# Transfer Learning

# Problem

Deep Networks has a lot of parameters

Real world problems often don’t have much data

Data gathering is expensive

Training is expensive

# Opportunities

Many tasks are similar

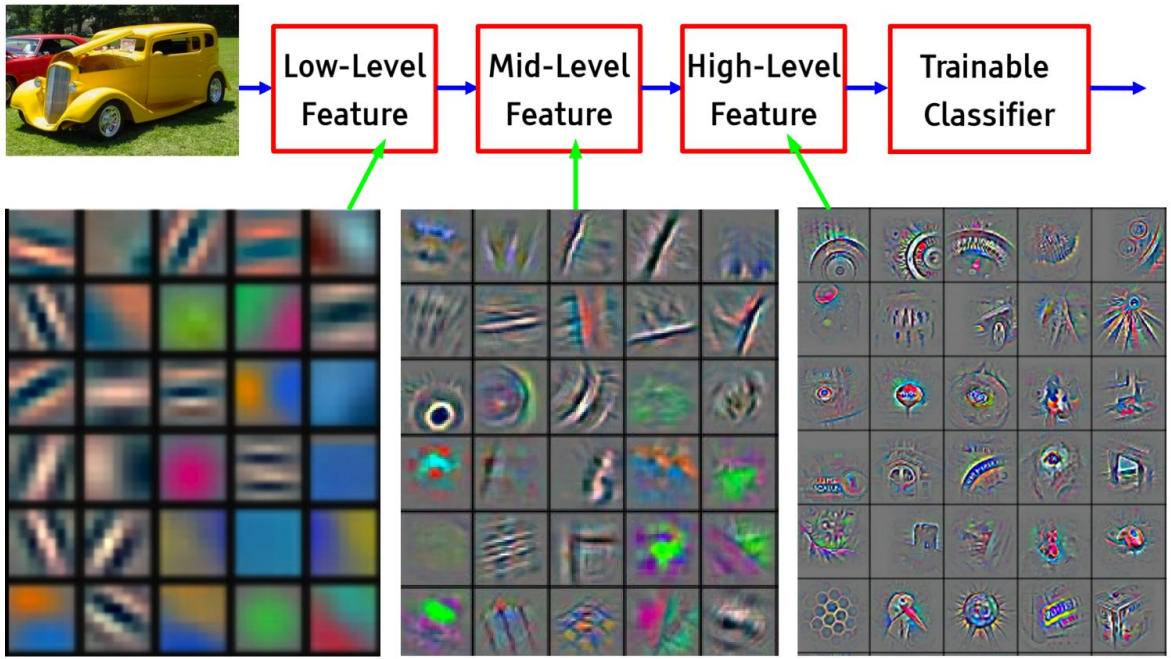
Open source

Common Representation

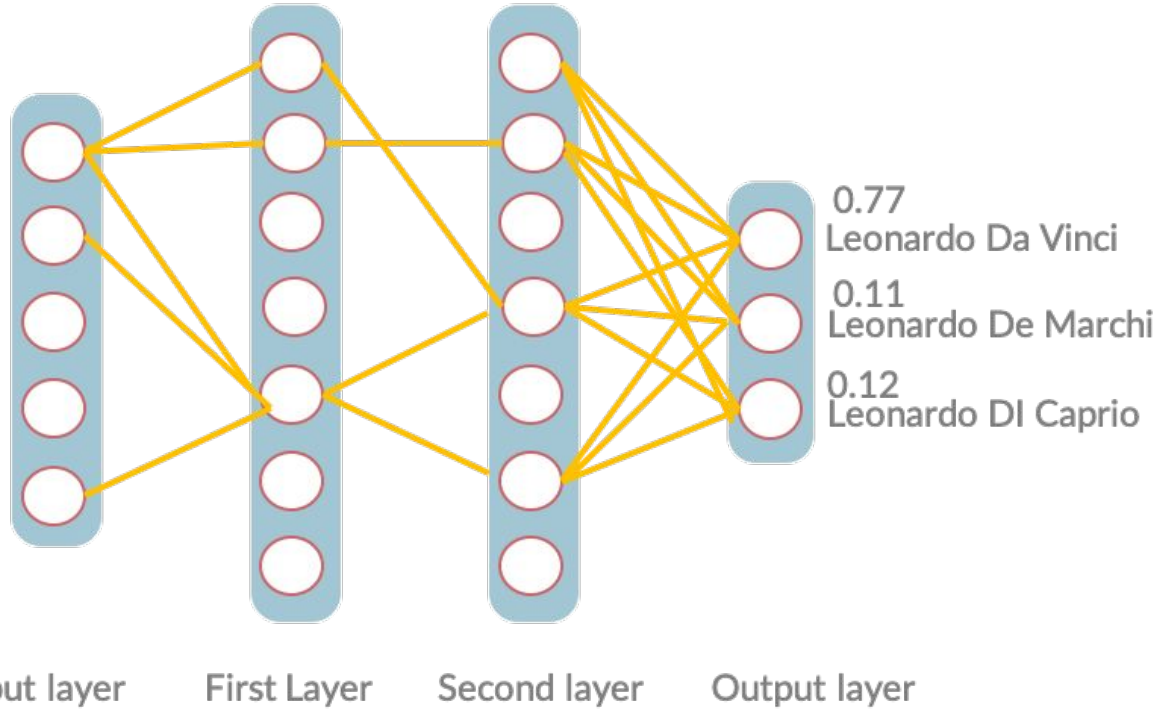
# Deep Learning = Learning Hierarchical Representations

Y LeCun

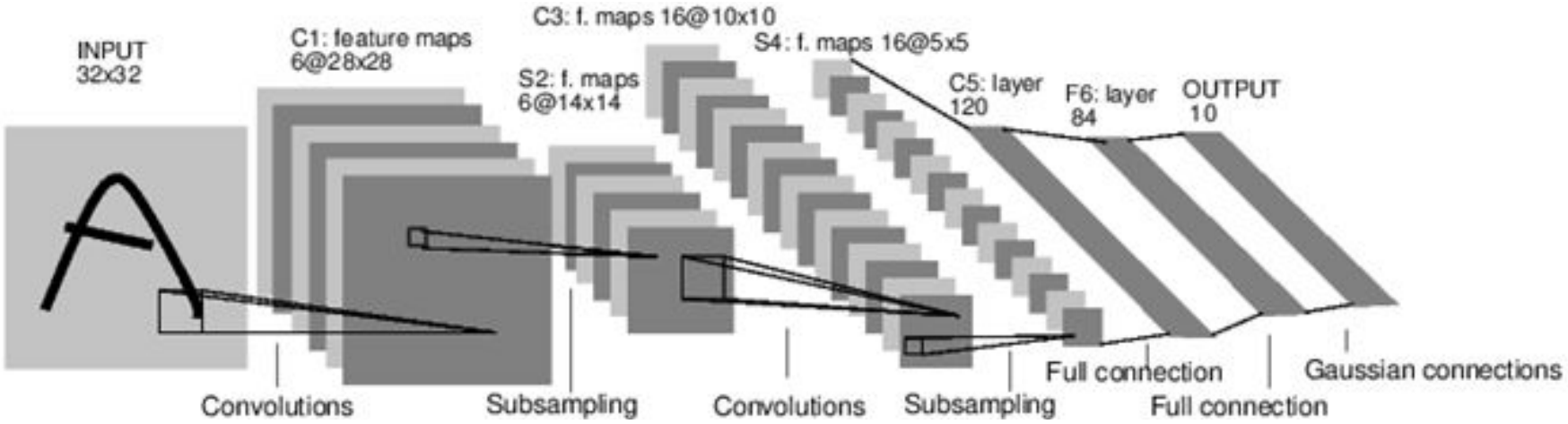
- It's deep if it has more than one stage of non-linear feature transformation




Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]


0.77
Leonardo Da Vinci
0.11
Leonardo De Marchi
0.12
Leonardo DI Caprio
Input layer
First Layer
Second layer
Output layer

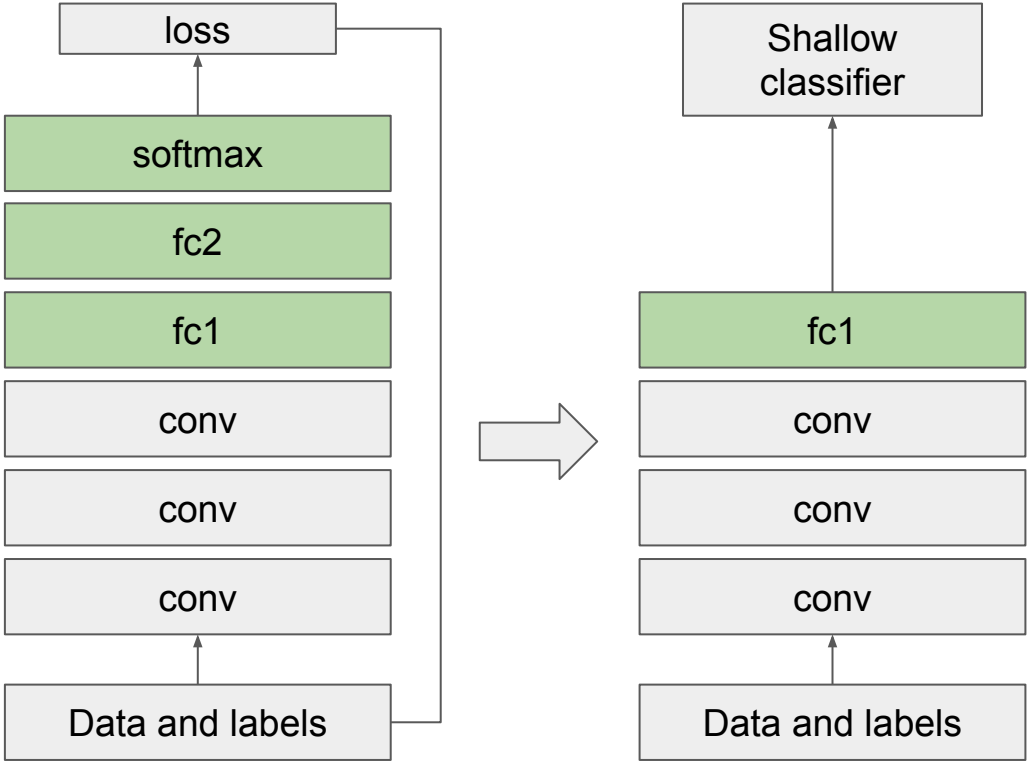
# Neural Network



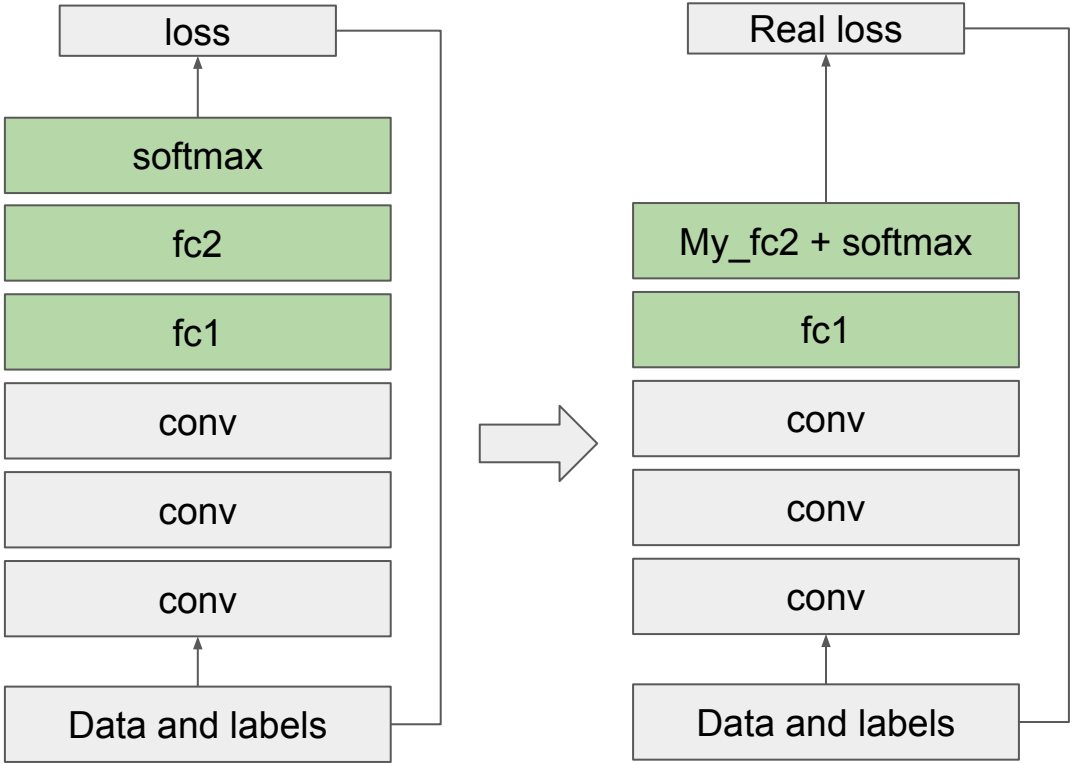

A Full Convolutional Neural Network (LeNet)

# Transfer Learning

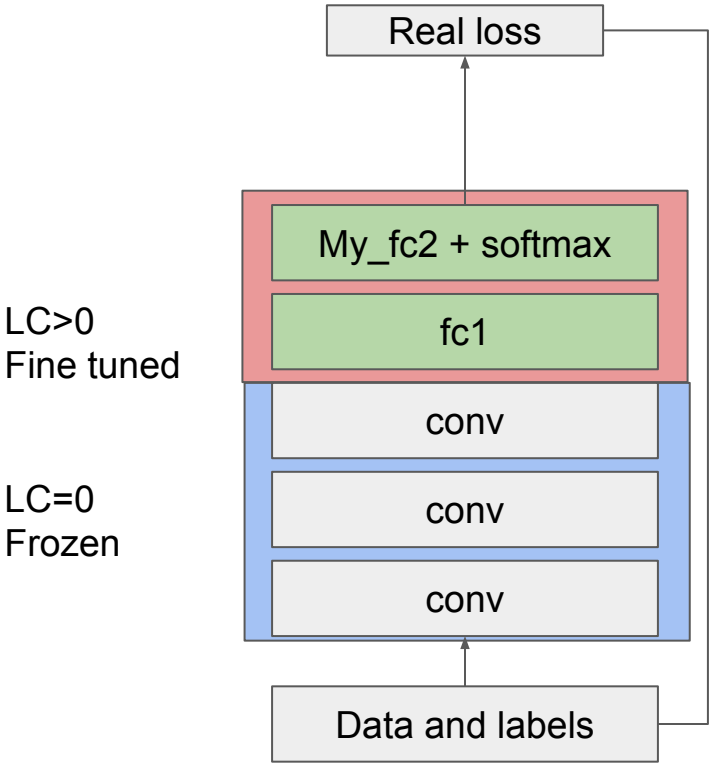
# Off the shelf



# Domain adaptation



# Domain adaptation



# Transferable features

Low layers: more general, no need to fine tune them

Higher layers: more task specific

Fine tuning improves generalization (with enough examples)

Often better performance than training from scratch

Even features from distant tasks are usually better than random weights

Also possible unsupervised domain adaptation

# Transferable features

Low layers: more general, no need to fine tune them

Higher layers: more task specific

Fine tuning improves generalization (with enough examples)

Often better performance than training from scratch

Even features from distant tasks are usually better than random weights

Also possible unsupervised domain adaptation

# Transferable features

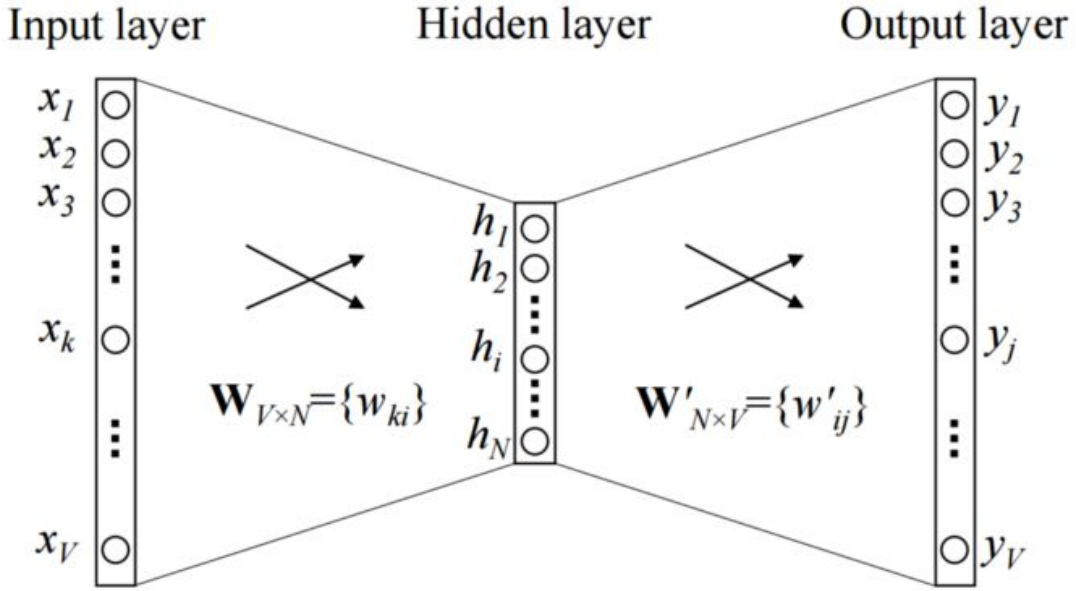
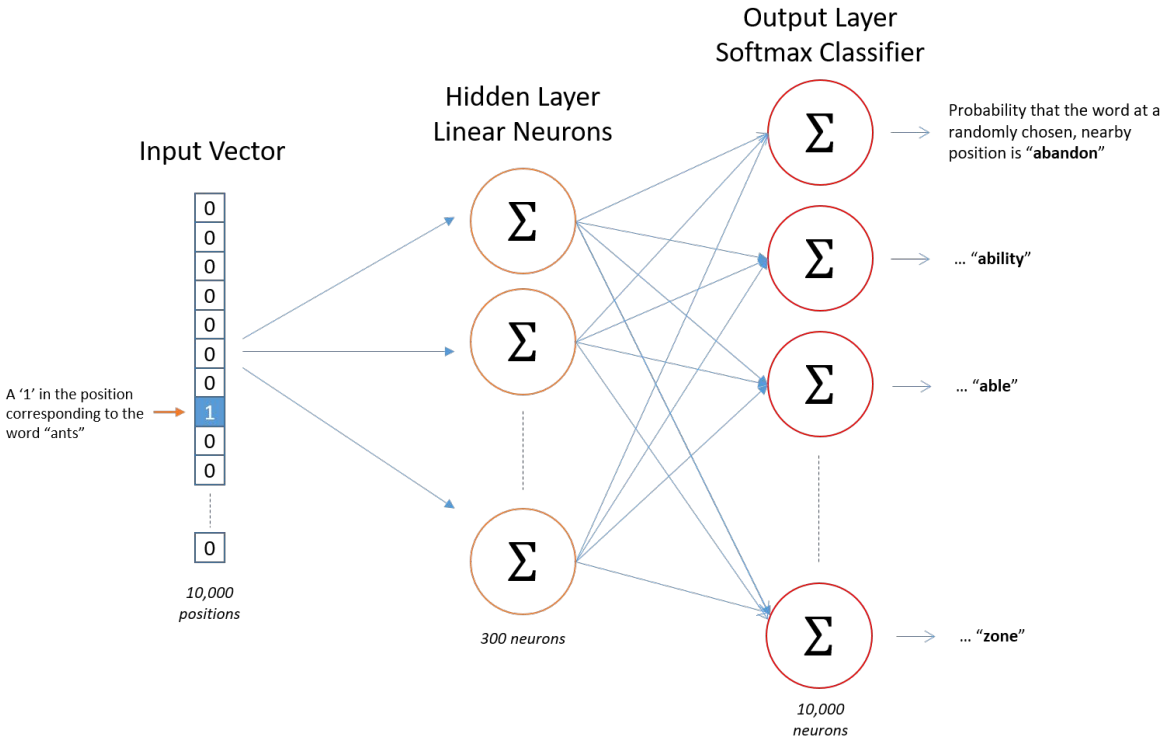



Figure 1: A simple CBOW model with only one word in the context

# Transferable features



# Transferable features

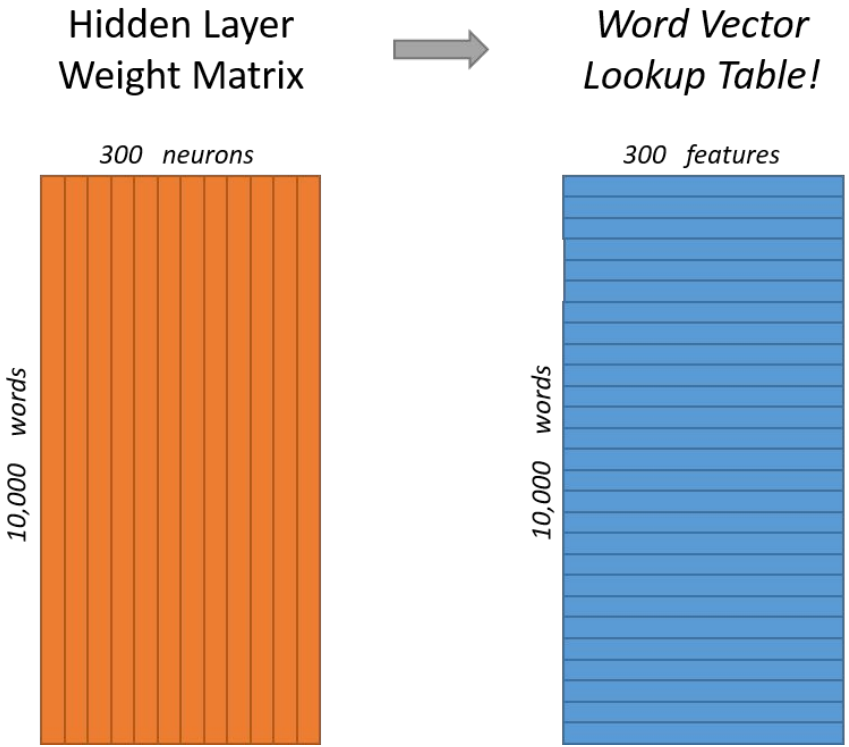
When training this network on word pairs, the input is a one-hot vector representing the input word and the training output is also a one-hot vector representing the output word. But when you evaluate the trained network on an input word, the output vector will actually be a probability distribution (i.e., a bunch of floating point values, not a one-hot vector).

http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

# Transferable features

For our example, we're going to say that we're learning word vectors with 300 features. So the hidden layer is going to be represented by a weight matrix with 10,000 rows (one for every word in our vocabulary) and 300 columns (one for every hidden neuron).

# Transferable features



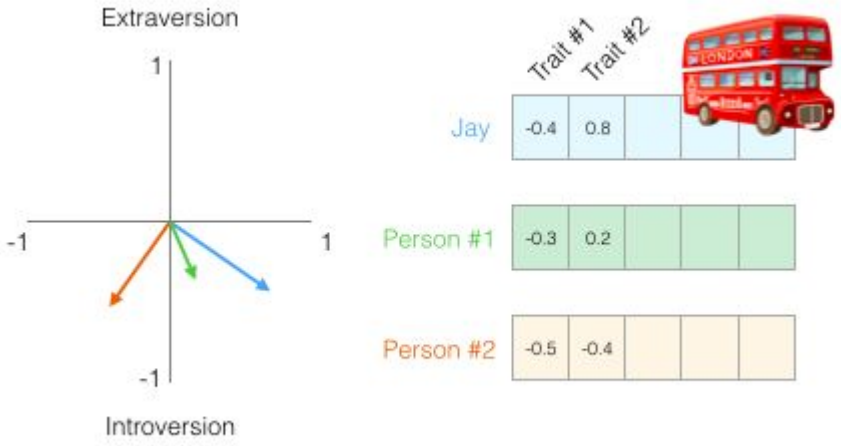
# Transferable features

If two different words have very similar “contexts” (that is, what words are likely to appear around them), then our model needs to output very similar results for these two words. And one way for the network to output similar context predictions for these two words is if the word vectors are similar. So, if two words have similar contexts, then our network is motivated to learn similar word vectors for these two words! Ta da!

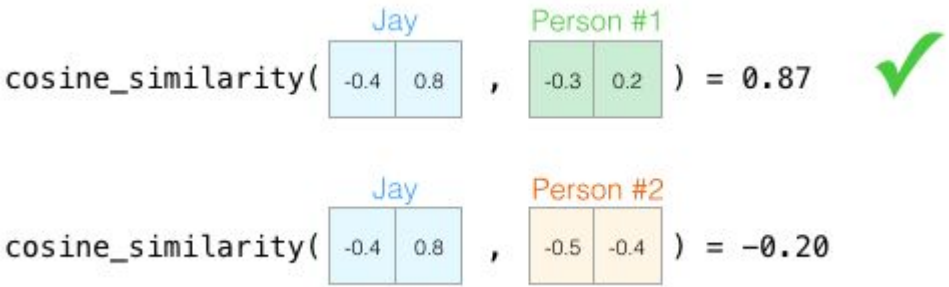
Can handle stemming

# Transferable features

# Transferable features



# Transferable features



# Transferable features

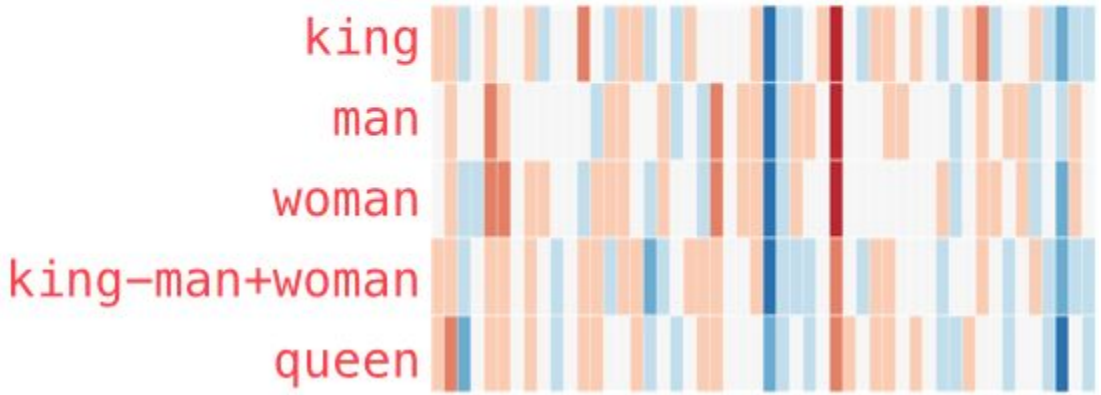
“king”

“Man”

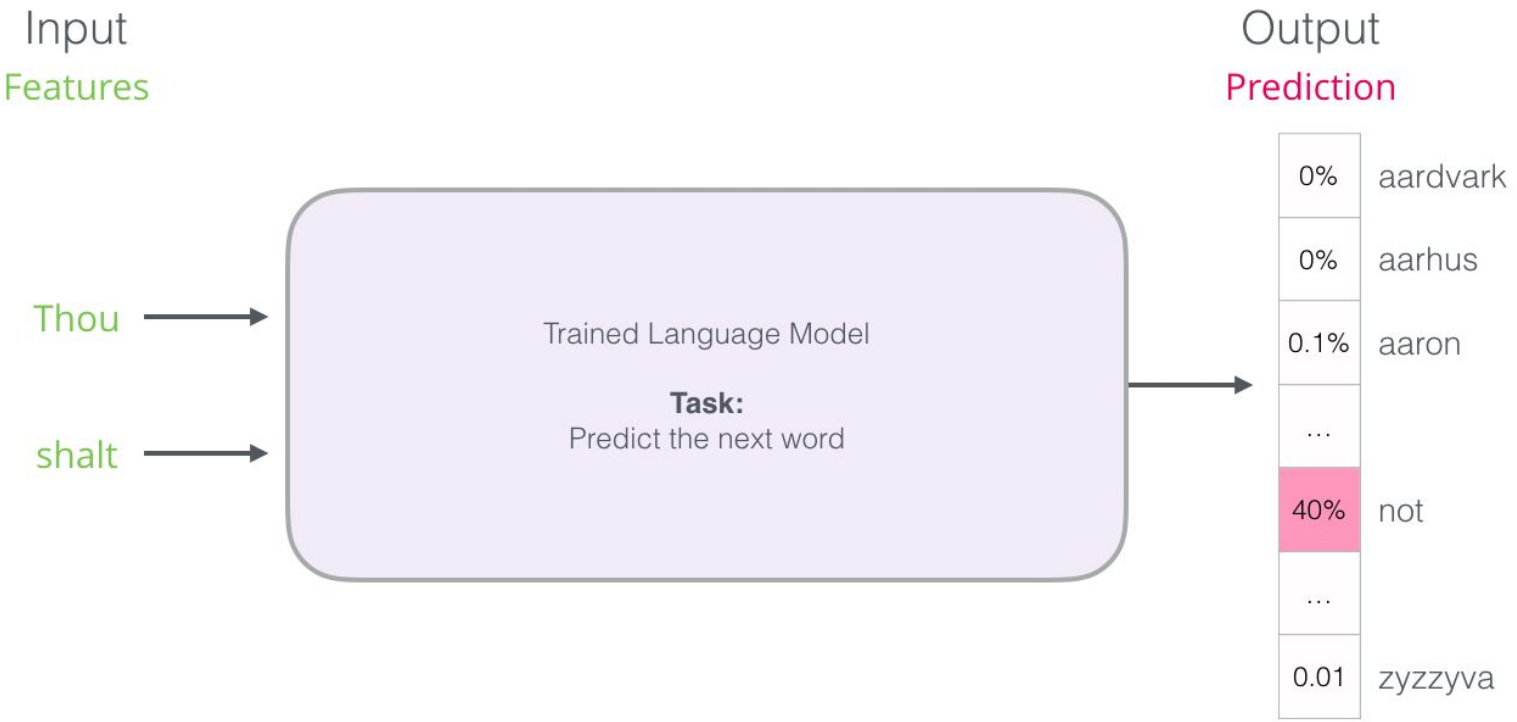
“Woman”

# Transferable features

king - man + woman ~= queen



# Transferable features



# Transferable features