



**Odisee**  
DE CO-HOGESCHOOL

# Machine Learning – Week 2



## How to participate?




Click on the projected screen to start the question

 [Copy participation link](#)

wooclap

 100 % 

26 



Go to **wooclap.com** and use the code **BTPMOD**



Welke termen houden verband met neurale netwerken?



Tensor  
Flow

jimmy  
neutron

Input/Output  
true

string  
rnd =  
new  
Random()

<insert  
random  
term>



brein

neuronen

Neurons,  
activatie  
functie

Brein

neuronen



neuronen

neuronen

activatief



wooclap



100 %



36



Go to **wooclap.com** and use the code **BTPMOD**



Wat is een activatiefunctie?



wiskundi  
ge  
functie  
die

Wanneer  
er  
genoeg  
verbodin

Lineaire  
regressie  
is een vb

Functie  
checkt  
als  
neuron

Activatief  
unctie  
bepaalt  
neuronac

Weet ge  
wnr ge

De  
manier



**wooclap**



100 %



7



Click on the projected screen to start the question

In welk geval gebruik je volgende activatiefunctie



tanh



zelden, eventueel voor binaire classificatie

lineair



regressie in output layer

relu



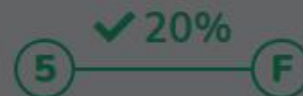
hidden layers

leaky-relu



af en toe voor hidden layers

sigmoid



multilabel classificatie

softmax



multiclass classificatie

Click on the projected screen to start the question



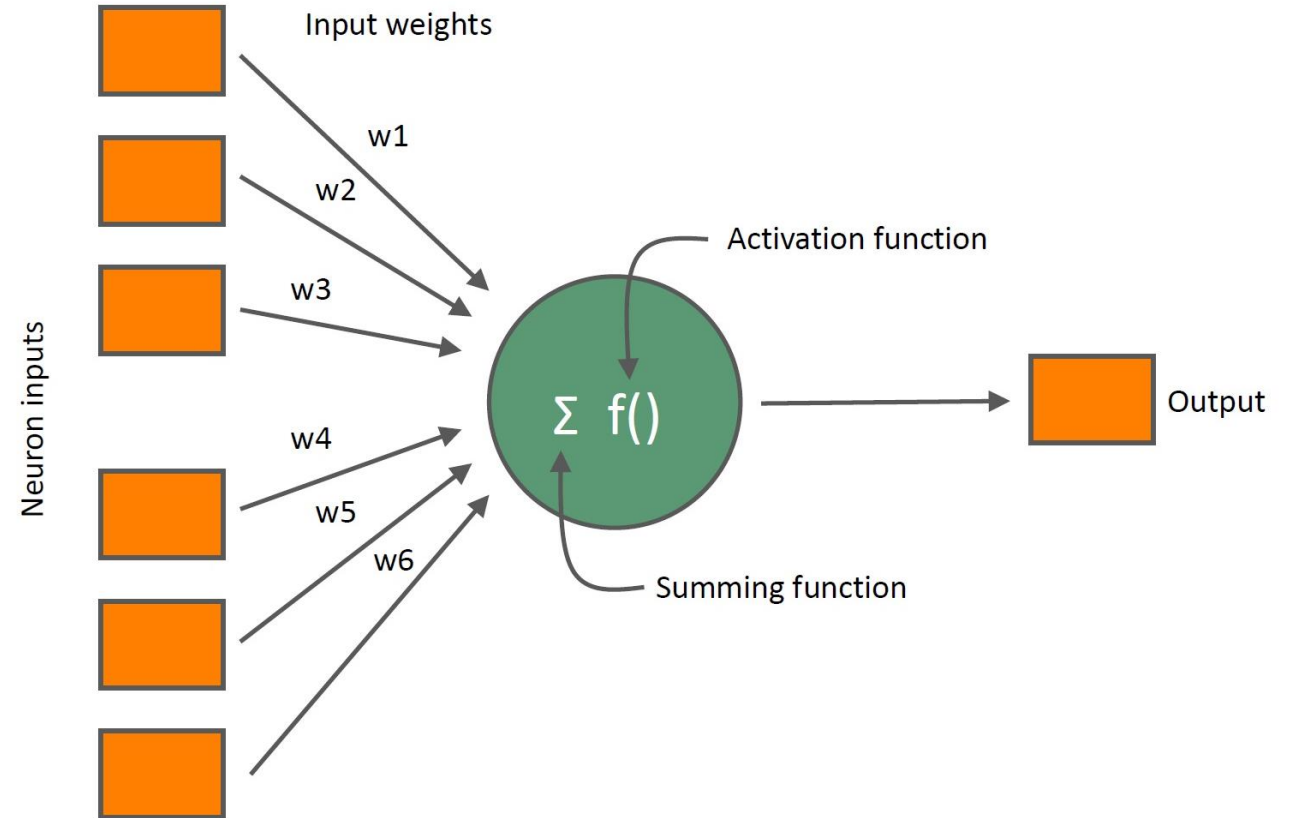
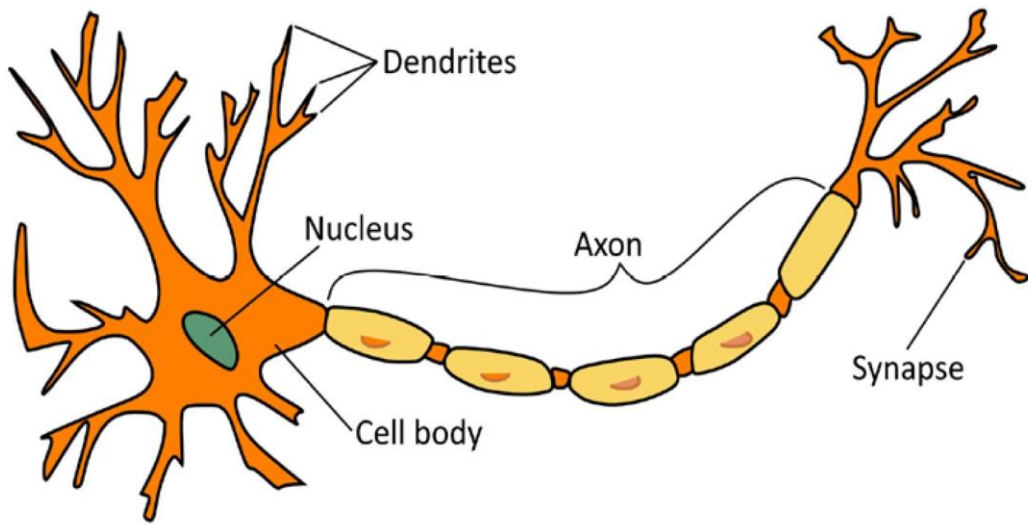
Welke uitdrukking(en) is/zijn waar?



- 1 In een feedforward neuraal netwerk gaat de data altijd in 1 richting 93% 14
- 2 In een netwerk gaat de data altijd in 1 richting 13% 2
- 3 vanishing gradient probleem wordt vermeden door het toevoegen van meer lagen 67% 10
- 4 dropout is een techniek om overfitting tegen te gaan 53% 8
- 5 l2-regularisatie is een techniek om overfitting tegen te gaan 60% 9
- 6 underfitting is niet mogelijk met neurale netwerken 13% 2



## Biologisch naar artificieel neuron





# Activation functions - samenvatting

## ▣ Hidden layers

- ▬ Relu eerste keuze, probeer erna leaky relu
- ▬ Geen sigmoid

## ▣ Output layer

- ▬ Regressie -> lineaire activatie functie
- ▬ Classificatie
  - Binaire classificatie -> sigmoid
  - Multiclass classificatie -> softmax
  - Multilabel classificatie -> sigmoid

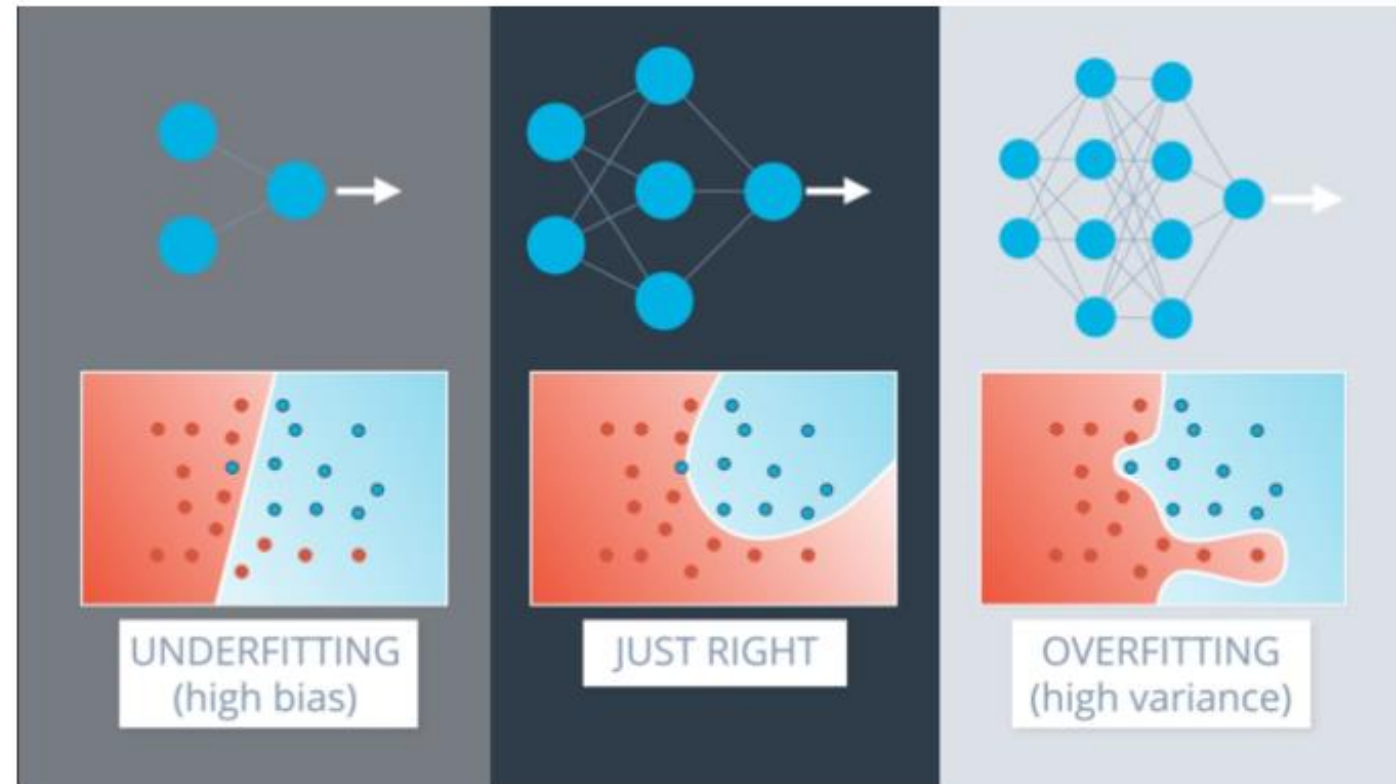
# Underfitting vs overfitting

## Overfitting

- Leert trainingsdata te sterk
- Testdata niet voldoende
- Netwerk te groot of onvoldoende data

## Underfitting

- Kan input niet goed modelleren
- Te klein netwerk

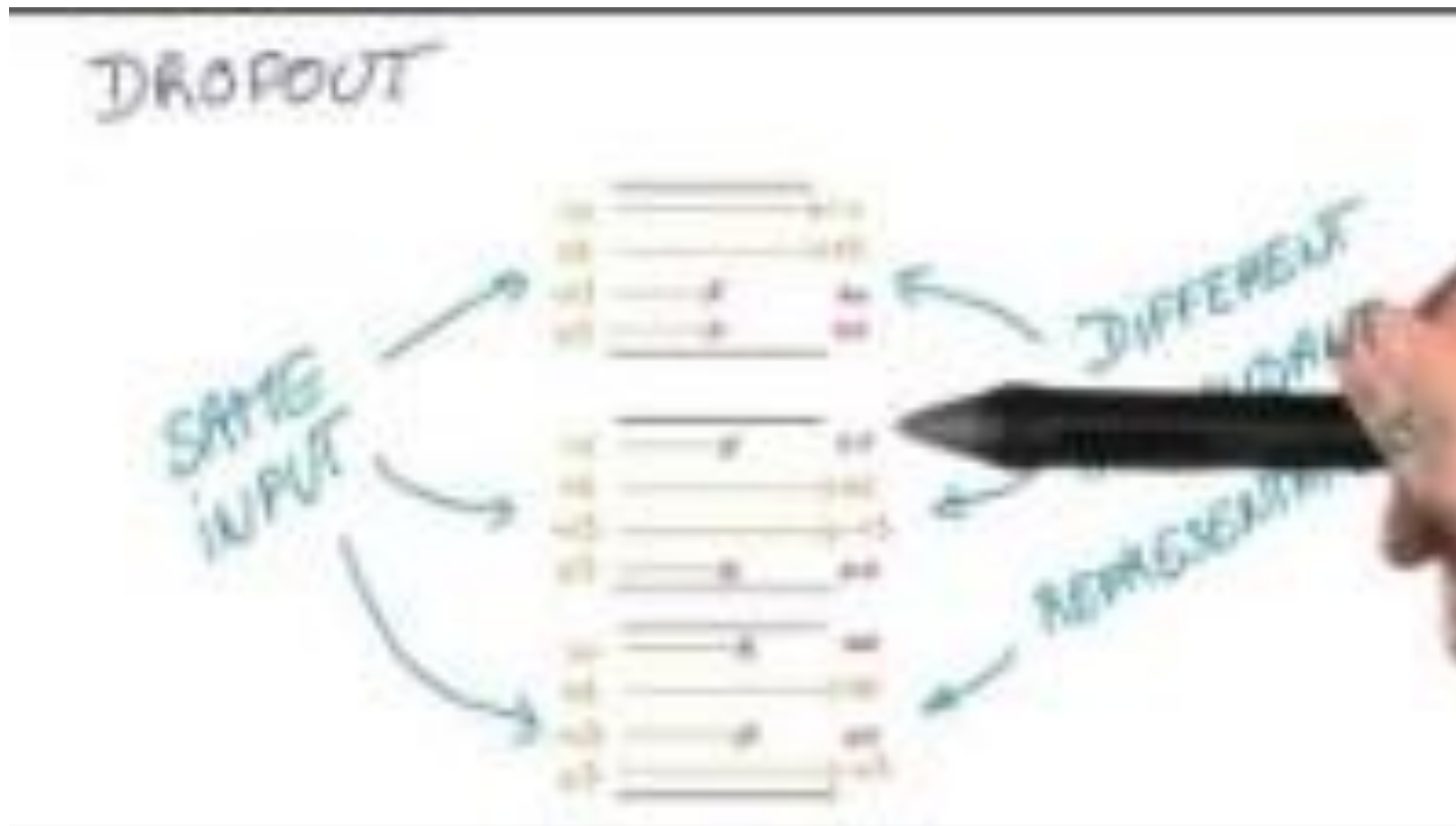


## Hoe overfitting tegengaan?

### ■ Weight regularisation

- ▬ Zorg ervoor dat de gewichten zo klein mogelijk zijn
- ▬ Hierdoor focust het netwerk op inputs die belangrijk zijn
- ▬ Extra kost in de loss/error functie op basis van de L2 of L1 norm
  - L2-norm = som van de kwadraten van de gewichten
  - L1-norm = som van de absolute waarden van de gewichten

## Hoe overfitting tegengaan?





## Hoe overfitting tegengaan

- ▣ Techniek om overfitting te voorkomen
- ▣ Willekeurig neurons uitschakelen bij training
  - ▬ Andere neurons moeten inspringen om een goed resultaat te bekomen
  - ▬ Vermijd dat andere neurons afsterven
- ▣ Bootst een ensemble van netwerken na in 1 netwerk
  - ▬ Robuster en accurater model



Inladen data



# Veel gebruikte datasets voor standaard-problemen/voorbeelden

▣ `tf.keras.datasets.xxxx.load_data()`

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

▣ Voor:

- Gestructureerde data
- Computer visie
- NLP

`boston_housing` module

`cifar10` module

`cifar100` module

`fashion_mnist` module

`imdb` module

`mnist` module

`reuters` module

## Gestructureerde data – methode 1 – data kan volledig in memory

- ▣ Laad eerst in via pandas
- ▣ Converteer het naar input tensors
  - Per kolom of in geheel
- ▣ Voorwaarde
  - Volledig ingeladen

```
titanic_file = tf.keras.utils.get_file("train.csv", "https://storage.googleapis.com/tf-datasets/titanic/train.csv")

df = pd.read_csv(titanic_file)
display(df.head())

# split inputs en outputs
titanic_features = df.copy()
titanic_labels = titanic_features.pop('survived')

inputs = {}
for name, column in titanic_features.items():
    dtype = column.dtype
    if dtype == object:
        dtype = tf.string
    else:
        dtype = tf.float32

    inputs[name] = tf.keras.Input(shape=(1,), name=name, dtype=dtype)

print(inputs)
```



age	InputLayer
input:	output:
[(None, 1)]	[(None, 1)]

n_siblings_spouses	InputLayer
input:	output:
[(None, 1)]	[(None, 1)]

parch	InputLayer
input:	output:
[(None, 1)]	[(None, 1)]

fare	InputLayer
input:	output:
[(None, 1)]	[(None, 1)]

sex	InputLayer
input:	output:
[(None, 1)]	[(None, 1)]

class	InputLayer
input:	output:
[(None, 1)]	[(None, 1)]

deck	InputLayer
input:	output:
[(None, 1)]	[(None, 1)]

embark_town	InputLayer
input:	output:
[(None, 1)]	[(None, 1)]

alone	InputLayer
input:	output:
[(None, 1)]	[(None, 1)]

concatenate_8	Concatenate
input:	output:
[(None, 1), (None, 1), (None, 1), (None, 1)]	(None, 4)

normalization_3	Normalization
input:	output:
(None, 4)	(None, 4)

string_lookup	StringLookup
input:	output:
(None, 1)	(None, 1)

category_encoding	CategoryEncoding
input:	output:
(None, 1)	(None, 3)

string_lookup_1	StringLookup
input:	output:
(None, 1)	(None, 1)

category_encoding_1	CategoryEncoding
input:	output:
(None, 1)	(None, 4)

string_lookup_2	StringLookup
input:	output:
(None, 1)	(None, 1)

category_encoding_2	CategoryEncoding
input:	output:
(None, 1)	(None, 9)

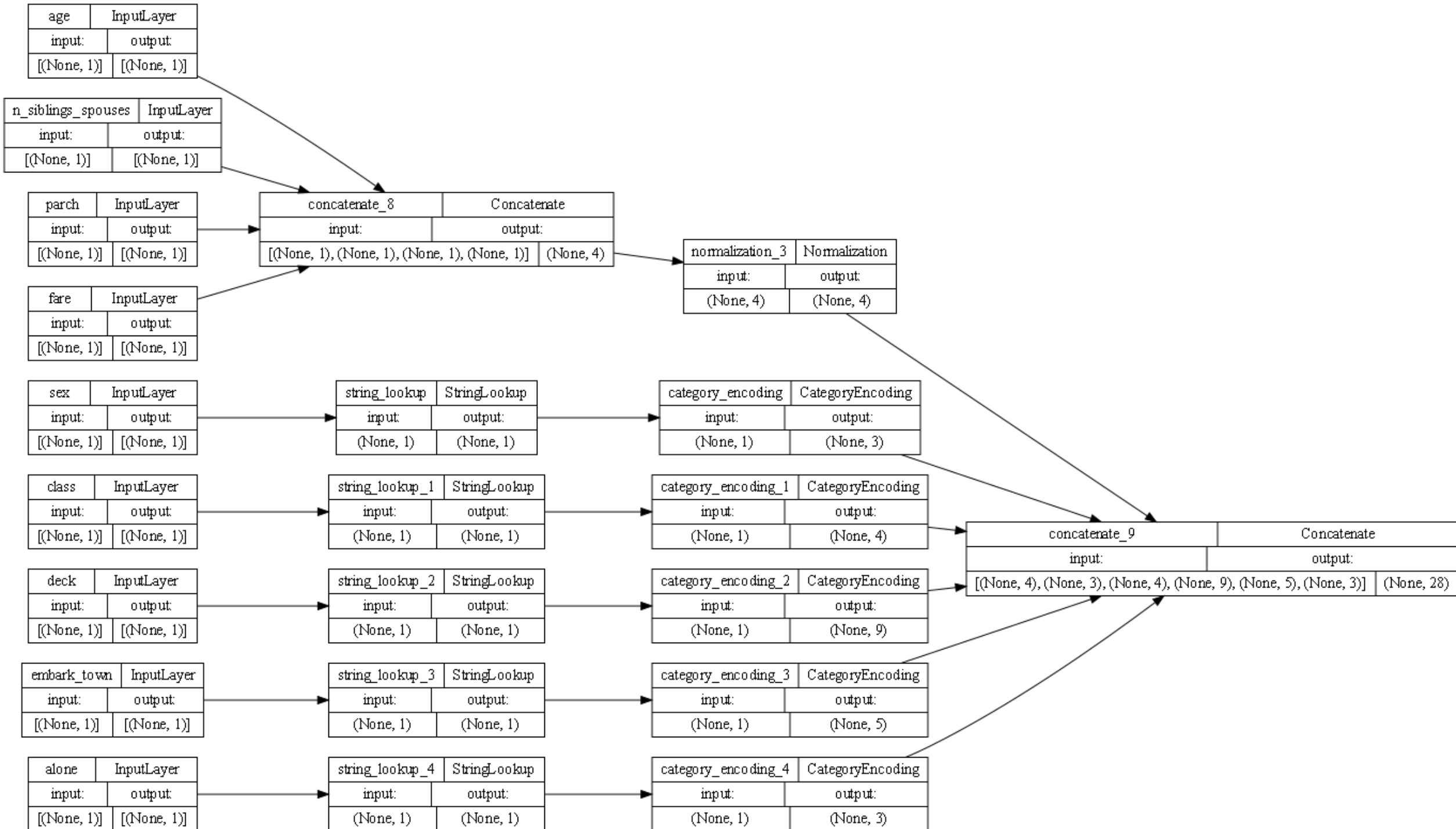
string_lookup_3	StringLookup
input:	output:
(None, 1)	(None, 1)

category_encoding_3	CategoryEncoding
input:	output:
(None, 1)	(None, 5)

string_lookup_4	StringLookup
input:	output:
(None, 1)	(None, 1)

category_encoding_4	CategoryEncoding
input:	output:
(None, 1)	(None, 3)

concatenate_9	Concatenate
input:	output:
[(None, 4), (None, 3), (None, 4), (None, 9), (None, 5), (None, 3)]	(None, 28)





## Gestructureerde data – methode 2 – niet volledig in Ram

- ▣ Maak gebruik van `tf.data.Dataset`
- ▣ Vanaf tensors of van file(s)



# Batching

- ▣ Batching

- Maak groepen van  $n$  elementen

- ▣ Repeat

- Herhaal de dataset  $x$  keer

- ▣ Shuffle

- Gooi de eerste  $n$  elementen door elkaar

- ▣ Volgorde van de functies is belangrijk!



# Preprocessing



## Beschikbare lagen

- ▣ [https://www.tensorflow.org/guide/keras/preprocessing\\_layers](https://www.tensorflow.org/guide/keras/preprocessing_layers)
- ▣ Tekstverwerking
  - TextVectorization -> string to tensor
- ▣ Getallen
  - Normalization of Discretization
- ▣ Categorieën
  - Category encoding, Hashing, StringLookup, IntegerLookup
- ▣ Images
  - Resizing, Rescaling, CenterCrop



## Adapt()

- ▣ Sommige van de lagen worden getrained in de .fit()
- ▣ Sommige niet en moeten voor de fit aangepast worden aan de trainingsdata met de adapt() functie voor de fit()
  - TextVectorization -> map tussen strings en integers
  - StringLookup en IntegerLookup -> map tussen inputs en integers
  - Normalization -> mean en standard deviation
  - Discretization -> bucket/bin boundaries
- ▣ Dit komt omdat alle data gekend moet zijn terwijl de fit batch per batch verwerkt

## Preprocessing voor het model

- ▣ Voer de taken uit op de `tf.data.Dataset`

```
# dit wordt typisch asynchroon uitgevoerd op de CPU en gebufferd en dan in batches doorgegeven aan de trainingsloop  
dataset = dataset.map(lambda x, y: (preprocessing_layer(x), y))  
# voeg deze lijn toe om de preprocessing in parallel met de training uit te voeren (optioneel)  
dataset = dataset.prefetch(tf.data.AUTOTUNE)  
model.fit(dataset, ...)
```

- ▣ Wanneer je traint op een TPU/GPU probeer zoveel mogelijk preprocessing in `tf.data` pipeline uit te voeren.

## Preprocessing als deel van het model

```
inputs = keras.Input(shape=input_shape)
x = preprocessing_layer(inputs)
outputs = rest_of_the_model(x)
model = keras.Model(inputs, outputs)
```

- ▣ Preprocessing synchroon met het model
- ▣ Wordt op hetzelfde toestel uitgevoerd als de training
  - Kan dus ook profiteren van de GPU indien aanwezig
  - De duidelijkste optie als alle data op dezelfde manier behandeld kan worden
- ▣ Verbeterd de flexibiliteit van je model
  - Wanneer je het model opslaat, worden ook de preprocessing lagen bewaart
  - Kleiner verschil tussen development en productie
  - Het model is gebruiksvriendelijker want kennis over de preprocessing is niet vereist





## Wat met verschillende soorten inputs

- ▣ Soorten inputs: bvb numerieke/categoriek?
- ▣ Maak gebruik van Functional API ipv Sequential model

# Normalization

```
# Load some data
(x_train, y_train), _ = keras.datasets.cifar10.load_data()
x_train = x_train.reshape((len(x_train), -1))
input_shape = x_train.shape[1:]
classes = 10

# Create a Normalization layer and set its internal state using the training data
normalizer = layers.Normalization()
normalizer.adapt(x_train)

# Create a model that include the normalization layer
inputs = keras.Input(shape=input_shape)
x = normalizer(inputs)
outputs = layers.Dense(classes, activation="softmax")(x)
model = keras.Model(inputs, outputs)

# Train the model
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy")
model.fit(x_train, y_train)
```

# Normalization

## ■ Tf.keras.layers.Normalization

- Werkt global, volledige data

```
# normalizer
normalizer = tf.keras.layers.Normalization()
normalizer.adapt(X_train_flat)
```

## ■ Speciale Soorten

- Source: <https://towardsdatascience.com/different-types-of-normalization-in-tensorflow-c>
- Batch
- Group
- Instance
- Layer
- Weight

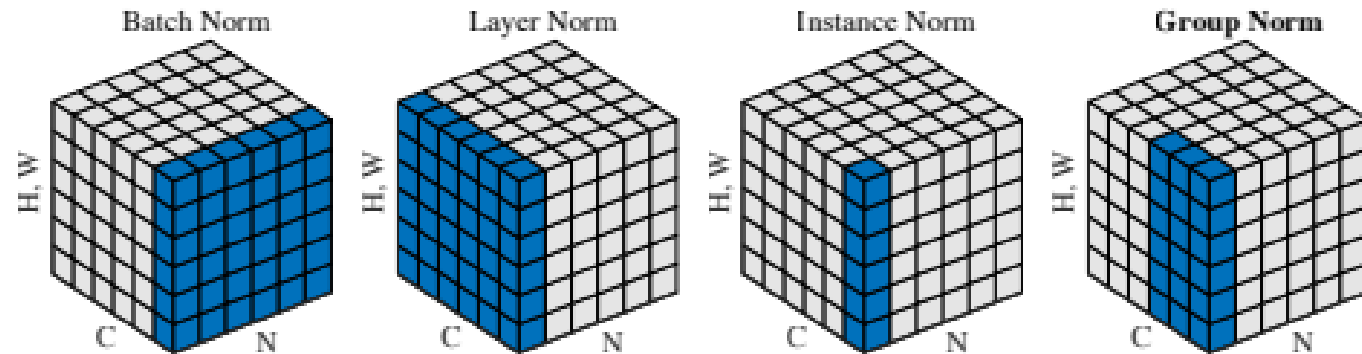


Figure 2. **Normalization methods.** Each subplot shows a feature map tensor, with  $N$  as the batch axis,  $C$  as the channel axis, and  $(H, W)$  as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

## Normalisation: Batch Normalisation

- ▣ Heel goed voor performantie
  - ▬ Reden nog niet volledig gekend
- ▣ Normalisatie binnen een batch van sample
- ▣ Werkt vooral goed bij grote batches
- ▣ Te gebruiken na elke laag met een activatiefunctie

```
#Batch Normalization  
model.add(tf.keras.layers.BatchNormalization())
```

## Normalisation: Group normalisation

- ▣ Vooral gebruikt bij computer visie
- ▣ Normalisation over groepen van kanalen ipv voorbeelden
  - ▬ Betere performantie bij kleine batches
- ▣ Werkt beter dan Layer en instance normalisation
  - ▬ Combineert de goede elementen van beide

```
#Group Normalization
model.add(tf.keras.layers.Conv2D(32, kernel_size=(3, 3),
activation='relu'))
model.add(tfa.layers.GroupNormalization(groups=8, axis=3))
```

## One-hot encoding

```
# Define some toy data
data = tf.constant([["a"], ["b"], ["c"], ["b"], ["c"], ["a"]])

# Use StringLookup to build an index of the feature values and encode output.
lookup = layers.StringLookup(output_mode="one_hot")
lookup.adapt(data)

# Convert new test data (which includes unknown feature values)
test_data = tf.constant([["a"], ["b"], ["c"], ["d"], ["e"], [""]])
encoded_data = lookup(test_data)
print(encoded_data)
```

```
# Define some toy data
data = tf.constant([[10], [20], [20], [10], [30], [0]])

# Use IntegerLookup to build an index of the feature values and encode output.
lookup = layers.IntegerLookup(output_mode="one_hot")
lookup.adapt(data)

# Convert new test data (which includes unknown feature values)
test_data = tf.constant([[10], [10], [20], [50], [60], [0]])
encoded_data = lookup(test_data)
print(encoded_data)
```

## One-hot encoding: hashing trick

- ▣ Woordenboek kan heel groot zijn (duizenden) maar veel woorden komen maar zelden voor
  - Inefficiënt om deze woorden een kolom toe te kennen
  - Hash de waarden naar een vector van een bepaalde grootte
  - Verkleint het aantal features en zorgt ervoor dat er geen expliciete indexing meer nodig is

```
# Sample data: 10,000 random integers with values between 0 and 100,000
data = np.random.randint(0, 100000, size=(10000, 1))

# Use the Hashing layer to hash the values to the range [0, 64]
hasher = layers.Hashing(num_bins=64, salt=1337)

# Use the CategoryEncoding layer to multi-hot encode the hashed values
encoder = layers.CategoryEncoding(num_tokens=64, output_mode="multi_hot")
encoded_data = encoder(hasher(data))
print(encoded_data.shape)
```

# Ngrams en multi-hot

## ▣ N-gram

- Een sequentie van N opeenvolgende woorden

## ▣ Multi-hot encoding

- In een n-gram staan meer woorden
- Deze staan op 1 bij multi-hot

```
# Define some text data to adapt the layer
adapt_data = tf.constant(
    [
        "The Brain is wider than the Sky",
        "For put them side by side",
        "The one the other will contain",
        "With ease and You beside",
    ]
)

# Instantiate TextVectorization with "multi_hot" output_mode
# and ngrams=2 (index all bigrams)
text_vectorizer = layers.TextVectorization(output_mode="multi_hot", ngrams=2)
# Index the bigrams via `adapt()`
text_vectorizer.adapt(adapt_data)

# Try out the layer
print(
    "Encoded text:\n", text_vectorizer(["The Brain is deeper than the sea"]).numpy(),
)
```





## Tf-idf

- ▣ Term frequency – inverse document frequency

```
# Instantiate TextVectorization with "tf-idf" output_mode  
# (multi-hot with TF-IDF weighting) and ngrams=2 (index all bigrams)  
text_vectorizer = layers.TextVectorization(output_mode="tf-idf", ngrams=2)  
# Index the bigrams and learn the TF-IDF weights via `adapt()`
```

### ▣ Term frequency – inverse document frequency

- ▬ Geef een lager gewicht aan termen die in heel veel rijen, entries, teksten voorkomen
- ▬ Woorden die in slechts een aantal documenten of teksten voorkomen krijgen een hoger gewicht
  - “Nigerian prince” is belangrijker om spam te detecteren dan “Kind regards”

```
# Instantiate TextVectorization with "tf-idf" output_mode
# (multi-hot with TF-IDF weighting) and ngrams=2 (index all bigrams)
text_vectorizer = layers.TextVectorization(output_mode="tf-idf", ngrams=2)
# Index the bigrams and learn the TF-IDF weights via `adapt()`
```



## Werken met meerdere in/outputs

▣ Lees en bestudeer de code op deze link:

- [https://www.tensorflow.org/guide/keras/functional#manipulate\\_complex\\_graph\\_to\\_topologies](https://www.tensorflow.org/guide/keras/functional#manipulate_complex_graph_to_topologies)



# Neural networks





## 2 manieren voor modellen op te bouwen

### ▣ Sequentieel model

- ▬ Maak een lijst met de te volgen stappen/lagen
- ▬ Eenvoudigere architecturen, alle data naar de volgende stap

### ▣ Functional API

- ▬ Lagen in het netwerk zijn functies
- ▬ Complexere architecturen mogelijk
  - ▣ Meerdere inputs/outputs, gedeelde lagen, niet-lineaire topologie, ...
- ▬ Architectuur moet nog steeds een gerichte acyclische graaf zijn

## Sequentieel model

```
# sequentieel model
from tensorflow.keras import models

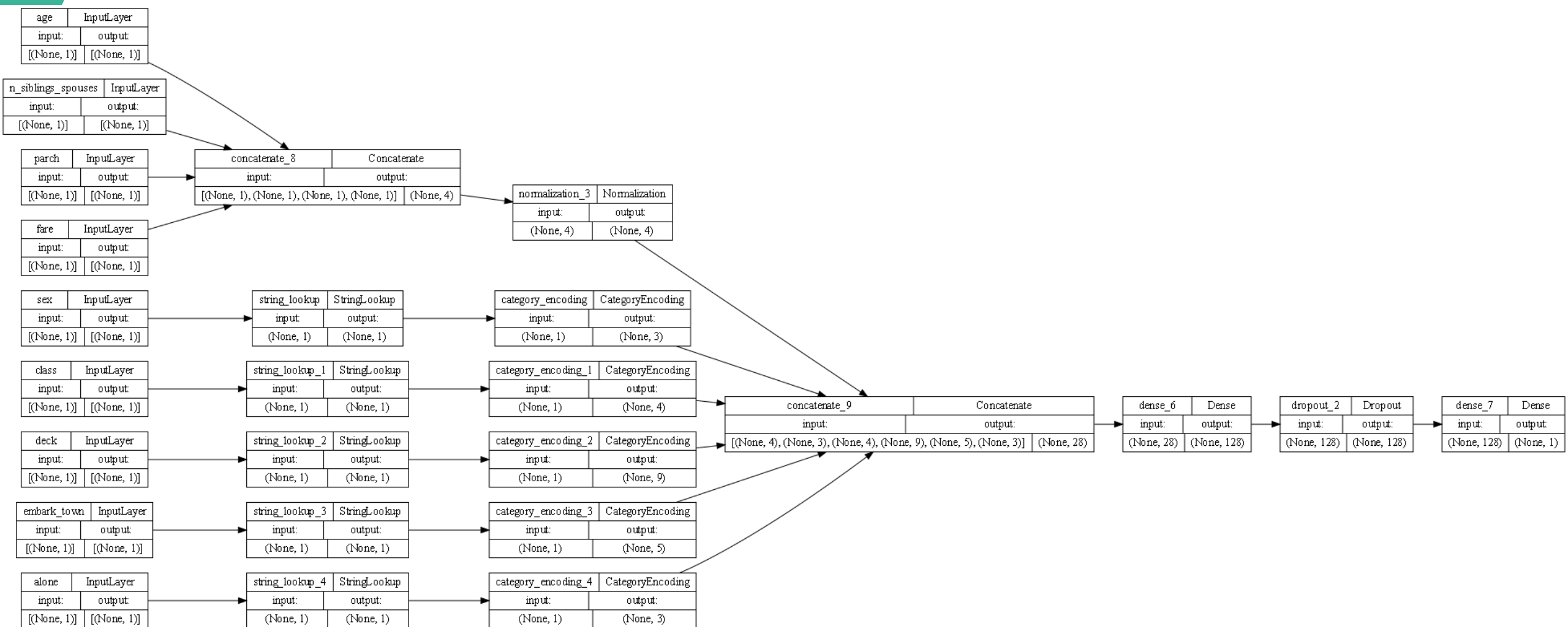
model = models.Sequential([
    layers.Dense(128, activation='relu', input_shape=(None, 28)),
    layers.Dropout(0.2),
    layers.Dense(1, activation='sigmoid')
])
```



## Functional API

```
x = layers.Dense(128, activation='relu')(preprocessed_inputs_cat)
x = layers.Dropout(0.2)(x)
x = layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(inputs, x)
```







# Training






## Benodigde stappen

### ▣ Compileer het model

- Loss-functie: `tf.keras.losses`
- Optimizer (learning rate): `tf.keras.optimizers`
- Metrics: extra te berekenen metrieken voor evaluatie (`tf.keras.metrics`)

### ▣ Train het

- Definieer de in- en outputs van de trainingsdata
- `Batch_size`
- Epochs
- Validation split



```
# model trainen
model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(),
    optimizer=tf.keras.optimizers.RMSprop(),
    metrics=[tf.keras.metrics.Accuracy()],
)

model.summary()

history = model.fit(x=titanic_features_dict, y=titanic_labels, batch_size=32, epochs=20)
```



# Loss functions

## ▣ Regression

- MeanSquaredError
- MeanSquared Percentage Error
- MeanAbsoluteError
- MeanAbsolutePercentageError



# Loss functions

## ■ Classification

- ▬ `from_logits=true`
  - Bij niet gebruik van softmax activatiefunctie
- ▬ `BinaryCrossEntropy`:
  - `y_true` is 0/1
  - `y_pred` is 1 integer
  - Ook voor multi-label (elke output is een binair probleem)
- ▬ `CategoricalCrossEntropy`:
  - `y_true` is one-hot encoding
  - `y_pred` is tensor of size `num_classes` (probability per class)



# Loss functies

## ▣ Classification

- SparseCategoricalCrossEntropy
  - `y_true`: integer met de klasse
  - `y_pred`: tensor met de kans van elke klasse



# Evaluation



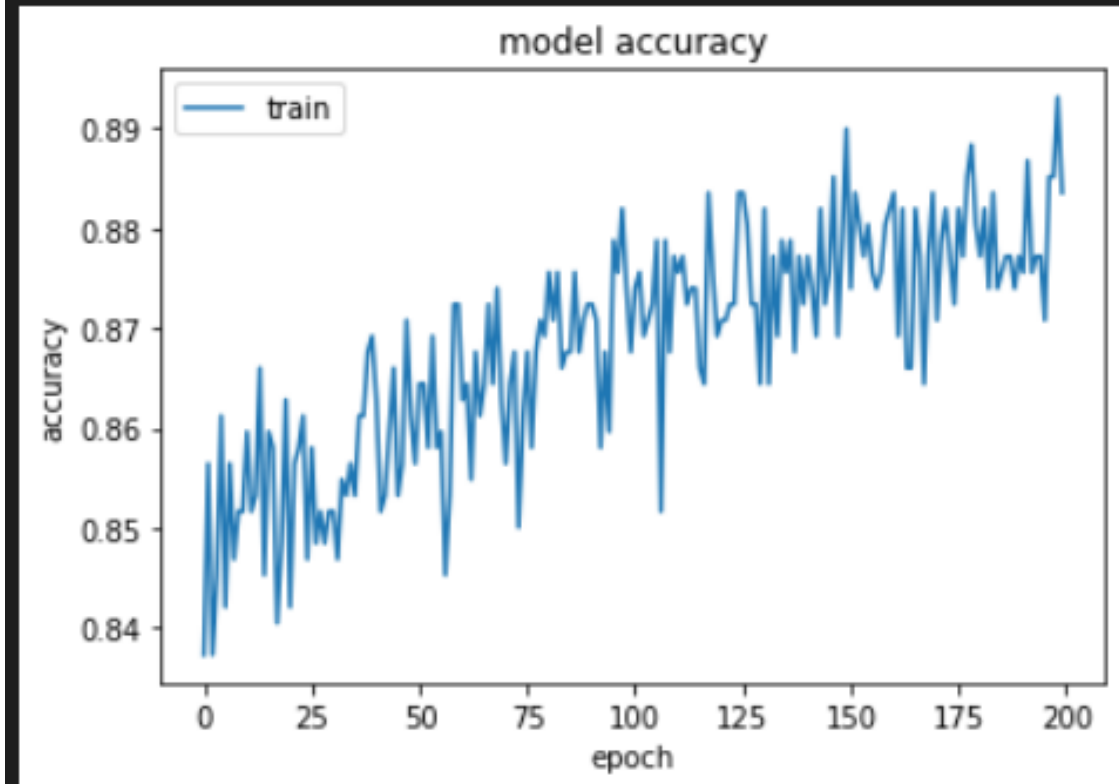
# Evaluatie

- ▣ Via grafiek

```
# plot histogram
import matplotlib.pyplot as plt

plt.plot(history.history['binary_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```

✓ 0.0s





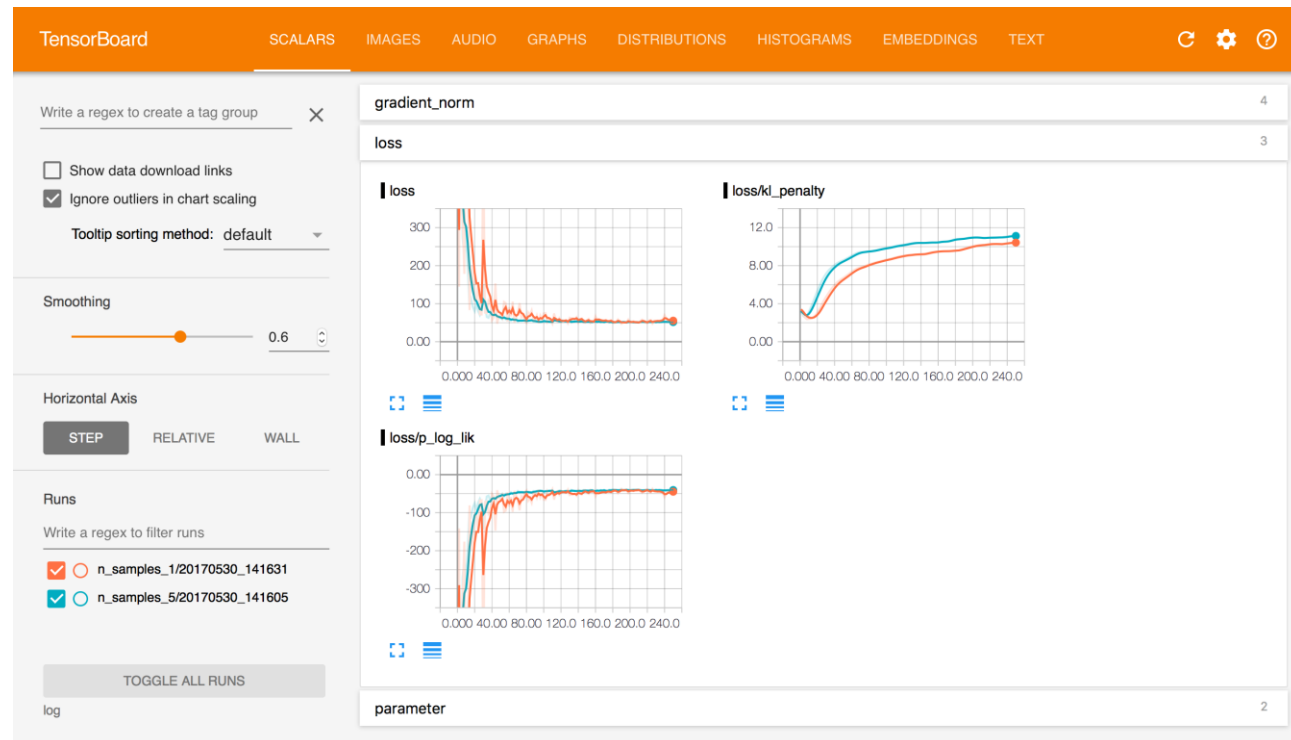


## Evaluatie

- ▣ Via histogram
- ▣ Via de `model.evaluate()` functie voor testdata

# Evaluatie

- ▣ Via histogram
- ▣ Via de `model.evaluate()` functie voor testdata
- ▣ Tensorboard





# Callbacks





## Callbacks

- ▣ Toegevoegd tijdens het leerproces
- ▣ Allerhande functionaliteiten
  - Preloading of data: voor efficiëntie
  - Early termination: voor vermijden overfitting
  - Checkpoints: backups bijhouden in het geval van error

## Callbacks – Tensorboard voorbeeld

```
# model trainen
callbacks = [
    tf.keras.callbacks.TensorBoard(
        log_dir="./logs",
        histogram_freq=1,          # How often to log histogram visualizationss
        update_freq="epoch",
        profile_batch=10
    )
]

history = model.fit(x=titanic_features_dict, y=titanic_labels, callbacks=callbacks, batch_size=32, epochs=2000)
```



# Huiswerk





## Huiswerk

- ▣ Bestudeer de code uit de les
- ▣ Bestudeer de volgende tutorial
  - [https://www.tensorflow.org/guide/keras/training\\_with\\_built\\_in\\_methods](https://www.tensorflow.org/guide/keras/training_with_built_in_methods)
  - Meer info, meer code-voorbeelden
- ▣ Maak de oefening over Neurale netwerken:
  - Link op Toledo
  - Op punten
  - Deadline volgende week zondag