Deep Learning for NLP: Word Vectors and Lexical Semantics

Ed Grefenstette
etg@google.com

UNIVERSITY OF OXFORD

DeepMind

How to represent words

Natural language text = sequences of **discrete symbols** (e.g. words).

Naive representation: one hot vectors in $R^{|vocabulary|}$ (very large).

Classical IR: document and query vectors are superpositions of word vectors.

$$\hat{d}_q = \arg\max_d sim(\mathbf{d}, \mathbf{q})$$

Similarly for **word classification problems** (e.g. document classification).

Issues: sparse, orthogonal representations, semantically weak.

How to represent words

We want **richer representations** expressing $\boxed{\textbf{semantic similarity}}$.

Distributional semantics:
"You shall know a word by the company it keeps." — J.R. Firth (1957)

Idea: produce **dense** vector representations based on the **context/use** of words.

Three main approaches: **count-based**, **predictive**, and **task-based**.

Count-based methods

Define a **basis vocabulary** *C* of context words.

Define a **word window** size *w*.

Count the basis vocabulary words occurring *w* words to the left or right of each instance of a **target word** in the corpus.

Form a **vector representation** of the target word based on these counts.

Count-based methods

... and the *cute* **kitten** *purred* and then ...

... the *cute furry* **cat** *purred* and *miaowed* ...

... that the *small* **kitten** *miaowed* and she ...

... the *loud furry* **dog** *ran* and *bit* ...

Example **basis vocabulary**: *{bit, cute, furry, loud, miaowed, purred, ran, small}*.

kitten context words: {cute, purred, small, miaowed}.

cat context words: {cute, furry, miaowed}.

dog context words: {loud, furry, ran, bit}.

Count-based methods

... and the *cute* **kitten** *purred* and then ...

... the *cute furry* **cat** *purred* and *miaowed* ...

... that the *small* **kitten** *miaowed* and she ...

... the *loud furry* **dog** *ran* and *bit* ...

Example **basis vocabulary**: *{bit, cute, furry, loud, miaowed, purred, ran, small}*.

$$\mathbf{kitten} = [0, 1, 0, 0, 1, 1, 0, 1]^\top$$

$$\mathbf{cat} = [0, 1, 1, 0, 1, 0, 0, 0]^\top$$

$$\mathbf{dog} = [1, 0, 1, 1, 0, 0, 1, 0]^\top$$

Count-based methods

Use inner product or cosine as **similarity kernel**. E.g.:

$$sim(\text{kitten}, \text{cat}) = cosine(\mathbf{kitten}, \mathbf{cat}) \approx 0.58$$

$$sim(\text{kitten}, \text{dog}) = cosine(\mathbf{kitten}, \mathbf{dog}) = 0.00$$

$$sim(\text{cat}, \text{dog}) = cosine(\mathbf{cat}, \mathbf{dog}) \approx 0.29$$

Reminder: $$cosine(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \times \|\mathbf{v}\|}$$

Cosine has the advantage that it's a *norm-invariant* metric.

Count-based methods

Not all features are equal: we must distinguish counts that are high because the are *informative* from those that are just *independently frequent contexts*.

Many **normalisation methods**: TF-IDF, PMI, etc.

Some **remove the need** for norm-invariant similarity metrics.

But... perhaps there are easier ways to address this problem of count-based methods (and others, e.g. choice of basis context).

Neural Embedding Models

Learning count based vectors produces an $\boxed{\textbf{embedding matrix}}$ in $R^{|\text{vocab}|\times|\text{context}|}$:

$$\mathbf{E} = \begin{array}{c} \\ \text{kitten} \\ \text{cat} \\ \text{dog} \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccc} \text{bit} & \text{cute} & \text{furry} & \dots \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & \dots \\ 0 & 1 & 1 & \dots \\ 1 & 0 & 1 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \end{array}$$

Rows are word vectors, so we can retrieve them with **one hot vectors** in $\{0,1\}^{|\text{vocab}|}$:

$$onehot_{cat} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad \mathbf{cat} = onehot_{cat}^{\top}\mathbf{E}$$

Symbols = unique vectors. Representation = embedding symbols with **E**.

Neural Embedding Models

(One) generic idea behind embedding learning:

1. Collect instances $t_i \in inst(t)$ of a word t of vocab V.
2. For each instance, collect its context words $c(t_i)$ (e.g. k-word window).
3. Define some score function $score(t_i, c(t_i); \boldsymbol{\theta}, \mathbf{E})$ with upper bound on output.
4. Define a loss:

$$L = -\sum_{t \in V} \sum_{t_i \in inst(t)} score(t_i, c(t_i); \theta, \mathbf{E})$$

5. Estimate:

$$\hat{\theta}, \hat{\mathbf{E}} = \arg\min_{\theta, \mathbf{E}} L$$

6. Use the estimated **E** as your embedding matrix.

Neural Embedding Models

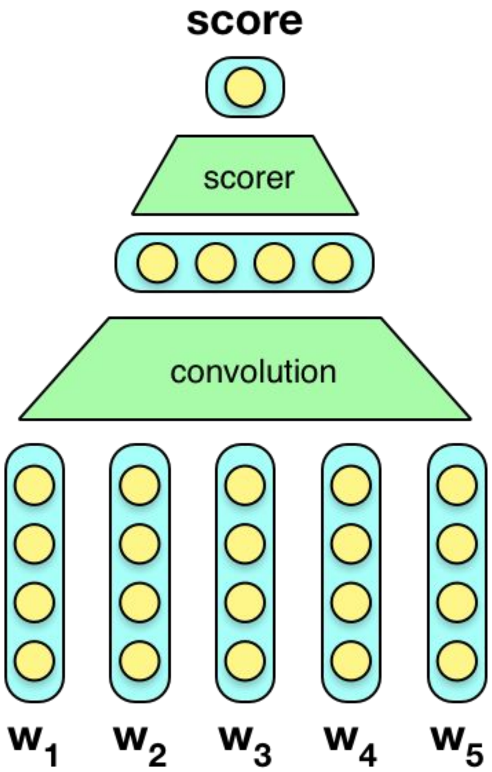
Scoring function matters!

Easy to design a **useless scorer** (e.g. ignore input, output upper bound).

Ideally, scorer:

- Embeds t_i with **E** (obviously).
- Produces a score which is a function of how well t_i is accounted for by $c(t_i)$, and/or vice versa.
- Requires the word to account for the context (or the reverse) more than another word in the same place.
- Produces a loss which is differentiable w.r.t. $\boldsymbol{\theta}$ and **E**.

Neural Embedding Models: C&W (Collobert *et al.* 2011)



Embed all words in a sentence with **E**.

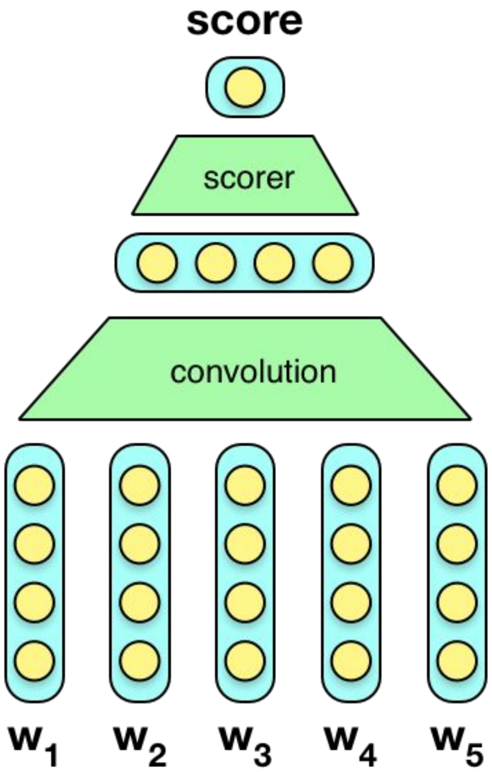
Shallow convolution over embeddings.

MLP projects output of convolution to a scalar score.

Convolutions and MLP are parameterised by a set of weights $\boldsymbol{\theta}$.

Overall network models a function over sentences s: $g_{\theta,E}(s) = f_{\theta}(embed_{E}(s))$

Neural Embedding Models: C&W (Collobert *et al.* 2011)



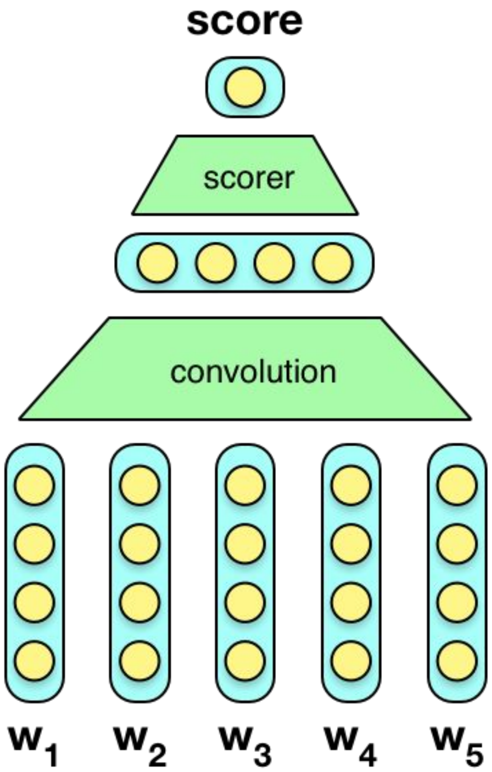
What prevents the network from **ignoring input** and outputting high scores?

During training, for each sentence s we sample a distractor sentence z by **randomly corrupting** words of s.

Minimise **hinge loss**:

$$L = max(0, 1 - (g_{\theta,\mathbf{E}}(s) - g_{\theta,\mathbf{E}}(z)))$$

Neural Embedding Models: C&W (Collobert *et al.* 2011)



Interpretation: representations carry information about what **neighbouring representations** should look like.

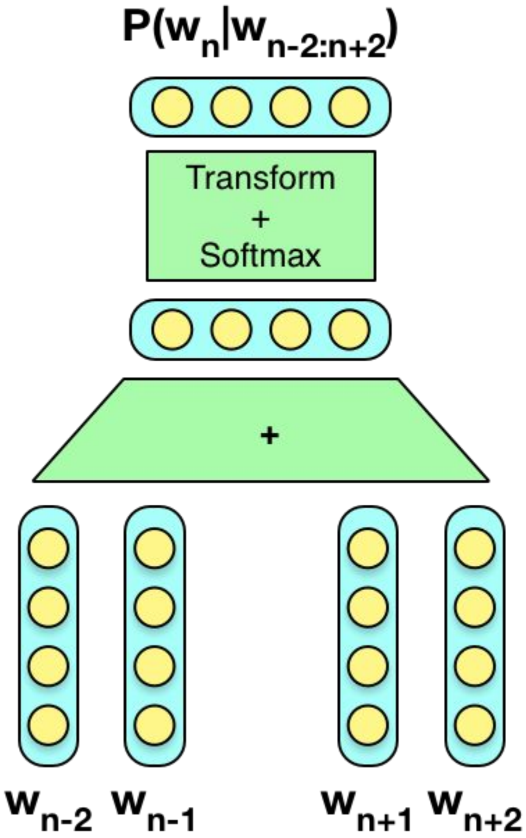
A lot like the **distributional hypothesis**?

A sensible model but:

Fairly **deep**, so not cheap to train.

Convolutions capture very **local** information.

Neural Embedding Models: CBoW (Mikolov *et al.* 2013)



Embed context words. Add them.

Project back to vocabulary size. Softmax.

$$softmax(\mathbf{l})_i = \frac{e^{l_i}}{\sum_j e^{l_j}}$$

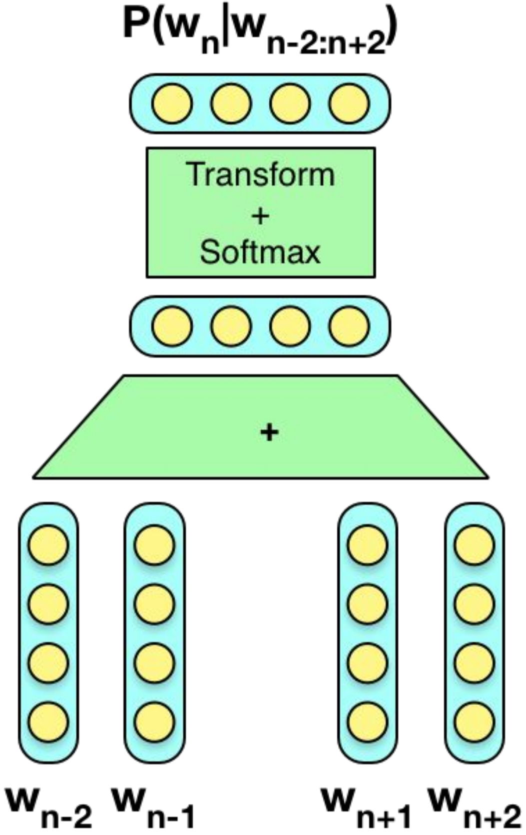
$$P(t_i|context(t_i)) = softmax\left(\sum_{t_j \in context(t_i)} onehot_{t_j}^{\top}\ \mathbf{E}\ W_v\right)$$

$$= softmax\left(\left(\sum_{t_j \in context(t_i)} onehot_{t_j}^{\top}\ \mathbf{E}\right) W_v\right)$$

Minimize Negative Log Likelihood:

$$L_{data} = -\sum_{t_i \in data} logP(t_i|context(t_i))$$

Neural Embedding Models: CBoW (Mikolov *et al.* 2013)



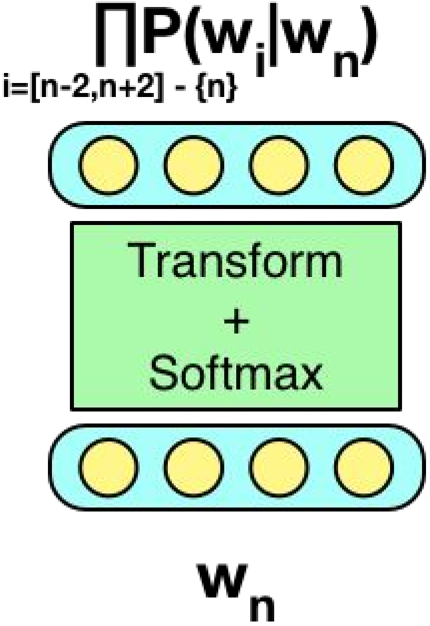
All linear, so very fast. Basically a cheap way of applying one matrix to all inputs.

Historically, negative sampling used instead of expensive softmax.

NLL minimisation is more stable and is fast enough today.

Variants: position specific matrix per input (Ling et al. 2015).

Neural Embedding Models: Skip-gram (Mikolov *et al.* 2013)



Target word predicts context words.

Embed target word.

Project into vocabulary. Softmax.

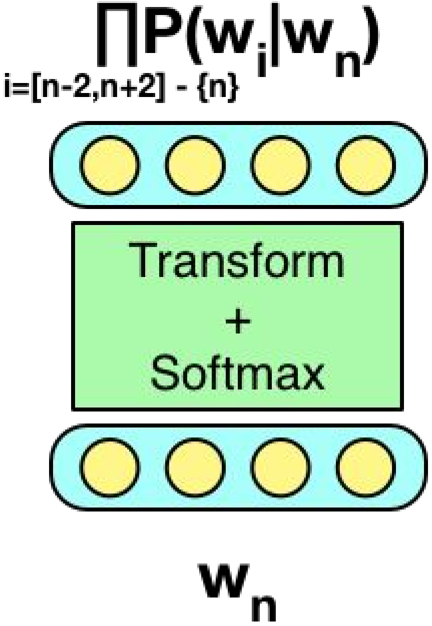
$$P(t_j|t_i) = softmax(onehot_{t_i}^{\top}\ \mathbf{E}\ W_v)$$

Learn to estimate likelihood of context words.

$$-logP(context(t_i)|t_i) = -log \prod_{t_j \in context(t_i)} P(t_j|t_i)$$

$$- \sum_{t_j \in context(t_i)} logP(t_j|t_i)$$

Neural Embedding Models: Skip-gram (Mikolov *et al.* 2013)



Fast: One embedding versus |C| embeddings.

Just read off probabilities from softmax.

Similar variants to CBoW possible: position specific projections.

Trade off between efficiency and more structured notion of context.

Comparison with Count-Based Models

Count based and objective-based models: same **general idea.**

Word2Vec == PMI matrix factorization of count based models
(Levy and Goldberg, 2014)

Count based and most neural models have **equivalent performance** when properly hyperoptimized (Levy *et al*. 2015)

Specific Benefits of Neural Approaches

Easy to learn, especially with good linear algebra libraries.

Highly parallel problem: minibatching, GPUs, distributed models.

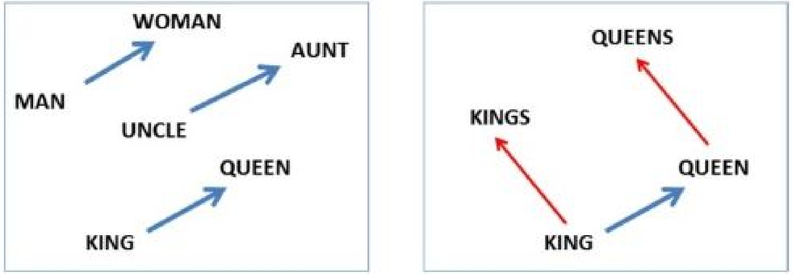
Can predict other **discrete aspects** of context (dependencies, POS tags, etc). Can estimate these probabilities with counts, but sparsity quickly becomes a problem.

Can predict/condition on **continuous** contexts: e.g. images.

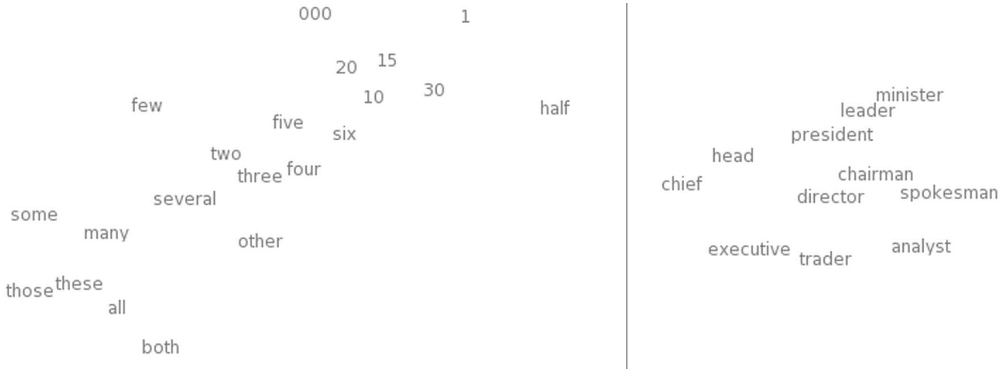
Evaluating Word Representations

Intrinsic Evaluation:

- WordSim-353 (Finkelstein *et al.* 2003)
- SimLex-999 (Hill *et al.* 2016, but has been around since 2014)
- Word analogy task (Mikolov *et al.* 2013), **queen = king - man + woman**
- Embedding visualisation (nearest neighbours, T-SNE projection)



Source: http://nlp.yvespeirsman.be/blog/visualizing-word-embeddings-with-tsne/



Source: http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

Evaluating Word Representations

Extrinsic Evaluation:

- Simply: do your embeddings improve performance on other task(s).
- More on this later...

Task-based Embedding Learning

Just saw methods for learning **E** through minimising a loss.

One use for **E** is to get input features to a neural network from words.

Neural network parameters are updated using gradients on loss L(x, y, **θ**):

$$\theta_{t+1} = update(\theta_t, \nabla_\theta L(x, y, \theta_t))$$

If **E** ⊆ **θ** then this update can modify **E** (if we let it):

$$\mathbf{E}_{t+1} = update(\mathbf{E}_t, \nabla_{\mathbf{E}} L(x, y, \theta_t))$$

Task-based Embedding Learning

We can therefore **directly train embeddings jointly with the parameters** of the network which uses them.

General intuition: learn to classify/predict/generate based on features, but also the **features themselves**.

Embeddings matrix can be **learned from scratch**, or **initialised** with pre-learned embeddings (fine-tuning).

Task-based Features: BoW Classifiers

We want to classify sentences/documents based on a variable number of word representations.

Simplest option: **bag of vectors**.

$$P(C|D) = softmax\left(W_c \sum_{t_i \in D} embed_{\mathbf{E}}(t_i)\right)$$

Projection into logits (input to softmax) can be arbitrarily complex. E.g.:

$$P(C|D) = softmax\left(W_c \ \sigma\left(W_h \sum_{t_i \in D} embed_{\mathbf{E}}(t_i)\right)\right)$$

Task-based Features: BoW Classifiers

Simple to implement and train.

Example tasks:

- Sentiment analysis (e.g. tweets, movie reviews).
- Document classification (e.g. 20 Newsgroups)
- Author identification, etc...

We learn **task-specific features**: e.g. notion of positive/negative words in sentiment analysis.

We can think of word meaning as being **grounded in the task**.

Task-based Features: BoW Classifiers

But, no notion of ***words in context*** (ambiguity, polysemy).

Cannot capture **saliency** of individual words. Everything contributes to the decision, so **more words = more noise**.

Grounding in classification tasks can yield quite **shallow semantics**. E.g.:

- There is more to the word "good" than the sentiment expressed.
- There is more to "CPU" than the fact that it predicts the topic "computer".

Task-based Features: Bilingual Features

Simple objectives can yield **better grounding** for word representations.

Example: recognising cross-lingual sentence alignment based on word vectors (Hermann and Blunsom 2014). Consider a dataset of English sentences and their German translations, $D=\{(e_i, g_i)\}_{i \in |D|}$

We want to produce representations $\mathbf{e}_i$ and $\mathbf{g}_i$ of the English and German sentence such the similarity between the vectors is maximised.

We use this objective to train embedding matrices $\mathbf{E}_e$ and $\mathbf{E}_g$, for English and German words.

Task-based Features: Bilingual Features

Sentence representations are produced with a **simple composition function**. You could do bag of words, or some aspect of word order, e.g.

$$\mathbf{e_i} = \sum_{t_j \in e_i} tanh(t_j + t_{j+1})$$

An obvious **loss** would be:

$$L = \sum_{(e_i, g_i) \in D} \|\mathbf{e}_i - \mathbf{g}_i\|^2$$

Obvious **degenerate solution** to this objective is:

$$\mathbf{E}_e = \mathbf{E}_g = \mathbf{0}^{|vocabSize| \times |embeddingSize|}$$

Task-based Features: Bilingual Features

So we avoid the degenerate case with a **noise-contrastive margin-based loss**:

$$L = \sum_{(e_i, g_i) \in D} max(-m, \|\mathbf{e}_i - \mathbf{g}_i\|^2 - \|\mathbf{e}_i - \mathbf{g}_j\|^2) \quad \text{where } j \sim \mathcal{U}(0, |D|) \text{ and } i \neq j$$

Sample a random German sentence per data point as **noise**. Impose the constraint that the **margin of similarity** between a paired pair of sentences and an unpaired pair of sentences be at least *m* (some hyperparameter).

Intuition: aligned sentences share **high-level meaning**, so embeddings should reflect the high-level meaning in order to minimise loss.

Task-based Features: Other Models

These models are all very simple (this is **not a bad thing™**). There are many other options.

How do we capture the relation between words? Disambiguation? The context they occur in? How do we use these embeddings effectively?

This is a recurring topic for the rest of this course.

Task-based Features: Interpretation

Task-based embeddings capture information **salient to the task**. Again, no guarantee this will capture "general" meaning beyond features useful for the task.

This can be overcome by using a **multi-task objective** but this comes with its own difficulties.

Alternatively, embeddings can be **pretrained and fixed**, relying on task-specific projections into the network, but is the pretraining objective general enough?

E.g. It might project "cat" and "kitten" into a similar part of the embedding space, but a task might need to radically differentiate these concepts.

Final Words

Learning and re-using word vectors is a form of **transfer learning**. It is particularly useful if you have little task-specific training data, or poor coverage of the vocabulary (in which case you might not want to fine-tune embeddings).

Generally speaking, if you have enough training data (and vocabulary coverage) you will benefit from training embeddings on the task, at the cost of reusability.

Take home message of this lecture:

Inputs to neural networks over text are embeddings.

We can train them separately, within a particular task, or both.

THANK YOU

Credits

Oxford Machine Learning and Computational Linguistics Groups

DeepMind Natural Language Research Group