Fragebogen der Fachschaft zu **mündlichen Prüfungen** im Informatikstudium

Dieser Fragebogen gibt den Studierenden, die nach Dir die Prüfung ablegen wollen, einen Einblick in Ablauf und Inhalt der Prüfung. Das erleichtert die Vorbereitung.
Bitte verwende zum Ausfüllen einen schwarzen Stift. Das erleichtert das Einscannen.

Barcode:

2 214100 128740

Dein Studiengang: .....

Prüfungsart:
- ☐ Wahlpflichtfach
- ☐ Vertiefungsfach
- ☐ Ergänzungsfach

Welches? .....

Prüfungsdatum: .....

Prüfer/-in: .....

Beisitzer/-in: .....

Prüfungsfächer und Vorbereitung:

Veranstaltung	Dozent/-in	Jahr	regelmäßig besucht?

Note: .....

Prüfungsdauer: ..... Minuten

War diese Note angemessen?

✎ Wie war der **Prüfungsstil des Prüfers / der Prüferin?**
(Prüfungsatmosphäre, (un)klare Fragestellungen, Frage nach Einzelheiten oder eher größeren Zusammenhängen, kamen häufiger Zwischenfragen oder ließ er/sie dich erzählen, wurde Dir weitergeholfen, wurde in Wissenslücken gebohrt?)

↪**Rückseite bitte nicht vergessen!**

P: Prüfer, I: Ich

P: Sie sind Master Informatik. Haben sie denn schon Vorwissen?

I: Ja v.a. bei neuronalen Netzen.

P: Dann erzählen sie mal was sie da gemacht haben

I: Ich habe hier sehr viel Vorwissen und wir haben c.a. 12 Minuten über meine Arbeit als Hiwi etc geredet. Wo er konkret nachgefragt hat:

- Wie funktioniert ein Objekt Detektor?
  - Ich habe als output eines Detektors eine Featuremap die sehr viele (~4000) Bounding Boxes prädiziert (jeweils 4 Werte x,y, breite höhe) und jeder Bounding Box wird eine Klasse zugewiesen
  - Fast alle von den ~4000 Bounding Boxes sind Hintergrund und nur sehr wenige, ich habe als Beispiel ~3 genannt, beeinhalten wirkliche Objekte. Also prädiziert das Netz fast nur die Hintergrund Klasse. Es ist dann für das Netz sehr einfach Hintergrund mit 100% Sicherheit zu prädizieren, weil es dann nur in 3 von 4000 Bounding boxen falsch ist
  - Focal Loss verhindert das, weil Prädiktionen bei denen sich das Netz sicher ist weniger in den Loss rein gehen.
- Wie trainiert man generell ein Neuronales Netz:
  - Man hat einen Input der durch das neuronale Netz geht und der Output des Neuronalen Netzes ist dann die Prädiktion des neuronalen Netzes (häufig eine Klasse oder eine kontinuierlicher Wert). Man hat ein Label/Target wo die Musterlösung für unseren Input drinsteht (also die richtige Klasse oder der richtige kontinuierliche Wert). Das Netz soll möglichst so funktionieren, dass der Output vom Netz und das Label identisch sind. Man benutzt eine Loss Funktion, welche die Ähnlichkeit zwischen Label und Prädiktion misst. Wenn der Loss sehr niedrig ist, dann prädiziert das Netz sehr ähnlich/gleich zu dem Label und funktionert gut. Der Wert vom Loss wird verwendet, um einen Gradientenabstieg mit Backpropagation auf alle Gewichte vom Neuronalen Netz zu machen.
  - Der Gradientenabstieg findet mit einer geeigneten Lernrate ein lokales Minimum der Loss Funktion (Loss wird immer kleiner bis im Minimum). Er findet nicht das globale Minimum (was übrigens meistens sowieso ein Overfit auf den Trainingsdaten ist). Ihm war noch wichtig, dass es sehr viele lokale Minima gibt und es uns relativ egal ist in welches wir kommen
  - Bei der Backpropagation kann es bei Netzen mit vielen Layern zu einem Vanishing Gradient Problem kommen. Die Skip Connections machen, dass der Gradient, der von der Loss Funktion ausgeht, über diese Skip Connections in sehr wenig Schritten auf jedes mögliche Gewicht kommen kann. Dann haben wir immer große Gradienten die nicht gevanished sind (also 0 sind). Wenn der Gradient 0 ist, dann updaten wir die Gewichte nicht und das Netz kann nicht lernen.

P: Dann genug von den CNNs kennen sie Neuronale Netze, die wir in der Naturwissenschaft benutzt haben

I: Ja, wir haben ein RNN, eigentlich Transformer mit Attention benutzt, um Smile Codes zu prädizieren. Wir haben ein RNN benutzt, weil Smile Codes eine variable Länge haben und ein Transformer/RNN für variablen Eingabe- Ausgabegrößen benutzt wird.

P: Erinnern sie sich was da genau prädiziert wurde?

I: Ja wir bekommen zwei Smile Codes von Molekülen und der Transforrmer prädiziert dann wie diese beiden Moleküle reagieren. Bei den Smile Codes ist die Attention wichtig, weil z.B. Ringschlüsse sehr weit weg im String voneinander entfernt sind.

P: Kennen sie sonst noch ML verfahren die was mit Chemie gemacht haben?

I: Reinforcement Learning mit einem Generativen und Prädizierer (Ich glaub nicht, dass Prädizierer ein geeignetes Wort ist, mir ist aber kein anderes eingefallen). Der Generierer ist vortrainiert, gültige SMILE Codes zu generieren. Der Prädizierer bekommt als Eingabe einen Smile Code und prädiziert anhand des gegebenen Moleküls seine Chemischen Eigenschaften. Mit dem Reinforcement Learning trainieren wir dann den Generierer (Achtung: der Prädiziere wird nicht neu trainiert, der bleibt gleich) damit er Moleküle generiert, von denen der Prädizierer dann bestimmte gewünschte Eigenschaften prädiziert.

P: Ok das waren die SMILE Codes kennen sie noch andere Möglichkeiten Moleküle zu enkodieren.

I: Ja mit Fingerprinting. Die Ursprüngliche Methode erzeugt iterativ für jedes Atom einen Hash, dessen Modulo-wert dann als Index in unserem finalen Ausgabevektor eingefügt wird. Jedes Atom fügt dann in jedem Schritt (in dem der Radius erhöht wird) eine „1" in einen bestimmten Index. Zwei unterschiedliche Moleküle, die aber identische subgruppen besitzen, haben dann im gleichen Index die 1 stehen, der für diese subgruppe steht. Die anderen Indexe sind dann unterschiedlich. Die modernere Art ist dann mit Message Passing Networks, wobei wir nicht mehr eine Hash Funktion benutzen sondern jeweils mit einer Gewichtsmatrix die summierten Werte skalieren, dann sigmoiden, dann nochmal mit einer anderen Gewichtsmatrix skalieren und dann Softmaxen. Der gesoftmaxte Wert wird dann auf den Ausgabevektor addiert. Er wollte noch explizit hören, dass das es ein iteratives verfahren ist.

P: Wie trainiert man denn dieses Zweite Verfahren:

I: Man benutzt das meistens als Encoder Netzwerk an das dann ein Prädiktionsnetzwerk gehängt wird, das die DFT prädiziert. Das ist dann ähnlich wie ein Autoencoder, dass der Fingerprint eigentlich unsupervised gelernt wird. Wir trainieren Ende zu Ende also:

Moleküleigenschaften als Input → Erstelle Fingerprint→Prädiziere DFT.

Und wir hoffen, dass das Neuronale Netz dann gute Fingerprints lernt.

(Ich glaube den Vergleich mit dem Autoencoder fand er nicht so passend, weil er gesagt hat, dass ja eigentlich jedes Layer in einem Neuronalen Netz außer dem letzen Layer ja unsupervised Werte erstellt die wir nicht vorgeben. Er meint wir lernen intern eine gute Repräsentation von den wichtigen Daten)

P: D.h. man braucht ein Target also die DFT Rechnung. Kann man das auch komplett unsupervised also völlig ohne Label machen?

I: Vielleicht mit k-means oder PCA, aber ob das gut funktioniert ist eine andere Frage.

P: Dann noch zu den Autoencodern, wie funktionieren die denn:

I: Wir haben ein Encoder und ein Decoder Netzwerk. Das Ziel ist den Input in eine kleinere Dimension zu komprimieren, aus dem der Decoder dann trotzdem den vollständigen Input rekonstruieren kann.

P: Wir hatten dann noch eine andere Version.

I: Ja den Variational Autoencoder. Unser Encoder im normalen Autoencoder prädiziert einen einzelnen Wert im latent space. Der Variational Autoencoder prädiziert eine Verteilung mit mü und sigma. Wir samplen dann aus dieser Verteilung (also zufälligen Wert aus dem Latent space) und der Decoder soll trotzdem den Input korrekt rekonstruieren. Also wir haben immer noch den Rekonstruktionsloss. Was jetzt aber trotzdem passieren könnte, dass der Encoder eine unglaublich kleine Varianz prädiziert und dann somit eigentlich wieder nur ein einzelner punkt aus dem latent space gesampelt werden kann (dann ist das eigentlich identisch zum normalen Autoencoder). D.h. wir benutzen die Kullback Leibler Distanz mit der normalen Gaußverteilung N(0, 1). D.h. der Encoder wird bestraft wenn er Verteilungen erstellt, die sehr unterschiedlich davon sind. (Beispiel von vorhin, varianz ist 0.0000001 aber er wird bestraft wenn er nicht varianz von 1 prädiziert)

P: Und was bringt uns das dann?

I: Wir können jetzt über den latent space interpolieren und Daten generieren die wir nicht im Testset hatten.

P: Wir hatten noch andere Generationsverfahren in der Vorlesung.

I: Ja die GANs. Wir haben ein Generator und ein Diskriminator (beides Neuronale Netze). Der Generator prädiziert Bilder (muss kein Bild sein, aber ich habe Bilder als Beispiel genommen). Der Diskriminator bekommt entweder das erzeugte Bild von dem Generator oder ein echtes Bild aus einem Datensatz. Der Diskriminator soll jetzt entscheiden, ob das Bild echt ist, oder von dem Generator stimmt. Die Prädiktion vom Diskriminator geht dann in eine Loss Funktion und beide Netzwerke werden dann danach trainiert. Der Generator versucht den Diskriminator zu „überlisten" damit der Diskriminator seine Bilder für echt hält, während der Diskriminator versucht die „fake" Bilder vom Generator zu erkennen. Wenn wir fertig trainiert haben, brauchen wir den Diskriminator nicht mehr und nehmen den Generator, der jetzt gelernt hat möglichst genau Bilder zu generieren die sehr ähnlich aussehen wie die Bilder im Datensatz (und auch von einem Menschen nicht mehr als fake Bilder zu erkennen.Tipp: https://thispersondoesnotexist.com zeigt Ausgabebilder von so einem Generator, wenn man die Seite neu lädt kommen neue Bilder).

P: Was ist denn jetzt der Input von dem Generator?

I: Zufällige Werte die selber keine Bedeutung haben. Meistens werden die aus einer Normalverteilung o.ä. gesampled. Wir wollen halt verhindern dass der Generator die Testbilder auswendig lernt, das soll durch den zufälligen Input verhindert werden.

P: Dann kommen wir noch zu Bayesscher Optimierung. Wie funktioniert denn das und wo benutzt man dass?

I: Bei der Hyperparametersuche von Neuronalen Netzen und bei der Molekülkomposition von Solarpanels. Für B. Optimierung brauchen wir ein Modell, dass zusätzlich zu seiner Prädiktion noch seine Unsicherheit prädiziert. Wir nehmen das dann und benutzen eine acquisition function die dann uns einen Punkt sagt, auf dem wir neu samplen wollen. Wir haben einen Gaußschen Prozess benutzt für das Modell und dem eine Glattheit zugrunde gelegt, weil wir nicht erwarten, dass wenn wir einen Hyperparameter nur ganz leicht ändern, wir komplett andere Ergebnisse bekommen. Die acquisition Function besteht aus einem exploration und exploitation Term. Wenn wir voll exploration

machen,dann wird immer da gesampled wo die Unsicherheit am größten ist, während ein reiner exploration würde da samplen wo unser mü am höchsten ist.

P: Muss man denn diese acquisition function benutzen?

I: Nein, in der Vorlesung hatten wir die Varianz + mü genommen und dann da gesampled wo der höchste Wert rausgekommen ist. Mann kann auch immer ein Mischding aus Exploration und Exploitation machen. Bei dem Reinforcement Learning startet man häufig mit einer großen exploration, aber je länger man da trainiert, desto mehr exploitation macht man.

P: Und wie sieht dass dann ganz konkret aus? (ich weiß die genaue Frage nicht mehr)

I: Man hat bereits Messpunkte, die acquisition Funktion gibt uns jetzt einen neuen Punkt, und wir ermitteln jetzt das Ergebnis an diesem neuen Punkt. Bei Hyperparametersuche von einem NN trainieren wir jetzt ein neues Neuronales Netz und berechnen die Accuracy, bei chemischem Experiment führen wir das jetzt mit den neu definierten Eigenschaften durch. (Ihm war hier wichtig, dass das ein iterativer Prozess ist).

P: Was ist denn jetzt ein Nachteil bei der B. Optimierung?

I: Das funktioniert sequenziell, d.h. bei NN trainieren wir ein NN, dann neu samplen, dann wieder nur ein NN trainieren. Wenn man viele GPUs hat kann man viele NNs gleichzeitig in parallel, mit zufälligen Hyperparametern trainieren. Das geht im Zweifel schneller und ergibt nur leicht schlechtere Ergebnisse. Wenn man nicht so stark parallelisieren will (chemische Experimente kosten Geld) dann erzeugt B. Optimierung in weniger Schritten gute Ergebnisse.