1 Kleine Zusammenfassung (25.10.19)

von Johannes Schmidt

1.1 Thema

Problem

Beim Training vor QM7 mit der sortierten Coulomb Matrix (CM) kommt das folgende Problem auf:
Aufgrund von festgelegter Länge D der CM ($D = 23$), gibt es Null Einträge, d.h. Zeilen in denen alle Einträge Null sind. Da die CM zeilenweise in das neuronale Netz (NN) gegeben wird, werden also auch diese Null Einträge übergeben. Nun gibt es aber keinen Grund warum das NN die dazu gehörige Energie gleich Null setzt. Physikalisch ist das natürlich klar: keine Atome, keine Ionisationsenergie. Im Allgemeinen ist es ratsam physikalisches Wissen in die Architektur des NN oder in den Datensatz mit einfließen zu lassen. Im besten Fall muss das NN diese Information nicht erst lernen. Im schlechtesten Fall fehlt eine entscheidende Information um eine angestrebte Genauigkeit zu erreichen (Hier ist vermutlich eher ersteres der Fall).

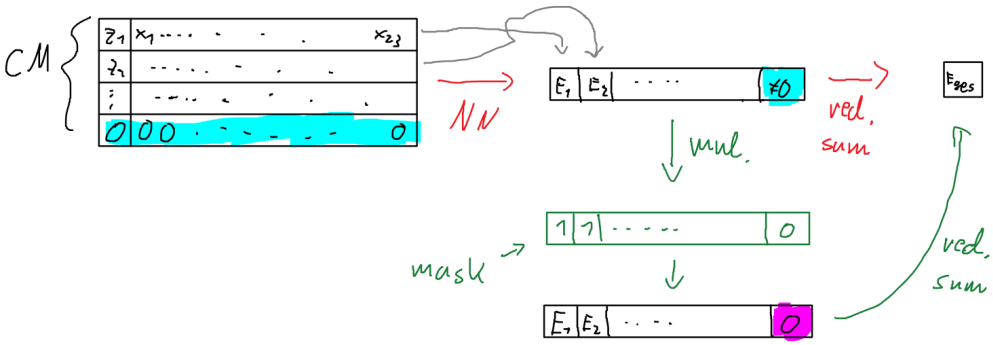

Figure 1: Grafik zur Maskierung der unphysikalischen Energien. Hier ist die batch size gleich 1.

1. Lösung: Bias raus

Bis jetzt wurde ein fully connectet multilayer perceptron sonnet NN verwendet. Dabei waren weights biases und die activision function (Hier tanh) die ganze Zeit aktiv. Also haben wir für den input x und den output $f(x)$ eines Nodes im Graphen den Zusammenhang,

$$f(x) = \tanh(W \cdot x + b)$$

und somit im Allgemeinen auch

$$f(0) = \tanh(b) \neq 0.$$

Ein Möglichkeit $f(0) \stackrel{!}{=} 0$ zu erfüllen ist es $b = 0$ zu wählen (weil $\tanh(0) = 0$), also alle biases aus dem NN raus zu nehmen. Somit wäre sichergestellt, dass die Energie eines 'leeren Moleküls' Null ist.

2. Lösung: Maskierung

Diese Methode ist in Figure 1 verdeutlicht. Wie oben beschrieben sollen die Zeilen der CM in Energien umgewandelt werden. Bevor man alle Einzelenergien aufsummiert multipliziert man noch mit einer Maske. Diese schmeißt die unphysikalischen Energie einfach raus, d.h. sie werden auf den Wert Null gesetzt. Somit enthält die Summe die unphysikalischen Energien nicht mehr.

1.2 Code

1. Lösung: Bias raus

Das sonnet Netz wird mit folgendem Code initialisiert.

```
model=snt.nets.MLP(output_sizes=[64,32,1],activation=tf.tanh,
                                        initializers=initializers,
                                        regularizers=regularizers)
```

Die initializers und regularizers werden an einer anderen Stelle im Code definiert. Diese Funktion liefert die einfache Möglichkeit alle biases mittels eines optionalen Arguments raus zunehmen.

```
model=snt.nets.MLP(output_sizes=[64,32,1],activation=tf.tanh,
                                        initializers=initializers,
                                        regularizers=regularizers,
                                        use_bias=False)
```

2. Lösung: Maskierung

Die naive Herangehensweise wird folgendermaßen implementiert.

```
x_data   = tf.placeholder(dtype=tf.float32,shape=(None,D*D))
x_data_r = tf.reshape(x_data, [-1,D])

output_data_tmp = tf.reshape(model(x_data_r),[-1,D])

output_data = tf.reshape(tf.reduce_sum(output_data_tmp,1)
    ,[-1])
```

Dabei ist x_data der placeholder für die CM. x_data_r ist dann zeilenweise sortiert was dann so ins model gegeben wird. Anschließend wird über die so

entstandene Liste output_data_tmp zeilenweise summiert. Also genau wie die roten Pfeile in Figure 1 angeben, nur mit dem Zusatz des batched trainings. Der modifizierte Code, welcher durch die grünen Pfeile in Figure 1 gekennzeichnet ist, ist folgendermaßen realisiert.

```
1 x_data   = tf.placeholder(dtype=tf.float32,shape=(None,D*D))
2 x_data_r = tf.reshape(x_data, [-1,D])
3 x_mask = tf.placeholder(dtype=tf.float32,shape=(None,D))
4
5 output_data_tmp = tf.reshape(model(x_data_r),[-1,D])
6 output_data_tmp_2 =  tf.reshape(tf.math.multiply(
      output_data_tmp,x_mask),[-1,D])
7
8 output_data = tf.reshape(tf.reduce_sum(output_data_tmp_2,1)
      ,[-1])
```

Der Datentyp von x_mask muss hier float32 gewählt werden. Mehr dazu in Abschnitt 1.4

1.3 Messverfahren

Hier wurde mit der sortierten CM mit $\alpha = 4$ trainiert.

1.

Nullmolekül testen, ob naiv nicht doch ok ist. Es könnte ja sein, dass das NN schon nach wenigen Schritten für keine Atome eine verschwindende Energie vorhersagt.

2.

Vergleiche (1) naiv (2) maske und (3) bias nach gleich vielen trainings schritten im Bezug auf MAE, um zu testen ob die Performance stark unter den Methoden leidet. (Hier ist mit MAE immer der test MAE gemeint.)

3.

Nullmolekül mit trainieren und nochmal gucken wie das naive Verfahren (1) ohne zusätzliche Modifikation das Nullmolekül auswertet.

1.4 Probleme bei der Implementierung

Datentyp von x_mask

Die tf.math.multiply Funktion funktioniert nur, wenn die zu multiplizierenden Tensoren den gleichen Datentyp haben. Auch wenn python die Funktionalität bietet True*0.5 auszuwerten, die tf.math.multiply tut es nicht.

Kein batch training

Bei der Methode der Maskierung ist es nur möglich eine batch size zu wählen die entweder 1 ist oder dem ganze Datensatz entspricht. Nach etwas Fehlersuche konnte das Problem nicht gelösst werden. Somit wurde mit batch_size = 1 trainiert. Das Problem liegt wahrscheinlich daran, wie Tensorflow intern das batch training managed. Denn eigentlich stimmen die shapses der einzeinen Arrays exact überein, also dürfte es eigentlich keinen Grund geben für eine Nichtübereinstimmung. Dennoch tritt bei einer batch size von 2 der folgende Fehler auf.

```
1 InvalidArgumentError: Incompatible shapes:[2,23] vs.[1433,23]
2          [[{{node Mul}}]]
```

Es könnte an der folgenden Zeile liegen,

```
1 x_data_r = tf.reshape(x_data, [-1,D])
```

weil hier nicht ganz klar ist was intern mit der batch size von x_data passiert.

1.5 Ergebnis und Diskussion

Wenn man den Graph initialisiert und alle biase ausstellt bekommt man wie erwartet für die Energie 0 raus. Also wird das hart auf Null setzten des bias sich wie erwartet verhalten. Zumindest im Bezug zur Energie des 0. Moleküls.

1.

Bevor die batch size auf 1 gesetzt wurde, wurde der das Ergebnis der letzten Zusammenfassung (vom 08.07.19) reproduziert und das Nullmolekül ausgewerted (Dort wurde bei der hier gewählten Architektur, [64,32,1], nach 400_000 Iterationsschritten und $\alpha = 1$ ein MAE von etwa 5.2 erreicht). Dabei kam es zu folgendem Ergebnis.

Iterationsschritte	MAE	Energie des Nullmoleküls [kcal]
1_000	740	-178
100_000	8.2	-50
400_000	4.9	12.15

Man sieht also deutlich, dass das NN nicht lernt, dass keine Atome keine Energiebeiträte liefert.

Als Vergleich, sowohl die bias als auch Maskierungsmethode geben, per Konstruktion, ab der ersten Iteration keine Energie für das Nullmolekül.

2.

In diesem Abschnitt wird für die Vergleichbarkeit bei allen Methoden eine batch size von 1 gewählt. Die Ergebnisse sind in Tabelle 1 dargestellt.

Iterationsschritte	MAE (1) naiv	MAE (2) maske	MAE (3) bias
100_000	32.3	45.9	23.8
400_000	8.5	35.4	17.4
600_000	7.0	34.9	11.7

Table 1: Ergebnisse des Trainings. Ergebnisse von (1) naiv etwas schlechter als vorher, da die batch size nicht optimal ist.

Das Ergebnis der Maskierung überrascht am meisten, da es deutlich schlechter performt als die anderen beiden. Die Vermutung hierfür ist, dass das NN wieder Freiheitsgrade verliert. Es ist ja so, dass das NN immer noch Energien ungleich Null vorhersagt, diese aber nicht in das Ergebnis mit einfließen. Somit überlässt man dem NN Parameter die optimiert werden, aber manchmal (je nach Größe des Moleküls) keinen Einfluss auf das Ergebnis haben. Wenn man sich die Energien anschaut bevor sie herausgenommen werden sieht man, dass das NN allen Umgebung zwar die gleiche Energie vorhersagt, diese aber nicht Null ist.

Zu erwarten wäre gewesen, dass die Lösung mit ausgeschaltetem bias am schlechtesten performt, weil man zeigen kann, dass der bias im NN die selbe Aufgabe hat wie der offset in kernel Methoden (Zitat Jan :D) und somit Freiheitsgrade fehlen. Diese Methode ist aber besser als die Maskierung.

Das NN performt also am besten, wenn man die unphysikalischen Energien mit einfließen lässt. Man könnte als weitere Maßnahme dafür sorgen, dass solche Null Zeilen garnicht erst ins Netz übergeben werden. Dann muss man sich aber mehr Gedanken bezüglich der Architektur des NN machen, weil dann die Moleküle unterschiedlich groß sind.

3.

Hier wird die batch size wieder auf 25 gesetzt und das Nullmolekül bei naiver Implementierung im trainings Datensatz hinzugefügt. Dies führt zu folgendem Ergebnis.

Iterationsschritte	MAE	Energie des Nullmoleküls [kcal]
1_000	761	-781
100_000	9.9	1.3
400_000	4.9	-1.6

Betrachtet man die Energie der Nullmoleküls nach jeder Generation sieht man, dass sie extremen Schwankungen unterliegt. Das hängt wahrscheinlich davon ab, ob das Nullmolekül gerade trainiert wurde oder nicht. Man sieht gleich schlechte Performance wie vorher auch. Manchmal springt die Energie des Nullmoleküls auf kleinere Werte (der beste war 0.0098) aber eine Generation danach liegt der Wert direkt wieder im ein bis zweistelligen Bereich.

Nichts desto trotz erreicht man hier einen genauso guten MAE bei verbessertem Wert der Energie des Nullmoleküls.

1.6 Fazit

Es scheint also im Bezug auf Performance keinen Unterschied zu machen ob nicht existente Atome wie solche behandelt werden, auch wenn das zumindest aus physikalischer Sicht zu erwarten gewesen wäre.