**In Jupiter-Notebook einfügen**

{

"cells": [

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"## KI und Weihnachtstexte\n",

"### Wie man Texte fÃ¼r Weihnachtslieder und Texte fÃ¼r Weihnachtsgedichte generiert\n",

"\n",

"Bald ist Weihnachten! Heute wollen wir Texte fÃ¼r Weihnachtslieder bzw. Weihnachtsgedichte generieren. \n",

"\n",

"Wir werden uns ein paar Vereinfachungen erlauben:\n",

"1. Wir generieren die Texte in Englisch, schreiben alles klein und lassen alle Satzzeichen weg. Dadurch werden die Parameter reduziert und das Problem etwas einfacher.\n",

"2. Wir lassen auch die ZeilenumbrÃ¼che weg, die nach den einzelnen Zeilen einer Strophe und zwischen den Strophen folgen. Diese verwirren einfache KI Modelle und wir brÃ¤uchten neben einem etwas komplexeren Modell auch mehr Trainingsdaten. \n",

"\n",

"Bevor wir Anfangen, setzen wir noch schnell alle Technik auf: Wir verwenden python3 und benÃ¶tigen fÃ¼r das Model einige Pakete um die KI Libraries zu benutzen. \n",

"\n",

"\n",

"#### BenÃ¶tigte Python Module"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"!pip install seaborn"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"!pip install nltk"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"!pip install tensorflow"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"Zuletzt testen wir noch schnell, ob TensorFlow eine Grafikkarte findet. Ohne Grafikkarte dauert das Trainieren sehr viel lÃ¤nger, ist aber trotzdem mÃ¶glich."

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"import tensorflow as tf\n",

"tf.test.gpu\_device\_name()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"#### Und nun importieren wir alle Module, die wir brauchen"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"\n",

"import pandas as pd\n",

"import numpy as np\n",

"import matplotlib.pyplot as plt\n",

"import seaborn as sns\n",

"from nltk.corpus import RegexpTokenizer\n",

"import keras\n",

"from keras.preprocessing.text import Tokenizer\n",

"from keras.layers import TextVectorization\n",

"\n",

"from keras.utils import pad\_sequences\n",

"from keras import layers\n",

"from keras.models import Sequential\n",

"from keras.utils import to\_categorical\n",

"from keras.callbacks import EarlyStopping\n",

"from keras.optimizers import RMSprop\n",

"from keras.callbacks import ModelCheckpoint\n",

"import re"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"import matplotlib.pyplot as plt"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"Die Technik ist fertig. Weiter geht es mit den Daten!\n",

"\n",

"#### Weihnachtsgedicht\n",

"\n",

"Damit eine KI lernen kann, braucht sie Trainingsdaten. Wir brauchen also ein paar Weihnachtsgedichte.\n",

"\n",

"Ich habe da mal 'was vorbereitet..."

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"data = pd.read\_json('christmascarols.json')\n",

"print(data.shape)\n",

"data.head()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"Das sind 40 Weihnachtslieder. Es sind also nicht so viele Beispiele.\n",

"\n",

"Wir wollen nur den Text generieren und verzichten somit auf Titel und Autor. AuÃŸerdem brauchen wir die Gedichte als einen langen String"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"poems = data['gedicht'].drop\_duplicates()\n",

"print(f\"Total Unique Texts: {poems.shape}\")\n",

"\n",

"\n",

"poems\_list = list(poems)\n",

"print(poems\_list[:2])\n",

"\n",

"all\_poems = \"\\n\\n\".join(poems\_list)\n",

"\n",

"#clean\n",

"all\_poems = re.sub(\"[,\\.!'\\-;]\",\"\",all\_poems)\n",

"\n",

"all\_poems[:500]"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"## Datenvorbereitung\n",

"\n",

"Die KI kann die Daten nicht so verwenden, wie sie sind. Das hat mehrere GrÃ¼nde:\n",

"1. Neuronale Netze erwarten, dass ihre Eingaben immer dieselbe Dimension bezÃ¼glich der Grundeinheiten\n",

"2. es ist unklar, was die Grundeinheiten sind: einzelnen Buchstaben, Abfolgen von Buchstaben, ganze WÃ¶rter oder ganze Zeilen?\n",

"3. Neuronale Netze mÃ¶chten gerne mit ihren Eingaben rechnen kÃ¶nnen, d.â€¯h. wir brauchen Zahlen\n",

"\n",

"FÃ¼r diesen Versuch werden wir WÃ¶rter als Grundeinheiten betrachten. Satzzeichen und Sonderzeichen werden wir ignorieren und zudem werden alle Buchstaben durch ihr kleingeschriebenes Ã„quivalent ersetzt: A zu a, E zu e, e zu e, usw. Der Text wird anhand von Whitespace-Zeichen in WÃ¶rter getrennt: Ein Wort sind also alle Nicht-Whitespace-Zeichen zwischen zwei aufeinanderfolgenden Whitespace-Zeichen. \n",

"\n",

"Ein Beispiel:\n",

"\n",

"Aus ` It's tall in the saddle we spend Christmas Day,` wird ` its tall in the saddle we spend christmas day` und daraus wird die folgende Liste an WÃ¶rtern ` [its,tall,in,the,saddle,we,spend,christmas,day] `\n",

"\n",

"\*\*Problem 2\*\* ist also gelÃ¶st: Unsere Grundbausteine sind WÃ¶rter. \n",

"\n",

"\*\*Problem 1\*\* lÃ¶sen wir, indem wir das Problem des Textgenerierens umschreiben: Anstatt Text zu generieren, wollen wir vorhersagen, welches Wort als NÃ¤chstes nach einer Wortsequenz folgt. Die LÃ¤nge der Wortsequenz ist fest (aber beliebig). Wir werden Wortsequenzen der LÃ¤nge 5 verwenden. Unsere Dimension ist nun also immer gleich.\n",

"\n",

"An unserem Beispiel:\n",

"Wir wollen erraten, dass nach ` [its,tall,in,the,saddle] ` das Wort ` we ` folgt und dass nach ` [tall,in,the,saddle,we] ` das Wort ` spend ` folgt usw. \n",

"\n",

"Text wird generiert, indem wir dieses Vorhersagen immer wieder machen und dabei die neu generierten WÃ¶rter als Eingabe mitverwenden. \n",

"\n",

"\*\*Problem 3\*\* ist etwas kniffliger. Aus den WÃ¶rtern sollen jedoch Zahlen werden. Wenn wir einfach nur durchnummerieren, selbst wenn wir dies auf eine sehr intelligente Art und Weise machen, implizieren wir eine Ã„hnlichkeit zwischen den WÃ¶rtern, die nicht erwÃ¼nscht ist. Implizit ist das Wort mit Zahlenwert 1 dem Wort mit Zahlenwert 2 Ã¤hnlicher als dem Wort mit Zahlenwert 5, weil der Abstand zwischen 1 und 2 (Abstand 1) kleiner ist als der Abstand zwischen 1 und 5 (Abstand 4). Daher verwenden wir Embeddings, im Speziellen One-Hot-Word-Embeddings oder auch One-Hot-Encodings. Dabei stellen wir jedes Wort durch einen Vektor dar, der so viele Stellen hat, wie es verschiedene WÃ¶rter gibt. Jede Stelle steht fÃ¼r ein Wort. Ein Vektor ist Ã¼berall 0 auÃŸer an der Stelle, die fÃ¼r das entsprechende Wort steht: dort ist er eins.\n",

"\n",

"Am Beispiel:\n",

"Wir haben 9 verschiedene WÃ¶rter: ` [its,tall,in,the,saddle,we,spend,christmas,day] `\n",

"Somit haben unsere Wortvektoren LÃ¤nge 9. \n",

"- ` its [1,0,0,0,0,0,0,0,0] `\n",

"- ` tall [0,1,0,0,0,0,0,0,0] `\n",

"- ` in [0,0,1,0,0,0,0,0,0] `\n",

"- ` the [0,0,0,1,0,0,0,0,0] `\n",

"- ` saddle [0,0,0,0,1,0,0,0,0] `\n",

"- ` we [0,0,0,0,0,1,0,0,0] `\n",

"- ` spend [0,0,0,0,0,0,1,0,0] `\n",

"- ` christmas [0,0,0,0,0,0,0,1,0] `\n",

"- ` day [0,0,0,0,0,0,0,0,1] `\n",

"\n",

"\n",

"Wenn die \*Eingabe fÃ¼r das Netz\* ` [its,tall,in,the,saddle] `, dann mÃ¼ssen wir dies zunÃ¤chst in ` [[1,0,0,0,0,0,0,0,0], [0,1,0,0,0,0,0,0,0], [0,0,1,0,0,0,0,0,0], [0,0,0,1,0,0,0,0,0], [0,0,0,0,1,0,0,0,0]] ` transformieren und erhalten dann hoffentlich als \*Ausgabe\* ` [0,0,0,0,0,1,0,0,0] `, was wiederum ` we ` entspricht.\n",

"\n",

"\n",

"\*Bemerkung\*: Im One-Hot-Encoding sind alle WÃ¶rter gleich Ã¤hnlich zueinander und, fÃ¼r die Mathelehramtsstudierenden unter euch, sind alle orthogonal zueinander und haben die EinheitslÃ¤nge. Somit sind sie einen Orthonormalbasis fÃ¼r den konstruierten Vektorraum."

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"## Tokeinization\n",

"\n",

"\n",

"seq\_len = 5 # LÃ¤nge der Wortsequenz\n",

"tokenizer = TextVectorization() # Normalisierung und Zerlegung in Worte\n",

"tokenizer.adapt(poems\_list) # erstellt Vokabular und weiÃŸt jeden Wort eine Zahl zu\n",

"\n",

"# Model um Normalisierung und Zerlegung zu erstellen\n",

"imodel = Sequential()\n",

"imodel.add(tf.keras.Input(shape=(1,), dtype=tf.string))\n",

"imodel.add(tokenizer)\n",

"\n",

"# jeder Text wird nun in WÃ¶rter zerlegt und die WÃ¶rter als ihre ZahlenreprÃ¤sentation wiedergegeben\n",

"poems\_integer = [imodel.predict(np.array([poem]))[0] for poem in poems\_list]\n",

"\n",

"# Vorberechnung wieviele Trainingsbeispiele (immer 5 aufeinanderfolgenden WÃ¶rter \n",

"# und das nÃ¤chste) und wieviele WÃ¶rter es gibt\n",

"samples = 0\n",

"for poem in poems\_integer:\n",

" samples += len(poem) - seq\_len - 1\n",

"total\_vocab = tokenizer.vocabulary\_size() \n",

"\n",

"# One-hot-Encodings der Trainingsdaten (X\_t ist der Input, y\_t ist der output)\n",

"X\_t = np.zeros((samples, seq\_len, total\_vocab), dtype=bool)\n",

"y\_t = np.zeros((samples, total\_vocab), dtype=bool)\n",

"\n",

"# Daten durchlaufen und Einsen setzen\n",

"pos = 0\n",

"for poem in poems\_integer:\n",

" for i in range(0,len(poem)-seq\_len-1):\n",

" for j in range(0,seq\_len):\n",

" X\_t[pos,j,poem[i+j]] = 1\n",

" y\_t[pos,poem[i+seq\_len]] = 1\n",

" pos += 1\n",

"\n",

"\n",

"\n",

"print(f\"Form von X\_t: {X\_t.shape}\")\n",

"print(f\"Form von Y\_t: {y\_t.shape}\")\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"Jetzt sind die Daten so, dass unsere KI sie verstehen kann. ðŸ¥³\n",

"\n",

"#### Das Model der KI\n",

"\n",

"Wir verwenden ein sehr hÃ¤ufig verwendetes Netzwerk. Das LSTM-Netzwerk (engl.: long short-term memory, dt.: langes KurzzeitgedÃ¤chtnis). Informationen dazu findet ihr zum Beispiel zusammengefasst auf Wikipedia (dt: [ https://de.wikipedia.org/wiki/Long\_short-term\_memory ], engl. [ https://en.wikipedia.org/wiki/Long\_short-term\_memory ]). Unser Netzwerk soll aus 2 Layern bestehen:\n",

"1. LSTM Layer: 128 Neuronen und den Input Dimensionen 5x1357\n",

"2. Dense Layer: jedes Neuron ist mit jedem Neuron des vorherigen Layers verbunden.\n",

"\n",

"Zum Optimieren wollen wir ein Root Mean Square Propagation benutzen (siehe die engl. Wikipedia Seite [https://en.wikipedia.org/wiki/Stochastic\_gradient\_descent#RMSProp]) und Categorial Cross-Entropy Loss (siehe z.B.: [https://peltarion.com/knowledge-center/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy]).\n",

"\n",

"\n",

"Genau genommen sagen betrachtet die KI ihre Aufgabe nicht darin, WÃ¶rter zu generieren, sondern Eingaben zu klassifizieren. Also klassifiziert die KI unsere 5 WÃ¶rter langen Eingaben in so viele Klassen, wie es in dem vorgegebenen Vokabular gibt. Die KI ist also ein Multi-Klassen-Klassifikator. LSTM-Modelle kÃ¶nnen das jedoch als Stream -- mit anderen Worten: fortlaufend. "

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"Jetzt rÃ¤umen wir mal kurz auf."

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"import gc\n",

"gc.collect()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"Wir haben die Daten im richtigen Format. Unser Model ist definiert. Nun kÃ¶nnen wir endlich die KI trainieren. \n",

"\n",

"### Training\n",

"\n",

"ZunÃ¤chst erzeugen wir das Model. Das geht dank der vorgefertigten Layer von Keras relativ einfach. Wir benutzen ein sequenzielles Modell und fÃ¼gen 2 Layer (LSTM-LAyer gefolgt vom Dense Layer) in der gewÃ¼nschten GrÃ¶ÃŸe hinzu. Hinzu kommt noch der Optimierungsalgorithmus und die Loss Funktion und fertig ist unser Model."

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"# Wir benutzen das Keras framework fuer unser Model\n",

"def create\_model():\n",

" model = Sequential() # wir moechten ein sequenzielles Modell (die Layer werden hintereinander geschaltet)\n",

" model.add(layers.LSTM(128, input\_shape = (X\_t.shape[1], X\_t.shape[2]))) # mit einem LSTM Layer \n",

" model.add(layers.Dense(total\_vocab, activation='softmax')) # und einem Dense Layer mit softmax activation\n",

" \n",

" # wir wollen den Root Mean Square Propagation opmitmierer mit einer laeerning rate von 0.001\n",

" optimizer=RMSprop(learning\_rate=0.001)\n",

" # das Model soll categorial cross entropy loss verwenden\n",

" model.compile(loss='categorical\_crossentropy', optimizer=optimizer)\n",

" return model\n",

"\n",

"model = create\_model()\n",

"model.summary()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"Das Training heiÃŸt in der Welt von keras und tensorflow fit, weil wir die Modellparameter an unsere Daten fitten. Wir rufen daher die Funktion ` fit ` von unserem Model mit unseren Trainingsdaten (X\_t - die Eingabe und y\_t die erwartete Ausgabe) auf. \n",

"\n",

"ZusÃ¤tzlich brauchen wir noch den Callback, der festlegt, wann das Training beendet wird und welche Metrik aufgezeichnet wird. Wir benutzen heute mal einen einfachen Model Checkpoint. Dieser zeichnet den Loss und die Modellparameter nach jeder Epoche auf. Eine Epoche ist ein kompletter Durchlauf durch die Trainingsdaten. Das heiÃŸt, nach jeder Epoche werden die Modellparameter in eine Datei exportiert und der Loss sowohl ausgegeben als auch in einem Objekt gespeichert. \n",

"\n",

"\*\*Achtung:\*\* Dieser Schritt dauert etwas!"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"\n",

"\n",

"# Training -- wir definierne uns einen Monitor, der das Training ueberwacht und die Losswerte einsammelt\n",

"callback = ModelCheckpoint(filepath='./christmasai\_model/model\_{epoch}.hdf5',monitor='loss')\n",

"history=model.fit(X\_t, y\_t, epochs=1000, batch\_size=128, callbacks=[callback])\n",

"\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"Wir wollen uns nun den Trainingsverlauf anschauen. In dem RÃ¼ckgabeobjekt der Funktion fit (wir haben dies ` history ` genannt) ist Folgendes abgespeichert."

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"print(history.params)\n",

"print(history.history.keys())\n",

"print(len(history.history['loss']))"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"Die Liste ` loss ` enthÃ¤lt also die Losswerte nach den verschiedenen Epochen. Diese kÃ¶nnen wir grafisch darstellen, um zu sehen wie das Training verlaufen ist. "

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"loss = []"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"loss += history.history['loss']\n",

"plt.plot(loss)\n",

"plt.show()\n",

"plt.plot(history.history['loss'])\n",

"plt.show()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### Text generieren\n",

"\n",

"Das Model ist trainiert und wir kÃ¶nnen es nun nutzen, um neuen Text zu generieren. Wie eingangs bereits erwÃ¤hnt ist, benutzen wir 5 aufeinander folgenden Worte, um das nÃ¤chste Wort vorherzusagen. Daher beginnen wir auch mit 5 WÃ¶rtern, welche wir am Anfang festlegen. Wir nehmen \"ill be home for christmas\". \n",

"\n",

"Dann kÃ¶nnen wir das 6. Wort vorhersagen und unsere Vorhersage basierend auf \"be home for christmas\" und dem vorhergesagten Wort basieren lassen. Das wiederholen wir, solange wie wir wollen, z.B. 100-mal oder 200-mal. \n",

"\n",

"Als zweites Beispiel benutzen wir \"frosty the snowman was a\". \*\*Achtung\*\*: beide Lieder sind Teil des Trainingskorpus. \n",

"\n",

"Wir definieren uns noch eine Funktion ` sample ` auf die spÃ¤ter eingegangen wird. \n"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"def sample(preds, temperature = 1):\n",

" preds = np.asarray(preds).astype(\"float64\")\n",

" scaled\_pred = np.log(preds)/temperature\n",

" scaled\_pred = np.exp(scaled\_pred)\n",

" scaled\_pred = scaled\_pred/np.sum(scaled\_pred)\n",

" scaled\_pred = np.random.multinomial(1, scaled\_pred, 1)\n",

" return np.argmax(scaled\_pred)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"# generate word-based data\n",

"sentence = \"ill be home for christmas\"\n",

"sentence\_as\_integer = list(imodel.predict(np.array([[sentence]]))[0])\n",

"sentence\_as\_integer = sentence\_as\_integer[-seq\_len:]\n",

"\n",

"generated = \"\" \n",

"print(sentence)\n",

"original = sentence\n",

"for i in range(100):\n",

" x\_pred = np.zeros((1, seq\_len, total\_vocab))\n",

" for t, code in enumerate(sentence\_as\_integer):\n",

" x\_pred[0, t, code] = 1.0\n",

" preds = model.predict(x\_pred, verbose=0)[0]\n",

" next\_index = sample(preds)\n",

" next\_word = tokenizer.get\_vocabulary()[next\_index]\n",

" sentence\_as\_integer = sentence\_as\_integer[1:] + [int(next\_index)]\n",

" generated += \" \" + next\_word\n",

" \n",

"\n",

"print(\"------\\n\"+generated)\n",

"generated = \"\" \n",

"for i in range(100):\n",

" x\_pred = np.zeros((1, seq\_len, total\_vocab))\n",

" for t, code in enumerate(sentence\_as\_integer):\n",

" x\_pred[0, t, code] = 1.0\n",

" preds = model.predict(x\_pred, verbose=0)[0]\n",

" next\_index = sample(preds)\n",

" next\_word = tokenizer.get\_vocabulary()[next\_index]\n",

" sentence\_as\_integer = sentence\_as\_integer[1:] + [int(next\_index)]\n",

" generated += \" \" + next\_word\n",

" \n",

"print(\"------\\n\"+generated)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"# generate word-based data\n",

"sentence = \"frosty the snowman was a\"\n",

"sentence\_as\_integer = list(imodel.predict(np.array([[sentence]]))[0])\n",

"sentence\_as\_integer = sentence\_as\_integer[-seq\_len:]\n",

"\n",

"generated = \"\" \n",

"print(sentence)\n",

"original = sentence\n",

"for i in range(100):\n",

" x\_pred = np.zeros((1, seq\_len, total\_vocab))\n",

" for t, code in enumerate(sentence\_as\_integer):\n",

" x\_pred[0, t, code] = 1.0\n",

" preds = model.predict(x\_pred, verbose=0)[0]\n",

" next\_index = sample(preds)\n",

" next\_word = tokenizer.get\_vocabulary()[next\_index]\n",

" sentence\_as\_integer = sentence\_as\_integer[1:] + [int(next\_index)]\n",

" generated += \" \" + next\_word\n",

" \n",

"\n",

"print(\"------\\n\"+generated)\n",

"generated = \"\" \n",

"for i in range(100):\n",

" x\_pred = np.zeros((1, seq\_len, total\_vocab))\n",

" for t, code in enumerate(sentence\_as\_integer):\n",

" x\_pred[0, t, code] = 1.0\n",

" preds = model.predict(x\_pred, verbose=0)[0]\n",

" next\_index = sample(preds)\n",

" next\_word = tokenizer.get\_vocabulary()[next\_index]\n",

" sentence\_as\_integer = sentence\_as\_integer[1:] + [int(next\_index)]\n",

" generated += \" \" + next\_word\n",

" \n",

"print(\"------\\n\"+generated)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"#### Text generieren mit etwas mehr KreativitÃ¤t\n",

"\n",

"Hierbei kommt die Funktion ` sample ` ins Spiel. Die Funktion ` sample ` nimmt die Vorhersagewahrscheinlichkeit skaliert sie mit der Temperatur und berechnet dann die Wahrscheinlichkeiten neu basierend auf einer multi-normal Verteilung. Eine Temperatur von 1 skaliert die Wahrscheinlichkeiten nicht und erzeugt somit dieselben Vorhersagen wie das Netz. Temperaturen grÃ¶ÃŸer 1 erlauben mehr KreativitÃ¤t, aber auch mehr Fehler in den Vorhersagen."

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"# generate word-based data\n",

"sentence = \"ill be home for christmas\"\n",

"sentence\_as\_integer = list(imodel.predict(np.array([[sentence]]))[0])\n",

"sentence\_as\_integer = sentence\_as\_integer[-seq\_len:]\n",

"\n",

"generated = \"\" \n",

"print(sentence)\n",

"original = sentence\n",

"for i in range(100):\n",

" x\_pred = np.zeros((1, seq\_len, total\_vocab))\n",

" for t, code in enumerate(sentence\_as\_integer):\n",

" x\_pred[0, t, code] = 1.0\n",

" preds = model.predict(x\_pred, verbose=0)[0]\n",

" next\_index = sample(preds,1.5)\n",

" next\_word = tokenizer.get\_vocabulary()[next\_index]\n",

" sentence\_as\_integer = sentence\_as\_integer[1:] + [int(next\_index)]\n",

" generated += \" \" + next\_word\n",

" \n",

"\n",

"print(\"------\\n\"+generated)\n",

"generated = \"\" \n",

"for i in range(100):\n",

" x\_pred = np.zeros((1, seq\_len, total\_vocab))\n",

" for t, code in enumerate(sentence\_as\_integer):\n",

" x\_pred[0, t, code] = 1.0\n",

" preds = model.predict(x\_pred, verbose=0)[0]\n",

" next\_index = sample(preds,2)\n",

" next\_word = tokenizer.get\_vocabulary()[next\_index]\n",

" sentence\_as\_integer = sentence\_as\_integer[1:] + [int(next\_index)]\n",

" generated += \" \" + next\_word\n",

" \n",

"print(\"------\\n\"+generated)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"# generate word-based data\n",

"sentence = \"frosty the snowman was a\"\n",

"sentence\_as\_integer = list(imodel.predict(np.array([[sentence]]))[0])\n",

"sentence\_as\_integer = sentence\_as\_integer[-seq\_len:]\n",

"\n",

"generated = \"\" \n",

"print(sentence)\n",

"original = sentence\n",

"for i in range(100):\n",

" x\_pred = np.zeros((1, seq\_len, total\_vocab))\n",

" for t, code in enumerate(sentence\_as\_integer):\n",

" x\_pred[0, t, code] = 1.0\n",

" preds = model.predict(x\_pred, verbose=0)[0]\n",

" next\_index = sample(preds,1.5)\n",

" next\_word = tokenizer.get\_vocabulary()[next\_index]\n",

" sentence\_as\_integer = sentence\_as\_integer[1:] + [int(next\_index)]\n",

" generated += \" \" + next\_word\n",

" \n",

"\n",

"print(\"------\\n\"+generated)\n",

"generated = \"\" \n",

"for i in range(100):\n",

" x\_pred = np.zeros((1, seq\_len, total\_vocab))\n",

" for t, code in enumerate(sentence\_as\_integer):\n",

" x\_pred[0, t, code] = 1.0\n",

" preds = model.predict(x\_pred, verbose=0)[0]\n",

" next\_index = sample(preds,2)\n",

" next\_word = tokenizer.get\_vocabulary()[next\_index]\n",

" sentence\_as\_integer = sentence\_as\_integer[1:] + [int(next\_index)]\n",

" generated += \" \" + next\_word\n",

" \n",

"print(\"------\\n\"+generated)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"#### Quellen\n",

"\n",

"Die Idee Weihnachtsliedtext zu generieren und welches Netzwerk funktionieren kÃ¶nnte, gibt es [hier](https://towardsdatascience.com/all-i-want-for-christmas-is-ai-write-the-next-christmas-hit-using-lstms-2e95ea3da25e)\n",

"\n",

"\n",

"Ein Guide fÃ¼r Textgenerierung mit char-level LSTMs in python findet man in diesem [kaggle-notebook](https://www.kaggle.com/code/hariharanhd/char-level-text-genereation) "

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

}

],

"metadata": {

"kernelspec": {

"display\_name": "venv",

"language": "python",

"name": "venv"

},

"language\_info": {

"codemirror\_mode": {

"name": "ipython",

"version": 3

},

"file\_extension": ".py",

"mimetype": "text/x-python",

"name": "python",

"nbconvert\_exporter": "python",

"pygments\_lexer": "ipython3",

"version": "3.8.10"

}

},

"nbformat": 4,

"nbformat\_minor": 4

}