Coursework 4: Music genre classification with the Million Song Dataset

Raúl Soutelo Quintela (s1685982)

March 21, 2017

Abstract

Deep learning is currently used to address many different classification problems in industry. It is a continuously evolving topic in research as well. Different architectures are chosen depending on the input data distribution to process data consequently and achieve high performances. In this report, different architectures will be evaluated to predict the genre of a song by the features extracted from segments of the song. Convolution in time will be used to do sequential modelling of the data. Afterwards, a combination of neural networks will be evaluated to improve the accuracy of the best model. A novel approach was developed to increase the performance of a model and compared with the accuracy obtained by combining the predictions of different models. The justification and advantages are then discussed.

Contents

1 Introduction

This report is the continuation of coursework 3 where a classification problem was addressed. The problem consists of predicting the genre of song function of a number of features extracted from different segments of a song. The dataset used is The Million Song Dataset. In this coursework sequential modelling will be implemented to improve the performance of the model.

Music information retrieval (MIR) is the science of retrieving information from music. It has many applications ranging from recommender systems to music generation. Different machine learning techniques have been used in MIR to tasks such as extracting features from the audio music [2]. In this coursework the goal is building a model able to predict the genre of a song function of some previously extracted features. There are two different task: the first one consists of predicting the genre out of 10 and the second one out of 25.

The Million Song Dataset consists of a set of precomputed features and labels for a million songs. Amongst the available features, there are 12 chroma features, 12 timbre features and various measures of the loudness. The most detailed information is related to a 'segment'. A segment is an automatically identified music event of variable length. The labels are features such as artists, album names, duration or other features extracted with a music analysis platform called The Echo Nest.

A data provider is given for the purpose of this coursework. This data provider encapsulates in one data structure the 12 chroma features, the 12 timbre features and a measure of the loudness for each segment. Each song consists of a fixed number of segments (120) to allow easy integration into feedfodward models. A preprocessing of the data is done to let each feature of each segment to have mean equals to zero and standard deviation equals to 1. The genres for the 10-genre task and the 25-genre task are provided by CD2C tagtraum genre annotations and MSD Allmusic Style Dataset respectively. The numbers of songs for each class, used for training and validation, is 5000 for the 10-genre task and 2000 for the 25-genre task.

Deep neural networks have many local optima and many combinations of the weights that perform well for the same model. Optimizing these large neural networks is usually hard [9]. When a model with a high representational power is needed and a fully-connected feedforward network is used, the number of weights could be huge. A model able to capture the structure of the data while maintaining low number of parameters will be easier to optimize, computationally more efficient and less susceptible to overfitting. Following this principle convolutional networks and recurrent neural networks are now the state of the art in many image detection and sequential modelling tasks [10][11].

The main motivation of this coursework is to use convolution layers to capture the sequential structure of the feature vectors of the segments. Having explored different architectures, which exploit different properties of the data, also motivates model combination. In this coursework the difficulties encountered when optimizing these combinations of models together are evaluated.

The research questions investigated in this coursework are:

- **Regularization in the baseline:** The baseline, which is shown in Figure 1, consists of two convolution, a fully connected and a output layer. We evaluate the need of regularization in all layers.

- **Sparse feature representation:** In the baseline, the two convolutional layers are intended to learn combinations of the features. A more sparse combination of these features is analysed.

- **Sequential modelling:** Convolution in time is implemented to model the sequential structure of the data. The influence of different pooling layers is then explored.

- **Model combination:** The models obtained along the coursework are combined to achieve a higher accuracy.

- **Correcting model:** An approach to improve the performance of an already trained model is proposed. Its relation with model combination and with the optimization is discussed.

2 Methods

In this section TensorFlow (TF), the framework used to implement these experiments, will be presented as well as the TF functions used along the coursework. The optimizer and the non-linearity will be also explained here, since they are used for all experiments.

TensorFlow is an open source software library, developed by the Google Brain team, used for numerical computation using data flow graphs. It allows work with complex models without having to implement the forward and backward propagation. TensorFlow is used via a Python interface while an efficient implementation in C++ runs the computational graph operations.

A skeleton of the code was given along with the data providers. Some TF functions included are: tf.matmul, tf.train, tf.reduce_mean, tf.equal, tf.nn.softmax_cross_entropy_with_logits. Additional TF functions implemented along the coursework are tf.reshape, tf.nn.bias_add and tf.nn.conv2d for the convolutional layers; tf.nn.l2_loss, tf.reduce_sum and tf.abs for regularization or tf.nn.avg_pool and tf.nn.max_pool for the pooling layers. The functions tf.SparseTensor and tf.sparse_tensor_to_dense were used to set some of the elements of a tensor to a desired value. This was needed to manipulate some weights element-wise when evaluating the sparse feature representation.

In order to implement model combination and the correcting model, some models needed to be previously trained and the weights and biases loaded into a new model. In most experiments, these weights and biases are constrained to these values. In order for TensorFlow not to update these values and do backward propagation they were set to tf.constant instead of tf.Variable.

The optimizer used in this coursework is the Adam optimizer[1]. It is implemented in the TF function tf.train.AdamOptimizer. Adam optimizer has a decreasing learning rate as RMSProp does but it also has momentum. In coursework 3 the influence of the variation of the value of the learning rate was evaluated since the default value was too large when a simple model was used. In this coursework all models are deeper and the default parameters work well.

The non-linearity used was ReLU. It is implemented in TensorFlow with the TF function tf.nn.relu. It has some advantages over tanh or sigmoid such as there is no positive saturation which result in slower learning and that is computationally efficient. ReLU has the problem of zero gradient when negative input which leads to no learning. ReLU was proven to work better than tanh and sigmoid for this task.

3 Baseline

The baseline used for this report was obtained in coursework 3. It does not assume any temporal relation in the data. If the segments are shuffled, the performance would be the same. The goal of this report is improve the performance obtained with the baseline by taking into account the temporal relation of the data.

The baseline is shown in Figure 1. It consists of 2 convolutional layers with 50 kernels each one and a fully connected layer of 200 hidden units. The kernels cover one feature vector at a time, namely each kernel of the first convolutional layer is supposed to learn one combination of the features of each segment. The second convolutional layer is intended to learn combinations of the activations to the patterns learned by the 50 kernels of the first convolutional layer.

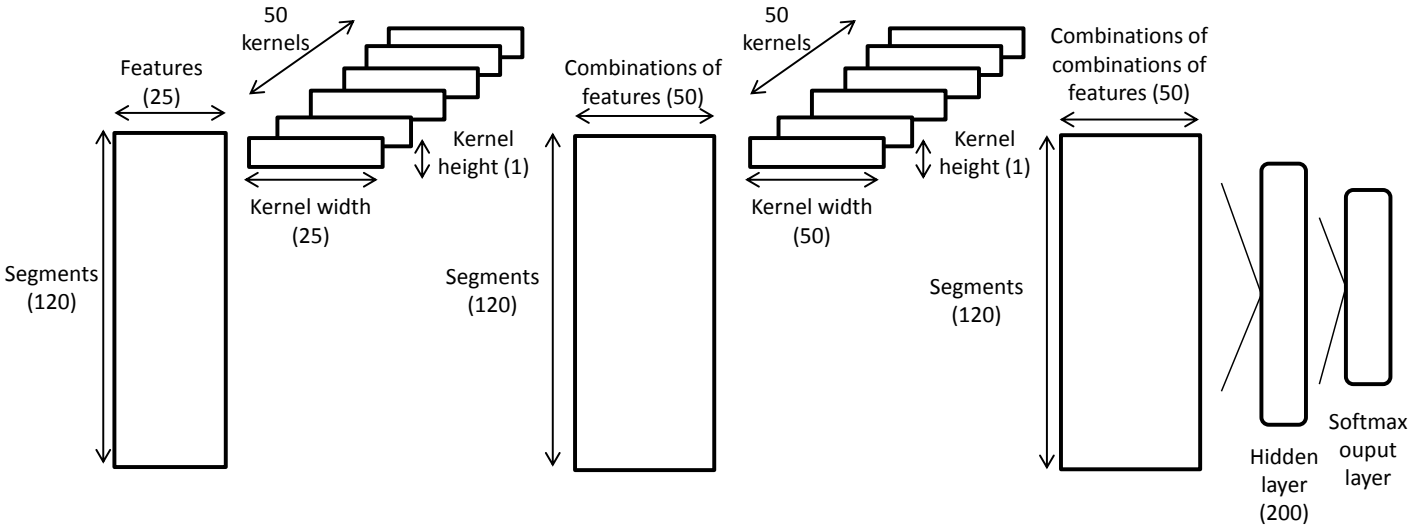

Figure 1: Model architecture of the baseline. It consists of two convolutional and a fully connected layers.

This architecture is slightly different to the image detection approach where the activations of the different kernels go to different features maps. The second convolutional layers is intended to look for combinations of the activations of the feature vectors to the patterns learned by the kernels of the first convolutional layer. The input data is converted into a 2-D image with just one input channel before the second convolutional layer.

In this section, before doing sequential modelling, two topics will be explored. First, the need for regularization in each layer will be evaluated. Then, a more sparse representation of the combinations of the features is explored.

3.1 Regularization

Regularization is a technique used to avoid overfitting when the complexity of the model is high. Limiting the increase of complexity both in the number of free parameters and its size avoid that the model perfectly fit the training data and generalize poorly in unseen data. The regularization technique used to prevent overfitting in this model is L2 regularization. L2 regularization prevents the weights in the model from being really large by adding and additional term in the error function. The error function is shown in equation 1. The coefficient that gave the best result is $\lambda = 0.01$.

$$E_\lambda(w; y, x) = \sum_{n=1}^{N} [y^{(n)} - f(x^{(n)}; w)]^2 + \lambda \sum_{k=1}^{K} w_k^2 \tag{1}$$

In order to do a more in-depth exploration of the regularization, the degrees of freedom of each layer are evaluated:

- **First convolutional layer:** number of weighs: 50(kernels)*26(features + bias) = 1300.
- **Second convolutional layer:** number of weighs: 50(kernels)*51(features + bias) = 2550.
- **Fully connected layer:** number of weighs: 2550(previous layer's hidden units)*200(hidden units) + 200 = 510200.

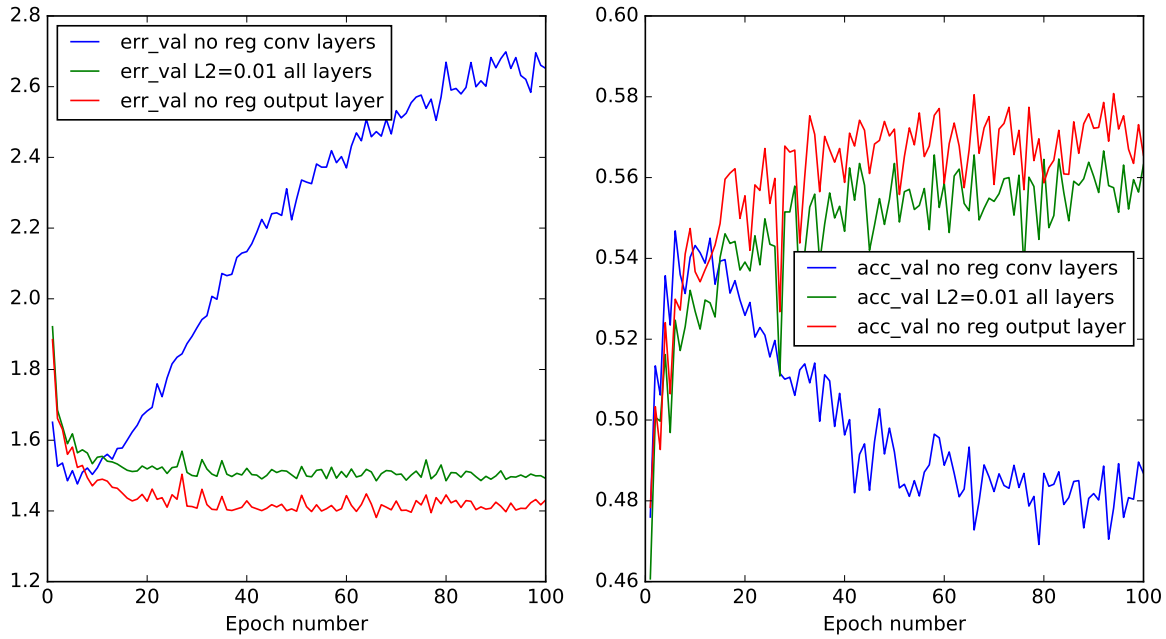

Figure 2: Learning process when the convolutional or the output layers are not regularized.

- **Output layer:** number of weighs: 200(previous layer's hidden units)*10(classes) + 10 = 2010.

The number of free parameters in the fully connected layer is much larger than in both the convolutional and the output layers. Therefore more regularization is needed here. Figure 2 shows the learning process when either the convolutional layers or the output layers are nor regularized.

It is shown in Figure 2 that removing the regularization in the convolutional layers results in overfitting, whereas not regularizing the last layer does give better results. The convolutional and the output layer have the same order of magnitude of parameters. Therefore this is not just related with the number of degrees of freedom of each layer.

The absence of regularization of the last layer let the weights be large. When a weight is too large the probability of one class could be highly conditioned on the influence of just one hidden unit of the previous layer rather than on the influence of all hidden units together. It is expected that whether it is worth regularizing the last layer or not depends on both the dataset and the model that is being used. As all other hyperparameters the value of the L2 regularization coefficient will be the one that gives the highest accuracy in the validation set, so that no regularization will be applied in the last layer.

3.2 Sparse feature representation

The objective of the convolutional layers is to learn combinations of the feature vectors that appear across the segments. However it is not clear if the kernels learn combinations of all the features of just a subset of them. A more sparse representation of the combinations is here desired. Decreasing the degrees of freedom is a regularization method as well. This is also motivated by the different origin of the features: 12 are chorma features, 12 are timbre features and the other is loudness.

L1 regularization, shown in equation 2, penalizes the sum of the absolute value of the weights. L1 regularization switches off weights that do not improve the performance in the

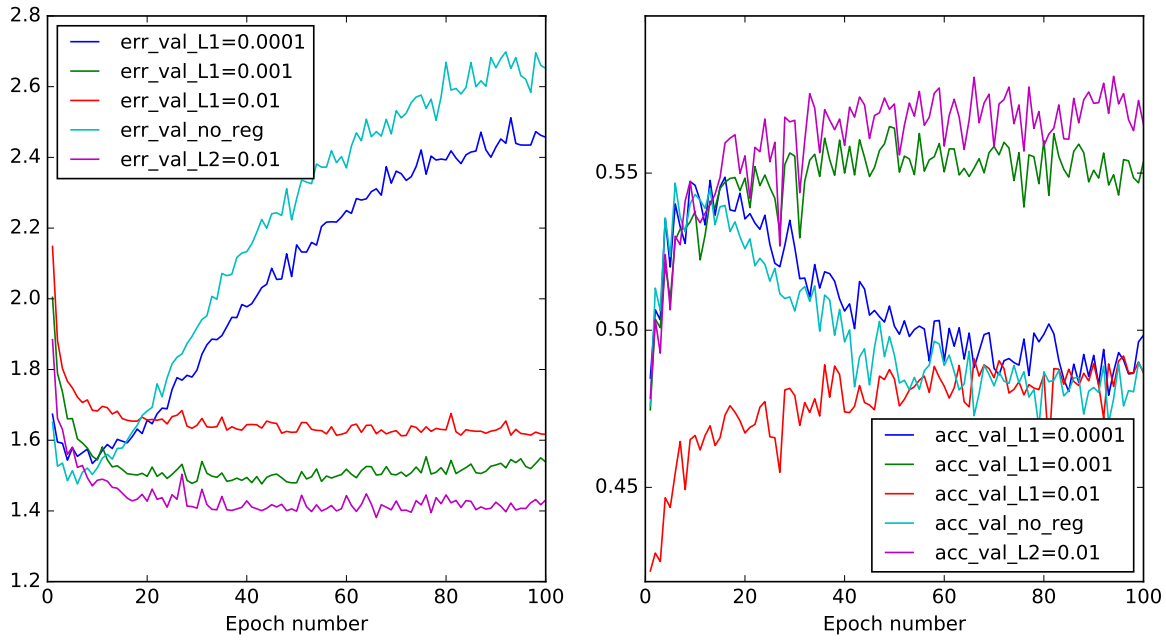

Figure 3: Comparison of the learning process applying different values of L1 regularization in the convolutional layers.

training set significantly whereas L2 decreases its value. L2 regularization penalizes more than L1 regularization really large weights. L1 encourages sparsity and L2 regularization discourages large weights.

$$E_\lambda(w; y, x) = \sum_{n=1}^{N} [y^{(n)} - f(x^{(n)}; w)]^2 + \lambda \sum_{k=1}^{K} |w_k| \tag{2}$$

Figure 3 shows the learning process applying L1 regularization in the convolutional layers instead of L2 regularization.

It is shown in Figure 3 than none of the values of the regularization coefficient for L1 regularization performs better than L2 regularization. L2 regularization is proven to work well as regularization technique in convolutional layers, specially when the number of kernels is high, which is the case for this model [7].

Figure 4 shows an histogram of the weights of both convolutional layers when applying L1 ($\lambda = 0.001$) and L2 ($\lambda = 0.01$) regularizations. These are the coefficients that provided the best performance in Figure 3.

It is shown in Figure 4 that L1 regularization switches off more parameters than L2 regularization. One problem that L1 regularization has is that it shrinks too much the non-zero weights. One alternative to avoid this problem is to fit a model with L1 regularization to detect the non zero weights and then fit the non-zero weights again with a L2 regularizer. This technique has been used in feature selection [8]. In this coursework it is applied to the kernels of a convolution layer.

Figure 5 shows the learning process for a model with some of the weights set to zero and L2 regularization applied to the non-zero ones. A previous learning process with L1 regularization ($\lambda = 0.001$) in the convolutional layers was done and the weights that are under a threshold are set to zero.

It is shown that none of the more sparse models give a better performance than the baseline. The less weights are set to zero, the highest is the accuracy obtained.

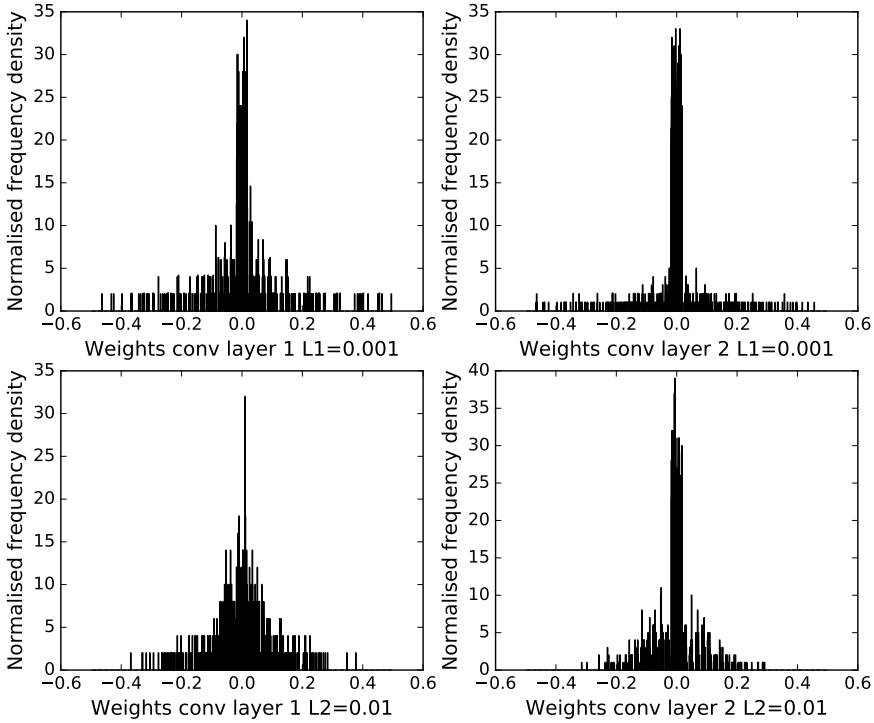

Figure 4: Histogram of the weights for the two convolutional layers when applying L1 regularization ($\lambda = 0.001$) and L2 regularization ($\lambda = 0.01$).

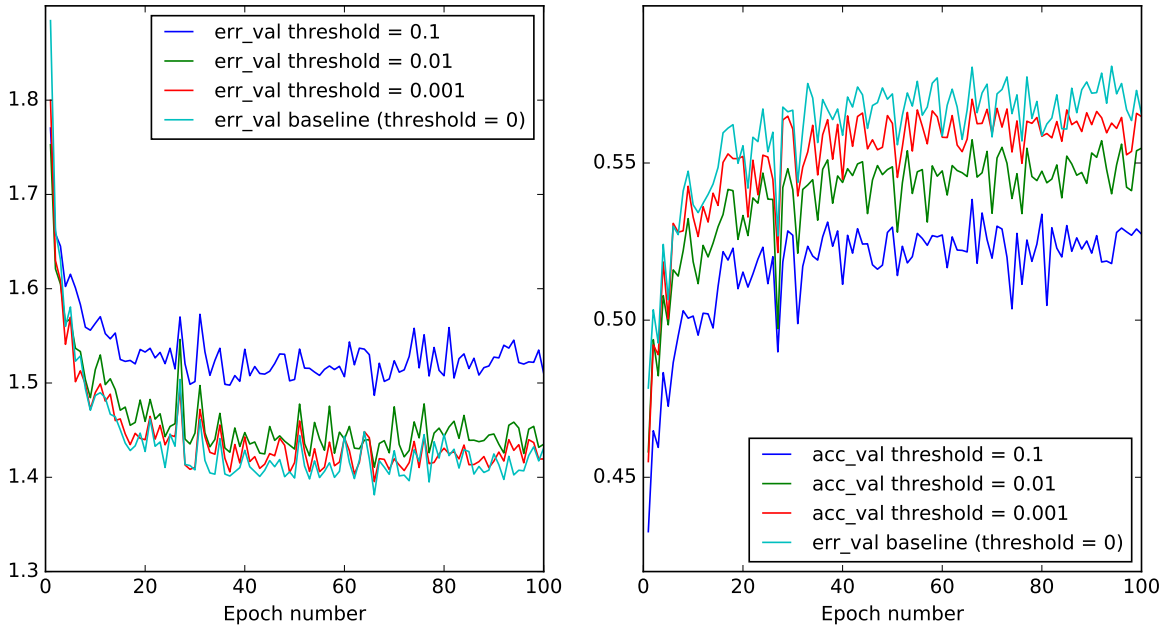

Figure 5: Learning process for a model with some of the weights set to zero in the convolutional layers applying L2 regularization ($\lambda = 0.01$). The weights set to zero are the ones that are under a certain threshold in a previous training with L1 regularization ($\lambda = 0.001$).

The complexity of the model is related with the number of free parameters and its length. When the complexity is too high there is a risk of overfitting. In this case reducing the number free parameters does not improve generalization. However, preventing the weights from growing too large does improve generalization and that is why L2 regularization in the convolutional layers is necessary, as shown in Figure 2.

As further work, it is proposed to apply the same procedure to the fully connected layer: set some of the weights to zero with L1 regularization and regularize the non-zero weights with L2 regularization. The number of free parameters in this layer is much higher than in any other layer, so that a more sparse representation could work as a regularization method.

4 Sequential modelling

In this section the data will be modelled sequentially. So far we not taken advantage of the fact that the features vectors correspond to consecutive time frames. There exist a correlation between the value of the features across the segments that could be exploited. Building a network that takes into account the distribution of the input features let the model process the data in a more structural way. It increases the representational power of the model while limiting its complexity by reducing the degrees of freedom.

4.1 Comparison between convolution in time and recurrent neural networks

There are two main approaches to do sequential modelling of the data: convolution in time and recurrent neural networks. Recurrent neural networks do time modelling by creating recurrent connections between hidden units. Convolution in time scans the input features with kernels covering the features vectors of consecutive time frames.

A feed forward network can do sequential modelling by taking as inputs not only the current frame but also previous ones. In this case, the size of the context taken into account depends on the number of previous frames considered. The context would be finite. Recurrent neural networks have recurrent connections creating cyclic connections in the graph. These recurrent connections make that the value of one hidden unit could not only depend on a number of previous frames but on the whole history. The context is then infinite.

These hidden units with recurrent connections act as memory. They can store information across infinite time frames. They also compress the information related to previous frames. Recurrent neural networks have the advantage that they are able to deal with inputs of variable length. However they are complex networks that need big datasets and are slower to train.

In this coursework, due to the limited computational resources and the availability of a preprocessed input data of fixed size, it was decided to explore the capabilities of convolution in time in The Million Song Dataset.

4.2 Implementation

In this section convolution in time will be evaluated. First, convolutional layers will be implemented where the kernels cover the feature vectors of consecutive time frames. Then, the influence of pooling after each convolutional layer will be tested. Finally, new possible architectures to apply convolution in time will be discussed.

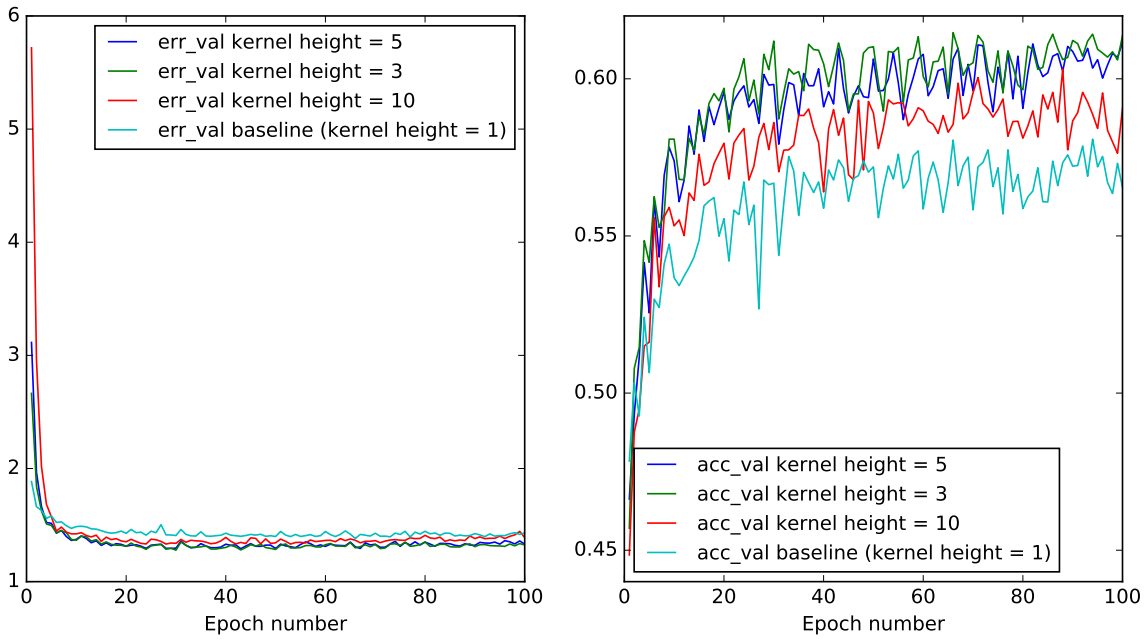

Figure 6: Evaluation of the kernel height for a model with two convolutional layers and a fully connected layer.

4.2.1 Convolution in time

Convolutional layers look for patterns in the input space that are repeated in different parts of the input space and across all the inputs. There exist a correlation between the features of a segment that holds for all segments. There is as well temporal correlation between the value of the features. Features are not assumed to be sorted in any way, so that a kernel that cover a subset of the features is not justified. Therefore, the first approach will be to scan a number of kernels, covering all the features of a variable number of consecutive segments, through all the input space.

If using 50 kernels, stacking two convolutional layers, and each kernel covering just one segment, the architecture would be the same as the baseline. It is shown in Figure 1. In order to apply convolution in time, it is needed that the kernels cover more than one segment. In Figure 6 the value of this hyperparameter *kernel_height* is evaluated.

It is shown in Figure 6 that, applying convolution in time, achieving a higher accuracy is possible. The models with 3 and 5 kernels provide roughly the same performance. The first is preferred over the second because the number of degrees of freedom of the model is smaller.

A model with three convolutional layers instead of two was also evaluated to see if an increase in the representational power is needed. Figure 7 shows a comparison between two and three stacked convolutional layers.

None of the models with three stacked convolutional layers increases considerably the accuracy with two stacked convolutional layers while increasing the number of degrees of freedom.

4.2.2 Convolution in time with pooling

After each convolutional layer, it is common to use a pooling function to reduce the dimensionality of the feature maps. The intuition is that the exact location of one feature is less important than its relative location to other features. Reducing the dimensionality of the features maps losses information but also reduce the degrees of freedom and therefore, the risk of overfitting.

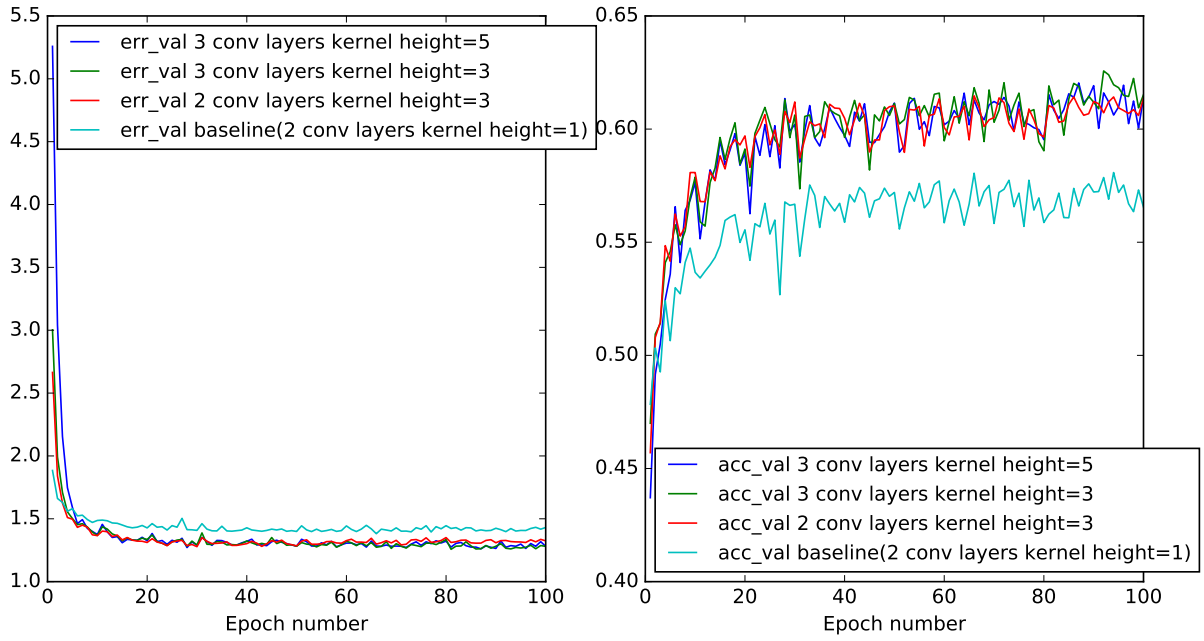

Figure 7: Evaluation of the influence of adding one stacked convolutional layer to a model with already two convolutional layers.

The influence of the use of different pooling operations has been an active area of research [6]. In the nineties sum pooling and average pooling were widely used. However, max pooling was proven to work better [5]. It is now the most common pooling operation.

In Figure 8 the impact of pooling after each convolutional layer is evaluated. Pooling is done in time dimension. Although kernels are 2-D, and the features of each segment have a relation, their relation does not depend on the distance. It is not justified to do pooling between different features or between combination of features.

It is shown in Figure 8 that max pooling does considerably improve the performance obtained without pooling. Max pooling with filter size of 3x1 and 2x1, without overlapping, gives good results. Average pooling performs worse than without pooling.

5 Model Combination

Model combination is used as a regularization method. It helps to obtain more moderate predictions. However it is also used to reduce underfitting. Combining the predictions of different models increases the representational power of the overall model. Model combination could be done by fitting the same model with different subsets of the training set. Bagging is one example that works well when there is not enough data and the model is sensible [4].

Another way of doing model combination is to fit different models in the same training dataset and combine their predictions. This works well when the component models are complementary.

Different approaches are used to combine the predictions from these different models. Averaging the output, applying a linear combination of the outputs, using a neural network or weighting the output of each predictor with how well they fit the data (mixture of experts) are widely used techniques.

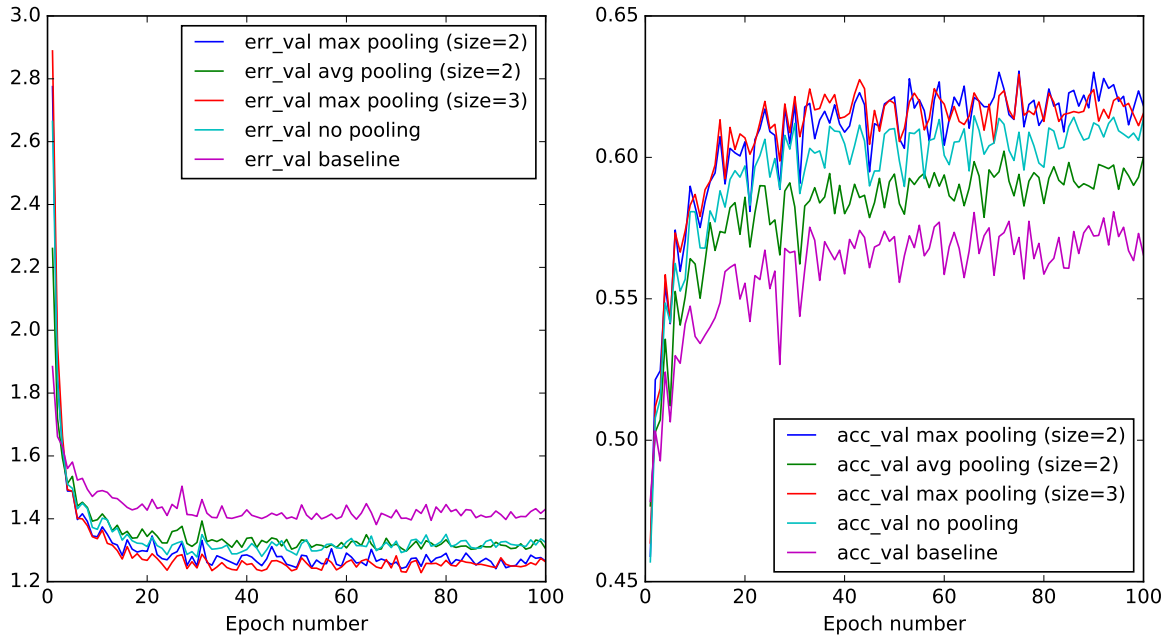

Figure 8: Influence of applying different kinds of pooling after each convolutional layer.

5.1 Implementation

In this coursework, three different models will be fitted in the same dataset and a combiner will be trained to combine their predictions. These models are intended to capture different properties of the data, so that they are expected to learn different things and work well together. These models are described below:

- **Fully connected:** This model does not assume any property of the input data. It is a 2 hidden layers model with 200 hidden units per layer. The accuracy obtained with this model is 0.50.

- **Baseline:** This is the best model obtained in coursework 3. It is used as baseline for this coursework. It assumes a relation between the features of the same feature vector. It is shown in Figure 1. The maximum accuracy obtained with this model is 0.58.

- **Convolution in time:** This model does sequential modelling of the data. It assumes that the features are correlated in time. It has the same architecture than the baseline, shown in Figure 1, but the *kernel_height* is 3 instead of 1. Covering the features of consecutive segments with the same kernel allows sequential modelling. Additionally, a max pooling layer is used after each convolutional layer.The maximum accuracy obtained with this model is 0.63.

Figure 9 shows the result of combining the three above mentioned already trained models in three different manners. They are also compared to the learning process of the model that does convolution in time in order to validate the hypothesis that the fully connected model and the baseline do actually help. The output layers, before applying the softmax operation, of these three already trained models feed the combiner. A linear combiner is first implemented. Then a non linear combiner with one hidden layer is evaluated. Finally a new experiment is done leaving all the weights and biases of the three models unconstrained. A linear combiner is used to combine the output layers. It is expected that the models could adapt to each other to achieve a better overall performance.

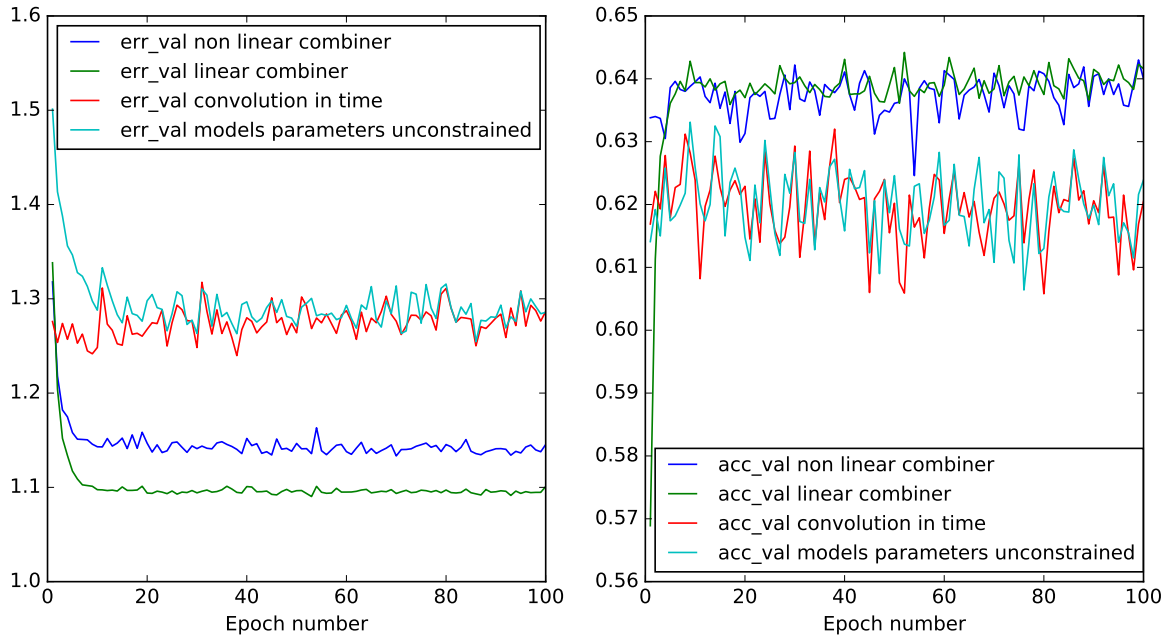

Figure 9: Comparison of the combination of three different models with a linear combiner, both constraining and unconstraining the models' weights and biases, and a non linear combiner. The learning process of the best model, which does convolution in time, is also shown to verify that model combination helps.

5.2 Discussion

It is shown in Figure 9 that model combination does improve the performance. The maximum accuracy is increased from 0.63, obtained with convolution in time, to 0.64. The linear combiner has enough representational power needed to combine the predictions. A non linear combiner does not increase the performance. Leaving the parameters of the models unconstrained while training the combiner does not help either. It was expected that the models' weights and biases could move to a new values depending on the other models to learn maybe less on its own but cover a larger input space as a whole. There are two factors that justify why a better performance was not obtained: the optimizer and the error function. Large neural networks are more prone to get stuck in a bad local optima. The optimizer may not be able to reach a good one. Even if the optimizer is guaranteed to reach the global optimum, the error function is now different. The combination of weights and biases that gives the minimum value for the error function depends on if the overall model is trained together or step-by-step.

As future work, regarding exploring the training of the whole system together, a new term indicating the similarity between the output layers, could be added to the error function. This term would encourage the models not to learn the same things and may lead to a better overall performance.

6 Correcting Model

The motivation of this section is to detect the input space where a model does not produce the correct class and train an additional model to improve the overall prediction. This new model is not intended to learn how to predict the output on itself but to modify the prediction of an original model.

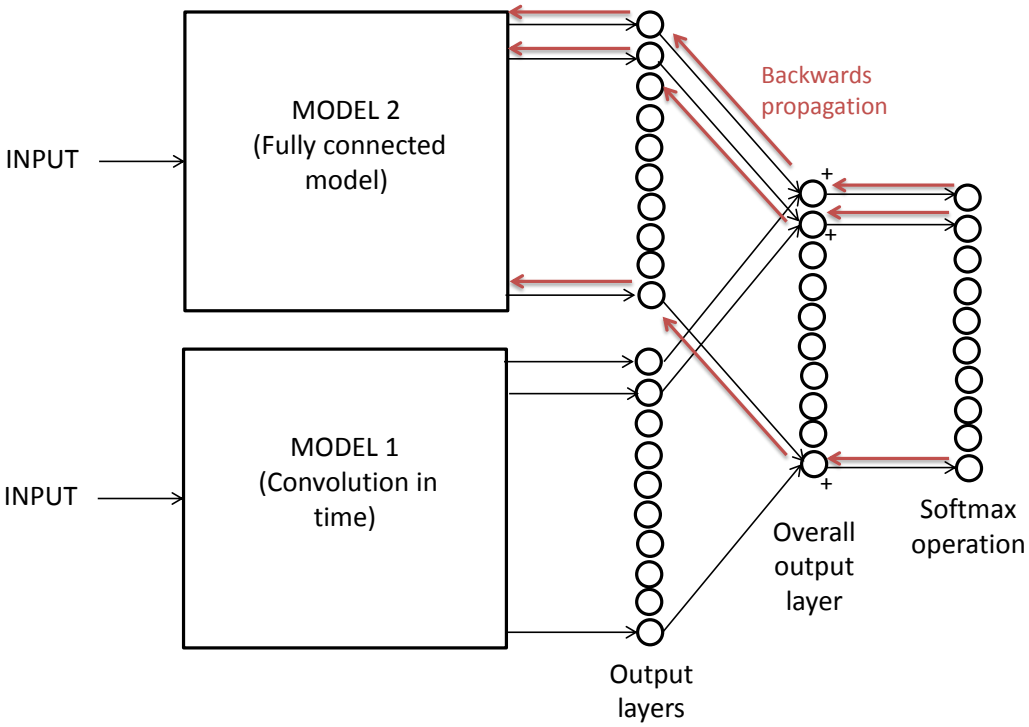

Figure 10: Overall picture of the correcting model. The model 1 is trained first and then the model 2 is trained, keeping model 1 constant, to modify the output layer of the model 1 in search of a smaller error.

6.1 Implementation

In order to implement that, first, the best model previously obtained with convolution in time is trained. Then, a model consisting of this already trained model and a fully connected model is built. This additional fully connected model consists of two hidden layers of 200 hidden units per layer. The output of the overall model (before normalizing the output with the softmax operation) is the summation of the outputs of both models. It is shown in equation 3.

$$Overall\ Output = Softmax(output_{original\ \ model} + output_{correcting\ \ model}) \tag{3}$$

The resulting model is trained keeping fixed all the weights and biases of the original model. Therefore the additional model will learn how to modify the output layer of the original model to better predict the output. As it is intended to modify the probability distribution over the classes, it is trained to reduce the error between the softmax of the resulting output layer, namely, the summation of the output layers of both models. Figure 10 shows how these two models are combined.

The comparison of this approach with the best combination of the models obtained in the previous section is shown in Figure 11. It is also shown in Figure 11 the learning process when the weights and the biases of the original model are left unconstrained.

It is shown in Figure 11 that this new approach performs better than the best model obtained by combining different models. Although the difference is not extremely large, it is expected that it could be improved by implementing a more complex model. Compared to model combination, only previously training a single model is needed. It is also shown in Figure 11 that leaving the weights of the original model unconstrained provides a worse performance. As further work related to this topic, stacking more correcting models is proposed. It is expected that each of these stacked correcting models could improve the accuracy.

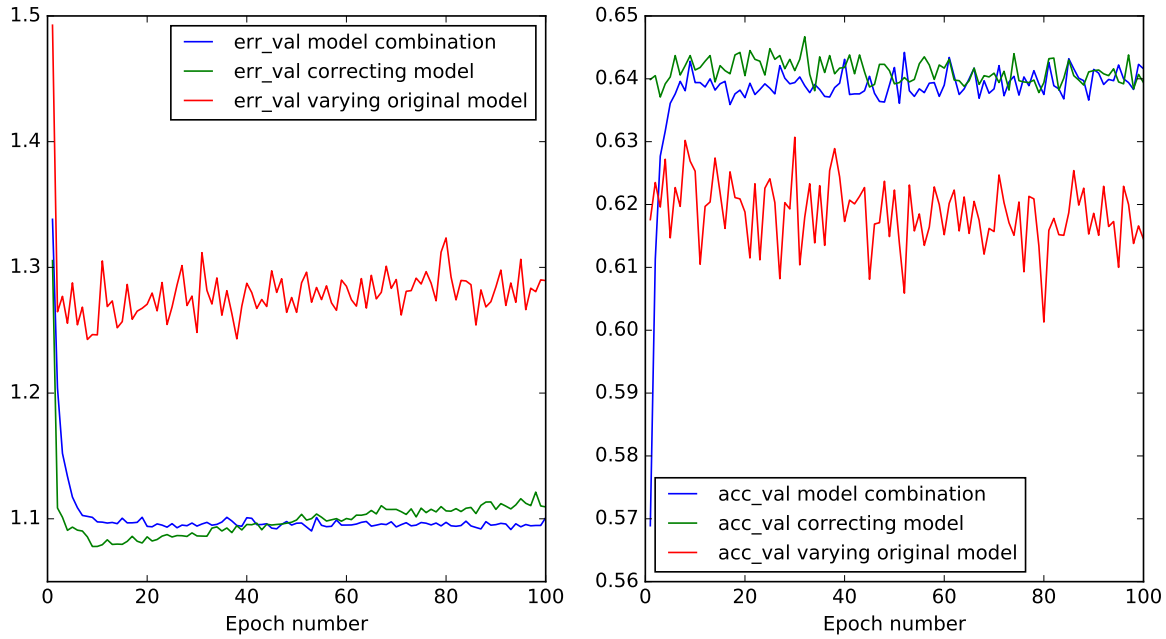

Figure 11: Comparison between the best model obtained with model combination, the one obtained with the correcting model and with the correcting model leaving the weights and biases unconstrained.

6.2 Adding the hidden representation as input

Multitask learning uses the hidden representation of the data when training one model to solve one task, to help improving another model to solve a different task with the same input [3]. Inspired on the idea of multitask learning, a new variant of the previous approach is here proposed. Compared to multitask learning, where the hidden representation is used for another task, here it is used in the same task but to feed another model. Not only does the correcting model take the features as inputs but also the hidden representation, including the output layer, of the original model. This is shown in equation 4. A comparison between the original approach and this new variant is shown in Figure 12.

$$Output_{correcting\ model} = f(inputs, hidden\ representation_{original\ model}) \tag{4}$$

It is shown in Figure 12 that adding the hidden representation and the output of the original model to the additional model as input does not improve the performance.

6.3 Comparison with model combination

In order to analyse the differences between the correcting model proposed and model combination, visualizing what the output layer has learned is considered essential. Figure 13 shows the comparison between the accuracy obtained by evaluating the class predicted with the output layer of the correcting model and a model of the same architecture trained alone to predict the target itself.

Figure 13 shows that the accuracy obtained by evaluating the output layer of the correcting model is about 10 percent. Taking into account that the number of classes to predict the output from is ten, it is clear that this model does not learnt to predict the class on its own. It has learned how to contribute to the output layer depending on the region of the input space to increase the overall performance of the summation of both output layers.

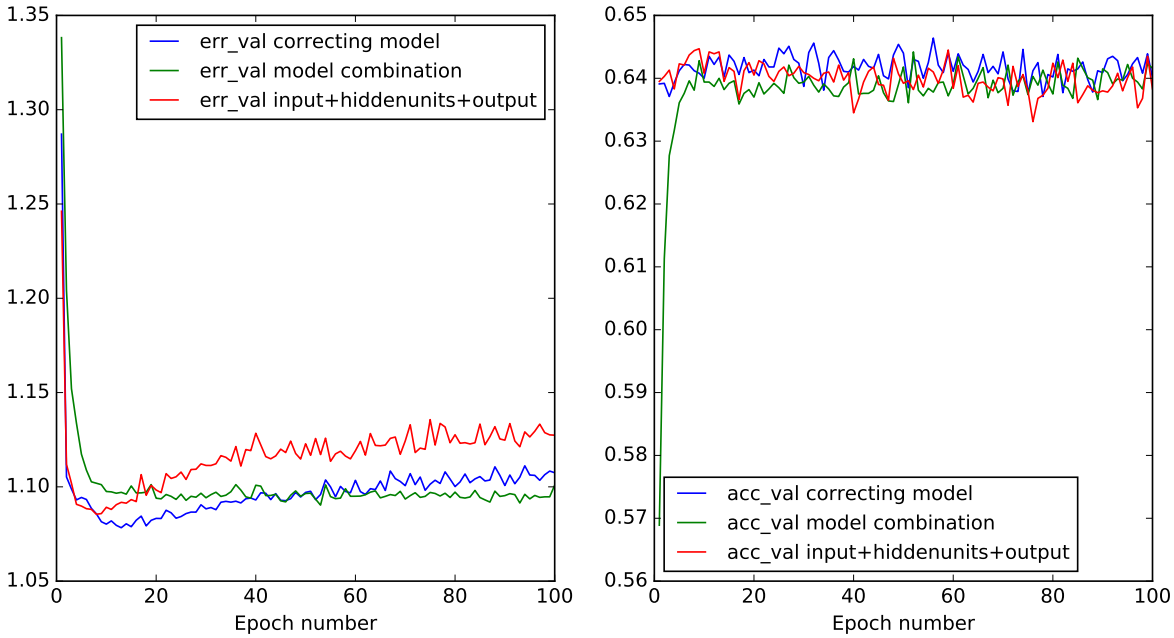

Figure 12: Comparison of the new variant of the correcting model taking into account the input and the hidden representation of the original model with the initial approach which only takes into account the input features.

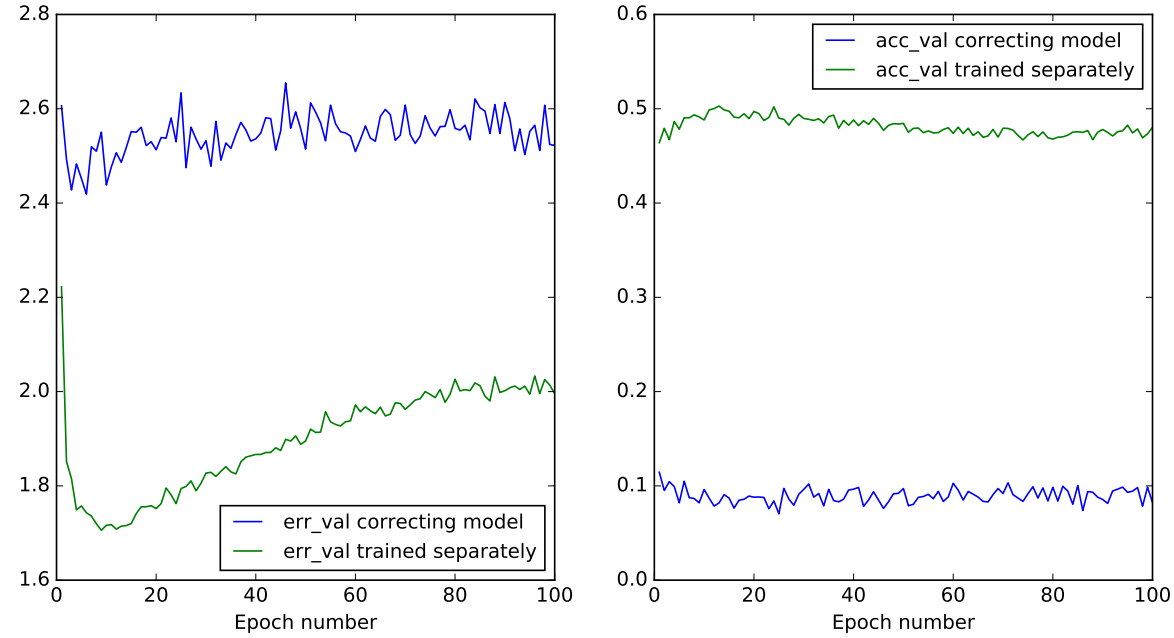

Figure 13: Comparison of the accuracy obtained by predicting the class with the output layer of the correcting model and a fully connected model of the same architecture trained alone to predict the target itself.

It could be seen as well as model combination. The same performance is obtained if both models' output layers are combined with a linear combiner. The original model needs to be trained before, and keeping fixed all the weights and biases of the first model the second model is trained together with the combiner. In this case the second model learns as well how to modify the first model prediction rather than predict a class on its own. Although both achieve the same performance, the correcting model was chosen over this way of combining models because it is more intuitive and has no additional degrees of freedom (free parameters of the linear combiner).

7 25-genre task

In this section, the models obtained in the 10-genre task will be evaluated in the 25-genre task dataset. Different architectures were explored to capture the input data distribution, to combine the predictions from different models or to improve the performance of an already trained model. The models evaluated in the 25-genre task dataset are:

- **Baseline:** Best model obtained in coursework 3. It is used as baseline for this coursework. It takes into consideration the relation between the features inside each segment but not the temporal relation of the features. It is shown in Figure 1.

- **Convolution in time:** This model does sequential modelling of the data. It assumes that the features are correlated in time. It has the same architecture than the baseline, shown in Figure 1, but the *kernel_height* is 3 instead of 1. Additionally, a max pooling layer is used after each convolutional layer.

- **Model combination:** This model combines three already trained models to get a better prediction. The models are the baseline and the one that does convolution in time, which are described above, and a fully connected model. The output layer of these already trained models feed a linear combiner that is trained to predict the target.

- **Correcting model:** A fully connected model is trained to add a correcting term to the output layer of an already trained model (convolution in time). The output layer is then the sum of both output layers, shown in equation 3. An overall representation of the training of the second model is shown in Figure 10, although a 25 length vector is now in the output layer instead of 10.

The reason for assuming that the models that perform well in the 10 genre-task will work in the 25-genre task is that both task have the same input features. The assumptions made about the input features also hold for this task. However, since the number of weights in the output layer is now bigger, not regularizing the last layer may not be a good idea. The performance of the model that does sequential modelling is evaluated both regularizing and without regularizing the output layer. Figure 14 shows this comparison.

It is shown in Figure 14 that also in the 25-genre task, the model performs better when the output layer is not regularized.

Figure 15 shows the learning process for the four models mentioned above in the 25-genre task.

It is shown in 15 that the results obtained are similar to the 10-genre task. Model combination gives nearly the same performance than convolution in time whereas the correcting model does considerably improve the performance of convolution in time.

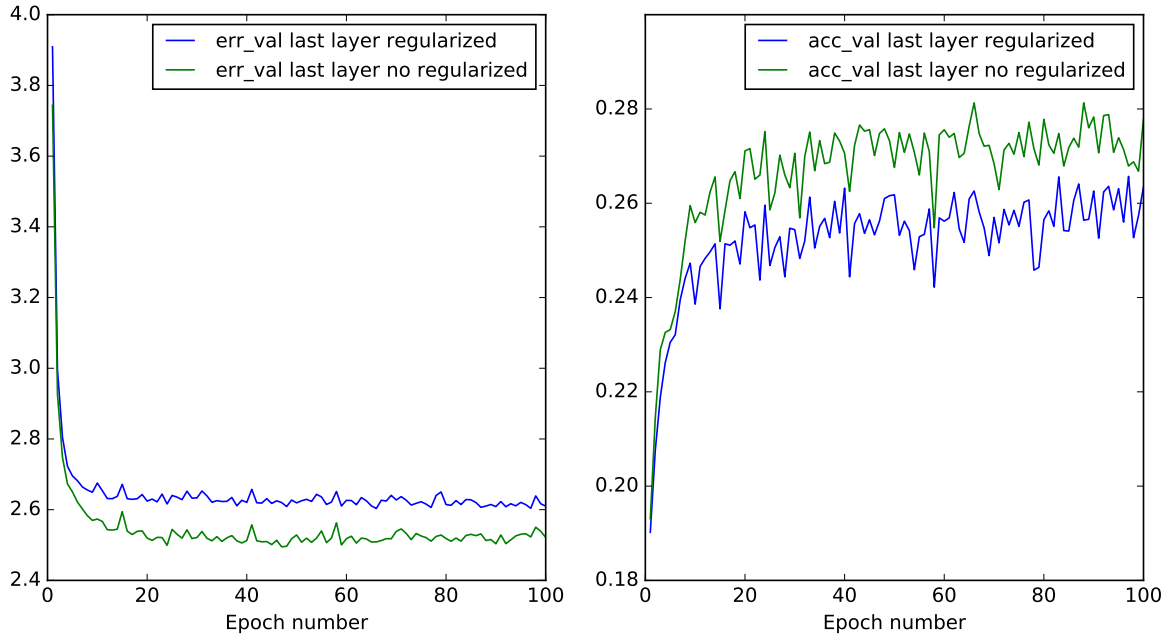

Figure 14: Comparison of the accuracy obtained with the best model obtained with convolution in time regularizing the last layer and without regularizing.

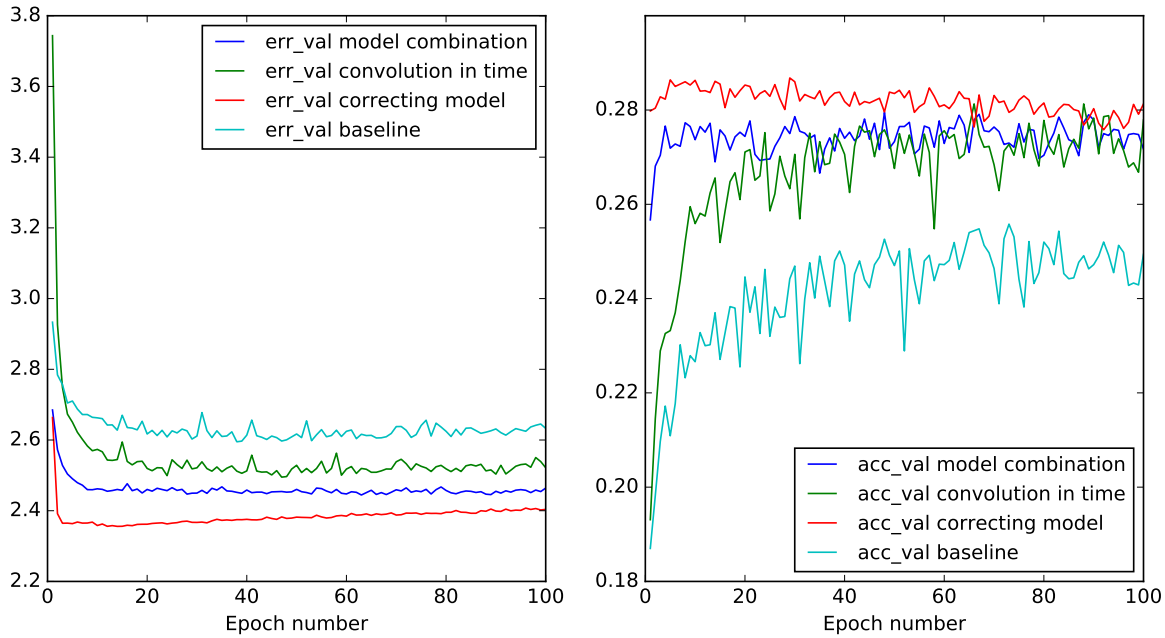

Figure 15: Learning process for the baseline, the model that does convolution in time, model combination and the correcting model in the 25-genre task.

8 Conclusion

In this coursework different research questions were investigated. Special effort was put on modelling the data sequentially and on combining the predictions from different models. Optimize large models is hard [9]. The problems encountered when optimizing large models consisting of different submodels were described. The effect of a step-by-step training was discussed. This sequential training leaded to a new approach that performed better than model combination.

Convolution in time was implemented to take advantage of the temporal relation of the values of the features. The modification with respect of the baseline, shown in Figure 1, just consisted of an increment of the *kernel_height* from one to three. A kernel covering the features of consecutive segments allows sequential modelling. An increase from 0.58 to 0.61 in the accuracy in the validation set of the 10-genre task was obtained. Adding max pooling in the time axis after each convolution layer increased the accuracy to 0.63. Max pooling reduces the free parameters of consecutive layers while keeping the response of the patches to the patterns learned by the kernels. The exact location of these patterns is after max pooling less precise. However the increase of accuracy indicates that the loss of information is not critical for this task.

Model combination was used to combine the predictions from different models. The models are: a fully connected model that does not assume any relation in the input data; the baseline intended to capture the patterns in the relation between the features of each segment; and the above mentioned model that does convolution in time. Combining different models could achieve better performance than any of the individual models. The more different these models are, the better performance is expected. A linear combiner fed with the output layer of these already trained models was trained and a accuracy of 0.64 was obtained, which is better than any of the models used.

A novel approach was then proposed to learn how to modify the output layer of an already trained model to increase the accuracy. This model is explained in Figure 10. A slightly higher accuracy(0.645) was obtained with this approach. Moreover, just one previously trained model is needed. In order to compare this approach with model combination two points are important. First, the correcting model does not learn how to predict the output itself, it learns how to complement the output layer of the first model. The output layer of the second model on its own does not achieve a better accuracy than a random predictor. Second, if these two models, the original one and the correcting model, are combined with a linear combiner; and the original model is trained first and then the second model together with the linear combiner, the same performance is obtained. The correcting model is a particular case of these model combination strategy, where all the weights of the linear combiner are set to one.

There are two reasons for this improvement in performance when different parts of a model are trained separately: the optimizer and the new cost function. Larger neural networks are more difficult to optimize and are more prone to get stuck at a local optima. The optimizer may not be able to reach the global minimum. The error function to be minimize is not the same as before. Even if the optimizer is guaranteed to find the global minimum, having fitted the parameters in two steps make that the result expected from both optimizations is different.

In order to compare the different models explored in this coursework, the main criteria was the accuracy obtained in the validation set. Specially in model combination or when implementing the correcting model, the models are not very interpretable. Large ensembles of models have been used to win many Kaggle competitions. These models are criticized for being impractical and difficult to interpret. The Netflix prize is a widely known example[12]. In this case the model was too complex and was decided not to implement it despite its good performance. The model that applies convolution in time obtained in this coursework is interpretable, efficient and provide useful information about the data such as the *kernel_height* that indicates how

long are the features of the segment correlated.

Further work have been proposed along the coursework for the different topics explored in this coursework. More generally, an implementation of recurrent neural network would be desired to contrast the assumptions made when discussing sequential modelling for this task. It should be compared in terms of free parameters of the model, accuracy and training time. It is also desired to do a more in-depth research in literature regarding model combination to put the correcting model approach more into context. Model combination literature was explored in search of sequential training or forcing models to learn different subspaces of the input space but not related work was found.

References

[1] Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[2] Lee, H., Pham, P., Largman, Y., & Ng, A. Y. (2009). Unsupervised feature learning for audio classification using convolutional deep belief networks. In Advances in neural information processing systems (pp. 1096-1104).

[3] Caruana, R. (1998). Multitask learning. In Learning to learn (pp. 95-133). Springer US.

[4] Breiman, L. (1996). Bagging predictors. Machine learning, 24(2), 123-140.

[5] Scherer, D., MÃijller, A., & Behnke, S. (2010, September). Evaluation of pooling operations in convolutional architectures for object recognition. In International Conference on Artificial Neural Networks (pp. 92-101). Springer Berlin Heidelberg.

[6] Boureau, Y. L., Ponce, J., & LeCun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. In Proceedings of the 27th international conference on machine learning (ICML-10) (pp. 111-118).

[7] Cortes, C., Mohri, M., & Rostamizadeh, A. (2009, June). L 2 regularization for learning kernels. In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (pp. 109-116). AUAI Press.

[8] Demir-Kavuk, O., Kamada, M., Akutsu, T., & Knapp, E. W. (2011). Prediction using stepwise L1, L2 regularization and feature selection for small data sets with large number of features. BMC bioinformatics, 12(1), 412.

[9] Goodfellow, I. J., Vinyals, O., & Saxe, A. M. (2014). Qualitatively characterizing neural network optimization problems. arXiv preprint arXiv:1412.6544.

[10] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[11] Graves, A., Mohamed, A. R., & Hinton, G. (2013, May). Speech recognition with deep recurrent neural networks. In Acoustics, speech and signal processing (icassp), 2013 ieee international conference on (pp. 6645-6649). IEEE

[12] Bennett, James, and Stan Lanning. "The netflix prize." Proceedings of KDD cup and workshop. Vol. 2007. 2007.