深層学習における革新的アーキテクチャ

Attention is all you need

目次

3.1Encoder構造

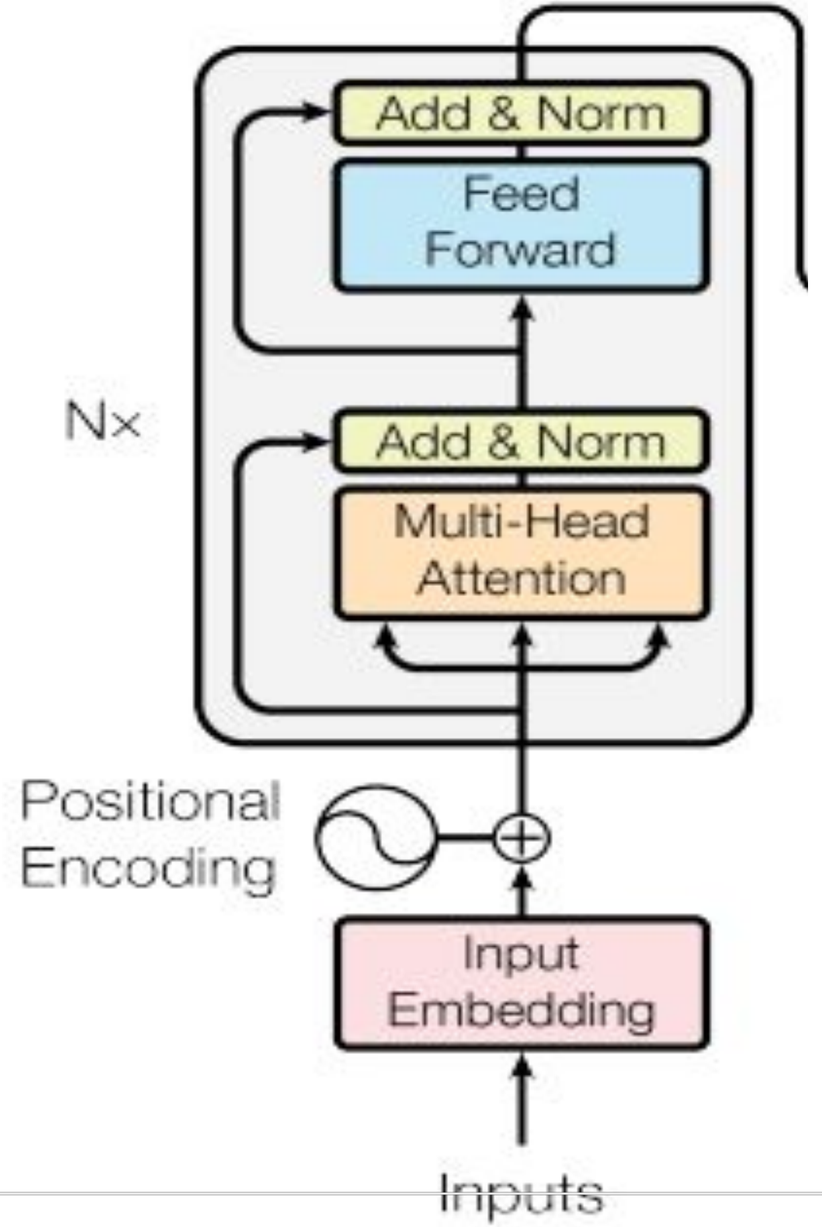
Encoderの出力を受け取り、
目的の系列を生成する役割

2つのSublayer

- Encoder同様の層構造
- 追加のAttention層
- マスキングによる未来情報遮断

Attention機構の特徴

- q,k,vを用いた計算
- Encoderからの情報利用
- 自己回帰的な出力生成



3.1Decoder構造

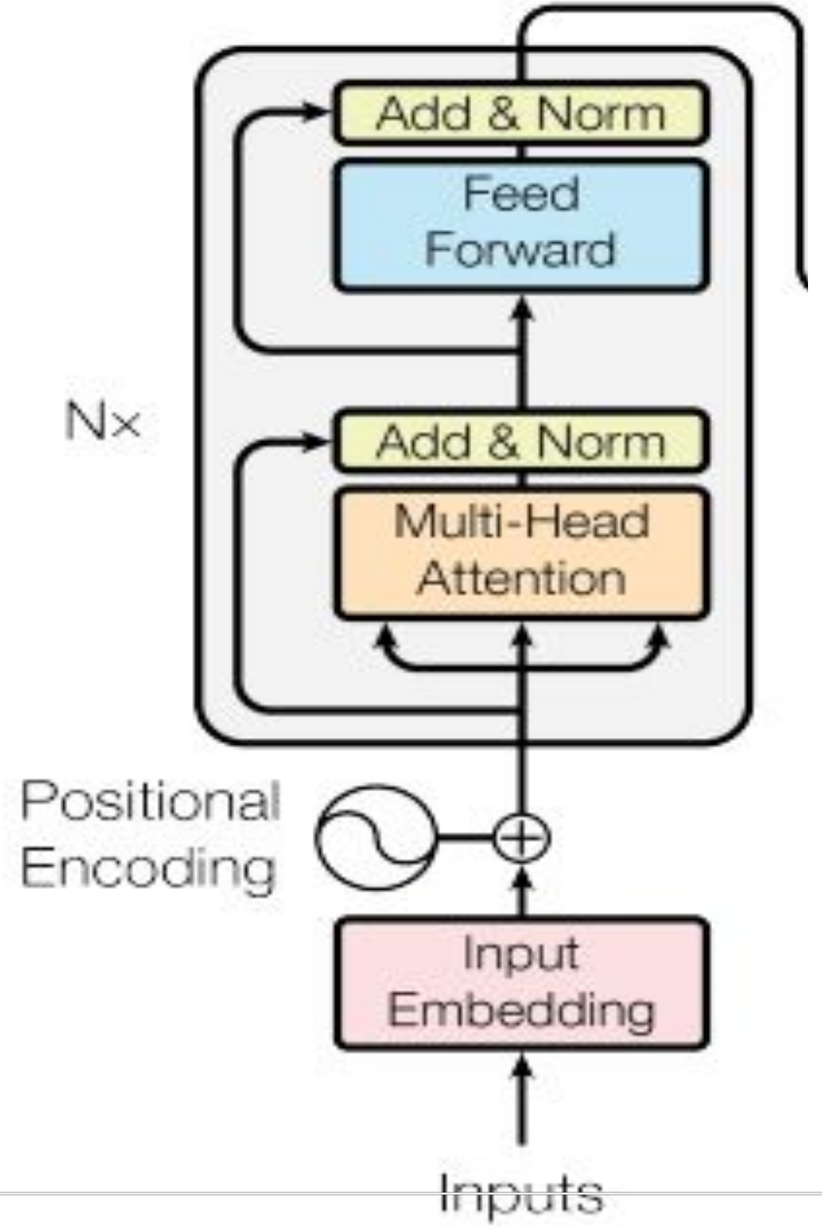
Encoderの出力を受け取り、
目的の系列を生成する役割

Encoderとの類似点と相違点

- Encoder同様の層構造
- 追加のAttention層
- マスキングによる未来情報遮断

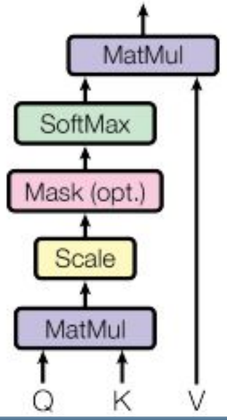
Attention機構の特徴

- q,k,vを用いた計算
- Encoderからの情報利用
- 自己回帰的な出力生成



3.2.1Scaled Dot-Product Attention

01



通常の方式との違い

- 内積に基づく注意機構
- クエリと全キーの類似度計算
- ソフトマックスで重み付け
- 値ベクトルの重み付き和

02

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

スケーリングの効果

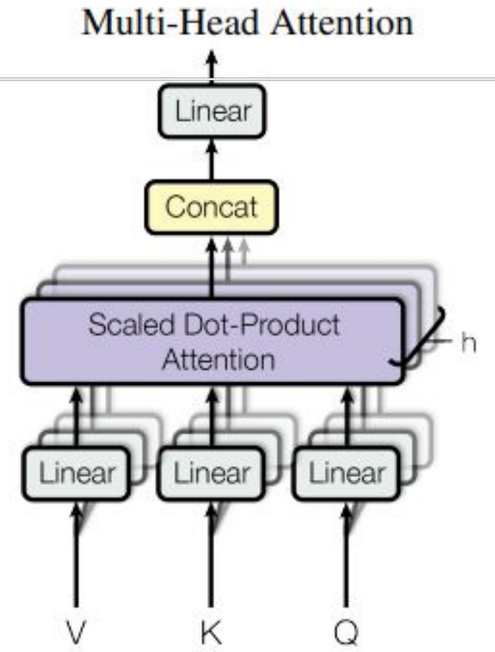
- 1/√(d_k)でスケーリング
- d_kは注意機構の次元数
- 勾配消失問題を回避
- 安定した学習を実現

3.2.2Multi-Head Attention

線形射影

- q, k, vを線形射影
- dk, dk, dvの次元にh回射影
- 並列にattention function適用
- 最後に結果を連結

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

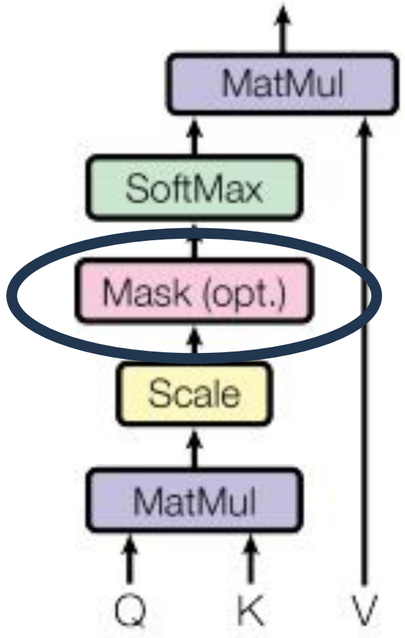


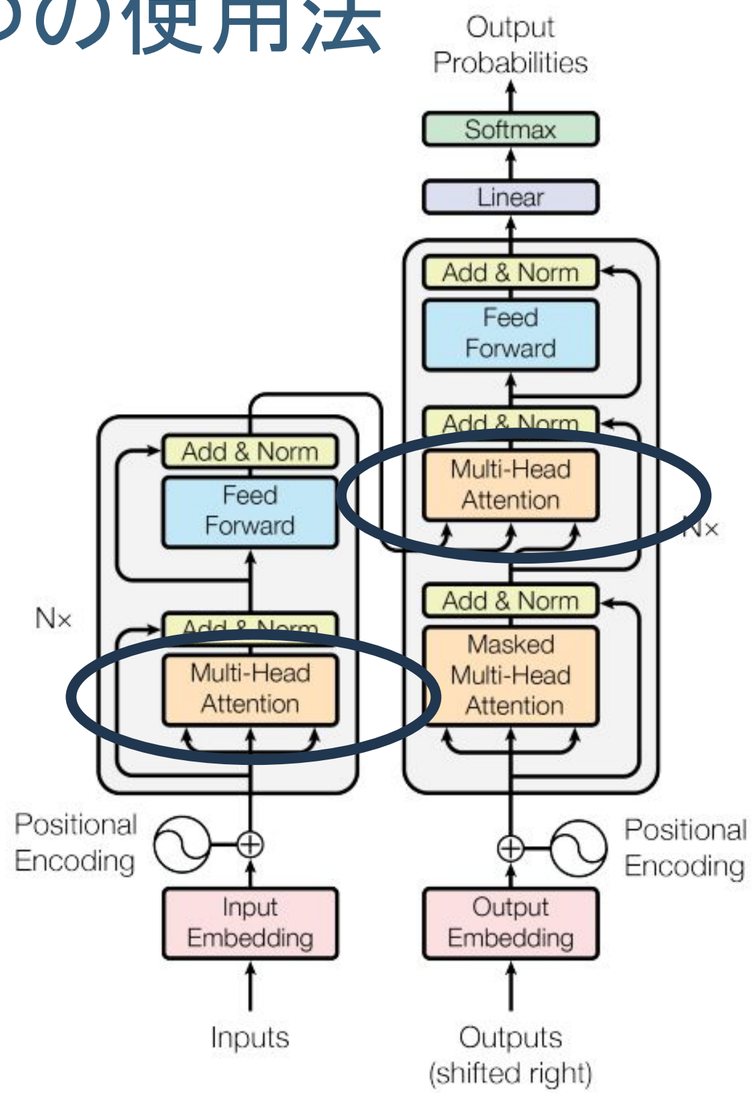
並列処理と結果の連結

- 複数の注意機構を並列処理
- 異なる表現部分空間を学習
- 効率的な情報抽出を実現

3.2.3Multi-head attention の3つの使用法

- encoderから
  key,valueを受け取る
- encoderで
  Self-attention層がある
- 無効な値の事実的抹消
  （未来の値は−∞）





3.3Position-wise FFN

01.構造

- 全結合層
- 2回の線形変換

02.数式

FFN(x) =max(0,xW1+b1)W2+b2

03.特徴

- 非線形性
- 層間の情報変換

0.4役割

- 特徴抽出
- モデルの表現力向上

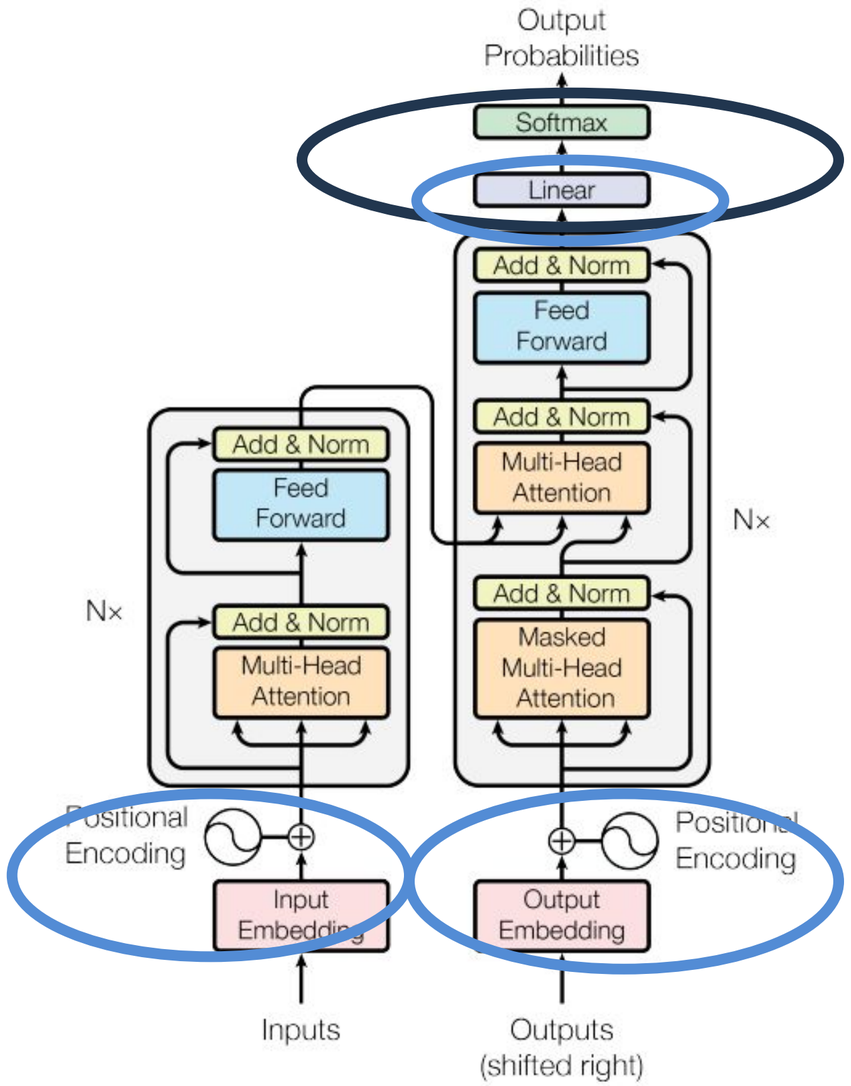
3.4Embedding and Softmax

黒.decoder output

-次の単語の発生確率を表す
- Softmax,線形変換を適用

青.weight matrix

- 同じ重み行列の使用
- 計算すべきパラメータ減る



3.5Positional Encoding

トークンの位置情報付与

- 系列内での相対的位置
- 絶対的位置の情報を提供
- トークンの順序を保持

sine･cosine関数使用

- 周期関数を利用
- 位置に応じた固有の値の**加算**
- 学習不要なエンコーディング

PE関数定義

$$\mathrm{PE}_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right)$$
$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right)$$

特徴1

- input embeddingと同じ次元
- 相対位置の計算が容易
- 加減算で比較

特徴2

- 長い系列に対応可能
- 外挿能力

3.5Positional Encoding(イメージ図)

PE関数定義

$$\mathrm{PE}_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right)$$
$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right)$$

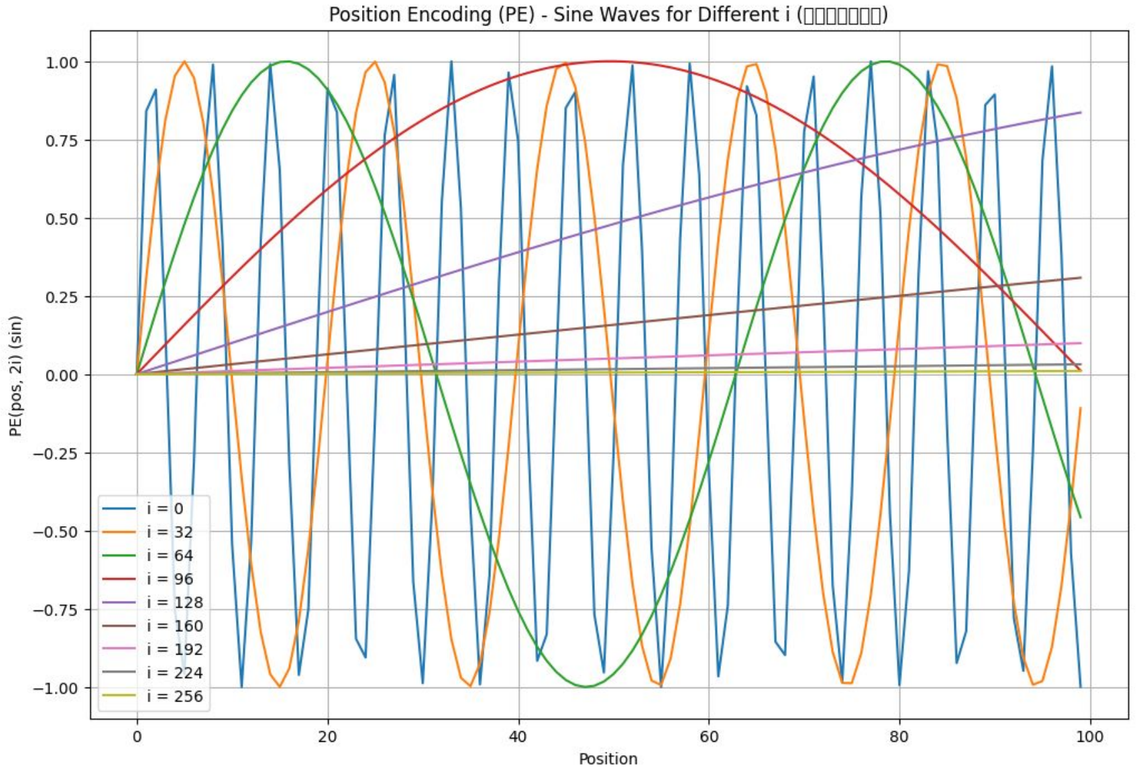
```python
import numpy as np
import matplotlib.pyplot as plt

# 位置の範囲（例として 0〜99）
pos = np.arange(0, 100, 1)
d_model = 512  # モデルの次元
i_values = [0, 32, 64, 96, 128, 160, 192, 224, 256]  # より細かい i の値

plt.figure(figsize=(12, 8))

for i in i_values:
    sin_wave = np.sin(pos / (10000 ** (2 * i / d_model)))
    plt.plot(pos, sin_wave, label=f"i = {i}")

plt.title('Position Encoding (PE) - Sine Waves for Different i (細かく刻んだ例)')
plt.xlabel('Position')
plt.ylabel('PE(pos, 2i) (sin)')
plt.legend()
plt.grid(True)
plt.show()
```



4 Why Self-Attention

層ごとの計算量

- 自己注意機構により効率的
- RNNやCNNと比較して優れた計算量

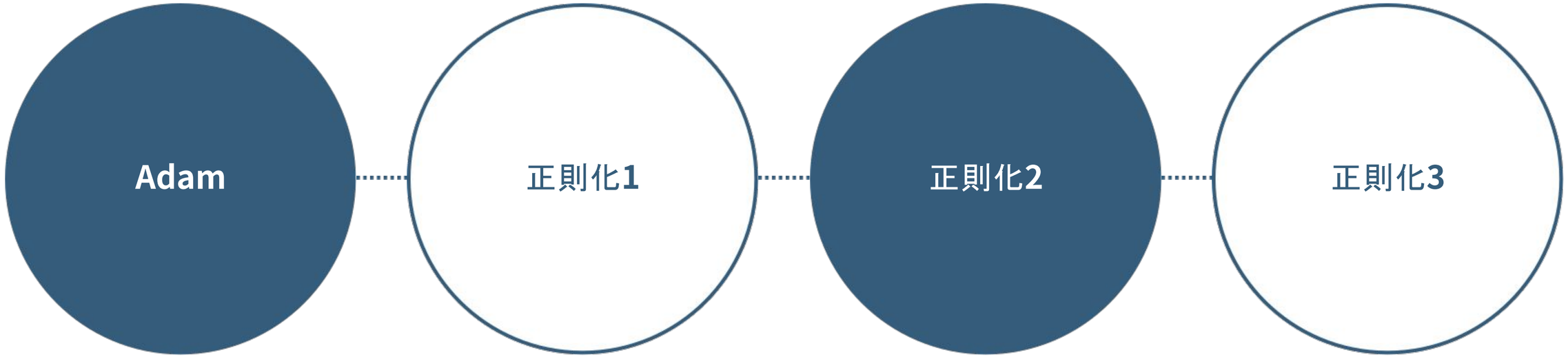
並列化による計算効率

- 各層で並列計算が可能
- GPUの活用で高速処理
- 学習時間の大幅な短縮

長距離依存関係の処理

- 全ての位置間の直接的な接続
- 固定長の計算ステップで情報伝達
- 長距離依存関係の効率的な学習

Transformerの学習方法



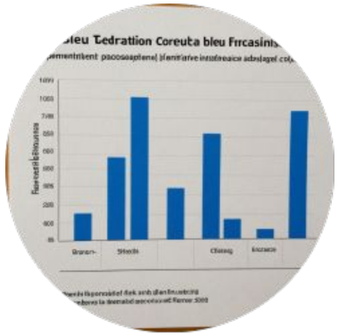
- 適応的学習率
- モメンタム使用
- 効率的な最適化

- サブレイヤー出力
- ドロップアウト適用
- 過学習防止

- 埋め込み
- 位置エンコーディング
- 合計にドロップアウト

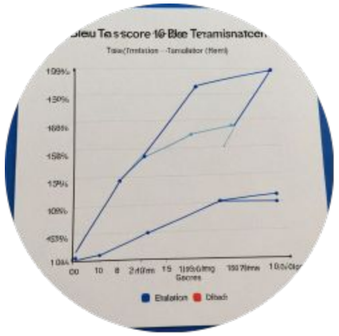
- ラベル平滑化

実験結果



EN-FR
タスク

- 41.0の過去最高BLEUスコア
- 訓練時間が短い



EN-DE
タスク

- 最良BLEUスコアモデルより2以上高い

BLEU
スコア

- 適切なhの調整
- dkの増加
- モデル数の増加
- ドロップアウトで向上

Beam
Search

- beam size: 4
- penalty: 0.6
- 最大入力長+50の出力長

性能
比較

- 異なる条件下で実験
- BLEUスコア向上要因を特定
- モデルの効果を検証