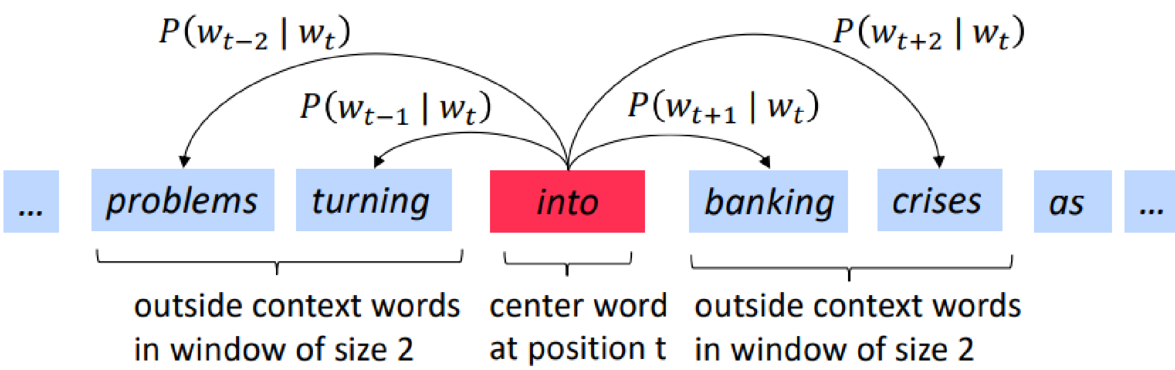
FastText

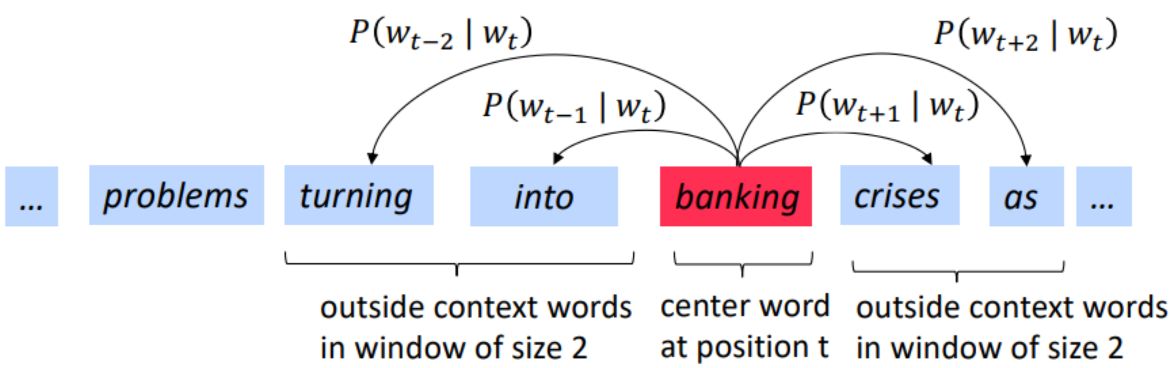
Word2Vec Idea:

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors Idea:

- We have a large corpus of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability

Example windows and process for computing: $P\left(W_{t+j} \middle| W_t\right)$





Objective Function:

For each position t = 1, …,T, predict context words within a window of fixed size m, given center word W_j.

Likelihood = $L(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m; j \ne 0} P\left(W_{t+j} \middle| W_t;\ \theta\right)$

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T}\log L(\theta)$$

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m\\ j\neq 0}} logP\left(W_{t+j}\middle|W_t;\ \theta\right)$$

Calculation of $P\left(W_{t+j}\middle|W_t;\ \theta\right)$:

We will use two vectors per word w:

- v_w when w is a center word
- u_w when w is a context word

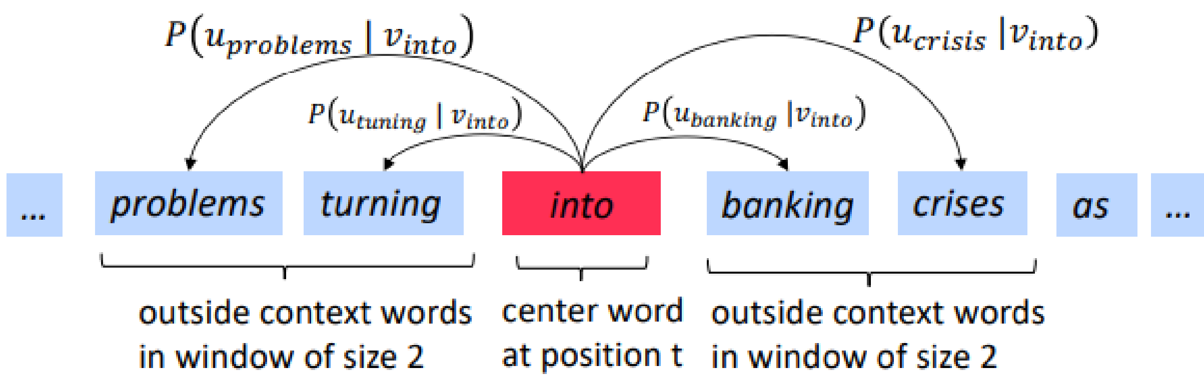
Then for a center word c and a context word o:

We know that, $Softmax(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n}\exp(x_i)}$

$$P(o|c) = \frac{\exp(u_0^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

Example:

$P(u_{problems}|v_{into})$ **short for** $P(problems|into; u_{problems}, v_{into}, \theta)$



Now, θ represents all model parameters, in one long vector. In our case with d-dimensional vectors and V-many words:

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

So every word has two vectors. We optimize these parameters by walking down the gradient.

For gradient descent we will have to do calculation of partial derivative with respect to u_0, v_c. We are going to calculate the partial derivative of cost function J(θ) with respect to v_c:

$$\frac{\delta}{\delta v_c} P(o|c) = \frac{\delta}{\delta v_c} log \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

$$\Rightarrow \frac{\delta}{\delta v_c} log \exp(u_o^T v_c) - \frac{\delta}{\delta v_c} log \sum_{w \in V} \exp(u_w^T v_c)$$

$$\Rightarrow \frac{\delta}{\delta v_c} (u_o^T v_c) - \frac{\delta}{\delta v_c} log \sum_{w \in V} \exp(u_w^T v_c)$$

Now, for the calculation of $\frac{\delta}{\delta v_c} (u_0^T v_c)$ we might consider to partially derive with respect to single component of the vector, v_c.

$$\frac{\delta}{\delta v_{c_1}} (u_{o_1} v_{c_1} + u_{o_2} v_{c_2} + \ldots + u_{o_{100}} v_{c_{100}})$$

$$\Rightarrow u_{o_1}$$

So, $\frac{\delta}{\delta v_c} (u_o^T v_c) = u_o$

Now,

$$\frac{\delta}{\delta v_c} log \sum_{w \in V} \exp(u_w^T v_c) = \frac{1}{\sum_{w \in V} \exp(u_w^T v_c)} \frac{\delta}{\delta v_c} \sum_{w \in V} \exp(u_w^T v_c)$$

$$= \frac{1}{\sum_{w \in V} \exp(u_w^T v_c)} \sum_{w \in V} \frac{\delta}{\delta v_c} \exp(u_w^T v_c)$$

$$= \frac{1}{\sum_{w \in V} \exp(u_w^T v_c)} \sum_{w \in V} \exp(u_w^T v_c) \frac{\delta}{\delta v_c} (u_w^T v_c)$$

$$= \frac{1}{\sum_{w \in V} \exp(u_w^T v_c)} \sum_{w \in V} \exp(u_w^T v_c)\ u_w$$

Hence,

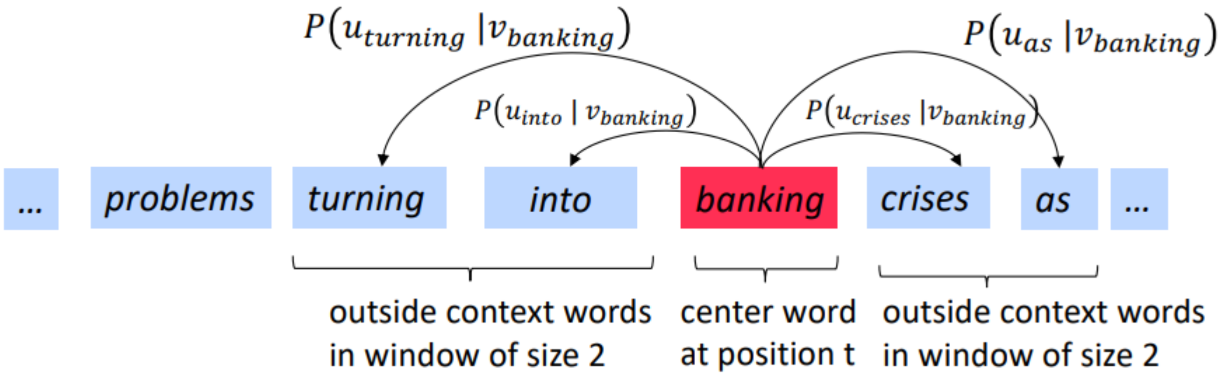
$$\frac{\delta}{\delta v_c} P(o|c) = u_o - \frac{\sum_{w \in V} \exp(u_w^T v_c)\ u_w}{\sum_{w \in V} \exp(u_w^T v_c)}$$

$$\frac{\delta}{\delta v_c} P(o|c) = u_o - \sum_{w \in V} \frac{\exp(u_w^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} u_w$$

$$\frac{\delta}{\delta v_c} P(o|c) = u_o - \sum_{w \in V} P(o|c) u_w$$

We went through gradient for each center vector v in a window, similarly we can find gradient for each outside vectors u.

Generally in each window we will compute updates for all parameters that are being used in that window.



$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

Finally two vector of the same word is averaged to get final word to vector values- $\frac{v_{aardvark} + u_{aardvark}}{2}$

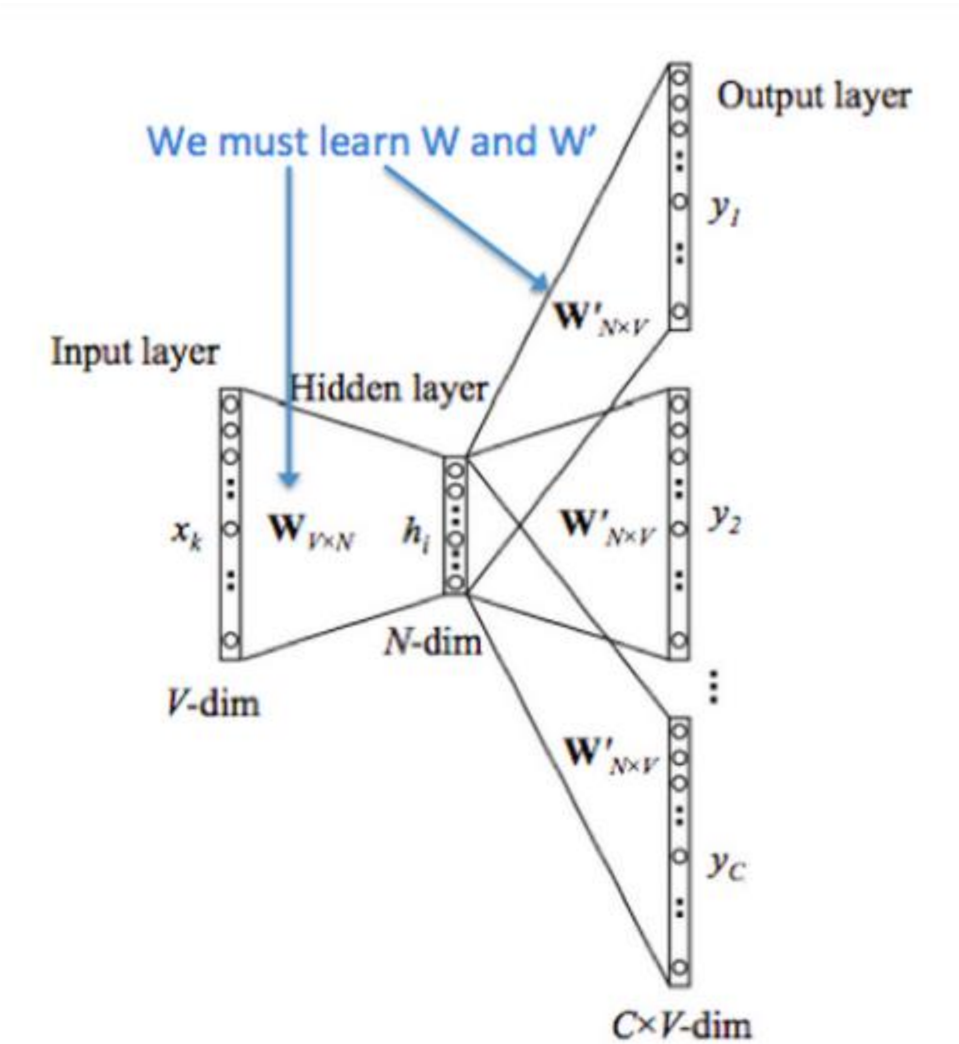
There are two model variant:

1. **Skip-grams (SG)** Predict context (”outside”) words (position independent) given center word. Example: Dataset would be
   (quick, the),
   (quick, brown),
   (brown, quick),
   (brown, fox),
2. **Continuous Bag of Words (CBOW):** Predict center word from (bag of) context words. Example: Dataset would be
   ([the, brown], quick),
   ([quick, fox], brown),
   ([brown, jumped], fox),

There are two training methods:

1. **Negative sampling** defines an objective by sampling negative examples.
2. **Hierarchical softmax** defines an objective using an efficient tree structure to compute probabilities for all the vocabulary.

Skip Gram Network:



Algorithm Steps:

1. We generate our one hot input vector $x \in \mathbb{R}^{|v|}$ of the center word.
2. We get our embedded word vector for the center word $v_c = Vx \in \mathbb{R}^n$
3. Generate a score vector $z = Uv_c$.
4. Turn the score vector into probabilities, $\hat{y} = softmax(z)$. Note that $\hat{y}_{c-m}, \ldots, \hat{y}_{c-1}, \hat{y}_{c+1}, \ldots, \hat{y}_{c+m}$ are the probabilities of observing each context word.
5. We desire our probability vector generated to match the true probabilities which is $y^{(c-m)}, \ldots, y^{(c-1)}, y^{(c+1)}, \ldots, y^{(c+m)}$ the one hot vectors of the actual output.