W2v: Predict surrounding words in a window of length $-/+m$, for every word. Objective: maximize the log probability of any context word given the current center. $J(\theta)=\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log p\left(w_{t+j}|w_t\right)$. each word gets 2 vectors: "center", another for when it serves as "outside" context. The algorithm learns the weights for these vectors that maximize p(o—c) according to J. $p(o|c)=\frac{\exp\left(u_o^T v_c\right)}{\sum_{w=1}^{W}\exp\left(u_w^T v_c\right)}$. The problem (Neg. Sampling) with Softmax is that the gradient is dependent on the summation across all classes. This means summing across ALL word in the vocabulary. $J_t(\theta)=\log\sigma\left(u_o^T v_c\right)+\sum_{j\sim P(w)}\left[\log\sigma\left(-u_j^T v_c\right)\right]$. Less frequent words are sampled more often. Words in the actual context are excluded.

CountFit: Inject lexical similarity into distributional similarity (Word2Vec) by attracting synonymous vectors and repelling antonymous ones while preserving the original distributional similarity.
$\sum_{(u,w)} Relu\left(1-d\left(\mathbf{v}_u',\mathbf{v}_w'\right)\right)$
$VSP\left(V,V'\right)=\sum_{i=1}^{N}\sum_{j\in N(i)}\tau\left(d\left(\mathbf{v}_i',\mathbf{v}_j'\right)-d\left(\mathbf{v}_i,\mathbf{v}_j\right)\right)$ restaurant reservation domain: "Chinese" far apart from "Indian" "cheap" far apart from "expensive".
Doesn't help in task with asymmetric lexical relations lexEntail.

LEAR: Helps in tasks with taxonomic relation LexEntail. In addition to above, it reflects the hierarchical structure of a taxonomic ontology by assigning larger norms to higher-level concept. The negative examples are used to: a) force ATTRACT pairs to be closer to each other than to their respective negative examples; and b) to force REPEL pairs to be further away from each other than from their negative examples. Issues: need extensive training data: WordNet, PPDB, etc. no practical use other than artificial tasks.

Word Embeddings can be used with other tasks such as causality detection and relationship extraction, but it requires handcrafted rules to define the center/outside vectors.

MEMM: Training a LR with word features to predict output (POS tag) of the first word, then the output tag is fed to another LR to predict the next tag. Simple to train; word w/ any features. Label Bias Problem: decisions are too local and doesn't account for the whole state sequence!

BiLSTM-CRF: LSTMs model sequences but don't capture too well dependencies between labels For example: I-PER cannot follow B-LOC. i.e. CRF replaces Softmax. For a sequence of predictions, y = (y1, y2, . . . , yn), their score is defined as: $s(\mathbf{X},\mathbf{y})=\sum_{i=0}^{n}A_{y_i,y_{i+1}}+\sum_{i=1}^{n}P_{i,y_i}$; P: matrix of scores produced by the bi-LSTM; A: matrix of transition scores "transition factors" in the traditional CRF! Combine word and char. embed. to account for NE sparsity.

Nested LSTM: Similar to above. used for hierarchical NER by stacking number of LSTM layers with sharing param. to compose the larger entities. Number of layers of architecture depends on the number of nesting.

ELMO: In contrast to word embeddings, ELMo embeddings are Contextual: The representation for each word depends on the entire context in which it is used. Deep: The word representations combine all layers of a deep pre-trained neural network. Character based: ELMo representations are purely character based, allowing the network to use morphological clues to form robust representations for out-of-vocabulary tokens unseen in training. ELMo allows domain transferability to other domains. Learn context-independent language model using a stack of LSTMs

bootstrapping: Two loops: Inner loop: uses a simple decision list classifier (learner) e.g. "If instance x contains feature f, then predict label j,"and selects those rules whose precision on the training data is highest. Outer loop: In each iteration, it uses rules from (1) to assign labels to unlabeled data. It selects those instances above certain threshold, and constructs a labeled training set from them. Call the inner loop to construct a new classifier (that is, a new set of rules), and the cycle repeats. **Gupta**: improve seed selection by using their KNN in the distributional space (embed.). Find both negative and positive samples.

Co-training: The algorithm is able to learn if the two views are independent of each other, and f1(x1) = f2(x2), where x1 and x2 are the two views of the same data point x, and f1 and f2 are the classifiers trained on view 1 and view 2

Semantic Drift: Semantic drift occurs when a candidate terms is more similar to recently added terms than to the high precision terms in the first iterations. Average distributional similarity of the candidate term to the first n terms extracted, and to the last m terms. $drift(term,n,m)=\frac{avgsim(L_{1\ldots n},term)}{avgsim\left(L_{(N-m+1)\ldots N,term}\right)}$. Find the top words with the highest drift. Cluster them using agglomerative clustering. Stopping criteria: Maximum clustering: stop as soon as a cluster of size k is created; Outlier clustering: stop as soon as a cluster of size k that contains the term with the highest drift is created; Add this cluster as a new category to be bootstrapped (just like the positive ones)!

domain adapt: Frustratingly easy domain adaptation:Feature Augmentation: $\Phi^s(x)=\langle x,x,0\rangle,\quad \Phi^t(x)=\langle x,0,0\rangle$ *Intuition*: training the source and target together: the domain-specific features (positive in one domain, but negative in another domain) will be washed out; if do the feature augmentation, the features which are common and generic will receive similar weights, and the domain-specific features would receive their different weights. *Advantages*: very simple to use (regardless of different machine learning framework); *Disadvantage*: works similar to other baseline models, when the source domain is similar to the target domain, and source domain have more data sets.

socialMedia: Feature augmentation in the neural network by replicating the feature representations before feeding into sigmoid classification layer. Use gender information through domain adaptations. Recent social media posts have more useful information. Bag-of word use custom dictionaries in feedforward NN are better for learning when the features are sparsely represented in datasets. *Disadvantage*: dataset is too small; the approach is useful in some special scenario: 1) knowing where the domain adaptation is from (in this paper: gender information);

BiDaf: 1. Character Embedding Layer maps each word to a vector space using character-level CNNs. 2. Word Embedding Layer maps each word to a vector space using a pre-trained word em- bedding model. 3. Contextual Embedding Layer utilizes contextual cues from surrounding words to refine the embedding of the words. These first three layers are applied to both the query and context. 4. Attention Flow Layer couples the query and context vectors and produces a set of query- aware feature vectors for each word in the context. 5. Modeling Layer employs a Recurrent Neural Network to scan the context. 6. Output Layer provides an answer to the query.

Tree Kernels we begin by enumerating all tree fragments that occur in the training data. Calc. the inner product between $K(T1, T2)$ To compute K we need first to define the set of nodes in T1 and T2. We also define and Indicator function $I(n)$ to be 1 if sub-tree i is seen rooted at node n and zero otherwise. $\mathbf{h}(T_1) \cdot \mathbf{h}(T_2) = \sum_i h_i(T_1) h_i(T_2) = \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \sum_i I_i(n_1) I_i(n_2) = \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} C(n_1, n_2)$ Solving peaked distribution: 1) Restrict the depth of the sub-trees considered 2) Down-weigh the contribution of tree fragments with their size $\mathbf{h}(T_1) \cdot \mathbf{h}(T_2) = \sum_i \lambda^{size_i} h_i(T_1) h_i(T_2)$ Perceptron 1) primal: if $y_m W \dot{x}_m < 0 : w+ = y_m W \dot{x}_m$ 2) Dual: if $y_m \sum_d \alpha_d x_d x_m < 0 : \alpha_m + = 1$

Attention: It encodes the input sentence into a sequence of vectors (the hidden states of each input word) and chooses a subset of these vectors adaptively while decoding the translation. **Transformers**: a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. Easy to parallelize. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head (what we saw before), averaging inhibits this. (+): No learning; They extrapolate to sequence lengths longer than the ones encountered during training; Nearly identical results to learned position embeddings (the classic approach). (-) Complex in structure and there is still some room for simplification.

Ladder Networks: an autoencoder with skip connections from the encoder to decoder. Noising and denoising occur at every layer. The skip connections relieve the pressure to represent details in the higher layers of the model because, through the skip connections, the decoder can recover any details discarded by the encoder. Implementation: 1. Train any standard feedforward neural network. 2.For each layer,analyze the conditional distribution of representations given the layer above and add Gaussian noise. 3. Define a function to reconstruct the previous state before noising. 4. Train the whole network using SGD.

Mean Teacher: a teacher-student algorithms that learn with unlimited unlabeled data and limited supervision. To mitigate confirmation bias of autoencoders, the teacher weights in MT are averaged over the training epochs, akin to the averaged perceptron. This provides a more robust scaffolding that the student model can rely on during training when gold labels are not available. It consists of two models with identical architectures where one is termed the teacher, and the other the student. The weights in the teacher network are set through an exponential moving average of the student weights. The input to both of these models is the same datapoint, which is augmented by some noise that is specific to each model. The cost function is a combination of two different type of costs: classification and consistency. The classification cost applies to supervised data points, and can be instantiated with any supervised cost function (e.g., we used categorical cross-entropy). The consistency costs is used for unlabeled data points, and aims to minimize the differences in predictions between the teacher and the student models.

Lightly-supervised: iteratively grows a pool of multiword entities and n-gram patterns for each category of interest c, and learns custom embeddings for both. The entity pools are initialized with a few seed examples for each category. Then the algorithm iteratively applies the following three steps for T epochs: (1) Learning custom embeddings for all entities and patterns in the dataset, using the current as supervision. (2) Pattern promotion: generate the patterns that match the entities in each pool, rank them and select the top ranked patterns for promotion to the corresponding pattern pool. (3) Entity promotion: Entities are promoted to using a multi-class classifier that estimates the likelihood of an entity belonging to each class.

Tasks: FEVER: Make the model as unlexicalized as possible; Semi-supervised learning: add more world knowledge. But how to choose what is relevant? Deterministic model: do not use ML. Instead align syntactic trees using an edit distance measure, and decide which transformations are meaning altering. PICO: PICO ideas: Models that capture the hierarchical structures of labels; Semi-supervised learning: Enhance dataset somehow; Use rules to seed the classifier; Use one of the modern semi-supervised methods we will cover.