機械学習における自然言語処理技術

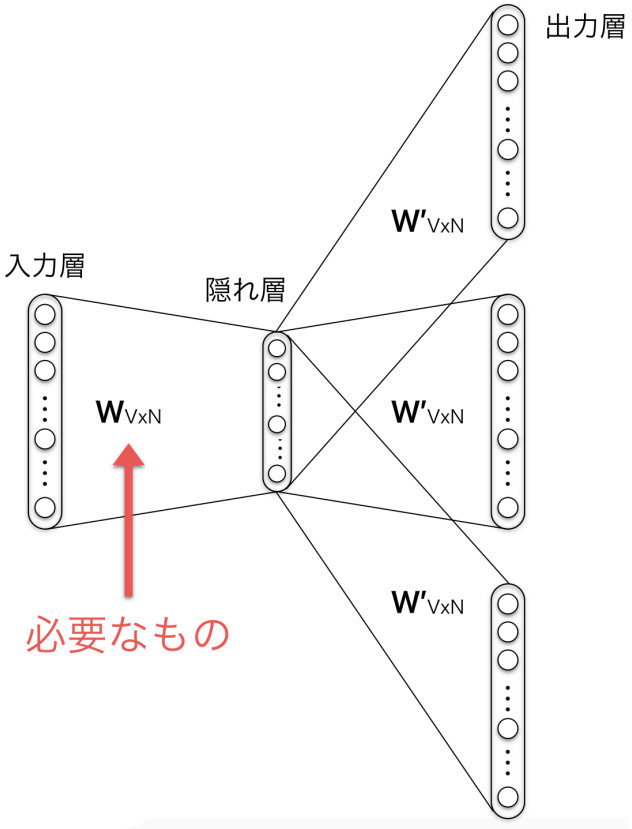
Embedding

one-hot表現

- 最もシンプルなembedding手法.
- 表現したい語彙をリストに表現して，各単語を表現する次元を準備する． **表現したい文章に含まれているか否か**のベクトルで表現する.

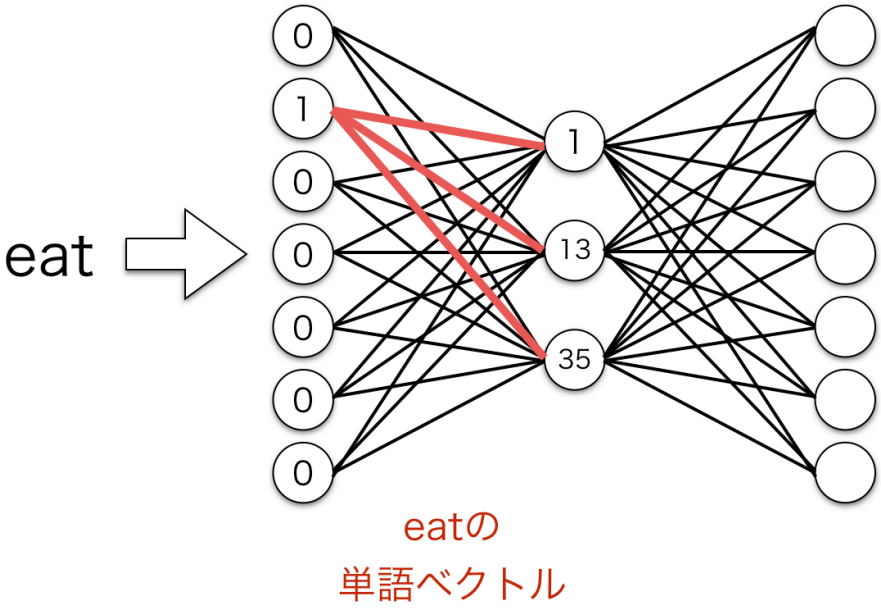
Word2Vec

- 論文: Efficient Estimation of Word Representations in Vector Space
- 大量のテキストデータを解析して，各単語の意味をベクトルで表現する方法.
- その中でもSkip-Gramモデル(ある単語の周辺に出現する単語の出現確率を計算する)が主に使われる
- Skip-Gram は２層のニューラルネットワークであり隠れ層は一つだけ．隣接する層のユニットは全結合している.



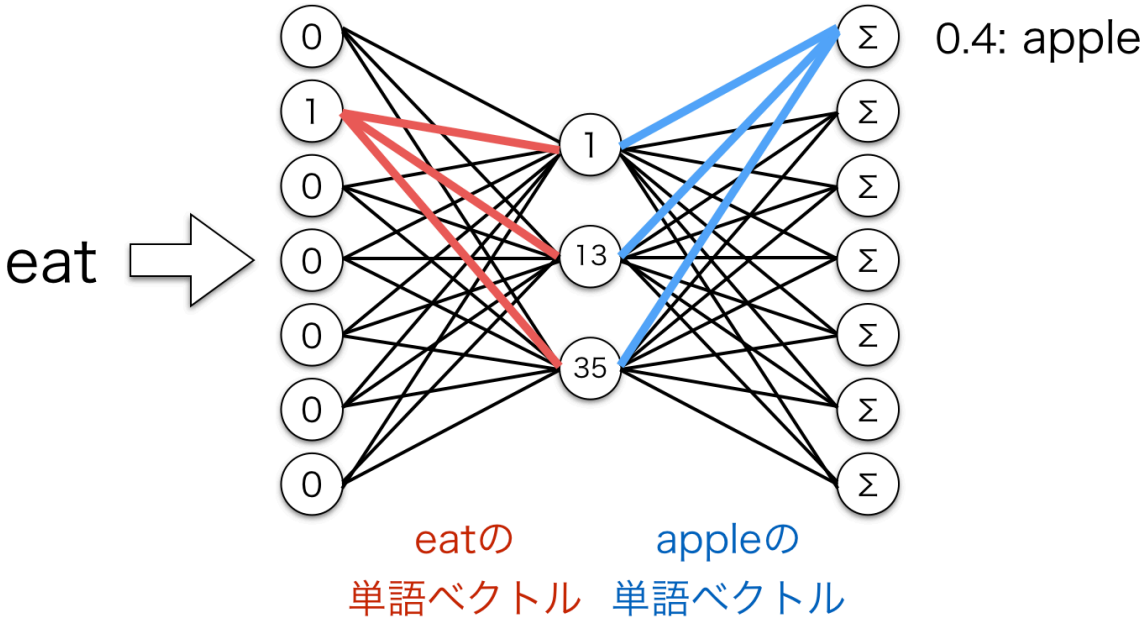
- 目的関数を設定して，2層のニューラルネットワークを構築するが， **word2vecにおいて必要なものは，モデル自体ではなく隠れ層の重み**であることに注意.
- 入力としてある単語，出力にその周辺単語を与えてニューラルネットワークを学習させることで，**「意味が近い(=意味ベクトルの距離が近い)時は周辺単語の意味ベクトルもまた距離が近いはず」**という仮説に基づいたembedding表現を得ることができる.

$$\begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0 & 0\end{bmatrix} \times \begin{bmatrix}74 & 29 & 98\\ 1 & 13 & 35\\ 65 & 31 & 37\\ 96 & 88 & 84\\ 45 & 94 & 96\\ 21 & 88 & 9\\ 6 & 78 & 94\end{bmatrix} = \begin{bmatrix}1 & 13 & 35\end{bmatrix}$$



eatの
単語ベクトル

- 上図のように，one-hotベクトルを入力として与えてあげれば，実際に対象の単語ベクトルを抽出する際は内積ではなくインデックスを使って抽出すれば良いだけなので単語数や隠れそうの次元を気にすることなくモデルを構築することができる



- 出力層は上図のよう．対象の単語の重みベクトルを得たあと，中間層から出力層の単語の重みベクトルとの内積をとっている．つまり単語同士の内積が出力となっていることがわかる．

LSTM

- 論文: Sequence to Sequence Learning with Neural Networks

- LSTM(Long short-term memory)は，RNN(Recurrent Neural Network)の拡張として1995年に登場した，時系列データ(sequential data)に対するモデル，あるいは構造(architecture)の1種．その名は，Long term memory(長期記憶)とShort term memory(短期記憶)という神経科学における用語から取られている．LSTMはRNNの中間層のユニットをLSTM blockと呼ばれるメモリと3つのゲートを持つブロックに置き換えることで実現されている．

Hochreiterの勾配消失問題

- 当時のRNNの学習方法は，BPTT(Back-Propagation Through Time)法とRTRL(Real-Time Recurrent Learning)法の2つが主流で，その2つとも完全な勾配(Complete Gradient)を用いたアルゴリズムだった
- しかし，このような勾配を逆方向(時間をさかのぼる方向)に伝播させるアルゴリズムは，多くの状況において「爆発」または「消滅」することがあり，結果として長期依存の系列の学習が全く正しく行われないといいう欠点が指摘されてきた
- Hochreiterは自身の修士論文(91年)において，時間をまたいだユニット間の重みの絶対値が指定の(ごくゆるい)条件を満たすとき，その勾配はタイムステップtに指数関数的に比例して消滅または発散することを示した．
- これはRNNだけではなく，勾配が複数段に渡って伝播する深いニューラルネットにおいてもほぼ共通する問題らしい．

- 例えば，単体のユニットuからvへの誤差の伝播について解析する．ステップtにおける任意のユニッ

トuで発生した誤差がqステップ前のユニットvに伝播する状況を考えたとき，誤差は以下に示すような係数でスケールする．

$$\frac{\partial v_v(t-q)}{\partial v_u(t)} = \begin{cases} f_v'(net_v(t-1))w_{uv} & q=1 \\ f_v'(net_v(t-q))\sum_{l=1}^{n}\frac{\partial v_v(t-q+1)}{\partial v_u(t)}w_{lv} & q>1 \end{cases}$$

- $l_q = v$ と$l_0 = u$を使用して、

$$\frac{\partial v_v(t-q)}{\partial v_u(t)} = \sum_{l_1=1}^{n}\cdots\sum_{l_{q-1}=1}^{n}\prod_{m=1}^{q} f_{l_m}'(net_{l_m}(t-m))w_{l_m l_{m-1}}$$

- 上式より，以下の場合はスケール係数は発散し，その結果としてユニットvに到着する誤差の不安定性により学習が困難になる．

$$|f_{l_m}'(net_{l_m}(t-m))w_{l_m l_{m-1}}| > 1.0 \quad for\ all\ m$$

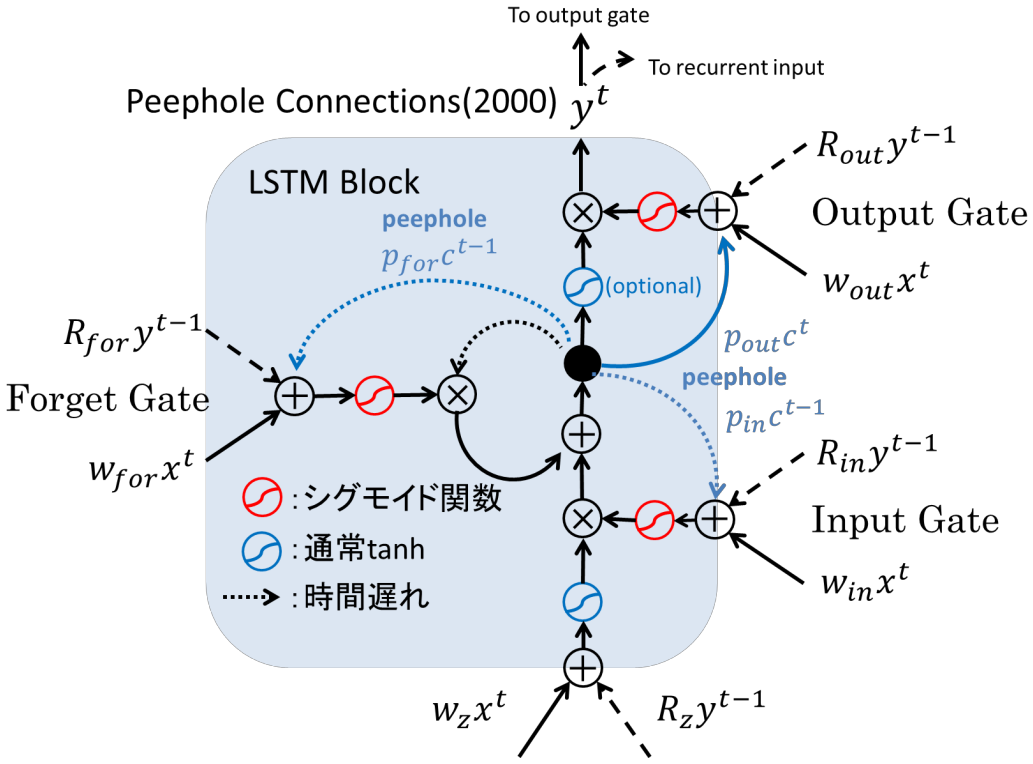
- 一方，以下の場合はスケール係数はqに関して指数関数的に減少する．

$$|f_{l_m}'(net_{l_m}(t-m))w_{l_m l_{m-1}}| < 1.0 \quad for\ all\ m$$

- これらの問題を解決するために考案されたのがLSTM

LSTMモデル

- RとWは重み行列



LSTMの順伝播計算

$$\bar{z}^t = W_z x^t + R_z y^{(t-1)} + b_z,\ z^t = g(\bar{z}^t)$$

$$\bar{i}^t = W_{in} x^t + R_{in} y^{(t-1)} + p_{in} \odot c^{t-1} + b_{in},\ i^t = \sigma(\bar{i}^t)$$

$$\bar{f}^t = W_{for} x^t + R_{for} y^{(t-1)} + p_{for} \odot c^{t-1} + b_{for},\ f^t = \sigma(\bar{f}^t)$$

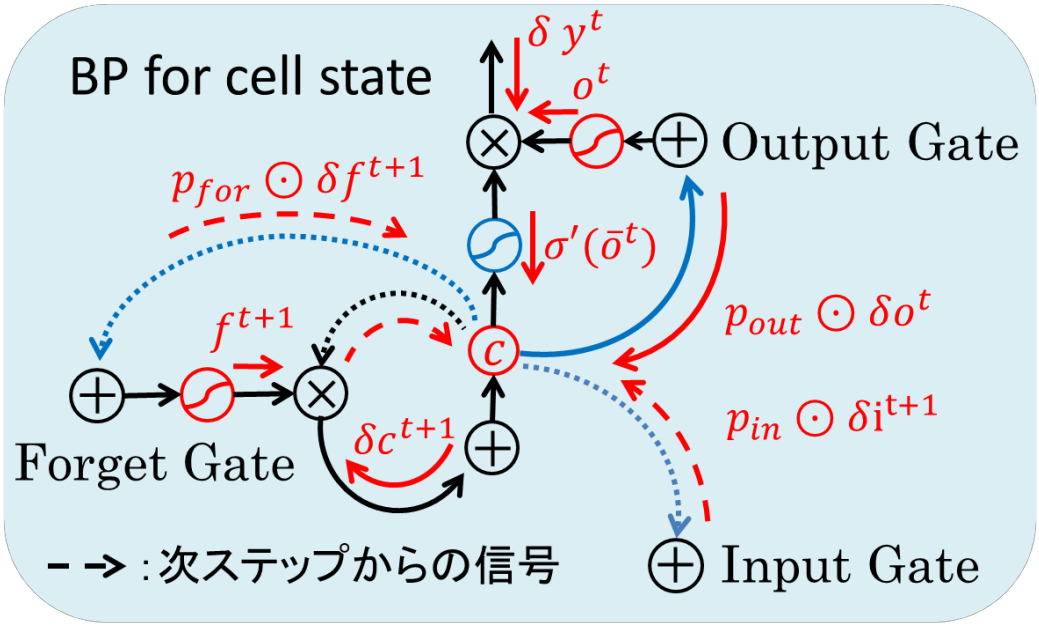
$$c^t = i^t \odot z^t + f^t \odot c^{t-1}$$

$$\bar{o}^t = \sigma(W_{out} x^t + R_{out} y^{(t-1)} + p_{out} \odot c^t + b_{out}),\ o^t = \sigma(\bar{o}^t)$$

$$y^t = o^t \odot h(c^t)$$

$$s.t.\ \sigma(x) = sigmoid(x) = \frac{1}{1+e^{-x}},\ g(x) = h(x) = tanh(x)$$

逆伝播



$$\delta y^t = \Delta^t + R_z^T \delta z^{t+1} + R_{in}^T \delta i^{t+1} + R_{for}^T \delta f^{t+1} + R_{out}^T \delta o^{t+1}$$

$$\delta o^t = \delta y^t \odot h(c^t) \odot \sigma'(\bar{o}^t)$$

$$\delta c^t = \delta y^t \odot o^t \odot h'(c^t) + p_{out} \odot \delta o^t + p_{in} \odot \delta i^{t+1} + p_{for} \odot \delta f^{t+1} + \delta c^{t+1} \odot f^{t+1}$$

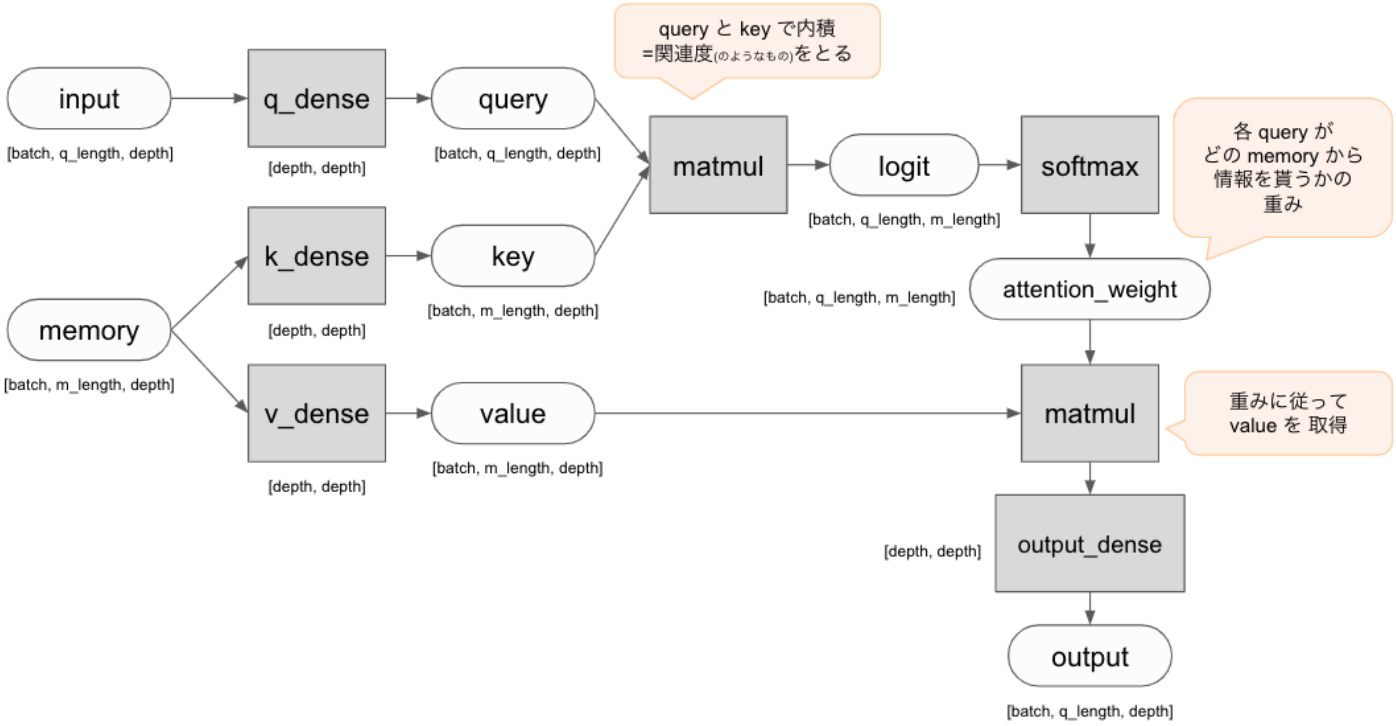
$$\delta f^t = \delta c^t \odot c^{t-1} \odot \sigma'(\bar{f}^t)$$

$$\delta i^t = \delta c^t \odot z^t \odot \sigma'(\bar{i}^t)$$

$$\delta z^t = \delta c^t \odot i^t \odot g'(\bar{z}^t)$$

Attention

- 論文: Effective Approaches to Attention-based Neural Machine Translation



- Attentionの基本は$query$と$memory$(key, $value$).
- Attentionとは$query$によって$memory$から必要な情報を選択的に引っ張ってくること．$memory$から情報を引っ張ってくるときには，$query$はkeyによって取得する$memory$を決定し，対応する$value$を取得する.

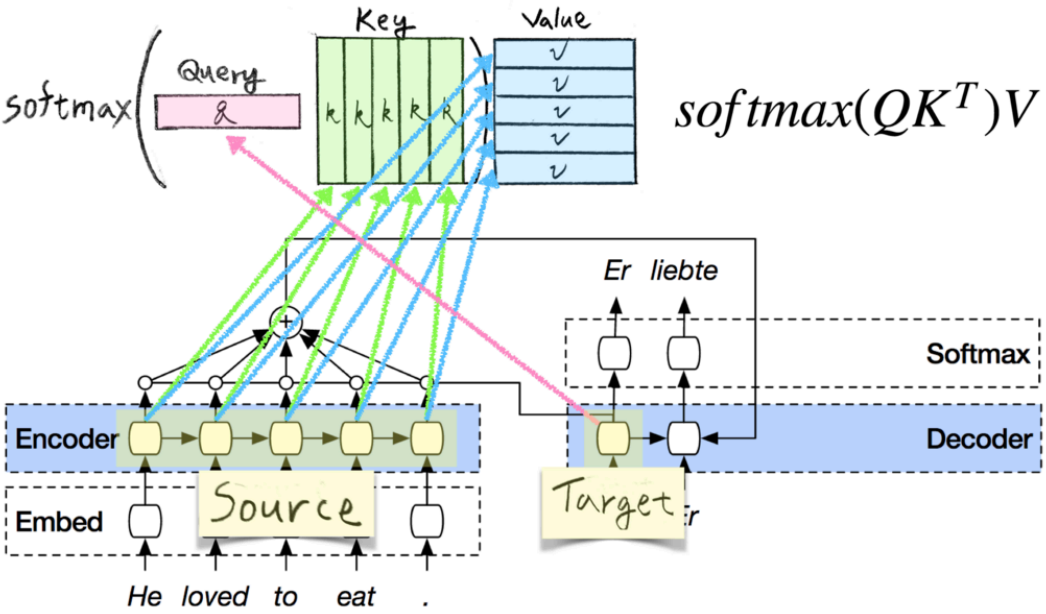
Encoder-Decoderにおけるattention

- 一般的なEncoder-Decoderの注意はエンコーダの隠れ層を$Source$，デコーダの隠れ層を$Target$として次式によって表される.

$$Attention(Target, Source) = Softmax(Target \cdot Source^T) \cdot Source$$

- より一般化すると$Target$を$query$(検索クエリ)と見做し，$Source$をKeyと$Value$に分離する.

$$Attention(query, Key, Value) = Softmax(query \cdot Key^T) \cdot Value$$

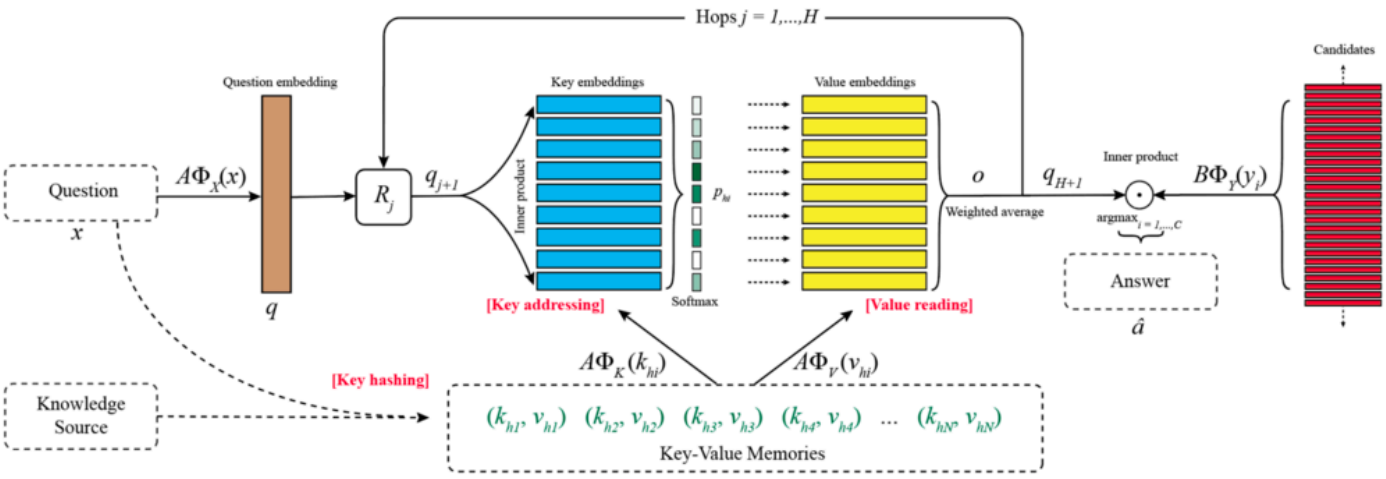


- この時Keyと$Value$は各keyと各$value$が一対一対応するkey-valueペアの配列，つまり辞書オブジェクトとして機能する.
- $query$とKeyの内積は$query$と各keyの類似度を測り，$softmax$で正規化した注意の重み (Attention Weight) は$query$に一致したkeyの位置を表現する．注意の重みと$Value$の内積はkeyの位置に対応する$value$を加重和として取り出す操作である.
- つまり注意とは$query$(検索クエリ)に一致するkeyを索引し，対応する$value$を取り出す操作であり，これは辞書オブジェクトの機能と同じである．例えば一般的な Encoder-Decoder の注意は，エンコーダのすべての隠れ層 (情報源)$Value$から$query$に関連する隠れ層 (情報)$value$を注意の重みの加重和として取り出すことである.

query の配列 Query が与えられれば，その数だけ key-value ペアの配列から value を取り出す.

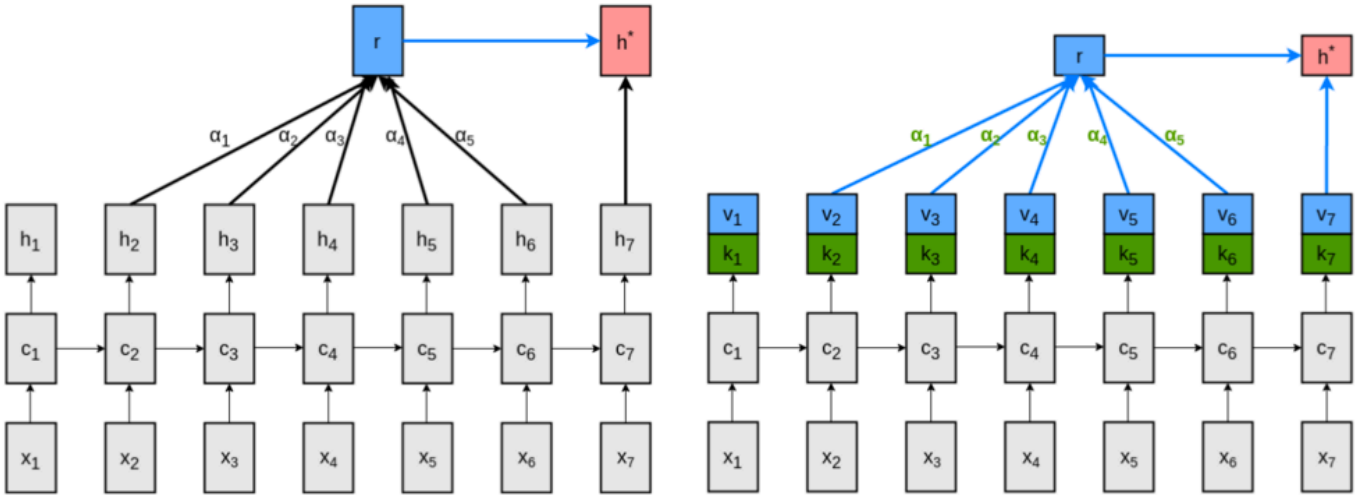
MemoryをKeyとValueに分離する意味

- key-valueペアの配列の初出は End-To-End Memory Network [Sukhbaatar, 2015] であるが，Keyを Input，$Value$を Output (両方を合わせて Memory) と表記しており，辞書オブジェクトという認識はなかった.
- (初めて辞書オブジェクトと認識されたのは Key-Value Memory Networks [Miller, 2016] である．)



- Key-Value Memory Networks では key-value ペアを文脈 (e.g. 知識ベースや文献) を記憶として格納する一般的な手法だと説明している．**$Memory$をKeyと$Value$に分離することでkeyと$value$間の非自明な変換によって高い表現力が得られる**という．ここでいう非自明な変換とは，例えば「keyを入力して$value$を予測する学習器」を容易には作れない程度に複雑な (予測不可能な) 変換という意味である．

- その後，言語モデルでも同じ認識の手法 [Daniluk, 2017] が提案されている．

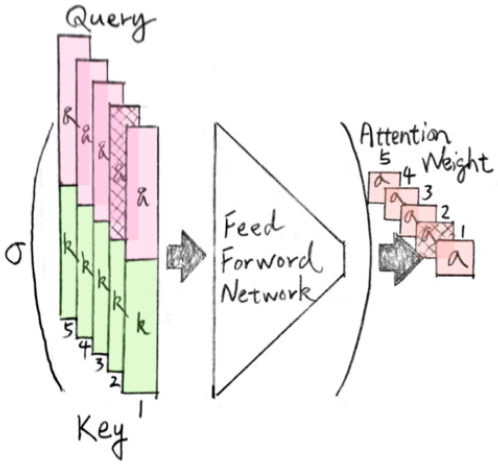


(a) Neural language model with attention.　　(b) Key-value separation.

attentionのweightの算出方法

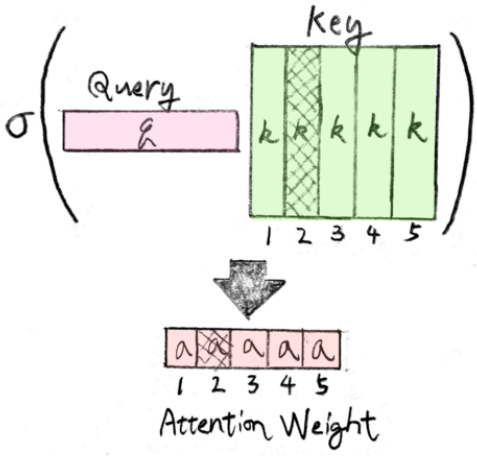
加法注意
(Additive Attention)

$$softmax(FFN([Q;K]))$$



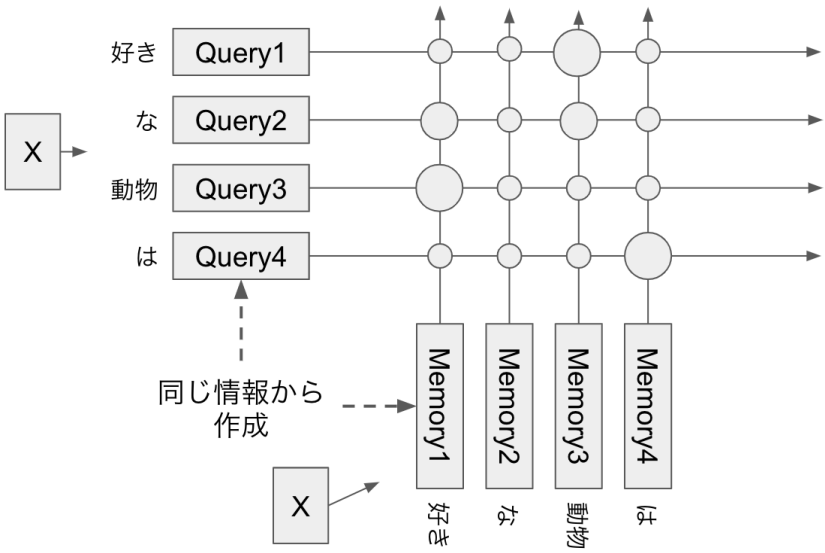
内積注意
(Dot-Product Attention)

$$softmax(QK^T)$$



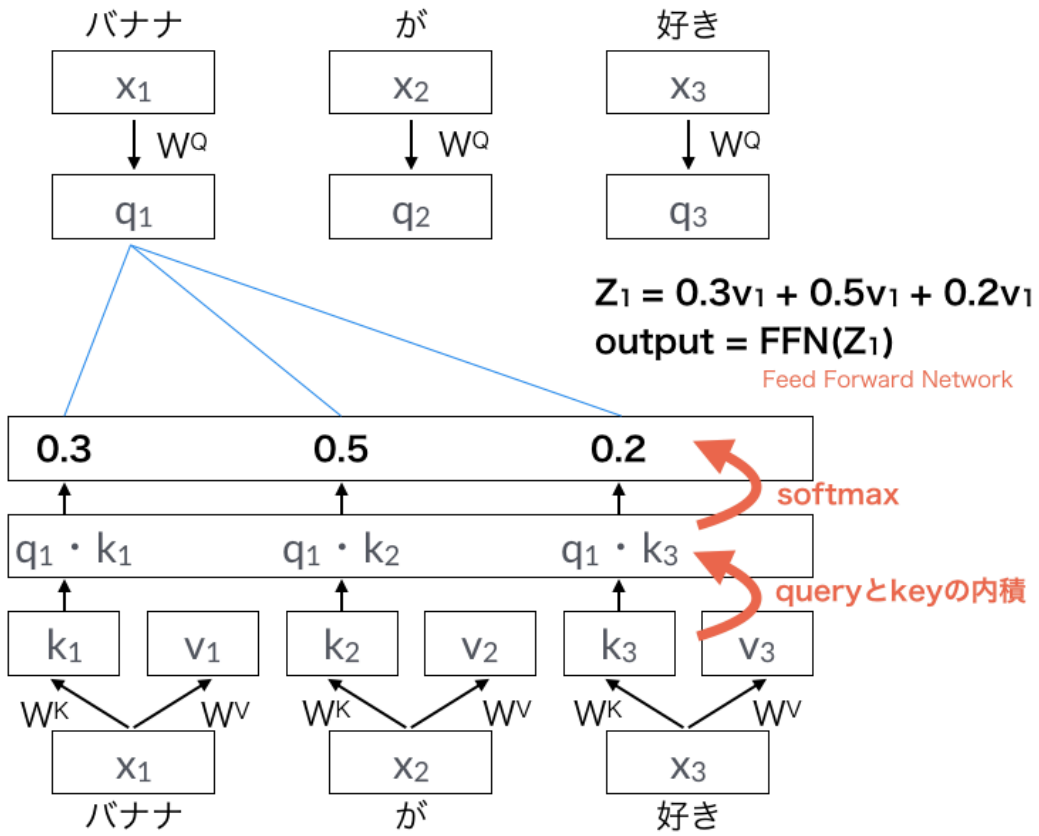
- 加法注意と内積注意があり，加法注意は一層のフィードフォワードネットワークで重みを算出する一方，内積注意はattentionの重みを$query$とkeyの内積で算出する．こちらは前者に比べてパラメータが必要ないため，効率よく学習ができる．

self-attention

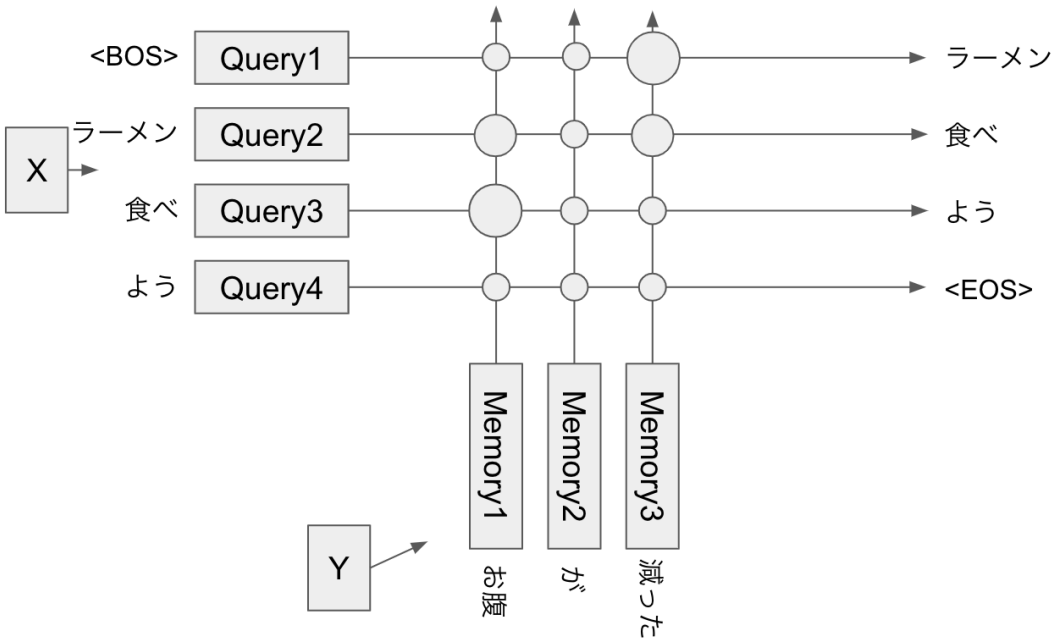


- $input(query)$と$memory(key, value)$すべてが同じTensorを使うAttention
- Self-Attentionは言語の文法構造であったり，照応関係（its が指してるのは Law だよねとか）を獲得するのにも使われているなどと論文では分析されている

- 例えば「バナナが好き」という文章ベクトルを自己注意するとしたら，以下のような構造になる．



Source-Target Attention

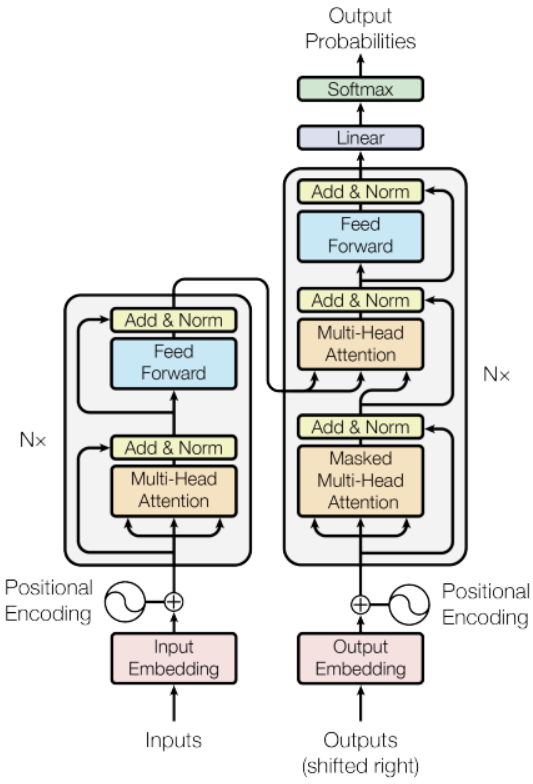


- Transformerではdecoderで使われる.

Transformer

- 論文: Attention Is All You Need

- 論文タイトルにもある通り，ATTENTION IS ALL YOU NEED．つまりRNNやCNNを使わずattentionのみを使用した機械翻訳タスクを実現するモデル.
- Google プロジェクトページ
- Pytorch model Github



- モデルの概要は以下の通り
  - エンコーダ: [自己注意, 位置毎の FFN] のブロックを6層スタック
  - デコーダ: [(マスキング付き) 自己注意, ソースターゲット注意, 位置毎の FFN] のブロックを6層スタック
- ネットワーク内の特徴表現は [単語列の長さ x 各単語の次元数] の行列で表される．注意の層を除いて0階の各単語はバッチ学習の各標本のように独立して処理される.
- 訓練時のデコーダは自己回帰を使用せず，全ターゲット単語を同時に入力，全ターゲット単語を同時に予測する．ただし予測すべきターゲット単語の情報が予測前のデコーダにリークしないように自己注意にマスクをかけている (ie, **Masked Decoder**)．評価/推論時は自己回帰で単語列を生成する.
- Transformerでは内積注意を縮小付き内積注意 (Scaled Dot-Product Attention) と呼称する．通常の内積注意と同じく$query$をもとにkey-valueペアの配列から加重和として$value$を取り出す操作であるがQとKの内積をスケーリング因子$\sqrt{d_k}$で除算する.
- また，$query$の配列は1つの行列Qにまとめて同時に内積注意を計算する (従来通りkeyと$value$の配列も K,V にまとめる).

縮小付き内積注意

- 縮小付き内積注意は以下のように表される.

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V$$

- Mask (option) はデコーダの予測すべきターゲット単語の情報が予測前のデコーダーにリークしないように自己注意にかけるマスクである (Softmax への入力のうち自己回帰の予測前の位置に対応する部分を1で埋める).

- Transformer では縮小付き内積注意を1つのヘッドと見做し，複数ヘッドを並列化した複数ヘッドの注意 (Multi-Head Attention) を使用する．ヘッド数$h = 8$と各ヘッドの次元数$d_{model}/h = 64$はトレードオフなので合計のパラメータ数はヘッド数に依らず均一である.

複数ヘッドの注意

- $d_{model} = 512$次元のQ, K, Vを用いて単一の内積注意を計算する代わりに，Q, K, Vをそれぞれ$h = 8$回異なる重み行列 W_i^Q, W_i^K, W_i^V で $d_k, d_k, d_v = 64$ 次元に線形写像して$h = 8$個の内積注意を計算する．各内積注意の$d_v = 64$次元の出力は連結 (concatenate) して重み行列W_oで$d_{model} = 512$次元に線形写像する.

- 複数ヘッドの注意は次式によって表される.

$$\begin{aligned}\mathrm{MultiHead}(Q, K, V) &= \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_\mathrm{h})W^O \\ \text{where } \mathrm{head}_\mathrm{i} &= \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

$$W_i^Q \in \mathbb{R}^{d_\mathrm{model} \times d_k}, W_i^K \in \mathbb{R}^{d_\mathrm{model} \times d_k}, W_i^V \in \mathbb{R}^{d_\mathrm{model} \times d_v}, W^O \in \mathbb{R}^{hd_v \times d_\mathrm{model}}$$

位置毎のフィードフォワードネットワーク

- FFNは以下のように表される

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

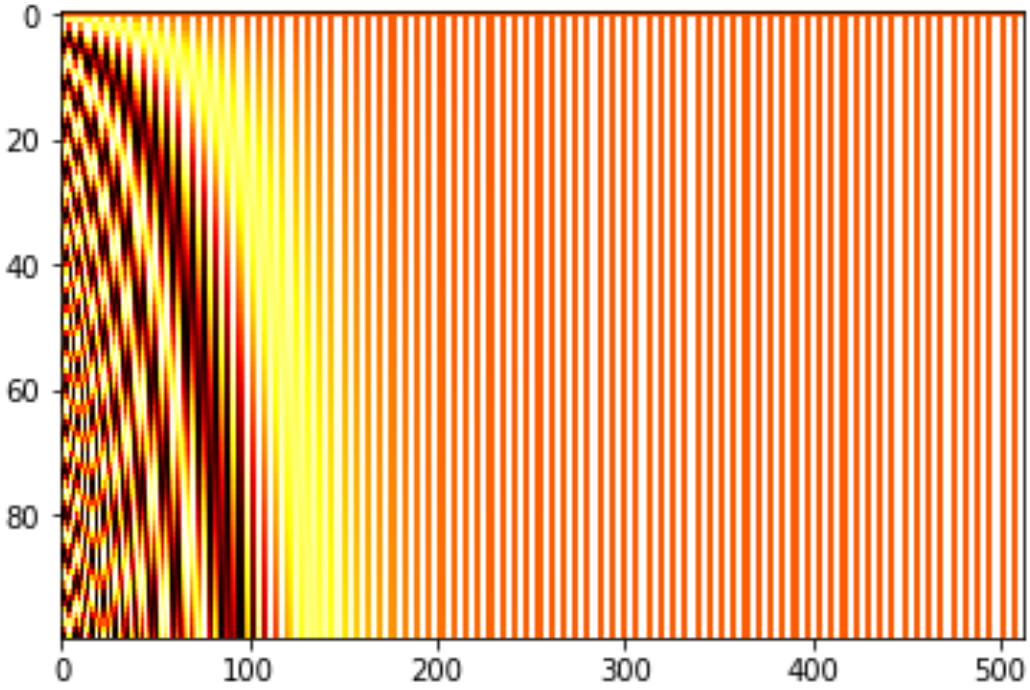
- $ReLU$で活性化する$d_{ff} = 2048$次元の中間層と$d_{model} = 512$次元の出力層から成る2層の全結合ニューラルネットワークである.

位置エンコーディング

- **TransformerはRNNやCNNを使用しないので単語列の語順(単語の相対的ないし絶対的な位置)の情報を追加する必要がある.**
- 本手法では入力の埋め込み行列(Embedded Matrix)に**位置エンコーディング(Positional Encoding)の行列PEを要素ごとに加算**する.
- 位置エンコーディングの行列PEの各成分は次式によって表される.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

- ここでposは単語の位置，iは成分の次元である．位置エンコーディングの各次元は波長が2πから$10000 \cdot 2\pi$に幾何学的に伸びる正弦波に対応する.



- 横軸が単語の位置(0 ~ 99)，縦軸が成分の次元(0 ~ 511)，濃淡が加算する値(-1 ~ 1).

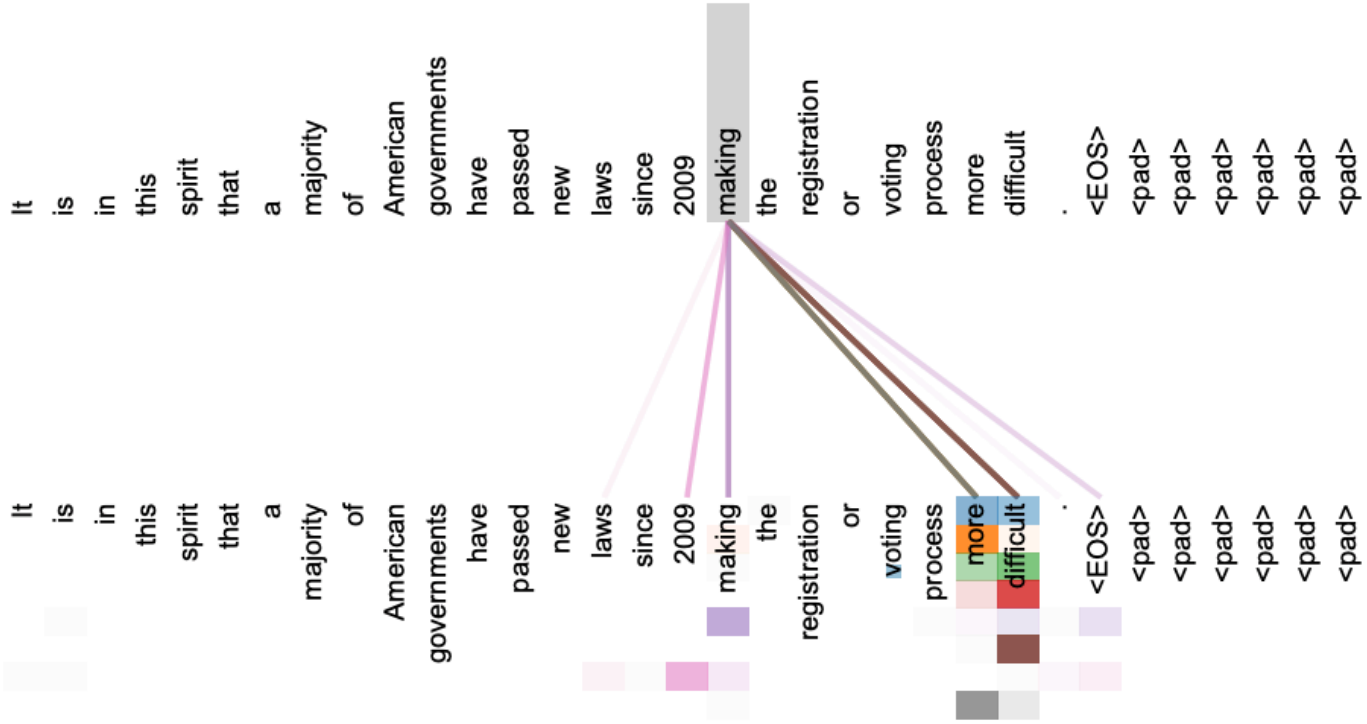
TransformaerにおけるAttention

- BERTの基本単位を構成するTransformerは言語タスクにおいて，人間の直感と近い注意の仕方をしていることが論文に記載されている.
- ポジネガ極性判定タスクを解かせると，極性をよく表す箇所にAttentionが当たっている様子が見える.

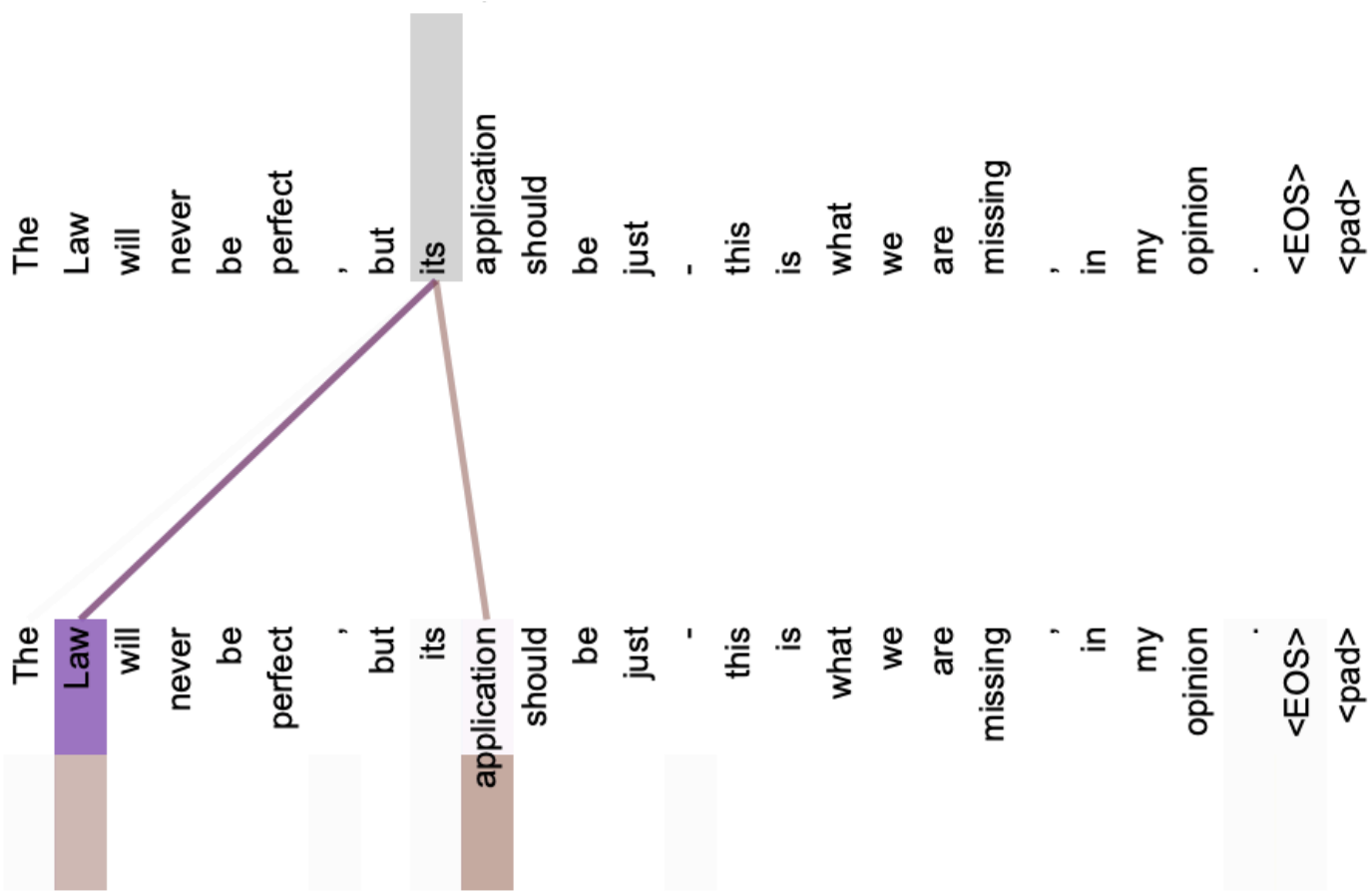
- love this place it really be my favorite restaurant in Charlotte they use charcoal for their grill and you can taste it steak with chimichurri be always perfect Fried yucca cilantro rice pork sandwich and the good tres lech I have had.The desert be all incredible if you do not like it you be a mutant if you will like diabeetus try the Inca Cola

- ネットワークがタスクに応じて必要な情報に注意できることから，BERTの事前学習でも予測単語を推測するための文章全体からの周辺情報の活用と，隣接分予測のための文章の構造および大意を把握する情報に注意を向ける傾向があると思われる．

- 以下の画像は `query` が `making` だった場合のAttention状況を示している．上がQueryで下がValue．`making` に対して `more` や `difficult` などに強いAttentionがあたっており、`making ... more difficult` という長距離で関係を持つ句関係を捉えていることがわかる．



- 次の画像は `query` が `its` だった場合のAttention状況．`Law` と `application` にAttentionがかかっており、`its = Law = application` という照応関係を捉えていることがわかる．

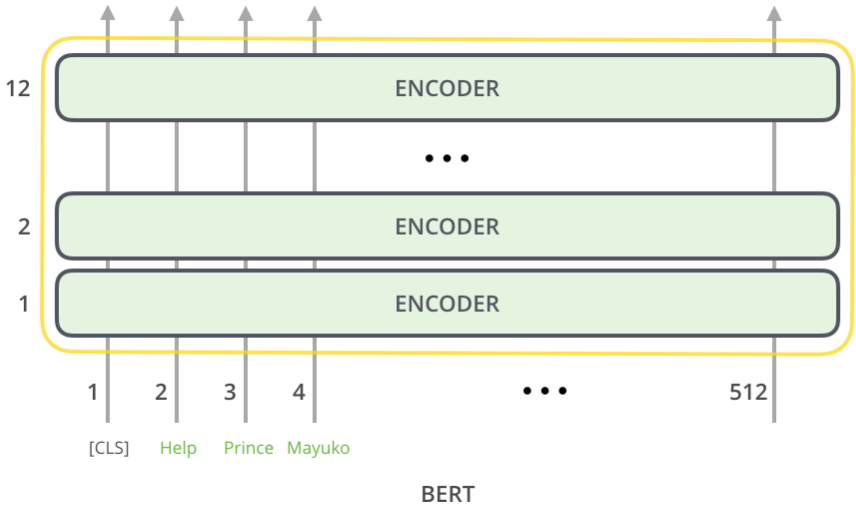


BERT

- 論文: [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding]

概要

- 単語の分散表現を獲得するための機構．TransformerのEncoderブロックから構成される．



- ネットワーク側ではなく学習データ側にマスクをかけてあげることで双方向transformerが実現した．

下図がモデルの概要.

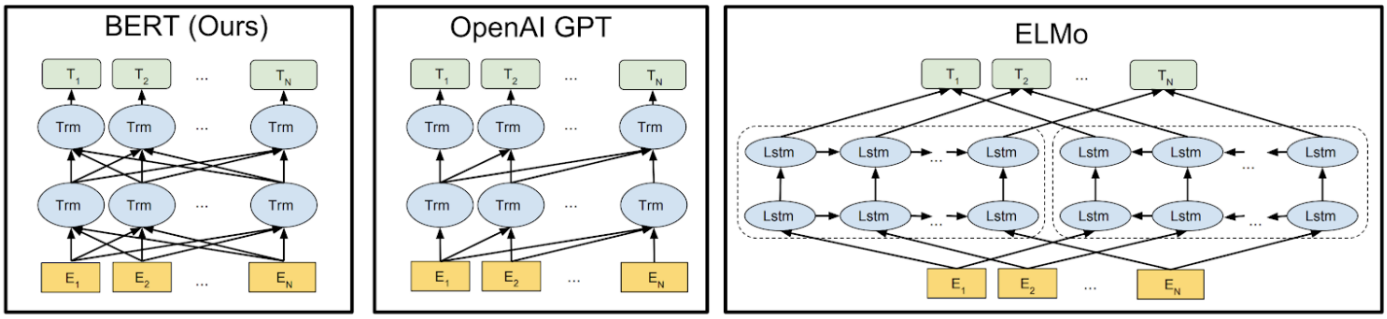
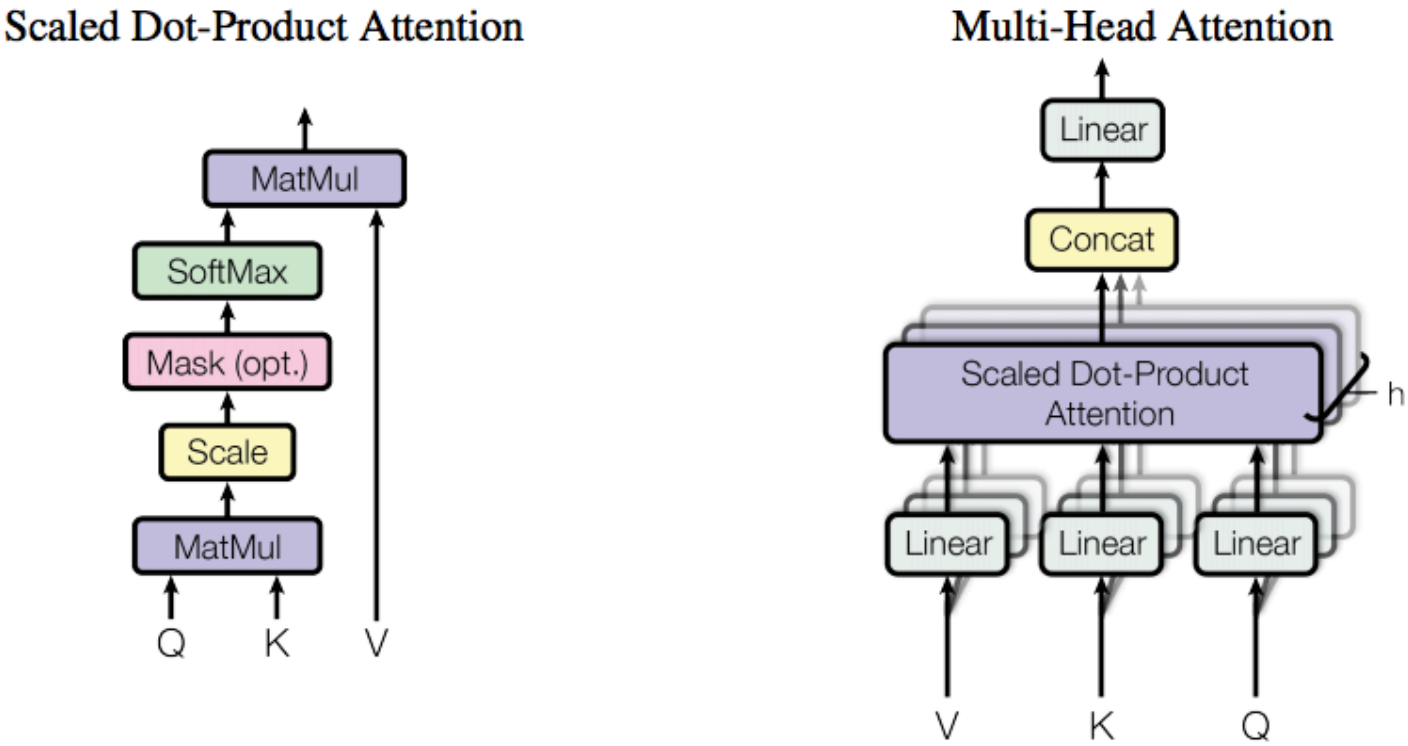


Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

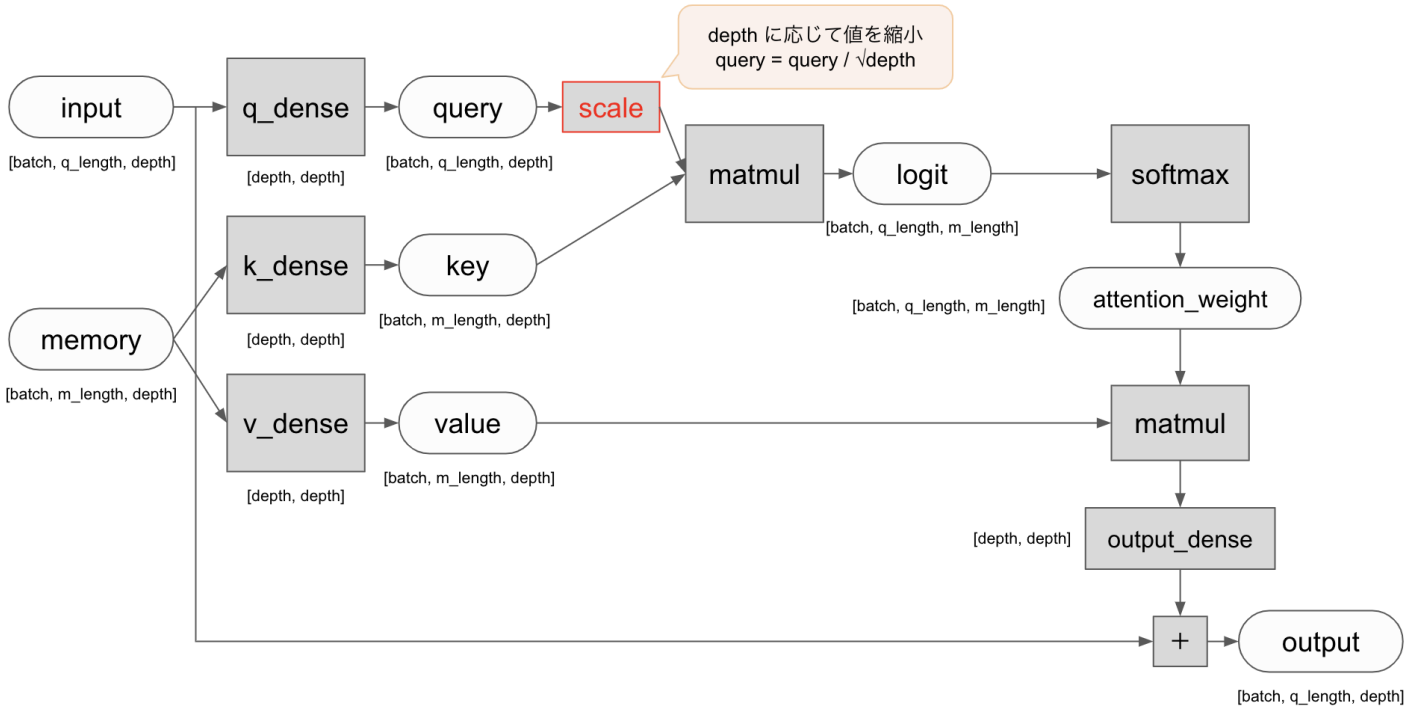
- transformerモデルのEncoder部分を全結合的に接続したのがBERTモデル.



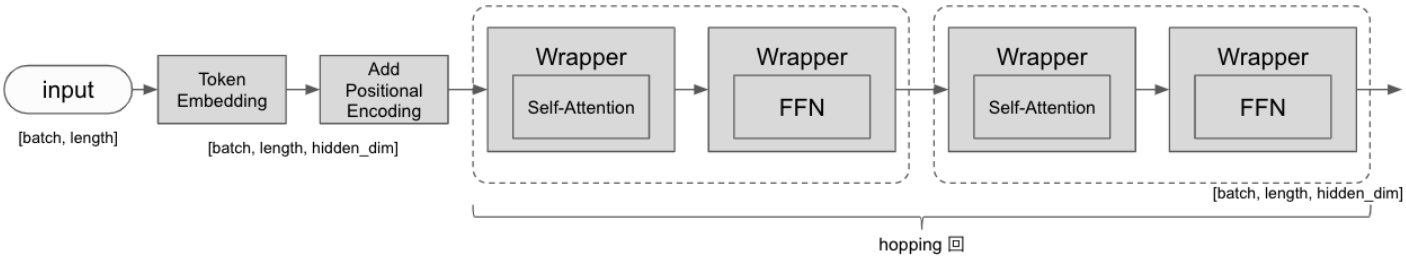
- 上図のScaled Dot-Product Attentionはself-attention. attentionの重みを計算する際, softmaxで値が大きくなった時に勾配が0にならないようにsoftmaxのlogitのqueryとkeyの行列積を以下のように調整してあげる.

$$attenton\ weight = Softmax(\frac{qk^T}{\sqrt{depth}})$$

$$where\ depth = dim\ of\ embedding$$



- 使用するTransformerのEncoderは以下のようになっている(しかしattentionは一つ．)



事前学習タスク

- どちらもBERTからはきだされた内部状態テンソルをInputとして一層のMLPでクラス分類しているだけ.
- これらを用いてBERTの事前学習を行う

事前学習1 マスク単語の予測

- 系列の15%を[MASK]トークンに置き換えて予測
- そのうち80%がマスク，10%がランダムな単語，10%を置き換えない方針で変換する

```python
class MaskedLanguageModel(nn.Module):
    """
    入力系列のMASKトークンから元の単語を予測する
    nクラス分類問題, nクラス : vocab_size
    """

    def __init__(self, hidden, vocab_size):
        """
        :param hidden: output size of BERT model
        :param vocab_size: total vocab size
        """
        super().__init__()
        self.linear = nn.Linear(hidden, vocab_size)
        self.softmax = nn.LogSoftmax(dim=-1)

    def forward(self, x):
        return self.softmax(self.linear(x))
```

事前学習2 隣接文の予測

- 二つの文章を与え隣り合っているかをYes/Noで判定
- 文章AとBが与えられた時に，50%の確率で別の文章Bに置き換える

```python
class NextSentencePrediction(nn.Module):
    """
    2クラス分類問題 : is_next, is_not_next
    """

    def __init__(self, hidden):
        """
        :param hidden: BERT model output size
        """
        super().__init__()
        self.linear = nn.Linear(hidden, 2)
        self.softmax = nn.LogSoftmax(dim=-1)

    def forward(self, x):
        return self.softmax(self.linear(x[:, 0]))
```

BERTモデルの応用

- 事前学習を行ったモデルを使って様々なタスクへの応用が行われている.