Natural Language Understanding, Generation, and Machine Translation

Lecture 10: Word Embeddings

Frank Keller

School of Informatics
University of Edinburgh
keller@inf.ed.ac.uk

Week of 7 February 2022 (week 4)

The Story so Far

Pre-training and Finetuning

Word Embeddings

Static Word Embeddings

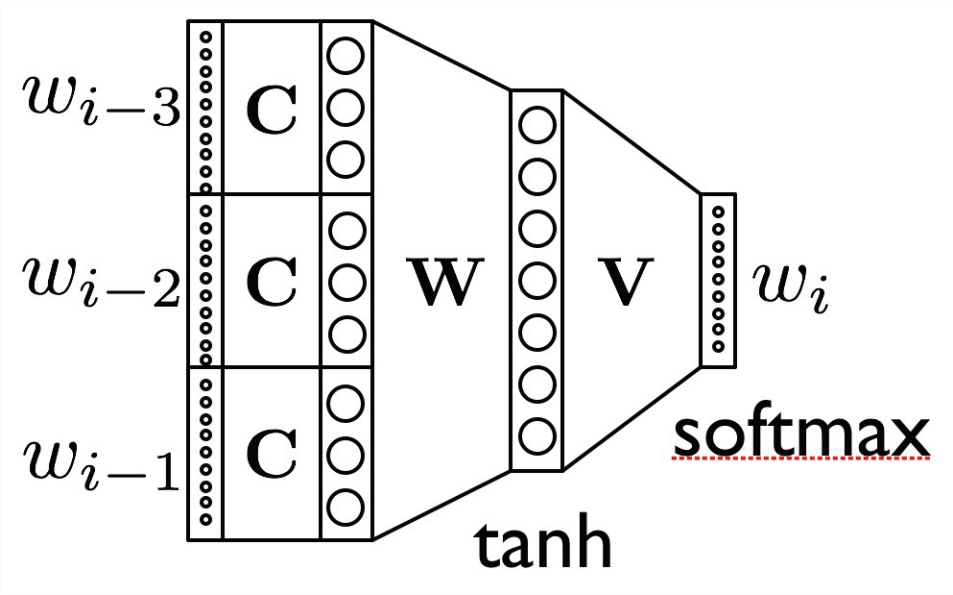
Reading: Mikolov et al. (2013), Smith (2019).

The Story so Far

Neural Language Models

We have seen how feed-forward networks can be used to estimate *n-gram probabilities:*

word embedding: C



$$P(w_i|w_{i-3}, w_{i-2}, w_{i-1}) = \text{softmax}(\mathbf{V}\mathbf{h}_2 + \mathbf{b}_2)$$

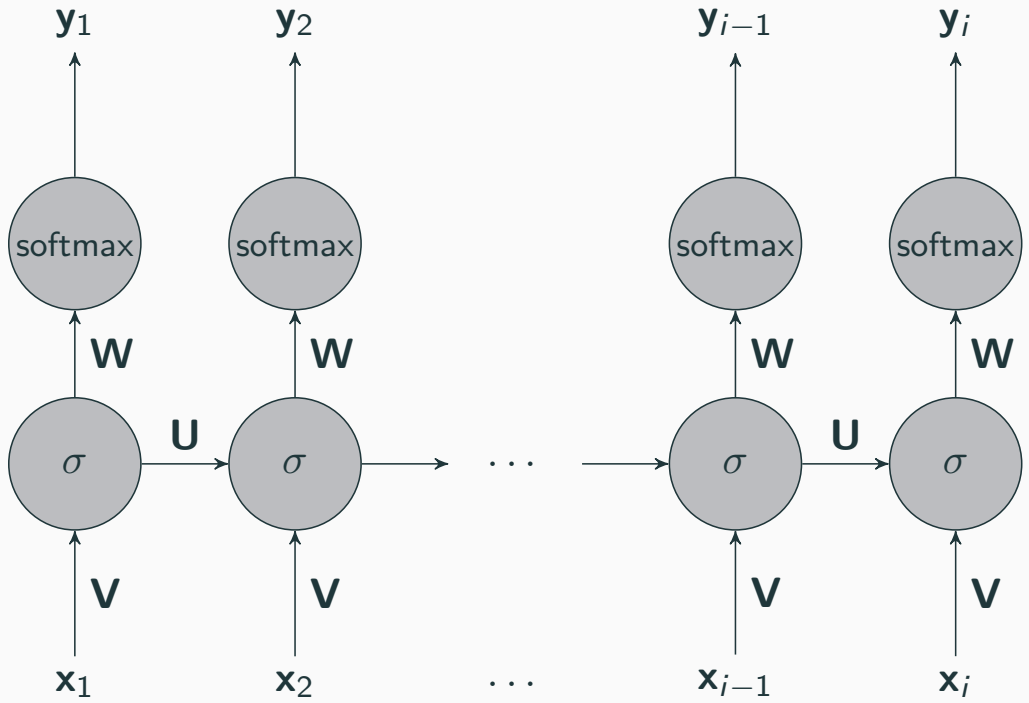
$$\mathbf{h}_2 = \tanh(\mathbf{W}\mathbf{h}_1 + \mathbf{b}_1)$$

$$\mathbf{h}_1 = \mathbf{C}\mathbf{w}_{i-3}; \mathbf{C}\mathbf{w}_{i-2}; \mathbf{C}\mathbf{w}_{i-1}$$

$$\mathbf{w}_i = \text{onehot}(w_i)$$

Neural Language Models

In recurrent neural networks, we drop the *Markov assumption:*



$$P(x_{i+1}|x_1,\ldots,x_{i-1}) = \mathbf{y}_i$$

$$\mathbf{y}_i = \text{softmax}(\mathbf{W}\mathbf{h}_i + \mathbf{b}_2)$$

$$\mathbf{h}_i = \sigma(\mathbf{V}\mathbf{x}_i + \mathbf{U}\mathbf{h}_{i-1} + \mathbf{b}_1)$$

$$\mathbf{x}_i = \text{onehot}(x_i)$$

Word Embeddings

- Each hidden layer of a neural language model is a representation of its input.
- Multiplying the one-hot vector $\mathbf{w}_i$ of word w_i by the weight matrix $\mathbf{C}$ selects a row of $\mathbf{C}$ (a vector).
- We call this vector the *word embedding* of w_i.
- Learned word embeddings have replace hand-engineered features word features in most NLP tasks.

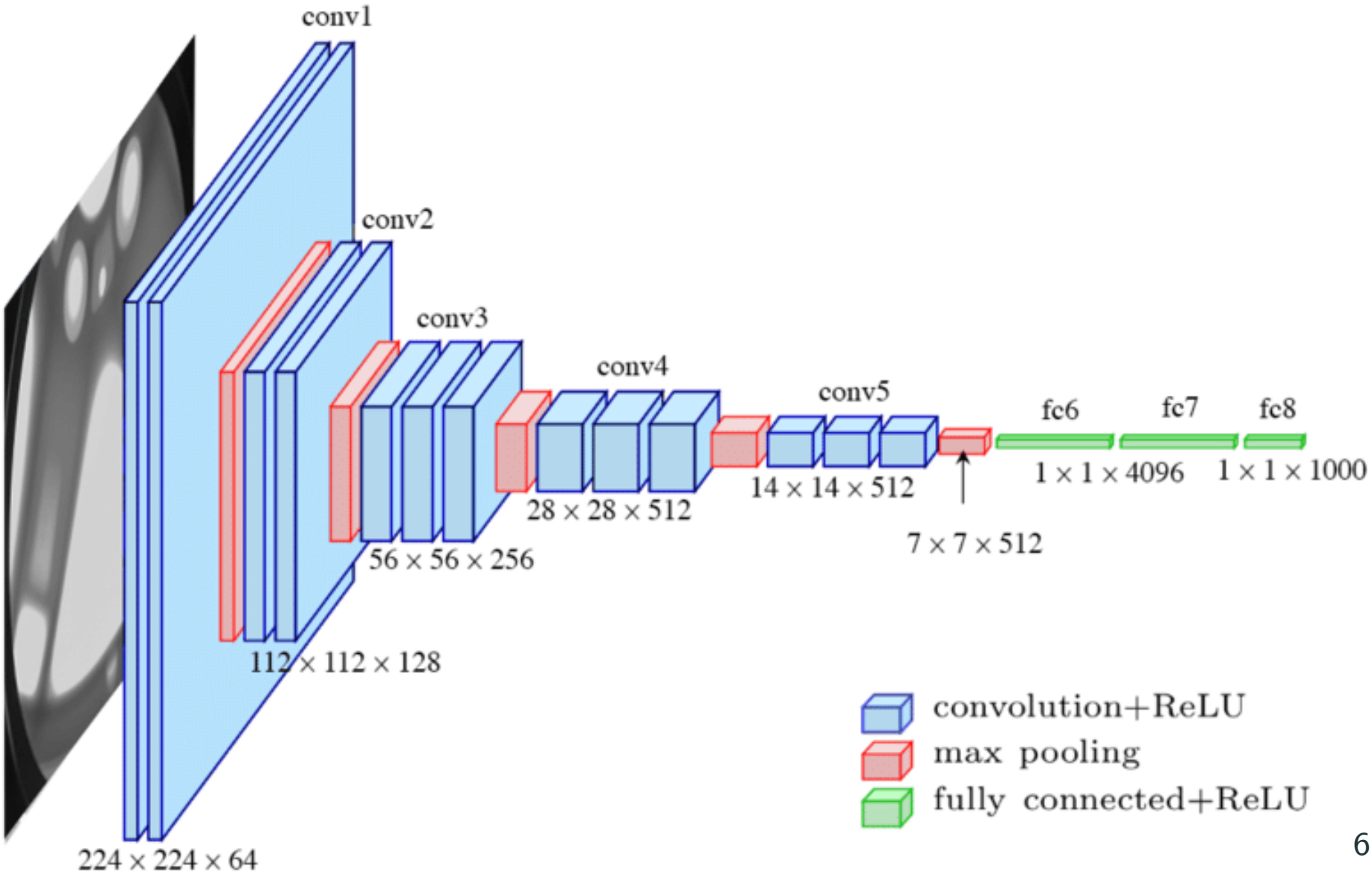
This lecture: Where do word embeddings come from, and how are they used? How do they relate to *transfer learning?*

First up: Once upon a time in computer vision . . .

Pre-training and Finetuning

VGG-16: A Typical Object Detection Model

(Simonyan and Zisserman 2015)

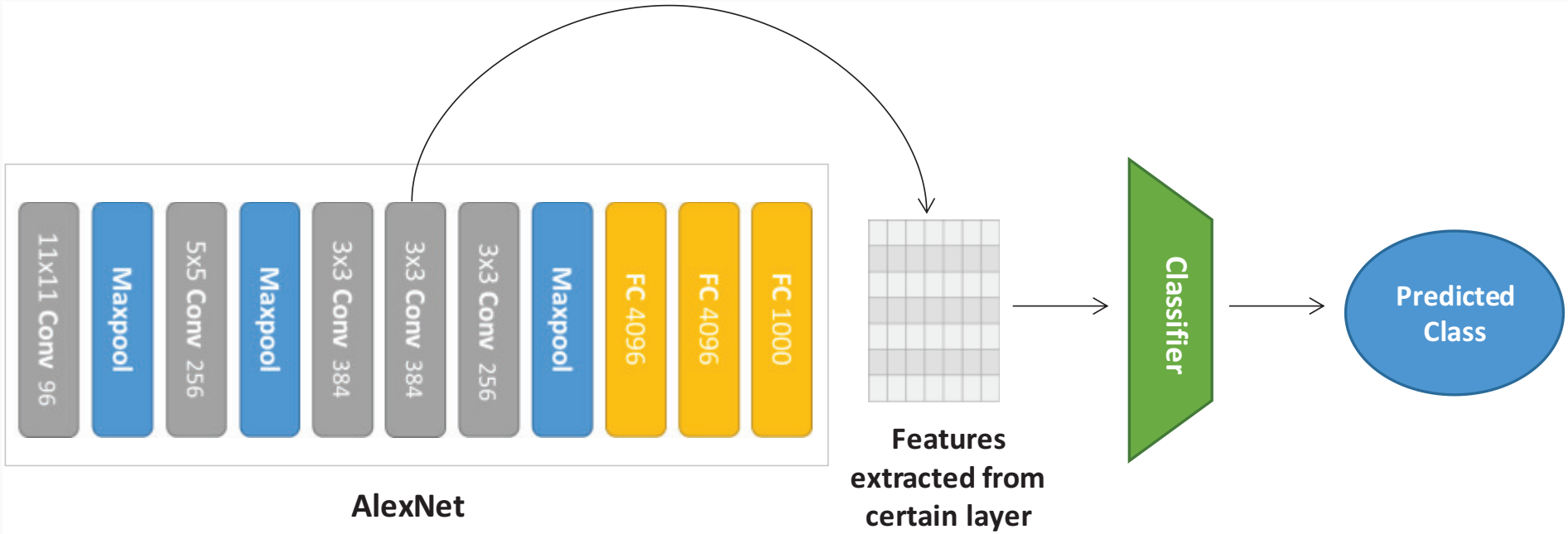


Feature Extraction

- A model like VGG-16 has a lot of layers.
- VGG-16 is already pretty dated; current object detection models can have hundreds of layers.
- They take forever to train and require very large training sets.
- However, computer vision people found that you can take the output of the last layer (here: fc8) as a feature vector.
- For a new task, use VGG-16 to turn an image into features, and then train a classifier for a new task on these features.

Transfer learning by feature extraction.

Feature Extraction



Pre-training and Finetuning

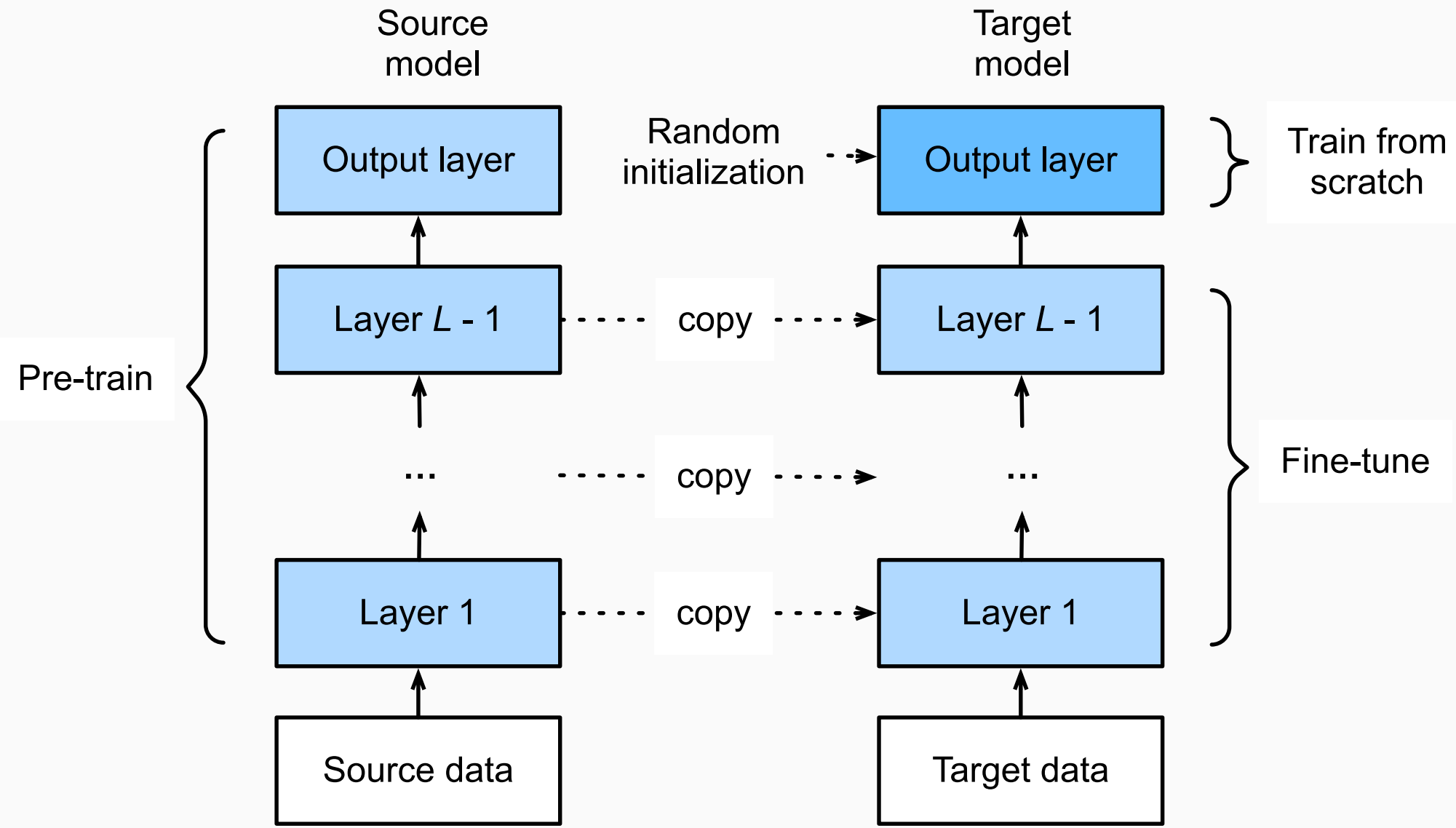
Instead of just extracting features, we can retrain the model for a new task using new data:

- *Pre-training:* train a generic **source model** (e.g., VGG-16) on a standard, large dataset (e.g., ImageNet).
- *Finetuning:* then take the resulting model, keep its parameters, and replace the output layer to suit the new task. Now train this **target model** on the dataset for the new task.

Transfer learning by finetuning.

You can think of pre-training as a way of *initializing the parameters* of your target model to good values.

Pre-training and Finetuning



Pre-training and Finetuning

Things to bear in mind when finetuning:

- Often, you truncate the last layer when you replace it (you have typically have fewer output classes).
- It's often a good idea to use a smaller learning rate when fine-tuning (you have already initialized to good values!).
- Typically, you only finetune a handful of final layers, you keep the other ones fixed: *weight freezing.*
- Finetuning without weight freezing is normally too slow.

Pre-training and Finetuning in NLP

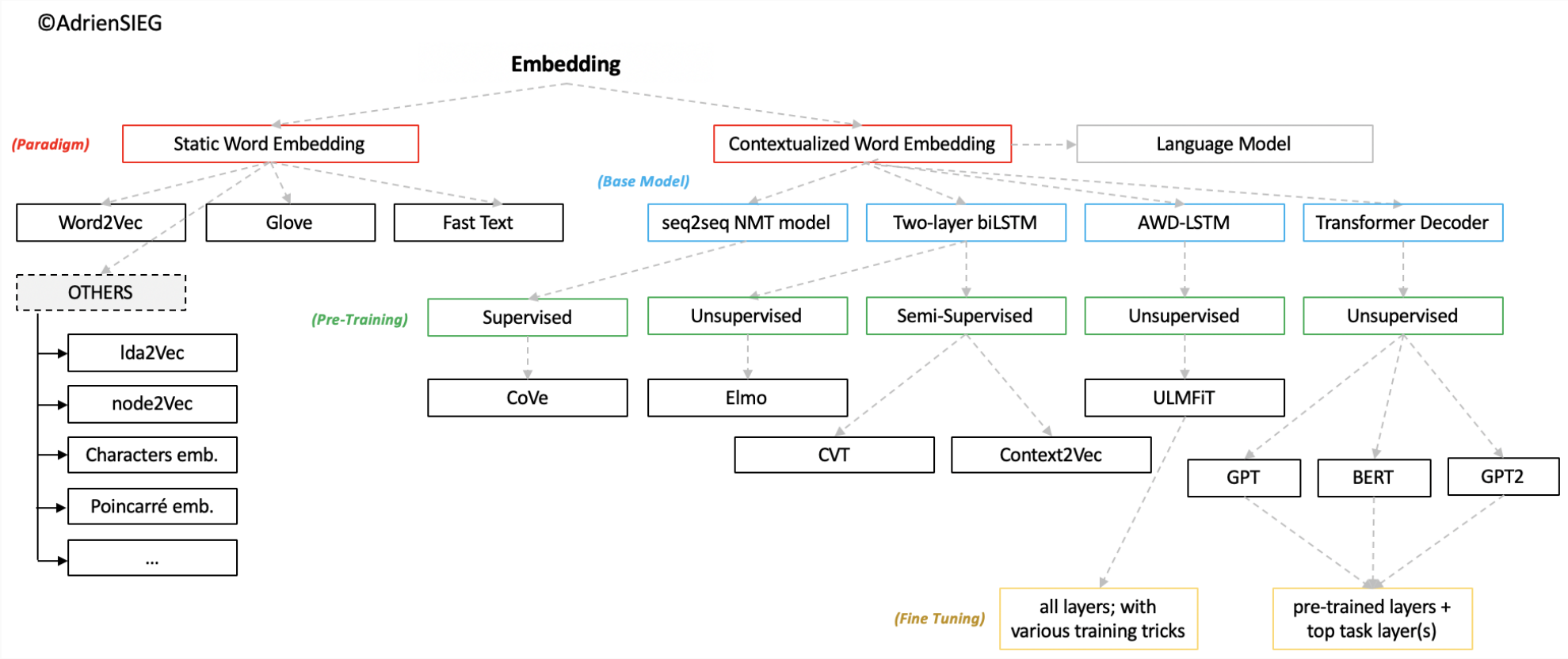
What does this have to do with NLP?

- When we use word embeddings as the input representations for a new task, then we're doing feature extraction.
- Our **source model** is the neural language model that the embeddings come from.
- The **target model** is often very different from the NLM, as our target task is rarely next word prediction.
- However, we can allow the weights of the embedding layer of the target model to be trained.
- This is a limited form of finetuning.

However, can we do full-scale *finetuning for NLP?* This became possible with *contextualized word embeddings.*

Word Embeddings

Overview of Word Embeddings



https://towardsdatascience.com/from-pre-trained-word-embeddings-to-pre-trained-language-models-focus-on-bert-34

Static Word Embeddings

- They assign a fixed vector to each word, independent of its context.
- They are used for feature extraction, i.e., to initialize the embedding layer of a target model.
- They are not designed to be used for finetuning.
- They are often efficient enough so that training from scratch is possible.
- Word2Vec, Glove, FastText are typical examples.
- Embeddings are available to download for many languages.

In this lecture: We will introduce *Word2Vec* as an example of static word embeddings.

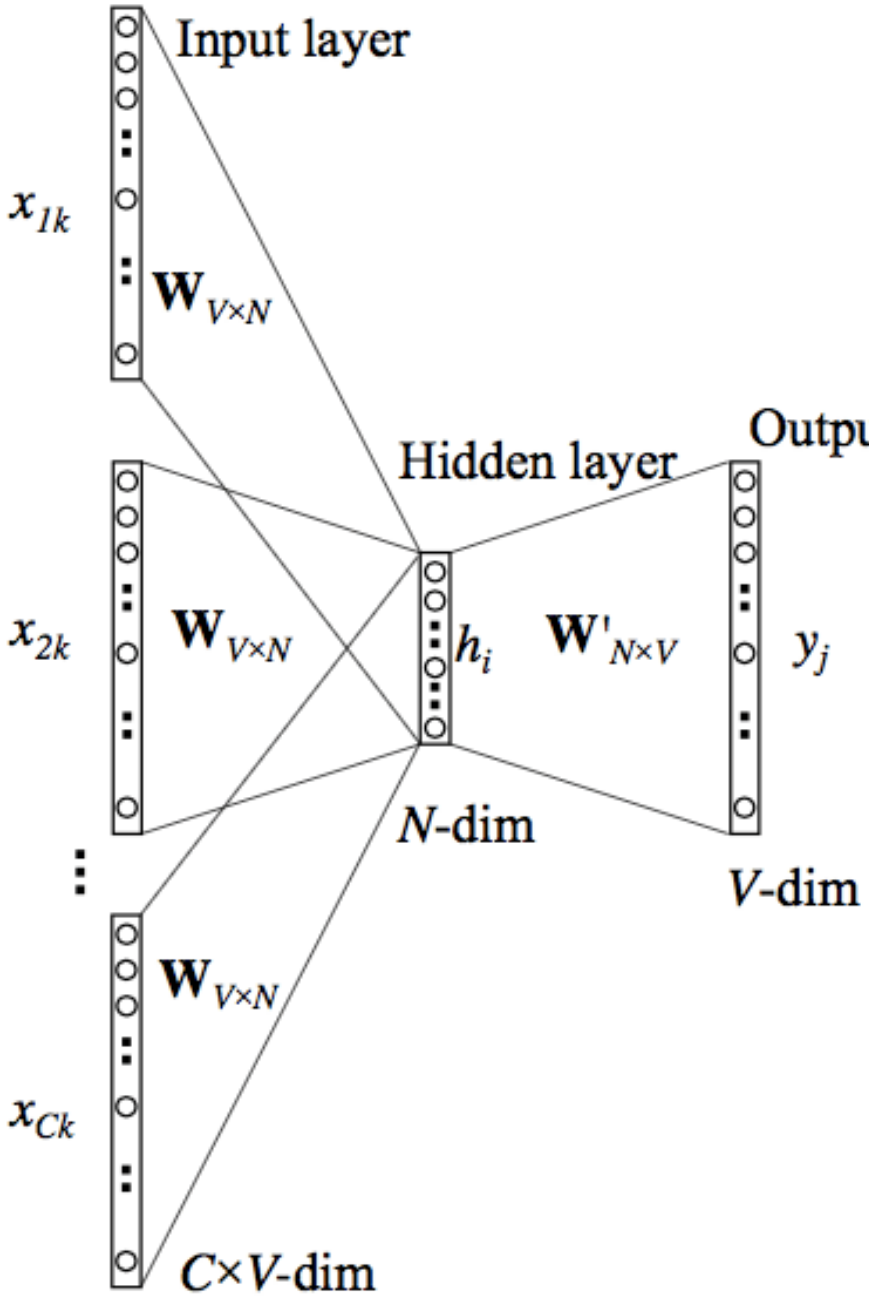
Contextualized (or Dynamic) Word Embeddings

- They assign a vector to a word that depends on its context, i.e., on preceding and following words.
- Can be used in a target model just like static embeddings.
- However, they can also be finetuned: we re-train some of the weights of the embedding model for the target task.
- Contextualized embeddings take a lot of memory and compute to train from scratch, so finetuning is their dominant use.
- Elmo, Bert, Ulmfit, GPT are the most prominent examples.
- Pre-trained versions can be downloaded for multiple languages.

In the next lecture: We will introduce *Bert* as an example of contextualized word embeddings.

Static Word Embeddings

Word2Vec: Continuous bag-of-words model (CBOW)

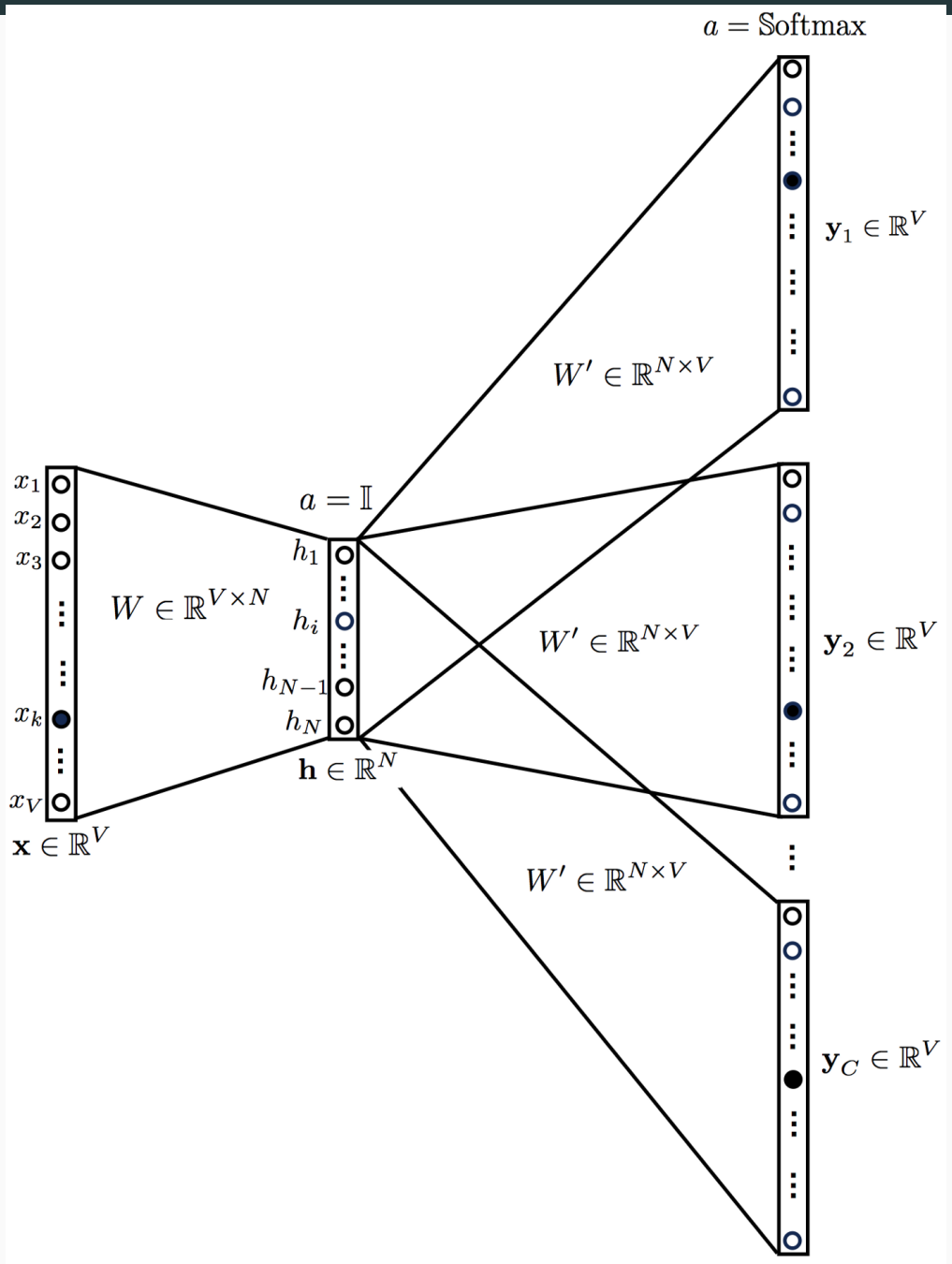


mask: we don't need the label

Word2Vec: Continuous bag-of-words model (CBOW)

- CBOW (Mikolov et al. 2013) uses the words within small window to predict the current word.
- Has only a single, linear hidden layer.
- The weights for different positions are shared.
- Window size is five in the original paper (two context words each to the left and right of the target words).
- Computationally very efficient.

Word2Vec: Skipgram



Word2Vec: Skipgram

- Skipgram (Mikolov et al. 2013) turns CBOW on its head: uses the current word to predict the context words.
- This finds word representations that are useful for predicting surrounding words in a sentence or document.
- Same overall architecture as CBOW.

Let's look at Skipgram in more detail. We follow Levy and Goldberg (2014).

Skipgram Details

- Each word $w \in W$ is associated with vector $v_w \in \mathbb{R}^d$.
- Each context $c \in C$ is associated with vector $v_c \in \mathbb{R}^d$.
- W words vocabulary, C context vocabulary; d embedding dimensionality.
- Vector components are *latent* (parameters to be learned).
- Objective function maximizes the probability of corpus D (set of all word and context pairs extracted from text):

$$\underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} p(c|w;\theta)$$

Skipgram Details

The basic skipgram formulation defines $p(c|w;\theta)$ using the softmax function:

$$p(c|w;\theta) = \frac{\exp(v_c \cdot v_w)}{\sum_{c' \in C} \exp(v_{c'} \cdot v_w)}$$

where v_c and v_w are vector representations for c and w and C is the set of all available contexts.

- We seek parameter values (i.e., vector representations for both words and contexts) such that the dot product $v_w \cdot v_c$ associated with "good" word-context pairs is maximized.
- The formulation is impractical due to the summation term $\sum_{c' \in C} \exp(v_{c'} \cdot v_w)$ over all contexts c'.

Training

- Stochastic gradient descent and backpropagation.
- It is useful to sub-sample the frequent words (e.g., *the*, *is*, *a*).
- Words are thrown out proportional to their frequency (makes things faster, reduces importance of frequent words).
- Non-linearity does not improve performance of these models, thus the hidden layer does not use an activation function.
- **Problem:** large output layer: size equal to vocabulary size, can easily be in the millions (too many outputs to evaluate).
- **Solution:** negative sampling (also hierarchical softmax).

Negative Sampling

- Instead of computing the full output layer based on the hidden layer, evaluate only only the output neuron that represents the positive class and a few randomly sampled neurons.
- The output neurons are treated as independent logistic regression classifiers.
- This makes the training speed independent of the vocabulary size (can be easily parallelized).

Negative Sampling Objective

Let D denote the dataset of observed (w, c) pairs. Let $p(D = 1|w, c)$ denote the probability that (w, c) came from D, $p(D = 0|w, c) = 1 - p(D = 1|w, c)$ that it didn't:

$$\begin{aligned}
&\operatorname*{argmax}_{\theta} \prod_{(w,c)\in D} p(D = 1|w, c; \theta) \prod_{(w,c)\in D'} p(D = 0|w, c; \theta) \\
&= \operatorname*{argmax}_{\theta} \sum_{(w,c)\in D} \log p(D = 1|w, c; \theta) + \sum_{(w,c)\in D'} \log(1 - p(D = 1|w, c; \theta)) \\
&= \operatorname*{argmax}_{\theta} \sum_{(w,c)\in D} \log \frac{1}{1 + \exp(-v_c \cdot v_w)} + \sum_{(w,c)\in D'} \log \frac{1}{1 + \exp(v_c \cdot v_w)}
\end{aligned}$$

See Levy and Goldberg (2014) for full derivation.

Model Evaluation: Question Answering

- 20K questions, 6B words, 1M words vocabulary
- Find word closest to question word (e.g., *Greece*), consider it correct if it matches the answer.

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Model Evaluation

Models trained on the same data (640-dimensional word vectors).

Model Architecture	Semantic-Syntactic Word Relationship test set	
	Semantic Accuracy [%]	Syntactic Accuracy [%]
RNNLM	9	36
NNLM	23	53
CBOW	24	64
Skip-gram	55	59

Model Evaluation

Subtracting two word vectors, and add the result to another word.

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Check more examples out at:

http://rare-technologies.com/word2vec-tutorial/

Summary

- Pre-training and finetuning is a form of transfer learning.
- Word embedding are feature vectors extracted from a pre-trained language model.
- Static embeddings assign the same word the same vector, independent of context.
- Contextualized embeddings take context into account. They can be used for finetuning.
- Word2Vec is an example of a static word embedding model.
- Negative sampling is important to make it efficient.

References

Levy, Omer and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*. Curran Associates, Red Hook, NY, pages 2177–2185.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations: Workshop Papers*. Scottsdale, AZ.

Simonyan, Karen and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations*. San Diego, CA.

Smith, Noah A. 2019. Contextual word representations: A contextual introduction. arXiv:1902.06006.