# Applied Machine Learning, GU/Chalmers, 2020
# Exercises, part 2: solutions

## 1 Practical Machine Learning Problems

### 1.1 Finding children in images

A car manufacturer would like to build a classifier that detects whether an image contains a child or not.
**(a)** What type of *data* would you suggest the company to collect? How should the data be collected?
**(b)** What *method* would you suggest that they use to build the classifier?
**(c)** How do you think they should *measure the quality* of the system?
**Solution.**
**(a)** We need to collect a substantial number of images; preferably, these images should be collected in similar circumstances as where the system will operate (e.g. using the same type of camera, mounted on a vehicle). Then, annotators will need to classify images as belonging to the positive class (contains children) or negative (no children).
**(b)** Clearly a use case for a convolutional neural network. The question is just whether we will train the CNN from scratch, or use a pre-trained model (as in Assignment 5) to generate features.
**(c)** Precision/recall is probably more meaningful than accuracy for this task: this is a "spotting" task where we are trying to find some class (in this case, children), and where probably we need to think of whether false positives or false negatives are more problematic.

### 1.2 Restaurant recommendation

You have been employed at a company that wants to produce a recommendation system for restaurants. The system will cover restaurants in Sweden initially, but will eventually include restaurants in other countries.

It is crucial that the system is able to give recommendations to new users; conversely, if a new restaurant opens, the system should be able to direct customers there immediately. The system should be able to give you recommendations in all cities, not only where you spend most of the time.
**(a)** What type of *data* would you suggest the company to collect? How should the data be collected?
**(b)** What *method* would you suggest that they use to give recommendations?
**(c)** How do you think they should *measure the quality* of the system?
**Solution.**
**(a)** The explicitly stated requirements suggest that we need to develop a recommender system that is at least partly based on descriptions of the restaurants and/or of the users – in recommender system terminology, we need to develop a *content-based* system. The alternative, *collaborative filtering*, probably cannot be used on its own because of the "cold start" problem: the system would have no meaningful data about new users and new restaurants.

So probably, each user will need to enter some descriptive parameters about him/herself. Conversely, each time a new restaurant is entered into the system, it needs to be described using some features (e.g. type of cuisine, price range, etc). Furthermore, the system needs to include some sort of feedback: whether a user liked a restaurant or not (a star rating or thumbs-up/thumbs-down).

Even though collaborative filtering does not seem meaningful on its own, it might still be useful to use it in combination with the feature-based approach.

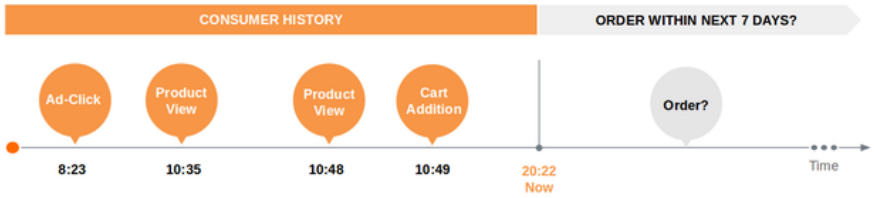
**(b)** For a purely feature-based approach, we can just treat prediction as a regression problem (if the feedback is star ratings) or classification problem (thumbs-up/thumbs-down).

**(c)** In the simplest case, evaluate directly as a regressor (MSE, $R^2$) or classifier (accuracy). More specialized methods may include user interface evaluation metrics such as click-through rate.

### 1.3 Will the customer place an order?

An e-commerce company in the fashion business would like to develop a classifier that determines whether a customer is likely to order an item within the next few days. (This information can be useful for advertising.)

The classification should be based on the customer's past behavior, which is described by an *history*. Here is an example.




Each event is described by a type name (e.g. *Product View*), but also by some additional parameters (e.g. a timestamp or the item category).

What type of classifier would you suggest for this prediction problem? How should it use the information in the customer's history?

**Solution.**

This idea comes from a blog post by a developer at Zalando: `https://jobs.zalando.com/tech/blog/deep-learning-for-understanding-consumer-histories/?gh_src=4n3gxh1`

In short, there are probably two approaches that are meaningful here (as discussed in the blog post). The traditional approach would be to define a set of features based on the consumer's history and use any classifier (linear, neural network, tree ensemble, etc). The more recent approach would be to apply some type of recurrent neural network (e.g. LSTM) that processes each of the events in a sequence, and then makes a prediction based on its internal state. The Zalando blog post claims that the second approach gives an improvement over a traditional feature-based approach.

## 2 Neural networks

### 2.1 General neural network questions

**(a)** What are the benefits of using methods such as *Adam* and *Adagrad*?

**Solution.**

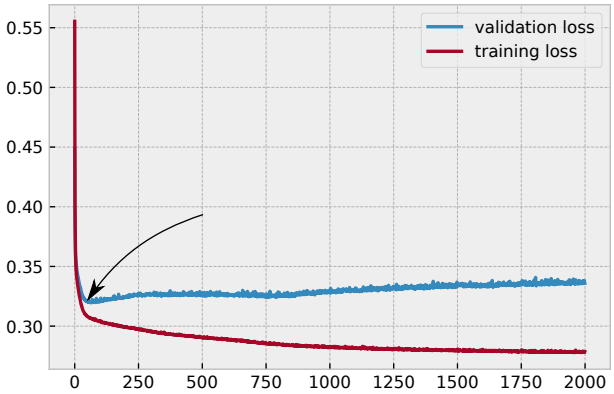
These methods automatically adjust the learning rate, to accelerate and slow down when needed. Using these methods, rather than stochastic gradient descent with a constant learning rate, will typically speed up training of a neural network.

**(b)** What is *dropout* and why do we use it?

**Solution.**
Dropout is a method to reduce the risk of overfitting in neural network. In dropout, hidden units are temporarily removed during training (typically before each "minibatch"). The idea is that this will make the network less dependent on a small number of hidden units.

**(c)** How does *early stopping* work?

**Solution.**
Early stopping also has the purpose of reducing the risk of overfitting. We set aside a subset of the training data: this is called a validation set or development set. During training, we continuously monitor the model's performance on the validation set. When the model starts to overfit, the model's performance on the validation set will deteriorate. Early stopping means that training is terminated when we no longer see improvements on the validation set.




**(d)** If we train a neural network using scikit-learn (`MLPClassifier` or `MLPRegressor`), we can give a "seed" value to the random number generator:

```
clf = MLPClassifier(random_state=12345)
```

Changing this random seed can sometimes have significant effects on the model's performance. Please explain why.

**Solution.**
The reason is that the objective functions optimized by neural networks have a "rugged" shape with many local minima. Training neural networks includes some steps involving randomness (e.g. weight initialization, selecting instances in SGD), which can influence in which local minimum we end up.

For this reason, it can be useful to try out a few different random seeds when training neural networks, and select the model that gives the best performance on a held-out set.

**(e)** What is the typical method for encoding *words* as the input to a neural network for some text processing task?

**Solution.**
In neural networks, words are typically encoded as *word embeddings* – short vectors that encode the "meaning" of the words. They can be trained as a part of the general training process for the task that we're solving, or in a separate pre-training step. In the latter case,

we can use a large amount of unlabeled text and apply an unsupervised algorithm such as `word2vec`.

### 2.2 Neural network classification [recycled exam question]

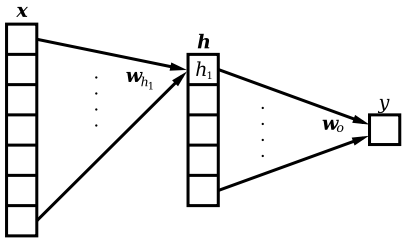
**(a)** Describe how classification is done in a binary feedforward neural network classifier with one hidden layer. Explain how the input looks, and the steps that are carried out by the classifier until the output is produced.
**(b)** Describe in general terms how a feedforward neural network can be trained.

**Solution.**
**(a)** As usual, the input to the classifier is a numerical vector $\boldsymbol{x}$ encoding our features. This is more or less like in the other classifier (SVM, LR), and $\boldsymbol{x}$ could be prepared using one of the vectorizers in scikit-learn. A difference in practice is that $\boldsymbol{x}$ tends to have a lower dimensionality when working with NNs, and in particular for words we prefer to use *embeddings* rather than one-hot encoding.

The first step in a feedforward NN is to compute the values in the *hidden layer*, which will result in a new vector $\boldsymbol{h}$. Each number in $\boldsymbol{h}$ is the result of applying a "sub-classifier": for instance, to compute the first number in $\boldsymbol{h}$, let's call it $h_1$, we use a weight vector $\boldsymbol{w}_{h_1}$ specifically dedicated to $h_1$. We compute the score by using the dot product between this weight vector and the input vector $\boldsymbol{x}$, and then applying an *activation function* $a_h$. For instance, the sigmoid is a common choice of activation function, and in this case the "sub-classifier" is like a logistic regression classifier.

When all the numbers in $\boldsymbol{h}$ have been computed in this way, we apply a similar procedure to compute the output. But now, we use $\boldsymbol{h}$ instead of the input vector $\boldsymbol{x}$.

Here is a figure illustrating this process:




Alternatively, using matrix notation, we can explain the NN classification in a less verbose way. We make the vector $\boldsymbol{h}$ of hidden-layer outputs by first multiplying the weight matrix $\boldsymbol{W}_h$ with the input vector $\boldsymbol{x}$, and then applying the activation function $a_h$ to the result of the matrix multiplication. (The activation function is applied element by element in the vector.)

$$\boldsymbol{h} = a_h(\boldsymbol{W}_h \cdot \boldsymbol{x})$$

And the output value is computed is a similar fashion, but using the hidden-layer vector instead:

$$y = a_o(\boldsymbol{W}_o \cdot \boldsymbol{h})$$

To answer this question and get a full score, you can draw a figure, use formulas, or just explain in plain text. A complete solution will mention that the input is a numerical vector, and how each "layer" is computed from a previous layer (weighted score and activation function).

**(b)** Just like for logistic regression and SVM in Assignment 4, this is about defining an objective function that we optimize. This will typically be a loss function (for instance the log loss)

applied to the network's output. Most commonly, we'll then apply some sort of stochastic gradient descent, as in the logistic regression training algorithm. The gradients of the weights are computed using the *backpropagation* algorithm, which works by first computing the value of the output and all hidden units, and then computing the gradients in each layer backwards from the output through the lower layers. A passable solution to this question will at least mention that we're doing optimization based on gradients.

### 2.3 Neural networks and other classifiers [recycled exam question]

**(a)** Why is training a support vector machine easier than training a feedforward neural network classifier (at least from a mathematical point of view)?
**(b)** When building systems based on classifiers such as SVCs or logistic regression, it is common to use *combination features* or *feature crosses*. To exemplify, we might use the features

```
{ 'gen':'female', 'occ':'farmer', 'gen+occ':'female+farmer' }
```

The combination feature looks redundant, but explain why it can be useful for the SVC but probably less useful when using a feedforward neural network classifier.
**(c)** In neural networks, when using features that represent *words*, we would normally prefer to use *word embeddings* instead of reserving one dimension for each word in the vocabulary (one-hot encoding). This is partly for technical reasons: shorter vectors use less memory. Explain why this approach has advantages in addition to saving memory, and in particular how this allows us to use a semi-supervised training setup.
**Solution.**
**(a)** As we discussed in one of the lectures, SVC has an objective function that is convex. For convex functions, if we find a minimum it is guaranteed to be unique. This means that it's quite easy to find the minimum using algorithms such as stochastic gradient descent. A neural network on the other hand is more difficult for SGD because its objective function is "rugged."
**(b)** This is because the linear SVC is a linear classifier, and they can't learn to distinguish data that isn't linearly separable. (Think of the exclusive-OR example, or the weather prediction example in the second assignment.) Using combination features will mitigate this problem. The NNs on the other hand don't have a problem to learn when the data is linearly inseparable, assuming of course that we have a sufficient number of hidden units.
**(c)** It has the advantage that words which behave similarly from the learning system's point of view can be mapped to vectors which are near each other in the vector space. This approach leads naturally to a simple way to make use of unlabeled text in our training (that is, a semisupervised training setup): we *pre-train* the word embeddings on the unlabeled text using a separate embedding learning algorithm, such as *skip-gram with negative sampling* (implemented in `word2vec`).

### 2.4 Neural network classification and regression

**(a)** What are the most common types of *output units* and *loss functions* used for neural network classifiers and regression models?

**(b)** Using scikit-learn, we train a neural network regression model:

```
MLPRegressor(activation='relu', hidden_layer_sizes=3)
```

We'd like to predict the output for the input $x = \begin{bmatrix} 1 & 2 \end{bmatrix}$. The weight matrices for the two

layers are

$$\begin{bmatrix} 0 & 1 \\ -4 & 1 \\ -2 & 3 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

What is the output of the neural network?

**(c)** Let's instead assume that we have trained a binary classifier:

```
MLPClassifier(activation='relu', hidden_layer_sizes=3)
```

Assuming that the classifier uses the same weights as the regression model above, what is the classifier's output for the input $x$? You may assume that the output unit uses a sigmoid activation.

**(d)** Finally, let's again assume that we have trained a classifier, this time a three-class classifier. For the first layer, we again reuse the weight matrix above. For the output layer, we instead use the following weight matrix:

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(That is: the weights for the first output unit are $\begin{bmatrix} -1 & 0 & 0 \end{bmatrix}$, for the second output unit $\begin{bmatrix} 0 & 3 & 0 \end{bmatrix}$, etc.) What output does the neural network give if we apply it to the instance $x$?

**Solution.**
**(a)** The typical choices are

- binary classifier: a single sigmoid output unit, which gives a probability for the positive class. The loss is then typically the log-loss (known in Keras as *binary cross-entropy*).
- classifier with $N$ classes, where $N > 2$: $N$ output units followed by a softmax to convert into probabilities. In this case, we typically use the cross-entropy loss.
- for regression, we typically use a linear output unit. The squared error is they most commonly used loss function.

**(b)** Let's solve this in a step-by-step fashion:

1. For the three hidden units, the scores are (before applying the ReLU):

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \end{bmatrix} = 2$$

$$\begin{bmatrix} -4 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \end{bmatrix} = -2$$

$$\begin{bmatrix} -2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \end{bmatrix} = 4$$

2. The ReLU will "clip" the negative values. So after applying the ReLU, the outputs of the hidden layer are $\begin{bmatrix} 2 & 0 & 4 \end{bmatrix}$.

3. The output is then

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 & 4 \end{bmatrix} = 14.$$

**(c)** The classifier's output score (before applying the sigmoid) is 14, as in (b). When applying the sigmoid, this gives a probability close to 1.

**(d)** The classifier's output scores, before applying the softmax, are

$$\begin{bmatrix}-1 & 0 & 0\end{bmatrix} \cdot \begin{bmatrix}2 & 0 & 4\end{bmatrix} = -2$$

$$\begin{bmatrix}0 & 3 & 0\end{bmatrix} \cdot \begin{bmatrix}2 & 0 & 4\end{bmatrix} = 0$$

$$\begin{bmatrix}0 & 0 & 1\end{bmatrix} \cdot \begin{bmatrix}2 & 0 & 4\end{bmatrix} = 4$$

It's enough to say that this will give a probability vector close to $\begin{bmatrix}0 & 0 & 1\end{bmatrix}$ (we predict the third class), but let's go through the details anyway:

$$P(\text{first}) = \frac{e^{-2}}{e^{-2} + e^{0} + e^{4}} \approx 0.002$$

$$P(\text{second}) = \frac{e^{0}}{e^{-2} + e^{0} + e^{4}} \approx 0.018$$

$$P(\text{third}) = \frac{e^{4}}{e^{-2} + e^{0} + e^{4}} \approx 0.98$$

### 2.5 Convolutional neural networks

**(a)** What is the idea behind *convolutional neural networks* (CNNs)? Why are they useful for classifying images?

**Solution.**
CNNs "sweep" through the image, looking for "patterns". The key property of CNNs, and the reason this model is useful for most image recognition tasks, is the notion of *location invariance* (or *translation invariance*): a feature detector (convolution) will react the same way to a "pattern" regardless of where in the image it occurs.

CNNs typically consist of a series of convolutional layers. The idea is that the first layers look for small low-level patterns (e.g. small patches of a certain color, or small vertical or horizontal lines). The higher layers look for more abstract patterns, defined in terms of the low-level patterns.

Instead of a CNN, it can be possible to apply a standard feedforward NN operating directly on the image data. This can work, but is generally less effective, since this network would include a much larger number of parameters, and would not have the property of location invariance.

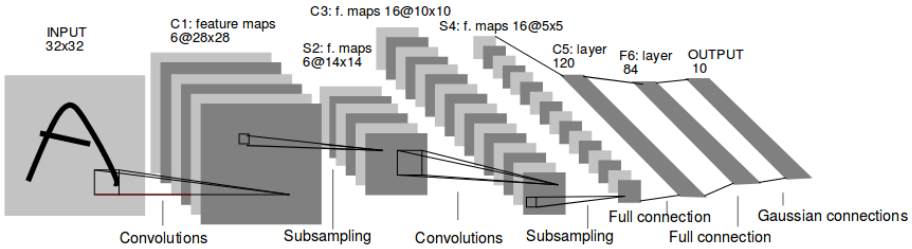
**(b)** Can you think of an application of CNNs where the input is not an image or text?

**Solution.**
CNNs are typically applied when the data can be seen as a "grid" (as an image is a grid of pixels), and we'd like the model to find "patterns" in this grid. This grid is typically two-dimensional but can also be one-dimensional or three-dimensional. So any classification task where you could think of the input as a grid would be fair game. Here is a small sample of examples of applications of CNNs:

- one-dimensional: classifying speech and other types of signals `http://publications.lib.chalmers.se/records/fulltext/249467/249467.pdf`

- two-dimensional: classifying game boards https://int8.io/chess-position-evaluation-with-convolutional-neural-networks-in-julia/
- three-dimensional: classifying molecules, detecting brain tumors http://cs231n.stanford.edu/reports/2017/pdfs/526.pdf

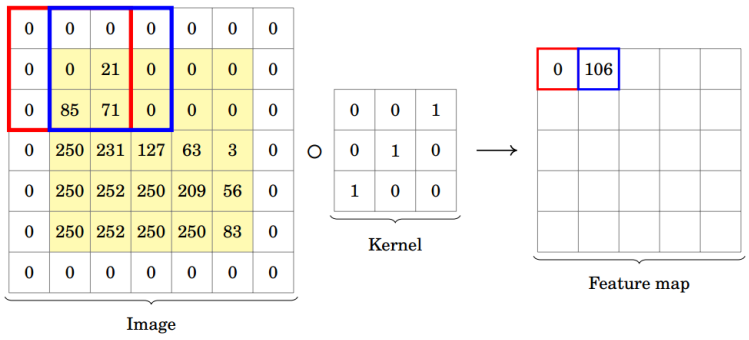
**(c)** Consider the figure below. Can we give an *interpretation* of the first layer (called *C1: feature maps* in the figure)? Can we interpret the second convolution layer (*C3: f. maps*)?




**Solution.**
As discussed in (a), the first layer looks directly for some low-level patterns in the image. For instance, one of the filters may look for small red patches, and another filter may look for a small vertical line. The second layer then defines patterns in terms of the patterns detected in the first layer, e.g. "a vertical line beside a diagonal line".

**(d)** What happens in a convolutional layer? What are the inputs and outputs? (For instance *C1* in the figure.)

**Solution.**
In a convolutional layer, the model "sweeps" through the image, looking for a particular pattern. Specifically, at each position in the image, the model applies a convolutional kernel:



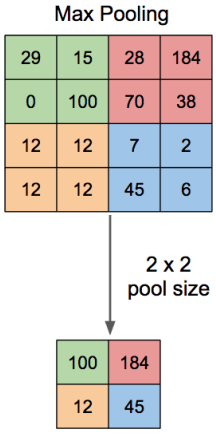

After applying the convolution, there is normally also an activation function: typically the rectified linear unit (ReLU).

The output of a convolutional layer is a *feature map* that shows the locations of where the model has spotted the relevant patterns. If there are $N$ convolutional filters, each of which looks for some pattern, there will be $N$ feature maps.

The lowest convolutional layer operates directly on the image (a three-dimensional "array": height x width x 3 colors). The inputs to a higher convolutional layer is the output from the previous layer (height x width x number of filters).

**(e)** What is a *pooling* layer? What are the inputs and outputs?

**Solution.**
A pooling layer takes a feature map and reduces it to a smaller size. The most common type of pooling is *max-pooling*, which selects the highest value in a small region.




**(f)** For the special case of training image classifiers, can you think of a method that we can use to reduce the risk of overfitting?

**Solution.**
Applying some modifications to an image each time the network observes it during training (e.g. rotations, scaling, color changes) can reduce overfitting and improve the robustness of the model.

**(g)** In Assignment 5, we used the lower layers of a pre-trained CNN to generate features for a classification task. When do you think this can be expected to be useful, and when would you advise against it?
**Solution.**
Most of the popular pre-trained CNN models (VGG, Inception) are trained on the ImageNet database. The images in this collections have been collected from various sources on the web, including user-generated content such as Flickr images.

The ideal scenario is probably that we'd like to classify images that are similar in style to typical web images, and that the categories we are looking for is similar to some categories in ImageNet. On the other hand, if we want to develop an image classifier for a highly specialized domain (e.g. medicine), it seems likely that the images would be too different from typical images occurring on the web (and in the ImageNet training set). But in the end, this is an empirical quesion that would need to be investigated in order to give a definite answer.

## 3 Sequence models

**(a)** What do we mean that our data is sequential?

**Solution.**
It means that the data consists of parts that can be ordered sequentially: in many cases, these parts are ordered *in time*. Examples include text (a sequence of words), time series (a sequence of continuous signals), event logs.

**(b)** What are the main types of machine learning tasks for sequential data?

**Solution.**
The most important machine learning tasks with sequential data are probably the following:

- *Classifying a sequence.* For instance, classifying a document, or the prediction task in question 1.3. For these tasks, it's not always the case that we need a specialized sequential learning method (RNN etc): for instance, for classifying documents, a simple bag of words is often enough.

- *Predicting the next element in the sequence.* For instance, if we have observed some events, what will the next event be?

- *Sequence tagging.* For an input sequence, produce an output sequence of the same length. This is common in natural language processing, for instance in named entity recognition and related tasks.

- *Predicting a sequence.* Examples include machine translation (the input is a sequence, the output is another sequence) and image captioning (input is an image, output is a sequence).

See also the overview in this blog post: `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

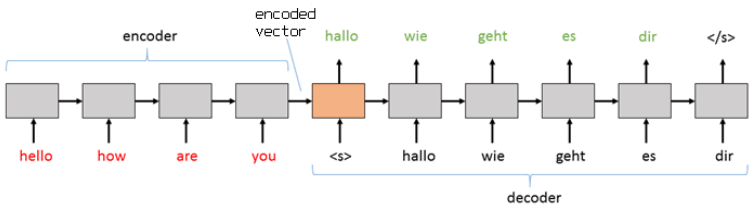
**(c)** Can you mention a neural network architecture that is specifically designed to work with sequential data? On a high level, how does this type of neural network operate?

**Solution.**
A *recurrent* neural network (RNN) processes the sequence in a step-by-step fashion. In each step, it will receive an input and update its state, and (optionally) produce an output.

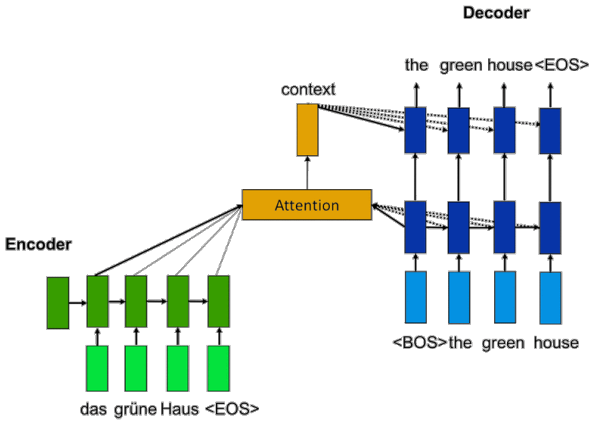
There are a few different types of RNNs. The most popular type nowadays is the *Long short-term memory* (LSTM). For details, see `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Other popular types of RNNs include the Gated Reccurent Unit (GRU). Here is a list of the different types of RNNs available in Keras.

**(d)** Explain on a high level how modern machine translation systems work.

**Solution.**
The basic idea is *sequence-to-sequence learning*, also known as the *encoder/decoder* architecture. In this type of model, a first RNN (the encoder) observes the input sequence (typically, a sequence of word embeddings). A second RNN (the decoder) receives the output of the encoder RNN and generates the new sequence.




Recently, it has been shown that it is also useful to include an *attention model*. While applying the decoder, the attention model can highlight certain parts of the input: that is, it can "focus its attention" on relevant parts of the encoder's states.




# 4 Ethical and legal questions

## 4.1 Interpretability of models

Think of some of the models for *supervised learning* that we have discussed during the course, e.g. linear models for classification and regression, trees, tree ensembles, neural networks. Order these models in terms of *interpretability* according to your own opinion. Motivate your ranking.

**Solution.**

This is of course subjective to some extent. Simpler models are probably more "interpretable" than more complex models. For instance, the weights weight vector in a linear model can be interpreted as "importance values" for each feature (for the positive of negative class). A decision tree is also interpretable as an explanation for the classification, at least when it doesn't grow too large.

Tree ensembles and neural networks seem more opaque. The interpretation of neural networks is a cottage industry in itself and there are many papers devoted to this topic.

Maybe a slightly arbitrary ranking could be: simple trees, linear models, tree ensembles, neural networks? But it also depends on the size of the models.

## 4.2 Machine learning and the law

The following text is taken from Article 15 of the *General Data Protection Regulation* (GDPR), and regulation of the European Union that went into effect in May, 2018.

> *The data subject shall have the right to obtain from the controller confirmation as to whether or not personal data concerning him or her are being processed, and, where that is the case, access to the personal data and the following information:*
>
> *[. . . ]*
>
> *(h) the existence of automated decision-making, including profiling [. . . ] and, at least in those cases, meaningful information about the logic involved, as well as the significance and the envisaged consequences of such processing for the data subject.*

Discuss how you think this affects systems based on machine learning.

**Solution.**

This text concerns automatic systems (whether based on machine learning or not) that have a direct effect on people (e.g. "will I get a bank loan?", "will I get a parole?"). It remains to be seen how the GDPR will be interpreted in practice, but it seems that this clause gives the *right to an explanation* of a decision carried out by an automatic system. It seems likely that this will

require that prediction systems produce decisions that to some extent are explainable, and it will probably not be enough to just blame the decision on the automatic system.

# 5 Unsupervised and semisupervised learning

## 5.1 Unsupervised learning, general questions

**(a)** When would we use unsupervised learning instead of supervised learning?

**Solution.**
Typically, we use unsupervised learning because we don't (yet?) have access to a reasonable amount of labeled data (that is, where there are "outputs" that we'd like our model to imitate). We also tend to use unsupervised learning when we'd like to *explore* the data, and these types of methods can be seen as a way to "summarize" the data (e.g. clustering, dimensionality reduction, modeling distributions).

Sometimes, unsupervised learning is a part of a semisupervised approach (see question 5.2). For instance, in text processing, we often train word embeddings using an unsupervised method, that just requires a large amount of text, no annotated training data.

**(b)** In what sense is clustering inherently more difficult than classification?

**Solution.**
Clustering is to some extent arbitrary. For instance, if we cluster documents into two groups, what is the "correct" grouping? By the gender of the author? The language? The size of the document? The topic? For this reason, clustering often requires a "helping hand", for instance by giving more weight to features we believe may be more important.

**(c)** How can we measure the quality of a clustering system?

**Solution.**
There are two high-level approaches to evaluating clustering systems:

- Inspecting the shape of the clusters. Ideally, clusters are compact and well-separated. For instance, the *silhouette score* is a metric of this type.
- Comparing the clustering to a gold standard. How well does the grouping agree with the grouping defined by the gold standard? The *Rand index* is an example of a metric of this type, but there are several others.

**(d)** Exemplify a clustering algorithm and give a sketch of how it works.
**Solution.**
See for instance

- $k$-means and $k$-means++: `https://en.wikipedia.org/wiki/K-means_clustering#Standard_algorithm` and `https://en.wikipedia.org/wiki/K-means%2B%2B`
- DBSCAN: `https://en.wikipedia.org/wiki/DBSCAN`

## 5.2 Semisupervised learning, general questions

**(a)** What is the goal of semisupervised learning? How is it related to supervised and unsupervised learning?

**Solution.**
Semisupervised learning is related to supervised learning because we'd like to learn to "imitate" the input–output pairs in a labeled dataset, as in supervised learning. It is related to unsupervised learning because there is also a dataset consisting of unlabeled examples. Typically, the labeled dataset is much smaller than the unlabeled one. The goal is to use the unlabeled dataset in some way, in order to improve over a purely supervised approach that just uses the labeled data.

A semisupervised approach may use an unsupervised learning method to learn representations that make it easier to generalize. Word embeddings are an example of this idea: in that case, the word embedding model is often trained on an unlabeled dataset that is several times larger than the labeled part.

**(b)** What are the main types of methods in semisupervised learning?

**Solution.**
On a high level, two important approaches in semisupervised learning are

- Adding new *instances* from the unlabeled data into the labeled data.
- Adding or modifying *features* based on the unlabeled data.

**(c)** Referring to the main types of methods in question (b): give examples of each method, and describe how they work on a high level.
**Solution.**

*Adding instances*. One simple idea is *self-training*. We start by training a classifier on the small labeled part, and then apply this to the unlabeled part. We select some of the newly labeled examples (for instance, the ones where the classifier is most confident) and add them back to the training set. Repeat the process. The risk of this process is that the classifier can reinforce its own mistakes.

A more sophisticated approach is *label propagation*: the unlabeled instance that are most *similar* to some labeled instances will inherit the label form the labeled instances. Repeat until all unlabeled instances have been labeled. (`http://scikit-learn.org/stable/modules/label_propagation.html`)

*Adding features*. Examples of this family of methods include the use of word embeddings and word clusters. In this case, we train the embeddings or clusters on a large amount of unlabeled text, and then use them as features when processing the labeled data.

## 6 Ensembles

**(a)** What is an *ensemble*? What is the intuition behind ensembles? Why would we expect them to work?

**Solution.**
Ensembles in machine learning are *collections* of learning models (classifiers or regressors). The idea is that the different models have different strengths and weaknesses, and that to some extent they can compensate for each other's weaknesses. So ideally, the models in an ensemble are *good* but also *different* (that is, the errors are as uncorrelated as possible).

**(b)** How are ensemble classifiers implemented?

**Solution.**
Most commonly, as *majority voting* (what is the most common output of the sub-classifiers?) or, if the classifiers give some sort of output score, as *averaging*. (The latter is typically also used in ensembles of regressors.)

**(c)** What is *bagging* and what is its purpose?

**Solution.**
As discussed in (a), we'd like the sub-models to be different. Bagging is a method to achieve this. The idea is that if we'd like to use $N$ sub-models (e.g. decision trees), we create $N$ different training sets by sampling randomly from the original training set. By training the models on these different training sets, they will become more different, and the ensemble as a whole will be less overfit.

**(d)** What is a *random forest*?

**Solution.**
There are some different definitions in the literature; in scikit-learn, a random forest is an ensemble of decision trees (for classification or regression), where there are two sorts of randomness involved in the training:

- *Instance bagging*, as described in (c).
- *Feature bagging* or *random subspace learning*. This means that for each of the $N$ training sets, we hide some randomly selected features. This is another approach to ensuring diversity of the different sub-models.

**(e)** What is *boosting*? How do boosting algorithms work in general?
**Solution.**
Boosting is a method to produce ensembles. The sub-models in the ensemble are trained in an iterative fashion. In each iteration, we train a new sub-model that tries to compensate for the mistakes made by the current version of the ensemble.

The classical boosting algorithm is *AdaBoost*, which works by upweighting instances where the latest classifier made a mistake, and downweighting instances that were handled correctly. A more recent algorithm is *gradient boosting*, which generalizes AdaBoost and formulates boosting as a form of gradient descent.