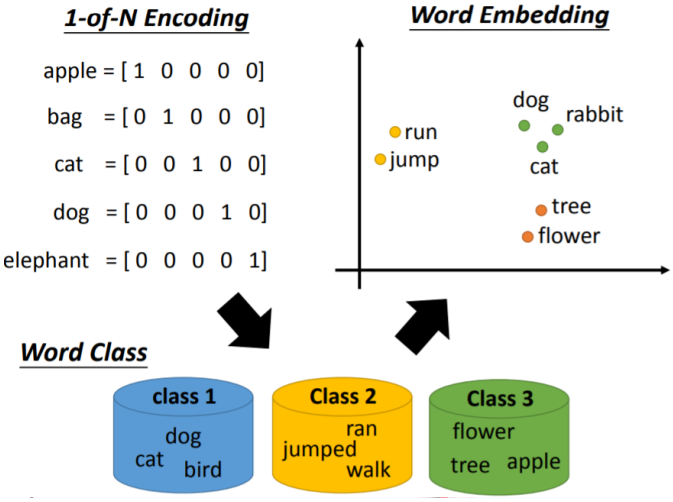
Word Embedding

Word Embedding

如果使用 1-of-N encoding，每个单词用一个 vector 表示，那么 5 个单词就需要 5 个 vector；如果我们把一个类别的单词都放到那个类里，既属于 class1 的单词有 dog、cat、bird，属于动物类的单词，同理可以得出 class2，class3；但只做 classify 是不够的，这些 class 之间也有一些其他的联系；比如 class1 属于动物，class3 属于植物，他们都是生物，只做 classify 并不能体现这种联系；

现在我们把每个 word 都 project 到一个两个 dimension 上，水平的 dimension 可以是表示生物(class1，class3)和其他类别 class2 之间的差距，竖直的 dimension 可以是会动(class2,class3)的和不会动 class1 之间的差距；

如果现在有 10w 个单词，1-of-N encoding 就需要 10w 个 vector，但 word embedding 可能只需要 50 维左右，就可以表示这些所有的 word



word2vec 是一个无监督学习问题，如果 network 的 input 为“Apple”，要输出器对应的 vector

Count based

基于计数的方法，记录文本中词的出现次数。如果两个单词w_i, w_j常常一起出现，那么我们就认为其对应的 vector $V(w_i), V(w_j)$之间就是非常接近的。

用$N_{i,j}$表示w_i, w_j在同一个 document 中出现的次数，那么我们希望找到对应的$V(w_i), V(w_j)$，其做 inner product 的值和这个次数越接近越好。

- **Count based**
  - If two words w_i and w_j frequently co-occur, $V(w_i)$ and $V(w_j)$ would be close to each other
  - E.g. Glove Vector: http://nlp.stanford.edu/projects/glove/

$V(w_i) \cdot V(w_j)$ ⟷ $N_{i,j}$

Inner product

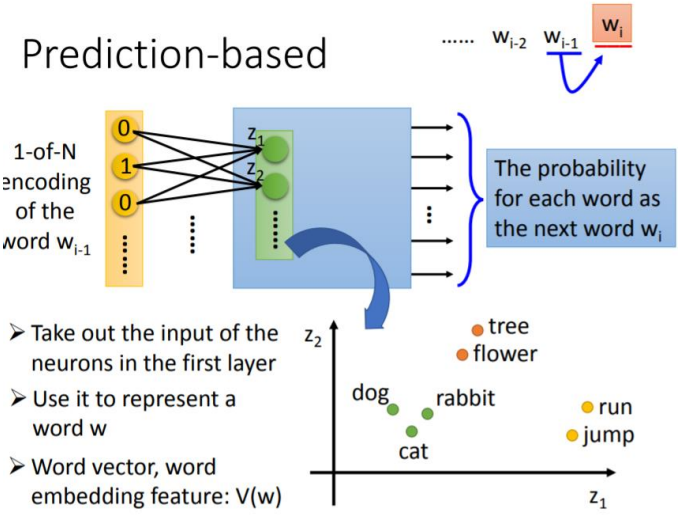
Number of times w_i and w_j in the same document

Prediction-based

基于预测的方法，既可以通过上下文预测中心词，也可以通过中心词预测上下文。中心词即我们要预测的词。在下文中，w_i是我们要预测的值，…，w_{i-2}, w_{i-1}就是其对应的上下文；

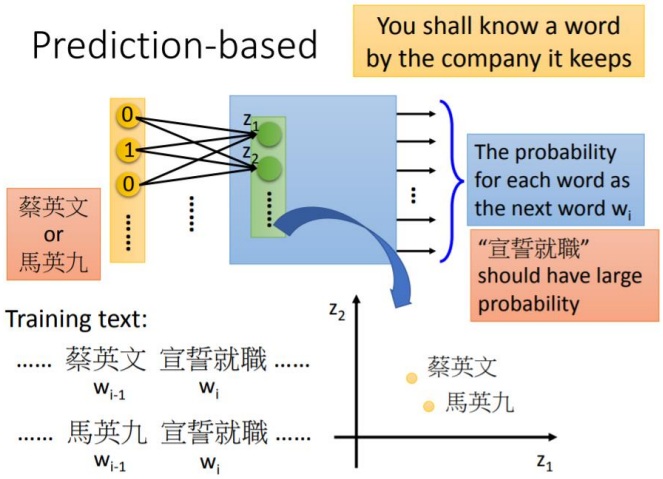
现在我们把w_{i-1}的 one-hot encoding 作为网络的输入，网络的输出为每个词作为下一个词w_i输出的概率，如果词袋中有 10w 个词，那么输出的维度就对应为 10w 维

把网络中第一个 hidden layer 的 input 拿出来，即$z = (z_1, z_2, \dots)^T$;如果输入为不同 1-of-N encoding，那么对应的 z 也是不一样的。我们就可以 z 来代表一个 word，即 z 就是我们要寻找的 word vector $V(w)$。



如果现在有两个 training text，“蔡英文宣誓就职”、“马英九宣誓就职”，如果输入的 1-of-N encoding 是“蔡英文”或“马英九”，那么我们希望在网络的输出概率中，“宣誓就职”的概率是最大的，同理我们也可以把网络的第一个 hidden layer 的输入作为 z，就是我们要寻找的 word vector $V(w)$。

这时我们就需要中间的 hidden layer 来做这样一件事，如果输入为不同的词汇(“蔡英文”，“马英九”)，那么我们希望中间的 hidden layer 可以把不同的词汇 project 到相同的空间，这样网络的输出才可能都是“宣誓就职”对应的概率最大



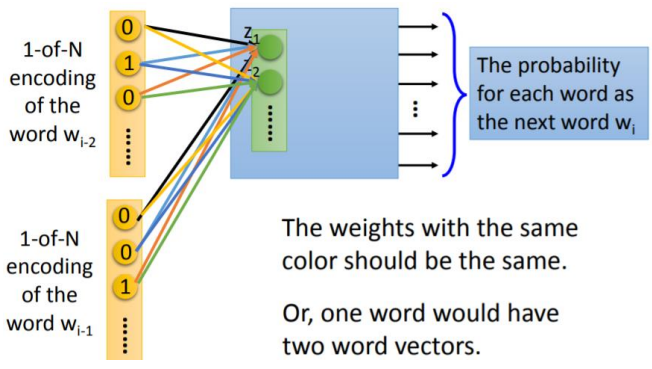
Sharing weight

只考虑前面的一个词汇，来预测下一个词汇会很难，我们可以考虑前面的几个词汇。现在我们来叙述考虑前两个词汇的结果。

现在我们并不能像之间的 neural network 那样，所有的输入都连成一个 vector 作为输入。但实际上，我们希望不同的 one-hot vector 的同一个维度之间是 tie 在一起的。即w_{i-1}, w_{i-2}的第一维对应z_1，其对应的 weight 是一样的；同理w_{i-1}, w_{i-2}的第二维对应z_1，其对应的 weight 是一样的，……

Q：为什么不同的 one-hot vector 的同一维度之间是 tie 在一起的呢?

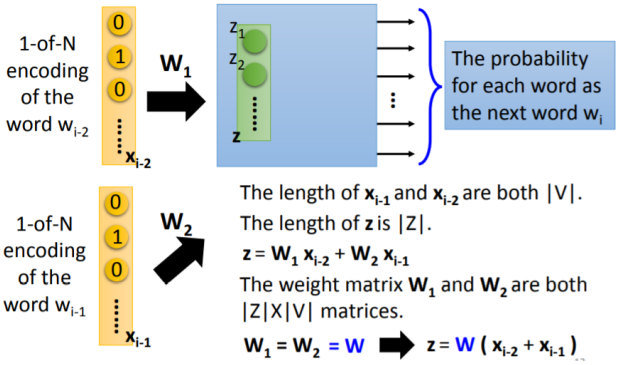
A: 我们现在把同一个 word 放到w_{i-1}, w_{i-2}的位置，如果每一维对应的 weight 都不一样，那么实际上就输入例如两个 vector。



如果我们设置x_{i-1}, x_{i-2}的长度都是$|V|, z$的长度是$|Z|$，那么

$z = W_1 x_{i-2} + W_2 x_{i-1}$

其中$W_1 = W_2 = W$,那么$z = W(x_{i-2} + x_{i-1})$，也就得到了 word vector $V(w)$



那么在实际的网络训练中，我们如何保证$W_1 = W_2$呢?

首先需要将w_1, w_2初始化为相同的值，那么

$$w_i \leftarrow w_i - \eta \frac{\partial C}{\partial w_i}$$

$$w_j \leftarrow w_j - \eta \frac{\partial C}{\partial w_j}$$

由于 gradient 的值不同，w_1, w_2在更新一次参数之后就不相等了，必须保证每次更新之后的值还是一样的，因此

$$w_i \leftarrow w_i - \eta \frac{\partial C}{\partial w_i} - \eta \frac{\partial C}{\partial w_j}$$

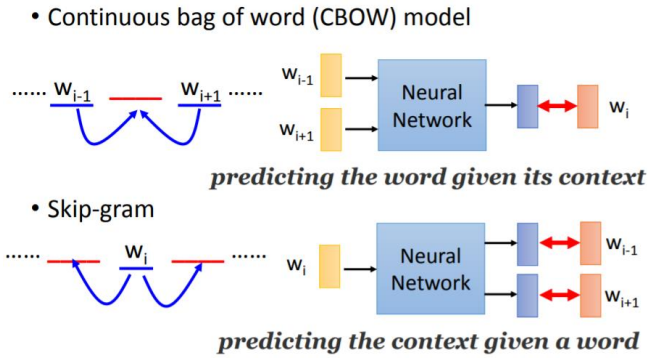
$$w_j \leftarrow w_j - \eta \frac{\partial C}{\partial w_j} - \eta \frac{\partial C}{\partial w_i}$$

这样每次更新的值都一样的，也就保证了$w_1 = w_2$

Various Architecture

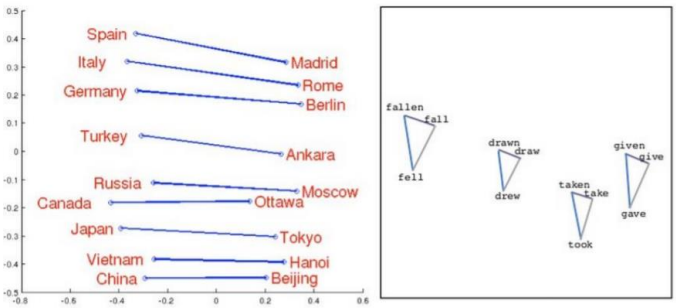
CBOW：根据上下文的词汇w_{i-1}, w_{i+1}来预测中心词w_i

Skip-gram：根据中心词w_i来预测上下文w_{i-1}, w_{i+1}。



Result

vec(Rome)-vec(Italy)≈vec(Berlin)-vec(Germany),Italy 和 Rome 之间有 is-capital-of 的关系，这种关系也恰好在 Madrid 和 Spain 之间出现



如果现在有人问机器一个问题，Rome 和 Italy 之间的关系就像是 Berlin 和什么的关系?我们就可以通过计算 vec(Berlin) ≈vec(Rome)-vec(Italy)+vec(Germany)

- Characteristics

$$V(Germany) \approx V(Berlin) - V(Rome) + V(Italy)$$

$$V(hotter) - V(hot) \approx V(bigger) - V(big)$$
$$V(Rome) - V(Italy) \approx V(Berlin) - V(Germany)$$
$$V(king) - V(queen) \approx V(uncle) - V(aunt)$$

- Solving analogies

Rome : Italy = Berlin : ?

Compute $V(Berlin) - V(Rome) + V(Italy)$

Find the word w with the closest V(w)