DEIM Forum 2023 1a-7-1

深層強化学習を用いた 文章の言い換えによる駄洒落生成モデルの検討

南　智仁† 清　雄一† 田原　康之† 大須賀昭彦†

† 電気通信大学　〒 182–8585 東京都調布市調布ヶ丘 1-5-1

E-mail: † minami.tomohito@ohsuga.lab.uec.ac.jp, {seiuny,tahara,ohsuga}@uec.ac.jp

あらまし　雑談応答システムとまた話したいと思わせるためには，システムの人間らしい振る舞いが必要である．人間らしい振る舞いのひとつしてユーモアが存在する．本論文では，ユーモアの中でも駄洒落に注目し任意の文章の言い換えによる併置型駄洒落生成モデルを提案する．学習済み GPT-2 を強化学習で fine-tuning することで，より多様性のある駄洒落の生成，データセットによらない駄洒落の生成を目指した．これにより，一部のモデルでは僅かながらも駄洒落らしさの理解と文章の言い換え能力を獲得できた．さらに本論文では，畳み込みニューラルネットワークを利用した新たな駄洒落の判定モデルを提案する．これにより SVM による手法や，ルールベースの手法を Precision と F1 スコアで上回ることができた．

キーワード　深層強化学習，自然言語生成，ユーモア，駄洒落

1　はじめに

近年，雑談応答システムの発展により一般のユーザーがスマートスピーカーやチャットボットを利用することが増えてきている．対話を目的とするシステムは適切な応答をすることの他に，人間らしい応答をすることも求められている．人間らしさのひとつにユーモアがある．システムがユーモアを獲得すればより人間らしさも増すと考えられる．しかし，雑談応答システムにおいては，ユーザーの発話に対してユーモアを含めた文章を返答することは難しい．そこで，ユーモアを意図的に生成できる技術が求められる．

ユーモアの中でも駄洒落は，日本人には馴染み深いユーモアである．駄洒落の生成を目指した手法としては，入力単語をデータベースに問い合わせて駄洒落を生成する手法 [1] や，駄洒落データベース [2] を用いて訓練した Transformer を用いて駄洒落を生成する手法 [3] がある．しかし，これらは入力として与えられる「単語」を含む駄洒落の生成に特化しており，望んだ「意味」を持つ駄洒落を生成することは難しい．チャットボットなどが人間とやりとりをする場合，意味も考慮するべきである．従って，意味を考慮した駄洒落の生成手法が求められる．

このような背景を踏まえ，本論文では入力された日本文を

（1） できるだけ近い意味で

（2） できるだけ併置型駄洒落として成立している

日本文に変換するモデルを検討する．近年の自然言語生成では深層学習を用いた手法がその表現力の高さから主流であるため，本研究においても深層学習を用いた手法を採用する．自然言語生成では教師データとして入力と出力のペアを用いた教師あり学習を行うことが多いが，ここで必要になる駄洒落とその言い換え文のペアからなるデータセットは存在しない．そこで本研究では深層強化学習を用いることで，言い換えペアなしに駄洒落の生成を行う．強化学習で駄洒落らしさのスコアと入力文との類似度のスコアを最大化させることで，モデルが目的の駄洒落を生成できるようになることを狙う．

この結果，モデルは僅かながらも駄洒落らしさの理解と文章の言い換え能力を獲得できた．また，機械翻訳モデルにより作成した駄洒落の言い換えペアで fine-tuning した GPT-2 モデルの出力と同水準のスコアの駄洒落を，強化学習のみで fine-tuning した GPT-2 モデルで生成できるようになった．モデルの出力の多くは駄洒落であるとは言い難いものとなってしまったものの，一部のモデルでは駄洒落に関するスコアがベースライン手法よりも上昇したことから，教師データを必要とせず強化学習のみで駄洒落を生成できる可能性が示唆された．

さらに本論文では，畳み込みニューラルネットワーク [4] を利用した新たな駄洒落の判定モデルを提案する．これにより畠山らの手法である PRS [3] や荒木らによる SVM を用いた手法 [5] を Precision と F1 スコアで上回ることができた．このモデルの出力を駄洒落らしさのスコアとして学習に使用する．

2　関連研究

2.1　駄洒落データベースを用いた駄洒落の生成

荒木らは Web 上の駄洒落を集め，67,000 件の駄洒落からなる駄洒落データベースを構築した [2]．駄洒落には音韻的に似ている 2 つの単語「種表現」と「変形表現」が含まれている．種表現はかけられている部分，変形表現はかかる部分を表す．また表 1 に示したように，駄洒落データベースには大きく分けて，併置型と重畳型の 2 種類の駄洒落が含まれている．併置型の駄洒落とは，表 1 に示したように，種表現と変形表現が背景知識や文脈によらず，明示的に文内に存在するものである．

2.2　Transformer による駄洒落の生成

畠山らは Transformer [6] を用いて日本語の併置型駄洒落を

表 1　駄洒落の種類 [2]

種類	説明	例
併置型 (Perfect)	種表現と変形表現が字面上完全に一致しているもの	(大将) が [大賞] を獲得
併置型 (Imperfect)	種表現と変形表現が字面上一致していないもの	(きちんと) 整理された [キッチン]
重畳型	種表現が背景知識，文脈上に存在し明示的には存在しないもの	[すいま千羽鶴]

注）( ) は種表現，[ ] は変形表現を表す．

生成するモデルを提案した [3] ．お題として種表現を入力すると，それを含む併置型の駄洒落が生成される．表 2 は提案されたモデル RLM-KL$_{\mathrm{U}}$ の出力例である．RLM-KL$_{\mathrm{U}}$ は駄洒落データベース [2] に含まれる併置型駄洒落で学習されている．

表 2　畠山らのモデル RLM-KL$_{\mathrm{U}}$ により生成された駄洒落の例 [3]

先攻のせいで先公センコーかいじゃない!
新譜はシンプルなドレスを着ている!
ハッチの件はちょっとやいたよと、はっち来なさい
酒屋がさかやっかいになった!
ノベルについて述べる!

2.3　Proximal Policy Optimization

Schulman らは，方策ベースの強化学習アルゴリズムとして Proximal Policy Optimization (PPO) [7] を提案した．PPO では $r_t(\theta)$ を $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)}$ とするとき，以下のように定義される目的関数 $L^{\mathrm{CLIP}}(\theta)$ を最大化するように学習する．

$$L^{\mathrm{CLIP}}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t,\ \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right] \quad (1)$$

ここで π_θ は現在の方策，$\pi_{\theta_{\mathrm{old}}}$ は更新前の方策，$\hat{A}_t$ は Advantage 関数，ϵ はクリッピングの閾値である．また，$\pi_\theta(a_t|s_t)$ は状態 s_t で行動 a_t を取る確率，clip はクリッピング関数である．PPO では過剰なパラメータの更新を防ぐために，更新率を微小な範囲に制限する．

Advantage 関数は以下のように定義される関数である．

$$\hat{A}_t = V_t^{\mathrm{targ}} - V_\theta(s_t) \quad (2)$$

V_t^{targ} は累積報酬，$V_\theta(s_t)$ は状態 s_t における状態価値である．さらに，価値関数と方策関数がパラメータを共有する場合には次の目的関数 $L_t^{\mathrm{CLIP+VF+S}}(\theta)$ を最大化するように学習することが提案されている．

$$L_t^{\mathrm{CLIP+VF+S}}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{\mathrm{CLIP}}(\theta) - c_1 L_t^{\mathrm{VF}}(\theta) + c_2 S[\pi_\theta](s_t)\right] \quad (3)$$

ここで c_1, c_2 は係数，$V_t^{\mathrm{VF}}(\theta)$ は予測した価値と累積報酬の二乗誤差，$S[\pi_\theta](s_t)$ は方策 $\pi_\theta(\cdot|s_t)$ のエントロピーである．

2.4　TrufLL

Martin らは，言語モデルを強化学習により一から訓練する新たな手法である TRUncated ReinForcement Learning for Language (TrufLL) [8] を提案した．

TrufLL ではトークンの選択を行動選択とみなし，強化学習を用いて言語モデルを学習する．一般に，強化学習では広い行動空間に適応することが難しい．この問題を克服するために TrufLL では外部の言語モデルを用いて行動空間の切り詰め (truncation) を行っている．

図 1 は truncation の流れを図式化したものである．まず，エージェントが行動を選択する際に，エージェントとは異なる外部の言語モデルの予測する次単語の確率分布を truncation 関数 (g_{trunc}) に入力する．Truncation 関数は与えられた確率分布から特定のルールに従っていくつかのトークンを選択し，そのトークンのリストを返す．エージェントはこのリストに含まれるトークンの中から次単語をサンプリングする．これにより，エージェントは言語らしさを獲得しながらも，報酬関数を最大化するような独自の文章を生成できるようになる．

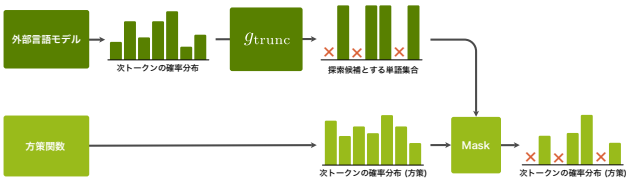


図 1　Truncation の例

3　提案手法

本研究では学習済み言語モデルを強化学習により fine-tuning することで，入力した文を併置型の駄洒落に変換する駄洒落生成モデルを学習する．より具体的には，**(入力文) [SEP]** の形式で言語モデルに入力を与えたとき，これに続けて「入力文を駄洒落に言い換えた表現」が生成されるように fine-tuning する．学習済みモデルには rinna 社が公開している GPT-2 モデルである `rinna/japanese-gpt2-medium` [9] を用いる．

Fine-tuning は強化学習により行う．文章生成時の単語選択を強化学習における行動選択とみなす．ただし次単語の候補は数万にも及ぶため，この広い行動空間で探索を行うことは困難である．そこで，Martin らによる TrufLL [8] を参考に truncation を行った確率分布から単語選択を行う．また，駄洒落が生成されやすくなるように工夫した truncation 関数を用いて，探索を効率的に進めることを狙う．

TrufLL と同様に学習には PPO (Proximal Policy Optimization) を用いる．報酬には，

（1）　入力文と生成文の意味の類似度
（2）　独自にトレーニングした駄洒落判定モデルの出力値
（3）　文章の長さに基づくスコア

の 3 つの和を用いる．これにより入力文と意味の近い，駄洒落らしい文章が生成されることを狙う．文章の長さに基づくスコアは，入力文に比べて極端に短い文や最大長以上の長さの文が生成されることを防ぐために使用する．

学習の流れは図 2 に示す．学習では行動と評価を繰り返す．行動フェーズでいくつか文章を生成した後，メモリーの情報を

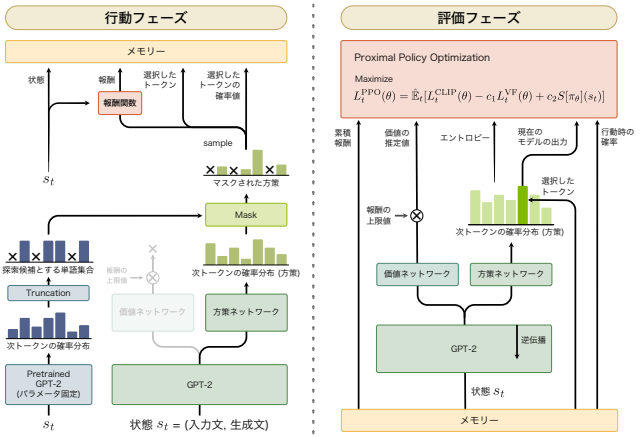


図 2　学習の概要図

用いて評価フェーズで 20 エポック学習する．

3.1　モデル

方策・状態価値関数として `rinna/japanese-gpt2-medium` を用いる．このモデルの言語ヘッドを方策関数とし，新たに用意した 2 層の全結合層を価値関数とする．価値関数の出力には sigmoid 関数を適用し，この値に報酬の最大値を乗算する．報酬は文章生成後に初めて与えられ，負の値にならないことが保証される．図 2 においては「GPT-2」「価値ネットワーク」「方策ネットワーク」の部分にあたる．

強化学習による言語生成のフレームワークとして TrufLL [8] を用いた．TrufLL は外部の言語モデルにより確率分布の truncation を行い，エージェントが効率的に探索を行えるようにする．

3.2　Truncation 用言語モデル

Martin らによる TrufLL では外部言語モデルの出力を利用し，エージェントの行動空間を切り詰める．外部言語モデル f_{LM} の語彙を $\mathcal{V}$ とする．このとき，行動空間を切り詰めるための関数である truncation 関数 g_{trunc} に f_{LM} の出力を与える．この truncation 関数から得られる語彙 $\mathcal{V}^- \subset \mathcal{V}$ ($|\mathcal{V}^-| \ll |\mathcal{V}|$) をエージェントの行動空間とする．本研究ではこの truncation を fine-tuning に用いる．

さらに，f_{LM} の出力を一部改良する．まず，ベースとなるモデルはエージェントと同様に rinna 社の GPT-2 モデルである `rinna/japanese-gpt2-medium` を用いる．このときパラメータはエージェントと共有しないものとする．GPT-2 モデルに対して次の形式のプロンプトを与える．

「[入力文]」を駄洒落らしく言い換えてください: [ここまでの生成文]

このプロンプトを入力したときの GPT-2 の出力である次単語の logits を $f_{\mathrm{LM}_{\mathrm{logits}}}(\cdot|w_{<t})$ とする．ここで $w_{<t}$ はステップ $t-1$ までの生成文である．

次に $f_{\mathrm{LM}_{\mathrm{logits}}}(\cdot|w_{<t})$ のうち，値の大きい上位 k_{pun} 単語を $\mathcal{V}^-$ とする．このとき，$f_{\mathrm{LM}_{\mathrm{bonus}}}(w_t|w_{<t})$ を次のように定義する．

$$f_{\mathrm{LM}_{\mathrm{bonus}}}(w_t|w_{<t}) = g_{\mathrm{filter}}(f_{\mathrm{LM}_{\mathrm{logits}}}(w_t|w_{<t}), w_t, w_{<t}) \cdot (1 + g_{\mathrm{bonus}}(w_t, w_{<t}, \mathcal{V}^-)) \quad (4)$$

$$g_{\mathrm{filter}}(x, w_t, w_{<t}) = \begin{cases} -\infty & (w_t \in w_{<t} \cup \mathcal{V}_{\mathrm{ASCII}}) \\ x & (\text{otherwise}) \end{cases} \quad (5)$$

$$g_{\mathrm{bonus}}(w_t, w_{<t}, \mathcal{V}^-) = \begin{cases} c_{\mathrm{bonus}} f_{\mathrm{pun}}((w_{<t}, w_t)) & (w_t \in \mathcal{V}^-) \\ 0 & (\text{otherwise}) \end{cases} \quad (6)$$

ここで c_{bonus} は定数，$\mathcal{V}_{\mathrm{ASCII}}$ は ASCII 文字や記号を含むトークンの集合，f_{pun} : (日本文) $\to [0, 1]$ は駄洒落判定モデルである．駄洒落判定モデルには後述の CPDN (Convolutional Pun Decision Network) を用いる．

また g_{filter} で $\mathcal{V}_{\mathrm{ASCII}}$ に属するトークンの logit を $-\infty$ としている．これは日本語の単語以外の探索を禁止して，エージェントが効率的に日本語の表現を探索できるようにするものである．さらにエージェントが同じ単語を繰り返して駄洒落としてしまうことを防ぐため，既に使用されたトークンの logit を $-\infty$ としている．ただし実際の実装では ひらがな 1 文字からなるトークンは 2 回までの使用を許可している．ひらがな 1 文字からなるトークンは助詞として出現する可能性があり，2 度目の出現を禁止すると文章の生成が難しくなる可能性があるためである．

最後に，

$$f_{\mathrm{LM}'}(w_t|w_{<t}) = \mathrm{softmax}(f_{\mathrm{LM}_{\mathrm{bonus}}}(\cdot|w_{<t})) \quad (7)$$

で定義される $f_{\mathrm{LM}'}(w_t|w_{<t})$ を truncation 用言語モデルの出力とする．$f_{\mathrm{LM}'}$ を truncation 用言語モデルとすることで，より駄洒落らしい表現が truncation 関数から選ばれやすくなることを期待できる．

Truncation には top-k truncation を用いる．$f_{\mathrm{LM}'}$ の上位 k 個のみを選択し，エージェントはその中から次単語を選択する．

3.3　単語選択

探索時の単語選択には truncation 用言語モデルと truncation 関数により得られる語彙 $\mathcal{V}^- \subset \mathcal{V}$ ($|\mathcal{V}^-| \ll |\mathcal{V}|$) を用いる．方策関数の logits を $\pi_\theta^{\mathrm{logits}}$ としたとき，次のように定義される確率分布 $\pi_\theta^{\mathrm{exploration}}$ を探索時の方策とする．

$$\pi_\theta^{\mathrm{exploration}}(\cdot|w_{<t}) = \mathrm{softmax}(\pi_\theta^{\mathrm{truncated}}(\cdot|w_{<t})) \quad (8)$$

$$\pi_\theta^{\mathrm{truncated}}(w_t|w_{<t}) = \begin{cases} \pi_\theta^{\mathrm{logits}}(w_t|w_{<t}) & (w_t \in \mathcal{V}^-) \\ -\infty & (\text{otherwise}) \end{cases} \quad (9)$$

ここから得られる方策 $\pi_\theta^{\mathrm{exploration}}$ で単語をサンプリングする．

なお，方策関数・価値関数の更新時に行う評価フェーズでは，方策関数の出力する方策 π_θ をそのまま用いる．

3.4　エピソード

強化学習におけるエピソードとは，エージェントが環境とやり取りをする一連の流れのことである．言語生成における「エ

ピソード」は入力に対して，文章を生成する流れを指す．本研究では，探索時に "。" または "</s>" (EOS) が選択された場合，またはトークン数が L_{limit} に到達した場合は探索を終了し，そこまでを 1 エピソードとする．

3.5 Proximal Policy Optimization (PPO)

本研究では PPO (Proximal Policy Optimization) [7] を用いて学習を行った．PPO では以下の目的関数を最大化する．

$$L_t^{\text{CLIP+VF+S}}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t)\right] \tag{10}$$

ここで c_1, c_2 は係数，θ は価値・方策ネットワークのパラメータである．さらに，V_t^{targ} は次のように定義する．

$$V_t^{\text{targ}} = \sum_{t'=t}^{T} \gamma^{t'-t} \text{reward}(s_{t'}) \tag{11}$$

T は文字列長，γ は割引率，$\text{reward}(s_t)$ は状態 s_t における報酬である．言語生成に強化学習を使用する場合，割引率に $\gamma = 1$ を用いることが多い．本研究においても $\gamma = 1$ を使用する．

3.6 報酬

S_{input} を入力文のトークン列，S_{pun}^t を t ステップ目における生成文のトークン列とする．状態 $s_t = (S_{\text{input}}, S_{\text{pun}}^t)$ のときの報酬関数として以下のものを設計し，使用した．

$$r(s_t) = c_{\text{sim}} r_{\text{sim}}(s_t) + c_{\text{pun}} r_{\text{pun}}(s_t) + c_{\text{len}} r_{\text{len}}(s_t) \tag{12}$$

ここで $c_{\text{sim}}, c_{\text{pun}}, c_{\text{len}} \in \mathbb{R}$ はそれぞれの報酬の重み，r_{sim} は意味の類似度報酬関数，r_{pun} は駄洒落スコア関数，r_{len} は長さ報酬関数である．

意味の類似度関数　入力文と意味の近い文章が生成されるように，入力文と生成文の意味的な類似度を報酬の一部とする．意味の類似度は，文章の埋め込みを生成するモデル f_{sim} の出力のコサイン類似度により求める．

$$r_{\text{sim}}(s_t) = \max(0, f(s_t)) \in [0, 1] \tag{13}$$

$$f(s_t) = \cos(f_{\text{sim}}(S_{\text{input}}),\ f_{\text{sim}}(S_{\text{pun}})) \in [-1, 1] \tag{14}$$

ここで，cos はコサイン類似度である．埋め込みベクトルのコサイン類似度が 1 に近いほど，2 つの文章の意味は似ているということになる．

f_{sim} には Web 上で公開されているモデル `sentence-transformers/paraphrase-multilingual-mpnet-base-v2` [10] を用いた．

駄洒落スコア関数　駄洒落らしい文章が生成されるように，生成された文章の駄洒落スコアを報酬の一部とする．駄洒落スコアは，独自に訓練した駄洒落判定モデル f_{pun} の出力とする．

$$r_{\text{pun}}(s_t) = f_{\text{pun}}(S_{\text{pun}}) \in [0, 1] \tag{15}$$

より駄洒落らしければ 1 に近づき，そうでなければ 0 に近づく．詳細は 4 節を参照．

長さ報酬関数　極端に短い文章や長い文章が生成されることを防ぐため，文章の長さに基づくスコアを報酬の一部とする．長さ報酬関数は次のように定義する．

$$r_{\text{len}}(s_t) = \begin{cases} \min\left(1, \frac{|S_{\text{pun}}|}{c_{\text{minl}}|S_{\text{input}}|}\right) & (|S_{\text{pun}}| < L_{\text{limit}}) \\ 0 & (|S_{\text{pun}}| = L_{\text{limit}}) \end{cases} \tag{16}$$

ここで $|S|$ は文 S のトークン長である．また，L_{limit} は出力の最大トークン長，c_{minl} は定数である．この関数にはエージェントが次の条件を満たす駄洒落を生成できるようにトレーニングする意図がある．

- 出力のトークン長が L_{limit} 未満となる
- 出力のトークン長が入力の c_{minl} 倍以上となる

4 駄洒落判定ネットワーク

駄洒落の評価を行うため，新たに畳み込みニューラルネットワーク [4] を用いた駄洒落判定ネットワーク CPDN (Convolutional Pun Decision Network) を構築した．

併置型の駄洒落は似た音素からなる単語が並ぶという特徴がある．この特徴を利用して駄洒落の判定を行う．

大文字のアルファベットの集合と padding `<pad>` の和集合を A，アルファベットのみから構成される長さ N の文字列を $S = (c_0, c_1, c_2, \ldots, c_N)$，関数 f を A^N から $\{0,1\}^{N\times N}$ への写像とする．また，f によりできる $N \times N$ 行列の各要素 a_{ij} $(1 \leqq i, j \leqq N)$ を以下のように定義する．

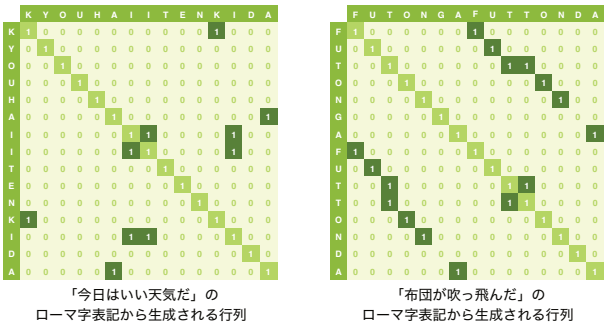
$$a_{ij} = \begin{cases} 1 & (c_i = c_j \neq \texttt{<pad>}) \\ 0 & (\text{otherwise}) \end{cases} \tag{17}$$

例えば，文字列 $S_{\text{pun}}, S_{\text{normal}}$ を以下のように定義する．

$$S_{\text{pun}} = (\texttt{F}, \texttt{U}, \texttt{T}, \texttt{O}, \texttt{N}, \texttt{G}, \texttt{A}, \texttt{F}, \texttt{U}, \texttt{T}, \texttt{T}, \texttt{O}, \texttt{N}, \texttt{D}, \texttt{A}) \tag{18}$$

$$S_{\text{normal}} = (\texttt{K}, \texttt{Y}, \texttt{O}, \texttt{U}, \texttt{H}, \texttt{A}, \texttt{I}, \texttt{I}, \texttt{T}, \texttt{E}, \texttt{N}, \texttt{K}, \texttt{I}, \texttt{D}, \texttt{A}) \tag{19}$$

S_{pun} は「布団が吹っ飛んだ」をローマ字表記にしたもの，S_{normal} は「今日はいい天気だ」をローマ字表記にしたものである．すなわち，S_{pun} は駄洒落の例であり，S_{normal} は駄洒落ではない例である．このとき，$f(S_{\text{pun}}), f(S_{\text{normal}})$ はそれぞれ図 3 のようになる．



「今日はいい天気だ」の
ローマ字表記から生成される行列

「布団が吹っ飛んだ」の
ローマ字表記から生成される行列

図 3　関数 f による行列を図示したもの．1 である要素を塗りつぶした．見やすさのために，必ず 1 の並ぶ対角成分は色を薄くした．

駄洒落である $f(S_{\text{pun}})$ には左上から右下に線を引くように 1 が並んでいることが分かる．一方で $f(S_{\text{normal}})$ にはそのような特徴は見られない．このことから，$f(S)$ を画像とみなして畳み込みニューラルネットワークを使うことで，駄洒落と駄洒落でないものの分類が可能であると考えられる．

一方で，駄洒落ではないにもかかわらず斜めに 1 が並んでしまう例もある．例えば `FUTONFUTONFUTONFUTON` (布団布団布団布団) のように意味とは関係なく同じ単語を繰り返した場合がその一例である．もし単純に左上から右下への斜線に反応するようにネットワークが学習すれば，このような例に対しても駄洒落と判定してしまうことになる．これを避けるため，負例としてランダムに単語を並べた文字列も学習に使用し，過剰な繰り返しへの反応を防止する．

4.1 モデルの構成

駄洒落判定モデルには日本語をローマ字表記にした文字列を入力する．ローマ字への変換には日本語の文字種変換ライブラリである jaconv [11] を使用した．また，漢字の読みの取得には漢字かな変換ライブラリである pykakasi [12] を使用した．

ローマ字表記の文字列を $S \in A^N$ とする．ただし長さ N に満たない文字列は `<pad>` トークンで padding するものとする．次にこれらの文字ごとの学習可能な埋め込み表現 $E \in \mathbb{R}^{N \times D}$ を得る．

次に，全結合層のネットワーク $N_1, N_2, \ldots, N_{\text{head}}$ へ E を入力し，新たな表現 $E_1, E_2, \ldots, E_n$ を得る．ここで N_{head} はヘッドの数である．Vaswani らによる Transformer [6] の Multi-head Attention に倣い，複数の表現に変換することにより精度を高めることを狙った．さらに $E_1, \ldots, E_n$ から以下に示す新たなテンソル T を得る．

$$T = \left[E_1^T E_1, E_2^T E_2, \ldots, E_n^T E_n \right] \in \mathbb{R}^{n \times N \times N} \qquad (20)$$

自身の転置行列との積をとることにより，各文字埋め込みの内積を取っている．

ここで，式 17 のような単純な文字比較ではなく，埋め込み表現から得られるベクトルの内積を用いるのは文字ごとの特徴も考慮するためである．例えば「は」「わ」のように異なる文字でも同じ音を持つ場合や，「か」「が」のように清音と濁音の関係にあるものなどは近い位置に存在するべきだと考えられる．文字ごとの埋め込み表現にすることにより，この問題を解決しようとしている．また，ベースラインとして単純な文字ごとの比較によるモデル (Simple-CPDN) も構築した．

次に，T を入力として，畳み込みニューラルネットワークに入力し，最終的にベクトル V を得る．これをネットワーク N_L に入力し，最終的に sigmoid 関数に通すことにより「駄洒落である確率」を計算する．

図 4 は，入力として与える各音素から埋め込み表現を得て，それらの内積を取り，CNN へ入力する一連の流れを図をして表したものである．モデルの詳細な構成図は図 5 に示した．

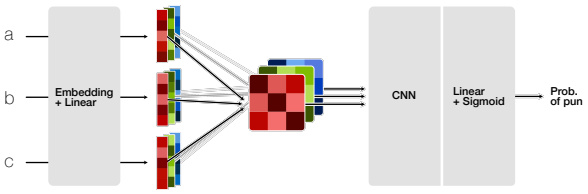


図 4　文字埋め込みを用いた CPDN のイメージ図

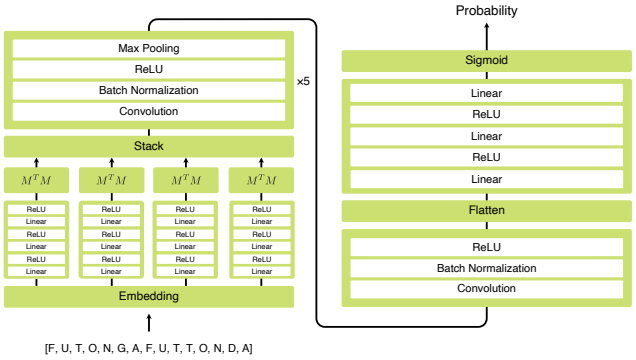


図 5　Convolutional Pun Desision Network の構成図

4.2 実験

データセット　正例のデータセットには駄洒落データベースに含まれる駄洒落のうち併置型のものを 64,219 文 使用した．負例のデータセットには名大会話コーパス [13], [14] から 119,855 文，CC100 (ja) [15], [16] から 50,000 文，“自動生成の単語繰り返し文” (後述) を 20,000 文使用した．データセットの各文には前処理として Unicode 正規化を行った．

負例の CC100 (ja) には非常に長い文章が含まれるため，句点「。」が存在する場合は一番最初の句点以降を削除した．

自動生成の単語繰り返し文は $N \in \{1, 2, \ldots, 15\}$ 個のひらがなからなる文字列を $M \in \{10, 11, 12, \ldots, 100\}$ 回繰り返したものである．最大文字列長は 200 文字として，それ以上の長さであった場合は 200 文字に切り詰めた．選ばれる文字列，N，M はそれぞれ一様乱数で決定した．さらにすべてのデータをシャッフルし，train (90%) / valid (5%) / test (5%) の 3 つにデータを分割した．テスト用には test (5%) のうち正例 3189 件，負例 3189 件を抽出し，6378 件のデータセットとした．また train の正例と負例の数を同じにした train (balanced) というデータセットも別途用意し，それぞれ学習を行った．

ハイパーパラメータ　ハイパーパラメータは表 3 に示す．

学習時はバッチサイズ 256，損失関数に Binary Cross Entropy，Optimizer には AdamW を使用した．学習率は最大値 1×10^{-2}，最小値 1×10^{-6} とし，cosine annealing によって学習率をスケジューリングした．エポック数は 100 とし，validation loss の最も小さかったエポックでのモデルを最終的なモデルとした．モデルの構造は表 3 に示した．

判定方法　入力に対し，モデルの出力が 0.5 以上ならば正例，そうでなければ負例と判定した．また，使用するモデルについては 100 エポック目ではなく，valid データセットにおける

表 3　駄洒落判定モデルのモデル構造

Multihead	N_{head}	8
	Hidden size (Emnbedding)	64
	Hidden size (Linear 1)	48
	Hidden size (Linear 2)	32
	Hidden size (Linear 3)	16
CNN	Conv 1 (out, kernel, stride)	$2N_{\text{head}}$, 8, 1
	Conv 2	$4N_{\text{head}}$, 8, 1
	Conv 3, 4	$8N_{\text{head}}$, 4, 1
	Conv 5, 6	$8N_{\text{head}}$, 2, 1
	Padding	`same`
	Max Pooling の stride	2
Linear	Hidden size (Linear 1)	1024
	Hidden size (Linear 2)	128

loss が最小となったエポック数でのモデルを採用した．

ベースライン　ベースライン手法には，アルファベットの埋め込みを使わず，単純な文字比較による行列 (式 17) を提案手法の畳み込みネットワークに直接入力する手法 Simple-CPDN を用いる．ハイパーパラメータは表 3 の N_{head} を 1 にしたものを使用した． また，参考値として畠山らの手法である PRS [3]，荒木らによる SVM を用いた手法 [5] による分類結果の値を畠山らの論文から引用した．

4.3　結果

分類結果は表 4 のようになった．

表 4　駄洒落判定モデルの分類結果

手法	Precision	Recall	Specificity	F1
SVM (参考値)	0.816	<u>0.956</u>	-	0.880
PRS (参考値)	0.904	0.824	-	0.862
Simple-CPDN (train)	<u>0.973</u>	0.867	<u>0.976</u>	0.917
Simple-CPDN (balanced)	0.959	0.891	0.962	<u>0.924</u>
CPDN (train)	**0.985**	0.914	**0.986**	**0.948**
CPDN (balanced)	0.921	0.823	0.929	0.869

Recall は SVM に劣るものの，その他の値では最もよい結果となっていることから CPDN (train) を駄洒落生成の報酬として用いる． また，CPDN (train) を除いたモデルの中で最も F1 が良いことから Simple-CPDN (balanced) を生成された駄洒落の評価に用いる．CPDN (train) を評価に用いないのは，モデルが報酬関数として用いる CPDN (train) に過適合しまっていた際に，不正に高く評価されることを避けるためである．

5　実験

第 4 節では駄洒落の報酬関数と，生成したモデルの評価に用いる駄洒落判定のためのモデル CPDN の実験を行った．ここでは CPDN を報酬関数の一部として利用し，第 3 節で述べた駄洒落生成モデルの実験を行う．

5.1　データセット

訓練データ　訓練時に入力文として与える文章のデータセットとして CC100 (ja) を用いる．このうち 60 文字以下であるものに対し，Unicode 正規化の処理をしたものを使用した．さらに句点が含まれるものについては，一番最初の句点までを用いた．

駄洒落言い換えデータセット　駄洒落データベースの併置型駄洒落から駄洒落でない文章に言い換えた「駄洒落言い換えデータセット」を構築した．Web 上で公開されている日英・英日翻訳モデルである `staka/fugumt-ja-en` [17]，`staka/fugumt-en-ja` [18] を用いて，駄洒落を英語に翻訳，さらに英語を日本語に翻訳することによって「駄洒落ではない言い換え文」を生成した．ただしこの方法では必ずしも言い換え文が得られるとは限らないため，報酬関数 $r_{\text{pun}}, r_{\text{sim}}$ を利用し品質の良いものを選択した．具体的には，駄洒落データベースに含まれる併置型駄洒落 S とその言い換え T について，

$$\begin{cases} r_{\text{sim}}(S,T) \geqq 0.8 & \text{(言い換え前と言い換え後の意味が近い)} \\ r_{\text{pun}}(S) \geqq 0.8 & \text{(言い換え前の駄洒落らしさが大きい)} \\ r_{\text{pun}}(T) \leqq 0.2 & \text{(言い換え後の駄洒落らしさが小さい)} \\ |T| \geqq 15 & \text{(言い換え後が十分に長い文)} \end{cases} \tag{21}$$

のすべて満たすようなペア (S,T) を駄洒落言い換えデータセットとした．これにより 2837 対の言い換えペアが得られた．このうち 2337 文を train， 250 文を test，250 文を valid と呼ぶことにする．

テストデータ　テストには訓練データとは異なる CC100 (ja) を 250 文，駄洒落言い換えデータセット (test) の言い換え文 250 文を用いた．CC100 の処理は訓練データと同様である．また，駄洒落言い換えデータセットにも Unicode 正規化の処理を行った．

5.2　ハイパーパラメータ

学習時は最大エピソード数 10000，GPT-2 の学習率を 3×10^{-5}，方策ネットワークの学習率を 1×10^{-5}，価値ネットワークの学習率を 1×10^{-3} とした．また，Optimizer に RAdam を用いた．価値ネットワークは Hidden size が 1024，活性化関数が tanh の 2 層の全結合層とした．

PPO におけるハイパーパラメータは $c_1 = 0.5, c_2 = 0.01, \epsilon = 0.2$ とした．また， 1 epoch ごとにメモリー全体をバッチとして更新を行い，各回 20 epochs 学習を行った．実験ではメモリサイズの制限から仮のバッチサイズを 40 とし，gradient accumulation により擬似的にメモリー全体を 1 つのミニバッチと捉えて更新を行った．更新間隔は 8 エピソード毎とした．

Truncation は Top-k truncation ($k = 40$) を用いた．また，$c_{\text{bonus}} = 2.0$ とした．

5.3　評価

評価は自動評価でのみ行う．評価には学習済みモデルを利用したものが含まれるが，報酬関数等で使用するモデルとは別のモデルを使用する．これはエージェントが報酬関数のモデルに過適合しまっていた際に，不正に高く評価されることを防ぐためである．

さらに 500 エピソードごとに validation データセットで推論を行い，報酬が最大となるようなモデルを保存し評価に用いた．ここで validation データセットとは

- CC100 のうち訓練時・テスト時とは異なる 250 文
- 駄洒落言い換えデータセット (valid) の言い換え文 250 文

の計 500 文である．この報酬の平均が最大となるようなモデルを評価に用いた．

駄洒落らしさ　駄洒落らしさの評価には Simple-CPDN (balanced) の出力値を使用する．

文の類似度　文の類似度の評価には報酬関数とは異なる `sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2` [10] を使用する．計算方法は報酬に使用した「意味の類似度関数」と同様である．

BLEU　駄洒落言い換えデータセットの言い換え語を入力としたときの出力について，実際の駄洒落との BLEU スコアを計算した．このスコアは駄洒落らしさの一部として評価する．

Distinct-N　Distinct-N は生成された文全体で n-gram が重複しているかを示す指標である．生成された文全体のユニークな n-gram 数を n-gram の合計で割ることにより計算される．今回は文字ごとの n-gram で $n = 1, 2$ について計算した．

Perplexity　生成文の Perplexity の平均値を計算した．言語モデルには **入力文 [SEP]** という形式のプロンプトを与えるが，この部分については計算対象から除外している．

単語繰り返しスコア　本研究で目指す駄洒落は併置型駄洒落であった．併置型駄洒落は意味の異なる似た音素の区間が存在する文章であるが，モデルは同じ単語を繰り返した文を生成することがある．そこで，生成文全体で同じ単語がどの程度繰り返されているかを示す指標を設計し，これを評価に用いる．

生成文を分かち書きした単語列を S とする．例えば「今日は良い天気ですね」を生成文とするとき，S は $S =$ (今日, は, 良い, 天気, です, ね) となる．S からひらがな 1 文字の単語を除外した単語列を S' とする．

単語繰り返しスコア $\mathrm{REP}(S)$ を以下のように計算する．

$$\mathrm{REP}(S) = |S'| - |\mathrm{set}(S')| \tag{22}$$

ここで，$\mathrm{set}(S')$ は S' の重複を除いた集合である．また，$|\cdot|$ は単語列と集合の要素数を表す．

このスコアは同じ単語が繰り返し出現した回数を表す．駄洒落生成においてこの値は少ないほど良いと言える．生成文全体での単語の繰り返しスコアの平均値をモデルの評価に用いる．

5.4 サンプリング手法

文章生成の際に行うサンプリングのパラメータは top-k = 50，top-p = 0.95 とし，各入力文に対して 8 文生成した．ベースライン手法についても同様にサンプリングを行った．

5.5 ベースライン

ベースラインには GPT-2 の few-shot 学習，畠山らによる Transformer での駄洒落生成を用いた．乱数のシード値はいずれも 42 に固定した．

GPT-NeoX の few-shot 学習　ABEJA 社の公開している GPT-NeoX モデル `abeja/gpt-neox-japanese-2.7b` を用いて few-shot 学習を行う．このモデルに対して次のようなプロンプトを入力し，その続きを駄洒落とする．

文章を駄洒落に言い換えてください。
布団が吹き飛びました。 → 布団が吹っ飛んだ。
池に行きなさい → 池に行け
［入力文］→

GPT-2 の教師あり fine-tuning

学習済み GPT-2 モデルである `rinna/japanese-gpt2-medium` を，駄洒落言い換えデータセット (train) で 7 エポック fine-tuning した．Fine-tuning は 学習率 1×10^{-5}，バッチサイズ 24，optimizer に AdamW を用いて行った．損失関数には cross-entropy loss を用いた．

5.6 比較実験

提案手法において一部のハイパーパラメータを変更したモデルについても実験を行った．

Truncation LM に単語の繰り返しを許可

提案手法では単語の繰り返しを避けるために，g_{filter} で一度使用した単語を再度使用しないようにしていた．比較対象として，truncation 用言語モデルに単語の繰り返しを完全に禁止させるのではなく，一度使用した単語の logit を 0.5 倍する方法も用いる．より詳細には truncation 用言語モデルにおける g_{filter} と g_{bonus} を次のように変更する．

$$g_{\mathrm{filter}}(x, w_t, w_{<t}) = \begin{cases} -\infty & (w_t \in \mathcal{V}_{\mathrm{ASCII}}) \\ x & (\mathrm{otherwise}) \end{cases} \tag{23}$$

$$g_{\mathrm{bonus}}(w_t, w_{<t}, \mathcal{V}^-) = \begin{cases} -0.5 & (\mathrm{if}\ w_t \in w_{<t}) \\ c_{\mathrm{bonus}} f_{\mathrm{pun}}((w_{<t}, w_t)) & (\mathrm{else\ if}\ w_t \in \mathcal{V}^-) \\ 0 & (\mathrm{otherwise}) \end{cases} \tag{24}$$

これを Repeat: Penalty と呼び，同じトークンの使用を禁止する方法を Repeat: Banned と呼ぶ．

Fine-tuning されたモデルの更なる fine-tuning

ベースライン手法として用いた駄洒落言い換えデータセットで fine-tuning した GPT-2 モデルを，更に提案手法で fine-tuning する．少ない教師データで fine-tuning されたモデルが，強化学習による fine-tuning でより良い方向に改善されるかどうかを確かめる．

報酬の変更

報酬の係数 $c_{\mathrm{sim}}, c_{\mathrm{pun}}, c_{\mathrm{len}}$ はハイパーパラメータであった．この値を変更すると学習にどのような影響が出るのかについて調査する．

5.7 結果

テストデータとして CC100 を用いたときの評価結果を表 5 に，駄洒落言い換えデータセットを用いたときの評価結果を表

6 に示す．各入力文に対して 8 文ずつ生成し，それぞれの指標の平均値をモデルの評価指標とする．

「PPL」は Perplexity，「REP」は単語繰り返しスコア，「駄洒落」は駄洒落判定モデルの出力，「類似度」は入力と出力の類似度を表す．また，「GPT-2」は `rinna/japanese-gpt2-medium`，「FT」はこの GPT-2 モデルを駄洒落データセットで fine-tuning したモデルをベースとして学習したことを示す．さらに，参考値として駄洒落言い換えデータセットの単語繰り返しスコア，駄洒落らしさ，意味の類似度，BLEU，Distinct-N も掲載した．生成例は表 7 に示した．

表 5　CC100 に含まれる 250 文 を入力としたときの結果．

base	報酬			Repeat	PPL (↓)	REP (↓)	駄洒落	類似度	Distinct-N	
	c_{pun}	c_{sim}	c_{len}						$N=1$	$N=2$
GPT-2	10	35	5	Banned	**26.8**	0.255	0.315	0.331	0.006	0.027
GPT-2	15	30	5	Banned	206.9	0.180	0.123	0.280	0.027	0.106
GPT-2	15	25	10	Banned	128.8	0.260	0.317	0.581	0.015	0.053
GPT-2	25	20	5	Banned	1.16×10^{7}	0.234	**0.800**	0.274	0.026	0.110
GPT-2	35	10	5	Banned	1278.7	3.108	0.352	0.291	0.009	0.058
FT	35	10	5	Banned	4.88×10^{6}	**0.177**	0.423	0.277	0.011	0.036
GPT-2	25	20	5	Penalty	490.8	1.740	0.531	0.607	0.021	0.190
FT	25	20	5	Penalty	1085.7	2.922	0.518	**0.627**	0.022	0.205
ベースライン: GPT-NeoX Few-shot					56745.8	1.003	0.331	0.378	0.025	0.249
ベースライン: 駄洒落 Fine-tuning					24475.1	0.674	0.483	0.626	**0.031**	**0.254**

表 6　駄洒落言い換えデータセット 250 文 を入力としたときの結果．

base	報酬			Repeat	PPL (↓)	REP (↓)	駄洒落	類似度	BLEU	Distinct-N	
	c_{pun}	c_{sim}	c_{len}							$N=1$	$N=2$
GPT-2	10	35	5	Banned	**28.1**	0.077	0.326	0.331	0.187	0.0008	0.013
GPT-2	15	30	5	Banned	250.8	0.005	0.450	0.251	0.172	0.0214	0.129
GPT-2	15	25	10	Banned	106.8	0.097	0.645	0.333	0.191	0.0032	0.009
GPT-2	25	20	5	Banned	1.27×10^{7}	0.022	**0.914**	0.246	**0.206**	0.0463	0.155
GPT-2	35	10	5	Banned	419.7	2.646	0.401	0.275	0.193	0.0068	0.042
FT	35	10	5	Banned	6.02×10^{6}	**0.000**	0.462	0.233	0.201	0.0029	0.005
GPT-2	25	20	5	Penalty	958.5	1.084	0.634	0.683	0.196	0.0271	0.237
FT	25	20	5	Penalty	1345.6	1.886	0.660	0.693	0.197	0.0328	0.258
ベースライン: GPT-NeoX Few-shot					71216.5	0.785	0.443	0.442	0.192	0.0302	0.246
ベースライン: 駄洒落 Fine-tuning					48870.2	0.205	0.634	**0.708**	0.199	**0.0521**	**0.362**
参考値: 駄洒落言い換えデータセット					-	0.184	0.752	0.839	1.000	0.2086	0.692

6　考察

6.1　同じ単語の繰り返し

結果より，Repeat: Banned のモデルよりも Penalty の方が駄洒落のスコアが高くなる傾向にある．しかし生成例を見ると Penalty のモデルでは同じ単語を繰り返していることが多い．「孔子私には私についてこれと私を私する。」がその一例である．また，単語繰り返しスコア (REP) を見ても Penalty の方が Banned よりも大きくなる傾向にあることが分かる．駄洒落判定モデルは音素の繰り返しのみを考慮するため，このような出力に対しても高いスコアを与えてしまう．すなわち，Repeat: Penalty のモデルは不正に駄洒落のスコアを高くしてしまっていると考えられる．

6.2　報酬の係数

報酬の係数をいくつか変えて実験を行った．

まず，駄洒落報酬の係数 c_{pun} と文類似度報酬の係数 c_{sim} の大小は，結果の駄洒落スコアと類似度スコアの大小に直結する傾向があると分かった．$c_{\text{pun}} < c_{\text{sim}}$ のときは，類似度が高くなり，$c_{\text{pun}} > c_{\text{sim}}$ のときは駄洒落スコアが高くなる傾向にある．しかし $c_{\text{pun}}, c_{\text{sim}}$ の比率が大きいほど駄洒落と類似度のスコアに差が出ているといった関係は存在していない．2 つの係数が近い値であっても駄洒落スコアと類似度スコアに大きな差が生まれることもあれば，逆に 2 つの係数が大きく異なる値であっても駄洒落スコアと類似度スコアの差が少なくなる例も存在する．このことから，この 2 つのバランスをとるような報酬設計をすることは難しいと考えられる．報酬設計は今後の課題としたい．

表 7　本手法による駄洒落の生成例

報酬			Repeat	入力文 / 生成文
c_{pun}	c_{sim}	c_{len}		
10	35	5	Banned	**入力文:** 私が乗船したのはその時でした **生成文:** はのととはは やとのごと は。
15	30	5	Banned	**入力文:** コーンが出ましたので、どうぞ、これです **生成文:** しかし。
				入力文: クレーターの写真送ってもいい **生成文:** 大好きな。
30	15	5	Banned	**入力文:** 関東地方で煮干ししたウニとウニ **生成文:** 今回は、ご紹介。
				入力文: あなたはモノレールに乗ることができます **生成文:** 今回は、ご紹介。
35	10	5	Banned	**入力文:** 相手をメールでフォローしよう **生成文:** そうそうあなたにそうメールをおくそう。
				入力文: お茶が飲めないなんて残念です **生成文:** ご自分のお茶を飲みたいのください。
				入力文: これは飲酒にも関わってくる。 **生成文:** 私たちは飲酒のし過ぎを注意される。
25	20	5	Penalty	**入力文:** 孔子は私にこれを行うように言った **生成文:** 孔子私には私についてこれと私を私する。
				入力文: おでん 食べたいなら 明かりの下で。 **生成文:** 明るければ 明るいところで。
				入力文: スノーモービルです。ビールを飲みましょう **生成文:** スノーですの。
				入力文: 出会いの１つとしてはアリだけど、 結婚までとなるとなかなか難しいですね。 **生成文:** アリの出会いにはアリアリです。

6.3　駄洒落のデータセットなしでの学習

駄洒落のような特殊な文章を集めたデータセットは多く存在しない．また，このようなデータセットを作ることには多大な労力がかかる．そのため，より高品質な駄洒落を作るためには，データセットを用いずに学習する方法を発見することが重要である．

結果より，GPT-2 モデル `rinna/japanese-gpt2-medium` を提案手法 (Repeat: Penalty) で fine-tuning したモデルは，駄洒落言い換えデータセットで fine-tuning したモデルよりも駄洒落のスコアで優れていた．Repeat: Penalty による学習は同じ単語の繰り返しが多いという欠点はあるものの，この事実は駄洒落らしさを学習する上で強化学習が有効であることを示唆している．今後，より良いアルゴリズムを利用して学習を行うことで，駄洒落を生成できることが期待できる．

6.4　探索の難しさ

本研究における駄洒落の生成は，駄洒落らしさを獲得することが必要である．また，一度使用したトークンは使用してはな

らないという制約から，より幅広いトークンを探索する必要がある．さらに駄洒落生成では，一度の探索で得られた知識も他の場面では活用しにくいという欠点がある．例えば「布団が吹き飛んだ」という入力に対して「布団が吹っ飛んだ」という表現を見つけることができたとしても，この知識は「犬がいました」などの別の入力文に適用できない．「犬がいました」を「犬が居ぬ」に変換する場合は，また新たに「居ぬ」という表現を発見する必要がある．このような点が，本研究における駄洒落の生成における探索の難しさにつながっていると考えられる．

今後はより効率的に探索を行うためのアルゴリズムの調査が必要である．

7 おわりに

本研究では学習済み GPT-2 の強化学習による fine-tuning で，入力文の意味を変えず併置型駄洒落に変換するモデルの構築を検討した．文章の類似度や駄洒落らしさ等を報酬に用いることにより，駄洒落らしい文章の生成を試みたが，まだ駄洒落らしいと言えるような文章を安定して生成することはできなかった．しかし駄洒落らしさのスコアを伸ばすことができ，僅かながらも駄洒落らしい出力を得られたことから，強化学習による駄洒落生成モデルの構築の有用性が示唆された．また，学習時に同じトークンを選ばせない工夫をすることで，同じ単語を繰り返すことによる不正な駄洒落の生成を防ぐことができた．

さらに，駄洒落の評価のために駄洒落判定モデル Convolutional Pun Desision Network を提案した．これは文章内での音の繰り返しを畳み込みニューラルネットワークで判定するモデルである．これにより，モデルが駄洒落らしい音の反復表現を学習でき，先行研究での手法を Precision と F1 スコアで上回ることができた．

今後はより効率的に，様々なトークンを探索可能なアルゴリズムについて調査し，より駄洒落らしい文章への変換を目指すことを今後の課題としたい．また，報酬関数のために作成した駄洒落判定モデル CPDN は日本語の特徴を利用したモデルであり，現状では他言語への対応が難しいという課題がある．駄洒落らしい音素の繰り返しを判定できる，言語によらないアルゴリズムの開発も今後の課題としたい．

謝辞

本研究は JSPS 科研費 JP21H03496, JP22K12157 の助成を受けたものです．

本研究は，電気通信大学人工知能先端研究センター (AIX) の計算機を利用して実施したものです．

本研究で使用した『駄洒落データベース』は科学研究費基盤研究 C（課題番号: 17K00294）において開発されたものです．駄洒落データベースを提供してくださった北海道大学大学院情報科学研究院メディアネットワーク部門情報メディア学分野言語メディア学研究室教授 荒木健治先生に感謝いたします．

文　献

[1] 荒木健治. 駄洒落データベースを用いた駄洒落生成システムの性能評価. ことば工学研究会資料, SIG-LSE-B703-8, pp. 1–15, 2018.

[2] 荒木健治, 内田ゆず, 佐山公一, 谷津元樹. 駄洒落データベースの構築及び分析. ことば工学研究会資料, SIG-LSE-B803-1, pp. 1–15. 人工知能学会 第 2 種研究会, 2018.

[3] 畠山和久, 徳永健伸. Transformer を用いた日本語併置型駄洒落の自動生成. 言語処理学会 第 27 回年次大会, 2021.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324, 1998.

[5] 谷津元樹, 荒木健治. 子音の音韻類似性及び SVM を用いた駄洒落検出手法. 知能と情報, Vol. 28, No. 5, pp. 875–886, 2016.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[7] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[8] Alice Martin, Guillaume Quispe, Charles Ollion, Sylvain Le Corf, Florian Strub, and Olivier Pietquin. Natural language generation (almost) from scratch with truncated reinforcement learning. In *Proc. of AAAI 2022*, 2022.

[9] rinna Co., Ltd. rinna/japanese-gpt2-medium · Hugging Face. `https://huggingface.co/rinna/japanese-gpt2-medium`. (Accessed on 12/26/2022).

[10] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[11] Yukino Ikegami. ikegami-yukino/jaconv: Pure-python japanese character interconverter for hiragana, katakana, hankaku, and zenkaku. `https://github.com/ikegami-yukino/jaconv`. (Accessed on 12/26/2022).

[12] Pykakasi documentation — pykakasi 2.0.1 documentation. `https://pykakasi.readthedocs.io/en/latest/index.html`. (Accessed on 01/24/2023).

[13] 藤村逸子, 大曽美恵子, 大島ディヴィッド義和. 会話コーパスの構築によるコミュニケーション研究. 2011.

[14] 藤村逸子, 滝沢直宏. 言語研究の技法：データの収集と分析. ひつじ書房, 2011.

[15] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8440–8451, Online, July 2020. Association for Computational Linguistics.

[16] Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. CCNet: Extracting high quality monolingual datasets from web crawl data. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pp. 4003–4012, Marseille, France, May 2020. European Language Resources Association.

[17] Satoshi Takahashi. staka/fugumt-ja-en · Hugging Face. `https://huggingface.co/staka/fugumt-ja-en`. (Accessed on 09/20/2022).

[18] Satoshi Takahashi. staka/fugumt-en-ja · Hugging Face. `https://huggingface.co/staka/fugumt-en-ja`. (Accessed on 09/20/2022).