Introduction to Machine Learning

Module 4: Deep Learning

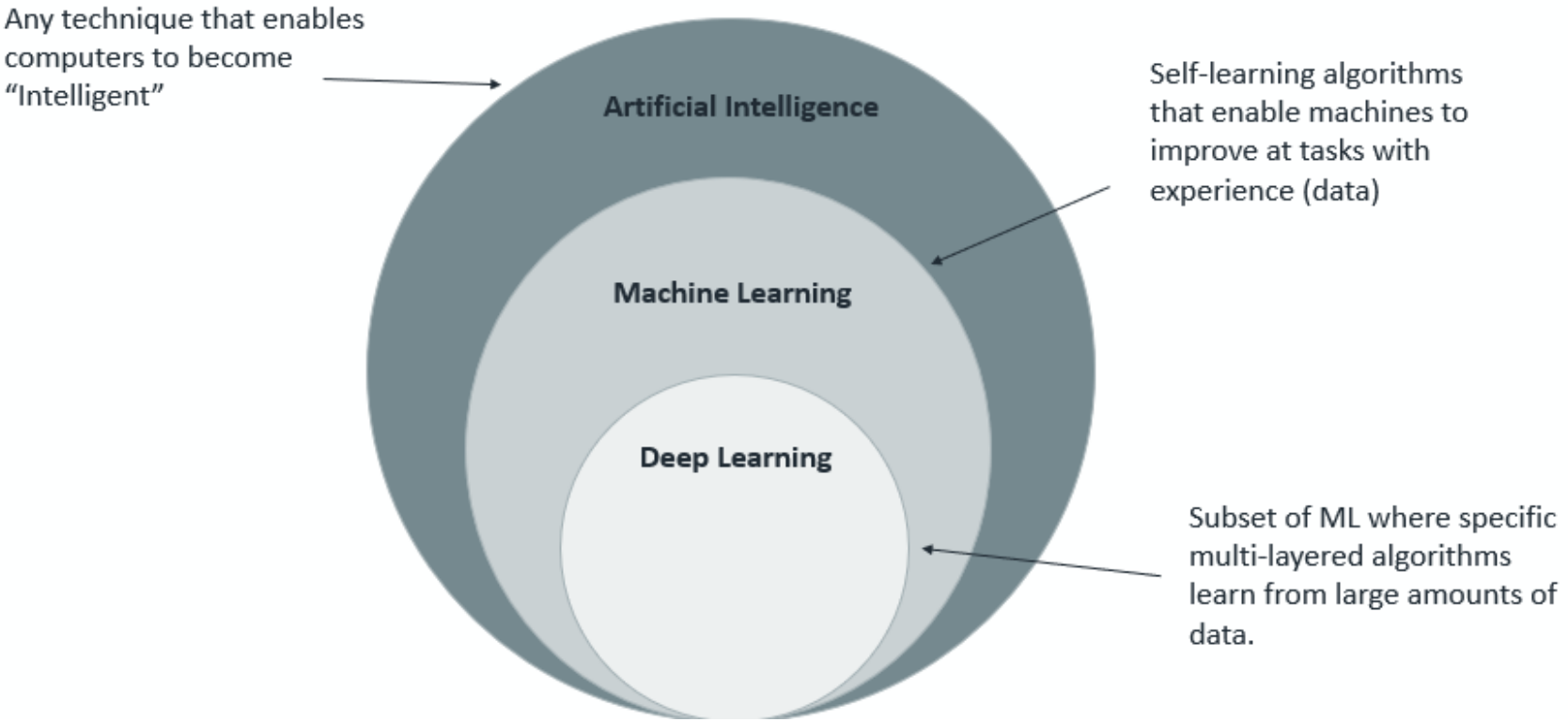
Tobias Rebholz

University of Tübingen

Fall 2024, SMiP Workshop

Deep Learning

Deep Learning



(https://towardsdatascience.com/redefining-data-science-d7f2026a021a)

Deep Learning

- Deep Learning (DL) is a sub-field of Machine Learning
  - Using multi-layered or "deep" neural networks (NNs) for classification, regression, and other data-driven decision-making tasks
- Most of the world's data is held in unstructured formats (e.g., text, images)
  - NN: Can process unstructured data quite well
  - Classical ML: Usually leverages structured data for predictions
- Removing some of the dependency on human experts
  - NN: Automates feature extraction
    - E.g., determining which features (e.g., shape of mouth, eyes, . . .) are most important in distinguishing emotional expressions from one another in a set of photos of different people
  - Classical ML: Feature hierarchies must be established manually
- NN: Can adjust and fit itself to improve the out-of-sample prediction performance
  - Through gradient descent and backpropagation (not discussed; see excurse for some ideas/concepts)

Deep Learning

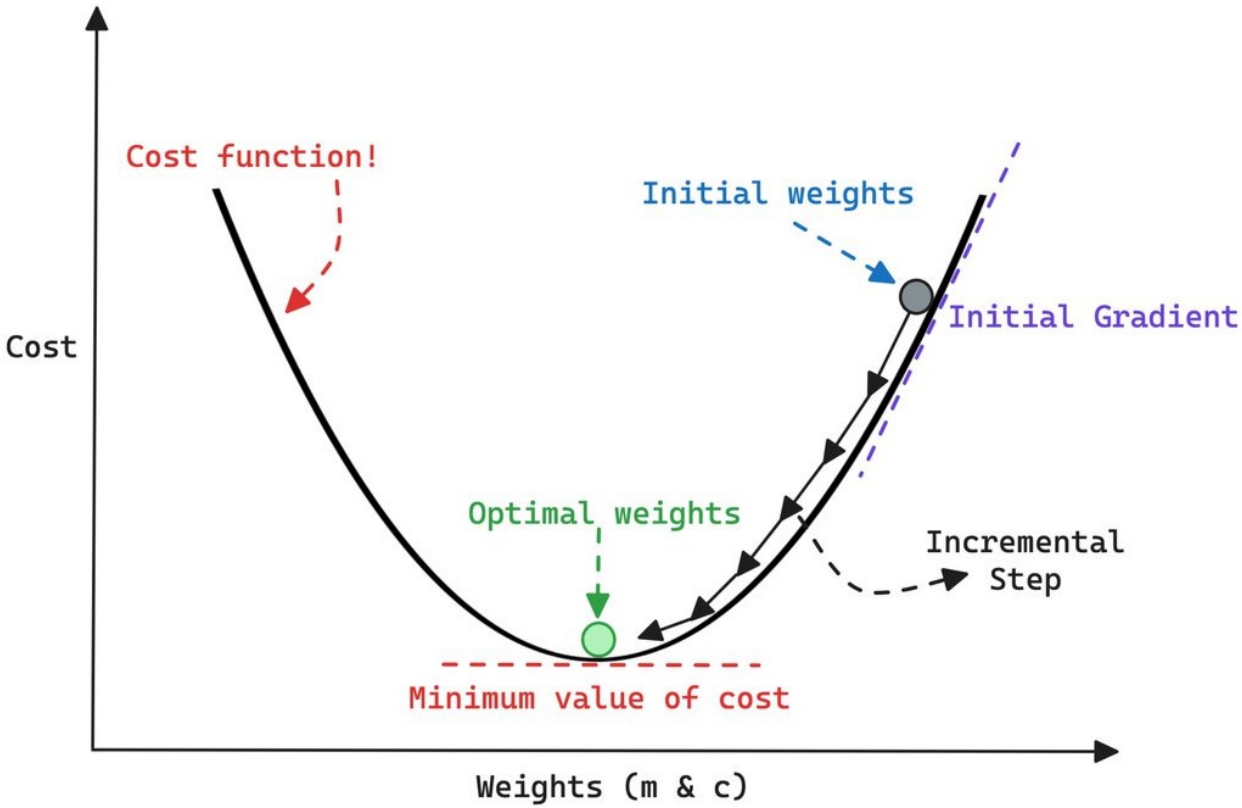
- Good performance: Usually requires intensive knowledge, time, and (computational) resources to train the models
  - There is a whole industry trying to make money out of training and benchmarking DL models
  - In other words, we cannot afford these resources!
- Instead of training the models ourselves, we focus on using pre-trained models to solve our own prediction problems in this module
  - Costs: We give up control over what the (pre-)trained model can actually do
    - Thus inevitably also some understanding of how these competences are achieved
  - E.g., the Hugging Face community provides access to many pre-trained models and datasets: https://huggingface.co/

Excurse: Optimization Algorithm

- Unlike classical statistical models, such as regression, NNs do not rely on traditional estimation techniques like maximum likelihood to estimate unknown model parameters
  - Instead, a stepwise optimization technique is used to iteratively improve the weights θ according to a cost or loss function $L(\theta)$
    - Small L: Good fit to the training data
- To better understand the problem of finding the weights, let's pretend we do not have a formula to estimate the coefficients (i.e., "weights") of a simple linear regression model $y = c + m \cdot x + \varepsilon$
  - Where: $\theta = (c, m)$

Excurse: Optimization Algorithm

- Prediction error's dependence on the weights:



(adapted from https://x.com/akshay_pachaar/status/1714619920955346995)

Excurse: Optimization Algorithm

- Gradient: A vector that points in the direction of the steepest ascent
  - For minimizing a cost function: Move in the opposite direction (i.e., steepest descent)
  - Think of rolling a ball downhill towards the lowest point in a valley
- Gradient descent algorithm: General, iterative solver to minimize the cost function (e.g., MSE in regression)
  1. Start with arbitrary initial values for the weights θ (e.g., regression coefficients)
  2. Compute the slope ("gradient") of the cost function at the current position
  3. Update the weights θ by moving in the direction opposite to the gradient (i.e., downhill) with a specified step size ("learning rate")
  4. Repeat until the change in the cost function becomes negligible ("convergence")

Excurse: Backpropagation

- Purpose: Efficiently computing the gradients of the cost function with respect to each parameter in the NN
  - Forward pass: Compute the predictions of the NN and evaluate the cost function based on the actual outputs
  - Backward pass: Use the chain rule to propagate the error backward through the layers of the NN, calculating the gradients of the cost with respect to each parameter
- I.e., gradient descent can be thought of the "outer loop" optimization process, while backpropagation is the "inner loop" method to compute the required gradients

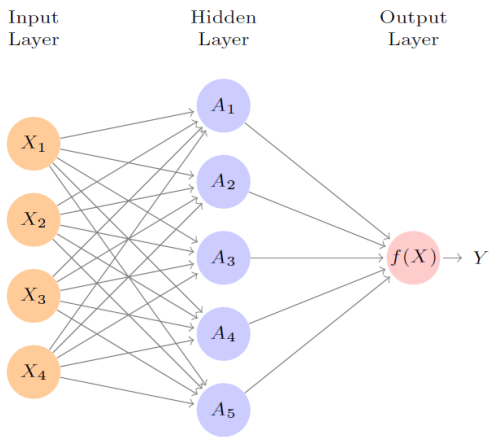
Neural Networks

Neural Networks

- State-of-the-art DL technology: Neural Networks (NNs)
  - Trained on massive amounts of data to identify and classify phenomena, recognize patterns and relationships, evaluate possibilities, make predictions and decisions, ...
  - Require a lot of tinkering, whereas newer methods (e.g., SVM, RF) are more automatic
- On many problems, the newer methods outperform **poorly-trained** NNs
  - But: NNs are successful especially on some **niche problems**, such as:
    - Image recognition (first half of this module)
    - Video classification
    - Speech and text modeling (second half of this module)
    - ...

Single Hidden Layer

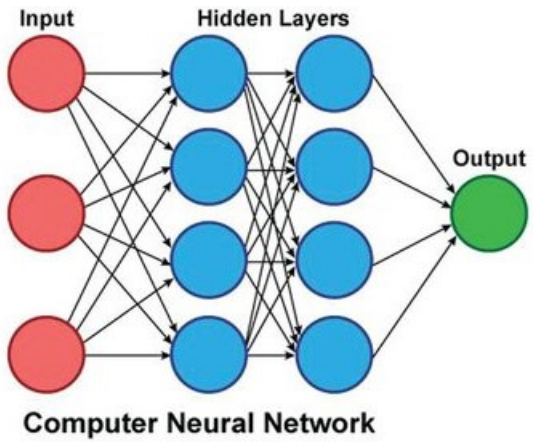
- The hidden layer computes so-called "activations" $A_k = h_k(X)$, i.e., (nonlinear) transformations of linear combinations of the vector of input features $X = (x_1, x_2, \ldots, x_p)$
  - "Hidden" because the activations A_k are not directly observed
  - The functions $h_k(\cdot)$ for transforming the observed features are not fixed in advance, but learned during the training of the NN
- The output layer is a **linear model** that uses the activations A_k as inputs, resulting in a function $f(X)$ to make predictions (cf. regression)



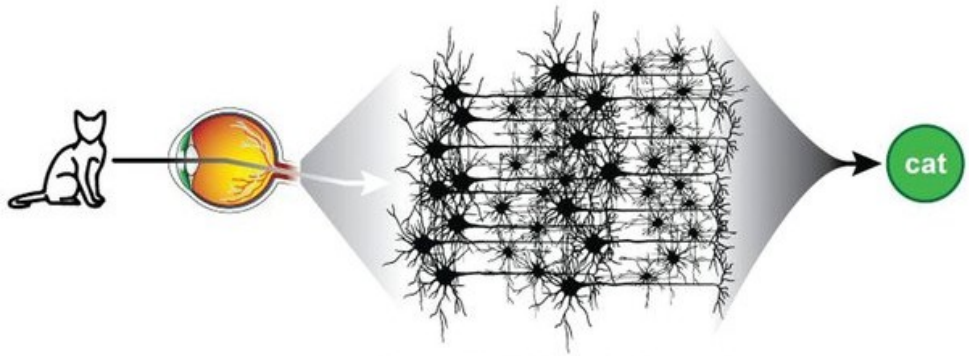
(James et al., 2021, Figure 10.1)

Multiple Hidden Layers

- Inspired by the functioning of brains:
  - Activations A_k: Neurons receive signals (inputs) from other neurons, process them, and send signals to other neurons
  - Learning of $h_k(\cdot)$: This process is akin to brains' ability to form new connections and strengthen existing ones based on experiences
  - Output: The brain processes information from neurons (i.e., activations) and generates responses (i.e., predictions)



Computer Neural Network

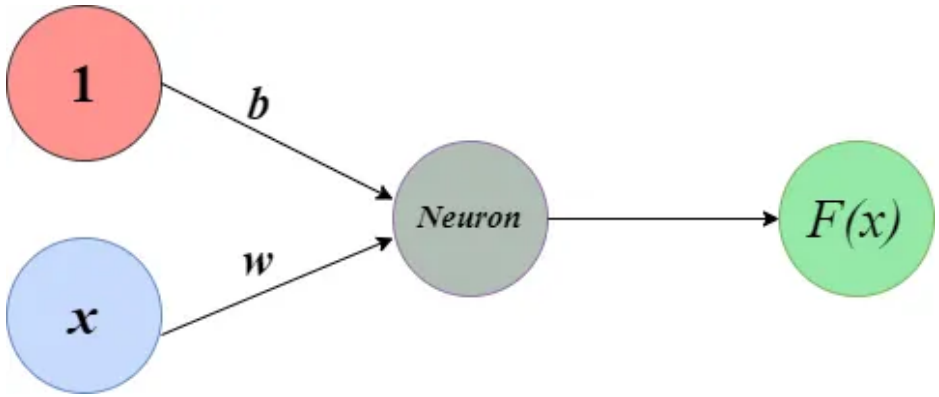


Biological Neural Network

(adapted from
https://www.quora.com/How-is-AI-similar-different-to-the-human-brain)

Single Hidden Layer

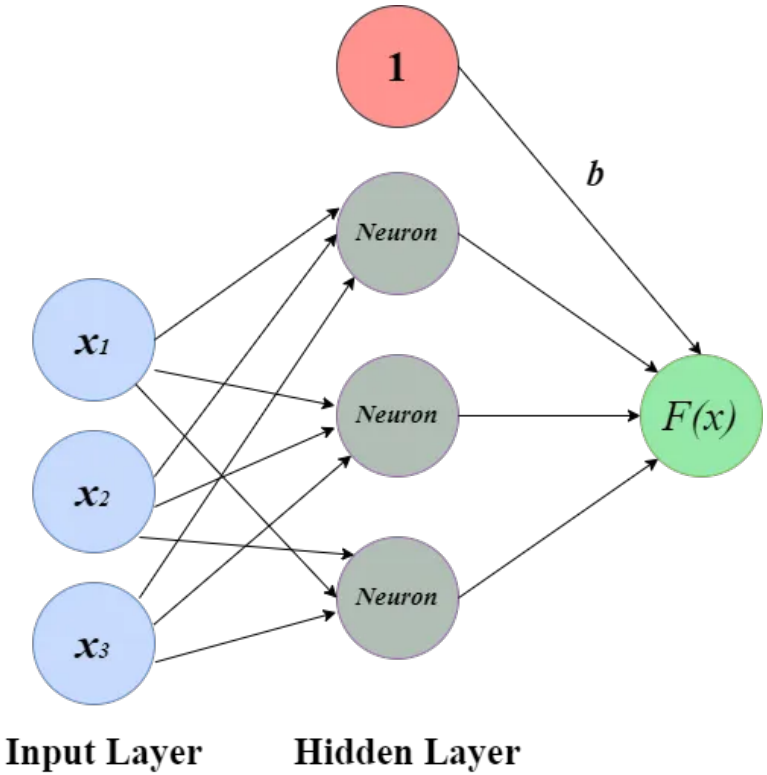
- At their core, NNs extend linear regression
  - A NN with a single hidden layer and a linear activation function reduces to linear regression: $f(x) = w \cdot x + b$
    - where: x – feature **vector**; w – weight; b – bias
  - Nonlinear activation functions: Enable the NN to learn any functional form (i.e., more complex patterns)



(https://medium.com/@asifurrahmanaust/lesson-3-neural-network-is-nothing-but-a-linear-regression-e05a328a0f23)

Single Hidden Layer

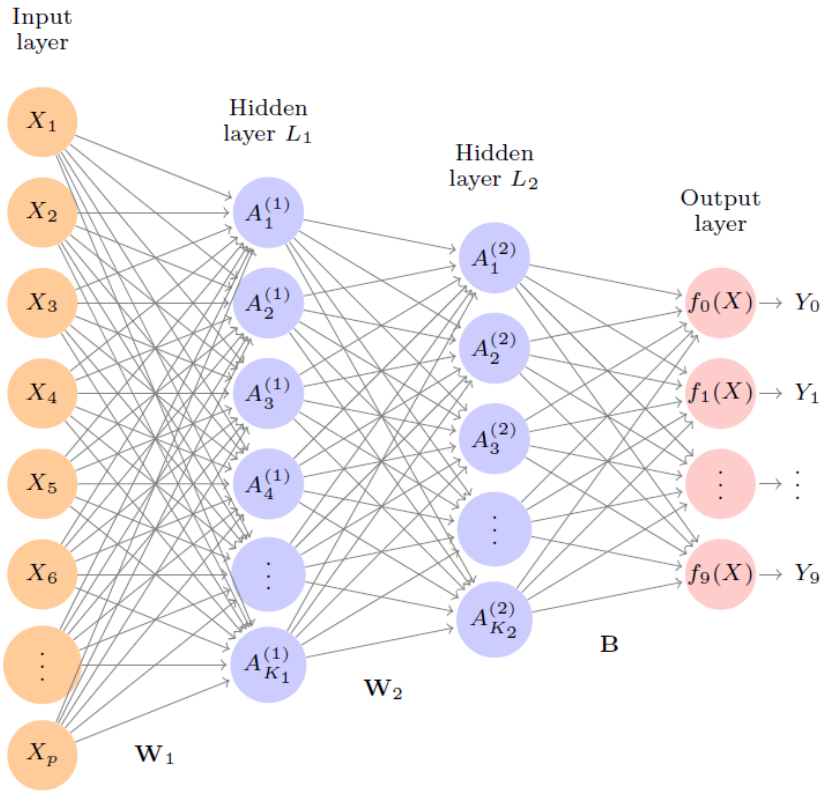
- Multiple regression: $f(X) = w^T X + b$
  - where: X – feature **matrix**; w – weight **vector**; b – bias



(https://medium.com/@asifurrahmanaust/lesson-3-neural-network-is-nothing-but-a-linear-regression-e05a328a0f23)

Multiple Hidden Layers

- Complex NN with multiple hidden layers and outputs:



(James et al., 2021, Figure 10.4)

Excurse: Most Important Types of NNs

- **Feedforward NNs:** Simplest type of NN, where information travels in one direction only, from the input layer, through any hidden layers, to the output layer
  - Widely used in pattern recognition and standard classification tasks
- **Convolutional NNs:** Designed to automatically and adaptively learn spatial hierarchies of features from the input (our focus here!)
  - Primarily used for image processing, classification, and segmentation
- **Recurrent NNs:** Have "memory" in the form of hidden state vectors that capture information about what has been computed so far
  - Mainly used for sequential data tasks, such as language modeling and time series prediction (more on this later!)
- **Autoencoders (AEs):** Encoding the input into a latent-space representation, and then decoding the latent representation back to the original input
  - Great for feature extraction, dimensionality reduction, compression, ...

Excurse: Hyperparameters

- **Activation functions** (e.g., ReLU, Sigmoid, Tanh)
  - Matching the type of variable that is predicted (e.g., binary or numerical)
  - Similar to choosing between linear and logistic regression
- **Number of layers:** The depth of the network (incl. input, hidden, and output layers)
- **Learning rate** of optimizer: Step size for updating weights during training
  - Too large: May overshoot the minimum or fail to converge
  - Too small: Convergence will be slow
- **Batch sizes:** Weights are adjusted after processing mini-batches of the training sample
  - Larger batches: More accurate steps towards minimizing the cost function, but require more computational resources
- **Epochs:** Training involves multiple passes through the training sample
  - Each epoch involves processing all mini-batches
- ...

Single Hidden Layer in `mlr3`

- Fitting a very simple NN in `mlr3` to predict (categorical!) educational attainment using the Big Five and some demographic features:

```
set.seed(1)
tsk <- as_task_classif(education ~ ., data = dat %>% select(-CASE))
mdl <- lrn('classif.nnet', size = 1)
mdl$train(tsk)
## # weights:  18
## initial  value 166.191340
## iter  10 value 143.221614
## iter  10 value 143.221614
## iter  10 value 143.221614
## final  value 143.221614
## converged
```

Single Hidden Layer in `mlr3`

- Model summary:

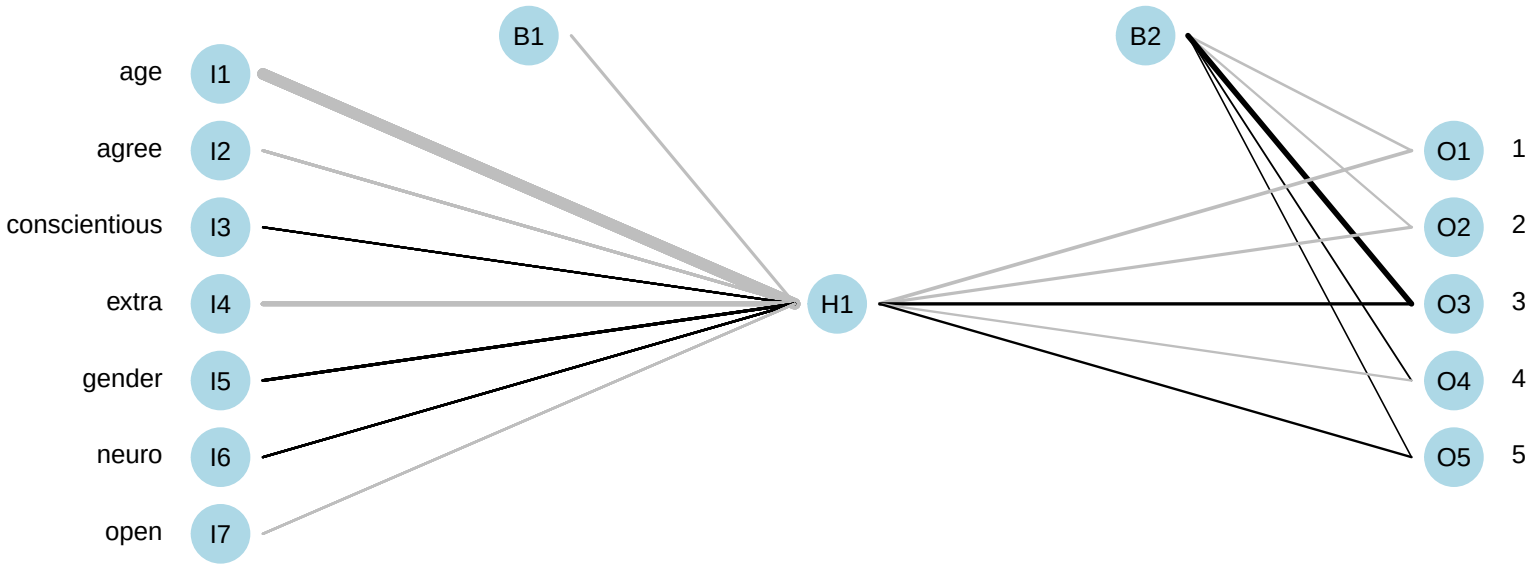
```
summary(mdl$model)
## a 7-1-5 network with 18 weights
## options were - softmax modelling
##  b->h1 i1->h1 i2->h1 i3->h1 i4->h1 i5->h1 i6->h1 i7->h1
##  -0.44  -2.76  -0.37   0.14  -0.85   0.37   0.20  -0.22
##  b->o1 h1->o1
##  -0.30  -0.60
##  b->o2 h1->o2
##  -0.21  -0.46
##  b->o3 h1->o3
##   1.13   0.50
##  b->o4 h1->o4
##   0.08  -0.22
##  b->o5 h1->o5
##   0.01   0.23

# cost:
mdl$model$value
## [1] 143.2216
```

Single Hidden Layer in `mlr3`

- Model visualization:
  - Line color: positive weights = black; negative weights = grey
  - Line thickness: Proportion to relative magnitude of each weight

```
plotnet(mdl$model)
```



Single Hidden Layer in `mlr3`

- (In-sample) Predictions:

```
pred = mdl$predict(tsk)
head(cbind('true' = dat$education, 'pred' = pred$response))
##      true pred
## [1,]    3    3
## [2,]    4    3
## [3,]    4    3
## [4,]    1    3
## [5,]    2    3
## [6,]    3    3

pred$confusion
##          truth
## response  1  2  3  4  5
##        1  0  0  0  0  0
##        2  0  0  0  0  0
##        3 11 12 46 16 15
##        4  0  0  0  0  0
##        5  0  0  0  0  0
```

Multiple Hidden Layers in `mlr3`

- We can do (much) better than that by using **more hidden layers**:

```
set.seed(1)
mdl2 <- lrn('classif.nnet', size = 5, maxit = 1e4)
mdl2$train(tsk)
## # weights:  70
## initial  value 199.194103
## iter  10 value 142.545638
## iter  20 value 133.436152
## iter  30 value 127.021710
## iter  40 value 114.464252
## iter  50 value 109.068810
## iter  60 value 107.265054
## iter  70 value 107.108533
## iter  80 value 102.869371
## iter  90 value 86.850234
## iter 100 value 81.793467
## iter 110 value 80.157844
## iter 120 value 79.745734
## iter 130 value 79.479151
## iter 140 value 79.463476
## iter 150 value 79.435179
## iter 160 value 79.378645
## iter 170 value 79.345396
## iter 180 value 79.343959
## iter 190 value 79.342064
```

Multiple Hidden Layers in `mlr3`

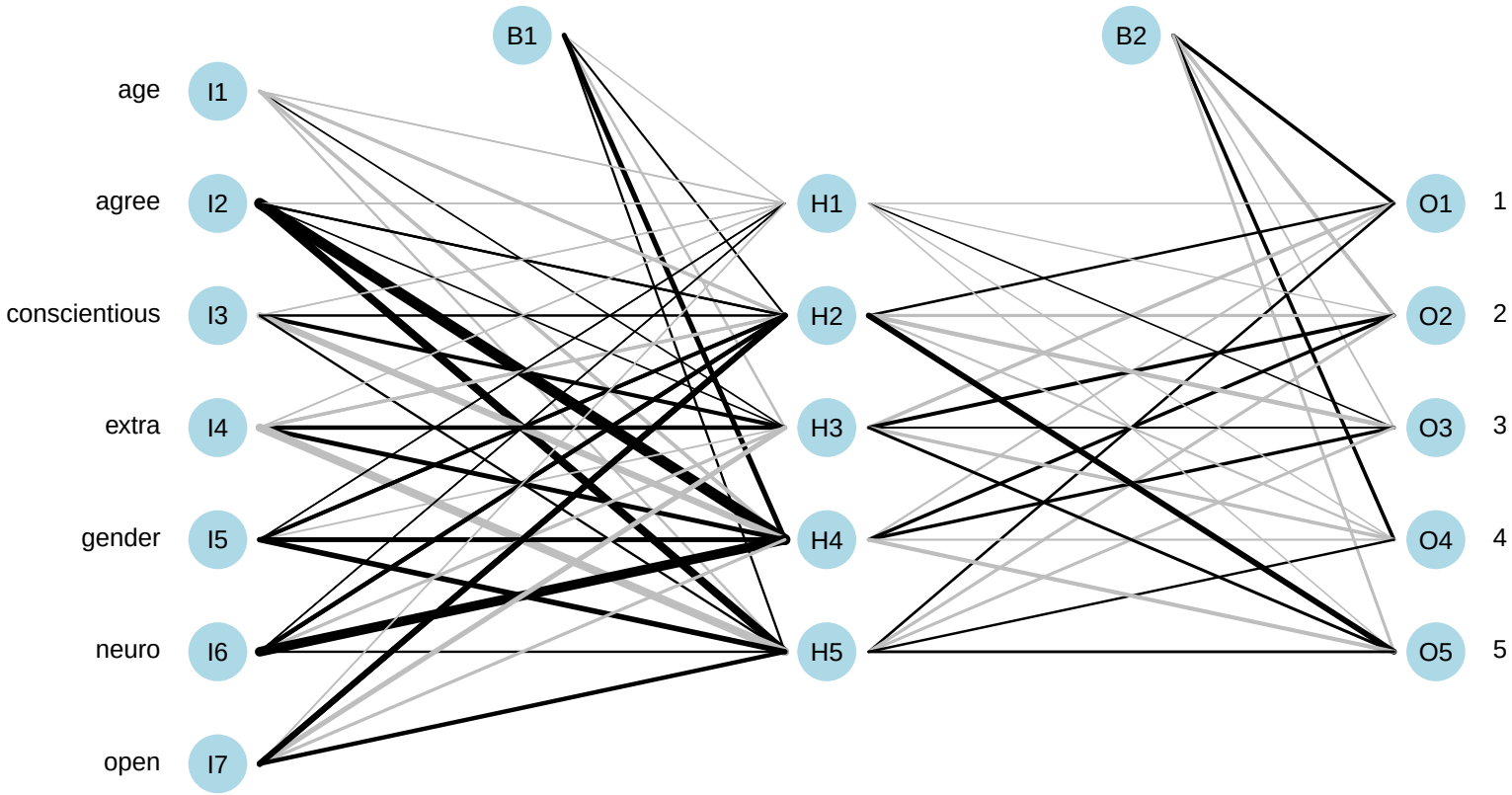
- Model summary: Much more complex model structure

```
summary(mdl2$model)
## a 7-5-5 network with 70 weights
## options were - softmax modelling
##   b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1
##   -0.48   -4.01   -0.53   -0.03   -1.00    0.30    0.13   -0.36
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2
##   29.82  -99.90   45.89   22.83  -97.03  127.21  157.62  250.66
##   b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3
##  -55.45    8.62    6.63  140.09  155.00  -18.31  -98.85 -202.79
##   b->h4  i1->h4  i2->h4  i3->h4  i4->h4  i5->h4  i6->h4  i7->h4
##  203.17 -127.31  535.47 -288.35  183.29  182.73  453.91  -94.21
##   b->h5  i1->h5  i2->h5  i3->h5  i4->h5  i5->h5  i6->h5  i7->h5
##   39.89  -52.73  365.17   44.74 -376.20  221.59   41.09  155.78
##   b->o1  h1->o1  h2->o1  h3->o1  h4->o1  h5->o1
##  117.95   -0.46   58.95 -118.95  -40.94   59.78
##   b->o2  h1->o2  h2->o2  h3->o2  h4->o2  h5->o2
## -131.00   -0.17  -90.79  130.70  111.13  -92.30
##   b->o3  h1->o3  h2->o3  h3->o3  h4->o3  h5->o3
##  -20.60    1.15 -167.74   21.16  104.07  -82.69
##   b->o4  h1->o4  h2->o4  h3->o4  h4->o4  h5->o4
##  115.01   -0.28  -40.96 -114.54  -30.84   48.25
##   b->o5  h1->o5  h2->o5  h3->o5  h4->o5  h5->o5
##  -81.23   -0.90  241.21   81.50 -144.31   68.18
```

Multiple Hidden Layers in `mlr3`

- Model visualization:

```
plotnet(mdl2$model)
```



Multiple Hidden Layers in `mlr3`

- (In-sample) Predictions: No longer predicting just the majority class

```
pred2 = mdl2$predict(tsk)
head(cbind('true' = dat$education, 'pred' = pred2$response))
##      true pred
## [1,]    3    3
## [2,]    4    3
## [3,]    4    4
## [4,]    1    1
## [5,]    2    3
## [6,]    3    3

pred2$confusion
##          truth
## response  1  2  3  4  5
##        1  7  0  0  1  0
##        2  0  5  0  0  0
##        3  4  7 45 12 10
##        4  0  0  1  3  0
##        5  0  0  0  0  5
```

Important Programming Notes

- The NNs trained on the previous slides were entirely for pedagogical purposes
  - I.e., they are not state-of-the-art!
- For technical reasons, powerful DL is tricky (if not impossible) in R, and thus established in Python, and typically requires specialized hardware
  - Therefore, many R packages actually use Python code in the background
  - Implication: DL tasks that are easy to implement in Python can be a bit tricky to get running in R
- `mlr3keras`: The DL package from the `mlr3verse` is still in a very early stage and under active development
  - Therefore, we will use the `reticulate` R-package as an interface to Python in order to access state-of-the-art DL technology (e.g., `tf-keras` for NNs, `transformers` for NLP)

Important Programming Notes

- Preparation in R:
  - This setup chunk only needs to be run once and takes a while until finished

```
if (!('reticulate' %in% installed.packages())) {
  # Python:
  install.packages("reticulate")
  library(reticulate)
  install_miniconda()

  # NNs:
  reticulate::py_install('torch', pip=T)
  reticulate::py_install('tf-keras', pip=T)

  # Transformers:
  reticulate::py_install('transformers', pip=T)

  # Hugging Face:
  reticulate::py_install('datasets', pip=T)
}
```

Image Recognition

Image Recognition

- Image recognition: A classification task where the goal is to assign a label to an image
  - Feature **matrix**: The pixel values that make up an image (incl. color, if applicable)
  - Target: The object shown in the image
    - Note: The target "object" can also be an entire scenery, specific situations or actions, ...
- Performance measurement: Usual metrics (i.e., classification error/accuracy)

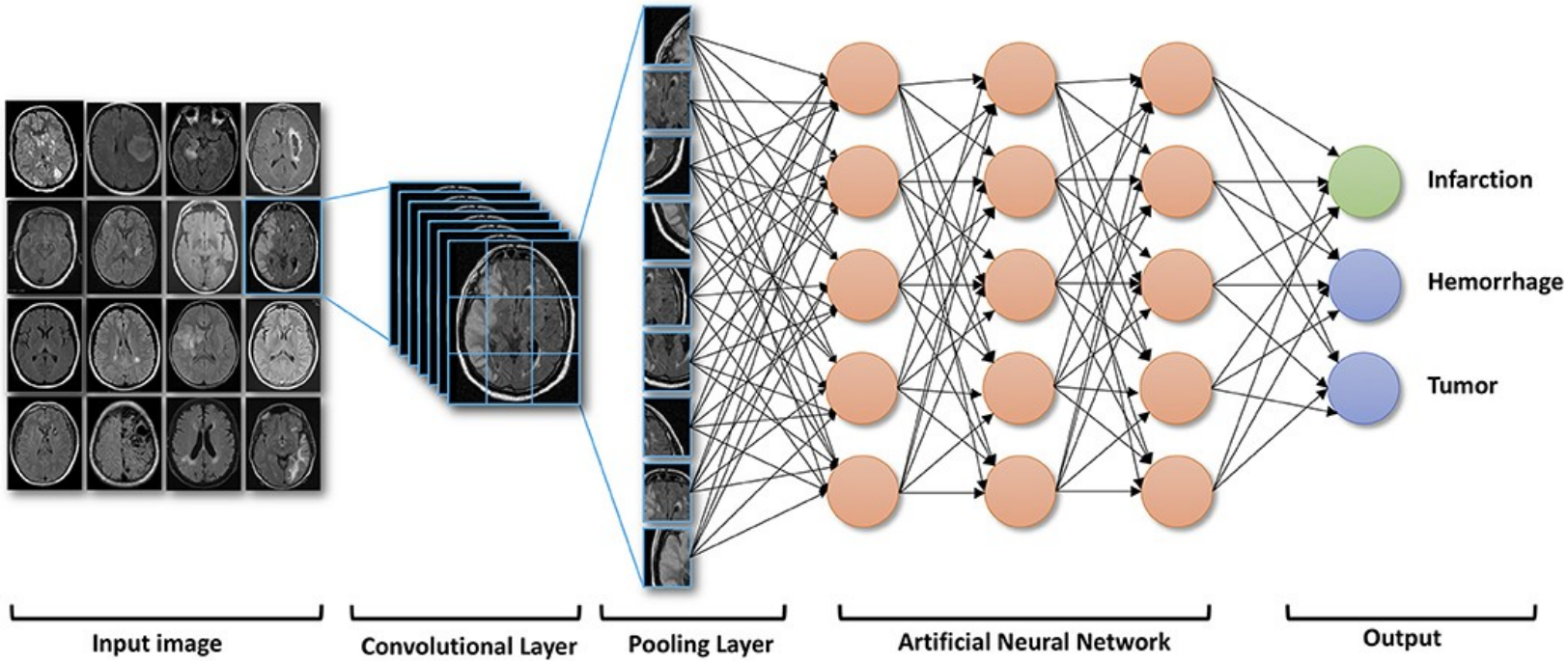
Applications in Behavioral Science: Handwriting Detection

- Classification of handwritten digits:

(James et al., 2021, Figure 10.3)

Applications in Neuroscience: Brain Scans

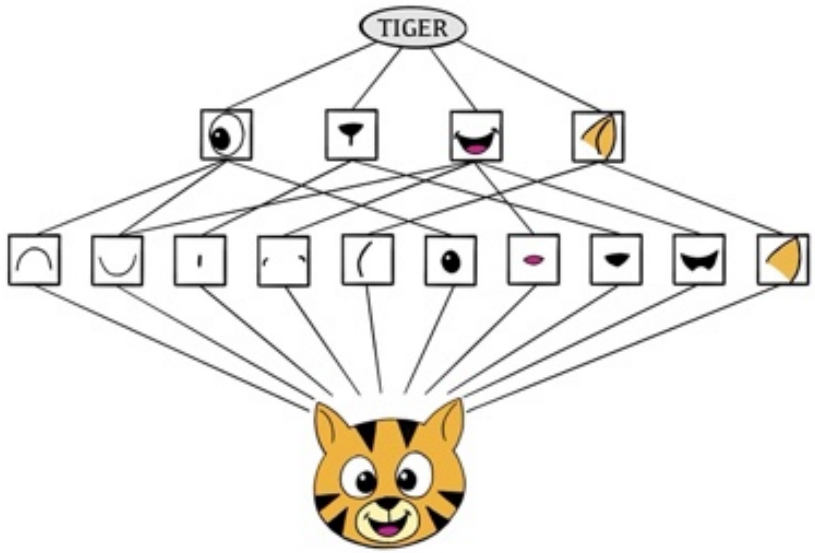
- Screening fMRI scans for signs of disease such as cancer:



(adapted from Zhu et al., 2019)

CNNs for Image Recognition

- CNN layers for image identification:
  - Takes in the image and identifies local features (e.g., a specific shape of a line or a color)
  - Combines the local features in order to create compound features (e.g., eyes and ears)
  - Compound features are used to output the target label "tiger"



(James et al., 2021, Figure 10.6)

Image Recognition in `tf-keras`

- Facial emotion recognition task:

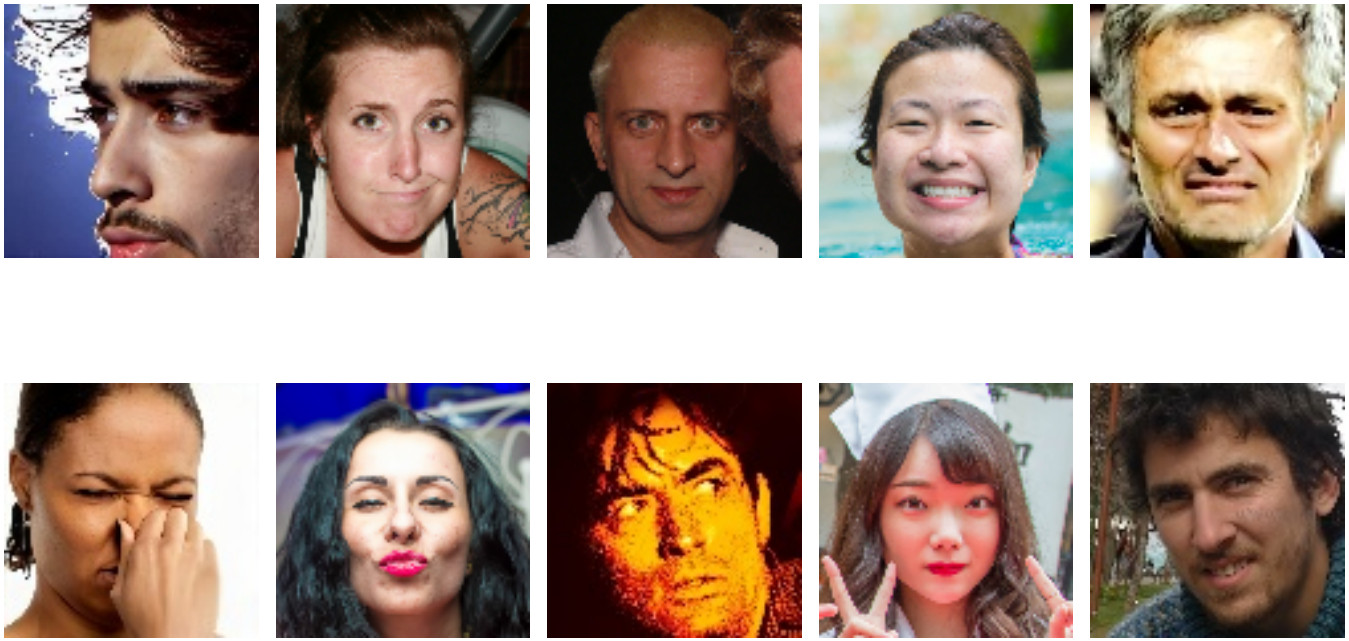


Image Recognition in tf-keras

- Reading the images as data into R:

```
library(reticulate) #; py_config() #; use_condaenv("r-reticulate")
datasets <- import("datasets") #import Python package as "variable" into R
dat = datasets$load_dataset("FastJobs/Visual_Emotional_Analysis", split = 'train')

# Translate true numeric class to verbal label and transforming Python to R data:
dict_emo <- dat$features$label$names
dat <- dat$to_pandas() %>%
  as_tibble() %>%
  mutate(emotion = dict_emo[label+1])

# Selecting a random subset of images:
set.seed(1)
dat_subset <- dat %>% sample_n(10)
head(dat_subset)
## # A tibble: 6 x 3
##   image             label emotion
##   <list>            <dbl> <chr>
## 1 <named list [2]>      6 sad
## 2 <named list [2]>      1 contempt
## 3 <named list [2]>      5 neutral
## 4 <named list [2]>      4 happy
## 5 <named list [2]>      2 disgust
## 6 <named list [2]>      2 disgust
```

Image Recognition in `tf-keras`

- Loading a pre-trained CNN that is fine-tuned for facial emotion recognition
  - As a pipeline: I.e., incl. image data pre-processor
    - E.g, standardizing the image size before fitting the model

```
transformers <- import("transformers") #import Python package as "variable" into R
prc <- transformers$AutoImageProcessor$from_pretrained("dennisjooo/emotion_classification")
ppl <- transformers$pipeline(model = 'dennisjooo/emotion_classification',
                             image_processor = prc)
ppl
## <transformers.pipelines.image_classification.ImageClassificationPipeline object at 0x000
##  signature: (
##    images: Union[str, List[str], ForwardRef('Image.Image'), List[ForwardRef('Image.Image
##    **kwargs
## )
```

Image Recognition in tf-keras

- Model predictions:

```
dat_subset <- dat_subset %>%
  rowwise() %>%
  mutate(pred = ppl(image$path) %>% as.data.frame())
tail(dat_subset)
## # A tibble: 6 x 4
## # Rowwise:
##   image            label emotion  pred$label $score $label.1 $score.1 $label.2
##   <list>           <dbl> <chr>    <chr>       <dbl> <chr>       <dbl> <chr>
## 1 <named list [2]>     2 disgust  disgust     0.360 anger      0.350  sad
## 2 <named list [2]>     2 disgust  disgust     0.438 anger      0.324  sad
## 3 <named list [2]>     1 contempt contempt    0.688 disgust    0.102  happy
## 4 <named list [2]>     3 fear     fear        0.708 anger      0.0705 sad
## 5 <named list [2]>     5 neutral  neutral     0.733 happy      0.0608 surprise
## 6 <named list [2]>     2 disgust  disgust     0.519 anger      0.191  contempt
## # i 5 more variables: pred$score.2 <dbl>, $label.3 <chr>, $score.3 <dbl>,
## #   $label.4 <chr>, $score.4 <dbl>
```

Excurse: Image Generation

- Technically, CNNs cannot only be used for image recognition, but also for image creation
- **Generative Adversarial Network (GAN):** Like a game between two players, one player being the Generator (i.e., the **artist**) and the other player being the Discriminator (i.e., an **art critic**, who tries to tell if an image is real or fake)
  - Generator-CNN: Takes random noise as input and transforms it through several layers to produce an image
    - Starts with basic features and gradually adds more detail to generate a complete image
  - Discriminator-CNN: Takes an image from the Generator-CNN as input and processes it through several layers
    - Produces a single output: **Fake vs. real**

Hands-on Practical Tutorial

- Now it's your turn:
  - Go to https://tobiasrebholz.github.io/teaching
  - Select "Introduction to Machine Learning" > "Materials"
    - Password: **smip24**
  - Download the "Deep Learning" tutorial
  - Work through the tasks

Natural Language Processing

Text Mining Applications

- **Content classification** of news stories, social media posts, journal entries from experience sampling studies, ...
- **Content filtering** (e.g., spam emails, hate speech on social media)
- Content **summarizing**, **paraphrasing**, and **translation**, ...
- Clustering of documents and web pages according to **shared features** (e.g., similar topics)
- Insights about **trends** (e.g., hashtags on Twitter/X, Google search)

Challenges in Text Mining

- Large and unstructured textual databases
  - Sometimes documents are not even available in electronic form
- Very high number of possible "dimensions", and extremely sparse
  - All possible word and phrase types in a language
- Complex and subtle relationships between concepts in text
  - Word ambiguity: E.g., Apple (the company) vs. apple (the fruit)
  - Context sensitivity: E.g., humor
  - Irony, sarcasm, ...
- Rather noisy data: E.g., spelling or grammar mistakes
  - Esp. relevant and problematic for social media data
- ...

Tokenization

- **Tokenization:** The tedata pre-processing process of breaking text into pieces (e.g., words, keywords, phrases, symbols, and other elements)
  - "Tokens": Often individual words, but they can also be be special characters, such as punctuation marks or emojis
- The tokens resulting from a tokenization are machine readable
  - I.e., can be analyzed using (ML-based) NLP techniques

Tokenization: Most Simple Forms

```
"Psychology is the science of the behavior."
## [1] "Psychology is the science of the behavior."
```

- **Bag-of-words:**

```
cbind(c("psychology", 1), c("is", 1), c("the", 2), c("science", 1), c("of", 1),
      c("behavior", 1))
##      [,1]         [,2] [,3]  [,4]      [,5] [,6]
## [1,] "psychology" "is" "the" "science" "of" "behavior"
## [2,] "1"          "1"  "2"   "1"       "1"  "1"
```

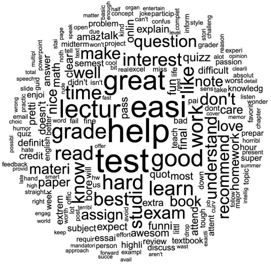
- **String-of-words:**

```
cbind(c("psychology", 1), c("is", 2), c("the", 3), c("science", 4), c("of", 5),
      c("the", 6), c("behavior", 7))
##      [,1]         [,2] [,3]  [,4]      [,5] [,6]  [,7]
## [1,] "psychology" "is" "the" "science" "of" "the" "behavior"
## [2,] "1"          "2"  "3"   "4"       "5"  "6"   "7"
```

- Why is this distinction so important?
  - Shakespeare's plays contain ca. 885,000 words, but the count of unique word forms is only 29,000 (https://www.opensourceshakespeare.org/statistics/)

Visualizations of Text Data

- **Word clouds:** Visualizing the information contained in text data (i.e., open comments in teaching evaluations) by displaying the most frequently occurring words
  - The size of each word represents its frequency (as a proxy for its **importance**) within the text corpus
  - Linking words (e.g., "and"), articles (e.g., "the"), and so on, are typically excluded because they have no meaning on their own



(Jacobucci et al., Figure 11.10)

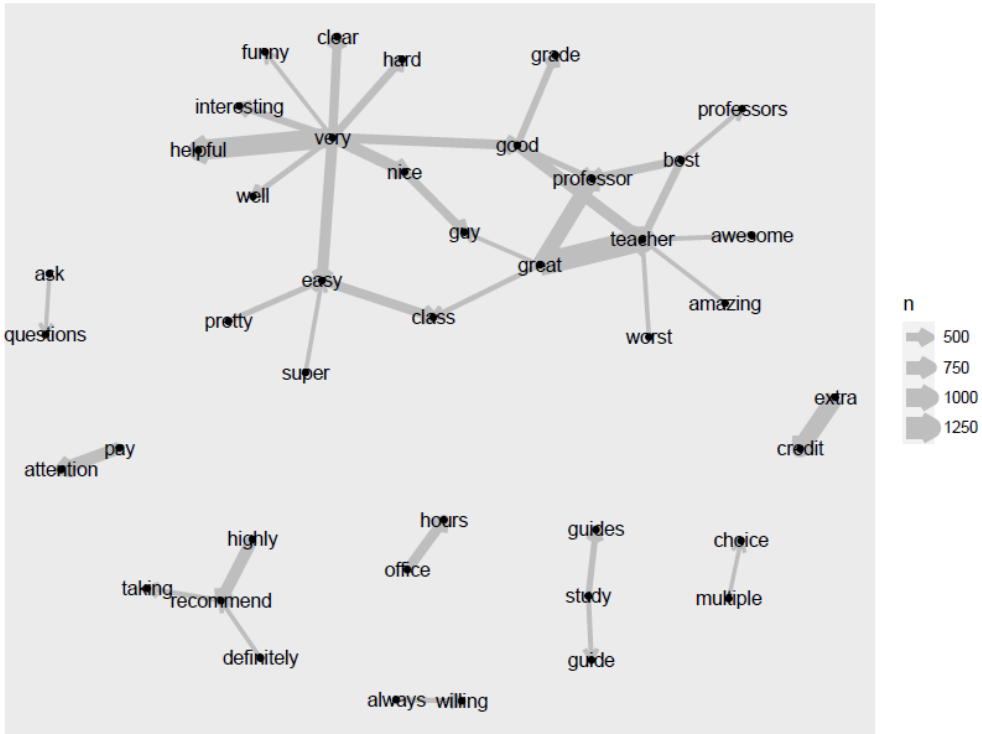
Visualizations of Text Data

- A little more informative in terms of content than individual word clouds:
  - The most frequently occurring **bigrams** (left) and **trigrams** (right)

(Jacobucci et al., Figures 11.12 & 11.13)

Word Embeddings

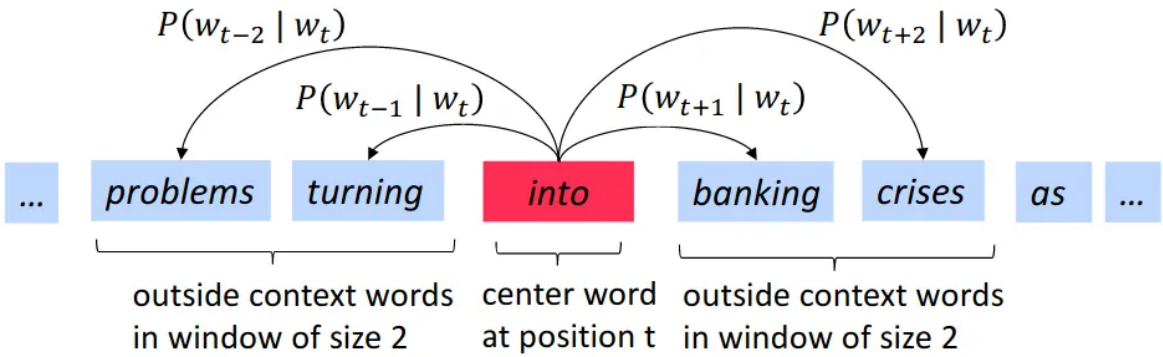
- In general, language is much more than a loose collection of its constituent components
  - **Network plot:** Visualizing the associations between the top used words



(Jacobucci et al., Figure 11.14)

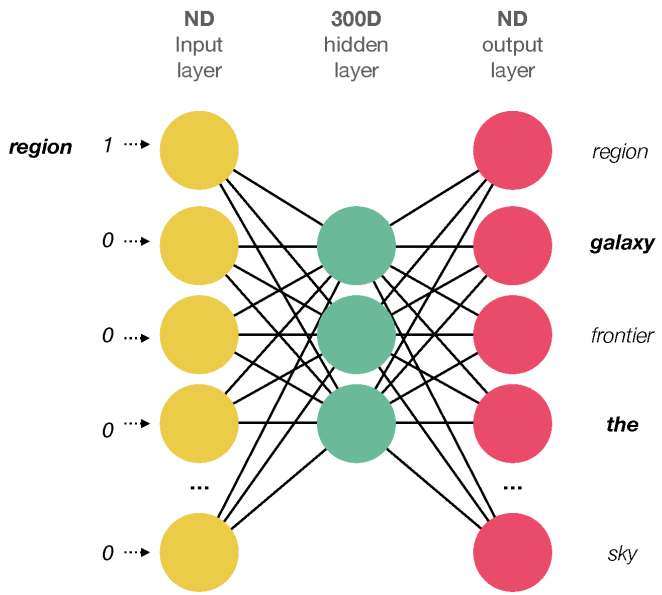
Word Embeddings

- **Word2Vec** (Mikolov et al., 2013): Unsupervised Learning technique to learn continuous, multi-dimensional vector representations of words
  - Idea: Similarity of words in a training corpus (i.e., how far or near a word is in vector space to other words) represents a notion of word-sense based on surrounding words
- Given a center word, learning to predict the most likely words in a fixed-sized window around it:



(https://towardsdatascience.com/word2vec-to-transformers-caf5a3daa08a)

Example I



"You're on your way, Kelvin. Good luck!" Moddard's voice sounded as close as before.

A wide slit opened at eye-level, and I could see the stars. The _Prometheus_ was orbiting in the region of Alpha in Aquarius and I tried in vain to orient myself; a glittering dust filled my porthole. I could not recognize a single constellation; in this region of the galaxy the sky was unfamiliar to me. I waited for the moment when I would pass near the first distinct star, but I was unable to isolate any one of them. Their brightness was fading; they receded, merging into a vague, purplish glimmer, the sole indication of the distance I had already travelled. My body rigid, sealed in its pneumatic envelope, I was knifing through space with the impression of standing still in the void, my only distraction the steadily mounting heat.

Suddenly, there was a shrill, grating sound, like a steel blade being drawn across a sheet of wet glass. This was it, the descent. If I had not seen the figures racing across the dial, I would not have noticed the change in direction. The stars having vanished long since, my gaze was swallowed up on the pale reddish glow of infinity. I could hear my heart thudding heavily. I could feel the coolness from the air-conditioning on my neck, although my face seemed to be on fire. I regretted not having caught a glimpse of the _Prometheus_, but the ship must have been out of sight by the time the automatic controls had raised the shutter of my porthole.

The capsule was shaken by a sudden jolt, then another. The whole vehicle began to vibrate. Filtered through the insulating layers of the outer skins, penetrating my pneumatic cocoon, the vibration reached me, and ran through my entire body. The image of the dial shivered and multiplied, and its phosphorescence spread out in all directions. I felt no fear. I had not undertaken this long voyage only to overshoot my target!

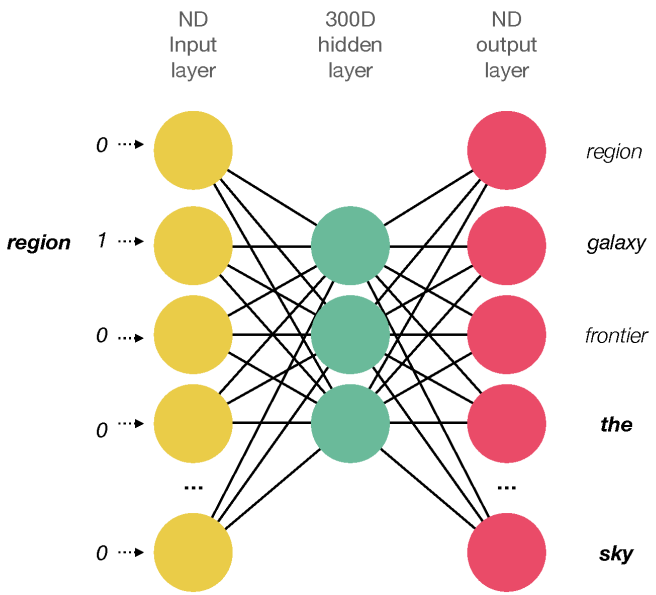
I called into the microphone:

"Station Solaris! Station Solaris! Station Solaris! I think I am leaving the flight-path, correct my course! Station Solaris, this is the _Prometheus_ capsule. Over."

I had missed the precious moment when the planet first came into view. Now it was spread out before my eyes; flat, and already immense. Nevertheless, from the appearance of its surface, I judged that I was still at a great height above it, since I had passed that imperceptible frontier after which we measure the distance that separates us from a celestial body in terms of altitude. I was falling. Now I had the sensation of falling, even with my eyes closed. (I quickly reopened them: I did not want to miss anything there was to be seen.)

(https://www.webnovel.com/book/solaris-2.0_25879657706474305)

Example II



"You're on your way, Kelvin. Good luck!" Moddard's voice sounded as close as before.

A wide slit opened at eye-level, and I could see the stars. The _Prometheus_ was orbiting in the region of Alpha in Aquarius and I tried in vain to orient myself; a glittering dust filled my porthole. I could not recognize a single constellation; in this region of the galaxy the sky was unfamiliar to me. I waited for the moment when I would pass near the first distinct star, but I was unable to isolate any one of them. Their brightness was fading; they receded, merging into a vague, purplish glimmer, the sole indication of the distance I had already travelled. My body rigid, sealed in its pneumatic envelope, I was knifing through space with the impression of standing still in the void, my only distraction the steadily mounting heat.

Suddenly, there was a shrill, grating sound, like a steel blade being drawn across a sheet of wet glass. This was it, the descent. If I had not seen the figures racing across the dial, I would not have noticed the change in direction. The stars having vanished long since, my gaze was swallowed up on the pale reddish glow of infinity. I could hear my heart thudding heavily. I could feel the coolness from the air-conditioning on my neck, although my face seemed to be on fire. I regretted not having caught a glimpse of the _Prometheus_, but the ship must have been out of sight by the time the automatic controls had raised the shutter of my porthole.

The capsule was shaken by a sudden jolt, then another. The whole vehicle began to vibrate. Filtered through the insulating layers of the outer skins, penetrating my pneumatic cocoon, the vibration reached me, and ran through my entire body. The image of the dial shivered and multiplied, and its phosphorescence spread out in all directions. I felt no fear. I had not undertaken this long voyage only to overshoot my target!

I called into the microphone:

"Station Solaris! Station Solaris! Station Solaris! I think I am leaving the flight-path, correct my course! Station Solaris, this is the _Prometheus_ capsule. Over."

I had missed the precious moment when the planet first came into view. Now it was spread out before my eyes; flat, and already immense. Nevertheless, from the appearance of its surface, I judged that I was still at a great height above it, since I had passed that imperceptible frontier after which we measure the distance that separates us from a celestial body in terms of altitude. I was falling. Now I had the sensation of falling, even with my eyes closed. (I quickly reopened them: I did not want to miss anything there was to be seen.)

(https://www.webnovel.com/book/solaris-2.0_25879657706474305)

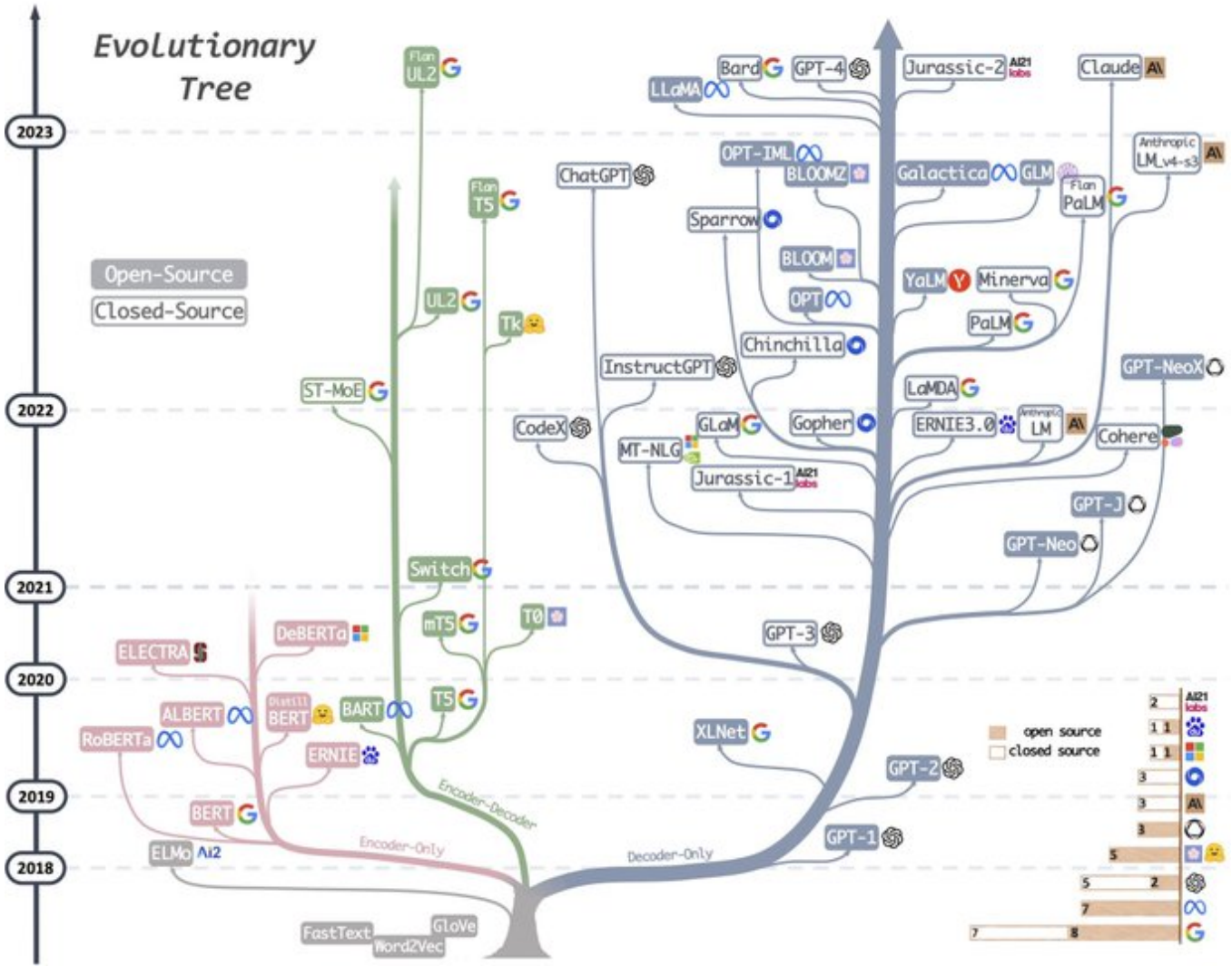
Transformers

Transformers

Word2Vec	**Transformers**
Generates **static** word embeddings $\Rightarrow$ The vector representation for a word is the same, regardless of its context	Generates **dynamic** word embeddings $\Rightarrow$ The vector representation for a word can change, based on its context
Learns representations based on the **local** context (i.e., surrounding words)	Learns representations based on the **entire** context of the sentence or even multiple sentences

- Transformers are the leading architecture for most contemporary Large Language Models (LLMs)
  - OpenAI's ChatGPT, Google's Gemini, Meta's LLaMA, Anthropic's Claude, ...
  - We will discuss some of the high-level concepts involved, but will mainly focus on using them in Python/R here

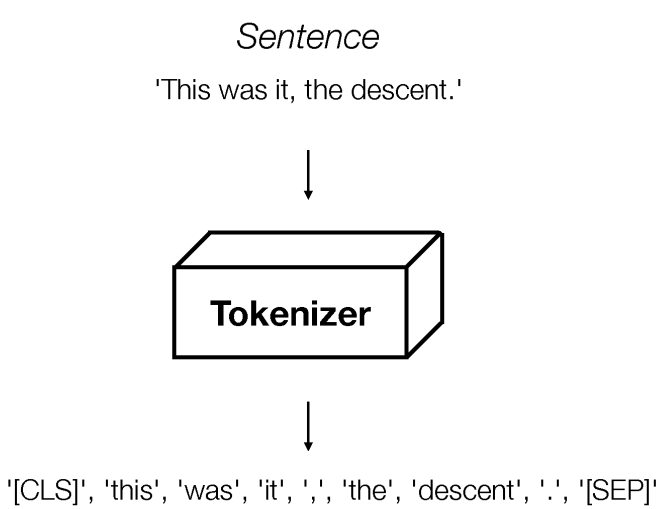
Evolution



(https://x.com/DrJimFan/status/1651968203701231616)

Tokenization: Example I

- Transformers require more advanced tokenization approaches:



"You're on your way, Kelvin. Good luck!" Moddard's voice sounded as close as before.

A wide slit opened at eye-level, and I could see the stars. The _Prometheus_ was orbiting in the region of Alpha in Aquarius and I tried in vain to orient myself; a glittering dust filled my porthole. I could not recognize a single constellation; in this region of the galaxy the sky was unfamiliar to me. I waited for the moment when I would pass near the first distinct star, but I was unable to isolate any one of them. Their brightness was fading; they receded, merging into a vague, purplish glimmer, the sole indication of the distance I had already travelled. My body rigid, sealed in its pneumatic envelope, I was knifing through space with the impression of standing still in the void, my only distraction the steadily mounting heat.

Suddenly, there was a shrill, grating sound, like a steel blade being drawn across a sheet of wet glass. This was it, the descent. If I had not seen the figures racing across the dial, I would not have noticed the change in direction. The stars having vanished long since, my gaze was swallowed up on the pale reddish glow of infinity. I could hear my heart thudding heavily. I could feel the coolness from the air-conditioning on my neck, although my face seemed to be on fire. I regretted not having caught a glimpse of the _Prometheus_, but the ship must have been out of sight by the time the automatic controls had raised the shutter of my porthole.

The capsule was shaken by a sudden jolt, then another. The whole vehicle began to vibrate. Filtered through the insulating layers of the outer skins, penetrating my pneumatic cocoon, the vibration reached me, and ran through my entire body. The image of the dial shivered and multiplied, and its phosphorescence spread out in all directions. I felt no fear. I had not undertaken this long voyage only to overshoot my target!

I called into the microphone:

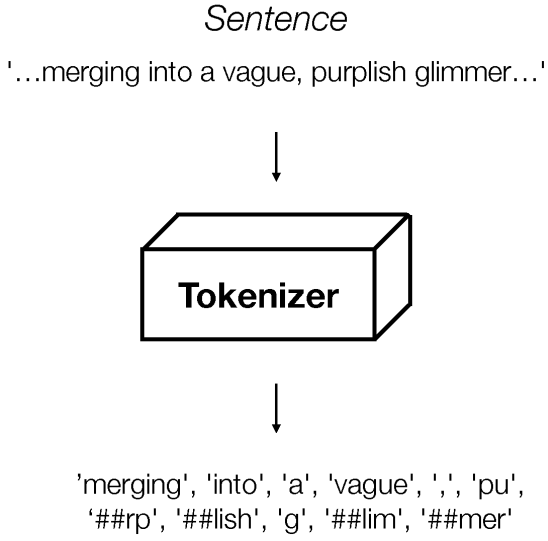
"Station Solaris! Station Solaris! Station Solaris! I think I am leaving the flight-path, correct my course! Station Solaris, this is the _Prometheus_ capsule. Over."

I had missed the precious moment when the planet first came into view. Now it was spread out before my eyes; flat, and already immense. Nevertheless, from the appearance of its surface, I judged that I was still at a great height above it, since I had passed that imperceptible frontier after which we measure the distance that separates us from a celestial body in terms of altitude. I was falling. Now I had the sensation of falling, even with my eyes closed. (I quickly reopened them: I did not want to miss anything there was to be seen.)

(https://www.webnovel.com/book/solaris-2.0_25879657706474305)

Tokenization: Example I

- Transformers require more advanced tokenization approaches:



"You're on your way, Kelvin. Good luck!" Moddard's voice sounded as close as before.

A wide slit opened at eye-level, and I could see the stars. The _Prometheus_ was orbiting in the region of Alpha in Aquarius and I tried in vain to orient myself; a glittering dust filled my porthole. I could not recognize a single constellation; in this region of the galaxy the sky was unfamiliar to me. I waited for the moment when I would pass near the first distinct star, but I was unable to isolate any one of them. Their brightness was fading; they receded, merging into a vague, purplish glimmer the sole indication of the distance I had already travelled. My body rigid, sealed in its pneumatic envelope, I was knifing through space with the impression of standing still in the void, my only distraction the steadily mounting heat.

Suddenly, there was a shrill, grating sound, like a steel blade being drawn across a sheet of wet glass. This was it, the descent. If I had not seen the figures racing across the dial, I would not have noticed the change in direction. The stars having vanished long since, my gaze was swallowed up on the pale reddish glow of infinity. I could hear my heart thudding heavily. I could feel the coolness from the air-conditioning on my neck, although my face seemed to be on fire. I regretted not having caught a glimpse of the _Prometheus_, but the ship must have been out of sight by the time the automatic controls had raised the shutter of my porthole.

The capsule was shaken by a sudden jolt, then another. The whole vehicle began to vibrate. Filtered through the insulating layers of the outer skins, penetrating my pneumatic cocoon, the vibration reached me, and ran through my entire body. The image of the dial shivered and multiplied, and its phosphorescence spread out in all directions. I felt no fear. I had not undertaken this long voyage only to overshoot my target!

I called into the microphone:

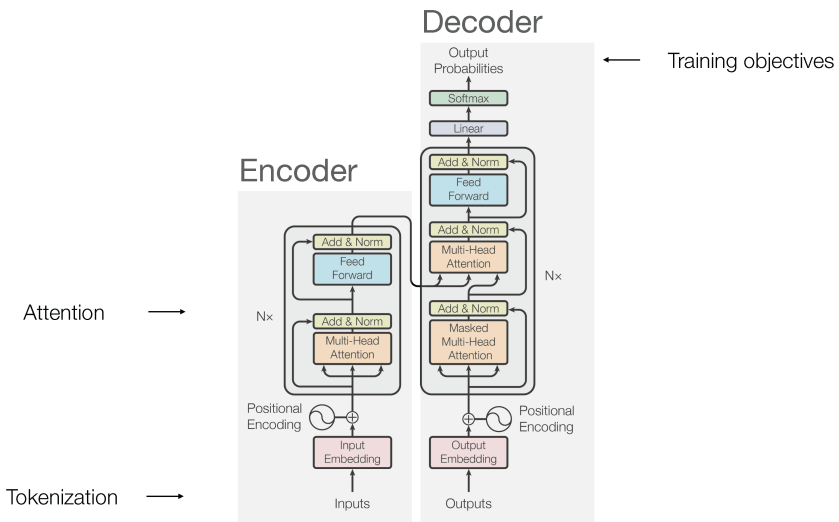
"Station Solaris! Station Solaris! Station Solaris! I think I am leaving the flight-path, correct my course! Station Solaris, this is the _Prometheus_ capsule. Over."

I had missed the precious moment when the planet first came into view. Now it was spread out before my eyes; flat, and already immense. Nevertheless, from the appearance of its surface, I judged that I was still at a great height above it, since I had passed that imperceptible frontier after which we measure the distance that separates us from a celestial body in terms of altitude. I was falling. Now I had the sensation of falling, even with my eyes closed. (I quickly reopened them: I did not want to miss anything there was to be seen.)

(https://www.webnovel.com/book/solaris-2.0_25879657706474305)

Transformer Architecture

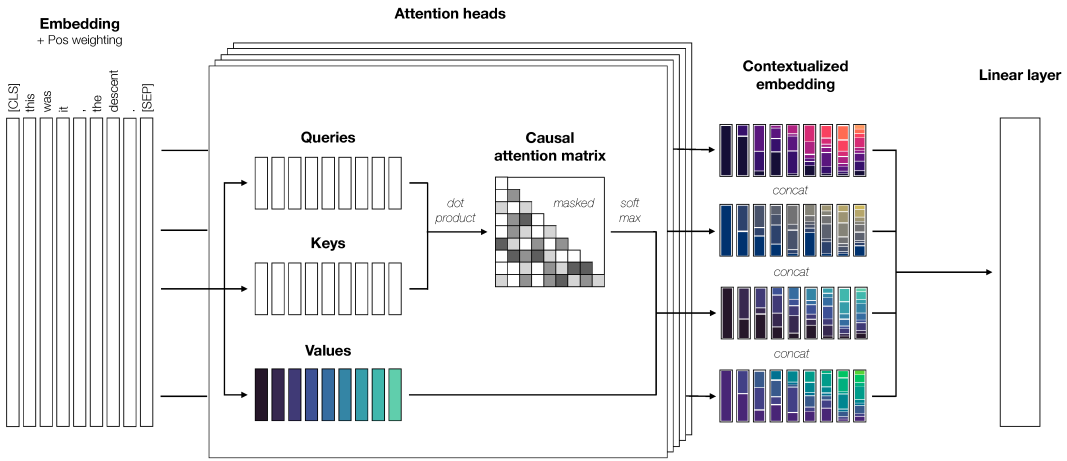
- **Encoder:** Produces an encoded representation of the input sequence
  - The target sequence is converted into **embeddings** (i.e., "meaning")
- **Decoder:** Processes the embeddings along with the Encoders output to produce an encoded representation of the target sequence
- **Output layer:** Converts encoded representations to word probabilities
  - **Loss function:** Compares this output sequence with the target sequence from the training data



(Vaswani et al., 2017)

Key Component: Multi-Head Attention

- Allows to encode multiple relationships and nuances for each word
- Technically: At each layer, each token is contextualized within the scope of the context window with other (unmasked) tokens
  1. Splitting the **input embeddings** into multiple "heads," each learning different types of information
  2. Calculating **attention scores** used to weight the embedding vectors
  3. The outputs from all heads are concatenated and transformed to produce **contextualized embeddings**



(adapted from https://x.com/dirkuwulff/status/1694988985225839048)

Transformers from Python in R

- Sentiment classification task using DistilBERT
  - Light and fast transformer LLM trained by distilling Google's BERT (Bidirectional Encoder Representations from Transformers)
    - The predecessor of Google's Gemini

```
library(reticulate) #; py_config() #; use_condaenv("r-reticulate")

# Loading the tokenizer (i.e., pre-processor) and transformer model:
transformers <- import("transformers") #import Python package as "variable" into R
tkn <- transformers$AutoTokenizer$from_pretrained('distilbert-base-uncased-finetuned-sst-2-
ppl <- transformers$pipeline(model = 'distilbert-base-uncased-finetuned-sst-2-english'
                             , tokenizer = tkn)
ppl
## <transformers.pipelines.text_classification.TextClassificationPipeline object at 0x00000
##  signature: (inputs, **kwargs)

# Exemplary sentiment predictions:
ppl(c("I like you!", "I don't like you!")) %>% as.data.frame()
##      label     score  label.1   score.1
## 1 POSITIVE 0.9998792 NEGATIVE 0.9989811
ppl(c("The sky is blue.", "The sky is red.")) %>% as.data.frame()
##      label     score  label.1   score.1
## 1 POSITIVE 0.9988331 NEGATIVE 0.7884832
```

Transformers from Python in R

- Hugging Face dataset: `emotion`
  - Contains English tweets and labels that classify them according to six basic emotions: joy, love, anger, fear, sadness, and surprise

```
datasets <- import("datasets") #import Python package as "variable" into R
dat = datasets$load_dataset("dair-ai/emotion", split = 'test')

# Translate true numeric class to verbal label and transforming Python to R data:
dict_emo <- dat$features$label$names
dat <- dat$to_pandas() %>% as_tibble() %>%
  mutate(emotion = dict_emo[label+1]) #NOTE: indexing starts at 0 in Python (vs. 1 in R)

# Selecting a random subset of tweets:
set.seed(1)
dat_subset <- dat %>% sample_n(100)
tail(dat_subset)
## # A tibble: 6 x 3
##   text                                                    label emotion
##   <chr>                                                   <dbl> <chr>
## 1 im sorry im feeling a little bitchy tacky looking women came in~     3 anger
## 2 i hope everyone can help with charity work without feeling stre~     0 sadness
## 3 i feel lost without you                                              0 sadness
## 4 i definitely feel there s some useful information here for anyo~     1 joy
## 5 i can imagine most young people might feel resentful about the ~     3 anger
## 6 i am a prolific writer in my fandom but do not feel that i am a~     1 joy
```

Transformers from Python in R

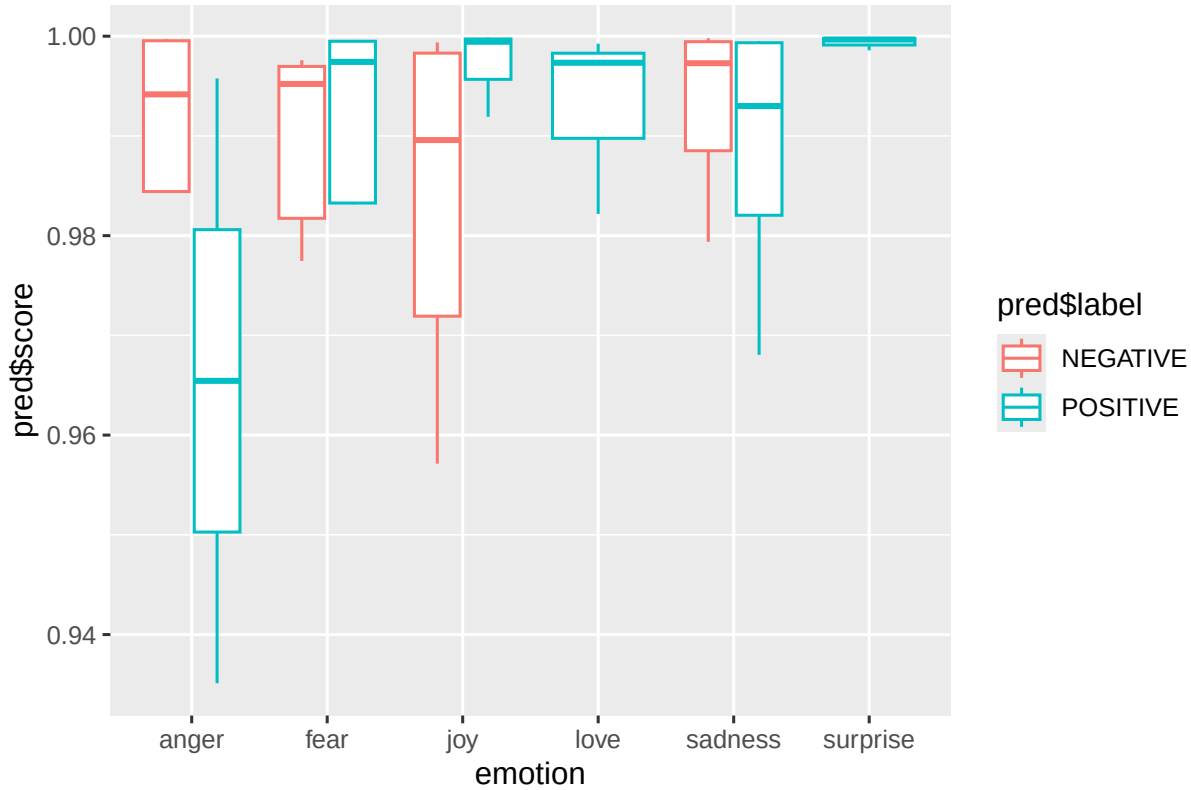
- DestilBERT's classification of tweets as positive vs. negative sentiment:

```
# Predictions for all tweets:
dat_subset <- dat_subset %>%
  rowwise() %>%
  mutate(pred = ppl(text) %>% as.data.frame())
tail(dat_subset)
## # A tibble: 6 x 4
## # Rowwise:
##   text                                                label emotion pred$label
##   <chr>                                               <dbl> <chr>   <chr>
## 1 im sorry im feeling a little bitchy tacky looking wo~     3 anger   NEGATIVE
## 2 i hope everyone can help with charity work without f~     0 sadness POSITIVE
## 3 i feel lost without you                                   0 sadness NEGATIVE
## 4 i definitely feel there s some useful information he~     1 joy     NEGATIVE
## 5 i can imagine most young people might feel resentful~     3 anger   NEGATIVE
## 6 i am a prolific writer in my fandom but do not feel ~     1 joy     POSITIVE
## # i 1 more variable: pred$score <dbl>
```

Transformers from Python in R

- Comparison of DestilBERT's predicted sentiment to the six actual basic emotions displayed (according to human experts):

```
ggplot(dat_subset, aes(x = emotion, y = pred$score, color = pred$label)) +
  geom_boxplot(outliers = FALSE)
```



Excurse: Generative AI from Python in R

- Google's BERT:

```
# Loading the tokenizer and transformer model:
tkn_bert <- transformers$AutoTokenizer$from_pretrained('distilbert/distilgpt2')
ppl_bert <- transformers$pipeline(model = 'distilbert/distilgpt2',
                                  tokenizer = tkn_bert)
ppl_bert
## <transformers.pipelines.text_generation.TextGenerationPipeline object at 0x0000028983E473
##  signature: (text_inputs, **kwargs)

# Exemplary text generation:
pred_bert <- ppl_bert("My name is Tobias and I am teaching a class on Machine Learning")
pred_bert[[1]]$generated_text %>% strwrap(., width = 80)
## [1] "My name is Tobias and I am teaching a class on Machine Learning for the"
## [2] "University of Cambridge and it involves building neural networks to detect the"
## [3] "emotions of a customer. The first step in this class is to use the Neural"
## [4] "Network (NNN). The first"
```

Excurse: Generative AI from Python in R

- OpenAI's GPT:

```
# Loading the tokenizer and transformer model:
tkn_gpt2 <- transformers$AutoTokenizer$from_pretrained('openai-community/gpt2-medium')
ppl_gpt2 <- transformers$pipeline(model = 'openai-community/gpt2-medium',
                                  tokenizer = tkn_gpt2)
ppl_gpt2
## <transformers.pipelines.text_generation.TextGenerationPipeline object at 0x0000028983DFE
##  signature: (text_inputs, **kwargs)

# Exemplary text generation:
pred_gpt2 <- ppl_gpt2("My name is Tobias and I am teaching a class on Machine Learning")
pred_gpt2[[1]]$generated_text %>% strwrap(., width = 80)
## [1] "My name is Tobias and I am teaching a class on Machine Learning at Yale"
## [2] "University. I am also on a team helping them design systems for social network"
## [3] "prediction.\""
## [4] ""
## [5] "This machine learning class will cover the fundamentals of the science, and is"
## [6] "set to"
```

Excurse: Transfer Learning

- **Transfer Learning**: Starting with layers trained to identify universal features (e.g., textures, shapes) and integrating new layers tailored to the specific task
  - I.e., optimizing for unique challenges without starting from scratch
  - Utilizing knowledge gained from previous training (i.e., leveraging pre-trained models) on a broad dataset and adapting it to new, often more specific tasks
- Technically: The last couple of hidden layers of the original NN are simply **replaced** with new layers trained for solving the new, specific task
  - Why? – Because it was found to work well
  - Goal: Saving computational resources and training time
- Use cases:
  - Limited training data
    - E.g., recognizing specific persons from only a few images
  - Complex classification tasks
    - E.g., automated driving: Distinguishing between hundreds of traffic signs and other categories

Summary

Summary

- DL is attractive for:
  - Extremely large, complex, and unstructured data
  - Low priority of model interpretability ($\rightarrow$ Module 5!)
- NNs are particularly suited for niche tasks, such as:
  - Image recognition
  - Speech and text modeling
- Classic NLP tasks:
  - Sentiment analysis, language translation, ...
  - Recurrent NNs and Transformers have shown great success in these areas
    - Due to their ability to handle sequential data and capture long-term dependencies