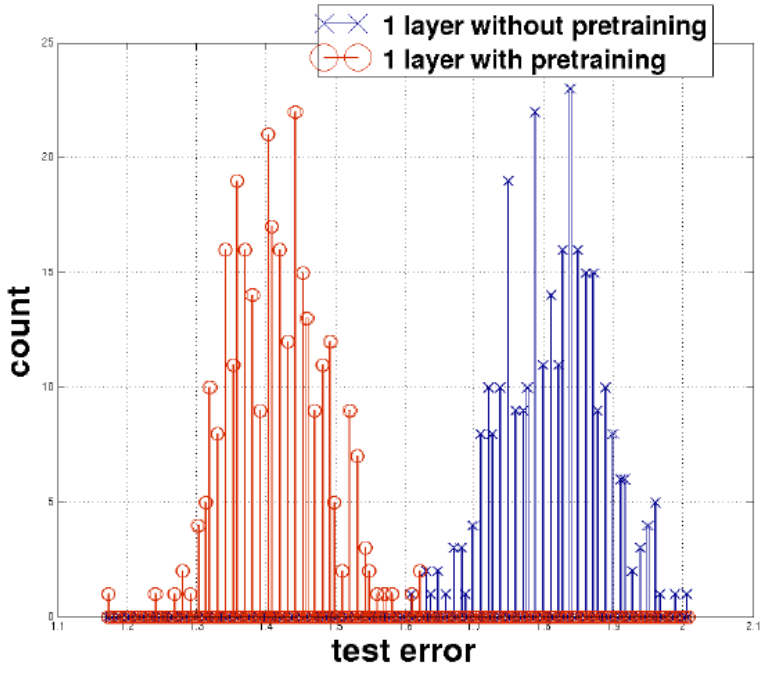
Why would we use autoencoders?

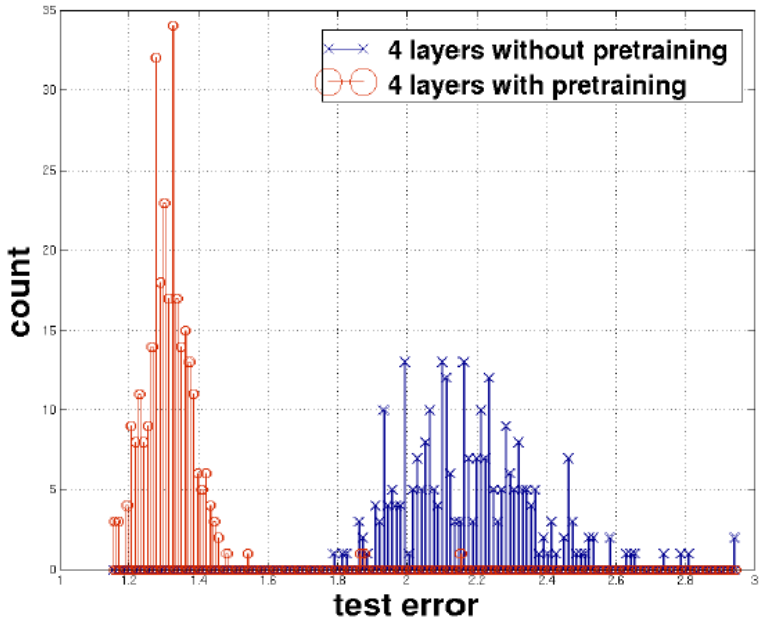
- Produce a compressed representation of a high-dimensional input (for example images)
- The compression is lossy. Learning drives the encoder to be a good compression in particular for training examples
- For random input, the reconstruction error will be high
- The autoencoder learns to abstract properties from the input. What defines a natural image? Color gradients, straight lines, edges etc.
- The abstract representation of the input can make a further classification task much easier

Dimension-Reduction can simplify classifcation tasks – MNIST Task



Dimension-Reduction can simplify classifcation tasks – MNIST Task





- Histogram-plot of test error on the MNIST hand written digit recognition.
- Comparison of neural network with and without pretraining

Source: Erhan et al, 2010, Why Does Unsupervised Pre-training Help Deep Learning?

Autoencoders vs. PCA

- Principle component analysis (PCA) converts a set of correlated variables to a set of linearly uncorrelated variables called *principle components*
- PCA is a standard method to break down high-dimensional vector spaces, e.g. for information extraction or visualization
- However, PCA can only capture linear correlations

PCA

Encoder:

$$f_\theta(x) = Wx$$

Decoder:

$$g_\theta(y) = W'y$$

Autoencoders

Encoder:

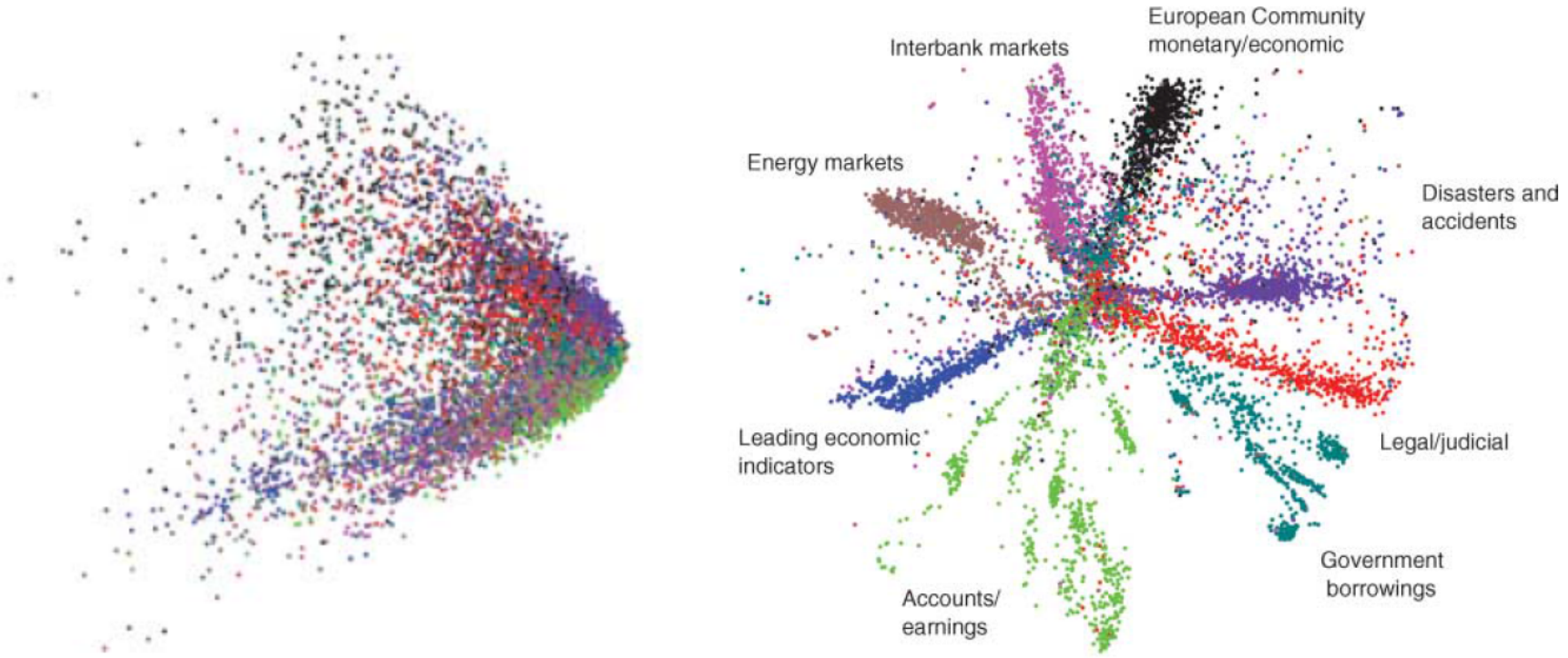
$$f_\theta(\mathbf{x}) = s(\mathbf{W}\mathbf{x} + \mathbf{b}).$$

Decoder:

$$g_{\theta'}(\mathbf{y}) = s(\mathbf{W}'\mathbf{y} + \mathbf{b}'),$$

Autoencoders vs. PCA - Example

- Articles from Reuter corpus were mapped to a 2000 dimensional vector, using the 2000 most common word stems



Deep Autoencoder

Source: Hinton et al., Reducing the Dimensionality of Data with Neural Networks

How to ensure the encodes does *not* learn the identity function?

Identify Function

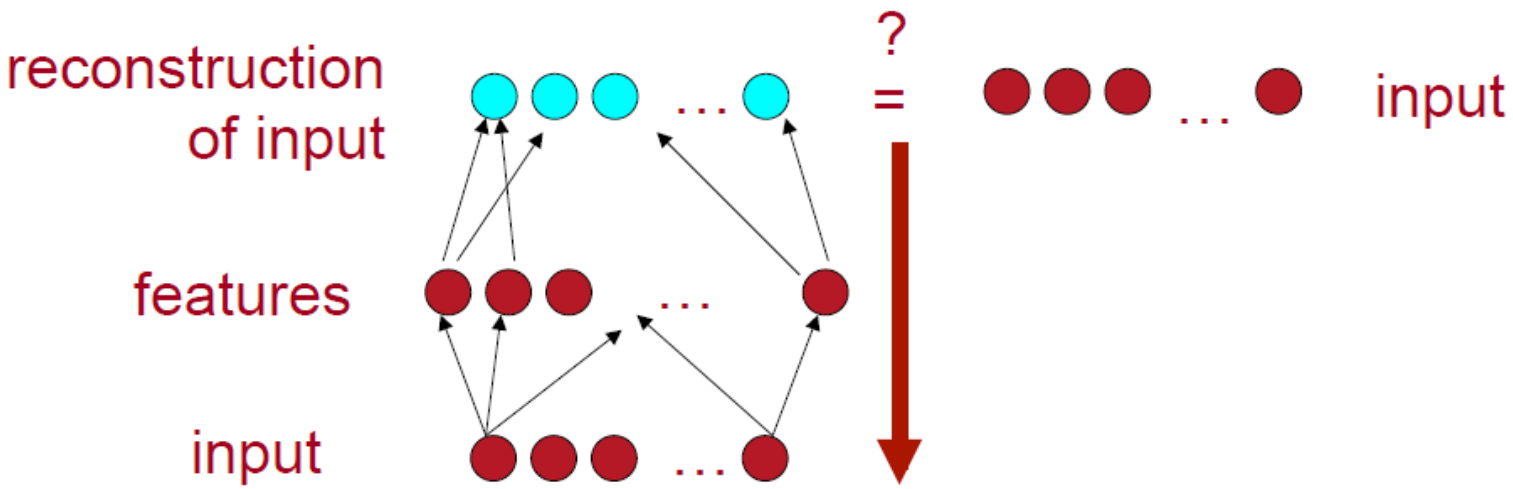
- Learning the identity function would not be helpful
- Different approaches to ensure this:
  - Bottleneck constraint: The hidden layer is (much) smaller than the input layer
  - Sparse coding: Forcing many hidden units to be zero or near zero
  - Denoising encoder: Add randomness to the input and/or the hidden values

Denoising Encoder

- Create some random noise ε
- Compute $\hat{x} = f(x + \varepsilon)$
- Reconstruction Error: $\hat{x} \approx x?$
- Alternatively: Set some of the neurons (e.g. 50%) to zero
- The noise forces the hidden layer to learn more robust features

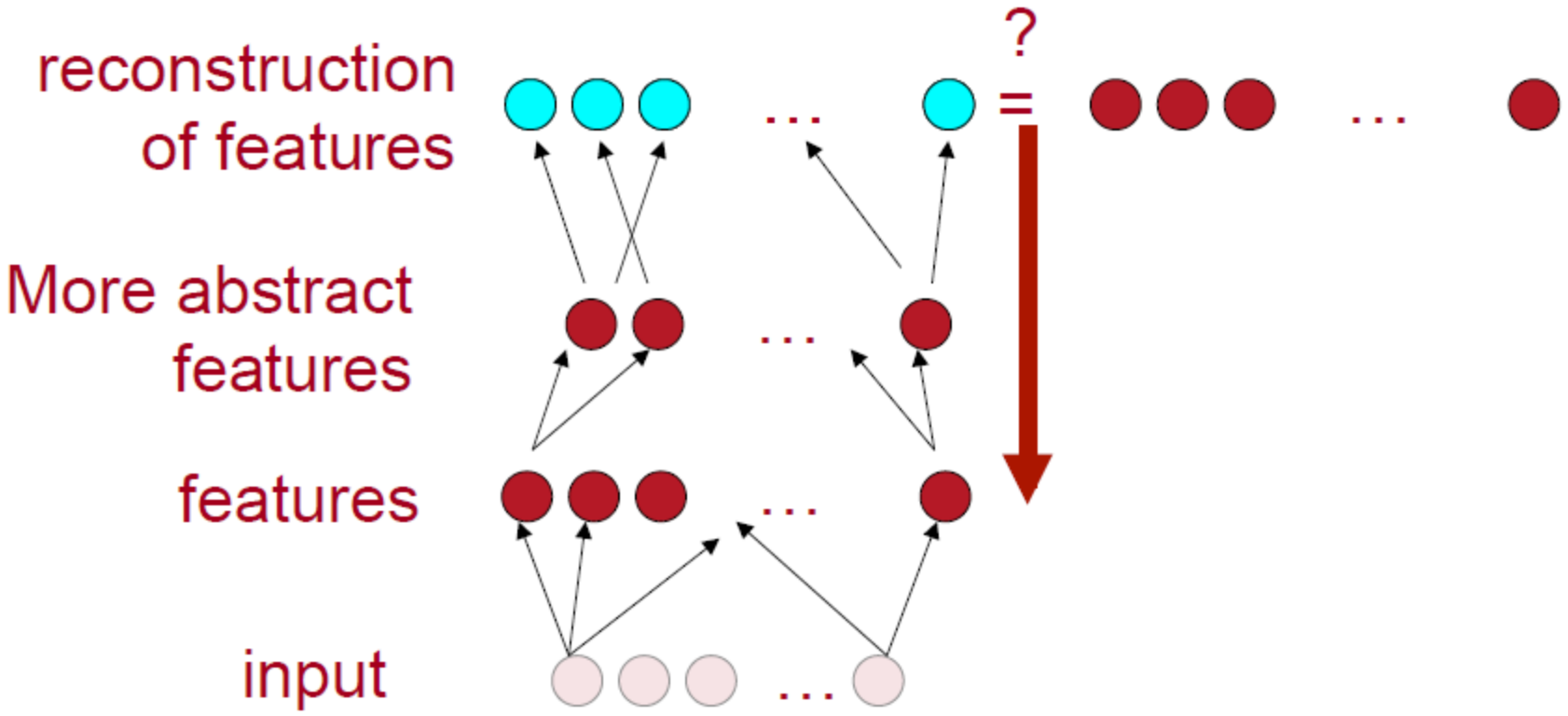
Stacking Autoencoders

- We can stack multiple hidden layers to create a deep autoencoder
- These are especially suitable for highly non-linear tasks
- The layers are trained layer-wise – one at a time



Step 1: Train single layer autoencoder until convergence

Stacking Autoencoders



Step 2: Add additional hidden layer and train this layer by trying to reconstruct the output of the previous hidden layer. Previous layers are will not be changed. Error function: $||\hat{h}_1 - h_1||_2$.

Stacking Autoencoders – Fine-tuning

- After pretraining all hidden layers, the deep autoencoder is fine-tuned

Unsupervised Fine-Tuning:

- Apply back propagation to the complete deep autoencoder
- Error-Function:

$$E(x, W^{(1)}, W^{(2)}, ...) = ||\hat{x} - x||_2$$

- Further details, see Hinton et al.
- *(It appears that supervised fine-tuning is more common nowadays)*

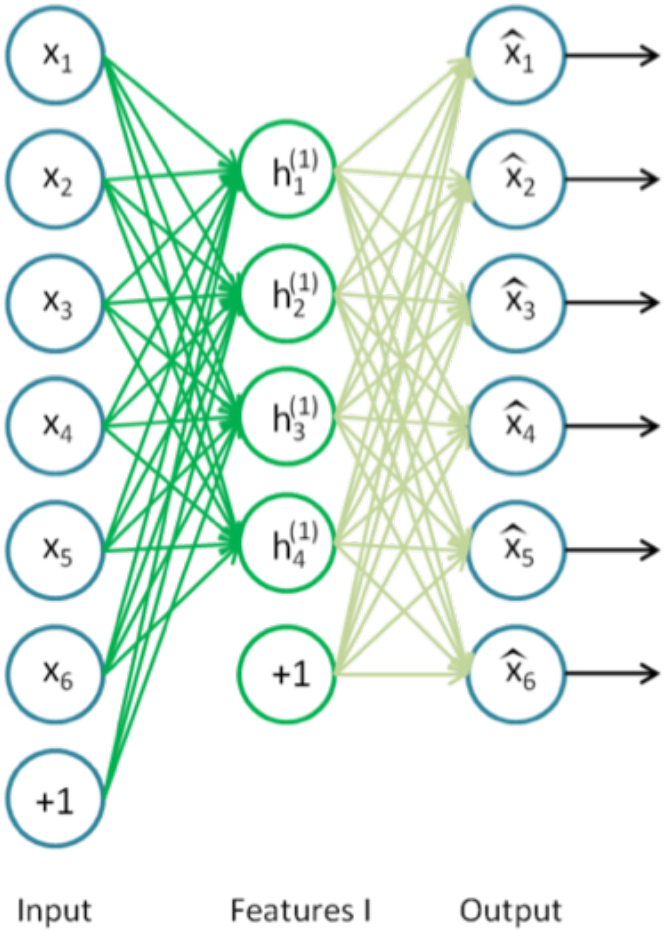
Supervised Fine-Tuning:

- Use your classification task to fine-tune your autoencoders
- A softmax-layer is added after the last hidden layer
- Weights are tuned by using back prograpagtion.
- See next slides for an example or http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

Stacking Autoencoders - Example

Pretrain first autoencoder

- Train an autoencoder to get the first weight matrix $W^{(1)}$ and first bias vector $b^{(1)}$
- The second weight matrix, connecting the hidden and the output units, will be disregarded after the first pretraining step
- Stop after a certain number of iterations

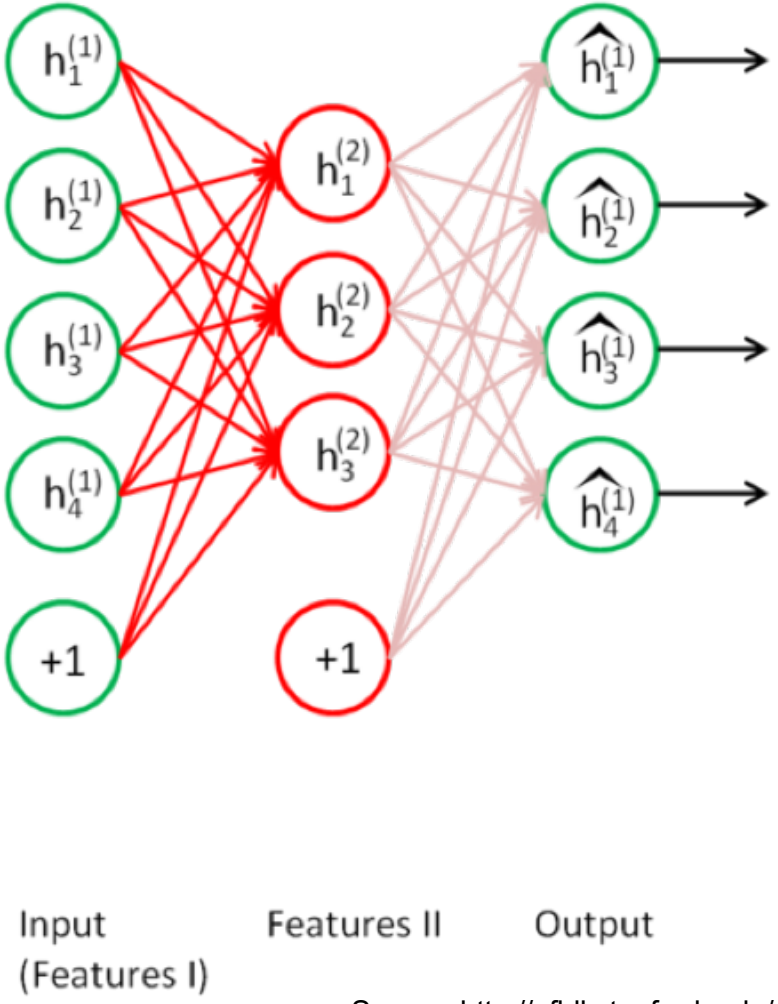


Source: http://ufldl.stanford. edu/wiki/

Stacking Autoencoders - Example

Pretrain second autoencoder

- Use the values of the previous hidden units as input for the next autoencoder.
- Train as before

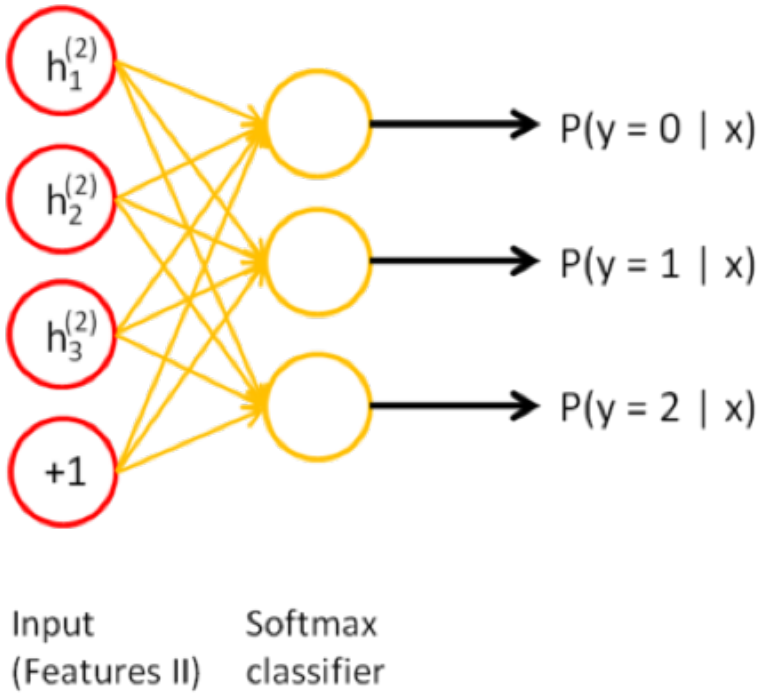


Source: http://ufldl.stanford. edu/wiki/

Stacking Autoencoders - Example

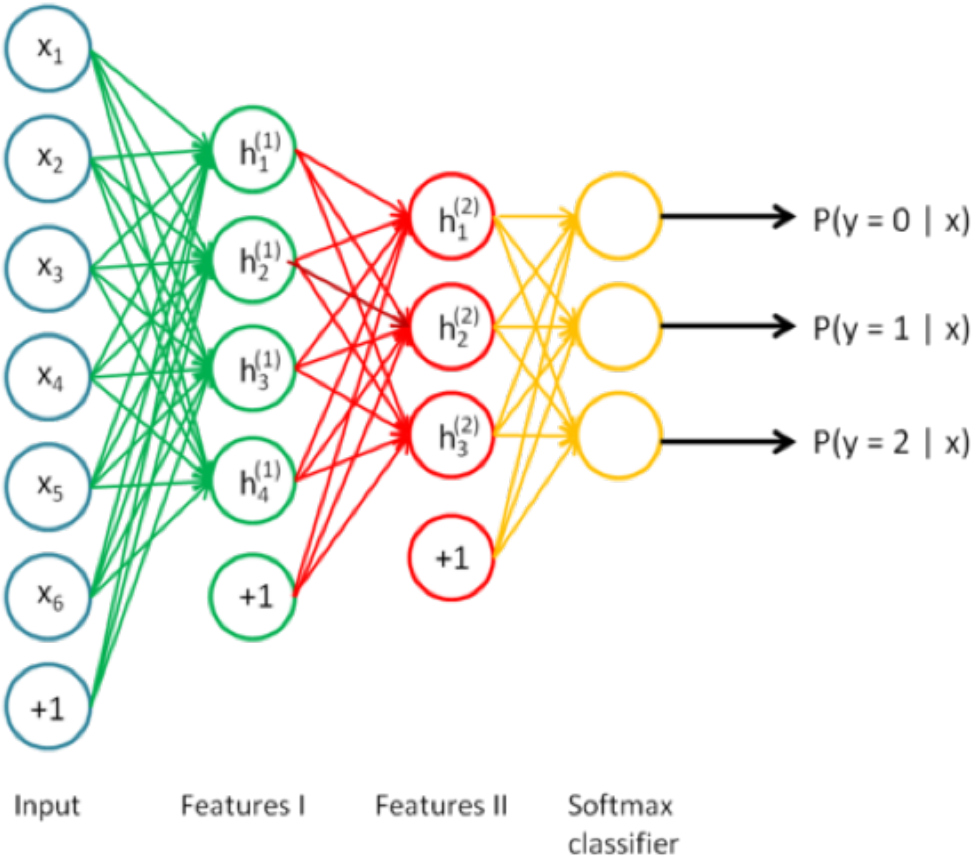
Pretrain softmax layer

- After second pretraining finishes, add a softmax layer for your classification task
- Pretrain this layer using back propagation



Source: http://ufldl.stanford. edu/wiki/

Stacking Autoencoders - Example



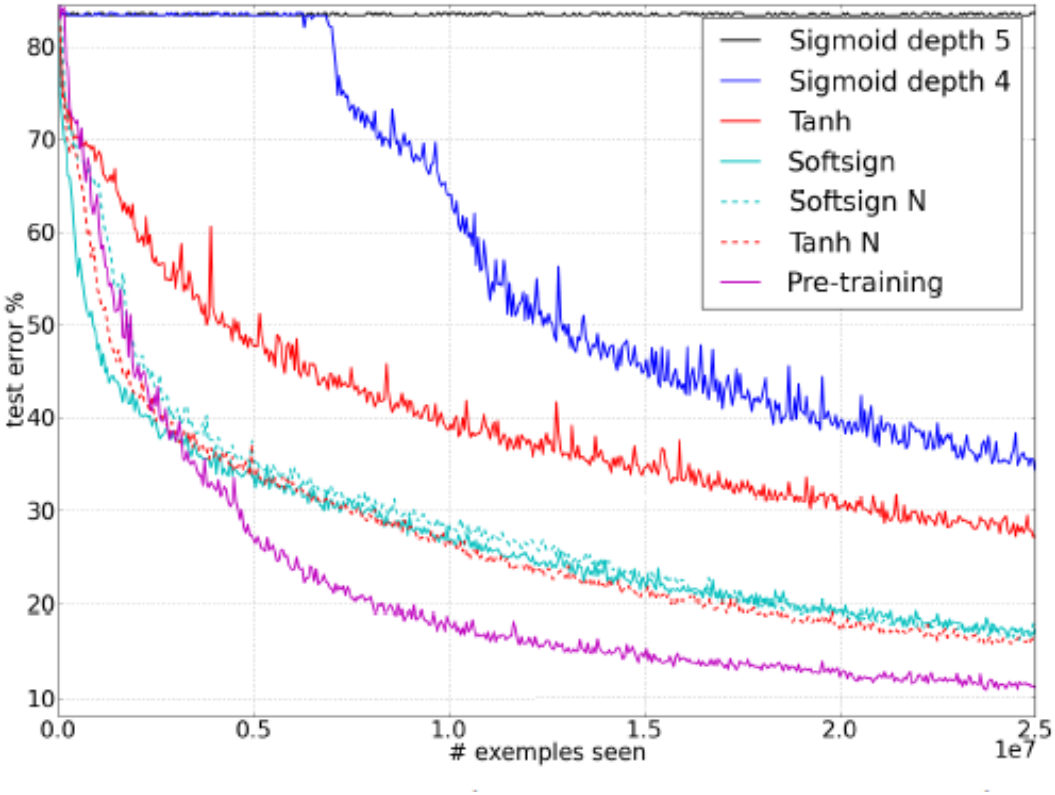
Fine-tuning

- Plug all layers together
- Compute the costs based on the actual input x
- Update **all** weights using backpropagation

Source: http://ufldl.stanford. edu/wiki/

Is pre-training really necessary?

- Xavier Glorot and Yoshua Bengio, 2010, *Understanding the difficulty of training deep feedforward neural networks*
- With the right activation function and initialization, the importance of pre-training decreases



Is pre-training really necessary?

- Pre-training achieves two things:
  - It makes optimization easier
  - It reduces overfitting
- Pre-training is not required to make optimization work, if you have enough data
  - Mainly due to a better understanding how initialization works
- Pre-training is still very effective on small datasets
- More information: https://www.youtube.com/watch?v=vShMxxqtDDs

Dropout in Neural Networks

TECHNISCHE UNIVERSITÄT DARMSTADT

Inspired by Hinton
https://www.youtube.com/watch?v=vShMxxqtDDs

For details:
Srivastava, Hinton et al., 2014, *Dropout: A Simple Way to Prevent Neural Networks from Overtting*

Ensemble Learning

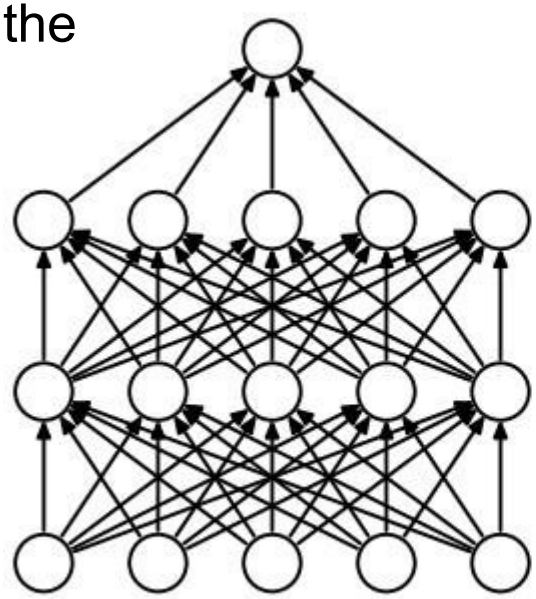
- Create many different models and combine them at test time to make prediction
- Averaging over different models is very effective against overfitting
- Random Forest
  - A single decision trees is not very powerful
  - Creating hundreds of different trees and combine them
- Random forests works really well
  - Several Kaggle competitions, e.g. Netflix, were won by random forests

Model Averaging with Neural Nets

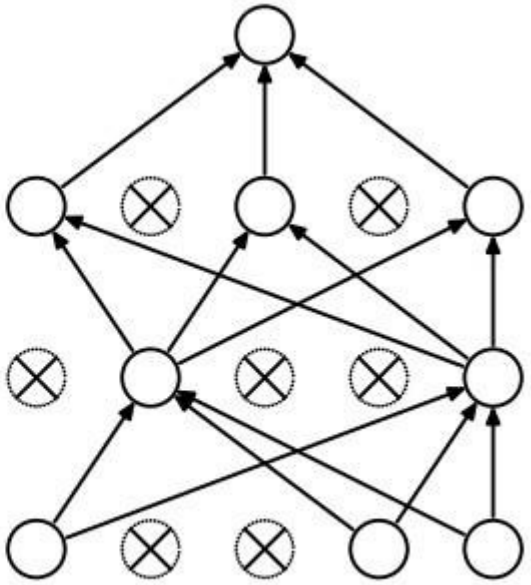
- We would like to do massive model averaging
  - Average over 100, 1.000, 10.000 or 100.000 models
- Each net takes a long time to train
  - We don't have enough time to learn so many models
- At test time, we don't want to run lots of large neural nets
- We need something that is more efficient
  - Use dropouts!

Dropout

- Each time present a training example, we dropout 50% of the hidden units
- With this, we randomly sample over 2^H different architectures
  - H: Number of hidden units
- All architectures share the same weights



(a) Standard Neural Net



(b) After applying dropout.

Img source: http://cs231n.github.io/

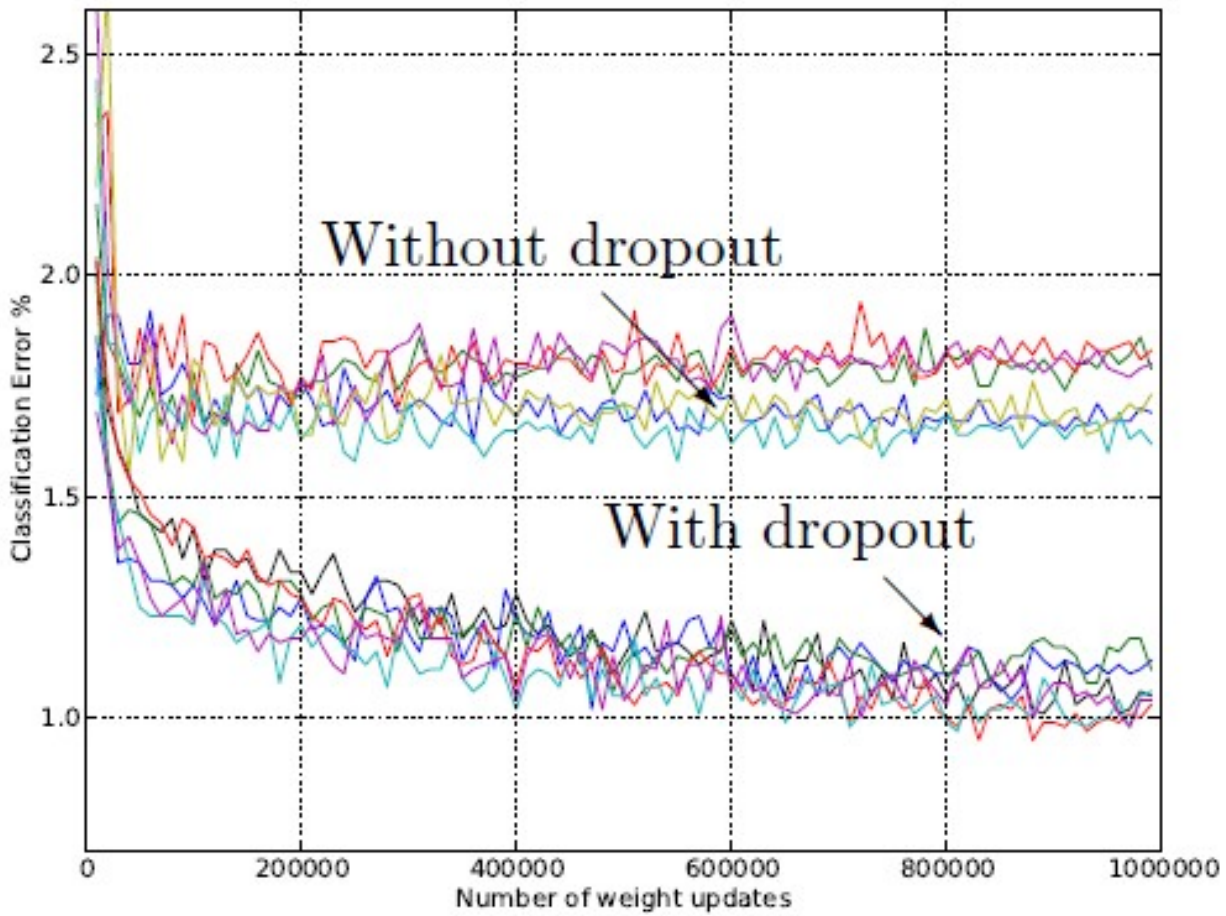
Dropout

- With H hidden units, we sample from 2^H different models
  - Only few of the models get ever trained and they only get 1 training example
- Sharing of weights means that every model is strongly regularized
  - Much better than L1 and L2 regularization, which pulls weights towards zero
  - It pulls weights towards what other models need
  - Weights are pulled towards sensible values
- This works in experiments extremely well

Dropout – at test time

- We could sample many different architectures and average the output
  - This would be way too slow
- Instead: Use all hidden units and half their outgoing weights
  - Computes the geometric mean of the prediction of all 2^H models
  - We can use other dropout rates than p=0.5. At test time, multiply weights by 1-p
- Using this trick, we train and use trillions of "different" models
- For the input layer:
  - We could apply dropout also to the input layer
  - The probability should be then smaller than 0.5
  - This is known as denoising autoencoder
  - Currently this cannot be implemented in out-of-the-box Keras

How well does dropout work?



Classification error on MNIST dataset

Source: Srivastava et al, 2014, Drouput A Simple Way to Prevent Neural Networks from Overtting

How well does dropout work?

- If your deep neural network is significantly overfitting, dropout will reduce the number of errors a lot

- If your deep neural network is not overfitting, you should be using a bigger one
  - Our brain: #parameters >> #experiences
  - Synapses are much cheaper then experiences

Another way to think about Dropout

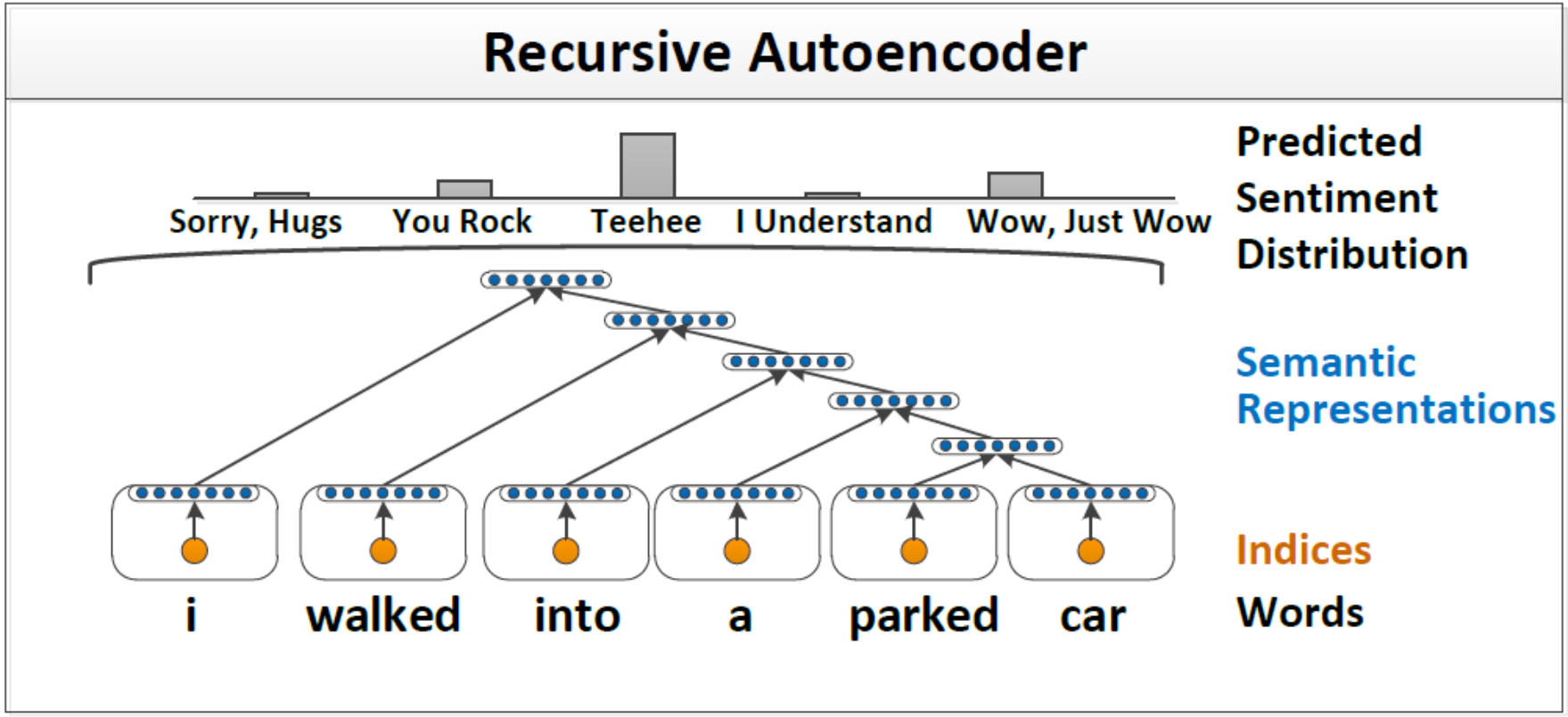
- In a fully connected neural network, a hidden unit knows which other hidden units are present
  - The hidden unit co-adapt with them for the training data
  - But big, complex conspiracies are not robust -> they fail at test time

- In the dropout scenario, each unit has to work with different sets of co-workers
  - It is likely that the hidden unit does something individually useful
  - It still tries to be different from its co-workers

Recursive Neural Networks

- Socher et al., 2011, *Semi-supervised recursive autoencoders for predicting sentiment distributions*
- Socher et al., 2013, *Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank*

Recursive Autoencoders



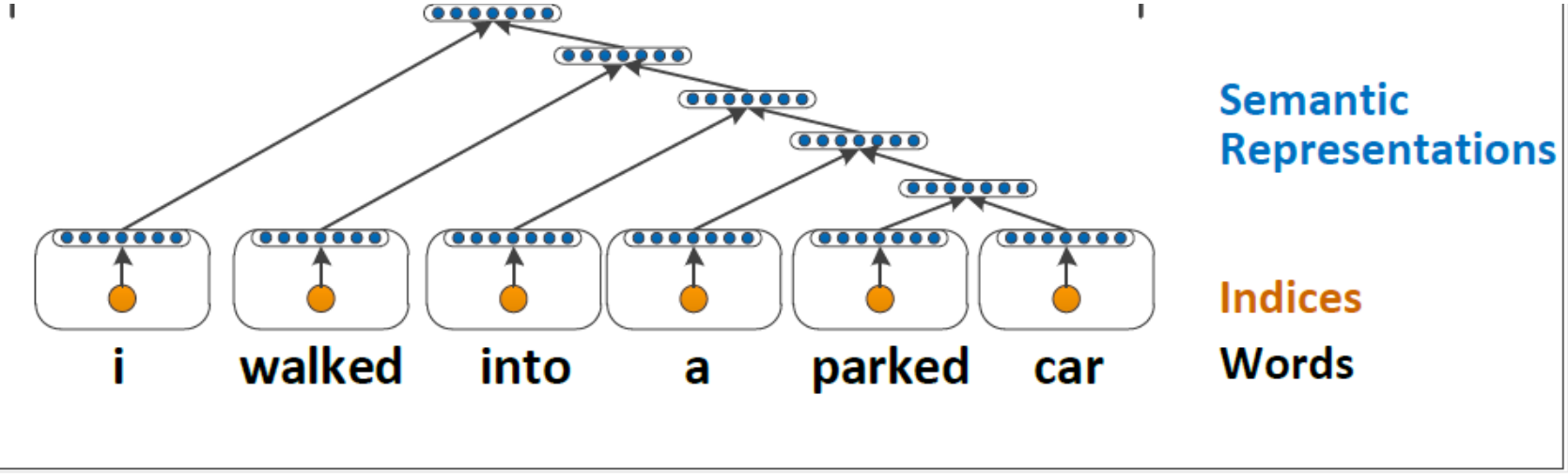
- In a first step, words are mapped to dense vectors (word embeddings)
- Iteratively they are combined and reduced to form a single compact representation of the sentence

Recursive Autoencoders (RAE)

- Given two embeddings x_1, x_2 each with length *n*
- The autoencoder takes $[x_1; x_2]$ as input and maps it to a hidden layer of size *n:*

$$y_1 = f(W[x_1; x_2] + b)$$

- The function is repeatedly applied for the whole sentence until we receive a single vector of size *n*, representing the semantic of this sentence



Selecting the nodes that should be combined

- The previous slides showed a joining of the vectors from right to left
- However, we can define any tree structure for the combination of two vectors, for example a parse tree
- Socher et al. present a greedy approach for the combination of vectors
  - Compute the reconstruction error for all neighboring vectors.
  - The two neighbors with the lowest error are selected and their nodes are replaced by the compressed representation.
  - Repeat the previous two steps until we end up with a single vector representing the semantics of the sentence
- A different, more recent approach, is to use parse trees

- The output of the recursive autoencoder can be used for a classification task by adding a final softmax layer:

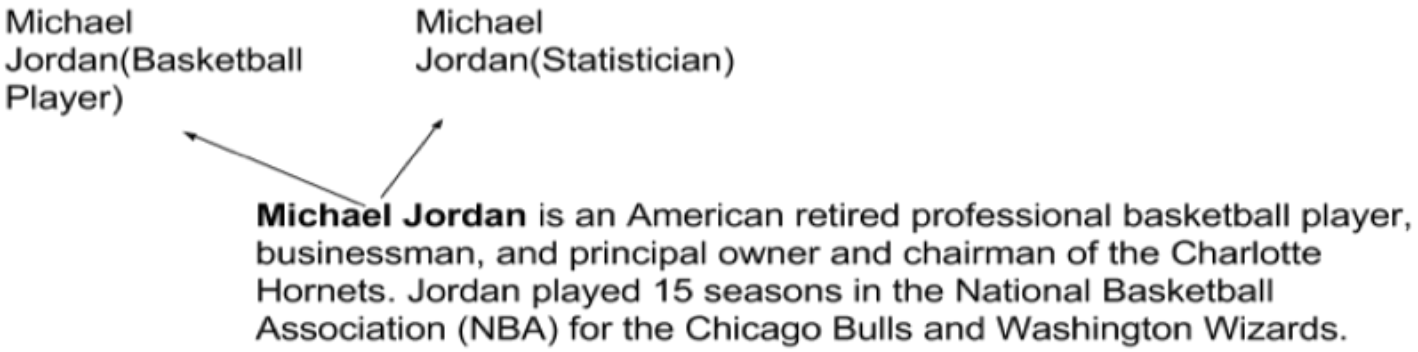
$$o = softmax(W^{label} y_m)$$

Named Entity Disambiguation

Introduction

- Named Entity Disambiguation is a central problem of Information Extraction where the goal is to link entities in a knowledge base (KB) to their mention spans in unstructured text.

- A knowledge base (KB) is a technology used to store complex structured and unstructured information used by a computer system.

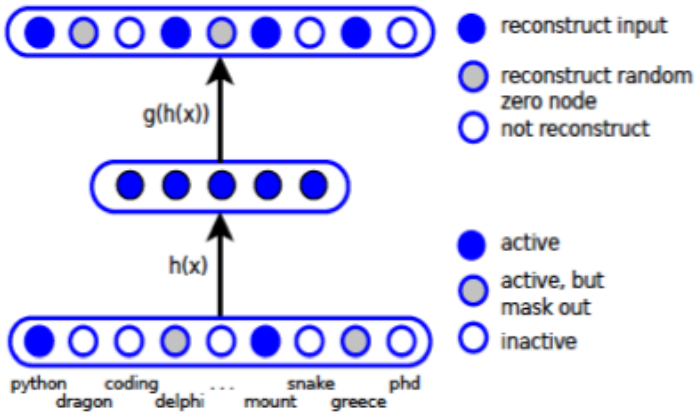


Learning Entity Representation for Entity Disambiguation [ACL'13]

- Deep Neural Network based approach to solve NED.
- Given a mention string *m*
  with its context document *d*
  a list of candidate entities *C(m)* are generated form
- for each candidate entity $e_i \in C(m)$:
  we compute a ranking score $sim(d_m, e_i)$ indicating how likely *m* refers to e_i.
- The linking result is $e = arg\ max_{ei}\ sim(d_m, e_i)$

Learning Entity Representation for Entity Disambiguation [ACL'13]

- The approach consists of two steps:
- **Step 1: Greedy Layer-wise Pre-training** :
- Goal is to minimize reconstruction error L(x, g(h(x))
- Thus using Denoising Autoencoder to retain important information while ignoring noise.



Learning Entity Representation for Entity Disambiguation [ACL'13]

- **Supervised Fine Tuning** :

1. Similarity score of *(d,e)* pair is
   a. *L(d, e) = max{0, 1 − sim(d, e) + sim(d, e')}*

2. Goal is to rank the correct entity higher than the rest candidates relative to the context of the mention.

$$L(d, e) = -log \frac{\exp sim(d, e)}{\sum_{e_i \in C_m} \exp sim(d, e_i)}$$

Learning Entity Representation for Entity Disambiguation [ACL'13]

- **Supervised Fine Tuning** :

3. For each training instance (d, e), contrast it with one of its negative candidate pair (d, e'). This gives the pairwise ranking criterion
   a. sim(d,e)=dot(f(d),f(e))

4. Alternatively, we can contrast with all its candidate pairs (d, e_i). That is, we raise the similarity score of true pair sim(d, e) and penalize all the rest sim(d, e_i). Then the loss function becomes

Minimize the following training objective across all training instances:

$$L = \sum_{d,e} L(d,e)$$

Learning Entity Representation for Entity Disambiguation [ACL'13]

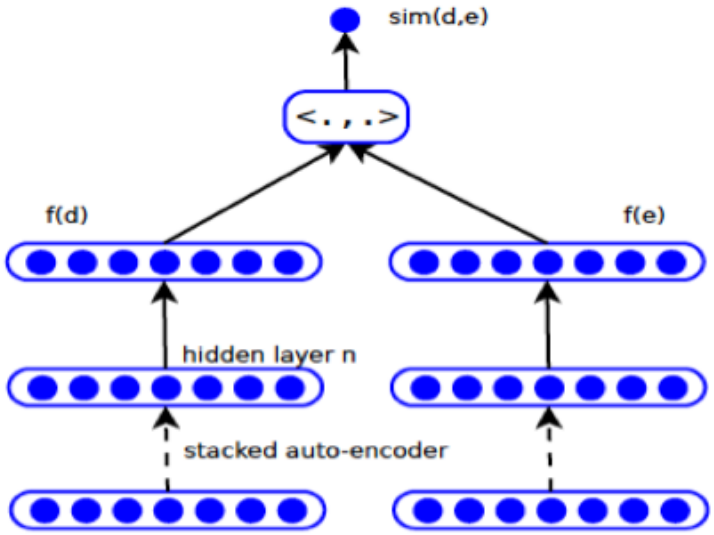


Figure 2: Network structure of fine-tuning stage.

Multi-task learning [AAAI 2018]

- As **multi-task learning**, jointly learning models for performing multiple labeling tasks.
- As **NED**, classify a *mention instance* embedded in a document, say "apple", into one of its possible *entities*, say apple (fruit) or apple (company).
- In our setting, each **mention** corresponds to a **task** and each **entity** corresponds to a **label**.
- Our training data consists of gold mention contexts X_u (for task u) with D dimension.
- Each gold mention also comes with a gold label $y \in Y_u$, where Y_u the label set admissible for task u. E is the number of global entities.
- Note that entities can be shared across multiple tasks.

Multi-task learning [AAAI 2018]

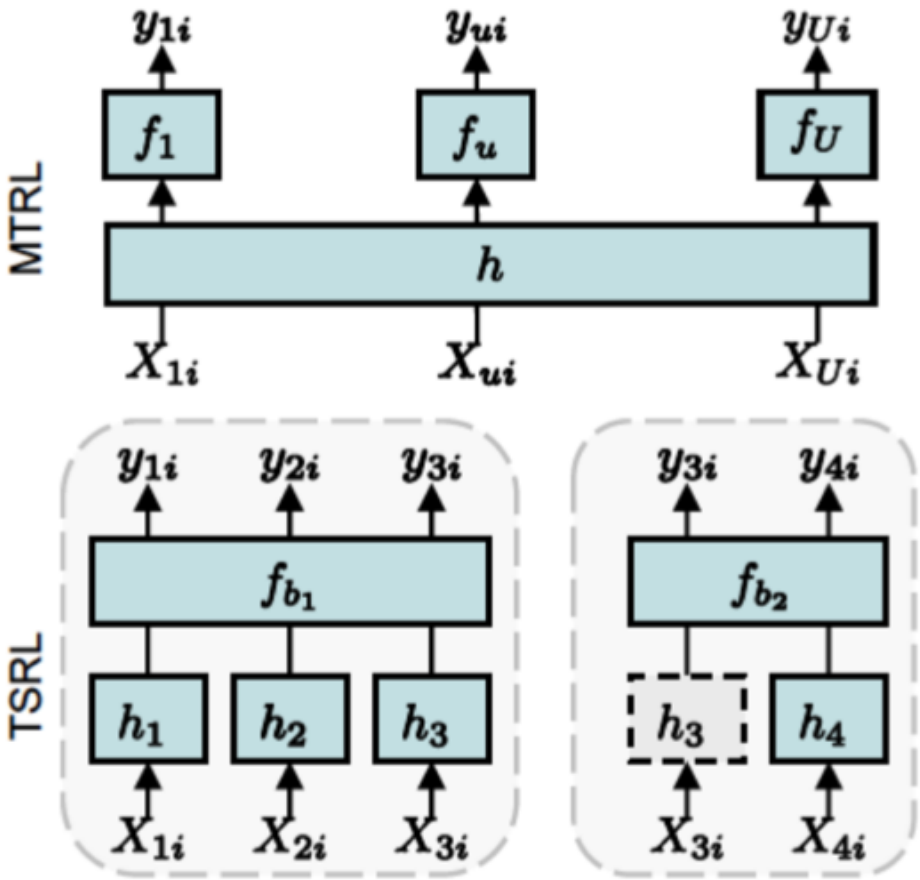
Table: Characteristics of AIDA-CoNLL testb data.

Unique mentions (tasks)	1685
Testing instances	4485
Training instances	10M
Training instances / mention	6K
Number of clusters	314
Max mentions in cluster	11
Min mentions in cluster	2
Training instances / cluster	34034
Number of raw features	16M
Number of entities	69959
Avg entities / mention	41

Multi-task learning [AAAI 2018]

- The problem of MTL-SH is handled by representation learning [Bengio et al., 2013] and multitask representation learning (MTRL) [Maurer et al., 2016], for the multitask learning setting.
- MTRL is two layer neural network which learns a shared representation in the first layer and task specific classifier in the second layer.
- The scoring function $g_u = f_u \circ h$
- f_u, the task specific scoring function defined by $f_u : \mathbb{R}^R \rightarrow \mathbb{R}^{|\mathcal{Y}_u|}$
- h, the representation function defined by $h : \mathbb{R}^D \rightarrow \mathbb{R}^R$
- We use ReLU activation functions in the first layer and L_2 norm regularization for task specific classifier.

Multi-task learning [AAAI 2018]

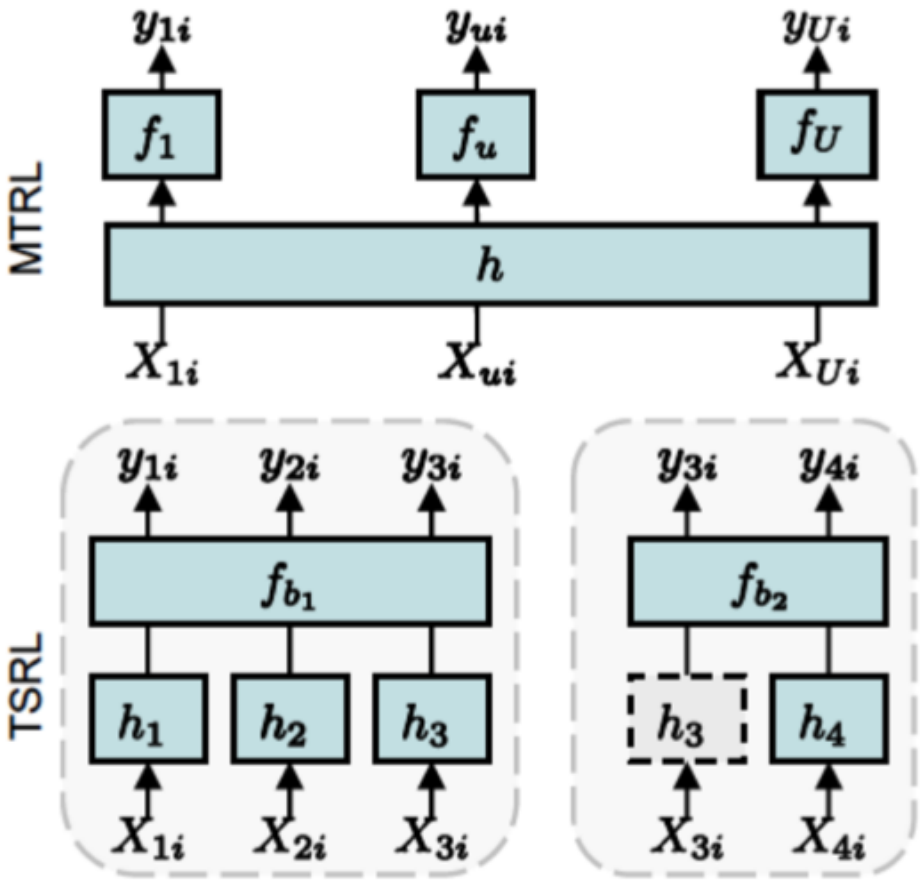


Multi-task learning [AAAI 2018]

Mention	Entity	Important Features
Bank	River Bank	breeze, view, bridge
	Financial Bank	loan, account, transaction
Apple	Apple (Fruit)	seed, sweet, sour
	Apple (Company)	mac, aesthetic, stock price

- Features should be dependent on a particular mention.
- Make the representation function h_u depend on the task u.

Multi-task learning [AAAI 2018]



Multi-task learning [AAAI 2018]

- AIDA-CoNLL[Hoffart et al., 2011] testb dataset results ~70,000 number of global entities
- In the first layer, task u cannot affect the representation of task u'.
- In the second layer, if two tasks have overlapping set of candidate entities, then two task can affect each other via shared classifier.
- By the above observation, we follow
  1. Choose a primary task, say u.
  2. Form a cluster of *related* tasks, C_u which have overlapping of candidate entities with u.
  3. Optimize (5) for only the subset of tasks C_u, and use the resulting model for task u.

Multi-task learning [AAAI 2018]

Table: Accuracy on CoNLL testb dataset.

Method	Micro-avg Accuracy	Macro-avg Accuracy
[Hoffart et al., 2011]	82.29	82.02
[He et al., 2013]	84.82	83.37
[Lazic et al., 2015]	86.4	-
[Globerson et al., 2016]	87.9	-
[Ganea and Hofmann, 2017]	88.8	-
PMC, gold mentions	85.9	89.1
MTL-SH, gold mentions	86.9	90.0
MTRL, gold mentions	82.0	86.7
TSRL, gold mentions	88.0	90.2
TSRL, gold+defn	**89.4**	**91.9**

- By adding entity definitions along with gold mentions gives the best local algorithm for NED.
- Improvement in Accuracy by using MTL-SH indicates benefits of task sharing.
- MTRL poor performance due to task clustering.

Multi-task learning [AAAI 2018]

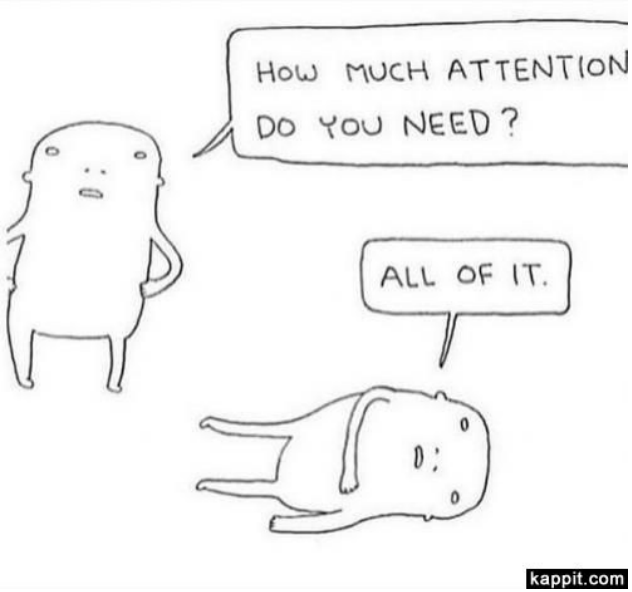
Table: Accuracy with respect to prior wise sorted entity ranking

Model	Acc in Bin-1	Acc in Bin-2	Acc in Bin-3	Acc in Bin-4	Acc in Bin-5
PMC	93.9 %	62.4 %	63.4%	64 %	35 %
MTL-SH, gold mentions	94.5 %	67.3%	69.2%	64%	–
TSRL, gold mentions	**95.5%**	**68.53%**	**73%**	64%	–

- Apart from retaining accuracy at large priors, TSRL shows clear benefits for entities with lower priors.

Questions?

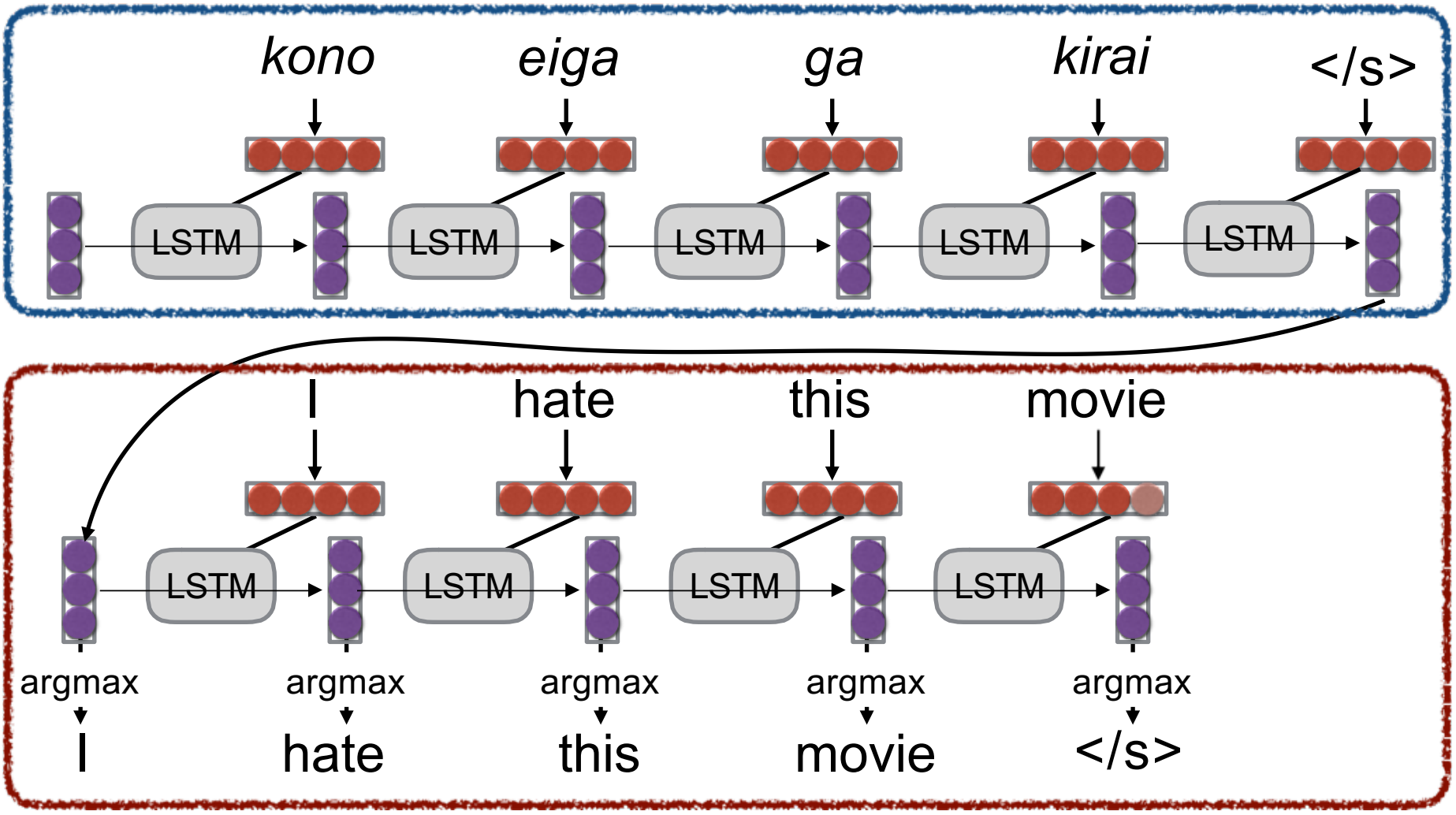
!!
Attention



Machine translation is a blackbox

Encoder-decoder Models

(Sutskever et al. 2014)



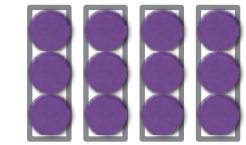
Sentence Representations

Problem!

"You can't cram the meaning of a whole %&!$ing sentence into a single $&!*ing vector!"
— Ray Mooney

- But what if we could use multiple vectors, based on the length of the sentence.

this is an example 

this is an example 

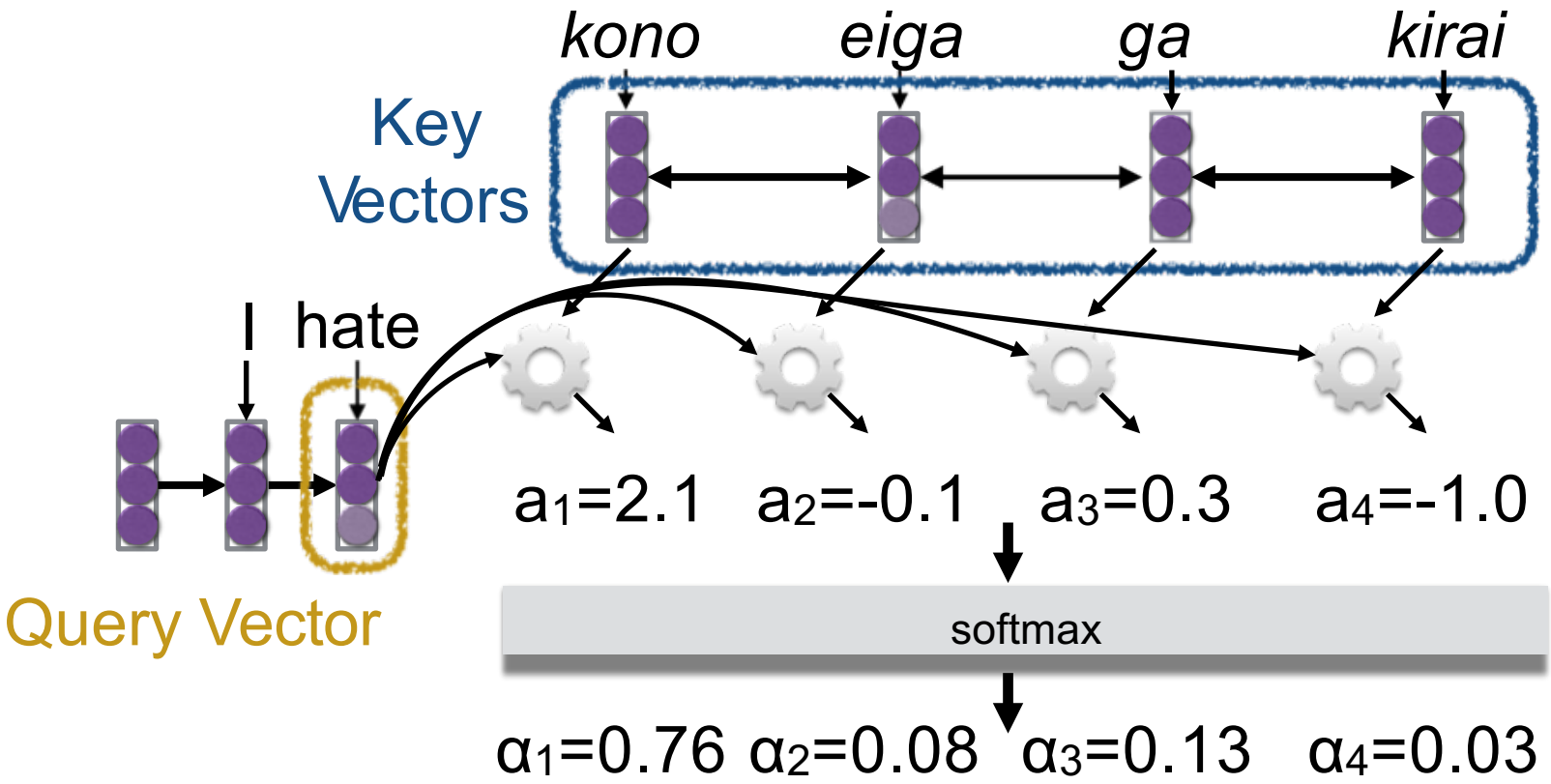
Attention - Basic Idea

(Bahdanau et al. 2015)

- Encode each word in the sentence into a vector
- When decoding, perform a linear combination of these vectors, weighted by “attention weights”
- Use this combination in picking the next word

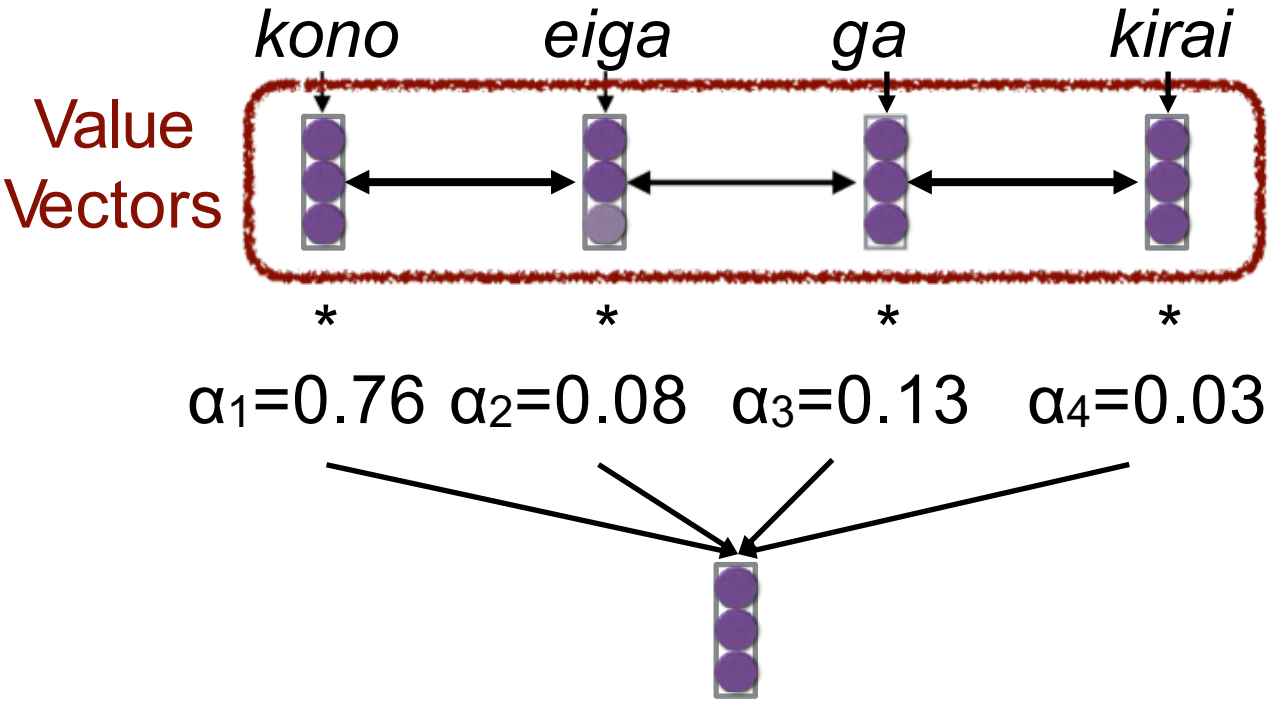
Calculating Attention (1)

- Use “query” vector (decoder state) and “key” vectors (all encoder states)
- For each query-key pair, calculate weight
- Normalize to add to one using softmax



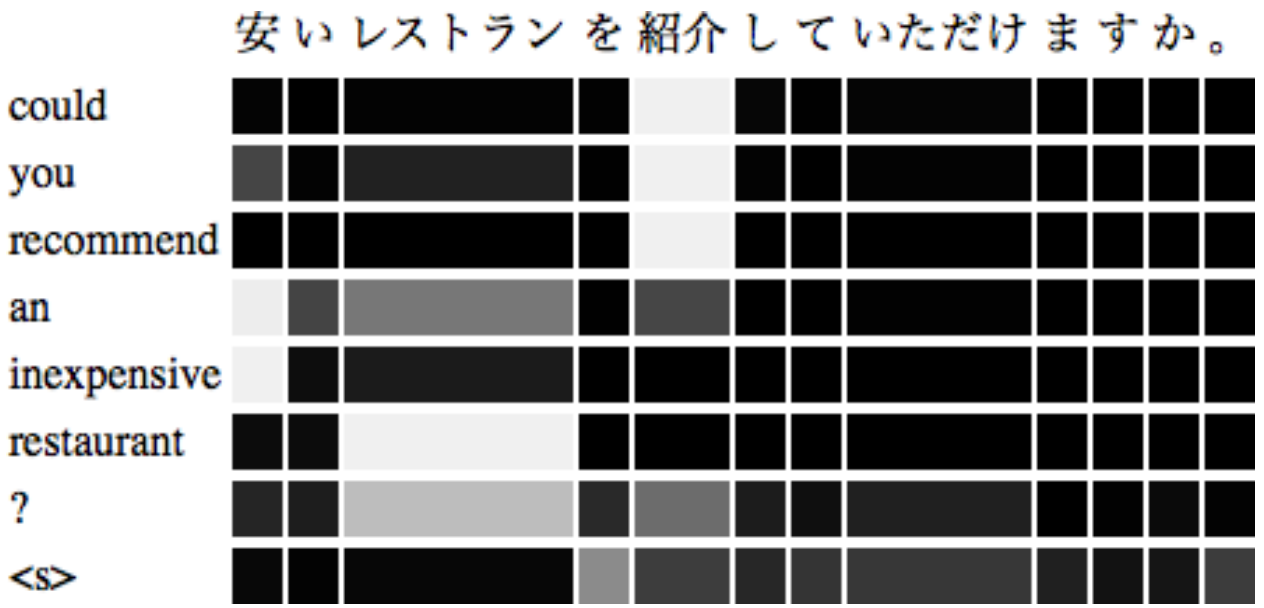
Calculating Attention (2)

- Combine together value vectors (usually encoder states, like key vectors) by taking the weighted sum



- Use this in any part of the model you like

A Graphical Example



Attention Score Functions (1)

- ***q*** is the query and ***k*** is the key
- **Multi-layer Perceptron** (Bahdanau et al. 2015)

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{w}_2^\intercal \tanh(W_1[\boldsymbol{q}; \boldsymbol{k}])$$

  - Flexible, often very good with large data

- **Bilinear** (Luong et al. 2015)

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^\intercal W \boldsymbol{k}$$

Attention Score Functions (2)

- **Dot Product** (Luong et al. 2015)

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^\intercal \boldsymbol{k}$$

  - No parameters! But requires sizes to be the same.

- **Scaled Dot Product** (Vaswani et al. 2017)

  - Problem: scale of dot product increases as dimensions get larger

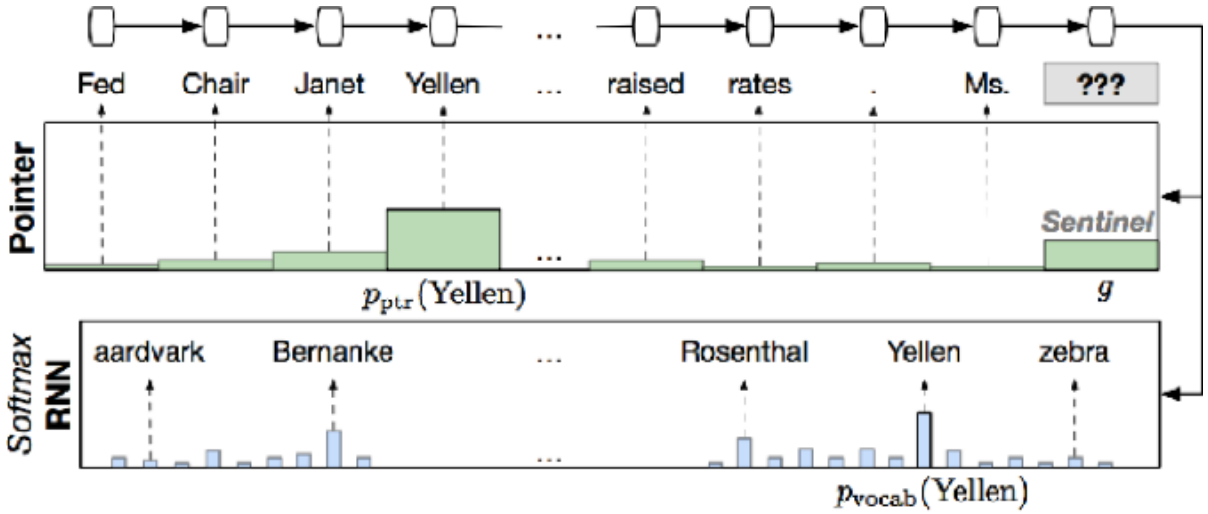
  - Fix: scale by size of the vector

$$a(\boldsymbol{q}, \boldsymbol{k}) = \frac{\boldsymbol{q}^\intercal \boldsymbol{k}}{\sqrt{|\boldsymbol{k}|}}$$

What do we Attend To?

Previously Generated Things

- In language modeling, attend to the previous words (Merity et al. 2016)

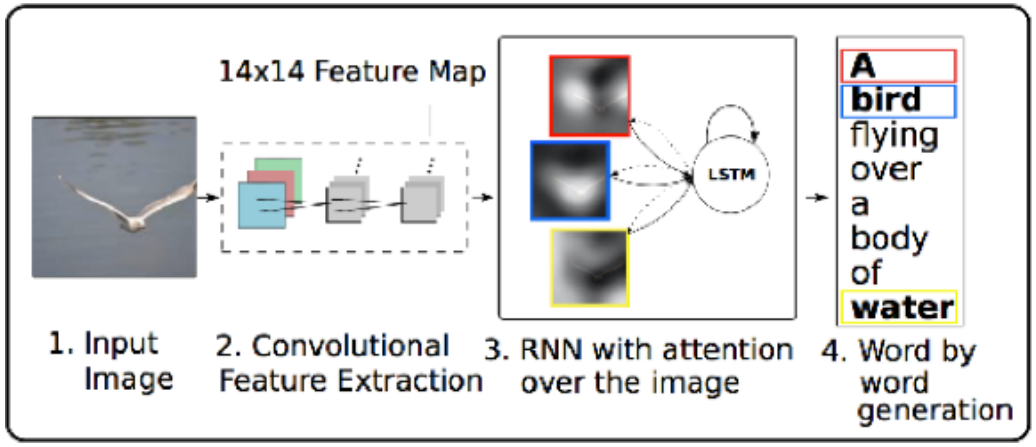


$$p(\text{Yellen}) = g\, p_{\text{vocab}}(\text{Yellen}) + (1-g)\, p_{\text{ptr}}(\text{Yellen})$$

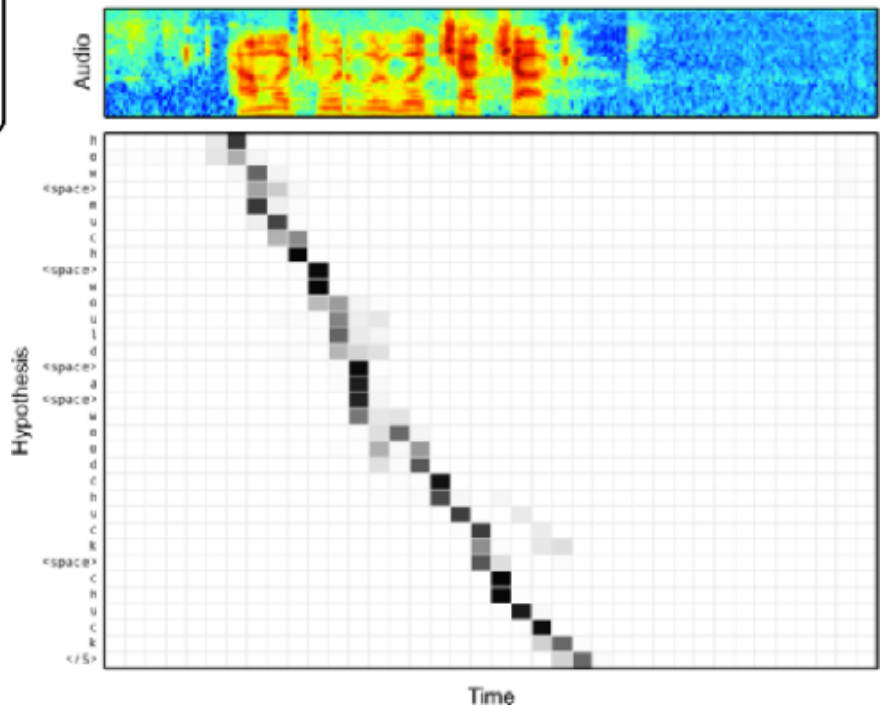
- In translation, attend to either input or previous output (Vaswani et al. 2017)

Various Modalities

- Images (Xu et al. 2015)



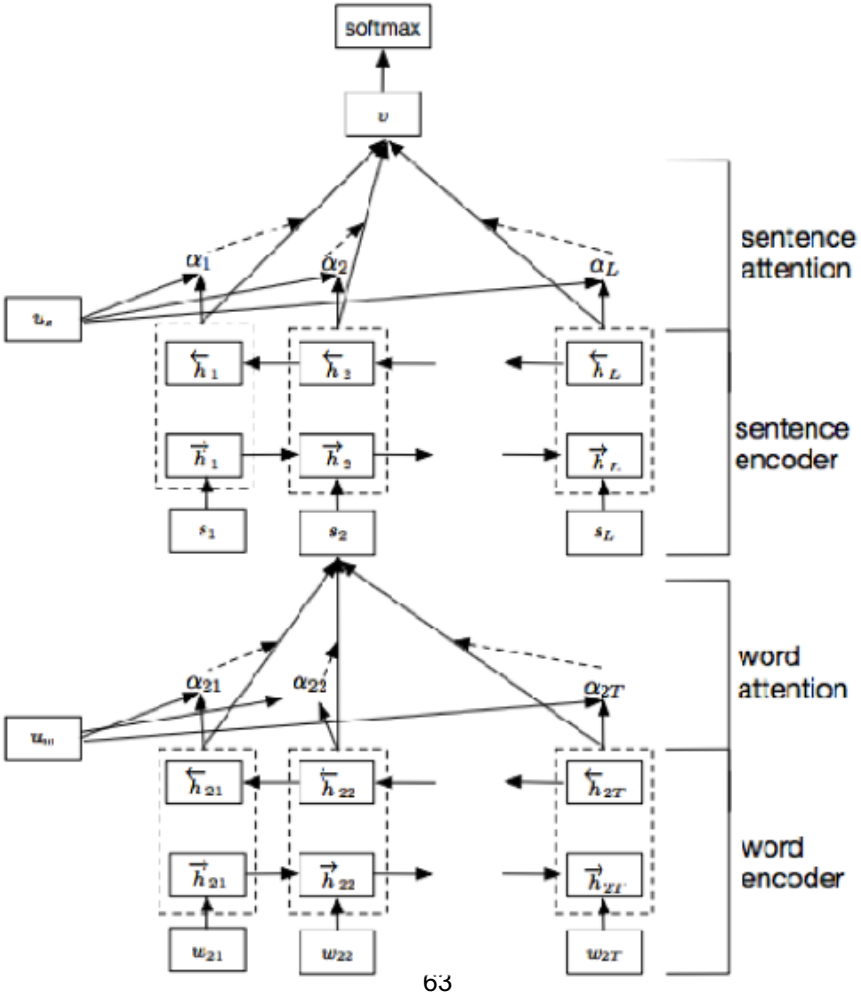
- Speech (Chan et al. 2015)



Hierarchical Structures

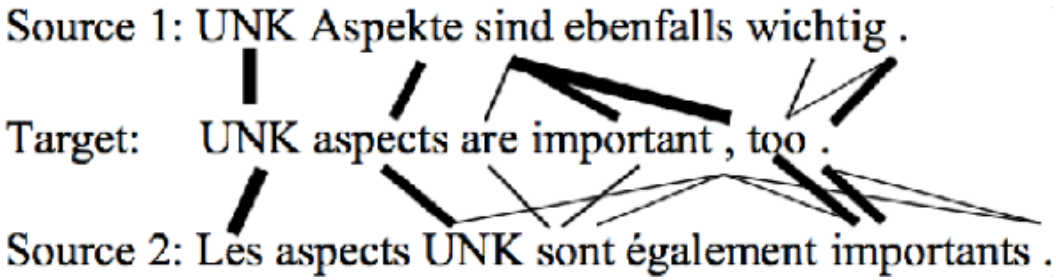
(Yang et al. 2016)

- Encode with attention over each sentence, then attention over each sentence in the document



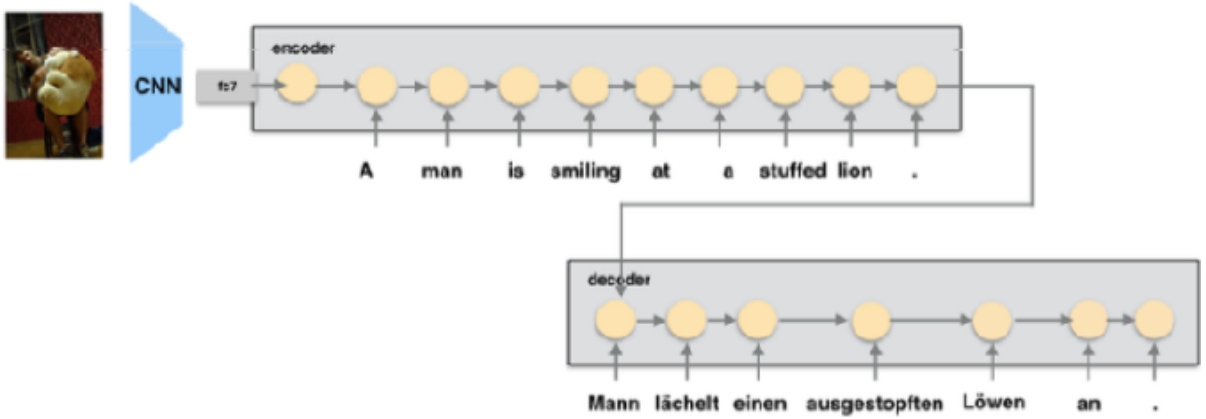
Multiple Sources

- Attend to multiple sentences (Zoph et al. 2015)



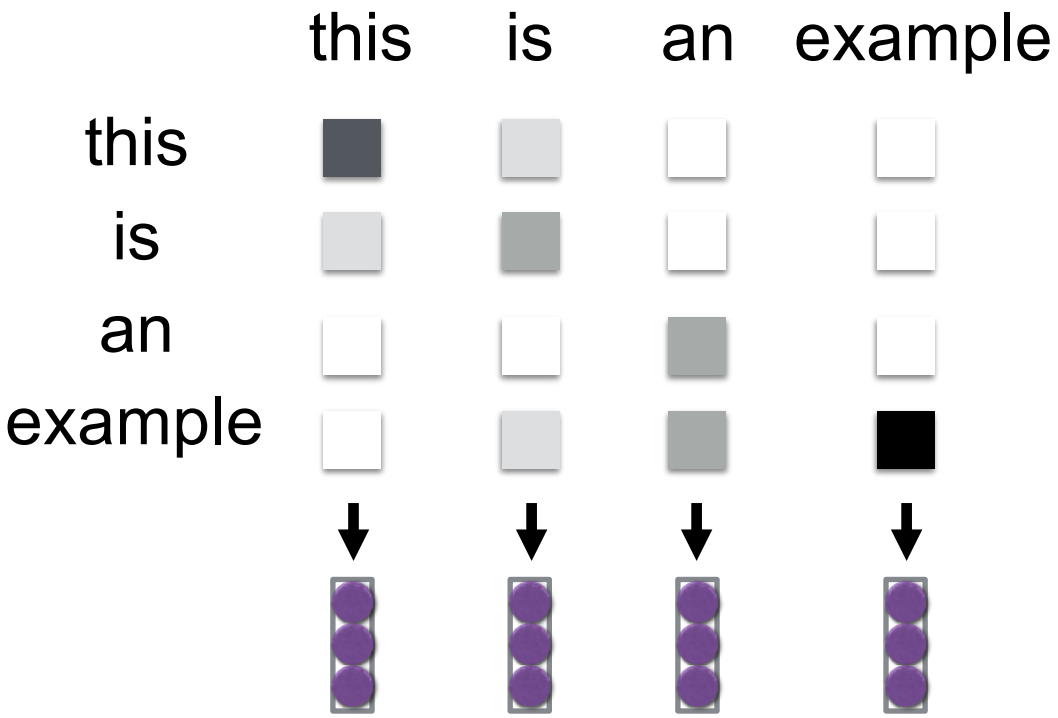
  - Libovicky and Helcl (2017) compare multiple strategies

- Attend to a sentence and an image (Huang et al. 2016)



Intra-Attention / Self Attention (Cheng et al. 2016)

- Each element in the sentence attends to other elements → context sensitive encodings!

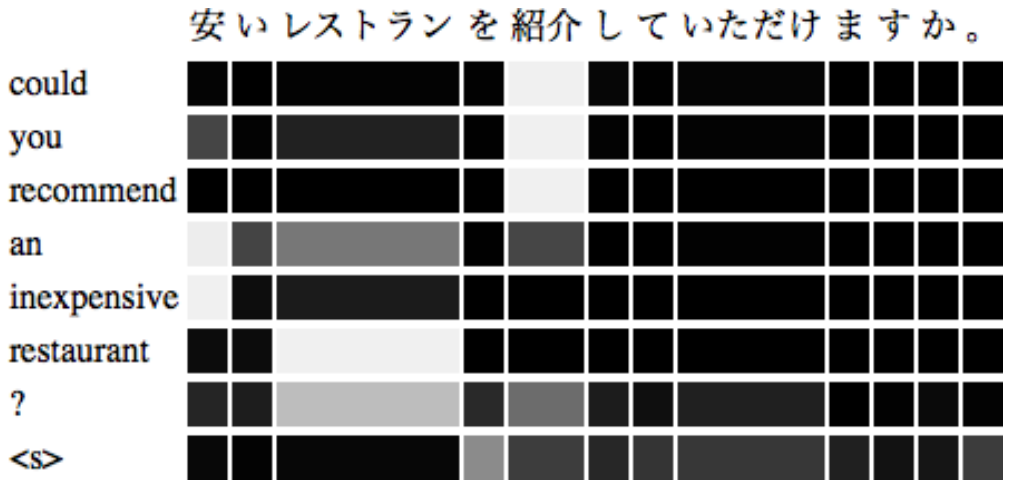


Coverage

- **Problem:** Neural models tends to drop or repeat content
- **Solution:** Model how many times words have been covered
  - Impose a penalty if attention not approx. 1 (Cohn et al. 2015)
  - Add embeddings indicating coverage (Mi et al. 2016)

Incorporating Markov Properties
(Cohn et al. 2015)

- **Intuition:** attention from last time tends to be correlated with attention this time



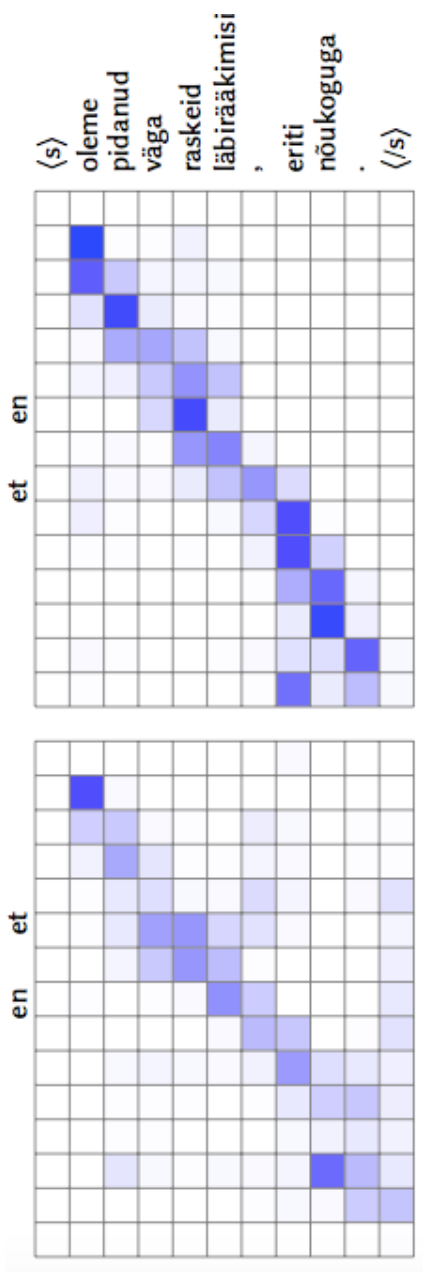
- Add information about the last attention when making the next decision

Bidirectional Training
(Cohn et al. 2015)

- **Intuition:** Our attention should be roughly similar in forward and backward directions

- **Method:** Train so that we get a bonus based on the trace of the matrix product for training in both directions

$$\mathrm{tr}(A_{X\to Y} A_{Y\to X}^{\intercal})$$

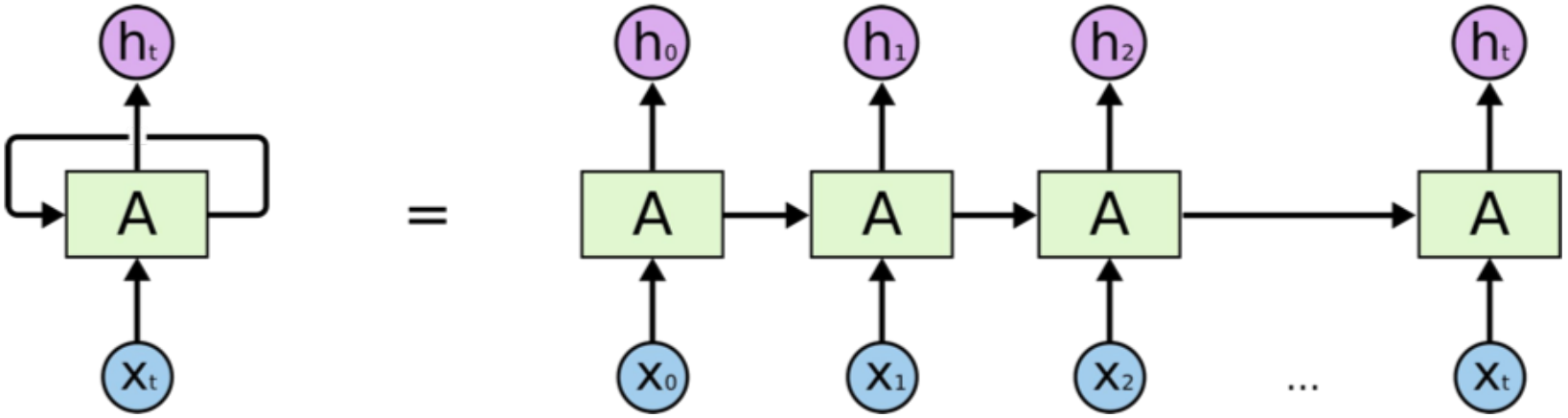


Supervised Training
(Mi et al. 2016)

- Sometimes we can get "gold standard" alignments *a-priori*
  - Manual alignments
  - Pre-trained with strong alignment model
- **Train the model to match** these strong alignments

An Interesting Case Study: "Attention is All You Need" (Vaswani et al. 2017)

Problem: RNN constrained by previous timestep computation

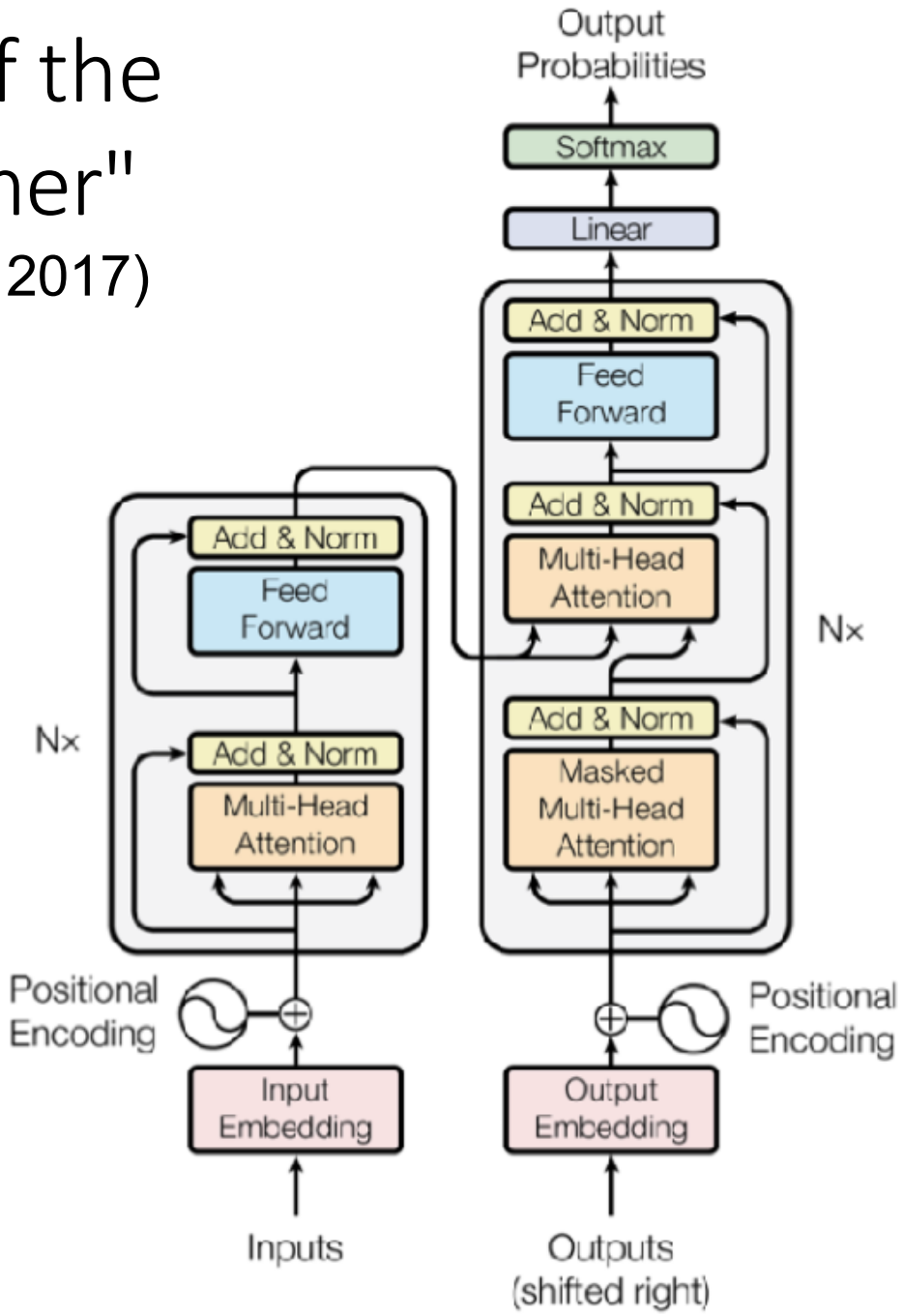


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Target is to Improve the perofmrance and get rid of sequential computation

Summary of the "Transformer" (Vaswani et al. 2017)

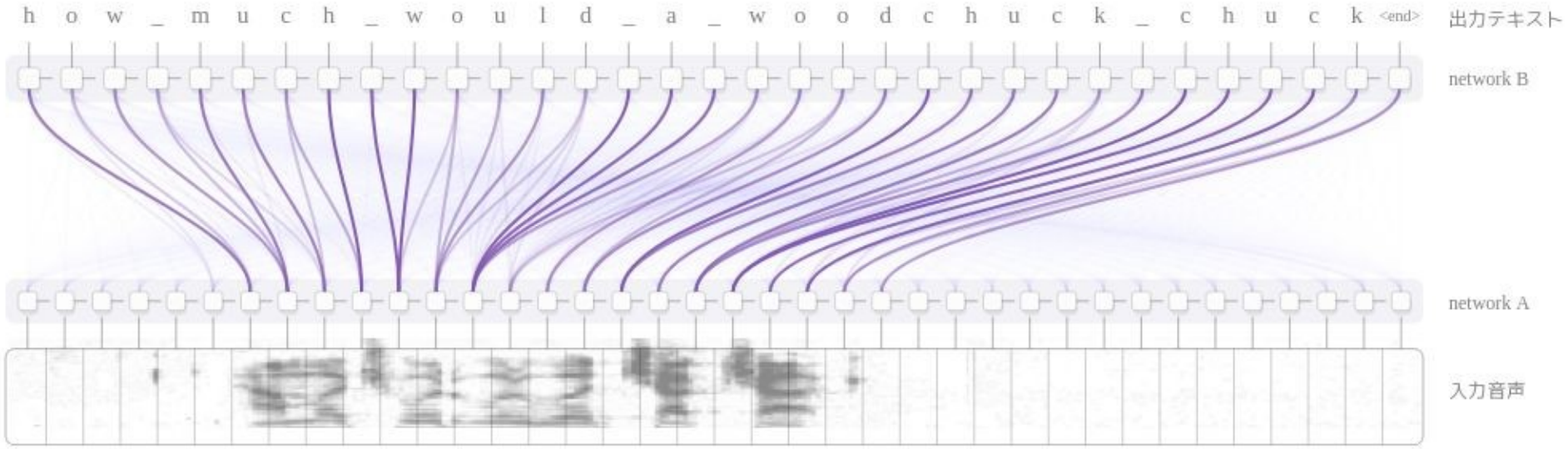
- A sequence-to-sequence model based entirely on attention
- Strong results on standard WMT datasets
- Fast: only matrix multiplications



Attention Tricks

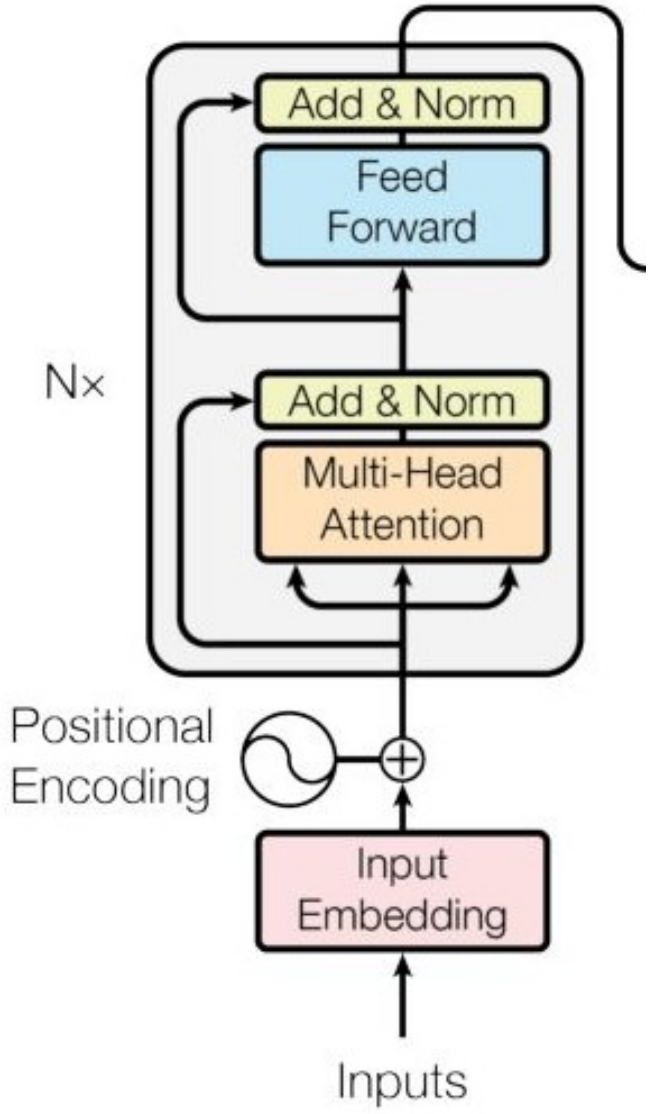
- **Self Attention:** Each layer combines words with others
- **Multi-headed Attention:** 8 attention heads learned independently
- **Normalized Dot-product Attention:** Remove bias in dot product when using large networks
- **Positional Encodings:** Make sure that even if we don't have RNN, can still distinguish positions

Self-Attention: focus on the important parts.



Model: Encoder

- N=6
- All Layersoutput size 512
- Embedding
- Positional Encoding
- Notice the Residual connection
- Multi-head Attention
- LayerNorm(x +Sublayer(x))
- Position wise feed forward

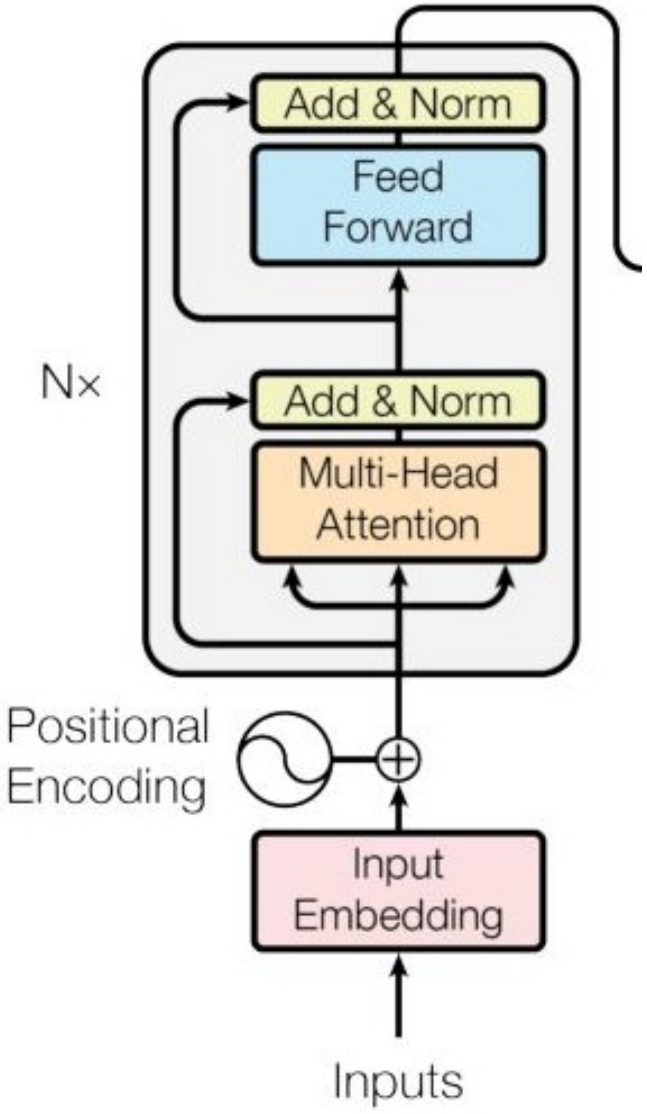


Model: Encoder

- N=6
- All Layers output size 512
- Embedding
- Positional Encoding

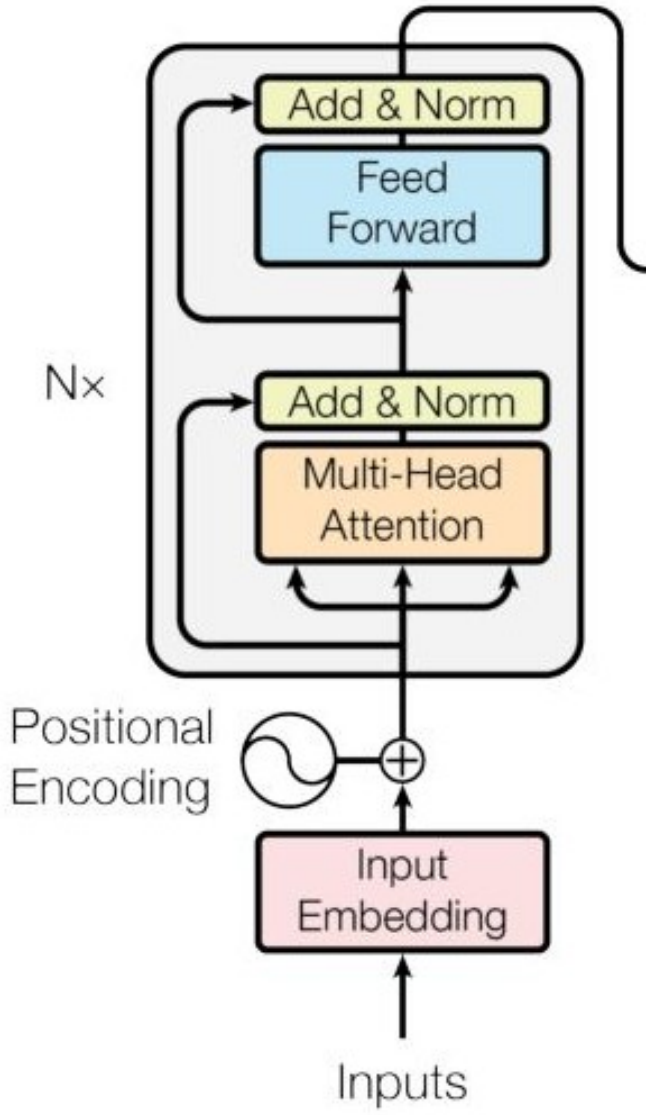
- Multi-head Attention
- LayerNorm(x +Sublayer(x))
- Position wise feed forward

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



Model:
Encoder

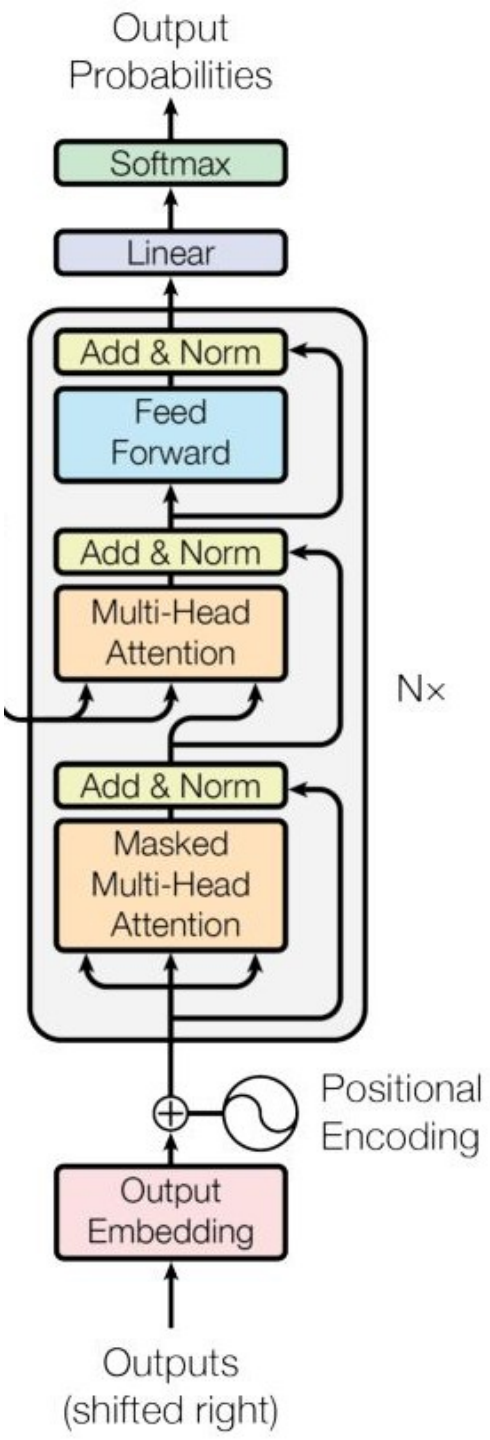
- N=6
- All Layers output size 512
- Embedding
- Positional Encoding
- Notice the Residual connection
- Multi-head Attention
- LayerNorm(x +Sublayer(x))
- Position wise feed forward



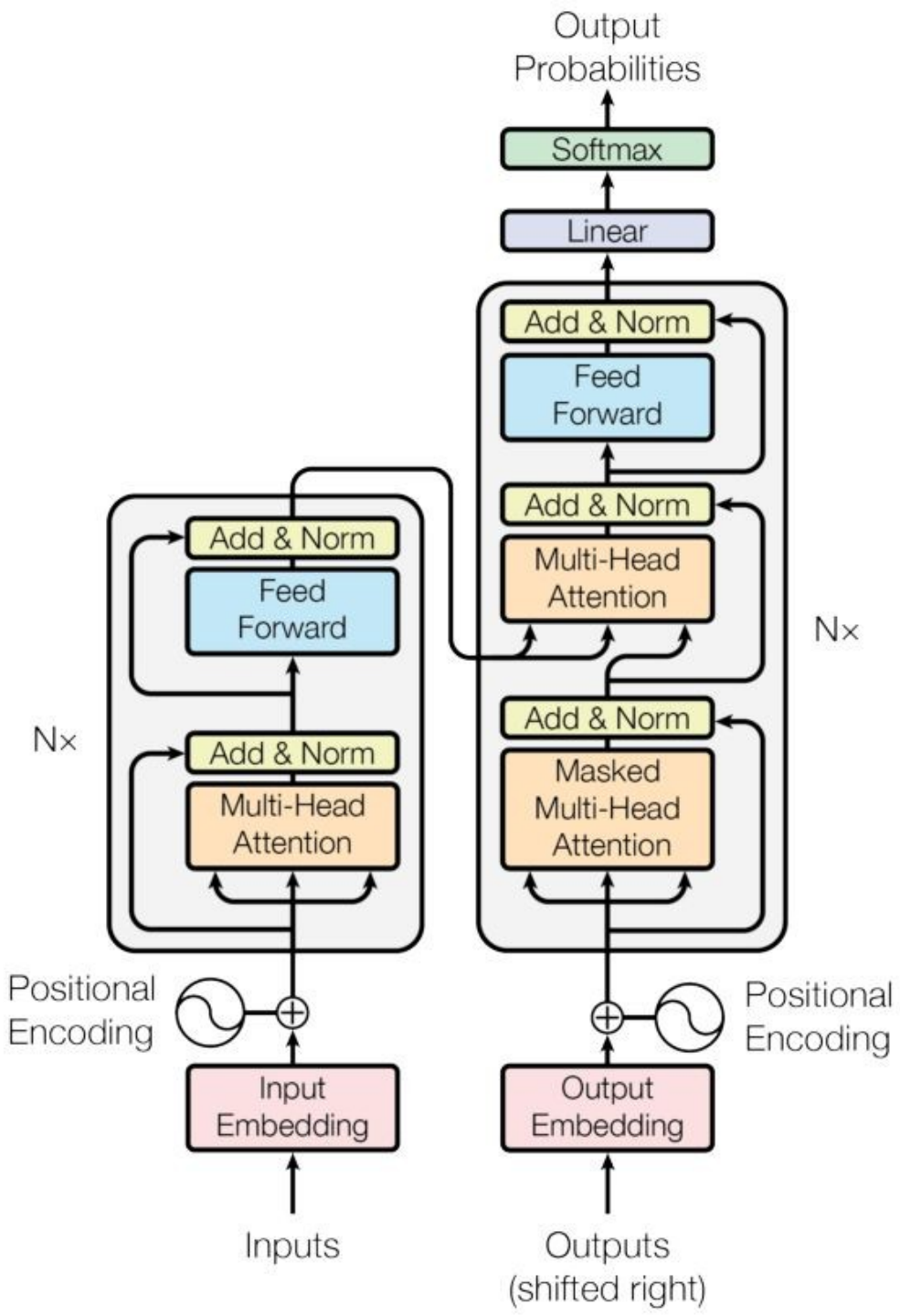
Model: Decoder

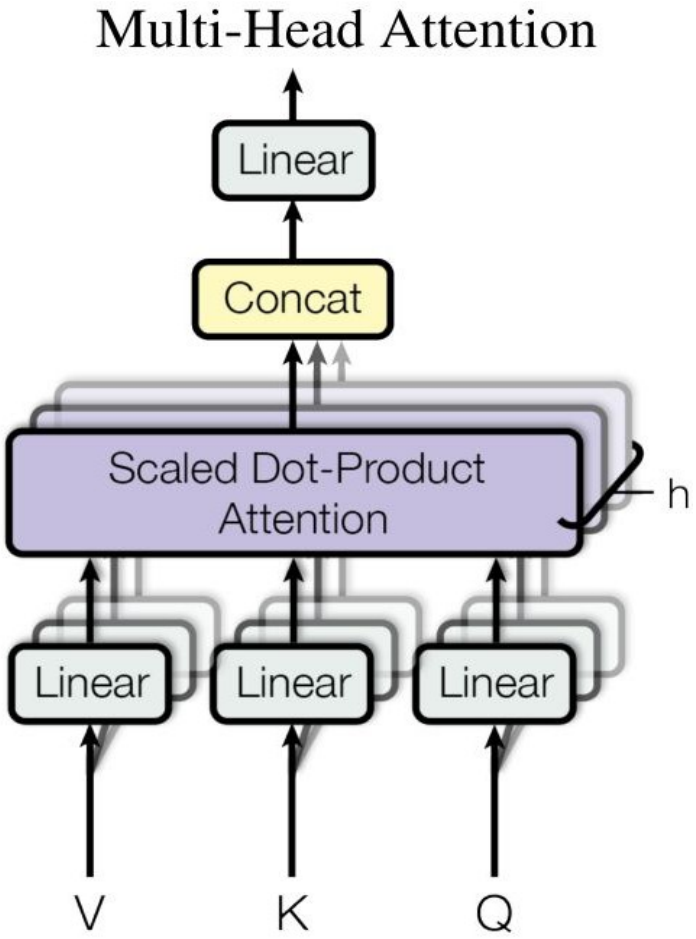
- N=6
- All Layers output size 512
- Embedding
- Positional Encoding
- Notice the Residual connection
- Multi-head Attention
- LayerNorm(x +Sublayer(x))
- Position wise feed forward

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$



Model: Complete

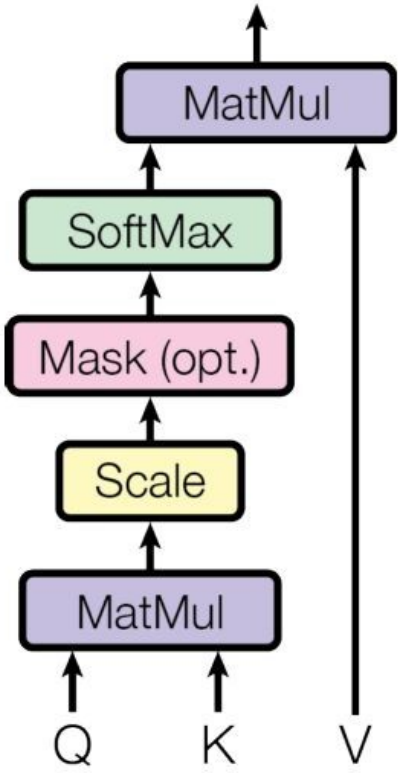




$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

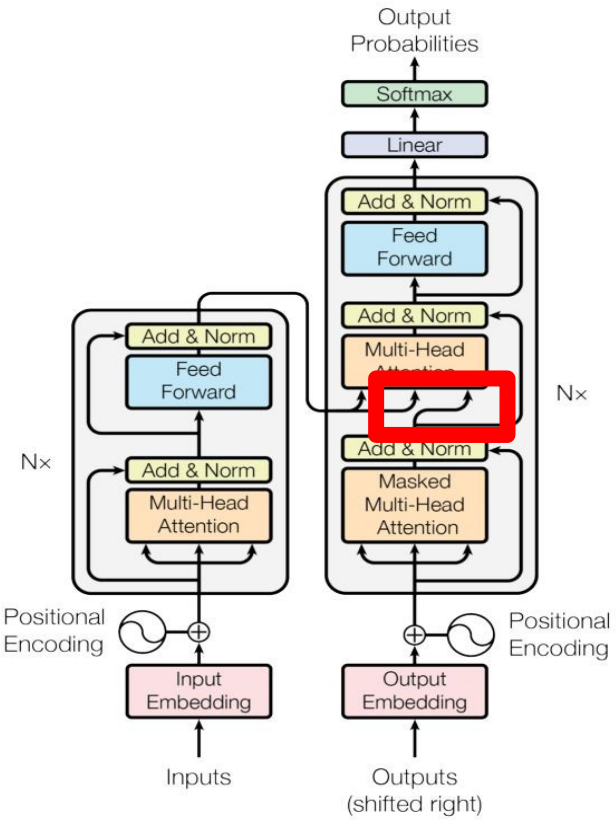
Scaled Dot-Product Attention



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Q,K,V

- "encoder-decoder attention" layers, the queries (Q) come from the previous decoder layer, and the memory keys (K) and values (V) come from the output of the encoder.”
- Otherwise: all three come from previous layer (Hidden state)



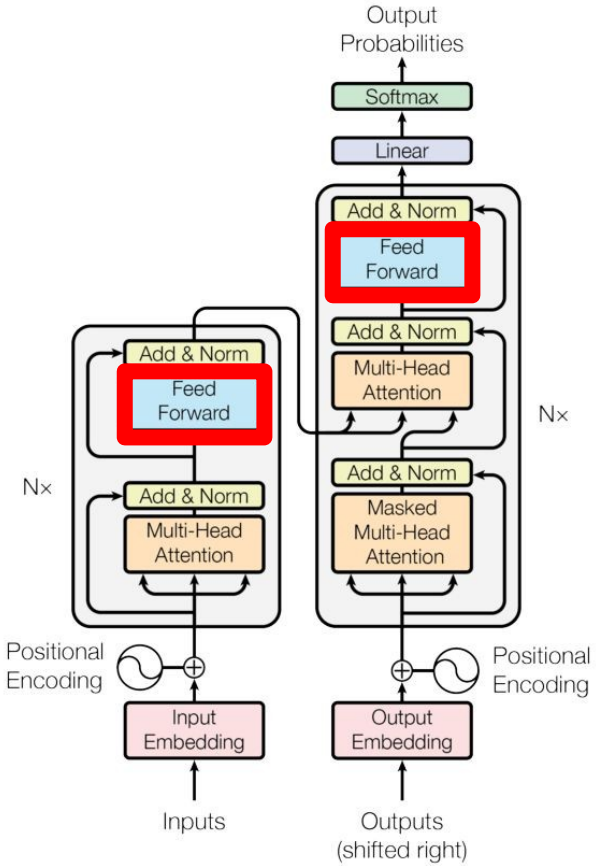
https://www.reddit.com/r/MachineLearning/comments/6kc7py/d_where_does_the_query_keys_and_values_come_from/

Complexity

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Position-wise Feed-Forward network



$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Training

- Data sets:
  - WMT 2014 English-German: 4.5 million sentences pairs with 37K tokens.
  - WMT 2014 English-French: 36M sentences, 32K tokens.
- Hardware:
  - 8 Nvidia P100 GPus (Base model 12 hours, big model 3.5 days)

http://www.prioritiesusa.org/recommendations-for-sports-training/

Training Tricks

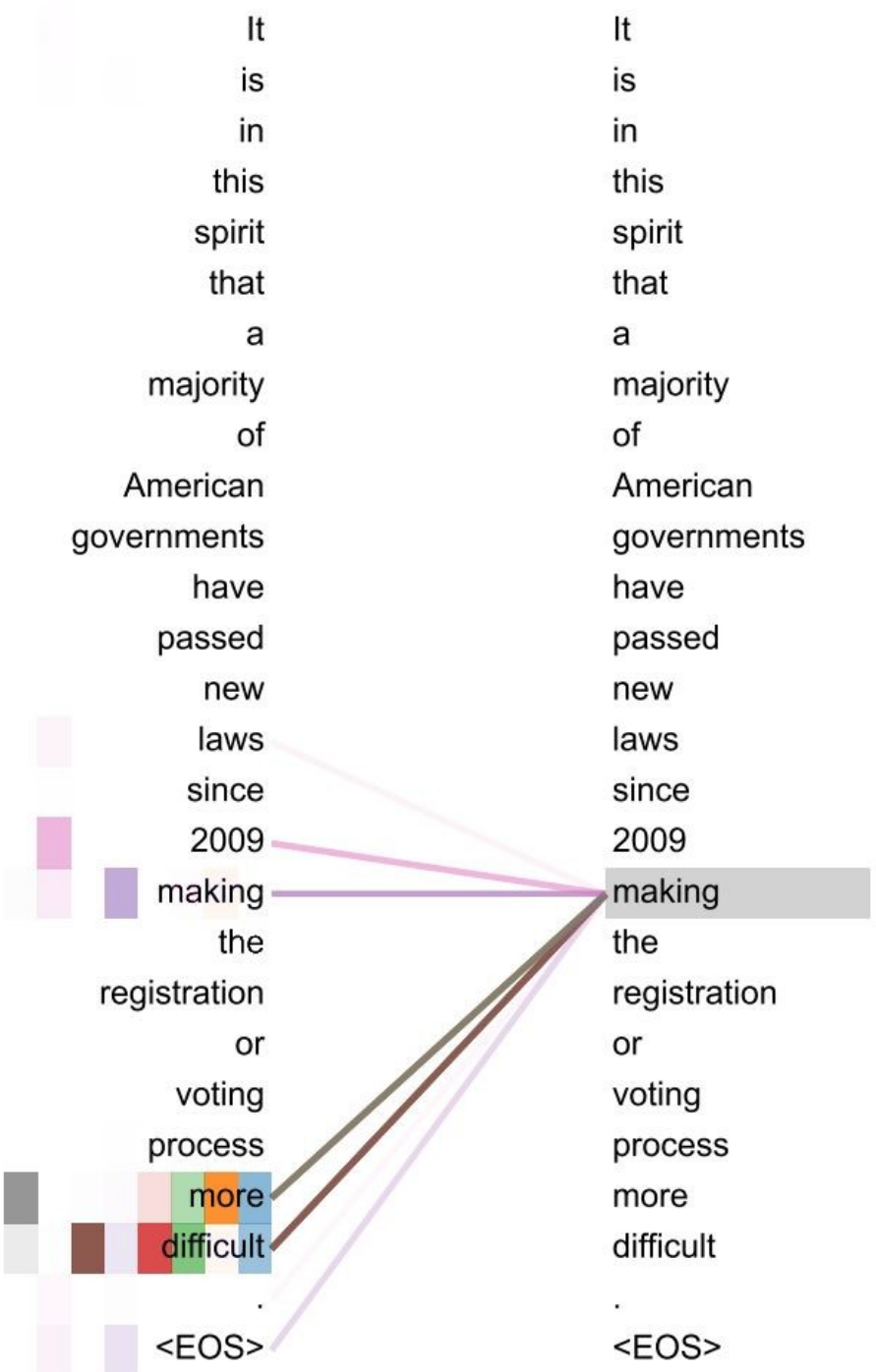
- **Layer Normalization:** Help ensure that layers remain in reasonable range
- **Specialized Training Schedule:** Adjust default learning rate of the Adam optimizer
- **Label Smoothing:** Insert some uncertainty in the training process
- **Masking for Efficient Training**

Results

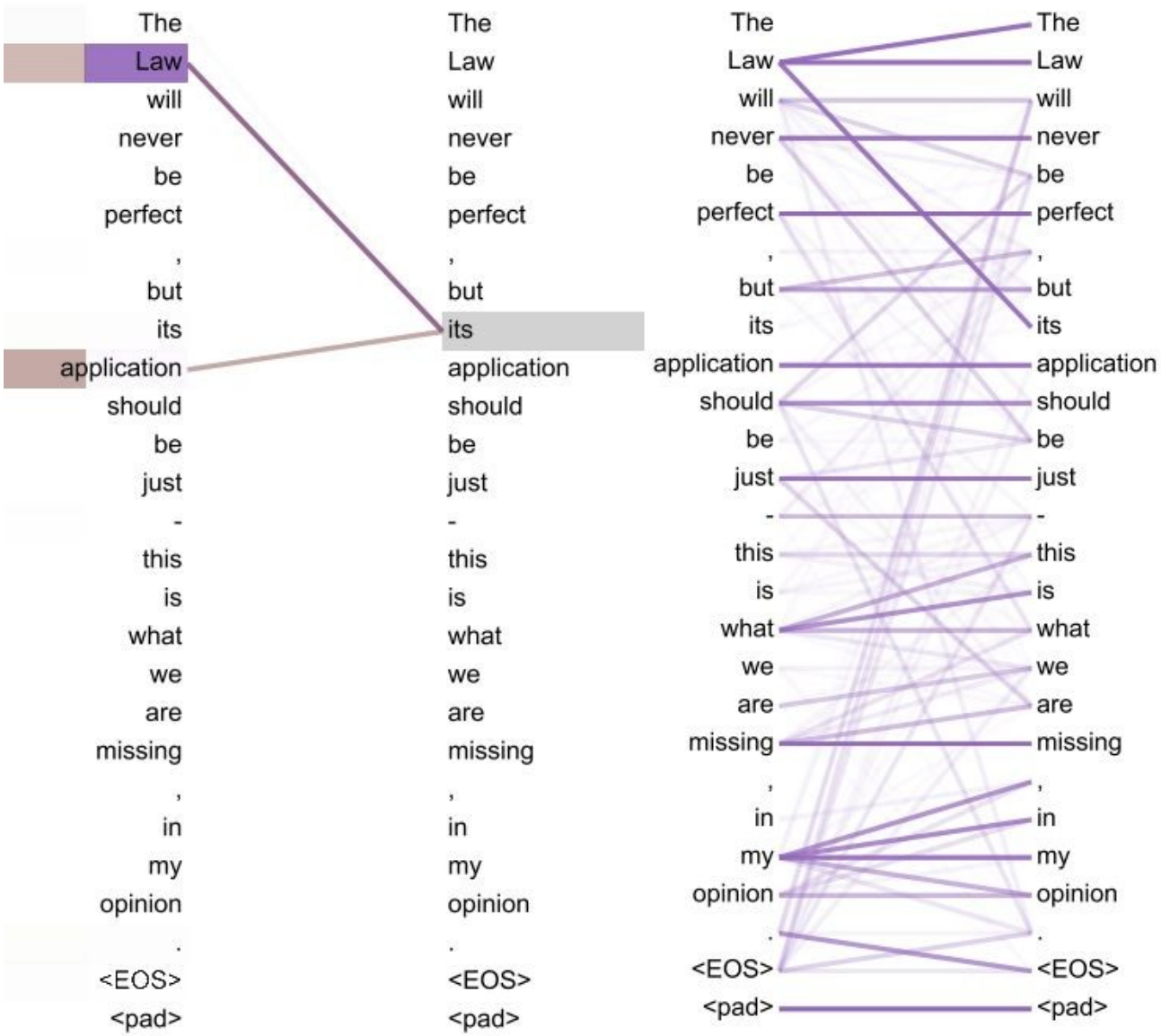
Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [17]	23.75			
Deep-Att + PosUnk [37]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [36]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [31]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [37]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [36]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Results


It
is
in
this
spirit
that
a
majority
of
American
governments
have
passed
new
laws
since
2009
making
the
registration
or
voting
process
more
difficult
.
<EOS>
It
is
in
this
spirit
that
a
majority
of
American
governments
have
passed
new
laws
since
2009
making
the
registration
or
voting
process
more
difficult
.
<EOS>

Results


The
Law
will
never
be
perfect
,
but
its
application
should
be
just
-
this
is
what
we
are
missing
,
in
my
opinion
.
<EOS>
<pad>
The
Law
will
never
be
perfect
,
but
its
application
should
be
just
-
this
is
what
we
are
missing
,
in
my
opinion
.
<EOS>
<pad>
The
Law
will
never
be
perfect
,
but
its
application
should
be
just
-
this
is
what
we
are
missing
,
in
my
opinion
.
<EOS>
<pad>
The
Law
will
never
be
perfect
,
but
its
application
should
be
just
-
this
is
what
we
are
missing
,
in
my
opinion
.
<EOS>
<pad>

Questions?

Opinion Dynamics
in Social Networks

Abir De, Isabel Valera, Sourangshu Bhattacharya, Niloy Ganguly and Manuel Gomez Rodriguez

Use social media to sense opinions

CNN

How social media is revolutionizing debates

People's opinion about political discourse

The New York Times

Campaigns Use Social Media to Lure Younger Voters

TC

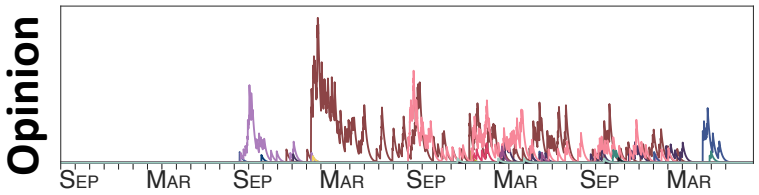
Twitter Unveils A New Set Of Brand-Centric Analytics

Brand sentiment and reputation

The New York Times

Social Media Are Giving a Voice to Taste Buds

Opinion evolves with time



93

Learning opinion dynamics

Can we **design a realistic model** that **fits real fine-grained opinion traces in a social network**?

94

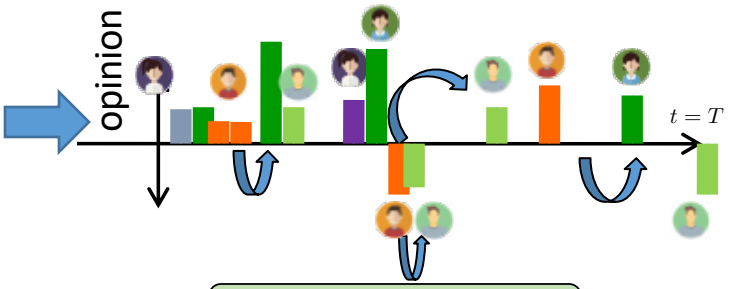
Opinion Dynamics in Social Network

Premise

People are connected

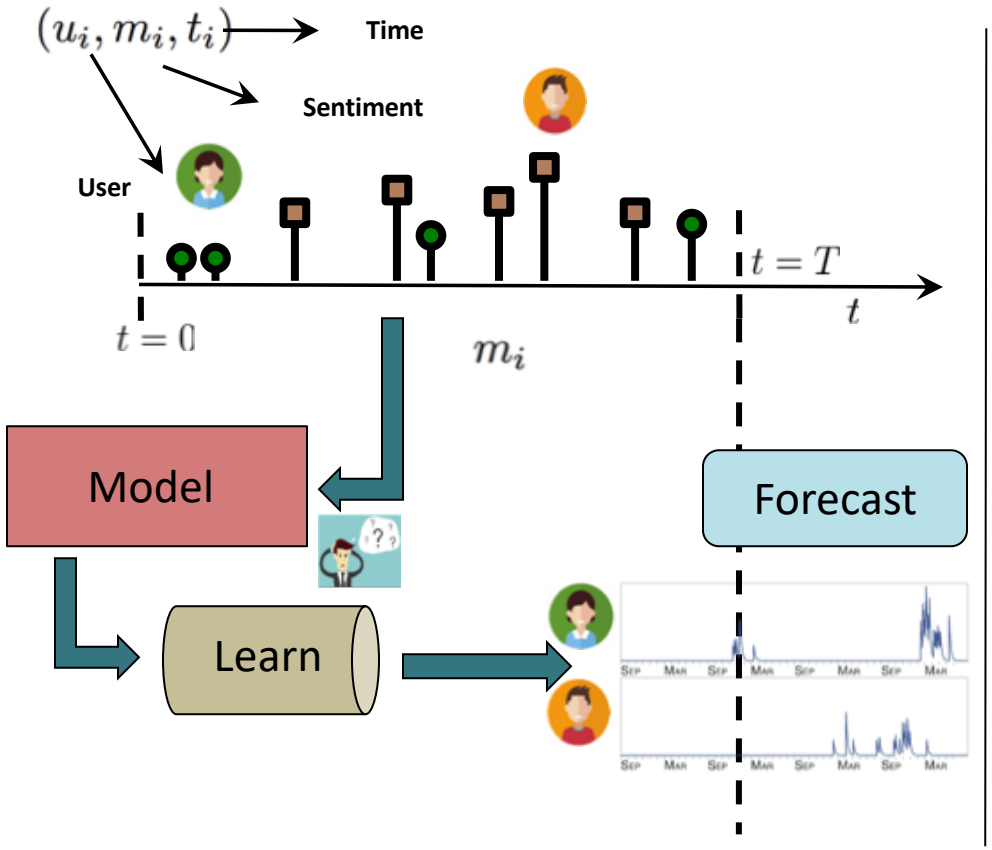
People Interact with each other

They often influence each others' opinion



Typical timeline

Objective



Message sentiments → Model → opinions

Model → What it achieves? → Forecasting

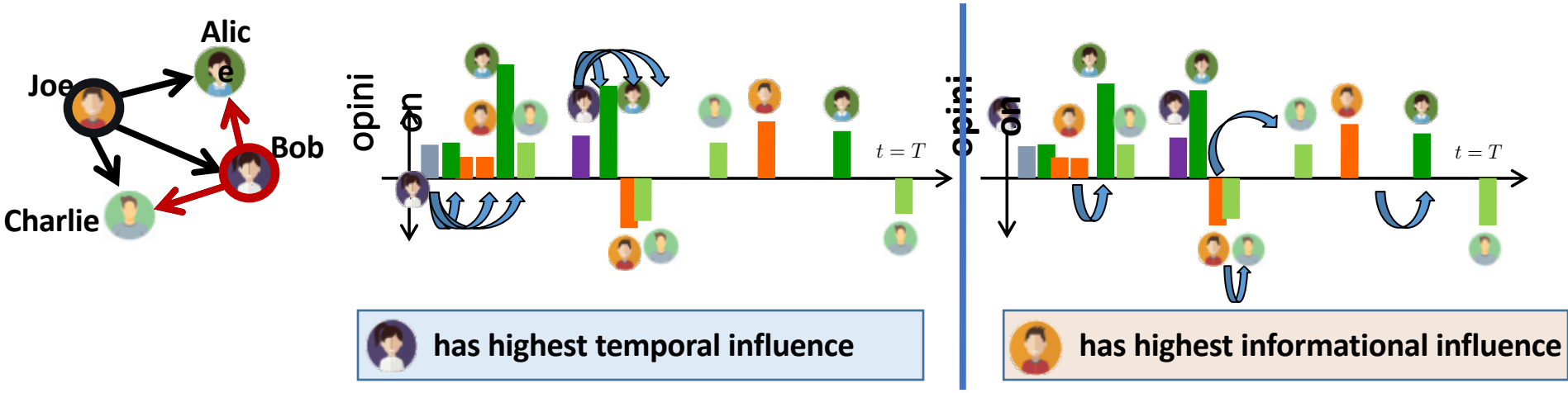
Opinion Dynamics : Is it new ?

There are a lot of **theoretical models of opinion dynamics,** but…

1. **Opinions are updated sequentially in discrete time**

Voter Model, DeGroot Model,
Flocking Model + Too many models!!

2. **Difficult to learn from fine-grained data and thus inaccurate predictions**
3. **Models informational dynamics of opinion flow — ignores temporal influence.**



A Recent Approach: SLANT [NIPS '16]

$$x_u^*(t) = \alpha_u + \sum_{v \in \mathcal{N}(u)} a_{vu} \sum_{e_i \in \mathcal{H}_v(t)} m_i g(t - t_i)$$

User's initial opinion

Previous expressed opinions by user's neighbors

Informational Influence

Times: $t_i \sim Hawkes(\lambda_u(t))$

Temporal Influence

Limitations of SLANT

1. Linearity of influence
2. Fixed parameterized representations
3. Semi-coupled dynamics of temporal and informational influence

Covered by SIAM-News

$$\lambda_u^*(t) = f_u(\cup_{v \in \mathcal{N}(u)} \mathcal{H}_v(t))$$

Fixed, linear, semi-coupled

Set of past events

$$x_u^*(t) = g_u(\cup_{v \in \mathcal{N}(u)} \mathcal{H}_v(t))$$

Proposed Approach

- ~~Linearity of influence~~

  Nonlinear influence structure

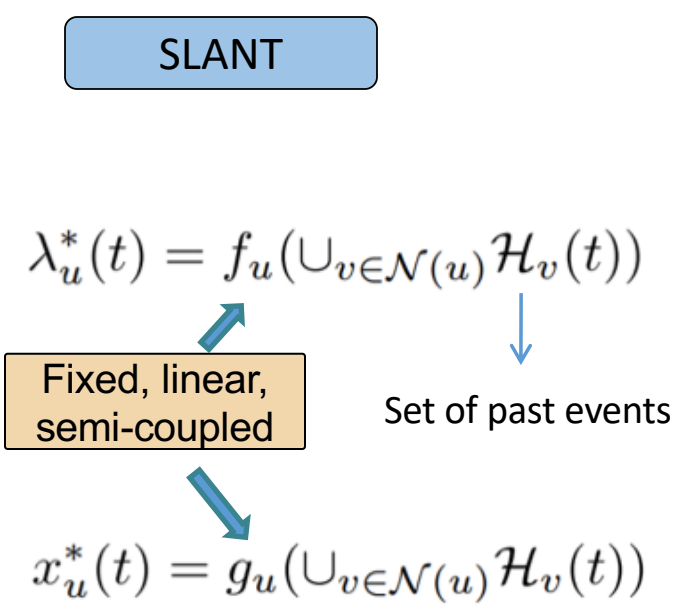
- ~~Fixed parameterized representations~~

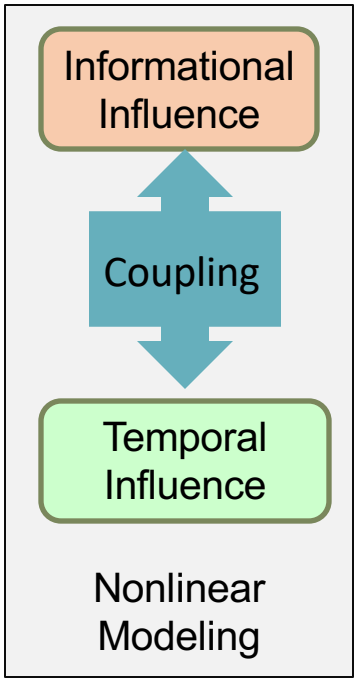
  Generalized representation **yet driven by data**

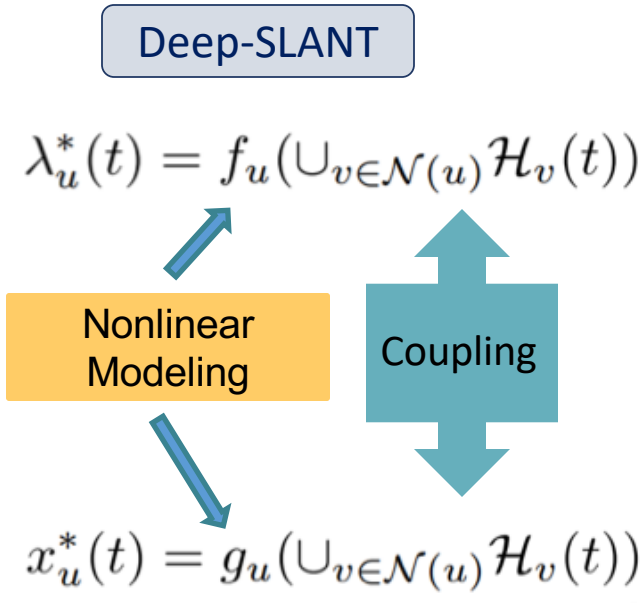
- ~~Semi-coupled dynamics of temporal and informational influence~~

  Coupled influence

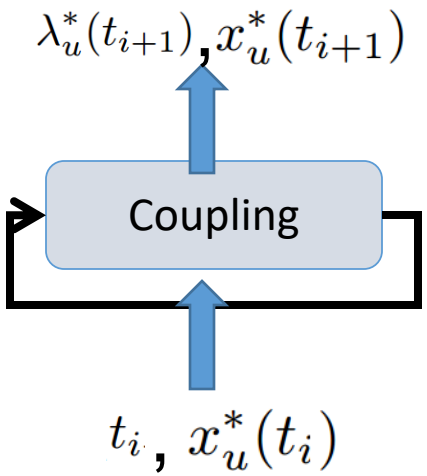
SLANT+: A nonlinear departure from SLANT







SLANT+: An intuitive approach

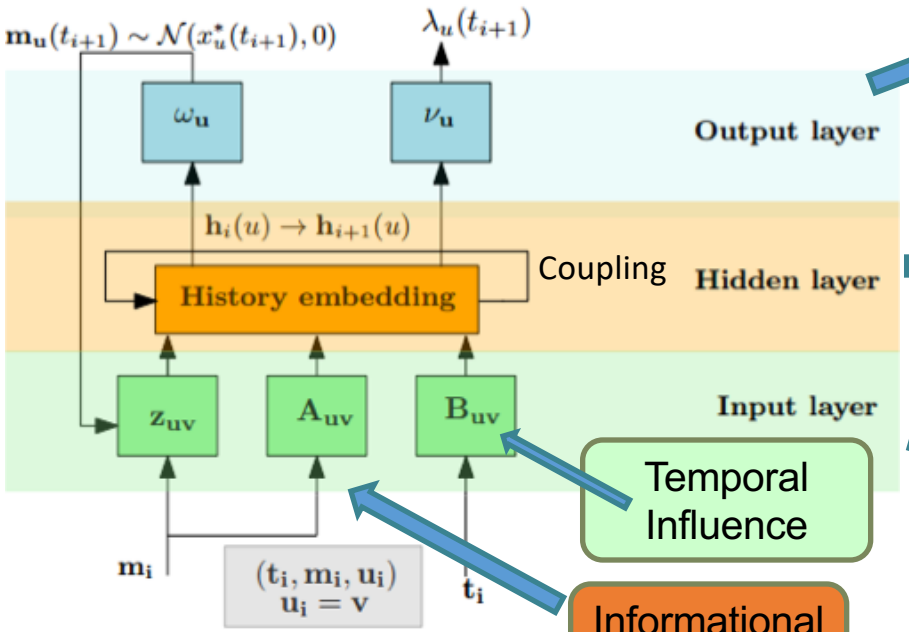


Capturing nonlinearity via recurrent neural network

A single RNN for whole network	Small RNNs per user
1. Expressivity issues 2. Training issues	1. Explainable 2. Decentralized

RMTPP, Du et al. 16, In KDD 2016

Nonlinear Modeling:
A networked guided RNN approach



Opinion and message generation

$$x_u^*(t) = \tanh(\nu_u^h h_{i+1} + \nu_u^t(t - t_i) + k^t)$$
$$\lambda_u^*(t) = \exp(\omega_u^h h_{i+1} + \omega_u^t(t - t_i) + b^t)$$

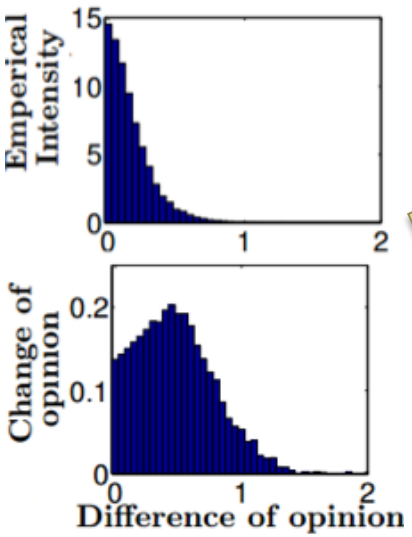
Embedding history: **captures coupling**

$$h_{i+1}(u) = \sigma_u\left(w_u^h h_i(u) + \boldsymbol{w_u^y}.\boldsymbol{y_u^i} + \boldsymbol{w_u^\theta}.\boldsymbol{\theta_u^i}\right)$$

Capturing Influence dynamics

$$\boldsymbol{y_u^i} = \boldsymbol{A}_{uu_i} m_i + \boldsymbol{\alpha}_u, \quad \boldsymbol{\theta_u^i} = \boldsymbol{b}_{uu_i} t_i + \boldsymbol{\beta}_u$$

Explainability: A data driven construction



Analysis on conversations on Delhi Election 2015

Construction of disagreement function from data

$$z_{uu_i}(t_i) = (w^1_{\sigma_u}\sigma(w^{(1)}_{uv}.|\Delta m_{uu_i}|) + w^2_{\sigma_u}\sigma(w^{(2)}_{uv}.|\Delta m_{uu_i}|))_+$$

Curation of embedding functions

$$h_{i+1}(u) = \sigma_u\left(w^h_u h_i(u) + \boldsymbol{w}^{\boldsymbol{y}}_u.\boldsymbol{y}^i_u + \boldsymbol{w}^{\boldsymbol{\theta}}_u.\boldsymbol{\theta}^i_u + w^z\sum_v z_{uv}(t_i)\right)$$

- Temporal data is usually limited, Twitter only allows 1% samples
- A prior embedding structure helps to learn from limited data.
  — works best for users who makes few comments

Datasets

We evaluate our model on several real-world stories:

Delhi Assembly Election, 12/2013

The Avengers: Age of Ultron, 05/2015

Mayweather vs Pacquiao, 05/2015

Salman Khan hit and run verdict , 05/2015

Games of Thrones , 05/2016

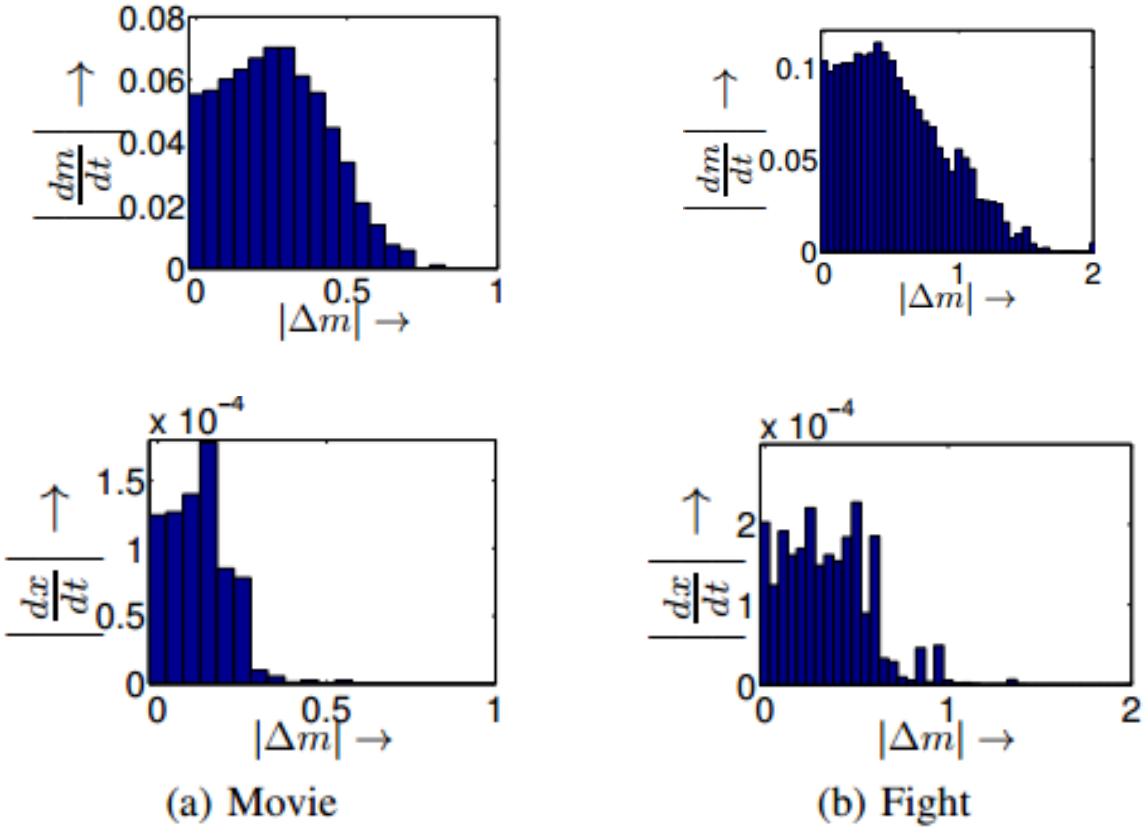
Experimental Results

	Mean Squared Error						
Dataset	SLANT+	SLANT	BVoter	Voter	AsLM	DeGroot	Flocking
Movie	0.007 (90.79)	0.076	0.755	0.822	1.367	0.499	0.69
Politics	0.038 (82.16)	0.213	0.771	0.670	1.023	0.875	0.76
Fight	0.045 (79.82)	0.223	1.351	1.477	1.514	0.963	1.31
Bollywood	0.049 (88.71)	0.434	2.015	2.132	3.579	1.724	1.94
Series	0.049 (32.88)	0.073	0.287	0.536	0.796	0.533	0.49
	Failure Rate						
Movie	0.00 (–)	0.0	0.0	0.0	0.0	0.0	0.0
Politics	0.03 (80.0)	0.15	0.51	0.51	0.51	0.46	0.58
Fight	0.06 (53.85)	0.13	0.59	0.59	0.54	0.43	0.54
Bollywood	0.01 (93.33)	0.15	0.43	0.44	0.50	0.42	0.43
Series	0.01 (66.67)	0.03	0.31	0.41	0.33	0.47	0.48

Disagreement fitting

Actual disagreement

Estimated Disagreement



(a) Movie (b) Fight

Summary

Dynamics of and on temporality

Joint nonlinear generative model of temporal and informational dynamics

RNN based approach of a continuous system without loss of information

1. Knowledge acquisition dynamics

2. Popularity dynamics of hashtags

Applications

Sources

- Deep Learning for NLP - Lecture October 2015 by Nils Reimers. https://github.com/UKPLab/deeplearning4nlp-tutorial/tree/master/2015-10_Lecture
- CMU CS 11-747, Fall 2017 **Neural Networks for NLP** by Graham Neubig http://www.phontron.com/class/nn4nlp2017/schedule.html
- Computational Neuroscience Seminar (MTAT.03.292) by Aqeel Labash https://courses.cs.ut.ee/MTAT.03.292/2017_fall/uploads/Main/Attention%20is%20All%20you%20need.pdf
- Learning Nonlinear Opinion Dynamics in Social Networks". B. Kulkarni, S. Agarwal, A. De, S. Bhattacharya, and N. Ganguly. *Accepted in IEEE International Conference on Data Mining (ICDM '17)*. Short paper. New Orleans, USA, 2017