Eindopdracht Advanced Datamining

Studiejaar 2023-2024, 2e gelegenheid

Inleiding

Dit is de *eindopdracht* behorende bij het vak *Advanced Datamining* (BFVH4DMN2) voor het *studiejaar 2023-2024 (2e gelegenheid)*. Op BlackBoard tref je eveneens een module `data.py` aan die diverse functies bevat die helpen bij het genereren en het visualiseren van de gebruikte datasets, en een bijbehorend data-bestand `EMNIST-mini.zip`.

Gebruik de `model` module die je in werkcollegeopdrachten 1, 2, 3, 4, en 5 & 6 hebt gemaakt om de onderstaande opdrachten uit te voeren. Deze eindopdracht bestaat uit twee delen:

- in **Deel A** worden een aantal cellen code gedraaid die als het goed is onmiddellijk zouden moeten werken met je model;
- in **Deel B** wordt je gevraagd om je gemaakte model zelf toe te passen, maar hoef je je module als het goed is niet te wijzigen.

> **Waarschuwing:**
>
> De code in je module mag gebruik maken van alle functies uit de Python Standard Library (zoals `math`, `random`, `itertools`, enzovoorts); het is *niet* toegestaan om functies toe te passen uit overige modules (zoals `numpy`, `sklearn`, `tensorflow`, enzovoorts).

Eerst zetten we wat initialisatie op en importeren we naast de `data` en `model` modules enkele onderdelen van `pandas`, `numpy`, en `time`. Plaats de cursor in de cel hieronder en druk op Ctrl+Enter (of Shift+Enter om meteen naar de volgende cel te gaan).

In [1]:
```
%matplotlib inline
%reload_ext autoreload
%autoreload 2
```

```
from pandas import DataFrame, __version__
print(f'Using pandas version {__version__}')

from numpy import array, __version__
print(f'Using numpy version {__version__}')

from time import perf_counter

import vlearning
from vlearning import data, __version__
from vlearning import activation_functions, loss_functions, layers
print(f'Using vlearning version {__version__}')
```

```
Using pandas version 2.2.2
Using numpy version 1.26.4
Using vlearning version 0.6.4
```

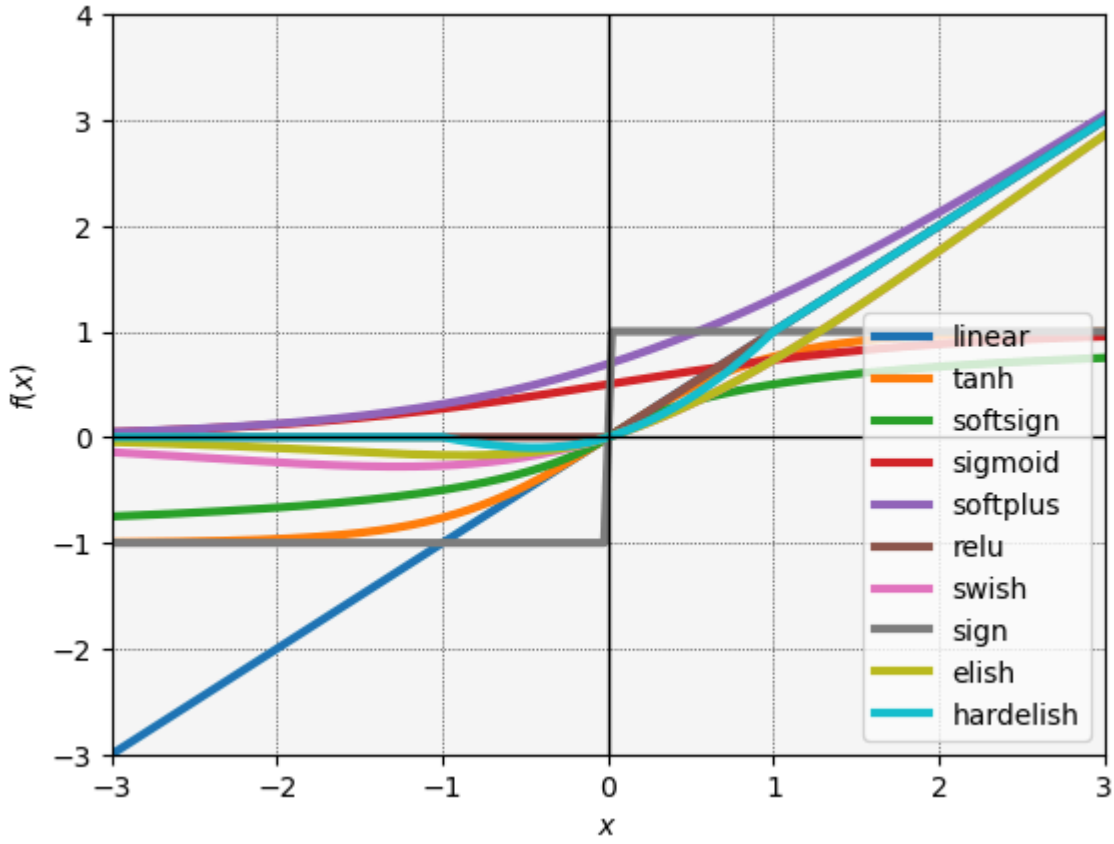
Deel A

Hieronder staan een aantal fragmenten code die je model *ongewijzigd* dient te kunnen uitvoeren. Voor verdere details omtrent deze gevraagde functionaliteiten, zie zonodig de werkcollege-opdrachten en/of de syllabus.

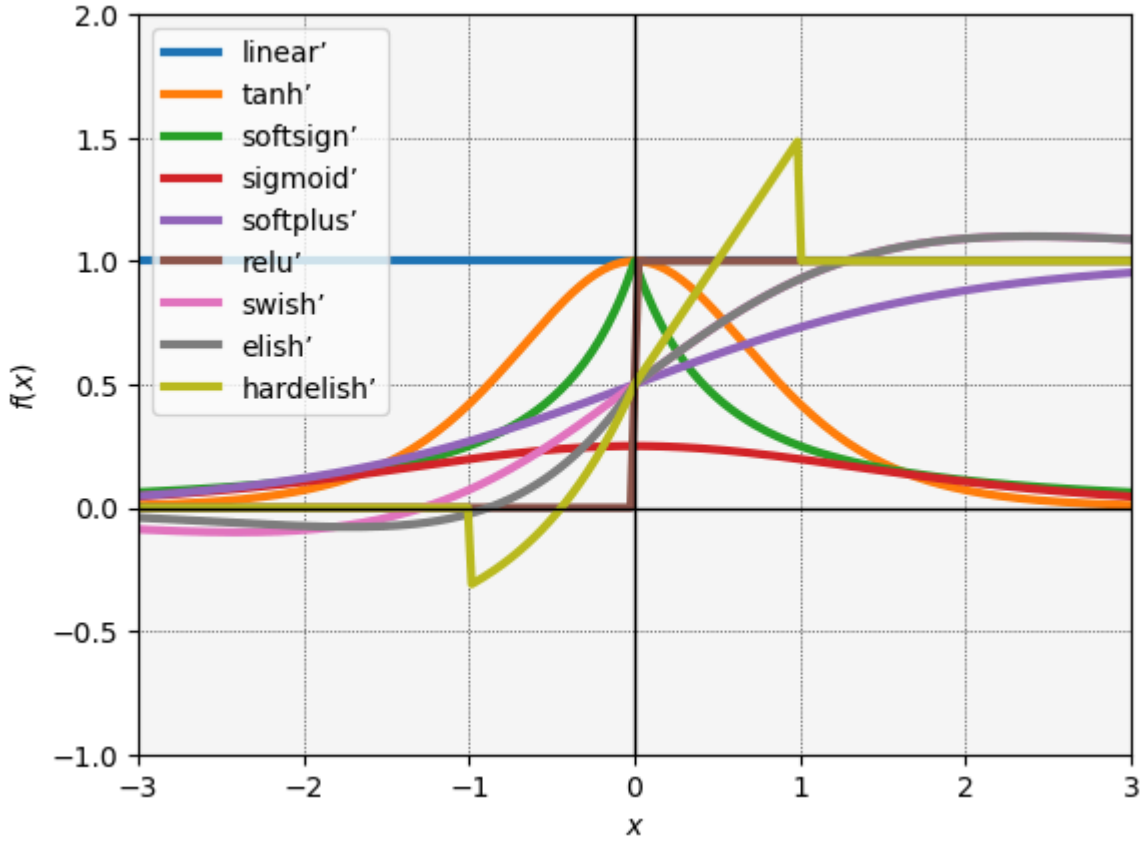
Activatiefuncties

In [2]:
```
my_activations = [
    activation_functions.linear,
    activation_functions.tanh,
    activation_functions.softsign,
    activation_functions.sigmoid,
    activation_functions.softplus,
    activation_functions.relu,
    activation_functions.swish,
    activation_functions.sign,
    activation_functions.elish,
    activation_functions.hardelish,
]
# my_activations = [
#     getattr(activation_functions, func_name)
#     for func_name in activation_functions.__all__
# ]
my_arguments =  [-1000, -1, 0, 1, 1000]
my_table = [[φ(a) for a in my_arguments] for φ in my_activations]
my_columns = [f'φ({a})' for a in my_arguments]
my_rows = [φ.__name__ for φ in my_activations]
```

In [3]:
```
data.graph(my_activations)
```



In [4]:
```
data.graph([vlearning.derivative(φ) for φ in my_activations if φ != activati
```



In [5]:
```
DataFrame(my_table, columns=my_columns).set_index(array(my_rows))
```

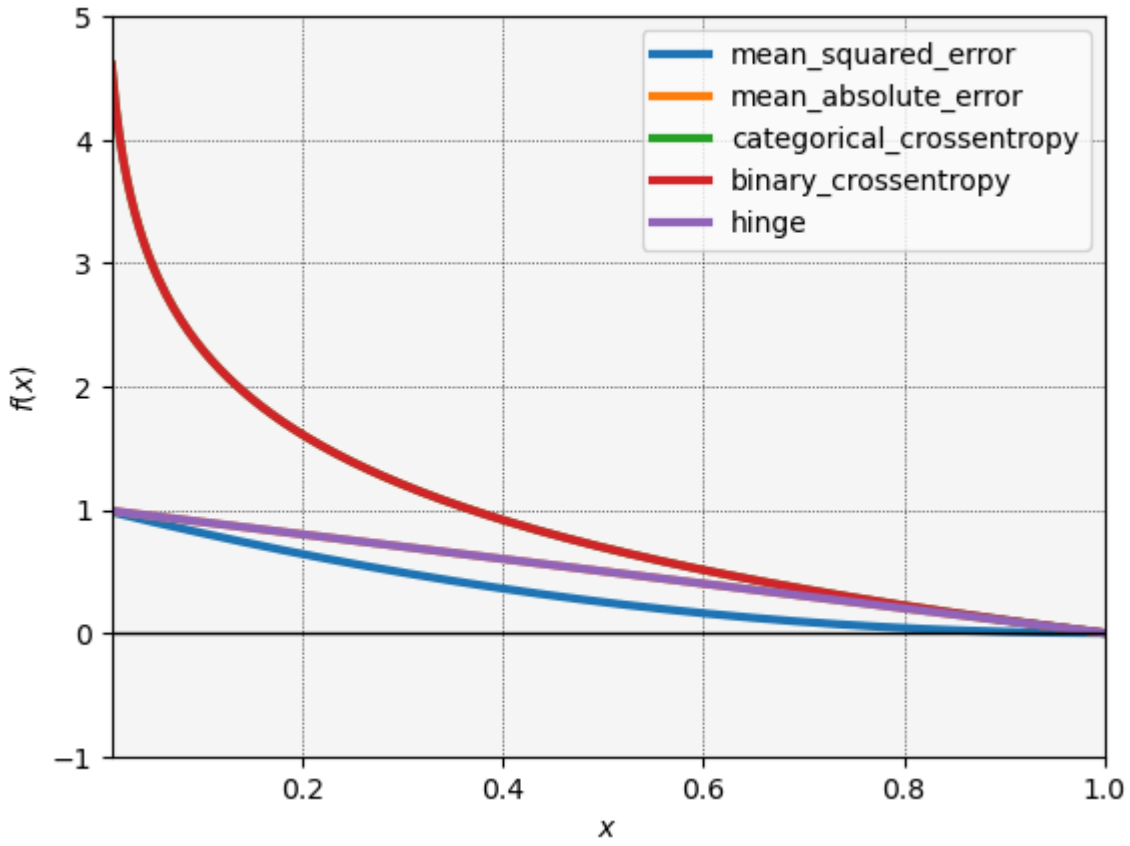
Out[5]:

	ϕ(-1000)	ϕ(-1)	ϕ(0)	ϕ(1)	ϕ(1000)
linear	-1000.000000	-1.000000	0.000000	1.000000	1000.000000
tanh	-1.000000	-0.761594	0.000000	0.761594	1.000000
softsign	-0.999001	-0.500000	0.000000	0.500000	0.999001
sigmoid	0.000000	0.268941	0.500000	0.731059	1.000000
softplus	0.000000	0.313262	0.693147	1.313262	1000.000000
relu	0.000000	0.000000	0.000000	1.000000	1000.000000
swish	-0.000000	-0.268941	0.000000	0.731059	1000.000000
sign	-1.000000	-1.000000	0.000000	1.000000	1.000000
elish	-0.000000	-0.170003	0.000000	0.731059	1000.000000
hardelish	-0.000000	-0.000000	0.000000	1.000000	1000.000000

Lossfuncties

In [6]:
```
my_losses = [
    loss_functions.mean_squared_error,
    loss_functions.mean_absolute_error,
    loss_functions.categorical_crossentropy,
    loss_functions.binary_crossentropy,
    loss_functions.hinge
]
# my_losses = [
#     getattr(loss_functions, func_name)
#     for func_name in loss_functions.__all__
# ]
my_arguments =  [0.01, 0.1, 0.5, 0.9, 0.99]
my_table = [[L(a, 1.0) for a in my_arguments] for L in my_losses]
my_columns = [f'L({a}; 1)' for a in my_arguments]
my_rows = [L.__name__ for L in my_losses]
```

In [7]:
```
data.graph(my_losses, 1.0, xlim=(1e-2, 1.0))
```



In [8]:
```
DataFrame(my_table, columns=my_columns).set_index(array(my_rows))
```

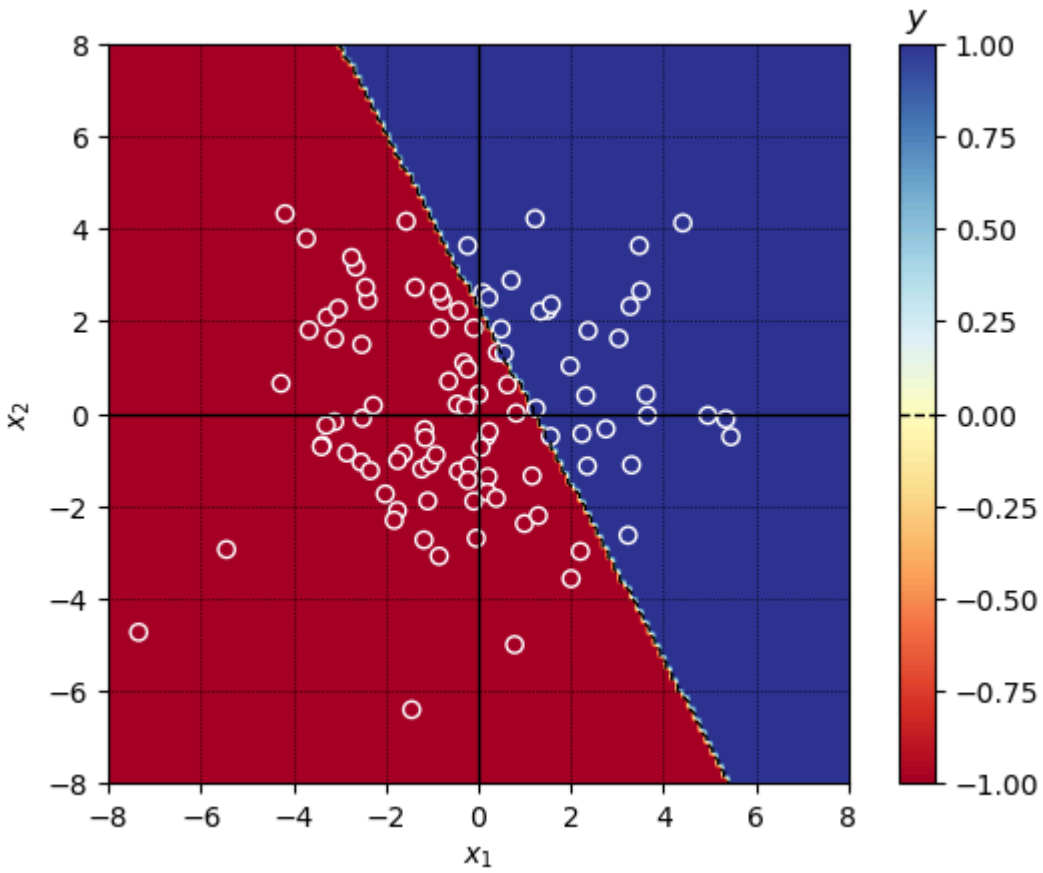
Out[8]:

	L(0.01; 1)	L(0.1; 1)	L(0.5; 1)	L(0.9; 1)	L(0.99; 1)
mean_squared_error	0.98010	0.810000	0.250000	0.010000	0.00010
mean_absolute_error	0.99000	0.900000	0.500000	0.100000	0.01000
categorical_crossentropy	4.60517	2.302585	0.693147	0.105361	0.01005
binary_crossentropy	4.60517	2.302585	0.693147	0.105361	0.01005
hinge	0.99000	0.900000	0.500000	0.100000	0.01000

Classificatie: single-layer perceptron

In [9]:
```
xs, ys = data.linear('nominal')
my_model = vlearning.Perceptron(dim=2)
my_model.fit(xs, ys)
data.scatter(xs, ys, model=my_model)
print(my_model)
```

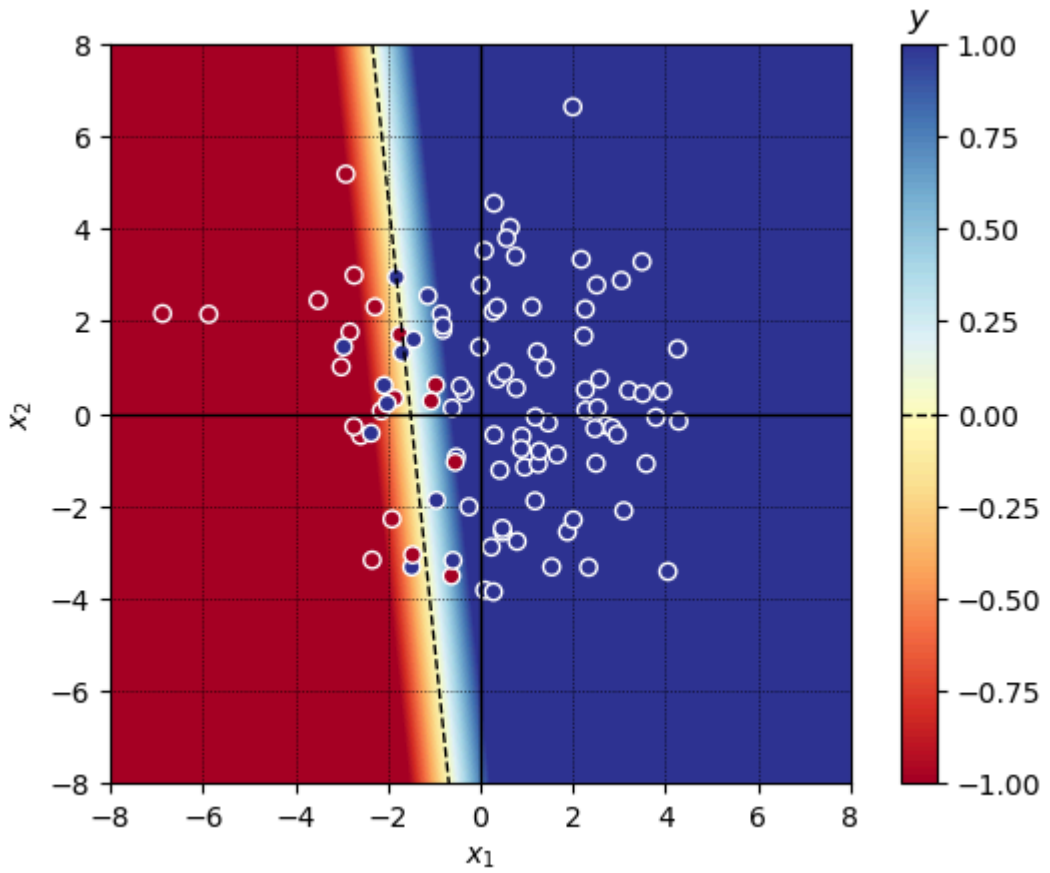
```
Model has been fully fitted after 20 epochs
```




```
Perceptron(dim=2)
```

Classificatie: support vector machine

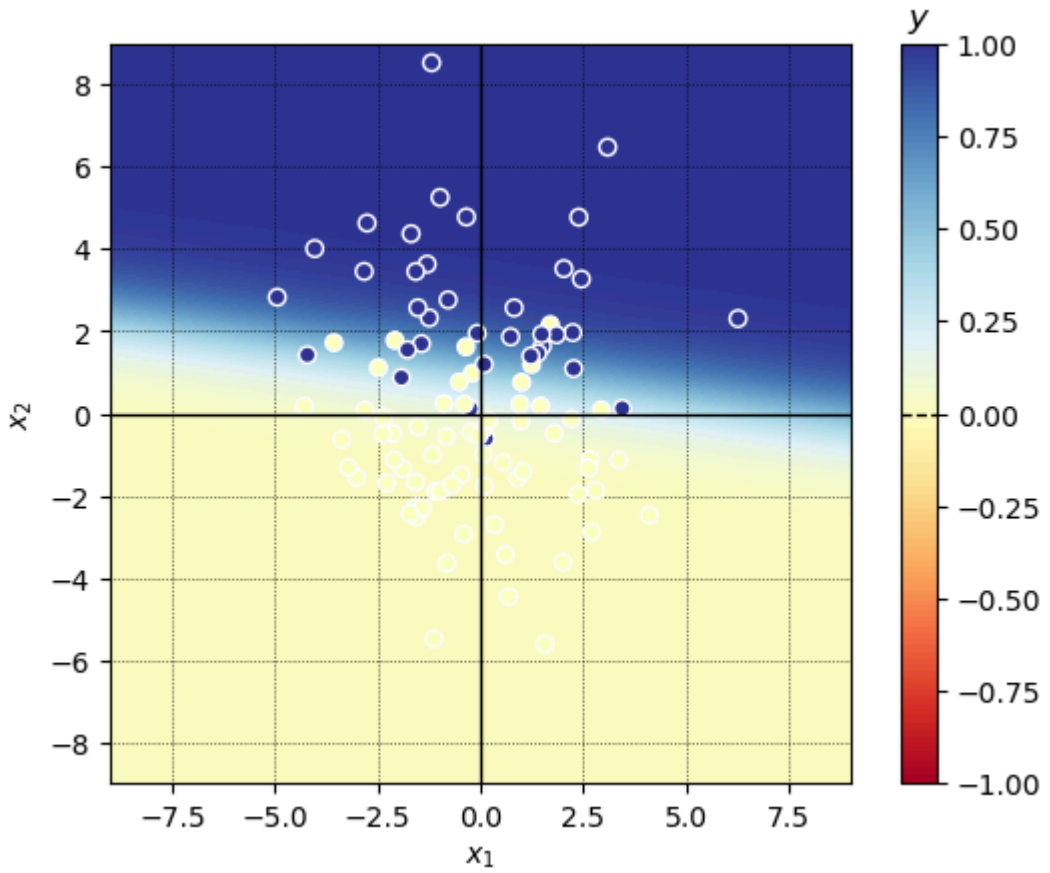
In [10]:
```
xs, ys = data.linear(outcome='nominal', noise=1.0)
my_model = vlearning.Neuron(dim=2, loss=loss_functions.hinge)
my_model.fit(xs, ys)
data.scatter(xs, ys, model=my_model)
print(my_model)
```




```
Neuron(dim=2, activation=linear, loss=hinge)
```

Classificatie: binomiale logistische regressie

In [11]:
```
xs, ys = data.linear(outcome='nominal', noise=1.0)
ys = [(y + 1.0) / 2.0 for y in ys]   # Convert labels -1/+1 to 0/1
my_model = vlearning.Neuron(dim=2, activation=activation_functions.sigmoid,
my_model.fit(xs, ys)
data.scatter(xs, ys, model=my_model)
print(my_model)
```

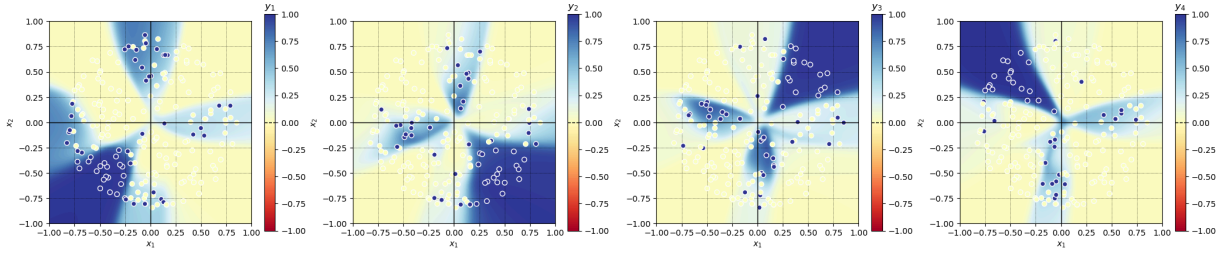



```
Neuron(dim=2, activation=sigmoid, loss=binary_crossentropy)
```

Classificatie: multi-layer perceptron

In [12]:
```
xs, ys = data.fractal(4)
my_model = (
    layers.InputLayer(2) +
    layers.DenseLayer(20) + layers.ActivationLayer(20, activation=activation
    layers.DenseLayer(10) + layers.ActivationLayer(10, activation=activation
    layers.DenseLayer(4) + layers.SoftmaxLayer(4) +
    layers.LossLayer(loss=loss_functions.categorical_crossentropy)
)
my_model.fit(xs, ys, alpha=0.5, epochs=100, batch_size=20)
my_model.fit(xs, ys, alpha=0.5, epochs=10)
data.scatter(xs, ys, model=my_model)
print(my_model)
```

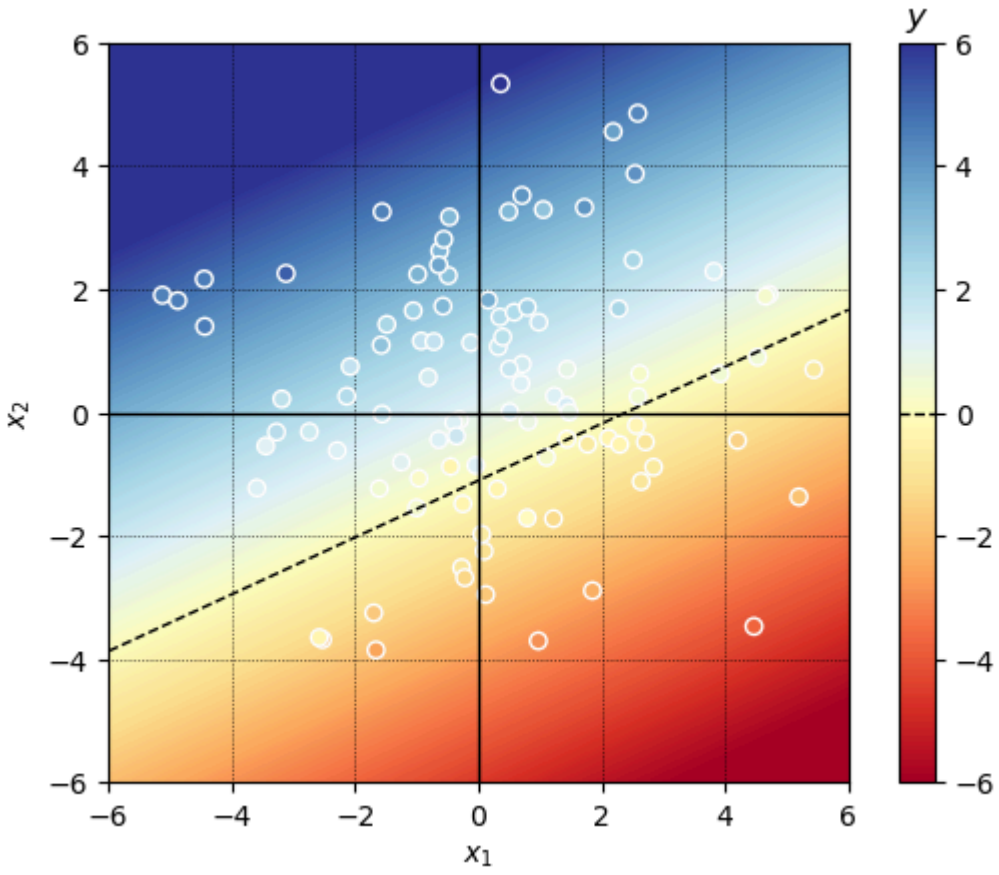
```
Epochs trained: 100%|████████████████████████████████████████████████████
████████████████| 100/100 [00:01<00:00, 65.64epoch/s]
Epochs trained: 100%|████████████████████████████████████████████████████
████████████████| 10/10 [00:00<00:00, 67.99epoch/s]
```



```
InputLayer(num_outputs=2, name='InputLayer_1') +
        DenseLayer(num_outputs=20, name='DenseLayer_1') +
        ActivationLayer(num_outputs=20, name='ActivationLayer_1', activation
='tanh') +
        DenseLayer(num_outputs=10, name='DenseLayer_2') +
        ActivationLayer(num_outputs=10, name='ActivationLayer_2', activation
='tanh') +
        DenseLayer(num_outputs=4, name='DenseLayer_3') +
        SoftmaxLayer(num_outputs=4, name='SoftmaxLayer_1') +
        LossLayer(num_inputs=4, name='LossLayer_1', loss='categorical_crosse
ntropy')
```

Regressie: lineaire regressie

In [13]:
```
xs, ys = data.linear('numeric', noise=0.5)
my_model = vlearning.LinearRegression(dim=2)
my_model.fit(xs, ys)
data.scatter(xs, ys, model=my_model)
print(my_model)
```



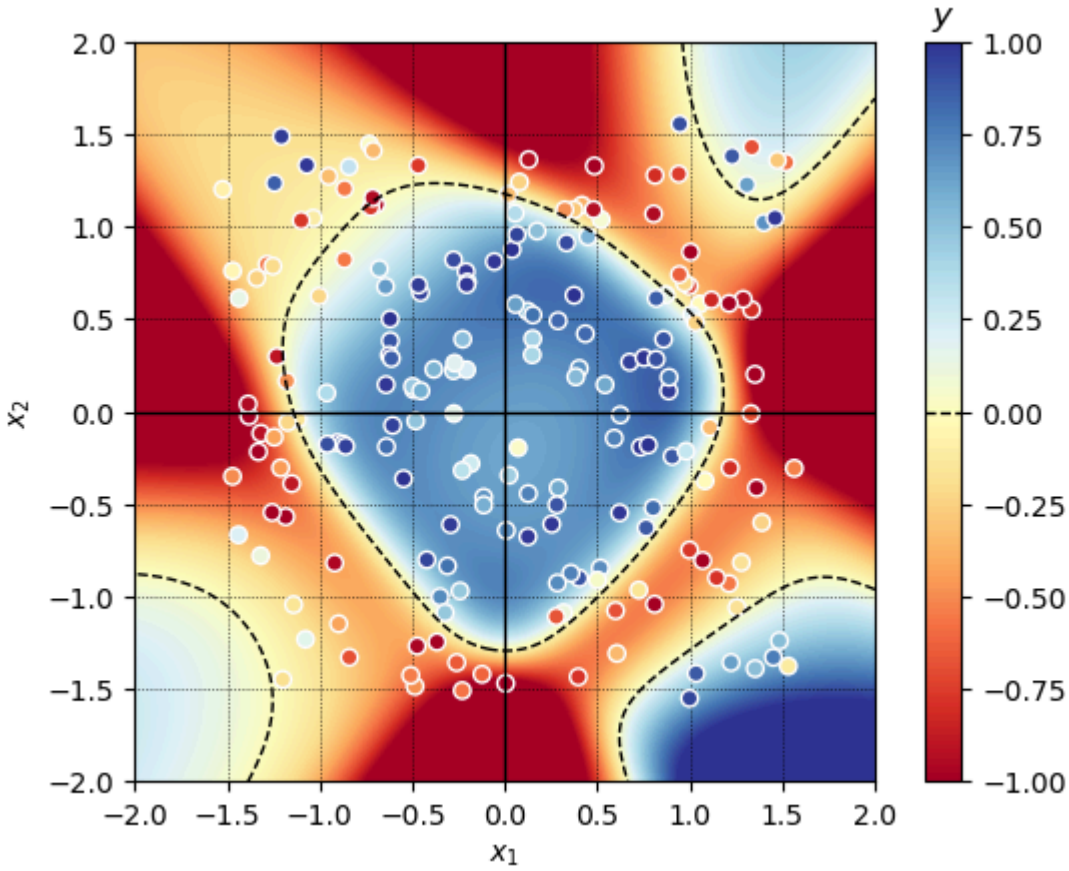

```
LinearRegression(dim=2)
```

Regressie: neuraal netwerk

In [14]:
```
xs, ys = data.concentric(noise=0.1)
my_model = (
    layers.InputLayer(2) +
    layers.DenseLayer(20) + layers.ActivationLayer(20, activation=activation
    layers.DenseLayer(10) + layers.ActivationLayer(10, activation=activation
```

```
    layers.DenseLayer(1) + layers.ActivationLayer(1, activation=activation_f
    layers.LossLayer(loss=loss_functions.mean_squared_error)
)
my_model.fit(xs, ys, alpha=0.05, epochs=200, batch_size=10)
my_model.fit(xs, ys, alpha=0.05, epochs=20)
data.scatter(xs, ys, model=my_model)
print(my_model)
```

```
Epochs trained: 100%|██████████████████████████████████████████████████
██████████████| 200/200 [00:02<00:00, 70.81epoch/s]
Epochs trained: 100%|██████████████████████████████████████████████████
██████████████| 20/20 [00:00<00:00, 67.82epoch/s]
```




```
InputLayer(num_outputs=2, name='InputLayer_2') +
        DenseLayer(num_outputs=20, name='DenseLayer_4') +
        ActivationLayer(num_outputs=20, name='ActivationLayer_3', activation
='tanh') +
        DenseLayer(num_outputs=10, name='DenseLayer_5') +
        ActivationLayer(num_outputs=10, name='ActivationLayer_4', activation
='tanh') +
        DenseLayer(num_outputs=1, name='DenseLayer_6') +
        ActivationLayer(num_outputs=1, name='ActivationLayer_5', activation
='linear') +
        LossLayer(num_inputs=1, name='LossLayer_2', loss='mean_squared_erro
r')
```

Deel B

In dit deel ga je met een klassieke dataset aan de slag, de Extended MNIST dataset die bestaat uit duizenden afbeeldingen van 28x28 pixels met handgeschreven cijfers en letters. Beschikbaar op BlackBoard is een bestand **EMNIST_mini.dat** (dat je dient te unzippen uit **EMNIST_mini.zip**) met een geminiaturiseerde versie met afbeeldingen van 12x12 pixels van alleen dat deel van de dataset dat betrekking heeft op de 26 verschillende hoofdletters. In totaal zijn er 52.000 instances beschikbaar, 2.000 van elke letter. De functie `data.emnist_mini()` kan gebruikt worden om een aantal instances op te vragen. Deze functie levert de attributen van de instances in de vorm van 144 pixel-intensiteiten tussen 0 en 1, en de klasselabels in de vorm van 26 getalwaarden met het juiste cijfer als een one-hot encoding.

In [15]:
```
help(data.emnist_mini)
```

```
Help on function emnist_mini in module vlearning.data:

emnist_mini(filename, num=52000, seed=None)
    Returns a number of different random 12x12 EMNIST images.

    Keyword arguments:
    filename -- full filename of the *.dat datafile
    num      -- number of images to randomly select (default 52000)
    seed     -- a seed to initialise the random number generator (default ra
ndom)

    Return values:
    xs       -- 144-element lists of pixel values (range 0.0-1.0)
    ys       -- 26-element lists of correct letters using one-hot encoding
```

Hiervan genereren we aanvankelijk om het simpel te houden slechts driehonderd instances elk voor de trainings-, validatie- en testdata. Onderzoek zelf de organisatie van deze data nader.

In [16]:
```
# STAP 1: DATAGENERATIE
xs, ys = data.emnist_mini('./EMNIST_mini.dat', num=900)
trn_xs, trn_ys = xs[  0:300], ys[  0:300]
val_xs, val_ys = xs[300:600], ys[300:600]
tst_xs, tst_ys = xs[600:900], ys[600:900]
```

Hieronder wordt een dummy model aangemaakt dat in dit geval bestaat uit een input, een hidden, en een outputlayer. Er is weliswaar vanalles aan te merken op dit overgesimplificeerde model, maar hanteer dit als een eerste uitgangspunt.

In [17]:
```
# STAP 2: MODELDEFINITIE
my_model = (
    layers.InputLayer(144, name='input') +
    layers.DenseLayer(26, name='hidden1') + layers.ActivationLayer(26, name=
    layers.LossLayer(name='output')
)
```

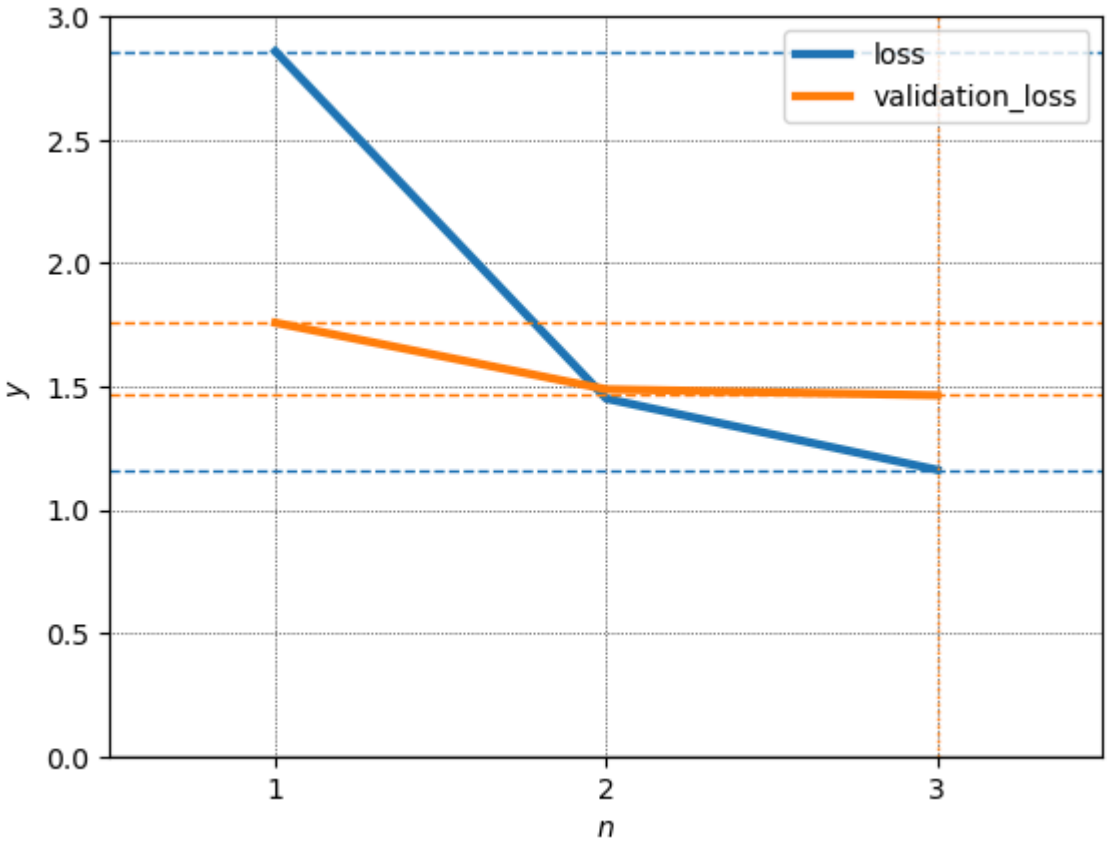
Vervolgens trainen en evalueren we dit model als volgt. Wederom zijn de gekozen parameters ongetwijfeld niet optimaal.

In [18]:
```
# STAP 3: TRAINING
my_history = my_model.fit(trn_xs, trn_ys, alpha=0.01, epochs=3, batch_size=1
```

```
Epochs trained: 100%|████████████████████████████████████████████████████
████████████████████| 3/3 [00:01<00:00,  2.67epoch/s]
```

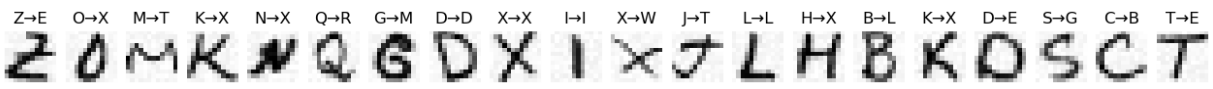
Echter, hiermee kunnen we een validatiecurve construeren.

In [19]:
```
# STAP 4: VALIDATIECURVE
data.curve(my_history)
```



Om inzicht te krijgen in de prestaties van het model, worden hieronder twintig instances uit de testdata getoond met voor en na de pijl respectievelijk de juiste en de voorspelde klasselabels.

In [20]:
```
# STAP 5: VISUALISATIE
data.letters(tst_xs[:20], tst_ys[:20], model=my_model)
```



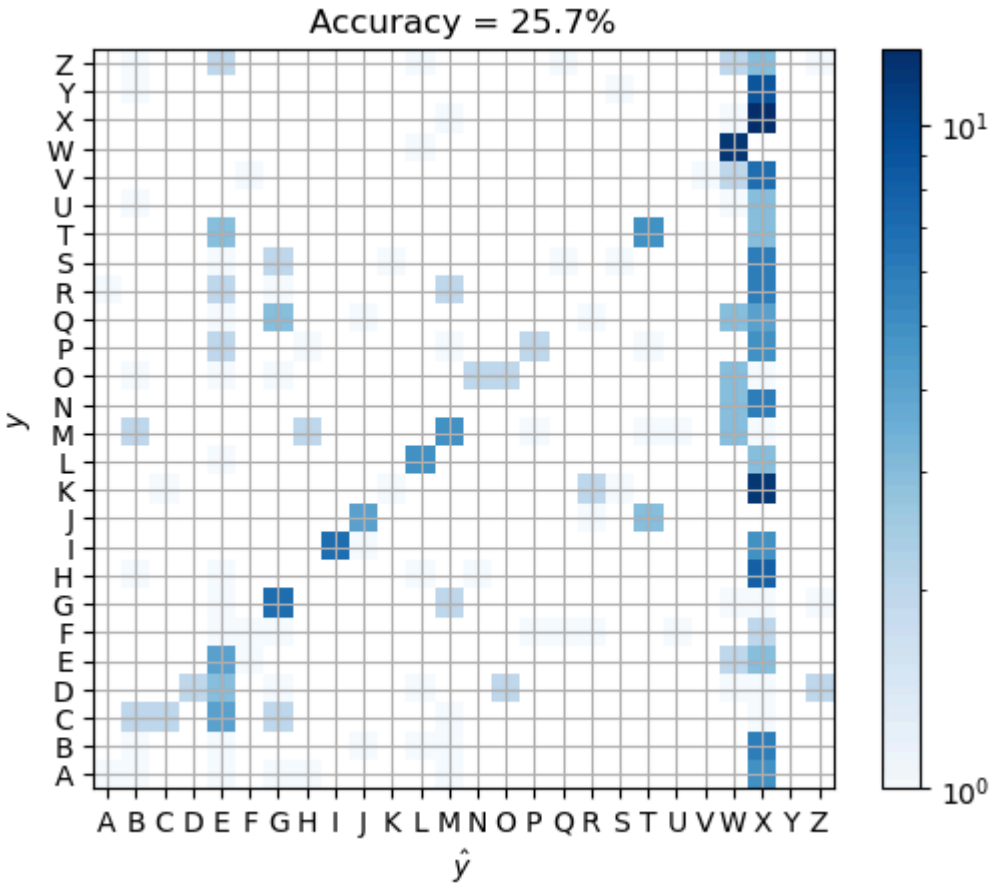
Berekenen we eens de gemiddelde loss op alle testdata.

In [21]:
```
# STAP 6: EVALUATIE
print(f'Loss: {my_model.evaluate(tst_xs, tst_ys)}')
```

```
Loss: 1.451217528759483
```

Dit getal zegt misschien nog niet zoveel. Daarom bekijken we een grafische weergave van de confusion matrix die weergeeft welke voorspelde klasselabels op de x-as aan alle echte klasselabels op de y-as worden toegekend (let op de logaritmische kleurschaal).

In [22]:
```
# STAP 7: CONFUSIONMATRIX
data.confusion(tst_xs, tst_ys, model=my_model)
```



Hoewel er best wat fouten worden gemaakt liggen er toch behoorlijk wat juist geclassificeerde instances op de diagonaal. Daarnaast, een aantal van de meest gemaakte fouten betreft letters die ook wel enigszins op elkaar lijken. Dit overdreven simpele model bereikt - afhankelijk van de willekeurig gekozen instances en initialisatiewaarden - een nauwkeurigheid van rond de 25%, wat wil zeggen dat ongeveer een vierde van de letters correct wordt herkend. Dit is nog niet indrukwekkend goed, maar gezien de eenvoudige opbouw van het model al best verrassend en in elk geval ruim boven de 1/26 ≈ 4% nauwkeurigheid die je mag verwachten op grond van kans alleen.

Pas nu hieronder eens het bovenstaande model aan tot een neuraal netwerk dat deze afbeeldingen redelijk betrouwbaar kan classificeren. Kies zelf een geschikte opzet van het model en bepaal door te experimenteren geschikte waarden voor de diverse parameters. Voer dezelfde zeven stappen uit als hierboven, maar dan met een effectiever en beter geoptimaliseerd model.

> **Opdracht:**
>
> Gebruik tenminste ergens in je model de *exponential linear sigmoid squashing* (ELiSH) of de *hard exponential linear sigmoid squashing* (HardELiSH) activatiefunctie die door Mina Basirat & Peter M. Roth als optimaal werden gerapporteerd (in "The Quest for the Golden Activation Function", 2018). Zoek hier zonodig informatie over op en implementeer deze in je module; houd daarbij rekening met overflow.

In [23]:
```
# Verander deze cel niet
starttime = perf_counter()
```

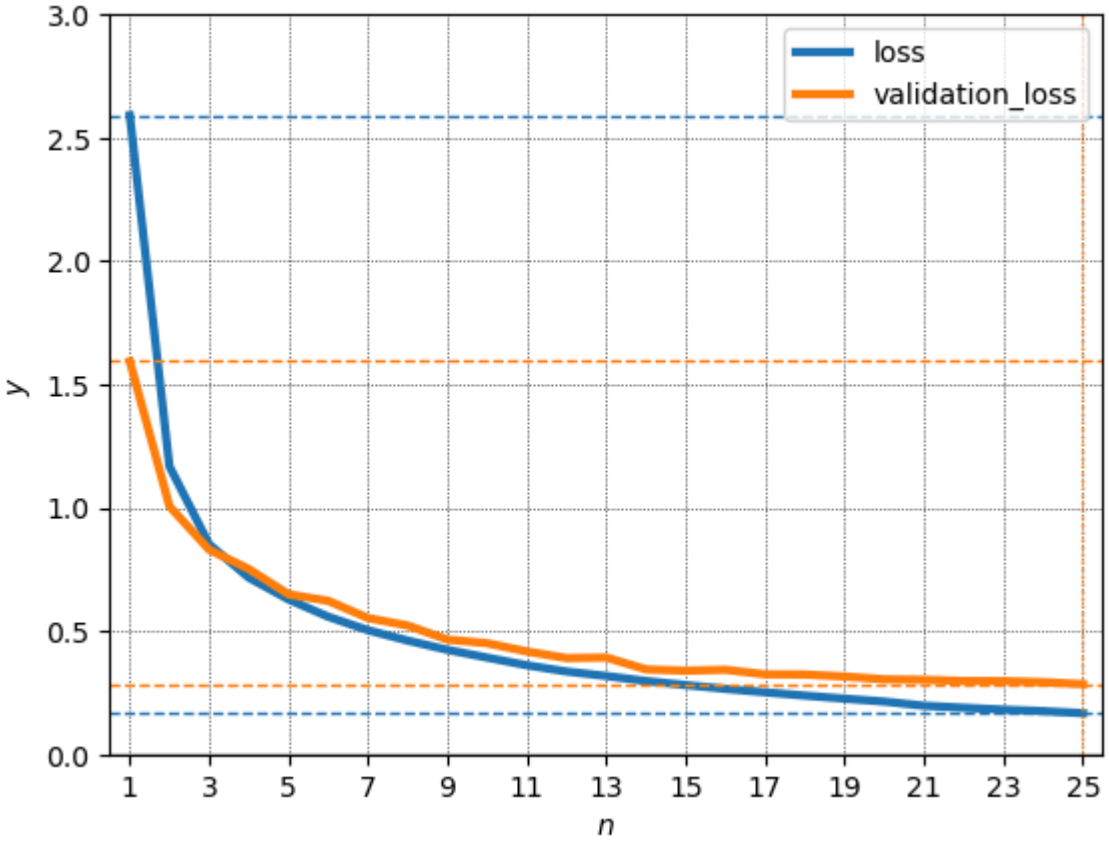
In [24]:
```
# STAP 1: DATAGENERATIE
# amount_of_instances = 52_000
amount_of_instances = 26_000
xs, ys = data.emnist_mini('./EMNIST_mini.dat', num=amount_of_instances)

trn_size = int(0.90 * amount_of_instances)
val_size = int(0.05 * amount_of_instances)
tst_size = int(0.05 * amount_of_instances)

trn_xs, trn_ys = xs[:trn_size], ys[:trn_size]
val_xs, val_ys = xs[trn_size:trn_size+val_size], ys[trn_size:trn_size+val_si
tst_xs, tst_ys = xs[trn_size+val_size:], ys[trn_size+val_size:]
```

In [25]:
```
# STAP 2: MODELDEFINITIE
my_model = (
    layers.InputLayer(144) +
    layers.DenseLayer(120) + layers.ActivationLayer(120, activation=activati
    layers.DenseLayer(60) + layers.ActivationLayer(60, activation=activation
    layers.DenseLayer(30) + layers.ActivationLayer(30, activation=activation
    layers.DenseLayer(26) + layers.SoftmaxLayer(26) +
    layers.LossLayer(loss=loss_functions.categorical_crossentropy)
)
```

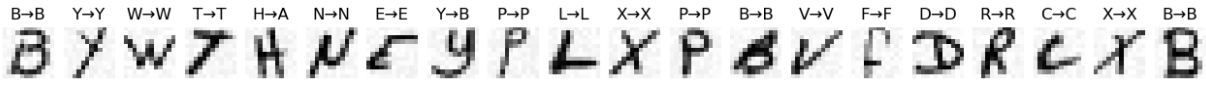
In [26]:
```
# STAP 3: TRAINING
my_history = my_model.fit(trn_xs, trn_ys, alpha=0.10, epochs=10, batch_size=
my_history = my_model.fit(trn_xs, trn_ys, alpha=0.05, epochs=10, batch_size=
my_history = my_model.fit(trn_xs, trn_ys, alpha=0.01, epochs=5, batch_size=1
```

```
Epochs trained: 100%|████████████████████████████████████████████████████| 10/10 [13:17<00:00, 79.73s/epoch]
Epochs trained: 100%|████████████████████████████████████████████████████| 10/10 [13:30<00:00, 81.03s/epoch]
Epochs trained: 100%|████████████████████████████████████████████████████| 5/5 [07:23<00:00, 88.71s/epoch]
```

In [27]:
```
# STAP 4: VALIDATIECURVE
data.curve(my_history)
```



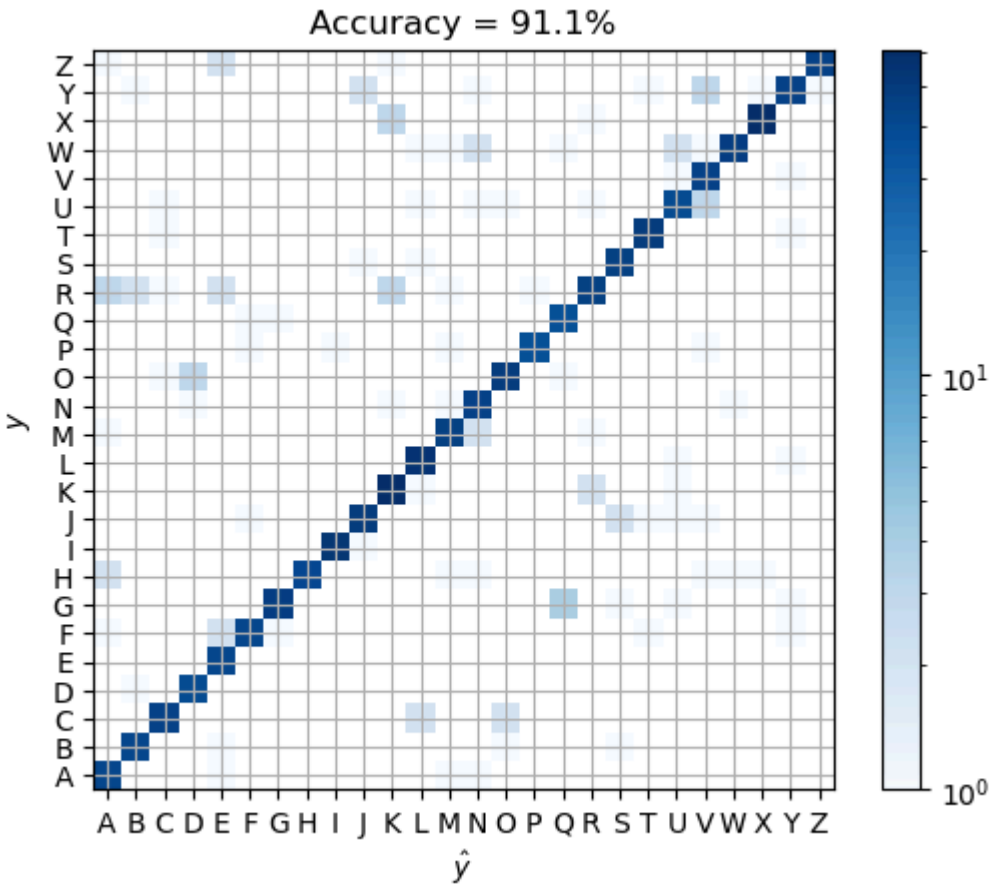
In [28]:
```
# STAP 5: VISUALISATIE
data.letters(tst_xs[:20], tst_ys[:20], model=my_model)
```



In [29]:
```
# STAP 6: EVALUATIE
print(f'Loss: {my_model.evaluate(tst_xs, tst_ys)}')
```

```
Loss: 0.3059161652120169
```

In [30]:
```
# STAP 7: CONFUSIONMATRIX
data.confusion(tst_xs, tst_ys, model=my_model)
```



In [31]:
```
# Verander deze cel niet
print(f'Verstreken tijd: {(perf_counter() - starttime) / 60.0:.1f} minuten.'
```

```
Verstreken tijd: 34.3 minuten.
```

Hint:

Voer achtereenvolgens de onderstaande ontwikkelstappen uit.

- creëer eerst een model met meerdere hidden layers waarin gezamenlijk in de ordegrootte van ruim een honderdtal neuronen verwerkt zijn;
- stel dan de invoer- & uitvoerlagen en activatie- & loss-functies zo in dat het model geschikt is voor deze classificatie-taak;
- begin met een datasetje van zeer beperkte grootte zodat het model in niet meer dan ongeveer een minuut over een klein aantal epochs te trainen is;
- kies een grootte voor de mini-batches die naar jouw inschatting net genoeg is om een enigszins representatieve steekproef van de data te vormen;
- probeer aanvankelijk een relatief grote learning rate uit en stel deze bij naar beneden zolang het model niet in staat is een dalende validatie-curve te tonen;

- start met enkele epochs en voer dit op totdat de validatiecurve aangeeft dat het model redelijk getraind raakt (de trainingstijd neemt hierbij evenredig toe);
- vergroot dan geleidelijk de grootte van de datasets (waarbij het nodige aantal epochs afneemt omdat er per epoch meer mini-batches getraind worden);
- je mag alle 52.000 instances uiteindelijk gebruiken, maar dat is niet verplicht;
- speel met de bovenstaande procedure tot je een model hebt gevonden dat in een werkbare tijd toch naar tevredenheid convergeert.

Opmerking:

Ter indicatie, een deugdelijk model is in zijn uiteindelijke vorm op een typische hedendaagse CPU na enkele tientallen minuten training (eis: maximaal 1 uur) in staat om ruim 80% (eis: minimaal 70%) accuracy te behalen zonder daarbij zichtbaar te overfitten.

Afsluiting

Als je klaar bent, lever dan je uitwerkingen als volgt in:

1. Sla je model vanuit je code-editor op als **model.py**;
2. Evalueer dit notebook door vanuit het menu *Kernel > Restart & Run All* te kiezen;
3. Controleer dat alle uitvoer correct en volledig wordt geproduceerd;
4. Exporteer dit notebook als **Eindopdracht_v2324.2.html** vanuit het menu *File > Download as > HTML (.html)*;
5. Verwijder vervolgens de uitvoer in dit notebook via het menu *Cell > All Output > Clear*;
6. Sla dit notebook op als **Eindopdracht_v2324.2.ipynb** middels het menu *File > Save and Checkpoint*;
7. Comprimeer alledrie de hierboven genoemde bestanden in één bestand **Eindopdracht_v2324.2.zip**;
8. Lever je zip-bestand uiterlijk **zondag 23 juni 2024** (23:59) in op BlackBoard;
9. E-mail de docent met je voorkeurstijdstippen voor het mondelinge tentamen.

Waarschuwing:

Verifieer dat je het juiste bestand uploadt, want eenmaal ingestuurd werk geldt als definitief!

Succes!
