Deep Learning Tips and Tricks (based on the book *Deep Learning with Python* by Francois Chollet)

Created by Aldo Zaimi

May 10, 2019

- **SECTION 1: MACHINE LEARNING AND NEURAL NETWORKS**
  - **The universal workflow of machine learning:**
    - Define the problem and assemble a dataset.
    - Choose a measure of success (performance metric).
    - Decide on an evaluation protocol (validation set).
    - Prepare the data (preprocessing).
    - Develop a model that does better than a baseline.
    - Experiment with more complex models (add layers, bigger layers, more epochs).
    - Regularize the model and tune the hyperparameters.
  - **The train/validation/test split:**
    - Training set: used to train your model.
    - Validation set: used to evaluate your model during development and to select the best hyperparameters.
    - Test set: used to evaluate the performance of your final model (train on both the training and validation sets).
    - Another alternative to the standard hold-out validation is the K-fold validation or the iterated K-fold validation with shuffling.
  - **How to do K-fold cross-validation:**
    - Split the training data into K partitions (typically 4 or 5).
    - Train the network K times on $(K-1)$ partitions and evaluate performance on the remaining partition.
    - Compute the average of the K validation scores obtained.
    - Another alternative: iterated K-fold validation with shuffling (the K-fold validation is performed multiple times and the data is randomly shuffled between each iteration).
  - **When working with small training datasets:**
    - Use K-fold cross-validation instead of the standard train/validation split.
    - Avoid using large neural networks (i.e. more layers and/or units) to prevent overfitting.

- **Gradient descent algorithm:**
  - Stochastic gradient descent (SGD): error calculation and model update for each example in the training dataset.
  - Batch gradient descent: error calculation for each example, but model updated after all training examples have been evaluated (i.e. after every epoch).
  - Mini-batch gradient descent: error calculation and model update after each small batch (i.e. batch size) in order to find a balance between the robustness of SGD and the efficiency of batch GD.
- **Mini-batch training:**
  - A small set of samples that are processed simultaneously by the network (i.e. a single gradient-descent update is computed after each mini-batch).
  - Batch size: the number of training samples to work through before the model updates its internal parameters.
  - Number of epochs: the number of complete passes through the training dataset.
  - Use a batch size that is a power of 2 (usually between 8 and 128).
- **How to configure mini-batch size:**
  - Start with a default batch size of 32 (based on literature).
  - Compare validation error learning curves with different batch sizes.
  - Batch size and learning rate should be tuned after tuning all the other hyperparameters.
- **When working with multiple numerical input features:**
  - Use feature-wise normalization (substract the mean and divide by the standard deviation).
  - Use the training set mean and standard deviation to normalize the test dataset.
- **To prevent overfitting:**
  - Get more training data.
  - Reduce the capacity of the network.
  - Add weight regularization.
  - Add dropout.
  - Use data augmentation.
- **When doing regression:**
  - Do feature-wise normalization if you have multiple input features.
  - The last layer should have one unit with no activation function.
  - The loss function should be the mean squared error.
  - A common evaluation metric used is the mean absolute error.
- **When doing binary classification:**
  - The last layer should be a Dense layer of one unit and using a sigmoid activation (probability between 0 and 1).
  - The loss function should be the binary cross entropy.
- **When doing single-label multiclass classification:**
  - Use one-hot categorical encoding to vectorize the label data.

    - The last layer should be a Dense layer with N units, where N is the number of output classes (categories).
    - The last layer should use a softmax activation.
    - The loss function should be the categorical cross entropy.
    - Another alternative is to encode the labels as integers and use the sparse categorical cross entropy as loss function.
  - **Adding weight regularization:**
    - The objective is to force the network to use small values for its weights by adding to the loss function a cost associated with having large weights.
    - L1 regularization: cost using the absolute value of the weight coefficients.
    - L2 regularization: cost using the square of the value of the weight coefficients.
  - **Adding dropout:**
    - Dropout rate: the fraction of the features that are dropped (usually between 0.2 and 0.5).
    - At test time: the layers output values should be scaled down by a factor equal to the dropout rate.

- **SECTION 2: CONVOLUTIONAL NEURAL NETWORKS**
  - **To determine if a CNN architecture is the right choice:**
    - Make sure that your problem can benefit from learning translation invariant patterns.
    - Make sure that your problem can benefit from learning spatial hierarchies of patterns.
    - All input samples need to be resized to the same format.
  - **When designing convolutional layers:**
    - Choose the size of the patches extracted from the inputs (typically 3×3 or 5×5).
    - Choose the depth of the output feature map (i.e. the number of filters computed by the convolution).
    - Determine if padding will be used.
    - Determine if a stride different than 1 will be used.
  - **When designing pooling layers:**
    - Determine what type of pooling to use (typically max pooling).
    - Determine the window size of the pooling operation (typically 2×2).
  - **How to determine the number of trainable parameters in a CNN:**
    - Input layer: no trainable parameters.
    - Convolutional layers: $(n \times m \times l + 1) \times k$, where $(n \times m)$ is the filter size, l is the number of input feature maps and k is the number of output feature maps. The *+1* term in the equation takes into account the bias terms.
    - Pooling layers: no trainable parameters.
    - Dropout layers: no trainable parameters.
    - Fully connected layers: $(n + 1) \times m$, where n is the number of input units and m is the number of output units. The *+1* term in the equation takes into account the bias terms.

- **When using data augmentation:**
  - Make sure all data augmentation transformations yield believable-looking images (based on your application).
  - Example of generic transformations: rotation, shifting (in both directions), shearing, zooming, horizontal and/or vertical flip.
- **When using dropout in CNNs:**
  - Only use dropout on the fully-connected layers of the network (not in the convolutions).
  - Dropout could also be applied between a flattening and a fully-connected layer.
- **Using pretrained CNNs:**
  - Use an original dataset that is large enough and general enough.
  - Two methods: feature extraction and fine-tuning.
  - Feature extraction: use the same trained convolutional base, but train a new densely connected classifier on top of it (randomly initialized).
  - Fine-tuning: unfreeze some of the top layers and jointly train the new added parts of the network and the unfreezed top layers.
  - Fine-tuning steps: (i) add custom network on top of the already-trained base network; (ii) freeze the base network; (iii) train the added part of the network; (iv) unfreeze some layers in the base network; (v) jointly train the unfreezes layers and the newly added parts.
  - The layers chosen for fine-tuning should be the more specific ones because they usually have more specialized features.
- **Visualizing intermediate CNN outputs (intermediate activations):**
  - Useful for understanding how CNN layers transform their inputs.
  - Method: use a sample image and output all the activations of all convolution and pooling layers of the model (scroll through the activations by changing the layer n and the channel m: $[n, :, :, m]$).
  - When plotting all activations together in a grid, make sure you preprocess them in terms of intensity to help visualize them.
  - The first activations should represent more generic features such as edge detectors, while the last activations should be more abstract, higher-level and sparse (empty activations meaning a specific patten is not found in the input image).
- **Visualizing CNN filters:**
  - Useful for understanding what visual pattern or feature each filter is receptive to.
  - Method: start from a black input image and apply gradient descent to the value of the input image of the CNN so as to maximize the response of a specific filter.
- **Visualizing heat maps of class activation in an image:**
  - Useful for understanding which parts of an image were identified as belonging to a given class (indicates how important each location is with respect to a given class).
  - Method: take the output feature map of a given convolution layer for a given input sample, and weight every channel in that feature map by the gradient of the class with respect to the channel.

* Visualizing the heat map on the original sample helps understanding why the network thought this image was a given class, and where are the elements of that class located in the picture.