Deep Learning
Lecture 6

Advanced NN Modelling
Dropout
Optimizers explained

Prof. Dr. Rainer Herrler

Phone: 09721/ 940-8710

Email: rainer.herrler@fhws.de

Pictures from Wikipedia / Pixabay

Some Pictures generated with Stable Diffusion

13

Recap of last lecture

- Multinomial Classification
  - Use `tf.one_hot(labels, num_classes)`
  - or `tf.argmax(predictions)`
  - Use “softmax+categorical_crossentropy”
- Repetition on means for evaluating classification problems
  - `sklearn.metrics.precision_score`
  - `sklearn.metrics.recall_score`
  - `pd.crosstab(ytrue, ypred)`

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

- How to choose epochnumber & batchsize
  - Effect on the learning progress
- What to vary in networks
  - Batch Normalization Layers
  - Activation Functions (Standard: relu and sigmoid)
  - Weight Initialization (Standard: Xavier aka Glorot)

Some ways for more convenient experimentation

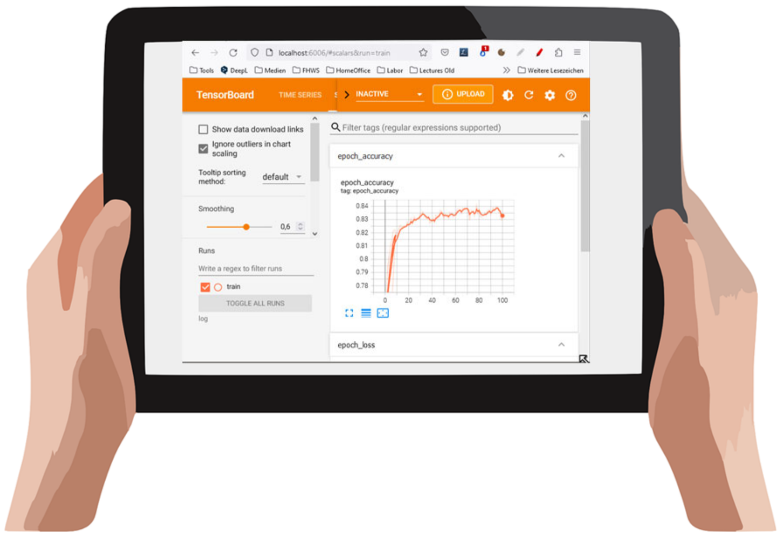
DL_008_Convenience.ipynb

Some ways for more convenient experimentation

- **Flatten layer**
  - Some layers just accept onedimensional training samples
  - The `Flatten`-layer flattens multidimensional samples
- **Loss Function for Categorial input**
  - Use of `SparseCategoricalCrossentropy` makes one-hot-encoding of labels obsolete
  - Work is shifted to the loss calculation
  - We still need one output-node per category (+argmax to convert probabilities to predicted classes)
- **Specify Validation-Data for the fit-Function**
  - If we set `validation_data` Validation is run after each epoch
  - We can track test and training-accuracy wile we are optimizing
  - Can be used to plot graphs or decide on early stopping
- **Creation of Sequence-Model with the constructor**
  - Add a list of layers in the constructor
  - Variant to use `model.add(…)`

Tensorboard

- Plotting accuracy or loss with matplotlib has **some drawbacks**
  - History is returned after training
  - Live view would be helpfull
- **Usage steps**
  - conda install -c conda-forge tensorboard
  - tensorboard --logdir log
  - Add “callback” for fit-method
  - Open http://localhost:6006
- **Features**
  - Shows progress while training
  - Further tool support for various advanced experimentation tasks (Parameter optimization, model inspection)
- **Alternative**
  - Livelossplot li leightweight and can be used directly in the jupyter notebook



DL_009_TensorboardVsLivelossplot

What are “Callbacks” ?

A callback is a function (in an callback object) that can be called during the training process (e.g at the end of an epoch, or an iteration)

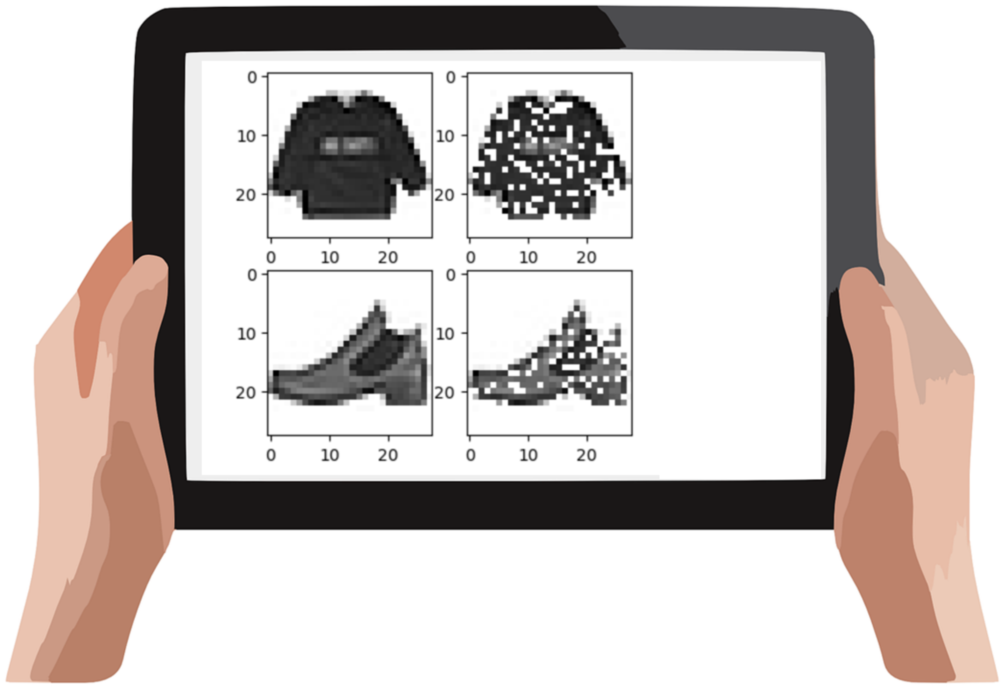
Some common use cases for callbacks in TensorFlow include:

• Saving the model weights after each epoch	(`ModelCheckpoint`)
• Visualizing the training progress using TensorBoard	(`TensorBoard`)
• Learning rate scheduling	(`LearningRateScheduler`)
• Early stopping to prevent overfitting	(`EarlyStopping`)

Any custom callback can be implemented by extending the base class `Callback`

Regularization

DL_010_Regularization.ipynb



Question (ML Recap):

What is the aim of regularization ?

What did we learn in the last notebook

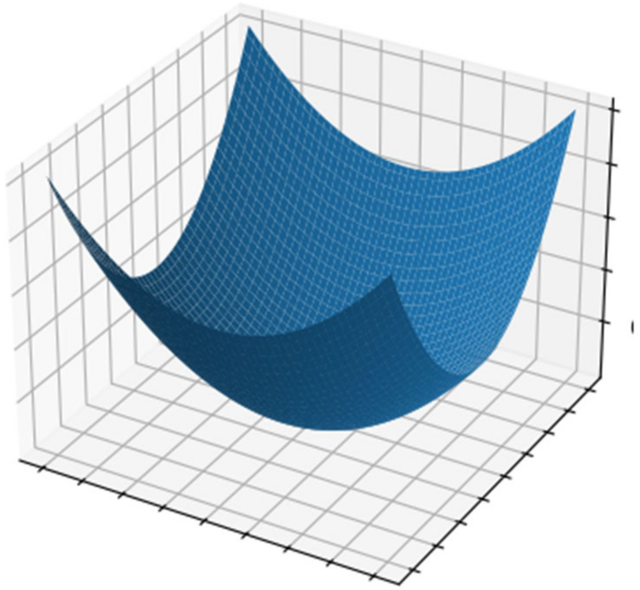
- Regularization helps to prevent overfitting
- Realizing Dropout
  - How it works
    - During Training Dropout randomly sets input activations to zero
    - Dropout can be applied to input as well as in-between inner layers
    - The model is forced to not rely on a single feature to perform good → It gets more robust to noise or variance
    - During inference the Dropout layer is inactive to make best predictions
  - Drawbacks
    - More epochs may be necessary to converge
    - Dropout rate has to be found (one more parameter to optimize)
  - If possible its preferable to collect more data to fight overfitting, if data is limited dropout might help.
- Realizing L2 or L1 Regularization
  - Regularization can be done for each layer individually

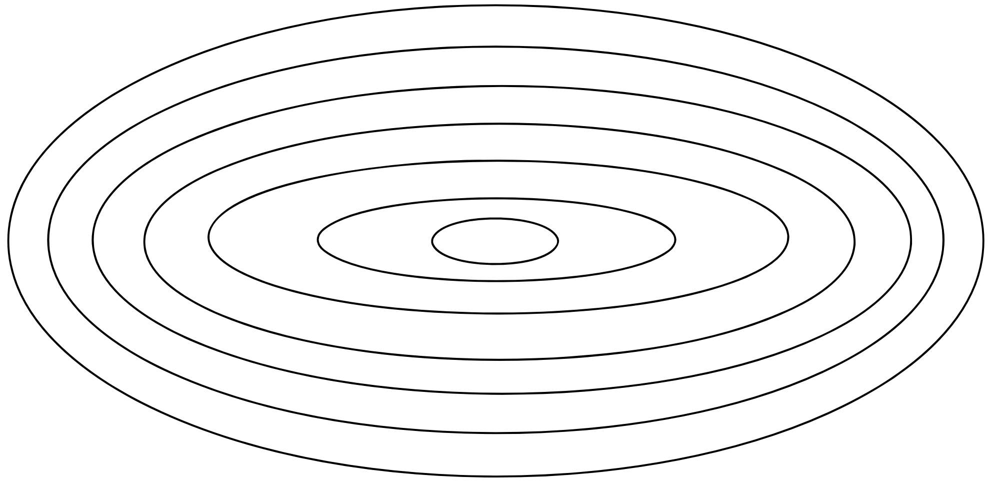
- What else can be done to avoid overfitting ?

Exercises with CIFAR-100

1. Create a suitable network (enough layers/nodes but not too many)
2. Add Batch Normalization to different parts of the network
   1. Classical approach between
   2. Alternative approach between
3. Test if different positions of Dropout lead to different results (is dropout better in early layers or in later layers)
4. Find out empirically if larger or lower values for the L2-regularization are leading to a less complex model.

Optimizer - Intuition





Three most important enhancements of gradient descent:

- Momentum
- RMS-Prop
- Adam

Optimizer

- What are optimizers ?
  - Optimizers are a enhanced strategy for updating weights
- Why optimizers ?
  - Pure gradient descent is sometimes ineffective and it can take a long time to converge
  - Pure gradient descent may overshoot the local minimia
- Idea
  - Don’t update just based on the current snapshot (gradient and learningrate) but also on the history of updates

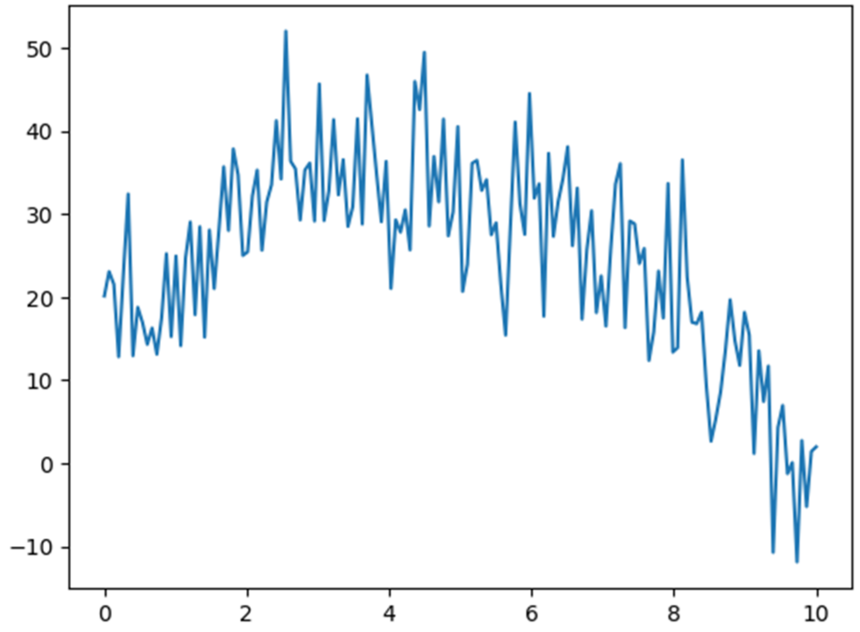
Optimizer – introducing “Momentum”

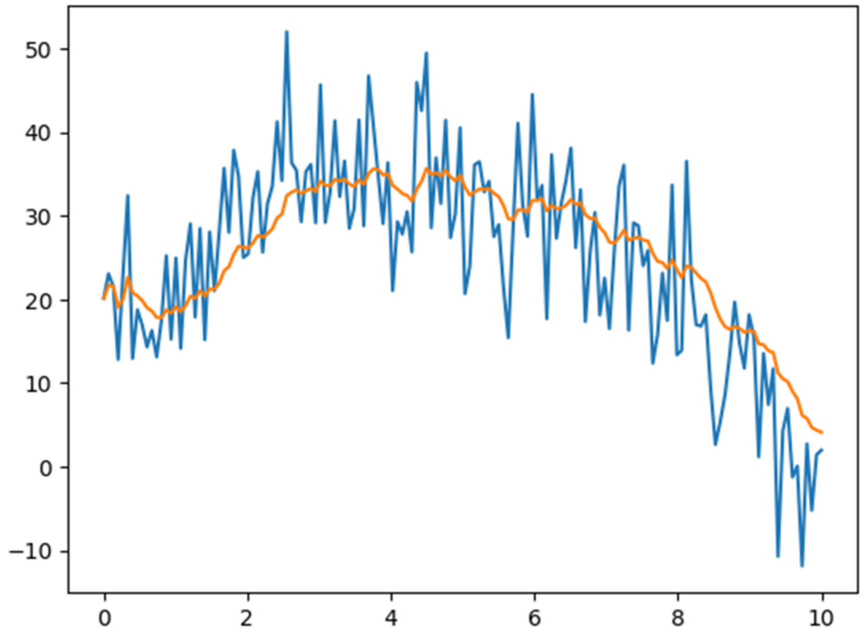
- Don’t update just based on the current snapshot (gradient and learning rate) but also on the history of updates

- Remember gradient of the past to
  - Preserve movement in the same direction
  - Prevent movement in other directions
  - Damp oscillations

- How can this be done mathematically?

Realizing Momentum

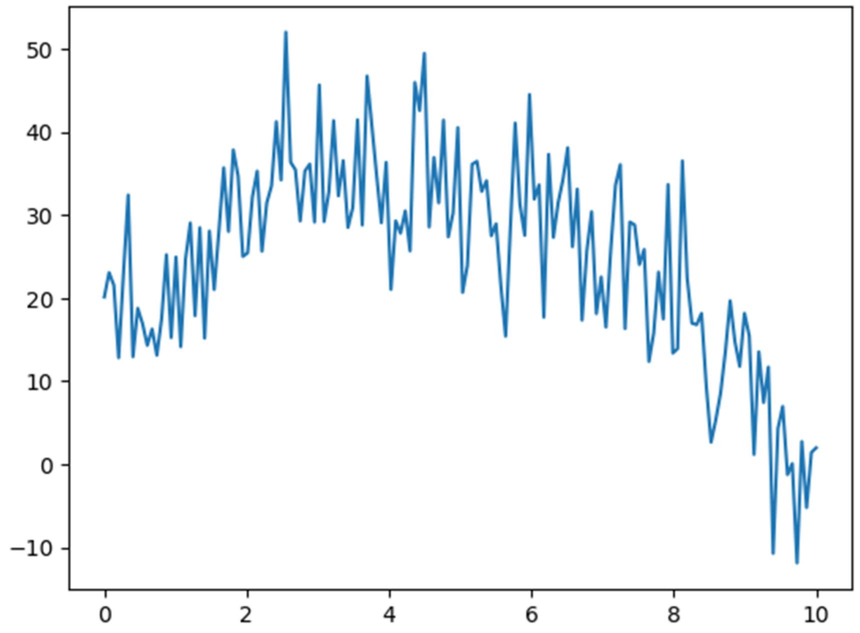


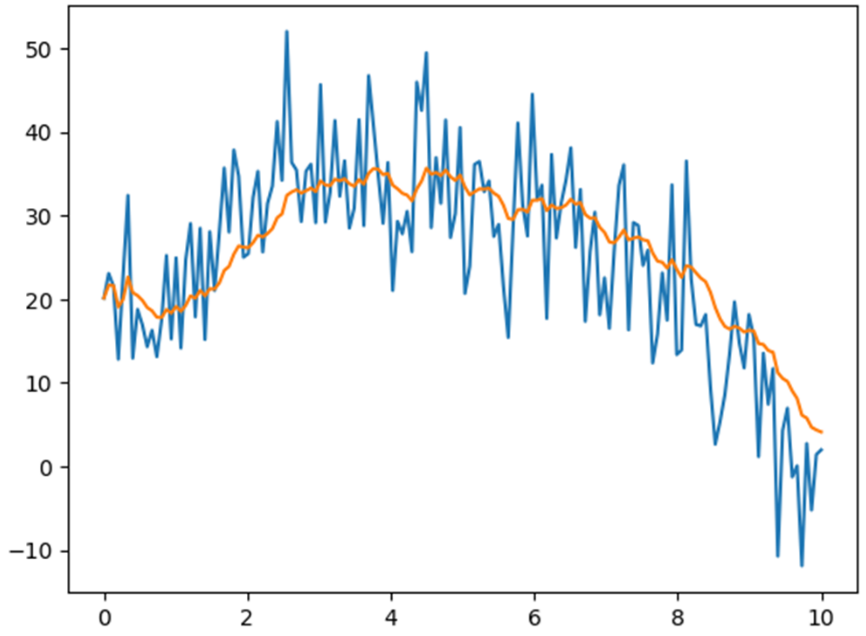


Idea:

- We calculate a “moving average” with a sliding window of relevant values
- This cuts of peaks
- Next step to I weight the newest point more than older points
- → EWMA is short for “Exponentially Weighted Moving Average”

Realizing Momentum with EWMA





Formula for a smoothed curve:

$$av(t) = \beta \cdot av(t-1) + (1-\beta) \cdot f(t)$$

Applied to weight updates:

$$\theta_i = \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta_i)$$

How to use Momentum in Keras

```
from keras.optimizers import SGD

model = Sequential()
sgd = SGD(lr=0.0001, momentum=0.8) # defaults are lr=0.01 momentum=0.0
model.add(Dense(100, activation="relu", input_shape=(784,)))
model.add(Dense(len(class_names), activation="softmax"))
model.compile(optimizer=sgd, loss="categorical_crossentropy“)
print(model.optimizer)
```

Optimizer – RMS Prop

- Different mathematical approach to restrict oscillations

$$E[g^2](t) = \beta E[g^2](t-1) + (1-\beta)\left(\frac{\partial}{\partial \theta_i} J(\theta_i)\right)^2$$

$$\theta_i(t) = \theta_i\,(t) - \frac{\alpha}{\sqrt{E[g^2]}} \frac{\partial}{\partial \theta_i} J(\theta_i)$$

- “Adaptive learning rate” taking into account how much a parameter was already changed in the past
- Exponential Weighted moving Average is also used to dacay older gradients
- Inventor: Geoffrey Hinton

How to use RMSProp in Keras

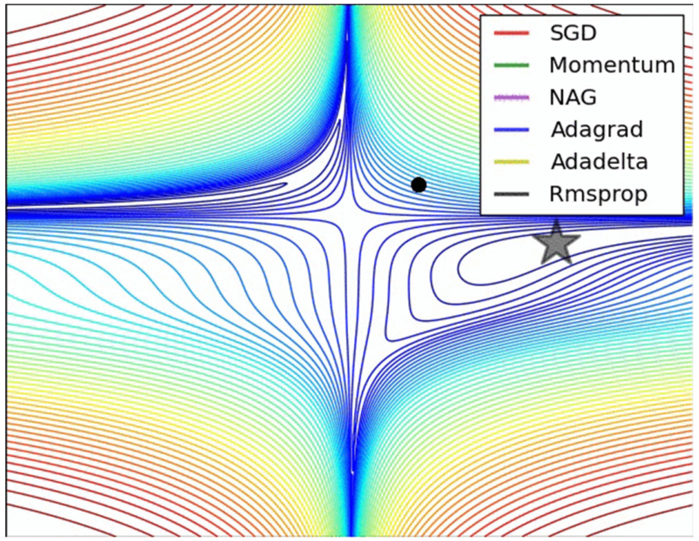
```
from tf.keras.optimizers.experimental import *

model = Sequential()
#opt = SGD(learning_rate=0.0001)
opt = RMSprop(learning_rate=0.0001)
model.add(Dense(100, activation="relu", input_shape=(784,)))
model.add(Dense(len(class_names), activation="softmax"))
model.compile(optimizer=opt, loss="categorical_crossentropy")
```

Optimizer Adam

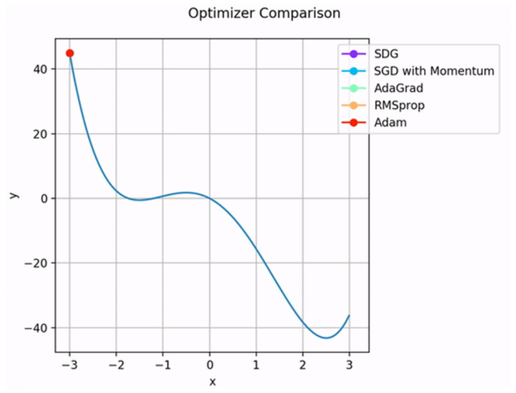
- Invented in 2014
- Combines Ideas of Momentum and RMSProp (Exact formula not presented here)
- Slightly better than RMSProp
- Works best on a great variety of problems
- Finding the proper learning rate is not so difficult any more

Visualization of different optimizers

https://raw.githubusercontent.com/jeffheaton/t81_558_deep_learning/master/images/contours_evaluation_optimizers.gif

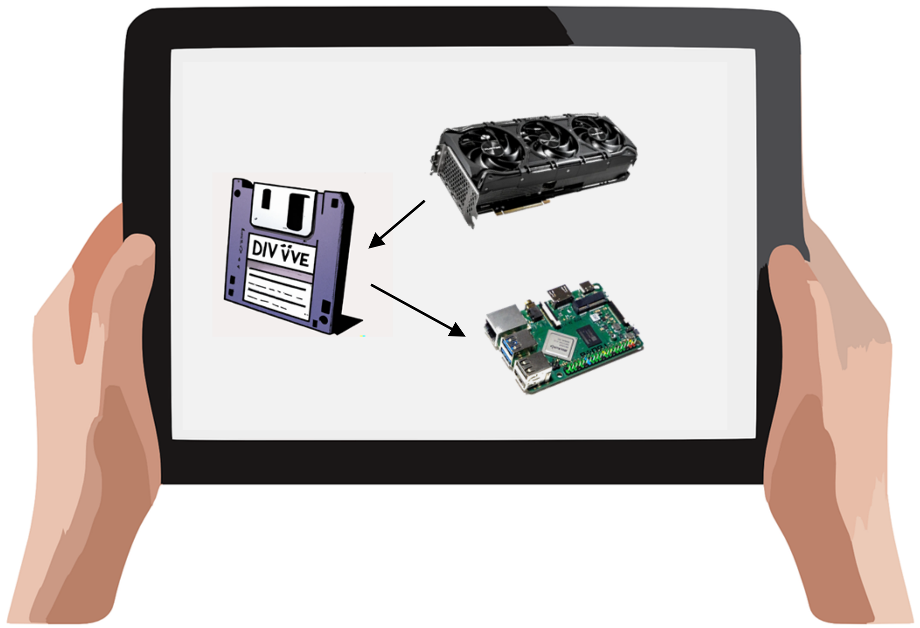


https://towardsdatascience.com/complete-guide-to-adam-optimization-1e5f29532c3d



Saving and Loading Models

DL_011_Saving and Loading.ipynb



Training needs compute!
Inference is fast!

- In robotics we can use powerful hardware to train models and comparatively weak hardware for applying the models!
- Saving and loading can be used on any computer where same Python/Tensorflow versions are running (E.g a raspberry pi)
- With “Tensorflow light” models can even be cross-compiled to run on microcontrollers (E.g an Voice command recognition on a ESP32)

Saving and loading models

1. How to save and load model weights
   - Commands
     - `model.save_weights(directory)`
     - `model.load_weights(directory)`
     - Different formats with optional parameter e.g. `save_format="h5"`
   - Model architecture may not be changed
   - Compile is necessary after loading
   - Continued training with further epochs is possible after loading
2. Save and load an entire model
   - What is saved
     - Weights
     - Model Architecture
     - Training Configuration (whats given in model compile)
     - Optimizer and states
   - Compile is not necessary before using
   - Commands
     - `model.save(directory)`
     - `keras.models.load_model(directory)`

Summary

- Various convenience Stuff
  - Flatten layer (reshaping input can be omitted)
  - SparseCategorialCrossEntropy (one hot encoding obsolete)
  - Set validation-data in fit step (Evaluation done during training)
  - Tensorboard / Livelossplot
- Regularization in Tesorflow
  - Dropout
  - L2-Regularization
  - (Batch Normalization)
- Optimizers
  - SGD
  - Momentum
  - RMSprop
  - Adam
- Saving and Loading Models