# Recurrent (Neural) Networks: Part 1

Tomas Mikolov, Facebook AI Research

Deep Learning course @ NYU, 2015

# Structure of the talk

- Introduction
- Architecture of simple recurrent nets
- Training: Backpropagation through time
- Application to language modeling
- Research directions
- Resources

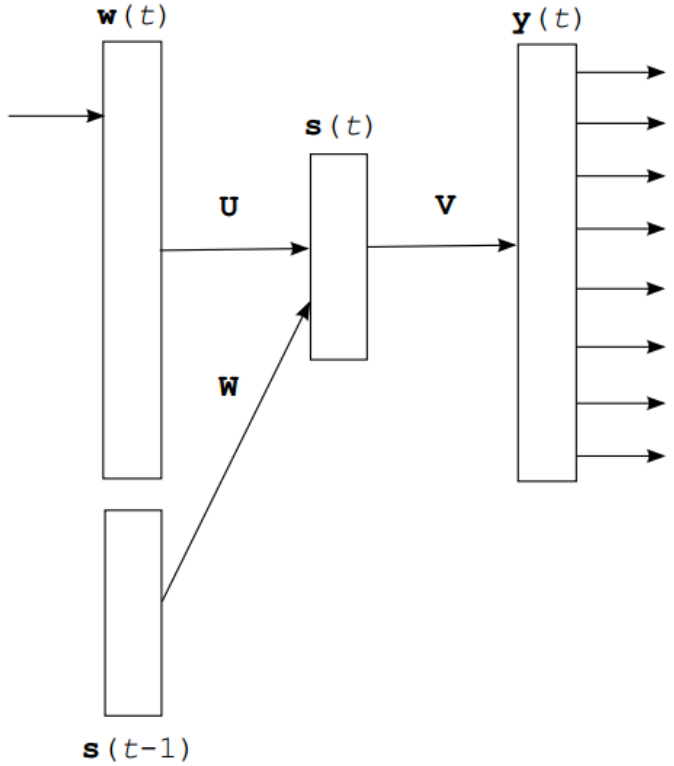
# Introduction

- Building models of sequential data is important: automatic speech recognition, machine translation, natural language, …

- Recurrent nets (RNN) are a simple and general framework for this type of tasks

# Architecture: simple RNN

- Input layer, hidden layer with recurrent connections, and the output layer
- In theory, the hidden layer can learn to represent unlimited memory
- Also called Elman network (*Finding structure in time*, Elman 1990)



# Architecture: simple RNN

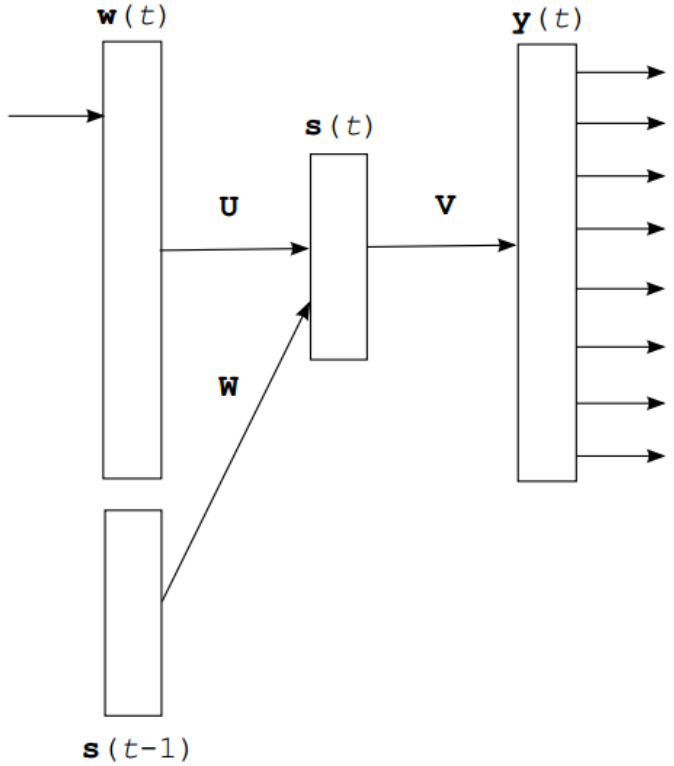
$$\mathbf{s}(t) = f(\mathbf{U}\mathbf{w}(t) + \mathbf{W}\mathbf{s}(t-1))$$
$$\mathbf{y}(t) = g(\mathbf{V}\mathbf{s}(t))$$

$f()$ is often sigmoid activation function:

$$f(z) = \frac{1}{1+e^{-z}}$$

$g()$ is often the softmax function:

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$

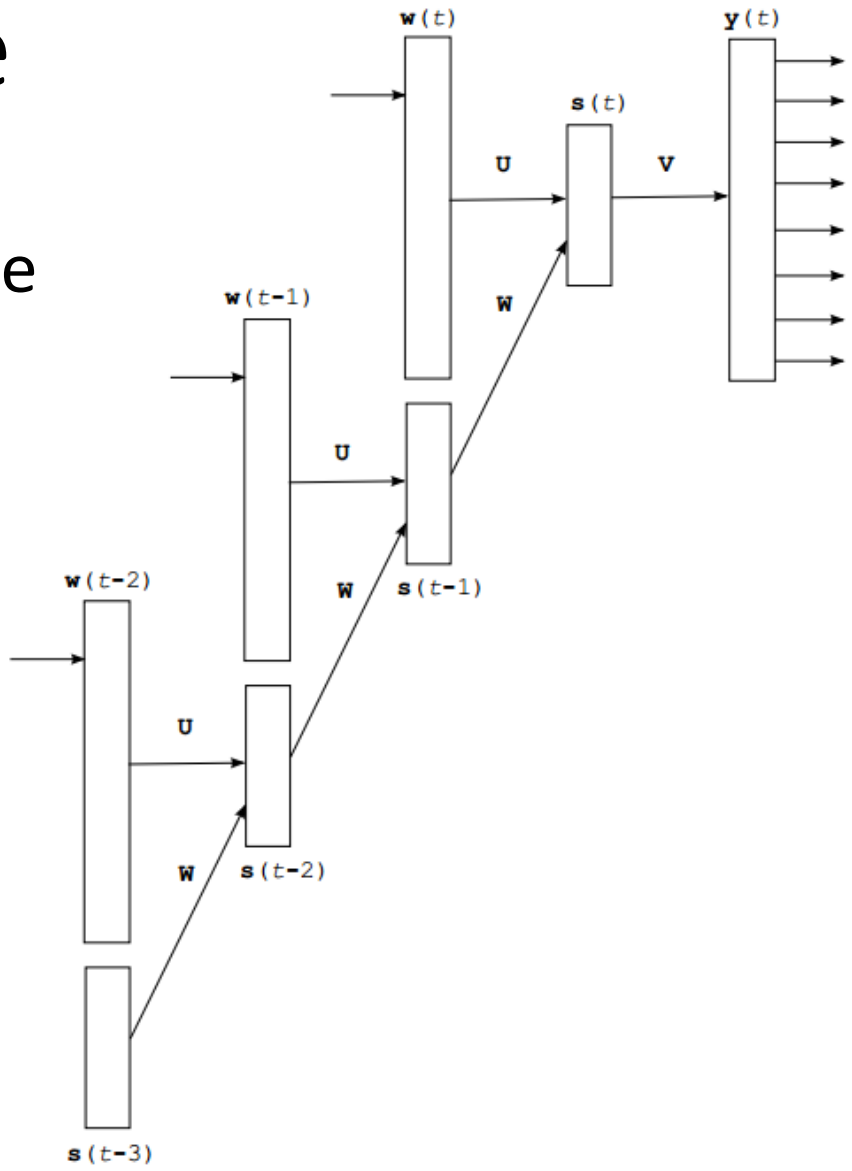


# Training: Backpropagation through time (BPTT)

- How to train the recurrent nets?
- The output value does depend on the state of the hidden layer, which depends on all previous states of the hidden layer (and thus, all previous inputs)
- Recurrent net can be seen as a (very deep) feedforward net with shared weights

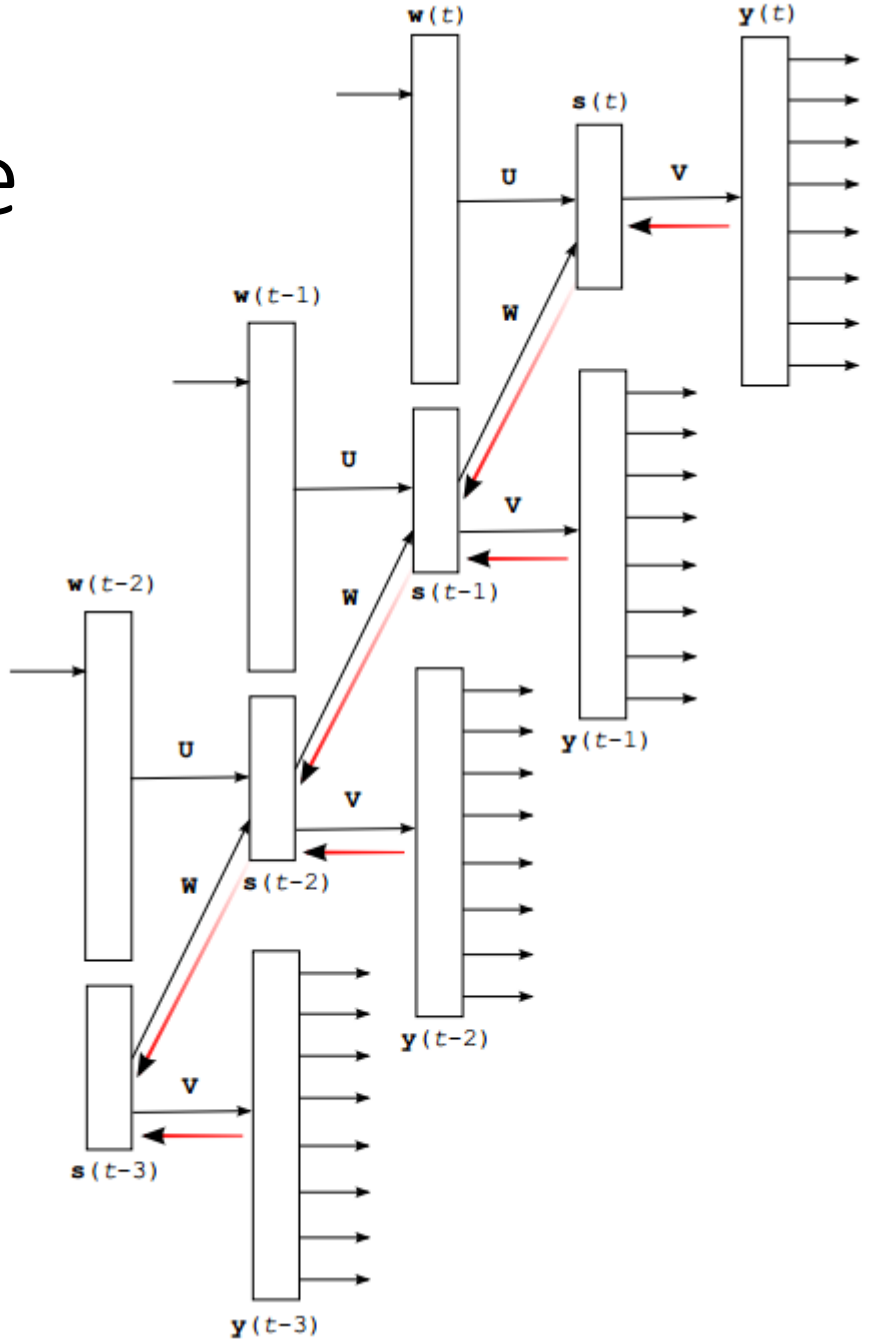
# Backpropagation through time

- The intuition is that we unfold the RNN in time
- We obtain deep neural network with shared weights **U** and **W**
- Unfolding for several steps is often enough (called *Truncated BPTT*)



# Backpropagation through time

- We train the unfolded RNN using normal backpropagation + SGD
- In practice, we limit the number of unfolding steps to 5 – 10
- It is computationally more efficient to propagate gradients after few training examples (batch training)



# Vanishing gradients

- As we propagate the gradients back in time, usually their magnitude quickly decreases: this is called “vanishing gradient problem”

- In practice this means that learning long term dependencies in data is difficult for simple RNN architecture

- Special RNN architectures address this problem:
  - *Exponential trace memory* (Jordan 1987, Mozer 1989)
  - *Long Short-term Memory* (Hochreiter & Schmidhuber, 1997))
  - will be described in the second part of this lecture

# Exploding gradients

- Sometimes, the gradients start to increase exponentially during backpropagation through the recurrent weights
- Happens rarely, but the effect can be catastrophic: huge gradients will lead to big change of weights, and thus destroy what has been learned so far
- One of the main reasons why RNNs were supposed to be unstable
- Simple solution (first published in RNNLM toolkit in 2010):
  clip or normalize values of the gradients to avoid huge changes of weights

# Application of RNNs: Language modeling

- Statistical language modeling is one of the oldest and most important NLP tasks
- The language models are core of machine translation, speech recognition and many other applications
- Historically, it was amazingly difficult to convincingly beat N-grams, especially on larger than tiny datasets

# Language modeling: example

```
A B C A B C A B _
```

- What symbol comes next?
- What is its probability?

```
Yesterday it was Sunday, so today it must be _
```

- How to predict the next word?
- What is this good for?

# N-grams

- Task: compute probability of a sentence *W*

$$P(W) = \prod_i P(w_i | w_1 \dots w_{i-1})$$

- Often simplified to trigrams:

$$P(W) = \prod_i P(w_i | w_{i-2,} w_{i-1})$$

- The probabilities are computed using counting on training data

# Language modeling with neural networks

- Main deficiency of N-grams is the exponential growth of number of parameters with the length of the context

- Neural networks address this problem by performing dimensionality reduction and parameter sharing

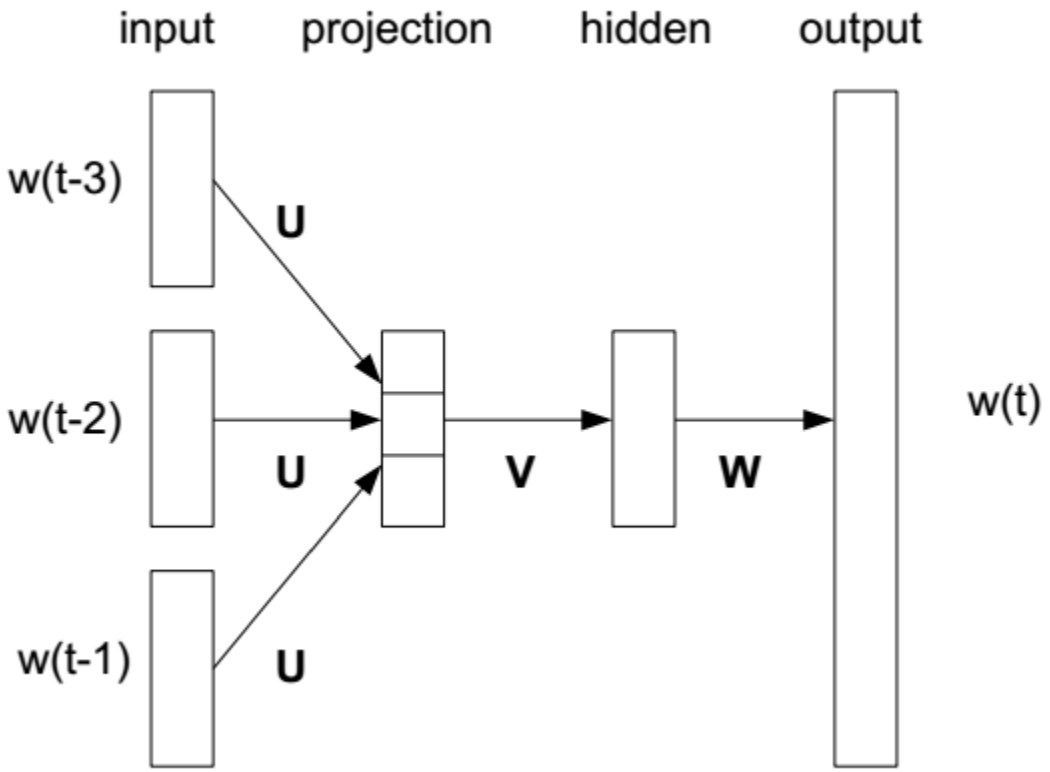
# Language modeling with neural networks

- Neural network language models are today state of the art, often applied to systems participating in competitions (ASR, MT)

- There are two main types of neural network architectures for language modeling: feedforward and recurrent

# Feedforward neural network LM

- Proposed by Bengio et al, 2003
- The projection layer is linear
- The hidden layer is non-linear
- Softmax at the output computes probability distribution over the whole vocabulary
- The basic architecture is computationally very expensive



# RNN language model

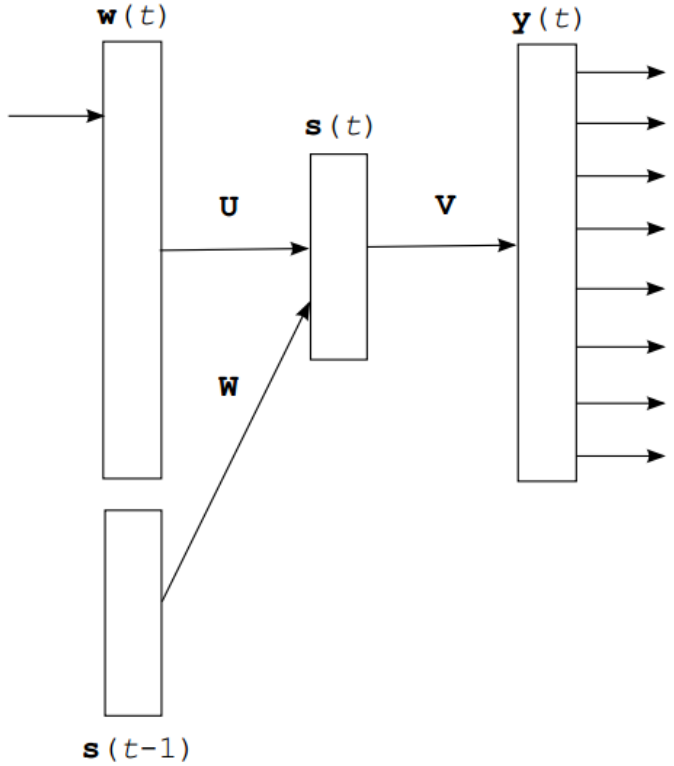
$$\mathbf{s}(t) = f(\mathbf{U}\mathbf{w}(t) + \mathbf{W}\mathbf{s}(t-1))$$
$$\mathbf{y}(t) = g(\mathbf{V}\mathbf{s}(t))$$

$f()$ is often sigmoid activation function:

$$f(z) = \frac{1}{1+e^{-z}}$$

$g()$ is often the softmax function:

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$



# Comparison of performance on small data: Penn Treebank

- Small, standard dataset, ~1M words
- Perplexity: lower is better
  (exp of average entropy per symbol)
- RNN outperforms FNN by about 10%

| Model | Perplexity |
|---|---|
| Kneser-Ney 5-gram | 141 |
| Maxent 5-gram | 142 |
| Random forest | 132 |
| Feedforward NNLM | 140 |
| Recurrent NNLM | 125 |

# Scaling up to large datasets

- While neural network LMs have been around for a while, their computational complexity complicated their use in real-world systems

Main computational bottlenecks:

1. Computation of probability distribution in the output layer
2. Dense matrix multiplication
   (FNN: projection -> hidden layer, RNN: recurrent matrix)

- Further, one can train RNNs on GPUs

# Softmax

- Softmax at the output layer computes probability distribution over the whole vocabulary:

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$

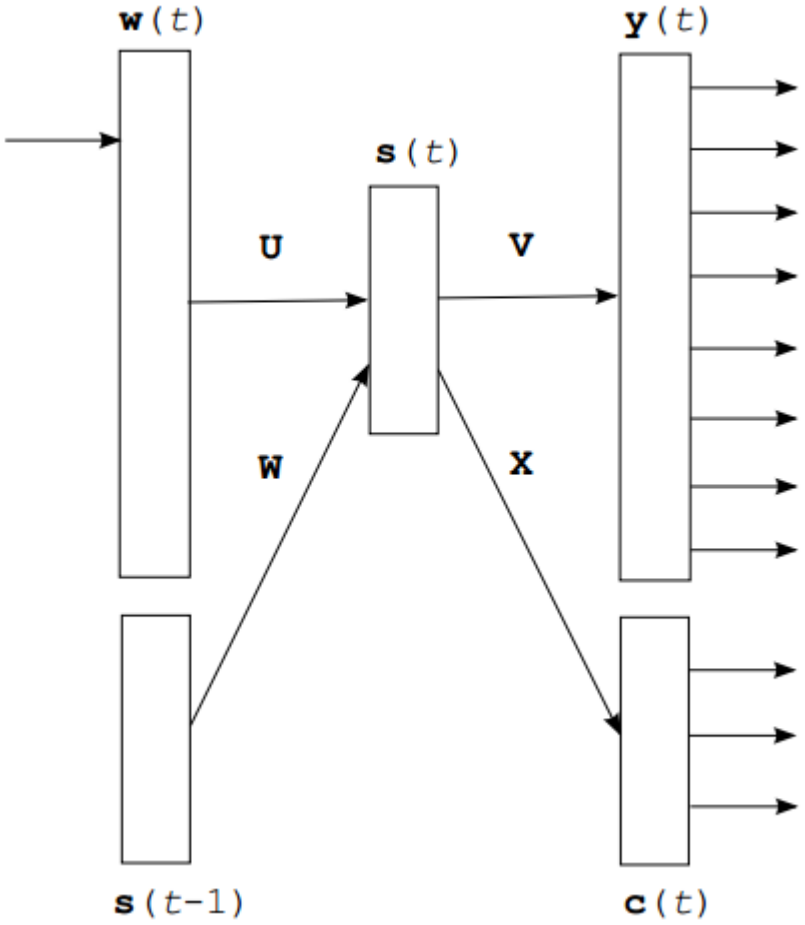
- Sums to 1, all output values are non-negative

# Scaling up: class based softmax

- Instead of normalizing probability over all words, we:

1. Assign each word to a single class
2. Normalize over the class layer
3. Normalize over words from within the current class

- Reduces complexity from |V| to about sqrt(|V|)



# How to assign words to classes?

Frequency binning:

- Simple, computationally efficient
- Slightly degrades accuracy

Brown classes:

- good accuracy
- has to be pre-computed, less efficient

Other clustering techniques work too (for example K-means on pre-trained word vectors, or combination of frequency + Brown).

# Joint training of RNN with Maxent model

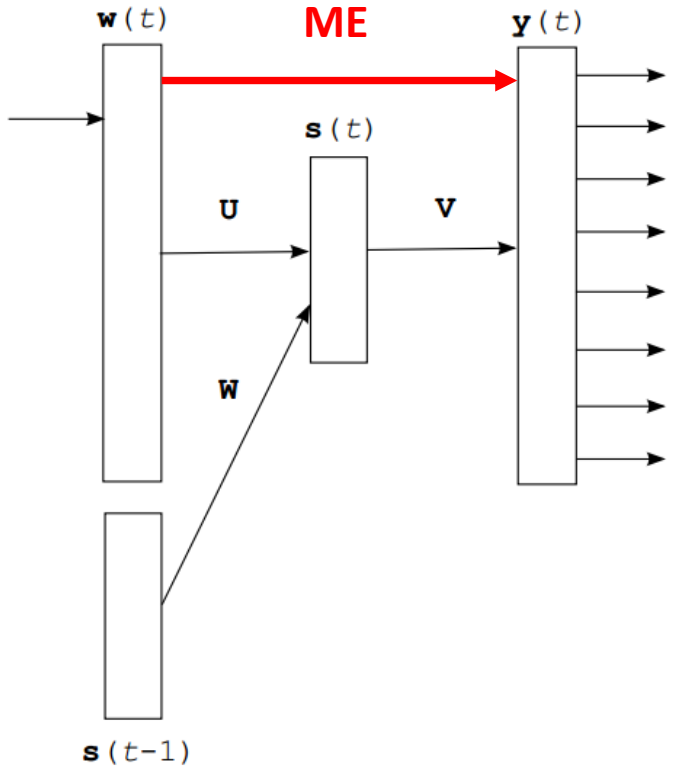
- Logistic regression and neural networks are closely related
- Maximum entropy model is how logistic regression is called in NLP

- Logistic regression: fast, scales to very large datasets
- Neural networks: more compact and robust, generalize better

- Why not combine both?

# Joint training of RNN with Maxent model

- Just another matrix of weights in the RNN
- This corresponds to RNN with bigram ME:
- We can use n-gram features for ME




*Strategies for training large scale neural network language models* (Mikolov et al, 2011)

# Joint training of RNN with Maxent model

Joint training allows:

- To use the fast, big sparse model (direct weights between inputs and outputs)

- The slow, dense part (hidden layer) can be much smaller

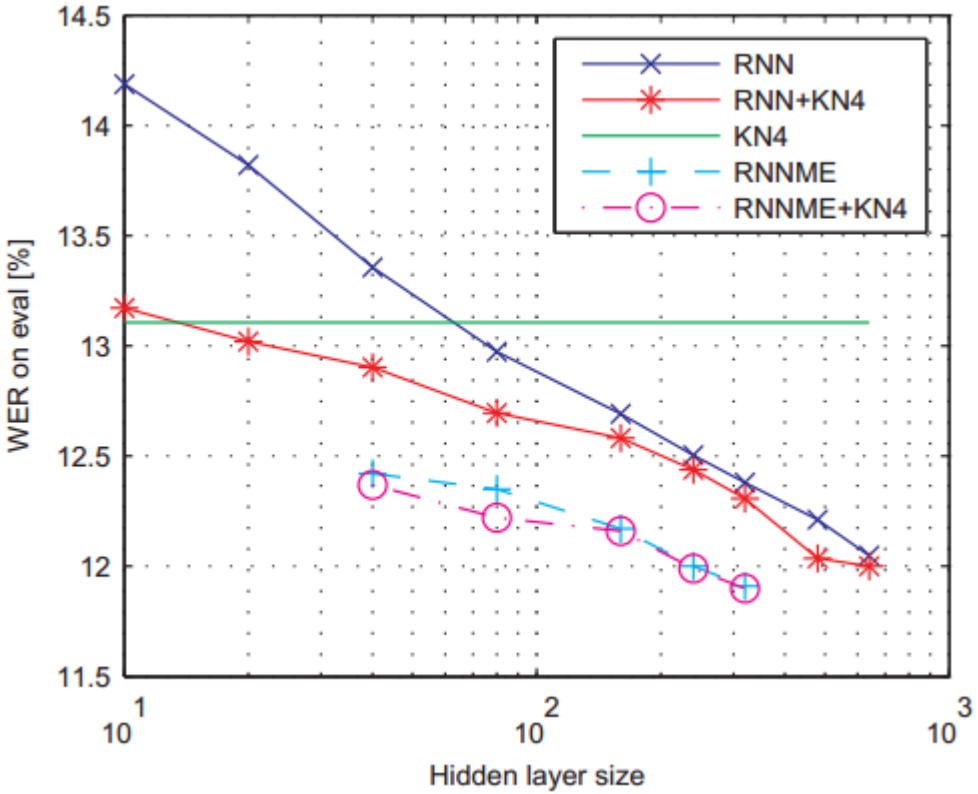
# Joint training of RNN with Maxent model

Why not train models separately and combine their predictions?

- If neural network is trained jointly with the maxent model, it can learn just the complementary information.

By using the direct connections, we can reduce the hidden layer size greatly and still achieve good performance.

# IBM RT04 speech recognition system




| Model | WER[%] |
|---|---|
| KN4 (baseline) | 13.11 |
| model M | 12.49 |
| 3xRNN | **11.70** |

- Strong ASR system from IBM, “model M“ was the previous state-of-art LM
- Bigger models = better results, RNNME works the best

# Multi-threaded training of RNNME

- Another speedup ~10x, scales training to billions of words

| Model | Training Time | | Perplexity |
|---|---|---|---|
| | [hours] | [CPUs] | |
| Interpolated KN 5-gram, 1.1B n-grams (KN) | 3 | 100 | 67.6 |
| Recurrent NN-256 + MaxEnt 9-gram | 60 | 24 | 58.3 |
| Recurrent NN-512 + MaxEnt 9-gram | 120 | 24 | 54.5 |
| Recurrent NN-1024 + MaxEnt 9-gram | 240 | 24 | 51.3 |

*One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling* (Chelba et al, 2014)

# GPU training of RNNLM

- Allows to use even larger models:

| Model | # Params [millions] | Training Time | Perplexity |
|---|---|---|---|
| KN 5-gram | 1,740 | 30m | 66.9 |
| RNN - 128 | 16.4 | 6h | 60.8 |
| RNN - 256 | 32.8 | 16h | 57.3 |
| RNN - 512 | 65.8 | 1d2h | 53.2 |
| RNN - 1024 | 132 | 2d2h | 48.9 |
| RNN - 2048 | 266 | 4d7h | 45.2 |
| RNN - 4096 | 541 | 14d5h | 42.4 |

*Scaling Recurrent Neural Network Language Models* (Williams et al, 2015)

# Language modeling summary

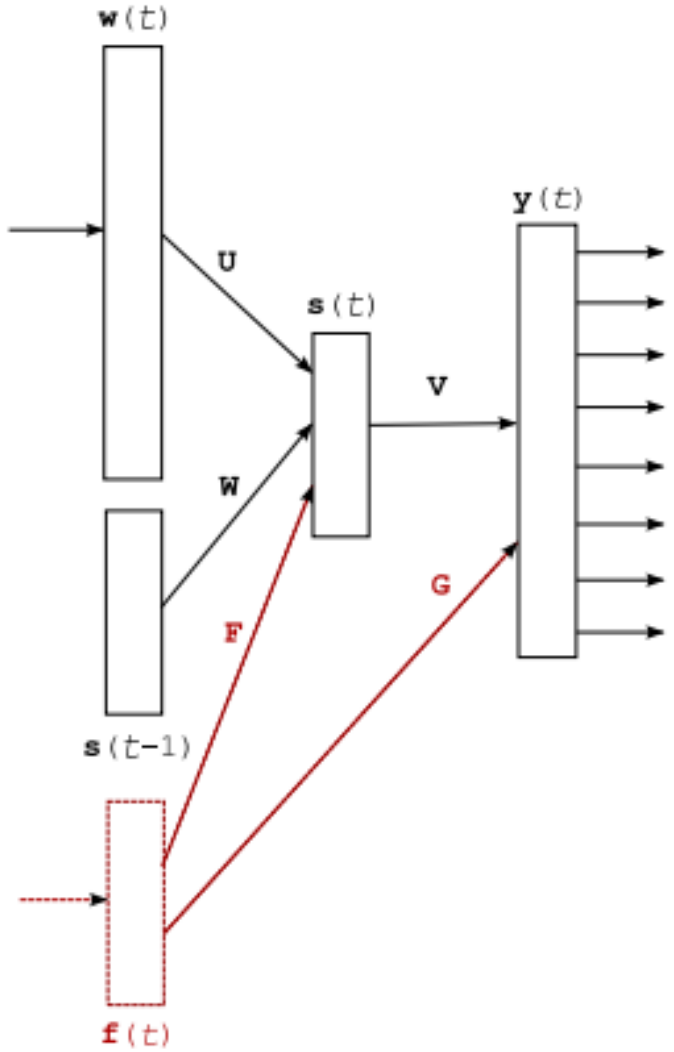
- RNN outperforms FNN on language modeling tasks, both are better than n-grams
- The question “are neural nets better than n-grams” is incomplete: the best solution is to use both
- Joint training of RNN and maxent with n-gram features works great on large datasets

# Extensions of RNNLM and application to MT

- RNN conditioned on extra features
- Simple solution to deal with the vanishing gradients
- Recent applications in MT

# Recurrent neural network with additional features

- We can use extra features (POS, external information, long context information, …) represented as additional inputs




*Context dependent recurrent neural network language model* (Mikolov & Zweig, 2012)

# Recurrent neural network with slow features

- We can define the extra features f(t) as exponentially decaying sum of word vectors w(t):

$$f(t) = f(t-1)\gamma + w(t)(1-\gamma)$$

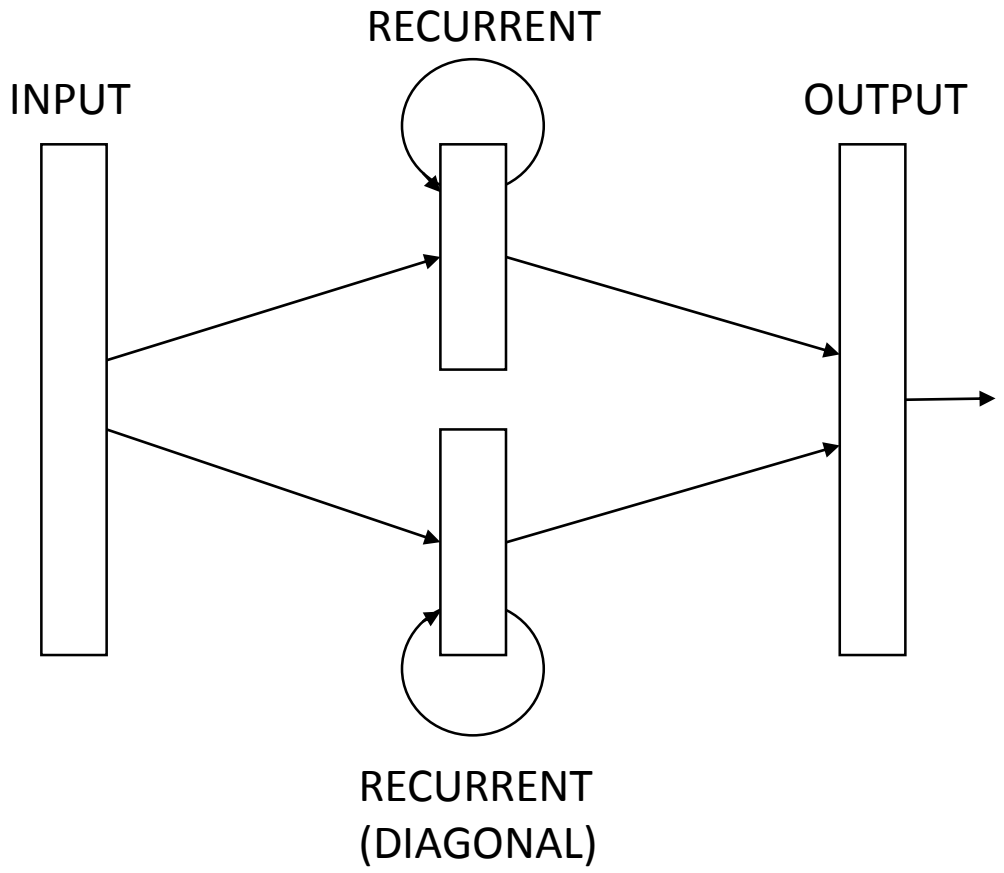
- This will give the model longer term memory

- Performance in language modeling similar to Long Short-Term Memory (LSTM) RNN – about 10% reduction of perplexity over simple RNN

LSTM-based LM will be described in the second talk today

# Recurrent neural network with slow features

- 2 hidden layers
- One usual, fully connected
- The other constrained (diagonal)
- The values in diagonal can be fixed (close to 1)




- Learns longer term patterns

# Application to machine translation

- Machine translation: input is a sequence of words in one language, output is the translated sentence in another language

- Example: Google Translate

# Application to machine translation

- Straightforward: N-best list rescoring with NNLMs on top of existing system

- Search space in MT is huge, integration into decoding is a better idea

| setting | dev | 2004 | 2005 | 2006 |
|---|---|---|---|---|
| baseline | 38.2 | 38.4 | 37.7 | 34.3 |
| reranking | 38.5 | 38.6 | 37.8 | 34.7 |
| decoding | 39.1 | 39.5 | 38.8 | 34.9 |

*Decoding with Large-Scale Neural Language Models Improves Translation*
(Vaswani et al, 2013)

# Application to machine translation

- Instead of conditioning the prediction of word just on the previous words within target sentence, we can use also the source sentence words

- We can use whole source sentence representation as additional input features

Modeling both translation and language model probability with NNLMs:
*Joint Language and Translation Modeling with Recurrent Neural Networks*
(Auli et al, 2013)

# Application to machine translation

| BOLT Test | | |
|---|---|---|
| | Ar-En | |
| | BLEU | Gain |
| "Simple Hier." Baseline | 33.8 | - |
| Standard NNJM | 40.2 | +6.4 |
| Self-Norm NNJM | 40.1 | +6.3 |
| Pre-Computed NNJM | 39.9 | +6.1 |

- Decoding with Joint NN models; BLEU metric = higher is better
- Additional inputs represent words in the source sentence around position that is currently decoded

*Fast and Robust Neural Network Joint Models for Statistical Machine Translation* (Devlin et al, 2014)

# Summary: Neural net Language Models

- NNLMs are currently the state-of-the-art in language modeling
- Considerable improvements in ASR, MT
- Significant ongoing efforts to scale training to very large datasets

# Summary: Neural net Language Models

- Much of the recent success is based on basic principles: big data, big models, algorithmic efficiency, neural nets from the 80's
- Neural network models *incrementally* extend existing paradigms: there is no single (important) task that RNNs can do that other models completely fail at
- So far there is no big success story of deep learning in NLP: maybe we will have to do something novel to make it work?

# Promising future research directions

- We should pick challenging problems that are really important by themselves

Examples:

- Language understanding by machines
- What computational models can represent human languages
- Learning language from increasingly complex examples, and through communication

# Future of AI research

Language understanding:

- Should aim to allow machines do what people can do
- Learn language through communication: intelligence is not about knowing answers to everything, but about ability to learn new concepts quickly

We need to revisit basic concepts in machine learning / NLP. Instead of doing just big data statistics, we need to develop new paradigms.

*Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks* (Weston et al, 2015)

# Resources

- RNNLM toolkit
- Links to large text corpora
- Benchmark datasets for advancing the state of the art

# RNNLM toolkit

- Available at rnnlm.org
- Allows training of RNN and RNNME language models
- Extensions are actively developed, for example multi-threaded version with hierarchical softmax: http://svn.code.sf.net/p/kaldi/code/trunk/tools/rnnlm-hs-0.1b/

# CSLM: Feedforward NNLM code

- Continuous Space Language Model toolkit: http://www-lium.univ-lemans.fr/cslm/

- Implementation of feedforward neural network language model by Holger Schwenk

# Other neural net SW

- List available at http://deeplearning.net/software_links/

- Mostly focuses on general machine learning tools

# Large text corpora

Short list available at the word2vec project: https://code.google.com/p/word2vec/#Where_to_obtain_the_training_data

- Sources: Wikipedia dump, statmt.org, UMBC webbase corpus

- Altogether around 8 billion words can be downloaded for free

# Benchmark datasets for language modeling

- The Penn Treebank setup including the usual text normalization is part of the example archive at rnnlm.org

- WSJ setup (simple ASR experiments, includes N-best list rescoring): http://www.fit.vutbr.cz/~imikolov/rnnlm/kaldi-wsj.tgz