MLP Report4:
Improvements on training sequential states of Recurrent Neural Network for Million Song Database 10 classification task

s1687487@sms.ed.ac.uk

March 21, 2017

In this report, Deep Neural Network of sequential inference model is introduced for the song data classification task. The task in nature refers to one label for the outcome and the network architecture may better to reckon temporal hidden factors to emit the final guess. This report introduces vanilla Recurrent Neural Network(vRNN) as the reference sequential inference model, and its limitation is discussed in both theoretical and empirical perspective in section 2. Subsequently, more advanced RNN models with gate mechanism, Long-Short Term Memory(LSTM) and Gated Recurrent Unit(GRU) cell structure, are introduced and how they advance from vRNN. Investigation on various aspects of engineerings for the RNN with gate mechanism follows.

By observing the weakness of the gated-RNN(gRNN) for the classification task, the report focuses on methods to improve the capability of the unique characteristic of gated-RNN(gRNN) architecture for the classification task, which is sequential cell states. Several candidates on dealing with this issue are introduced in section 3. In terms of improvements on input injection to gRNN, Convolutional Neural Network(CNN) embedding method is investigated as well as sensible bridging techniques for CNN embedding and gRNN. Improving the use of cell states on the other hand, Attention mechanism and sequential Auto-Encoding(AE) pre-training are introduced, which ends up with discussion on an objective function looking over the whole outputs in the temporal space. The report then investigates on the combined model of the suggested improvement methods - defined in section 4 - in terms of controls on hyper-parameters, over-fitting, under-fitting the performance with comparison to that of a baseline gRNN model in section 5. Analysis on the experimental results are theoretically analysed and deduces how the result of investigations empirically support the idea of improving the cell states for the classification task.

It is found that CNN embeddings for gRNN improves in performance when the dimension of temporal space is conserved via not applying pooling layer on the temporal space. In addition, sensible cross-entropy measure over the whole output states is a better curriculum for the gRNN model eventhough the classification task in nature measures the accuracy from the last output state of gRNN. Conclusion of the report is in section 6.

Contents

1 Million Song Database classification task

Million Song Database(MSD) is a collection of acoustic data of songs labeled with genre of the songs they best fit. MSD10 is the dataset having 10 genres and MSD25 is of 25. Our goal is to build up a neural network model to best predict the genre of each song data. The nature of MSD is that the dataset is big and complexly patterned. Big data makes the use of traditional Gaussian Process strategies computationally intractable, and complexity of representations underlying in acoustic dataset usually requires unaffordable feature engineerings in general, and very precise kernel designs for kernel methods. For the coursework, MSD10 has 25 feature dimension over 120 time-steps per input for over 70,000 training data. This makes the classification on MSD hugely challenging in conventional approach.

While we see the improvements on the above issues by directly challenging against them, neural network architecture gives new aspect of the machine learning task on this venue. The bipartite neural network with multi-layer perceptron error measure [1] we have been using in this course provides one of the most simplest optimisation objective on linking factors. The learning scheme is as follows:

$$H = \sum_{i \in \forall data} CrossEntropy(output_i, target_i)$$

where the $output_i = \mathcal{M}(input_i)$ is yielded by the model $\mathcal{M}$ we specify. Layer-wise transfer of activation consists of simple affine transformation and non-linear activation in MLP enables that the error measure can rapidly back-propagate from the top hierarchy of the model to the input level [2]. The distinctive feature of this architecture is that we can design complex factor graph model with little concern about hidden variables inference task, as the efficient back-propagation for the gradient descent over the whole model properly feeds the update[3]. According to the coursework 1, 2, and 3, reasonable search on model design, regularisation, and learning schedule give good local optimums in affordable time-span.

However, previous courseworks mainly concerns about hierarchies of the model which is bipartite in one way. This means that one layer of hidden variables can only get fed from another layer specified. This nature is not favourable for the sequential inference in particular. It is because we normally regard the observation in the past affects on the present state and

future. Hence factors from the past and input in present affect on the perception in present simultaneously. Hidden Markov Model is a good example. Then the question boils down to following: can we implement the neural network model that is sequentially linked, and how would it affect on the properties of learning?

Starting from the above issue, this report introduces a sequentially-inferring neural network model called Recurrent Neural Network(RNN). We will see the goodness of having such structure over the report. However, there remains another important question: where should we infer about the class of the data? This is a distinct problem made by the sequential inference model from the fully connected layered models we investigated in the previous courseworks. See section 2.d and 5.b.2 regarding about this issue for detail.

2 Sequential learning via Deep Neural Network: Baseline

2.a Vanilla recurrent neural network

Considering that Million Song Data is a sequential data with temporal axis, modeling the inference with 'cell states' as hidden variables of the neural network architecture is a reasonable option. If the cell states stores what happen in each time and are backed up by cell states in the past, the network may understand temporal dependencies along the time space of the data. Hence this report introduces Recurrent Neural Network(RNN) as one of its type.

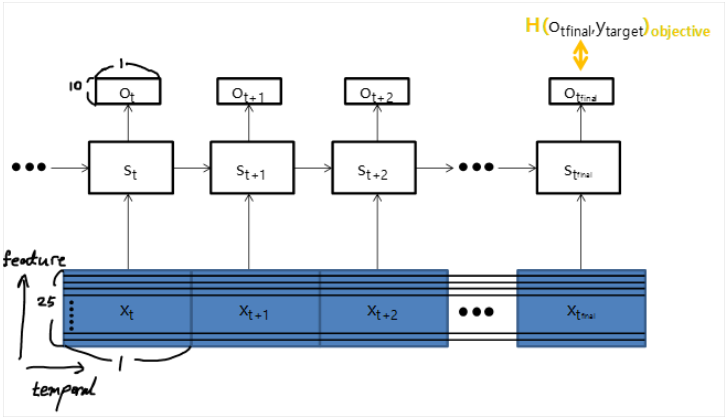


Figure 1: Structure of Recurrent Neural Network(RNN) for classification task. For vanilla RNN, s_t is simply $\underline{c_t}$. Other RNNs introduced later has additional memory state $\underline{m_t}$ inside s_t.The objective function commonly refers to the last output. For MSD10, timestep of input per cell is $1(Dim(x_t)_{tmp})$ over the whole 120 steps, and 25 features$(Dim(x_t)_{ft})$.

The simplest vanilla-RNN(vRNN) reads each time-step of data with same weight and bias, giving cell state of the time step by adding recurrence of the last cell state.

$$\underline{c_t} = f_{activation}(\underline{\underline{W_T}} \cdot \underline{x_t} + \underline{\underline{W_C}} \cdot \underline{c_{t-1}} + \underline{b_T})$$

where $\underline{c_t}$ is the cell state 1D matrix at time-step t and $\underline{x_t}$ is the input at the time-step. The cell state then becomes an output typically by treating with plain fully connected layer(FCL): $\underline{o_t} = g_{activation}(\underline{\underline{W}} \cdot \underline{c_t} + \underline{b})$. The biggest highlight on this model from simple fully-connected layer model(sFCL) is that we split the input space into time-steps and make a series of inference on hidden variable over time. In sFCL model, we just put all the input space into one matrix and mix up for the affine transformation to the next layer. Therefore, RNN retains the locality of data over the temporal space into the learning and inference process.

In addition, update is the interesting part. For the classification task we are dealing with, we measure the softmax cross-entropy($H(\underline{y}, \underline{o_{final}})$) at certain time-step - we shall typically be the last time step t_{last} - and backpropagate the error to the parameters we saw in the above expression $\underline{c_t}$. As the backpropagation at $\underline{c_t}$ links to the one at last time-step $\underline{c_{t-1}}$, the update of (W_T, W_C, b_T) walks until it meets the initial time-step t_0, in theory:

$$
\begin{aligned}
\frac{\partial H}{\partial W_T} &= \frac{\partial H}{\partial c_t}\frac{\partial c_t}{\partial W_T} \\
&= \frac{\partial H}{\partial c_t}\sigma_t(1-\sigma_t)\big(x_t + W_C\frac{\partial c_{t-1}}{\partial W_T}\big) \\
&\quad \text{if } f_{activation} \text{ is sigmoid } \sigma, \text{ where } \sigma_t = \sigma(a_t) = \sigma(W_T \cdot x_t + W_C \cdot c_{t-1} + b_T) \\
&= \frac{\partial H}{\partial c_t}\sigma_t(1-\sigma_t)\Big(x_t + W_C\sigma_{t-1}(1-\sigma_{t-1})\big(x_{t-1} + W_C\frac{\partial c_{t-2}}{\partial W_T}\big)\Big) \\
&\quad \text{... until the expansion reaches to } t_0
\end{aligned}
$$

Similar recurrence also happens for W_C and b_T updates. This nature of update guarantees that the parameter space regards with not only present time-step, but also with $\forall x_i$, hence the concept of sequential inference works.

However, such contribution of the past memories weakens as the distance of time goes bigger. Looking up the above update again, the update of W_T by x_{t-1} takes additional factor $W_C\sigma_{t-1}(1-\sigma_{t-1})$ than that of x_t. As the sigmoid function always sits in [-1,+1], older inputs as well as older cell states alway contribute less to the time-step when we classify. Therefore a lack of long-term dependency occurs. This is called 'gradient-diminishing' phenomenon where the vRNN particularly performs worse when the time step gets longer[4]. In terms of forward-feed during the process, $t_{final} - t_i$ times of $f_{activation}$ wrappings on x_{t_i} also blurs our its contribution to o_{final}.

Figure 2 shows that the weak performance of vRNN may stem from this phenomenon. The reason consists of two steps. Firstly, vRNN performs less than that of FCL which actually does not discriminate distant input vectors in temporal axis. Secondly, when we control the norm of gradient by clipping[1] using '*tf.clip_by_norm*' function, then the performance becomes on par.

2.b RNN with gates: Long-Short Term Memory model case study

What is good about inferring by sequence then? In terms of probabilistic modeling, learning by sequential step learns how the sequence generates, by optimising the distribution of parameters linking the time-steps. If the vRNN works well in this idea, it would give better result than FCL as we recognise songs by perceiving them over time. The task given to our neural network is to mimic human classifier, so the linking parameters need to be learned better. However, bluntly forcing the norm of gradient into certain interval does not hugely gives on this point as we saw. Therefore we need some methods to enforce the cell states to contribute better.

Based on this idea, this report introduces a gated RNN called LSTM[5]. Mathematical forwarding rule of signal in LSTM is below (fig 3b):

[1] more details about implementation is in section 2.b.2

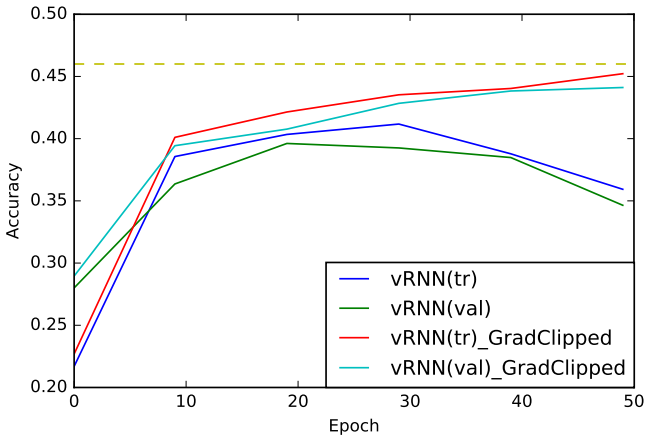
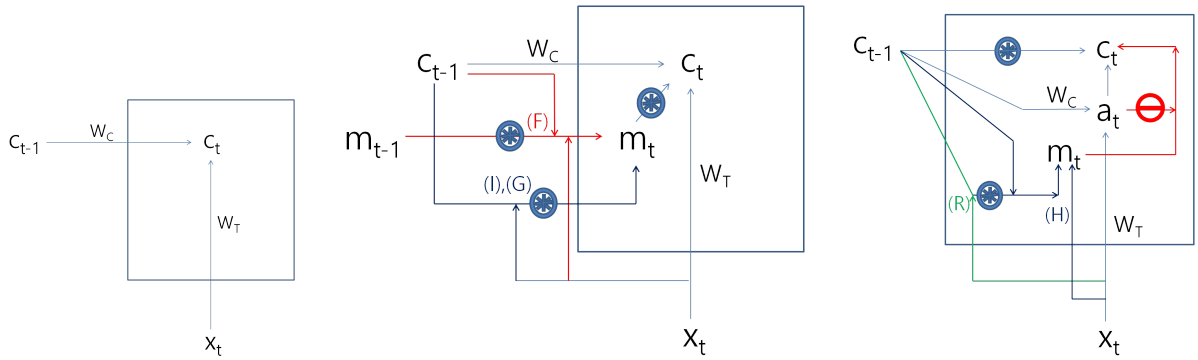


Figure 2: Classification accuracy of vanilla RNN(vRNN) by epoch with/without gradient clipping on the state recurrence part. Yellow line is learning performance of a fully connected layer network baseline.



(a) Vanilla RNN (b) Long-Short Term Memory (c) Gated Recurrent Unit

Figure 3: Cell structure of various RNN design. Advance comes from the gate mechanism$(*), (-)$ with memory state $\underline{m_t}$ covering long-term dependency problem caused by gradient diminish phenomenon during the update.

$$\underline{c_t} = f_{activation}(W_T \cdot x_t + W_C \cdot c_{t-1} + \underline{b_T}) * Tanh(m_t) \tag{1}$$

where m_t is the memory activation given by

$m_t = F(a_t^{(F)}) * m_{t-1} + I(a_t^{(I)}) * G(a_t^{(G)})$ where F,I,G are saturating transfer functions and a_t is the activation at time t, which is $W_T \cdot x_t + W_C \cdot c_{t-1} + b_T$

but having $a_t^{(F,I,orG)}$ means we have three additional sets of parameters

to the one for a_t, which is often noted as $a_t^{(O)}$

The element-wise product(*) between mapped input and cell state is the part where we call as gate. Hence LSTM simply element products the activation transfer with a non-saturating *Tanh* map controlled by activation-independent memory m_t consists of past memory and three gates.

In theory, LSTM hence improves on long-term dependency(or contribution diminishing over time) by two folds: non-saturating $Tanh(m_t)$ gives that c_{t-i} can have multiplier which

> 1 magnitude, and the memory activation m_t also has addition(+) operation so that it can also hold the contribution by m_{t-i}. Hence if LSTM performs better than vRNN for MSD10 dataset when we take the classification one-hot vector at t_{final}, it becomes an experimental proof that some knowledge located distant from t_{final} present useful representation by flowing over $\{m_{t>i}\}$ *and* $\{c_{t>i}\}$.

2.b.1 Memory improves the learning performance

[2]

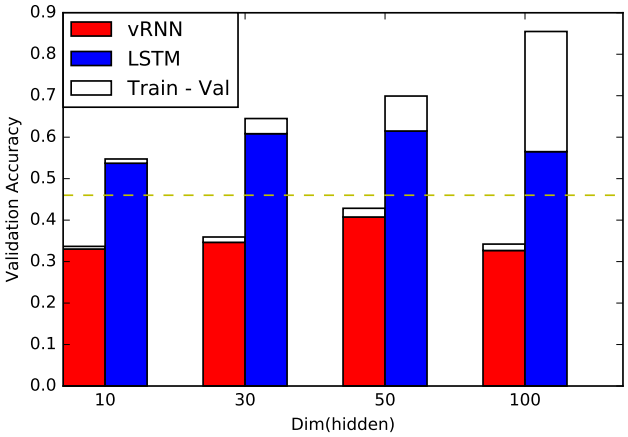


Figure 4: Classification accuracy of vanilla RNN(vRNN) and LSTM with various dimension of hidden state under classification layer. White bar is a gap between training set and validation set. Yellow line is the performance of the FCL baseline.

Figure 4 above clearly shows that LSTM works better with a big jump. The performance hugely exceeds both vRNN and FCL. By exceeding FCL's performance, the experiment shows that LSTM learns the distribution of variables linking the formation of input slice per time step, as FCL does not have resolution on temporal dependency of system. By exceeding vRNN, it confirms that the source of better inference can sit distant from the final time-step and gate concepts allow the sequential representations to flow over time in that range, not to let them diminished. Out-large state space results in overfitting and underfitting for the other way round like typical FCL models.

Hows does this 'allowance to the norm of gradients over 1 for each entry' allow this much progress? Theoretically, the gates can be understood as numerical realisation of filters. They have local patterns to magnify or mute so forged over sequence so that each input slice over time step can gradually better the inference as well as learning. Unlike vRNN, LSTM has two channels (c,m) to flow such state memories so that they counteract not to forget meaningful representations. Hereby the report proposes the first hypothesis to investigate in this report.

Hypothesis 1 *Gating the recurrent states of RNN improves retrieval power of long-term dependency in sequential data, hence the classification performance.*

[2] In 0000(-) by LSTMbaseline_MSD.ipynb

2.b.2 Gradient control using gradient clipping and forget-gate bias

As mentioned above, two state representations {c} and {m} interact dynamically. Considering there high dimensionality, we do not guarantee whether the whole LSTM system becomes bifurcate, complex, or even diverge by their geometry of parameters[4]. This is a problem as we want to optimise the system, but the optimisation scheme is not the same as what we expect from FCL that provides guaranteed local optimum.

The above difficulty can be rewritten as problems on back-propagation during its training which can now also explodes as well as vanishes through time [4]. This is because the back-propagation through the state-space is a chain of matrix multiplications of the state gradient plus addition of gates. For example, both 0.6 and 0.7 are less than 1.0 but summing them gives bigger than 1. Gradients which vanished in vRNN now can even diverge through time - but not geometrically diverges as each element for the sum are sigmoided which ceils the ouput up to 1. Therefore, poorly controlled LSTM may give one of two results: vanishing gradient through time weakens the long-term dependency of the architecture [6] and explosion obviously makes the state-space saturated[3].

Tensorflow library we use for this course deals with the above issues by its own by providing its users nice *LSTMcell* module and *dynamicRNN* structure, so that we cannot truly observe the fluctuating gradients - cells and gradients are dynamically wrapped up as one united cell and gradient so that gradients over time are not stored as series[4].

However, there are two suggested ways found to be effective on dealing with this issue so that we can see the gradient control affects on the performance. Firstly, we initialise the bias of the forget gate near 1 for all dimension [6], which is a substantial injection to the forget gate. Secondly, we can regulate(or clip) the gradient update up to a certain norm [4].

Forget bias [5] By controlling *forget_bias* parameter provided in *tf.nn.LSTMcell*, we measure the performance of n_{ly} layer LSTM, showing the optimum point around 1 in figure 5. Giving less or more initial bias makes the model under-fitted. The reporter believes that giving too large bias as initial saturated the forget gate(F) so that signal forwarding becomes too crowd, and zero initial on the other hand frustrates the memory activation state m_t by heightening the threshold that $Tanh$ transfer has.

Gradient clipping [6] Unlike the forget bias initialisation control, clipping the gradient does not hugely work by its own. As figure 6 shows, hard-ceiling the gradient to small norm obviously interrupts the proper update via back-propagation resulting in lower learning performance. Other than that extreme, saturating gates may smooth out the clipping effect. Ceiling the norm of temporal gradient also means that discriminating time-steps with various level of update might be interrupted.

With initial forget-bias given, the model becomes even more tolerant to the norm control of the temporal gradient. This means that the near optimum forget bias initialisation controls the update gradient lower but not become convergent to zero through time. Sensitive to small gradient can possibly mean that any reasonable cross-entropy error from the classification task are given opportunities to update the model and not overwhelmed by particular data points, while they is also controlled to have small update gradient.

[3]Most of LSTM models use Sigmoid transfer for F,I, and G gates, but we have seen in previous assignments 2,3 that even non-saturating transfers neither avoid poor performance when the situation rises

[4]It is also not possible to see them by running the session and the tensor object is limited not to iterate.

[5]In 0304/2(27438) by $CL_LSTMbaseline_forgetbias_MSD.ipynb$

[6]In 0306/0(-) by $LSTMbaseline_GradientClip_MSD.ipynb$

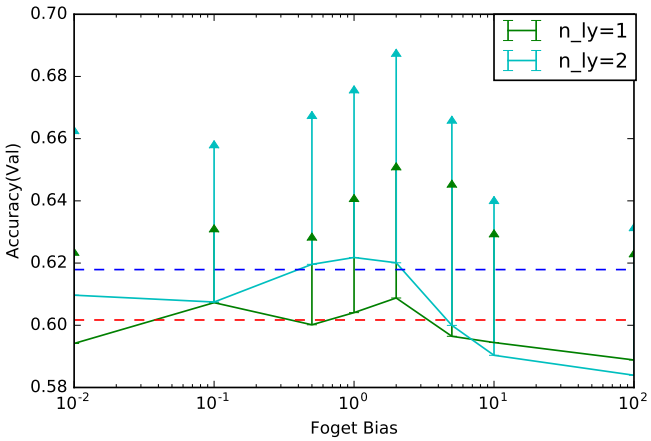


Figure 5: Classification accuracy of LSTM with various Forget-bias initialisation and number of LSTM cell layers(n_{ly}). Arrows indicate gaps between training and validation accuracies. Dashed lines are ones when no forget biases are initialised(red/blue: 1/2 layers).

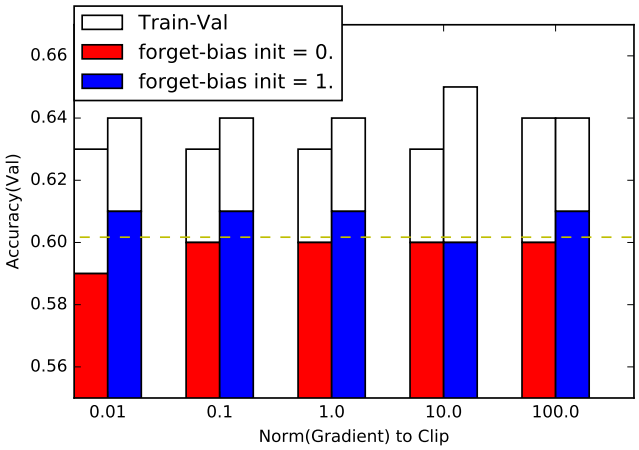
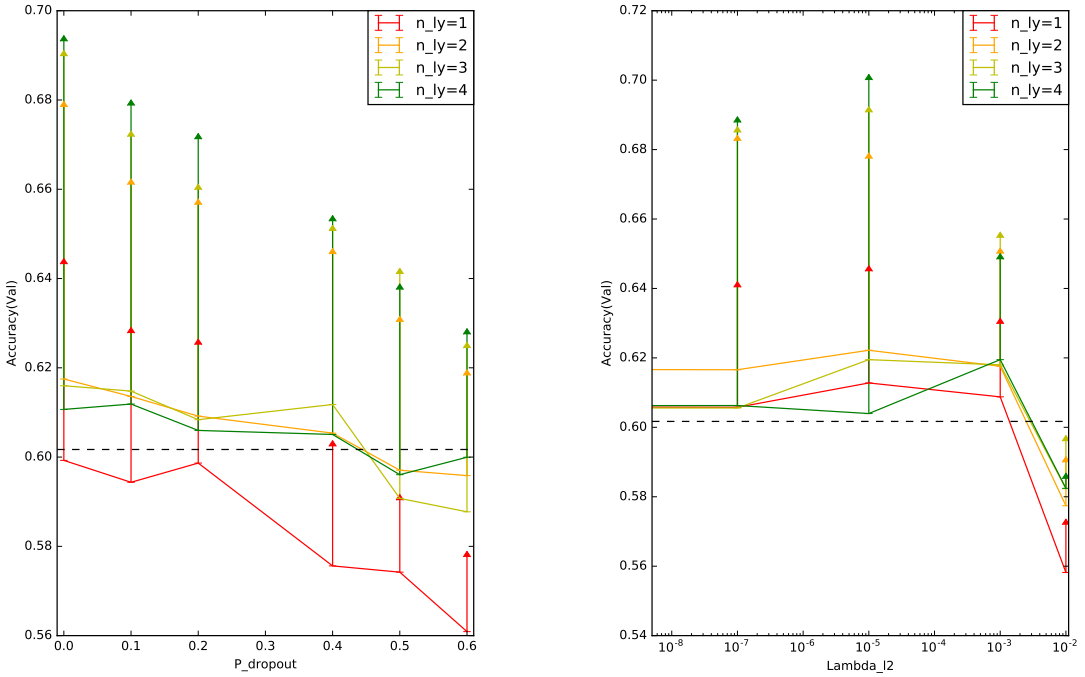


Figure 6: Classification accuracy of LSTM - operate via *tf.nn.**dynamic**_rnn*- with control of gradients by norm-clipping along cell memories. White bars are the gap between performance between training and validation set. Yellow dashed line is the one for baseline LSTM without gradient control.

2.b.3 Regularisation

To counter against overfitting problem of gated RNN such as LSTM with extended parameter space, we need regularisation method to manage it. We introduced two methods - forget bias initialisation and gradient clipping - working as boiling chips to the sequential feed forwardings and gradient backwardings. However, they do not directly intervene on how the activations actually shape the parameter. Hence in this subsection, 2 typical regularisation methods are introduced: Dropout layer wrapping, and L_2 norm penalty[7].

[7]In this part, we only apply them with 1 layer of LSTM, but later in section 2.c we will revisit this for deeper models



(a) Dropout

(b) L_2 norm penalty

Figure 7: Classification accuracy of LSTM with Dropout or L_2 regularisation and various number of recurrent layers(n_{ly}). Arrows indicate the gap between training and validation set learning accuracy. Black line is performance of the baseline LSTM without regularisation.

Dropout [8] Dropout wrapping drops out P_{DO} probability of activation transfers in a layer. The drop-out mask is arbitrarily designated to each mini-batch of data. Hence the dropout wrapping supports diverse representation in the wrapped layer as only portion of its parameter space will be updated by that mask, and hence the layer has mixture of experts [7] as much as the number of mini-batches the model has. Since the dropout wrapping makes a trade off between the diversity and the capacity of usable parameters for transfer between layers, its effectiveness is better found on the models that the representation capabilities are sufficiently large owing by having bigger or deeper parameter space.

Figure 7a shows that the dropout wrapping let the performance of single-layer LSTM drops down. Larger it drops, worse the performance be. This can be easily understood because the model loses the representational capability by not using the part of parameter space the model has and the deserved. It is like having smaller parameter space which decreases the learning performance, as we have seen in many experiments. Even when the LSTM becomes deep, the validation set accuracy goes down. However, such down turn dwindles for deeper model and training set accuracy is effectively regularised. For the quadraple-layer LSTM with $P_{DO} = 0.4$, sudden kink comes in and the performance nearly matches with the one without dropouts while the training set's accuracy is well regularised.

Performance drop by dropout is not always the truth or it is rather in opposite when the structure of model becomes complex. We will revisit this issue and also exploits along the report, especially in section 3.b.1, 5.a.2, and 5.a.2.

[8] In 0304/0(27324) by $LSTMbaseline_MSD - DropoutMulticell_LoopExp.ipynb$

L_2 norm penalty [9] L_2 norm regularisation penalties the objective function by L_2 norm of weight matrices which we want to regularise. Hence it is the intention of the penalty that we wanna regulate the weight matrices not to have large amount of value but not sparse.

It is generally well-known that L_n norm should not be preferred to use when recurrent neural network is used [8] but the dropout wrapping shall be. This is because L_2 norm selectively penalties certain elements in the weight matrices if its magnitude is high, and this high magnitude means that it is important to forming activation of the layer. For the gated RNN in particular, such penalty means the distortion of the sequential logic gate so that it would be better to avoid using, in theory. We check out this theory by applying the penalty system into the layers of LSTM regularising all the weights the model possesses, including gates.

In reality, L_2 norm penalty works well for the model as shown in figure 7b. By the number of the penalty controlling hyper-parameter to the objective function, the performance varies but in wide range of the coefficient λ_{L_2} it certainly makes the model better than without it. For some λ_{L_2}, the performance exceeds the one without regularisation. Interestingly, applying bigger L_2 penalty is found not to reduce the training set error in certain interval(see λ_{L_2} from 10^{-7} to 10^{-4}), so over-fitting is not resolved by this penalty. This is not the thing we saw in past courseworks where L_2 gives huge control on the training set accuracy. The reporter deduces that this may be caused by the nature of LSTM, where the gates control the sequential information flow and penaltised gates actually distribute their representational capability not focused on certain dimensions. For example, electric circuit may not take its resonance bandwidth to small for not losing important but less intense signals over the inputs. We normally take sufficiently large bandwidth and sometimes multi-modal bands if necessary. Reporter believes that the L_2 penalty plays a similar role to the above instance, a method to assure appropriate bandwidth over the gate parameter space[10]

2.b.4 Multi-layer LSTM cells

Recurrent neural network can be also stacked to achieve deep hierarchy. Multi-layer RNN is a stack where one layer of RNN sources another RNN's output. This provides multiple memory cell states for the gated RNNs as much as n_{layer} per each time-step compared to 1 for the single-layer one so that more complex logical representation, or decision boundary for classification in each time, is enabled.

There are two ways to stack up: simply pile up the layers as written above(unidirectional recurrence), or put two cells in one layer per each time-step having opposite recurrence direction(bidirectional recurrence) but sharing one common cell state.

Out of many versions of stack up, this report utilises the standard version of stack ups, multi-layer process of the cell state $c_t^{(n_{ly})}$ [9] as shown below. The hierachial recurrence flows from bottom of the hierarcy at t=0 to the top layer at t_{final}.

$$\begin{aligned}\underline{c_t^{(n_{ly})}} &= LSTM^{(n_{ly})}(\underline{c_t^{(n_{ly}-1)}}) \\ &= f_{activation}(W_T^{(n_{ly})} \cdot c_t^{(n_{ly}-1)} + W_C^{(n_{ly})} \cdot c_{t-1}^{(n_{ly})} + b_T^{(n_{ly})}) * Tanh(m_t^{(n_{ly})})\end{aligned}$$

where all the notations in this expression comply with the on in equation 1,

and $c_t^{(n_{ly}=0)}$ is x_t.

Birectional RNN has two directions of recurrence that share one memory cell at each time-step

[9] In 0304/1(27435) by $LSTMbaseline_regL2_MSD.ipynb$

[10] As every dimension of parameter has certain representation coming from the gradient descent, the reporter does not believe that L_2 penalty is too risky unlike the common belief.

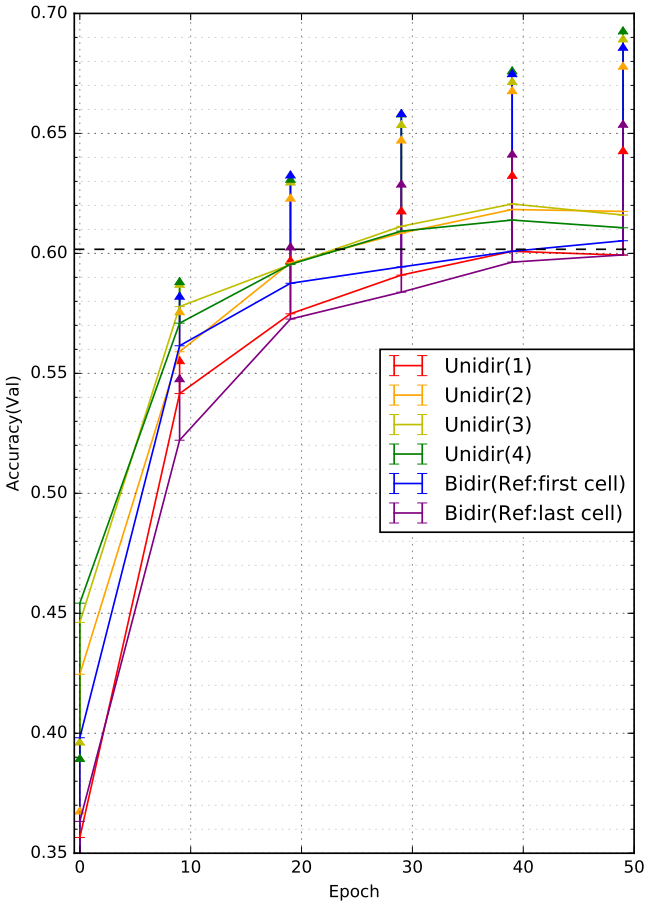


Figure 8: Classification accuracy of double-layered(n_{layer}=2) Uni-directional versus Bi-directional LSTM - structure which refers to cell states via both forward and backward directions. The best one sits at 0.62 in terms of validation set accuracy. Black dashed line is performance of the single-layer LSTM baseline.

as shown below. It is simply a combination of one straight and one reverse LSTM[10].

$$(\underline{c^{(n_{ly})}_{t,forward}}, \underline{c^{(n_{ly})}_{t,backward}}) = LSTM^{(n_{ly})}_{Bidirectional}(\underline{c^{(n_{ly}-1)}_{t,forward}}, \underline{c^{(n_{ly}-1)}_{t,backward}})$$
$$= f_{act}\Big((W^{(n_{ly})}_{T,fw} \cdot c^{(n_{ly}-1)}_{t,fw}, W^{(n_{ly})}_{T,bw} \cdot c^{(n_{ly}-1)}_{t,bw}) + (W^{(n_{ly})}_{C,fw} \cdot c^{(n_{ly})}_{t-1,fw}, W^{(n_{ly})}_{C,bw} \cdot c^{(n_{ly})}_{t+1,bw}) + (b^{(n_{ly})}_{T,fw}, b^{(n_{ly})}_{T,ww})\Big)$$
$$* Tanh(m^{(n_{ly})}_{t})$$
where $c^{(n_{ly}=0)}_{t,fw\ or\ bw}$ is x_t.

By having multi-layer of gated-RNNs, we have n_{ly} folds of parameter space per each time-step with n_{ly} hierachial gated transfer from bottom to top stack. Hence in theory, we anticipate to have better sequential representation on the top layer as we have series of logic gates giving more power to approximate as time-series programmes(or noisy channels we observe) that our MSD10 data is based on. On the other hand, having bidirectional gated-RNN means we enforce the memory cell states $\{m^{n_{ly}}_{t}\}$ not only assume that its observation is made by one sequential programme but also has in reverse order. This gives extra strength to find long-term dependencies

which unidirectional LSTM sometimes lacks in practice [13]. For example, analysing a sentence 'Only ten out of hundreds of you studying this course now can secure PhD positions to progress.' is to start from 'can' then goes backward to check 'ten' then goes forwards to see 'secure', then 'PhD positions' via dependency parsing. Without bidirectionality, it is nearly impossible to infer in both directions and without hierarchy it is hard to change the direction of our inference.

We implemented multi-layer LSTMs for both directionality to check whether the above idea works for MSD10 dataset. It is not as explicit as the natural language parsing, but we certainly know that songs also have long-term dependency and we do feel direction at some time t by previously knowing what happended in t_{past}.

The result presented in figure 8 shows than stacking has limited on the performance - see various number of layers stacked up for the unidirectional LSTMs. This results to a reasoning that one double-layer of gated-RNN may be enough to deal with dependencies presented in MSD10 dataset, so gate mechanism is that much power when we only have 120 time steps. Neither the bidrectional LSTM also does not higher the learning performance, as we might wonder if we listen songs forward and backward unlike we do for hearing conversations. We will also revisit this idea in section 5.a.2 with deeper models we suggest in later, as this may due to the lack of capabilities of the LSTMs on learning the training data(the training set accuracy staggers at 70% in figure 8). Another point to look at is that for the bi-directional gated RNN, it is not important to refer from either initial time step or final time step of the output state for the backward recurrence. We investigate this time-step reference issue further on this issue in section 2.d.

2.c Gated Recurrent Unit: a stable variant of LSTM

As the last part of exploring the hypothesis 2.b.1 in terms of gate mechanism and sequential inference, we may question if the structure of LSTM is sufficient-necessary condition to do this much good. The design of LSTM having four different weights and biases via having forget(F), input(I), gain(G), and the real activation may be too much in that it requires unnecessarily sophisticated regularisation and other hyperparameter design.

Gated Recurrent Unit(GRU) is a competitive variant of LSTM that reduces the number of gates from four to three [11]. This structure of GRU is as follows (fig 3c):

$$\underline{c_t} = (\underline{I} - f_{activation}(\underline{a_t})) * Tanh(m_t) + f_{activation}(\underline{a_t}) * \underline{c_{t-1}} \tag{2}$$

where a_t is an activation $W_T \cdot x_t + W_C \cdot c_{t-1} + b_T$ same as LSTM in equation 1,
but m_t is different as $W_T^{(H)} \cdot x_t + \left(c_{t-1} * R(a_t^{(R)})\right) \cdot W_C^{(H)} + b_T^{(H)}$
hence we have 2 gates(H,R) in addition to a_t.

The main difference of GRU cell to LSTM cell is that the recurrence of memory state is gated by reset(R) gate. Hence the current time step actually filters what to refer from the past, which means the control becomes more dynamic. In addition, the cell state is rather updated from the past cell state like vRNN with addition of a residual term element-wise multiplied by the memory cell of the current state, where the LSTM in equation 1 updates the cell state with *Tanh*-ed memory transfer. In a plain text, GRU works by seeing the past logic filter but complementing the inference by invoking some residual terms filtered by its memory state. On contrary, LSTM works by seeing its memory state and that memory state heavily refers to both input and past cell state. Hence GRU seperates the role of memory and cell state that LSTM complexly mixes up.

This intuitive theoretical aspect of GRU - we get help from the past to infer but the memory helps what we lost - has been reflected as a stabler version of gated RNN than LSTM[12]. Many

controls on LSTM cell discussed above [4][6] are actually came by boosting the performance of LSTM to GRU. Therefore, we take two experiment on GRU model with various parameter space and stack up style without any regularisation.

	Unidir	Bidir-last	Bidir-reverse
LSTM	0.62(0.68)	0.60(0.66)	0.61(0.69)
GRU	**0.64(0.70)**	0.62(0.69)	0.62(0.73)

Table 1: Classification accuracy table of validation set(training set) for double layered($n_{layer} = 2$) LSTM and GRU. 'Uni' has two layers of normal recurrent units from start to end, 'Bi-' has a reverse recurrent unit instead of the normal one. 'last' means the reverse unit refers to the hidden unit in the last time step for classification in usual manner, while for 'reverse' the unit refers the first time step's hidden unit. Dim(hidden) is taken as 30 for each cell.

First experiment on comparing GRU and LSTM in double-layer structure (table 1) shows that GRU is favoured in our architectures investigated. It is quiet interesting to see that GRU without any regularisation performs better by 1 or 2% than the LSTMs we coarsely optimised along the course. It is not very clearly understandable why GRU works stable without and control that we struggled with LSTM. However, as this report mainly discusses about the effective design for given gated-RNN, we leave this issue at this level.

$n_{ly} \backslash n_h$	5	10	20	30	50
1	0.49(0.48)	0.55(0.56)	0.60(0.62)	0.62(0.65)	0.70(0.63)
2	0.50(0.50)	0.58(0.59)	0.62(0.65)	**0.64(0.70)**	0.61(0.79)
3	0.50(0.50)	0.57(0.58)	0.63(0.66)	0.63(0.72)	0.59(0.86)

Table 2: Accuracy performance on validation set by uni-directional GRU with various number of layers wrapped(n_{ly}) and dimension of hidden unit($Dim(hidden)$ or n_h)

Second experiment on various depth and width of GRU model (table 2) shows that this gated RNN also its limit on performance, so too large or deep structure hampers the model with over-fitting problems. The main point we observe from this experiment is that GRU model does not sufficiently learn from the data point in that the optimal point only provides 70% of training set accuracy. If we push the model to study the training data better, can we possibly have better classification performance on the validation data set? This is the start point of the rest of this report.

To summarise on GRU, GRU not only benefits by reducing the memory to hold, but also works better on gradient control. Buying this, we take GRU as our baseline[11] gated-RNN structure.

2.d Are all the sequential memories necessary for the classification task?

We saw in figure 8 that it is not important for bi-directional gated-RNN(gRNN) to set which part of the output state to refer for classification. Does it mean that we can choose wherever time-step out of 120 to do classification so that backpropagation through time(BPTT) is really not the critical thing? Then what is the nature of the classification task for sequential data? In this end of exploring the concept of gated-RNN for sequential data[12], we set an experiment to see the importance of learning the sequential data by the full temporal scope.

[11] Double-layered unidiretional GRU with 32 dimensional cell state(n_h)

[12] For reference, *GRUclassfier.ipynb* is attached to the submission as a demonstration code for the gated RNN baseline.

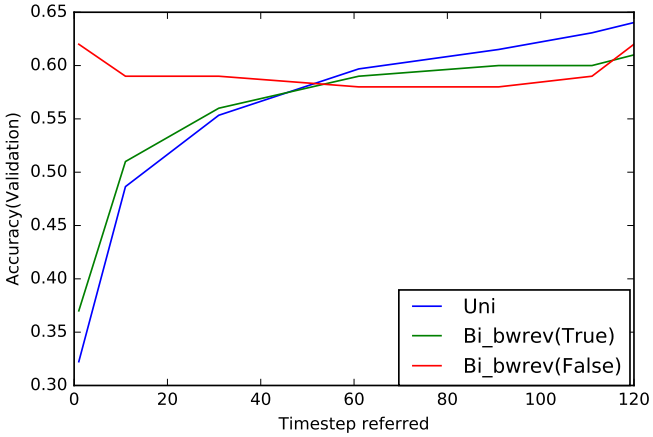


Figure 9: Classification accuracy for the validation set by double-layered(n_{layer}=2) Uni-directional GRU versus Bi-directional GRU at various time-step referred. For example, 120 at horizontal axis means the model takes the 120^{th} output(which is the last output) of the model-processed data for comparison with real label. *bwrev* for bidirectional GRU means that the output sequence of the backward recurrence is reversed. If it is reversed, then the classification vector from the model refers to the $output_{bw}$ at t=0 and $output_{fw}$ at t=120 when $t_{refer} = 120$.

Figure 9 below shows that training the entire 120 sequential memories are important. For example, when we set our objective function from the first 10th output from the uni-directional GRU, the model performs only 48% accuracy on the validation set. This is because another 110 cells are not trained - BPTT only updates the path the model recurs - nor used for inference(classification). $Bidirectional\ GRU_{True\ bwrev}$ refers to the final output for both recurrence($t = 0 + t_{refer}$ for backward and $t = 120 - t_{refer}$ for forward). This model also struggles on the reduced sequence length for look up. The only stable model with less look up is the bidirectional GRU which the classification layer looks up both forward and backward recurrence's output in the same time step, hence the performance is symmetric[13].

Consequently, it is found to be very important to process the sequential inference for the full time length from t_0 to t_{final}. Eventhough we classify only with final output($o_{t_{final}}$) and have nothing to do with any other past output, past cell and memory states contribute to make the output sensible to its input. With fully connected layer(FCL) model used in the previous coursework, it was not important to consider how the sequence is formed. Giving a sequential hierarchy over the temporal space thus proves to be important on applying gRNN to MSD10 dataset, which empirically support the idea assumed in section 1.

3 Improvement on sequential memory network

Along the discourse in section 2, we found the capability of the gated RNNs on the sequential data classification task over the layer-wise fully connected multi layer perceptrons, which we observed in the coursework 3. However, this report steps further to make improvements by dealing with issues we found on our GRU baseline: staggered training set accuracy around 70% for optimal specification.

[13]This actually provides that the machine can understand the song even if it listens the song reversely, unlike human

We assume this may be the limitation of the gRNN structure and introduce four methods to make the GRU greater: enforcing memories to learn sequential representation, and engineering the input space to let GRU better incoming of representations.

3.a Enriching temporal memory

We would like to make memory and cell states of our GRU model more useful, not just to get BPTTed gradients over tens of time-steps passed from the t_{final}. In this report, gRNN autoencoder pre-training and Attention mechanism methods are investigated.

3.a.1 Sequential Autoencoder

Auto-encoding our sequence via gRNN as a pre-training can be one of the solution. Deep neural network Autoencoder is a structure is optimised to yield the most similar output to the input. Out of many versions of gRNN, the one we use in this report - *Future predictor model*[16](figure 10) - is as below: a model $\mathcal{M}$ that reads the input the time-step t yields the output resembles to the input at the future time-step $t + \Delta$:

$$\mathcal{M} = \underset{\mathcal{M}}{argmin} \sum_{t} |\mathcal{M}(x_t) - x_{t+\Delta}|^2 \tag{3}$$

where the model $\mathcal{M}$ consists of encoding part we will use for the classification task, and decoding part on top which will be disposed after pretraining[14]

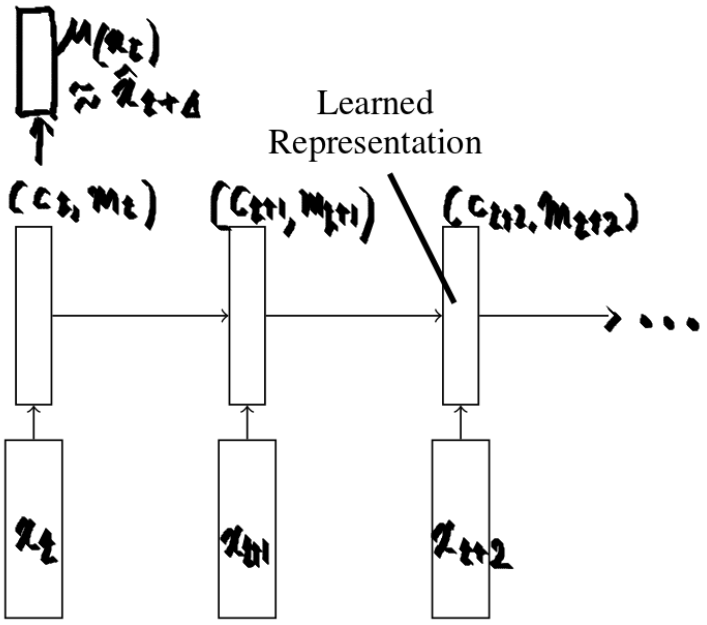


Figure 10: (From [16]) Sequence predictor model using gated RNN. The inputs $\{x_t\}$ become the target to be resembled in the time-step (Δ) past t in this diagram.

If the model has gRNN such as GRU near the top, memory and cell states shall have the representations for their time-steps and recurrent flow gives the sequential inference on mirroring the song data. This is because the model is forced to generate the data as close as possible

[14]Decoder is typically FCLs but using gRNN is also possible to use.

during the pre-training. It also means that pre-training would enable the effective use of the attention mechanism. If we get ride of the decoding part and top up another GRU layers, then the top ups can infer based on the states what the pre-trained encoder does best about learning the sequential representation about the data. Hereby we propose our second hypothesis. We also hope that the good quality of AE pretraining leads the stacked model to better validation accuracy performance.

Hypothesis 2 *Enforcing the sequential inference for each time-step improves the representation that memories possess hence better up the classification result.*

The moral of this hypothesis is that we expect the sequential representation - learned from the unsupervised auto-encoding pre-training - can be transfer to the classification task if the model shares the encoding part inferring sequentially[15].

3.a.2 Attention mechanism: highlight over temporal horizon

Attention mechanism is a method having an additional layer on top of a gRNN layer that enables the model to invoke on specific specific time-step over the temporal space. The attention mechanism used in this report is *soft global attention mechanism* provided in Tensorflow as *tf.contrib.attention_wrapper()*. The mechanism works as below[14][15]:

$$\underline{c_t^{(Attn)}} = f_{act}(\underline{\underline{W_T}} \cdot [\underline{c_t^{(gRNN)}}; \underline{cnt_t}] + \underline{b_T}) \tag{4}$$

where $c_t^{(Attn)}$ is the cell state made by the attention mechanism
from the gated-RNN's cell state $c_t^{(gRNN)}$ concatenated with the context vector
$cnt_t = \underline{\underline{A}}_{(:,t)} \cdot c_t^{(gRNN)}$ and $\underline{\underline{A}}$ is the attention matrix of size $(t_{final} - t_0, t_{final} - t_0)$.
Each attention vector[16] $\underline{\underline{A}}_{(:,t)} = \frac{e_t}{\sum_{t_0}^{t_{final}} e_t}$ is a softmax of *score* vector e_t,
where e_t is caculated as $c_t^{(gRNN)} \cdot \underline{\underline{W_a}} \cdot \underline{\underline{c_{t_0 \sim t_{final}}^{(Attn)}}}$ in Tensorflow - many variants exist.

This report also cites figure 11 to help the understanding of the attention mechanism.

As shown in the equation above, the attention mechanism is essentially a pseudo-recurrent neural network layer which recurs the past from the context vector, not the last attention cell state. As the context vector is made by the attention matrix invoking the whole gRNN cell states and attention cell space, each attention cell state globally refers the sequential representations. Hence there is no direction to the recurrence but it is a map from the whole temporal space to the particular time-step, only where the gRNN layer provides the direction of time evolves.[17].

The theory naturally gives very intensive back-up for the recurrent cell states in terms of recognising long-term dependencies from either direction(forward or backward of time evolution).

[15]We actually found that a good enforcement strategy for each cell-state may be measuring there cross-entropy error for classification rather than the sequential AE, as the representations learned from sequential AE pre-training is not theoretically guaranteed to transfer to the classification task. We discuss this issue in section 5.b, especially in 5.b.2.

[16]Cell can have multiple attention vector(by having multiple $\underline{\underline{W_a}}$ as like kernels of CNN) and can be specified by *attn_length* in tf.contrib.AttentionWrapper

[17]Side fact from the equation: The attention mechanism is computationally expensive not only due to global look up, but also the repetitive update of the attention weight matrix $\underline{\underline{W_a}}$, which should be guaranteed to provide sufficient dimensions for the attention cell states to properly get the score vectors. Also, it is only applicable to the off-line framework, where the time-length of all episodes are fixed, by the attention weight matrix to be formed.

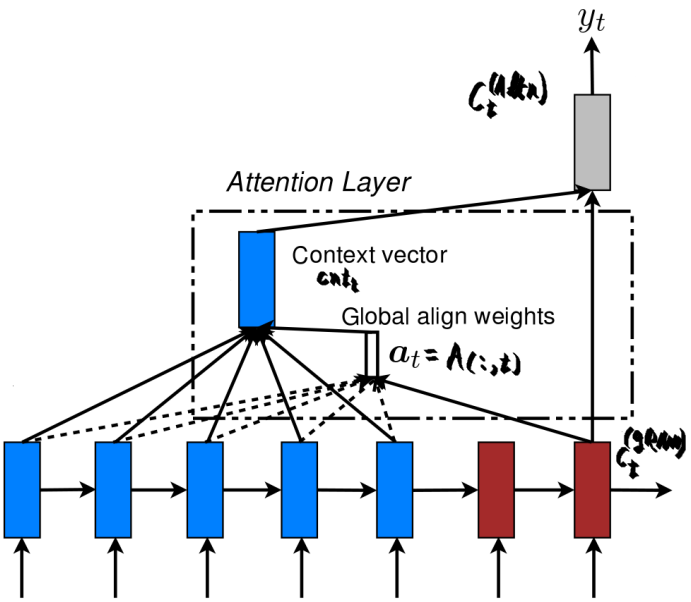


Figure 11: (From [15]) Diagram of Attention mechanism. Attention layer defines an attention vector(a_t) for each time step by looking over the whole cell states of gRNNs($\{h\}_{0\sim t}$) - so far we have noted them as $\{c\}_{0\sim t}$ -, which gives the context state c_t for the time step t that works as a gate for the final cell state($\tilde{h}_t$) at time t.

However, it has a critical limit to solely be applied to our classification task. This is because the attention layer deeply cares about all the outputs along the temporal axis, but we do only care about the one at the final time-step.

$n_{ly} \backslash Dim_{Attn}$	None	1	3	5	10
1	0.62(0.65)	0.615(0.661)	0.622(0.660)	0.641(0.667)	0.620(0.665)
2	0.64(0.70)	0.618(0.681)	0.623(0.691)	0.643(0.701)	0.619(0.702)

Table 3: Classification accuracy - validation(training) - of 1 layer GRU with attention cell state with various dimension(None to 10). Compare to table 2 and 1, the performance already matches with the one from the best multi-cell GRU. However, the performance does not improve from there. Standard over-fitting problem appears when the attention state length becomes 10.

Table 3 shows that without sensible curriculum to look over the whole outputs, the attention mechanism does not work - it is rather fortunate that such 'un-taught' attentions does not hamper the classification task. This brings the necessity to let each output useful eventhough our really interest is on the final time-step, which the sequential auto-encoder can largely contribute. We discuss about sensible auto-encoding and consequently effective application of the Attention mechanism in section 5.b.1.

3.b Hierarchial approach

While the hypothesis 2 above considers more effective training on the gRNN cells by tuning the upper-side, we would also like to explore any sensible embeddings that gRNN can get for better learning other than the raw input.

3.b.1 Convolutional Neural Network as Embeddings

Motivated from recent progress in language modeling via Convolved word embeddings [17], we suggest to elaborate the input space of the recurrent neural network as the output space of a convolutional neural network(CNN).

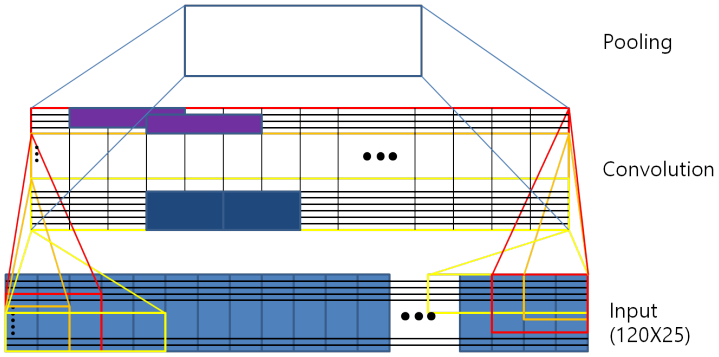


Figure 12: Structure of Convolutional Neural Network(CNN). CNN is usually consist of two layers: convolutional layer by kernels(1,2,3 in the diagram), and pooling layer for compression(1,2).

CNN is a structure giving a convolution to the input with kernels, and pools it if necessary(fig 12):

$$Conv(\underline{\underline{x}})_{Kernels} = \underset{\forall(i,j)}{Concat}(Pooling_j(\underline{\underline{x}} \otimes Kernel_i))$$

where $\otimes$ is the convolution operator giving

standard scalar product $(\underline{\underline{x}} \otimes \underline{\underline{Krn}})_{a,b} = < \underline{\underline{x}}^{[b,b+Dim(Krn)_y]}_{[a,a+Dim(Krn)_x]}, \underline{\underline{Krn}} >$ in 2D.

Pooling is a sampling map from the given pool size to one the method designates , such as maximum one, average of L_n norm, or else.

Each kernel gives convoluted representation of the input as much as the number of output channels - in our case the input of size $(Dim(\underline{\underline{x}})_{time}, Dim(\underline{\underline{x}})_{feature}) = (120, 25)$ for each MSD10 data - then pooling reduces the convoluted space by the stride of pools for each dimensionality, which we then concatenate them along the output channel.

Having CNN under the Recurrent unit, we now have a convolved input space for the recurrent unit that enables to perceive how the sequence is shape over the time horizon as much as the size of the kernel of CNN. By doing so, we hope that such look up give better underground embeddings for the sequential inference model.

Hypothesis 3 *Embedding the Convolutional Neural Network under Recurrent Neural Network gives better representation for recurrent memories, hence the model better performs on the classification task.*

In this section[18], basic performance test for CNN model onto MSD10 data is conducted to check how the models perform with various hyperparameters of CNN - number of output channels per kernel, kernels size, number of CNN stacked, and Dropout probability - before we port this onto our GRU baseline. The moral of applying CNN embedding to our model is to provide each GRU cell representations from more one-time step by convolution process.

[18]As we intensively investigate CNN in the coursework2, we leave this part without further theoretical details. Please refer to the second coursework report for details if interested.

Number of Output channels Table 4 shows that the number of output channels provided is a key factor to ensure the performance, as it defines the size of hidden variables that CNN possesses. As like other relatives, this quantity has the limit on improving the performance, which is 24 for a single-layer CNN in this case. Dropout for the single-layer CNN is found to be harmful giving a huge downturn on both validation and training set accuracies.

$n_{ch} \backslash P_{dropout}$	0.0	0.1	0.3	0.5
6	0.45(0.57)	0.45(0.55)	0.44(0.50)	0.42(0.46)
12	0.48(0.69)	0.47(0.62)	0.45(0.57)	0.45(0.51)
24	0.49(0.72)	0.47(0.70)	0.45(0.56)	0.44(0.54)
36	0.49(0.76)	0.47(0.70)	0.44(0.59)	0.35(0.40)

Table 4: Validation(Training) learning accuracy of single-layer CNN connected to a 30-perceptrons layer ($n_{perc} = 30$) with various number of output channels and dropout probability. Dropout is applied at the fully-connected layer, the output state of CNN.

CNN with multiple kernel size: temporal axis Tables 5 shows that a combination of kernels can also gives better resolution of the input space for the classification task. With single kernel with size 3 for the temporal axis performs 0.49(0.72) by best in the table 4 whereas the one with two kernels of size 3 and 5 gives 0.50(0.63) and 0.52(0.65) with three kernels of size 3,5, and 7. The number of output channels or the number of perceptrons in FCL on top of the CNN drive the over-fitting problem as they enlarge the hidden variable space.

$n_{ch} \backslash n_{perc}$	30	60	90
6	0.50(0.63)	0.47(0.71)	0.43(0.79)
12	0.50(0.72)	0.46(0.85)	0.44(0.93)
24	0.49(0.77)	0.46(0.89)	0.46(0.95)
36	0.49(0.77)	0.46(0.90)	0.46(0.94)
72	0.47(0.91)	0.47(0.94)	0.47(0.97)

(a) $n_{Kernel} = 2$: shape of kernels=$(Dim(input_t), 3\ and\ 5)$

$n_{ch} \backslash n_{perc}$	30	60	90
6	0.52(0.65)	0.47(0.73)	0.45(0.80)
12	0.50(0.75)	0.46(0.87)	0.45(0.93)
24	0.50(0.76)	0.48(0.89)	0.47(0.93)
36	0.48(0.87)	0.47(0.89)	0.47(0.94)
72	0.47(0.93)	0.47(0.97)	0.46(0.96)

(b) $n_{Kernel} = 3$: shape of kernels=$(Dim(input_t), 3\ and\ 5\ and\ 7)$

Table 5: Validation(Training) learning accuracy of single-layer CNN connected to a various number of perceptrons layer (n_{perc}) with various number of output channels when the number of kernel size n_{Kernel} is more than one. The tables are the ones without dropout which worsens the performance for the single-layer CNN.

Depth of CNN with effect of Dropout While the dropout wrapping hinders the performance of the single-layer CNN in table 4, multiple-layer CNN rather favours with dropout as shown in table 6. When the model gets deeper, dropout becomes a raiser of the validation set accuracy not only works as a reducer of the training set accuracy. This may be an empirical

proof of the dropout mechanism that the dropout in the multi-layer model provides a diverse mixture of experts in each layer as different subspace of the layer learn different mini-batch of data [7].

The effectiveness of the dropout wrapping depending on the depth of the model is observed several times along the experiments conducted in this report, hence we utilise this technique as the main over-fitting controller of our deep models.

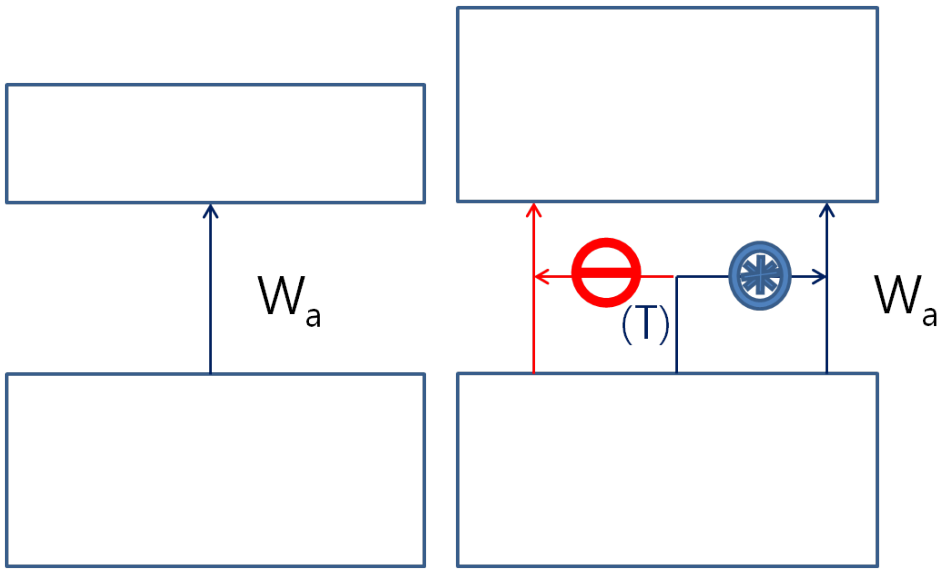
$n_{layer} \backslash P_{DO}$	0.0	0.1	0.3	0.5
1	0.45(0.57)	0.45(0.55)	0.44(0.50)	0.42(0.46)
2	0.51(0.93)	0.52(0.76)	0.52(0.74)	0.51(0.76)
3	0.51(0.97)	0.52(0.84)	**0.53**(0.81)	0.51(0.81)

Table 6: Classification accuracy table of base set-up[19]CNN with various number of CNN layers(convolution + pooling) and dropout probability.

To sum up, there is effective sets of CNN hyperparamters in terms of the models' performances determined by the depth of CNN layers(2 or 3), output channels per kernel(6 to 24), dropout probability(0.1 to 0.3 if the multiple layers are stacked), and the dimension of the FCL at the top.

3.b.2 Sensible bridge between gRNN and CNN

Applying the CNN into our model, we left with a problem of linking CNN and gRNN. We typically set FCL layers on top of CNN to represent global perceptions from local invariants found by CNN. This report introduces a variant of FCL that may capture the convolved output better hence improves the GRU performance by providing better embeddings.



(a) Simple fully connected layer: Bipartite affine transformation

(b) Highway layer: Affine transformation with a gate($*$) and a residual gate($-$)

Figure 13: Fully connected layer structure. This layer will be used as a bridge between CNN embeddings and GRU for section 5.

[19]Kernel($Dim(input_t), 3$) each having 32 output channels, pooling stride is (1, 2) with pool size($Dim_{feature}$,2), and 30 fully connected perceptrons on top as above.

Highway network layer, which is essentially a gated fully-connected layer(FCL), is claimed to provide better resolution of representation then simple FCL multi-layers perceptrons energy measure(or sFCL)[18]. By topping this layer on top of CNN, we expect to lose less local representational power of CNN when its final pooling layer converts to a fully-connected layer(FCL). This notion has been proven to work in Natural Language Model related learning tasks, where the embeddings of input space of sequential inference model are sophisticatedly vectorised[20][21]. The exact expression is as below:

$$\underline{\underline{y}} = f_{activation}(\underline{\underline{x}}\underline{\underline{W_a}}) * T(\underline{\underline{x}}\underline{\underline{W_T}}) + \underline{\underline{x}} * \big(1 - T(\underline{\underline{x}}\underline{\underline{W_T}})\big)$$

where x, y are input, output matrix respectively(which are tensors for in each mini-batch), W's for weight matrices(above equation omitted biases but they essentially do exist), * for element-wise multiplication, $f_{activation}$ for standard FCL, and T, C are target gate and carry gate which are sigmoid transfers in common. Compared the above expression to the simple FCL, $\underline{\underline{y}} = f_{activation}(\underline{\underline{x}}\underline{\underline{W_a}})$, the Highway layer essentially gates the activation so that every embeddings can effectively contribute by some measure as what we saw in gated RNNs. We expect the Highway layer's residual gate(-) can bring up useful representations what sFCL forgets.

Another aspect of the highway layer is that it cannot reduce the size of the feature space as it gates the input by element-wise product in the right-hand side of the equation, although this may indicate that the highway layer can perceive what the residual of the input(right term of the expression) for the affine transformation(left term), similarly as Residual network layer[19].

$n_{ly} \backslash Type_{FCL}$	sFCL	Highway
1+1	0.49(0.65)	0.50(0.66)
1+2	0.50(0.68)	**0.51(0.66)**
1+3	0.50(0.74)	0.51(0.66)

Table 7: Validation(Training) learning accuracy of single-layer CNN connected to n_{ly} of sFCL or Highway layers with $n_{perc} = 30$. Single FCL on this CNN gives 0.49(0.72) in table 4

In this section, the simplest structure of the suggested model (1 CNN - Highway/FCL - 1 GRU) in section 4 is trained for reference, which will be discussed in section 5 in detail. According to the table 7, the highway layers in the simplest model improves the learning performance by mere margin. Two things to note in this base camp experiment however, are that (1)this sophisticated gated FCL prevents the model from over-fitting by depth of FCL layers(whereas sFCL with $n_{ly} = 3$ performs lower than that of $n_{ly} = 2$), and (2)does not increase the computational cost[20].

We will revisit the bridging control in term along the whole section 5, as such (1)sFCL or Highway layer mixes up the temporal knowledge - hence we implement 'temporally' connected layer in section 5.a.2 -, and compressing the feature space may be crucial for the sequential autoencoder pretraining in section 5.b.1 which the Highway layer cannot.

4 Suggestion of Network models and Claim of problems

In section 2 of the report, we have discussed the effect of the gated-RNN on the sequential data for classification over FCL. Following that, section 3 suggests the methods to improve on what the simple gRNN lacks of - better understanding on the data and use of sequential states under

[20]about 5 $\sim$ 10% time taken per 5 epochs for both models with sFCL or Highway layers

the classification task. Thereby two hypothesis(hyp 2, 3) are proposed - to resolve the problems on applying sequential neural network inference model for the classification task. They are (1) each state is too localised by looking up only one time-step, (2) output states are not used except the last one.

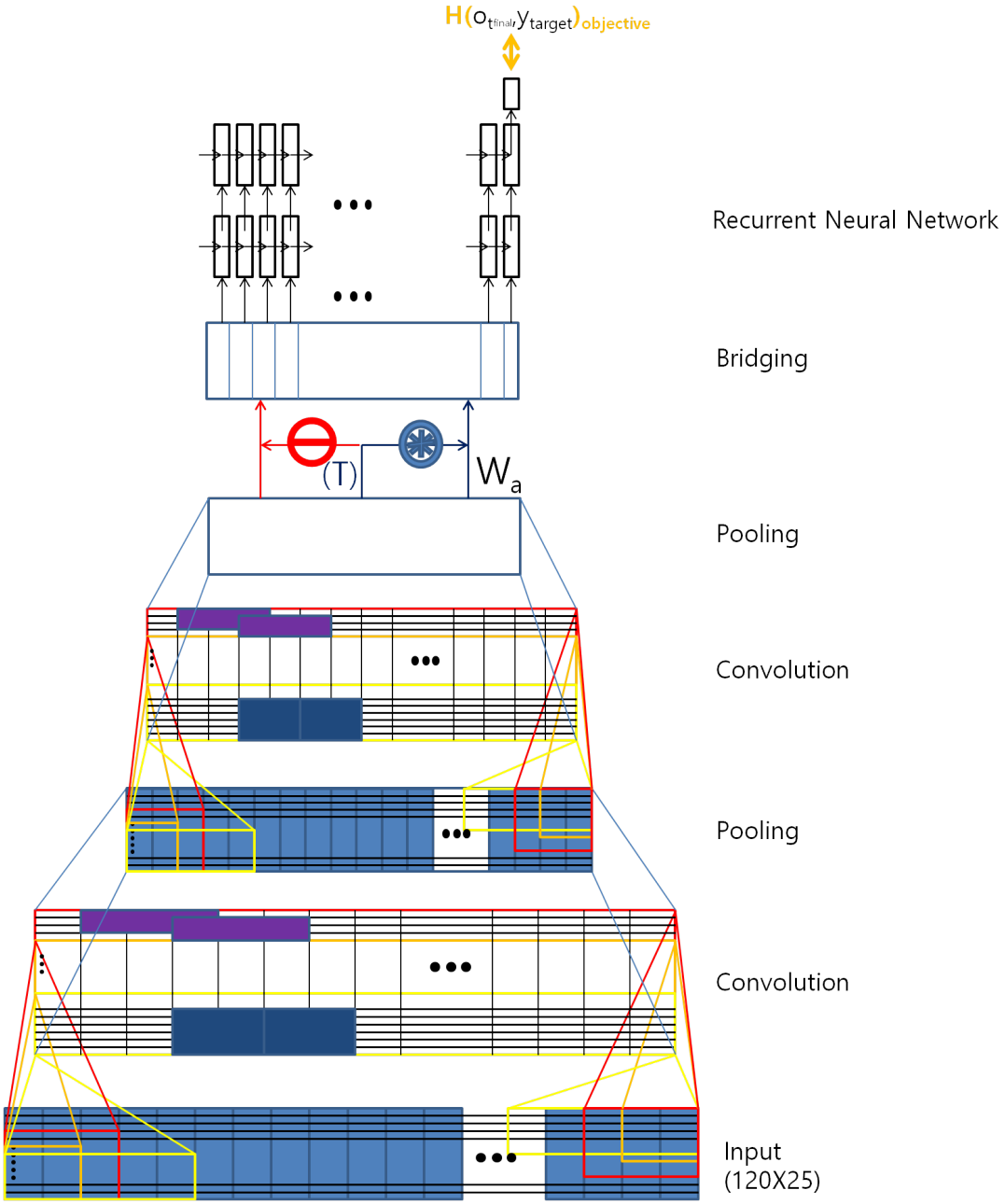


Figure 14: Structure of the suggested model: Multilayered GRU-Bridged-embedded by CNN. In the previous sections, each part of the model has been introduced, and the following part of the report examines how they collaborate to make progress on the classification task. Hence engineerings on each building block are examined with this combined model architecture from now on.

For the rest of this report, we implement the improvements introduced in the section 3 into our GRU baseline model suggested in section 2.c. It is to examine how the new models(figure

14) behave on MSD10 classification task hence resolve the two problems summarised in the above paragraph. By observing the performance with in-and-out control of model design and hyperparameters, we expect to prove that (1)sensible embedding for the sequential inference model, and (2)enforcing the memory and cell states of the sequential inference model can improve the classification task by attacking what the baseline model is troubled with.

The new models shall have layers of GRU with CNN-embedding and bridge layers between them, and the attention mechanism with sequential AE pretraining shall be provided. In terms of performance, our goal is to exceed the one we got from previous models summarised in table 8.

Model \ Perform	$Acc_{Val}(Acc_{Tr})$	Comment
Fully Connected Layers	0.45(0.46)	Plain multi-layer perceptrons
$vRNN_{Grad.Clipped}$	0.46(0.49)	Deficiency on long-term memory retrieval force
$LSTM_{2ly}$	0.62(0.68)	Temporal memory boosts the retrieval force
GRU_{2ly} *baseline*	0.64(0.70)	Limited to be trained further
$CNN_{3ly+'SAME'pad}$ +DropOut.	0.53(0.81∼0.97)	Use as embeddings, expect to boost $Acc_{training}$

Table 8: Summary of best learning performance of various models discussed in this report. Specification of the models are footnoted below.

From now on, we notate the specification of our model as follows:

$$GRU(number\ of\ layers)^{Uni\ vs\ Bi-directional}_{Attention\ state\ length} - Bridge\ as\ FCL\ or\ HW(number\ of\ layers)$$
$$- CNN(number\ of\ layers)(number\ of\ channel_{(out)}\ per\ kernel, size\ of\ kernel, dropout)$$
$$\_pretraining\ via\ Autoencoding(which\ part)$$

5 Result and analysis

In the first part of this section(subsection 5.a), CNN embedding's performance with various specification of hyperparameters is investigated. In the following part(subsection 5.b), sensible strategies to utilise the cell states over the whole temporal space are investigated.

5.a Difficulties on applying CNN as embeddings

The reporter first starts the experiment by running Double-layered GRU with attention units of which embeddings is Double-layered CNN highway-ed up to GRU. However, the result was daunting. Over the range of hyperparameters(Dropout probabilities, depth of highway layers) - number of output channels for CNN is fixed as 32 with kernel size $(Dim(input_t), 3)$, this baseline does not perform over 0.62, 0.69 for the accuracy of validation and training set, respectively. The performance, which matches to that of optimised GRU without CNN embeddings, means that CNN embeddings did not contribute to the sequential inference. Not only by staggered validation set performance, but the training set accuracy ceiled under 0.70 . CNN embedding may not help GRU learn the time series.

5.a.1 Pooling, Kernels and Depth of CNN

Subsequently, the reporter checks the performance of CNN itself without GRU. It turned out that CNN with 'VALID' padding - refer to section 3.b.1 - does not learn as much as that of

'SAME' padding. In 'VALID' padding, the training accuracy staggers at around middle of 0.6 in various hyperparameters space(even when the number of output channels double to quadraple up from 32). Although the validation set's accuracy gives similar performance and computational time was significantly reduced, the reporter deduce the representational power of CNN is too weak with this model baseline, hence investigate further into two ways: pooling method and size of kernel.

Firstly, the nature of pooling might matter. As the length of time-step significantly reduces by the stride of it and we applied multiple of CNN layers - for example when we do max-pooling twice with stride 2, only have 30 dimension for the temporal axis left from 120 dimension - the reporter hypothesized that the length of stride for max-pooling excludes out what we need to take. The default setting of the stride was 2, the reporter set up this as 1 and redo the experiment. However, the performance decreases. Obviously, max-pooling with stride 1 yields many duplicated representation hence results in poor learning performance and longer stride loses the representation which we concerned above.

n_{ly} \ stride	1	2	3
2	0.11(0.11)	0.51(0.59)	0.47(0.56)
3	0.10(0.10)	0.52(0.64)	0.47(0.59)

Table 9: The CNN kernels output (32,64,128) output channels for each depth.

Hence the reporter investigates about pooling methods itself and implemented L_p-norm pooling[22] for CNN layers for sake of providing broader representational diversity on the pooling layer[23]. The L_p-norm pooling outputs the average L_p norm of the pool zone rather than taking the maximum one.

$$u(\underline{\underline{x}}) = (\frac{1}{N}\sum_{i=1}^{N} |\underline{x}_i|^p)^{\frac{1}{p}}$$

where the output vector u among the pool $\underline{\underline{x}} \in \underline{x}_i$ is the average L_p norm of the pool elements from convolutional layer. This pooling method geometrically ensures each representation from CNN has L_p norm boundary line - if p=1, then it is a hyperplane and p=2 gives a hyper-ellipsoid ([22]-refer to figure 2). The reporter takes L_2 norm pooling into CNN structure. As shown in the table10, there is about 5(21) improvement by applying L_2 norm with max-pooling for training(validation) set, meaning that we can improve pooling layer by including information of non-maximum receptive units from the convolutional layer. It is much interesting by two folds: the double-up strategy for output-channels with sole max-pooling did not work in table 4, and the table does not clearly show which is the hot spot over the heat map(table 10).

n_{ly} \ Pooling	Max	L_2	Max+L_2
2	0.51(0.59)	0.54(0.95)	0.47(0.65)
3	0.52(0.64)	0.59(0.75)	0.57(0.85)

Table 10: The CNN kernels output (128,64,32) output channels for each depth. Dropout with $P_{DO} = 0.1$ is applied for every CNN layer and input space.

Hence we further investigate the Max + L2-norm poolings model with various output channel numbers in table 11. Again, the GRU with CNN embeddings model does not agree on our conjecture made in CNN model: L_2 pooling in CNN is reasonable to boost up the learning performance.

	$n_{ch}^{(ly3)} \backslash n_{ch}^{(ly2)}$	32	64	128
$n_{ly} = 3$	32	0.61(0.78)	0.57(0.85)	0.61(0.81)
	64	-	0.60(0.83)	0.60(0.83)
	128	-	-	0.56(0.91)[21]
$n_{ly} = 2$	-	0.54(0.96)	0.54(0.95)	0.56(0.79)

Table 11: when $n_{ch}^{ly1} = 128$.

Secondly, the reporter investigates whether the dimension of the kernel for input mode must be equal to the dimension of the input. So far in this section, we only take kernels looking up the whole dimension so that no local representation is allowed for each time-step. Considering that CNN with allowance on local look up via 'SAME' padding learns the training set well, we apply several kernels of dimension for x_{input} less than 25 for MSD10. Ground of this conjecture is that some of key patterns of sound may be localised in some frequency intervals which kernels can perceive. For example, low frequency may catch up the sound of bass or cello. The sound pattern must be different from instruments and genres they are played so that the convolved space may be distinctive to give better resolution for the low-level pattern of songs. Theory is sound, but the computational limit really hits for experimenting this idea so the reporter leaves this investigation with an experiment on 1 layer CNN shown in table12.

Kernels	$(Dim(x_i)_t = 25, 3)$	$+(14, 3)_{stride6}$	$+(8, 3)_{stride6}$
$Acc_{Val}(Acc_{Tr})$	0.54(0.96)	0.56(0.97)	0.57(0.98)

Table 12: Learning performance of double-layer CNN embedding models[22]with smaller kernel add ups. Strides of feature space for additional kernels are 6 to avoid computational crush on DICE.

Due to the time cost, it was unable to conduct the experiment further with variety of kernel sizes on the input dimension per each time step. However, it still gives that the convolution along the feature space may also affect on better learning, as we conjectured in the above paragraph.

Finally, we put our above findings on CNNs into our model architecture(GRU-Highway-CNN) and summarise in table 13 below. As it shows, suggested models work daunting in terms of the accuracy for validation set, although it seems to learn on the training set deeper than the one without CNN embeddings. Most shocking thing is that CNN embedding with L_2 pool deters the validation accuracy even-though for CNN it ran better than the one without(see second and third model in table 13.

Furthermore, the overall performance we saw is almost same as those without GRUs on top in table 11, which means that GRU - the sequential inference structure with memories - does not get the distributed representations from CNN against our hope. Attention mechanism still gives any improvement. We examine these further in the next subsection.

5.a.2 Enforcing CNN embeddings to represent sequentially

According to the table 13, something is still wrong within the model eventhough we improved CNN - the sequential learning structure does not improve by temporally local representation.

[21]Exclusively run for 100 epochs as 50 epochs were not enough for this case. Maximum Acc_val was reached at 40th epoch having 0.61(0.71) as $Acc_{Val}(Acc_{Tr})$.

[22]10% dropout is applied for each convolutional layer and $n_{ch}^{(out)} = 128, 64$ for the first and second convolutional layer, respectively. Strides for temporal space is set as 2.

Model \ Perform	$Acc_{Val}(Acc_{Tr})$	Comment
$GRU2^{uni}$-noCNN	0.64(0.70)	Section 2.c
$GRU2^{uni} - Hw2 - CNN2\_baseline$	0.62(0.69)	w/o L_2 pool
$GRU2^{uni} - Hw2 - CNN2(out_{ch} : [64, 128], P_{DO} : [0.1] * 3)$	0.53(0.99)	"Model 1"
$GRU2^{uni} - Hw2 - CNN3(out_{ch} : [32, 64, 128], P_{DO} : [0.1] * 4)$	0.54(0.99)	"Model 2"
$GRU2^{uni} - Hw2 - CNN3(out_{ch} : [128, 128, 128], P_{DO} : [0.1] * 4)$	0.56(0.98)	"Model 3"
$GRU2^{uni} - Hw2 - CNN2(Kernels : [25, 3] + [14, 3]_{strd7}$, $out_{ch} : [64, 128], P_{DO} : [0.1] * 3)$	0.56(0.97)	Kernel with $Dim_{ft} < 25$
$GRU2^{uni}_{Attn5} - Hw2 - CNN2(out_{ch} : [64, 128], P_{DO} : [0.1] * 3)$	0.53(0.98)	Attention

Table 13: Summary of learning performance of suggested model with regards to CNN embeddings for MSD10 dataset. Each model runs for 100 epochs. First to models are the baselines, and rest are pooled by both max and L_2 norm. All double-layered GRUs used in this table are unidirectional and with optimised $n^{(final)}_{hidden}$ - dimension of $output_{t_{last}}$ of GRU - as 32.

What it means is that our model still does not get trained as well as CNN, while not validated as well as GRU. In this part of report, the output of CNN is investigated to represent its invariant sequentially by applying temporally connected layer on top, and CNN auto-encoder.

Keeping temporal embeddings as local Typical highway layers or sFCLs are not appropriate to use on our sequential learning task as a bridge between CNN and GRU. This is because these FCLs simply connect all the input space to make output space so that data at t = 100 is not guaranteed to contribute more on inferring output at t = 101 than the one at t=0. By this sense we might fail to keep the local pattern representation made by CNN in local cell at GRU. This report hypothesizes that this forces GRU to loss all the temporal representational resolution hence just inherits the performance of CNN-only structure. By removing that FCL[23], we predict that the resultant performance goes into new shape.

In addition, this report suggest *temporally connect layers*(TCL) as an alternative to FCL to deal with this issue. It is simply a layer where the features in the same time-step are connected. Concretely, when we have an input batch $\underline{\underline{X}}$ of dimension $[n^{(batch)}_{data}, Dim(time), Dim(feature)]$, sFCL gives the output

$$\underline{\underline{Y_{mat}}} = \underline{\underline{X_{mat}}} \cdot \underline{\underline{W}} + \underline{b}$$

where $dim(\underline{\underline{X_{mat}}}) = [n^{(batch)}_{data}, Dim(time) * Dim(feature)]$. TCL on contrary updates the input as

$$\underline{\underline{Y_t}} = \underline{\underline{X_t}} \cdot \underline{\underline{W_t}} + \underline{b_t}$$

where $Dim(\underline{\underline{X_t}}) = [n^{(batch)}_{data}, Dim(feature)]$ and $t \in [t_0, t_{final}]$. As the output at timestep t only refers to the input at time-step t, locality of the representation made by CNN will not be lost. Similarly, temporally connected highway layer(THw) will give

$$\underline{\underline{Y_t}} = f_{activation}(\underline{\underline{X_t}} \cdot \underline{\underline{W^{(h)}_t}} + \underline{b^{(h)}_t}) * T(\underline{\underline{X_t}} \cdot \underline{\underline{W^{(T)}_t}} + \underline{b^{(T)}_t}) + \underline{\underline{X_t}} * C(\underline{\underline{X_t}} \cdot \underline{\underline{W^{(T)}_t}} + \underline{b^{(T)}_t})$$

Table 14 shows the result of the above two methods. It turns out that the fully connected layers are better not to be put between CNN and RNN whether or not the bridge is temporally connected, as zero FCL models make the best improvement. The temporally fully connected

[23]It is absolutely possible because we can simply put final output channels into the input dimension for GRU. For example, if the $n^{(out)}_{ch} = 64$ for the top CNN, n_{input} for each time-step that GRU takes is 64.

Model \ Perform	$Acc_{Val}(Acc_{Tr})$	Change
Model 1 but zero Highway layer	0.61(0.82)	+0.08(-0.11)
Model 3 but zero Highway layer	0.61(0.88)	+0.05(-0.10)
Model 1 with 2 THws	0.55(0.96)	+0.02(+0.03)
Model 1 with 2 TCLs	0.55(0.96)	+0.02(+0.03)
Model 3 with 2 THws	0.57(0.92)	+0.01(-0.01)
Model 3 with 2 TCLs	0.58(0.94)	+0.02(0.)

Table 14: Learning performance with modified models for FCLs by not having FC-layers(First two models) or having temporally-connected 2 layers. Compare the performances to the models with FCLs in table 13.

bridge models only gives a marginally improvement from the one fully-connected, and still less performs than the one without L_2 norm pooling. Not only about the poor performance but also considering the L_2 norm pooling's negative effect on our RNN-CNN model (remind that the pooling implementation raised the representational resolution of CNN itself in table 10), we can conclude that there must be some good disciplines to train RNN, not just provide good representation embeddings. Without it, the model would continue being suffered by over-fitting matters[24].

CNN pretraining via Auto-encoding So far we have discussed how to push nicer input into GRU, and it is proven as not effective. How can we ensure the embeddings to properly represent invariants along the temporal axis? Training it as an auto-encoder may be one way to do that. This is because auto-encoder is updated for representing the input with reduced dimension it gets as output by best, so that the network should effectively contain the representation of the input, and it abides to both temporal(eventhough CNN does not have any sequential inference capability) and feature space representation for recurrent networks. Hence we hypothesise that the auto-encoding pretraining on CNN for our classification model gives better temporal representation of the input for GRU so that the model better utilises its capability of sequential inference.

Based on this idea, we conduct auto-encoding pretaining on CNN + Two FCLs as decodr, and port the CNN to double-layered unidirectional GRU. For the suggested model with AE pre-train, we still do not exceed the performance of the double GRU layer baseline model investigated in 2.c. Over various combination of CNN dropout probabilities(15), number GRU layers(table 16), and attention mechanism[25], our proposed model performs 0.61 by its best validation set accuracy in table 15 - which is 0.03 lower than the baseline without CNN embeddings. Eventhough we reaffirmed the effect of the above engineerings into our deep model, it is not consistent to say that the AE pre-training enforces useful the cell memories of GRU effective sequential representation to do classification task better. Briefed details follow up, and we do not extend the AE pre-training on CNN because of the computation demand of AE is so high, and performance does not meaningfully improve.

Stride of pooling matters Eventually, we question about the length of temporal space that CNN layer yields. If the stride of pooling is bigger than 1, the time length becomes reduces by

[24]Although not tabularised in this report, further proceedings on various Dropout controls on CNN did not resolve this issue as well

[25]Experiments with various length of attention vector are omitted in this report as they did not show any change, at least margin, in learning performance of our model.

[25]P_{DO} is taken as 0.1, 0.3 for input layer and CNN layers, respectively.

$P_{DO}(x) \setminus P_{DO}(CNN(x))$	0.1	0.3	0.4
0.1	0.54(0.95)	0.57(0.86)	0.56(0.77)
0.2	0.53(0.93)	0.60(0.84)	0.61(0.76)
0.3	0.52(0.87)	0.58(0.83)	0.58(0.78)

Table 15: Learning performance for MSD10 validation(Training) set for an AE pre-trained double-layer CNN with 2GRU-2THw model with various dropout probabilities on CNN.

n_{GRU}	2	3	4
0.53(0.79)	0.57(0.86)	0.57(0.88)	0.60(0.84)

Table 16: Same experiment to table 15 but with various number of GRU layers.

that stride and the performance of learning may depend on it as the number of cells that gRNN possesses is reduces.

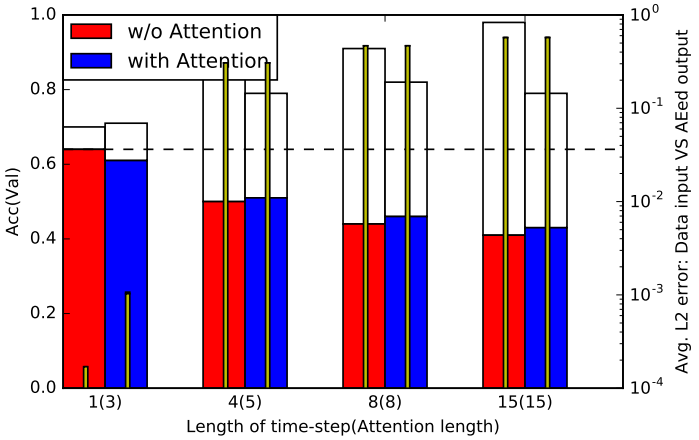


Figure 15: Classification accuracy of Autoencoding-pretrained[26]GRU double layer with or without attention units. White bars are the accuracy difference between validation and training data set. Yellow bars are the Euclidean distance error measured during the Auto-encoding step. Black line is the GRU baseline performance introduced in section 2.c.

We test our GRU baseline on various length of time from 120(default) to 8 as shown in figure 15. There it is found that shirinking the length of temporal space is critically diminishes the classification performance as well as the AE capability of our model. Also, the GRU model loses its sequential representation power as the time-step gets bigger. This means that the length of temporal axis GRU reads is critically important to train GRU in terms of the sequential representation resolution and classification performance.

Before it is found, we expected GRU learns good representation from 30 dimension of temporal space represented by two stacks of CNN with pooling stride 2 so that 64% accurate but just 70% trainable GRU gets much back up on its performance by nearly over 90% accurately studied - in terms of $Acc_{Tr.}$ but not as good as GRU to classify with validation set - CNN embeddings. It is actually not true as GRU in 30 temporal dimension could only learn about 50% with 83% training set accuracy. **Hence what has been really happening in section 5.a and 5.a.2 is that CNN embedding improves the GRU only model - with 4 or 8 time units per its time-step - by 7 $\sim$ 11% in terms of validation set accuracy.** This

shows that CNN embedding for GRU works for MSD10 classification, although AE pre-training only weakens the effect of CNN embedding so far.

Therefore, three options left to resolve this problem of reduced time length: taking 1 stack of CNN as embeddings to reduce the loss of RNN performance to reduce the time length loss(but single layered CNN only performs around 0.45(0.57) by its best in table 6), reducing the stride of pooling to 1(Max-pooling totally loses its filtering capability with stride 1 in table 9), or get rid of the pooling layer of CNN.

Pooling filter	Max	L_2
$Acc._{Val}(Acc._{Tr})$	0.10(0.10)	0.10(0.10)
Max+L_2	L_2+Average	$L_2 + L_1$
0.10(0.10)	0.10(0.10)	0.10(0.10)
Diff.	L_2+Diff.	$L_2 + L_1$+Diff.
0.10(0.10)	0.10(0.10)	0.10(0.10)

Table 17: Learning performance of 2GRU-2THw-2CNN model[27]with $stride_{time} = 1$ for pooling layer on MSD10 validation(Training) set. The above result comes over $Dim(Pool)_{time}$=2 $\sim$ 5, $Dim(Kernel)_{time}$=2 $\sim$ 5. Other hyperparameters are set in typical manner as previous experiments in this report. Strict 10% accuracy - random game - results come out.

Table 17 shows that striding only one time-step for pooling is not a good idea. Every attempt on this with various hyperparameters of CNN results failure - pure random guess as 10%. During the series of experiences with stride 1 pooling, Max-pooling, $L_{1,2}$ and time-differential[28] pooling methods are applied as various combinations. This result stems from the nature of pooling, a sensible convoluted space compression method within the pool. If pools are mostly shared, the pooled results mostly overlapped where the model loses its pattern resolution. Given by this empirical support, we now desert the pooling layer from out CNN design.

5.a.3 No-pooling on CNN embeddings

Finally, our investigation on effective CNN embeddings boils down to not applying the pooling layer so that we secure 120 dimension on the temporal axis. Pooling method has been renowned as an effective compression of representations which not only reduces the size of parameter space, but also stimulates the model to sensibly perceive the invariant patterns within the data [25]. However, we arrive to the point that the compression along the temporal axis harms the performance hence need to investigate on the side of 'what if not?'. While we go without pooling on the temporal axis from now, we apply pooling to the feature space when we have kernels of partial length to the input's feature space in table 20.

Unlike the previous experiments, no-pooling strategy enables our suggested model(CNN embedding on GRU) to exceed its performance over the double-layer unidirectional GRU baseline investigated in section 2.c. Following paragraphs revisit the issues discussed in the previous sections to confirm their effectiveness and summary of the models is given in table 18[29].

[28]The reporter invented this pooling method. It is a 1 dimensional pooling of pooling size for temporal axis is 2^n where $Pool([x_t, x_{t+1}]) = \frac{x_t - x_{t+1}}{2} \equiv v_{t+1}$, $Pool([x_t, ..., x_{t+3}]) \equiv v_{t+3} - v_{t+1} \equiv a_{t+3}$ and so on. It can be simply implemented by iterating *tf.multiply*-ing a pool with [-1,1] tiled tensor and *tf.contrib.avgpool* it with [2,1] size pool.

[29]For reference, *GRU-tempHighway-ConvnoPool.ipynb* is attached to the submission as an implementation of the GRU with CNN embeddings and temporal bridge between them.

[30]Double-layered CNN with $Dim(Kernel)_{time} = 2$, $n^{(out)}_{channel} = 128, 64$ for first and second CNN, $pool_{time} = 2$

[31](0315-0,1.log) $Dim(Kernel)_{time} = 2, 3, 5$. $n^{out}_{channel} = 64, 32$ for each kernel. $P_{Dropout} = [0.1, 0.3 * 2]$ where 0.1 is for input and 0.3 is for convolutional layer. 20% dropout is applied to GRU. Same performance is achieved with bidirectional GRU with 30% dropout

Model \ Perform	$Acc_{Val}(Acc_{Tr})$	Comment
$GRU2^{uni}$-noCNN	0.64(0.70)	The baseline found in section 2.c
$GRU2^{uni} - Hw2 - CNN2^{30}$	0.62(0.69)	$2CNN_{stride2}$ gives $len(time)_{embed} = 30$
$GRU3^{uni} - Bridge0 - CNN2_{noPooling}{}^{31}$	0.66(0.74)	len(time) = 120 via no-pooling makes the first progress over the baseline

Table 18: Summary of learning performance of suggested model with regards to CNN embeddings for MSD10 dataset. Each model runs for 50 epochs.

Dropout for GRU Optimal model is found when reasonable amount of dropout is applied to CNN as well as GRU. It is against our previous findings in section 2.b.3 figure 7a, where GRU stacks are even harassed by dropout wrapping. Providing that LSTM is not particularly sensitive to dropout compare to GRU, the result shown in figure 16 provides this. Without dropout on GRU, the GRU with CNN embedding performs worse than the one without CNN embedding, which means that the dropout contributes on resolving the sequential representation which the GRU bases for inference. In addition, the dropout does not effectively reduces the training set accuracy while improves the validation set accuracy. Providing that the training set accuracy is no more at around 90% or over, dropout does not force the model to lose its capability of understanding the data. Lastly, no significant change in performance is made by putting bidirectional GRU rather than the unidirectional one, which is in line with figure 8.

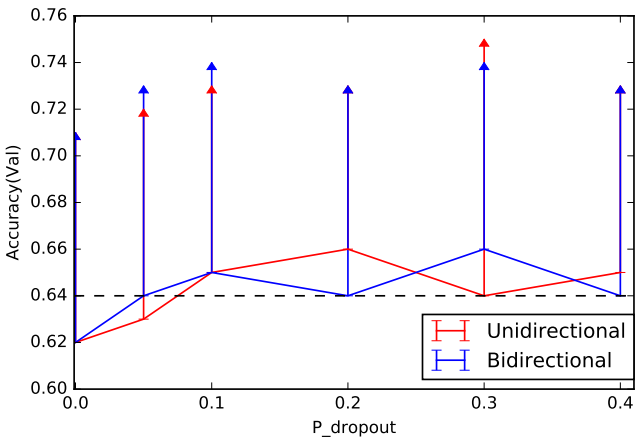


Figure 16: Classification accuracy of double-layer GRU with double-layer CNN embedding at various dropout probability for GRU[32]. Black dashed line is the one for double-layer GRU without CNN embedding nor dropouts.

Hence this experiment empricially supports that the sensible dropout regularisation improves the performance of deep neural network with recurrence units[8]. As the model becomes more complex for being more capable of representing the data, mini-batch wise dropout helps the parameter space equip with more diverse patterns useful for classification by training the model with (1-P_{DO}) of the whole parameters arbitrarily masked, but 100% in use for validation and test set.

[32]Kernels with full feature dimension and 2,3,5 for the time dimension($Dim(Kernel)_{time}$) with 64, 32 output channels for the first and second convolutional layer for each kernel.

More kernels Table 19 shows that kernels with various time-step helps the model to improve, although such addition also is limited to extend in terms of performance[33]. The improvement of performance can be understood by considering that the longer kernel actually enables some hidden variables to contain representations on longer temporal horizon. For instance, double-layer convolutional layer with one of kernel size is 5 on temporal axis, some the final outputs channel of the kernel have looked up 9 unit time-steps by being a left corner of one layer and a right corner for another. As some of representations provide long horizon, GRU using this embedding can better cope with long-term dependencies useful for classification.

$Dim(kernel)_{time}$	(,2)	+(,3)	+(,5)	+(,10)
$Acc_{Tr}(Acc_{Val})$	0.62(0.71)	0.63(0.73)	**0.66(0.74)**	0.64(0.83)

Table 19: Classification performance of 3GRU-0Bridge-2CNN_noPooling[34]with various bag of kernels(accumulated) of different temporal axis size(full dimension on feature axis).

We revisit the idea of using kernels with fraction of feature dimension discussed in table 12. By setting up the size of kernel in feature dimension as proportion of the full dimension(ex:1.0 for full dimension) and also using it to determine the pool size and the equal number to stride of the pool, we realise the feasible multi-layer CNN. For example, for 0.5 kernel on the feature dimension takes 12 feature dimension for the input space(of which $Dim(x)_{feature} = 25$) and the pool becomes 7 with stride 7 for the convoluted output channels of feature dimension 14, hence feature space dimension reduces to 2, which is inverse of 0.5 .

$Dim(Kern)_{ft} \backslash Dim(Kern)_t$	(,2)	+(,3)	+(,5)
(1.full,)	0.62(0.71)	0.63(0.73)	0.66(0.74)
+(0.5full,)	0.63(0.67)	n/a[35]	n/a
+(0.25full,)	0.65(0.70)	n/a	n/a

Table 20: Classification performance of 3GRU-0Bridge-2CNN_noPooling with various bag of kernels(accumulated) of various size (feature,temporal) axis. Each model runs for 50 epochs[36]. Other specifications are the same of table 19.

The first data column of table 20 shows that the kernels fractional dimension in feature dimension improves the representational capability of CNN embedding resulting the better result in GRU. It is not only interesting by improving the performance, but also shows the difference in natural of MSD data and word embeddings in NLP, in that the word vectors embeddings in as input representation in natural language modeling should necessarily be treated as with full feature dimensional convolution as each word vector is only meaningful when relative similarity among vectors are concerned, hence partial convolution makes the word-embeddings valueless[24]. However, features in MSD data have their local meaning to represent the semantics of the song like images, hence the kernels with fractional feature dimension helps - eventhough we substantially max-pools out.

Lastly, the reporter attempted to apply additional pooling methods($L_{1,2}$) onto the feature dimension to see how the various compression methods result in our model's performance.

[33]Lower performance on the model with kernels $(, 2|3|5|10)$ might be caused by the over-fitting matter as the hidden variable space becomes larger than ever, although $(, 2|3|5)$ works the best in that line.

[34]$P_{DO}^{GRU} = 0.2,\ P_{DO}^{CNN} = [0.3 * 2], P_{DO}^{x} = 0.1,\ n_{ch^{(out)}} = [64, 32]$ per kernel

[34]Memory overflow problem happens on the DICE and force terminated on MscCluster.

[34]For models having kernels of fractional feature dimension are not over-trained so that their training set accuracies are not blown up, but we measure the validation set accuracy safely converged.

However, by the limitation of the computational power of both DICE and MscCluster, the runnings were not made even with considerably reduced output channel numbers to [16, 8] from [64,32] for each first and second convolutional layer. We left the detailed investigations on fractional feature space kernels and pooling methods for the further research as they are not viable at this moment.

5.b Curriculum for training cell states over sequence

This subsection consists of three parts: upto which part of the model should be the encoding part of the AE pre-trainer, how the dimension of the feature space at the output of the encoder, and when does the attention mechanism works.

5.b.1 Effective sequential autoencoder by bottleneck layer

It is known that shrinking the dimension of the upper layers makes the AE effectively represent the representation. For example, double-FCL works as a principal component analysis machine where the rank of output is determined by the dimension of the upper layer. However, this does not mean that reducing the upper layer dimension is the sufficient condition to make a good AE pre-trainer because efficiency does not guarantee the overall learning performance. To investigate on this issue, we set a model AE which is double unidirectional GRU with double CNN embedding as the encoder and sFCL on top of the encoder as the decoder. The AE models with various dimension of the cell states of GRU (3 to 32) are pre-trained (so that each state needs to represent 25 dimensional vector), and then stacked up by another double-layer of GRU for the classification. Hereby we would observe how the narrowed hierarchial path in AE affects on the performance of AE (in terms of Euclidean distance between input and output of time-slide $\Delta = 1$ in equation 3) and stacked classifier.

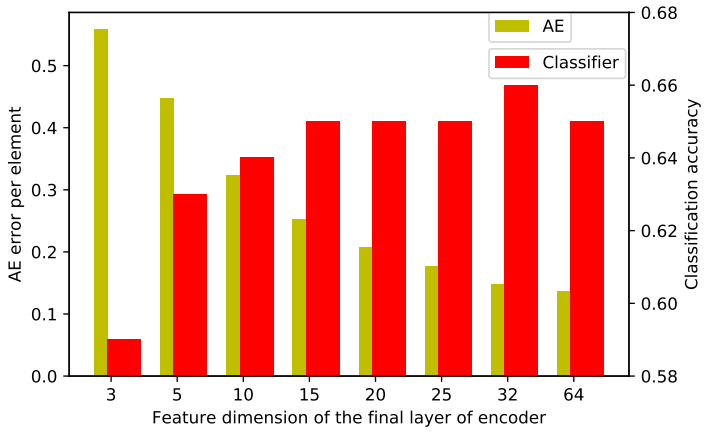


Figure 17: Autoencoding error(thin yellow bars) of double-layered unidirectional GRU with CNN embedding on various dimension of the final encoding layer($n_{hidden}^{(finalGRU)}$) per time-step. Thick red bars are classification accuracy of the double-unidirectional GRU stacked-up on the encoders. Final dimension of GRU-CNN encoder per time-step decoded by double-layer sFCL is 25, hence bottle-neck effect at the final encoding part plays a role when $n_{hidden} < 25$.

Figure 17 shows sequential AE pre-training for the classifier does not improve the learning performance. The classification performance(66% validation set accuracy) is sames as the one without sequential AE pre-training, by best. Also, reduced dimension of the final hidden

encoding layer hampers the performances of both AE and classification task. This rebuts the consistency of the above-mentioned conjecture that the pre-training for making reduced dimensional representations is helpful to classify the song data.

Before we conclude that this pre-training is not useful, one possible justification is left that the sequential AE pre-training may require global scope over the temporal space as AE is utilising the whole output space, so that global overlook might be crucial. Based on this, we put the attention wrapper(of attention dimension 5 which was found as optimum for GRU baseline in table 3) to the GRU at the encoding part and conduct the same experiment. We anticipate that the ability of the attention mechanism to look up the global embedding space improves the auto-encoding performance which will then transfer to the classification force.

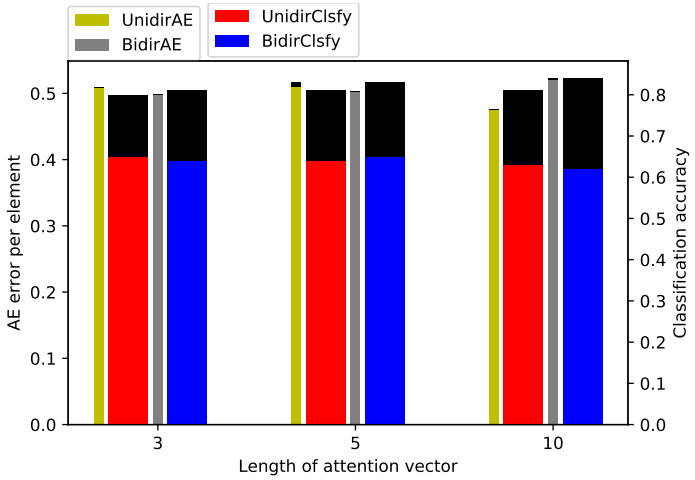


Figure 18: **Classification not updated: update via Displayer.ipynb with job** AE-pretraining error and the classification accuracy of the Attention mechanism wrapped models specified in figure 17. n_{hidden} is fixed as 25 as the model gives sufficiently good AE and classification performance with this setup. Black bar is the performance difference between the training set and the validation set.

However, the result shown in figure 18 is not much interesting as the attention mechanism does not make improvement on performance(the model shown above gives 64,65% accuracies) and it rather enlarges the AE error(from middle of 0.2 without the attention mechanism, it now becomes about 0.5).

The attention mechanism, which is originally called as 'joint alignment model'[14], is the method used for neural machine translator, a gated recurrent network auto-encoder where every output the input slice emits is considered equally as target of the objective function. Hence the mechanism is made for generative model. In here, we found that the effectiveness of the attention mechanism and sequential AE pretraining proven in the papers are not applicable. This means that the representations trained from these methods are not transferable to the classification task. Hence our empirical result does not support the hypothesis 2 as that the better representations for mirroring the input is not the good way to train the cell-states for classification task.

5.b.2 Cross-entropy measure from all time-steps

In this subsection 5.b, we have applied sequential autoencoder as a pretraining and attention mechanism to the encoding part to make the model understand the invariant of the data better. This is because we believe that better understanding of the invariants would lead to the better

classification result[25]. However, there is a catch. Is that really the case? For example, sequential auto-encoding is like we train ourselves to memories all the pages of Deep Learning textbook by predicting the context of next pages. Will it really improve ourselves to classify what that book is? If our brain is sufficiently capable of learning the representations, that sequential enforcing of predicting the next pages may not worth(input of the next time-step as output). What if we just constantly notice us that the book is about deep-learning(classification vector as output) over the all time-steps?

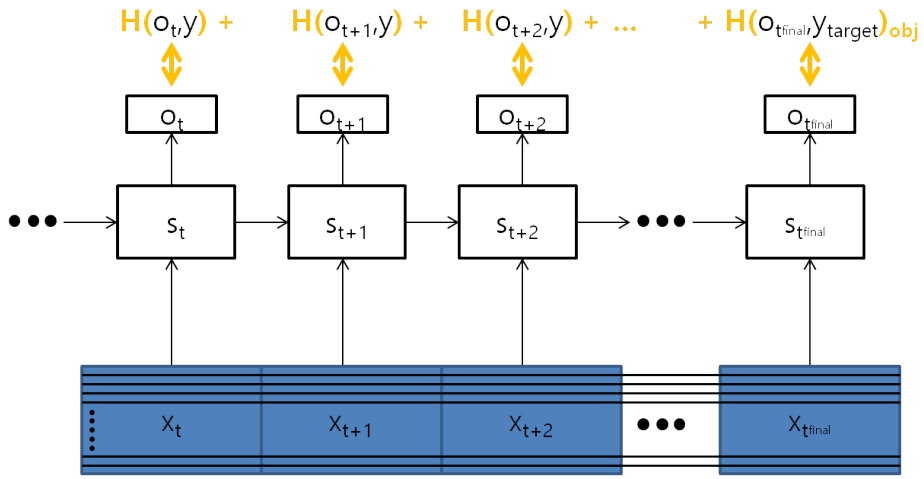


Figure 19: Training RNN by optimising all the outputs. Compared to the figure 1, the objective function not only refer to the output in the final time-step which we also use for classification.

Recalling section 2.d, we simply change the objective function our double-layer GRU baseline as the cross-entropy of all the outputs over the temporal space, not only the last one. The result shown in figure 20 gives to surprises. One is that the new objective function makes the progress on the performance on par to the best of the complex models we found so far in this report - most of complex strategies such as AE, pooling, attention mechanism did not even reach close -, and the other is that the performance falls from the top even when we apply sensible regularisation method.

The first point can be understood as that sequential enforce of the desired output can lead the output of the final time-step more desirable(see 2% increment of performance from the one without this enforcement). It means the cell and memory states of gated RNN $\{\underline{c}_i^{n_{ly}}\}, \{\underline{m}_i^{n_{ly}}\}$ $\forall i \in [t_0, t_{final}]$ are more opted to output the classication vector as output, and they help the final state to yield more correct guess on the output. This deduction on the empirical result in figure 20 supports the hypothesis 2, as that we found a sensible way to teach the states on the classification task other than just seeing the last output for the objective function. Reminding that the optimal n_{hidden} was found at around 30 not 64 for the GRU baseline in table 2, but it now becomes 64. This is the another proof that optimising all the outputs enables the cell states to effectively contribute to the classification task, eventhough we refer to the last time-step to measure the accuracy.

The second point, declining performance, may not be solely due to the over-fitting problem as the validation set accuracy usually does not fall down as much as 4%, but goes converged. It would be rather more plausible that the updates run against the improvement of the validation accuracy. Even with strong L_2 penalty with the penalty coefficient as 0.01, the decrease still exists[35]. Although it may be stemmed from weak regularisation or too large parameter space, the

[35]Same thing happened even when the coefficient becomes 10.0

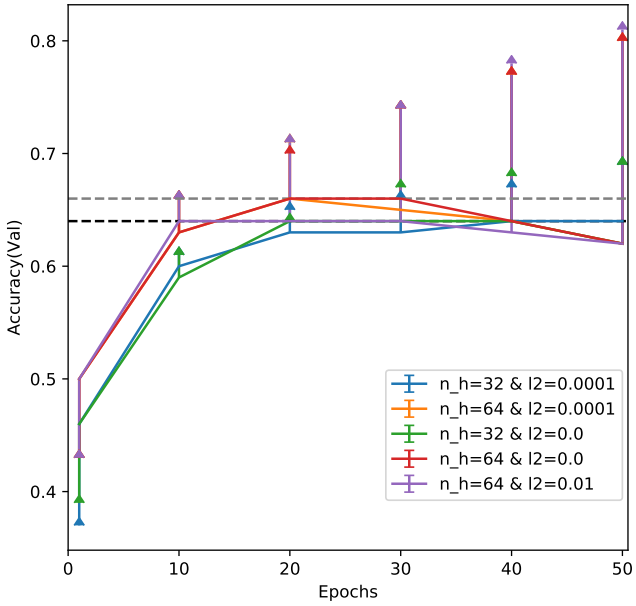


Figure 20: Learning performance of simple double-layer GRU with one-layer sFCL on top with output vectors from all time-step for the objective function. Black line is the performance of the double-layer GRU baseline with only last time-step output for objective function, and the gray line is the performance of the best model(2GRU-0Bridge-2CNN) found in section 5.a.3 . 20% dropout is applied to GRUs and [0.1,0.3,0.3] for input and CNN as usual. L_2 penalty to the top sFCL is given for some cases.

reporter conjectures that diminishing contribution of the last output to the objective function over epochs is the reason. According to figure 9, optimising the outputs far from the final time-step is hard and gives lesser improvement. This means that the proportion of their cross-entropy rises when the training progresses, as the outputs near the final decrease faster. Then the updates become more irrelevant to the final output for classification as the update gradient less care about the final time-step. Figure 21 shows that this scenario is actually the case. Hence we investigate on both contribution of the last output to the error measure, and the over-fitting control.

Based on the above conjecture, we need to guarantee the certain magnitude contribution of the last time-step's error as we end up classifying the data with the last output or otherwise the performance becomes poor(fig 2.d). To resolve the diminishing phenomenon, the reporter now clips the sum of cross-entropy errors[36] for every time-steps for each mini-batch to the norm of the last time-step one and proceed the learning. The bounded norm of error may ensure the magnitude of parameter back-propagation from each time-step becomes evenly made hence the contribution of the last time-step's norm becomes well delivery during the whole training.

Several clipping designs are applied and the following figure 22 are the results of the clipped training for the double-layered unidirectional GRU baseline. As it is shown, any controls attempted in below did not improve the performance against our wish. The expectation might

[36]It is extremely important to clip the sum of errors, not the norm of the error vectors. First is that the norm of cross-entropy does not give anything in terms of information theory, and secondly the diminishing effect still holds with the norm-clipping.

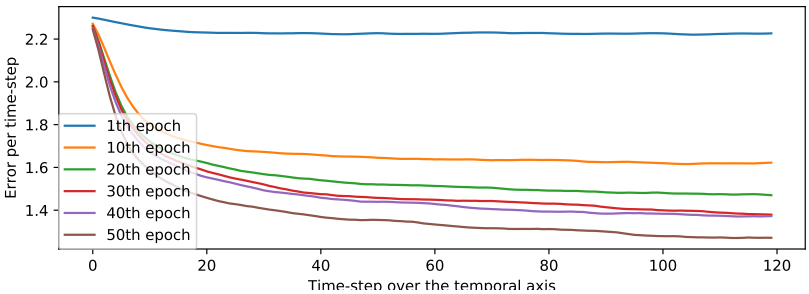
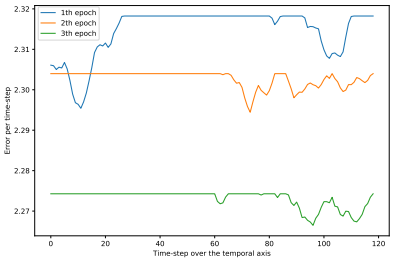
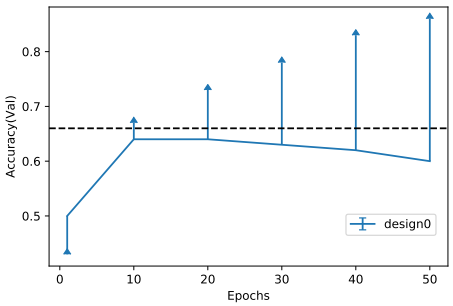


Figure 21: Error measure of the training set over the time-step during the training for the double-layer uni-directional GRU model. As anticipated in the report, the last one diminishes faster.

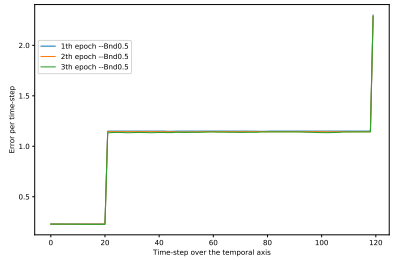
be wrong at the beginning as the gradient descent over the whole time-period are the optimal path to lower the energy measure, which somehow directs the direction to update. Magnitude of errors determine the portion of contribution that each time-step's output give to. Distorting them must deters the update gradient from the optimal path. Hence Diminishing contribution is just one of the step we counter during the descending path so that changing the path interrupt the optimisation path.



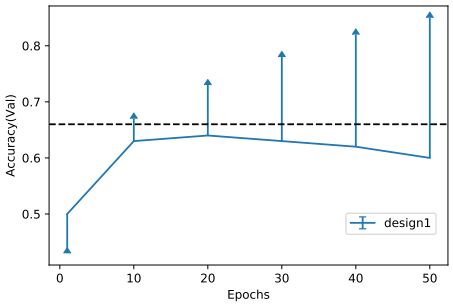
(a) Design 0: straight line - even contribution by best.



(b) Classification result of design 0

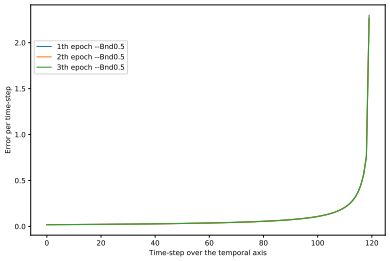


(c) Design 1: step function1. 10% of the last error for the first 20 time-steps, (*boundrate*)% for the rest

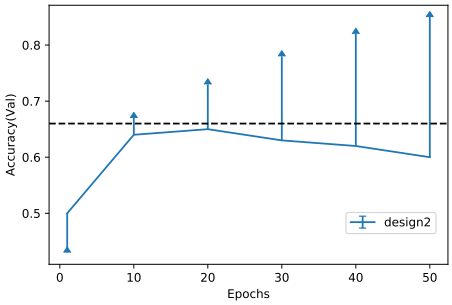


(d) Result of 1

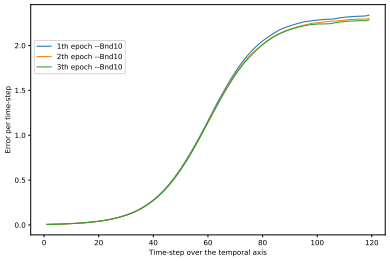
Neither dropout wrapping is found(table 21) to be useful for controlling the over-fitting problem for all time-step look up energy measure, as like L_2 did not perform. Models with more than 20% dropout on GRU layers give 64% by the maximum performance, which is not desirable.



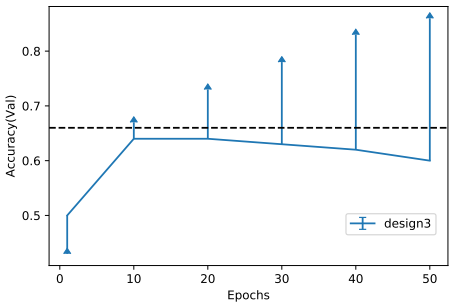
(e) Design 2: decay from the last to the first. $error_{clipped}(t)=\frac{boundrate-119}{boundrate-t}$



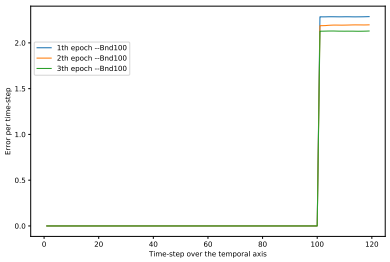
(f) Result of 2



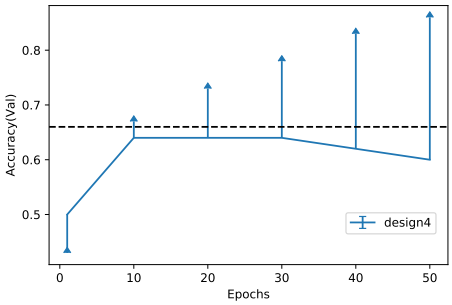
(g) Design 3: sigmoid. $error_{clp}(t) = \frac{1}{1+exp(-(t-t_{origin})/boundrate)}$



(h) Result of 3



(i) Design 4: step function2. 100% to the last output norm from t = boundrate and dropout for the earlier steps.



(j) Result of 4

Figure 22: Design of error clipping over the temporal space and the result on the double-layer GRU baseline. Boundrate represents various parameters of each design function. Black horizontal line is the same GRU model with all time-steps error measure but without error contribution control found in figure 20. Designs on the error clipping make the models perform lesser than the one without design in figure.

5.b.3 Bidirectional recurrence for all time-step look ups

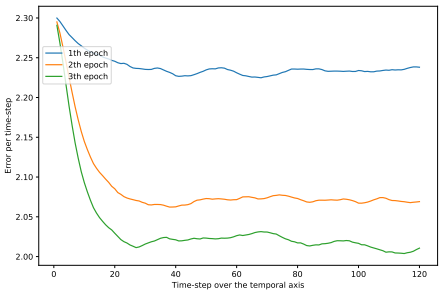
So far we have investigated methods that directly control the contribution of cross-entropy measure for each time-step, to make the sequential states more favourable to produce better classification vector at the last output. We expected that injecting AE or classification vector with corresponding error measure can evoke the higher capability of the cells for sequential inference by transferring their learned representations to the classification time-step. However, none of these direct curriculum has not proven their use for improvement, in that we still do not control the weak point of gRNN itself, the contribution diminishing phenomenon found in figure

$Clip_{err}$ Design	0	1	2	3	4
$Acc_{Tr}(Acc_{Val})$	0.65(0.74)	0.65(0.74)	0.64(0.75)	0.65(0.74)	0.65(0.74)
$P^{Opt.}_{dropout}$	0.2	0.2	0.2	0.2	0.2

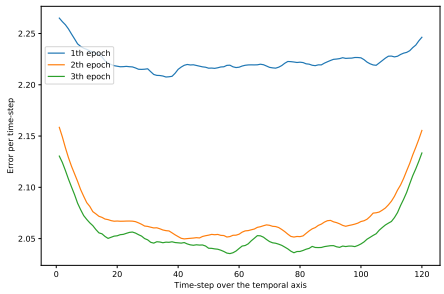
Table 21: Optimal learning performance of $2GRU^{Uni} - 2CNN$ model with design 2(fig 22c) error clippings investigated over a range of dropout probabilities:[0.0, 0.1, 0.2, 0.4, 0.5]. Under the manipulated error contribution scheme, the regularisation does not work in our case. Validation accuracy declination is even found in 50% dropout.

21. Manipulation of error measure via error clipping, or alignment of representations that each time-step favours via attention mechanism did not facilitate the issue even with their global scope for reference. What if we just put a sequential DNN compartment that it can effectively learn the sequence back and forth so that all the time-steps may get guaranteed to contribute on update at certain level?

Recalling the unidirectional GRU gives. If the contribution of last output diminishes on the error measure faster, making such ends for both sides of the temporal space can be a solution. This recurrence structure for both ways is called bi-directional RNN, which was introduced in section 2.c. If the both sides become the optimal classification point with lowest error, then diminishing contribution problem will be resolved. By that we can push the variables we tried to control into the structure of hidden variables in DNN.

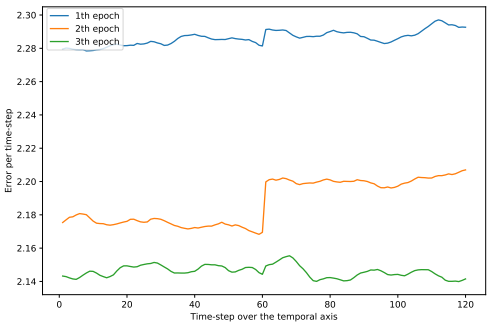


(a) Bidiretional double-layered GRU's performance over epochs. 'rev' means the backward recurrent states and reversed.



(b) All tuples of bidirectional outputs are referred for the error measure. Having two ends by two recurrence means both ends become hardly trainable.

We thereby introduce three bi-directional output reference strategy to cope with the all time-steps contribution on the error measure, in figure 23. The first two are visited in figure 2.d. Both possess the problem of having stiff error at their temporal sides. for inference so that the last output vector is made by the first cell in the backward state and the last cell in the forward state. Hence the model inherits the diminishing contribution problem as like with unidirectional GRU in figure 21. As we have seen so far, reduction of error measure increases when the cells locate farther from the initial time-step. Hence we suggest a model(in fig. 23c) which looks up the first half of the output state from the backward recurrence part, and the last half of time-steps come from the forward recurrence part to combining best of each only. Figure 24 shows that applying bidirectional layer with the figure 23c type of output lookup gives a marginal progress in term of the validation set accuracy, to 67%. One of the most interesting thing is that the fast over-fitting problem is disappeared when the optimal $P_{dropout} = 30\%$ is applied. As the bidirectional recurrence with 'long-term trained' outputs over the whole time-step gets ride of the stiff layer of the error measure at the corner (compare fig 23c to the figure 23a) the



(c) Model of which its output state is a combination of the first half of the backward recurrence and the second half of the forward recurrence. Stiff hill at the edges of temporal axis are disappeared.

Figure 23: GRU output reference methods. While (a) and (b) are conventional, the new concatenation method of backward and forward recurrence resolves the problem of stiff edge at the corner of the time-space.

improvement is sufficiently guaranteed. This experiment supports that we can stably manage the sequential enforcement on the states of gRNNs, at least indirectly by putting bidirectional gRNN and update with sequential curriculum on classification error for well trained outputs(fig 23c) over the whole temporal step.

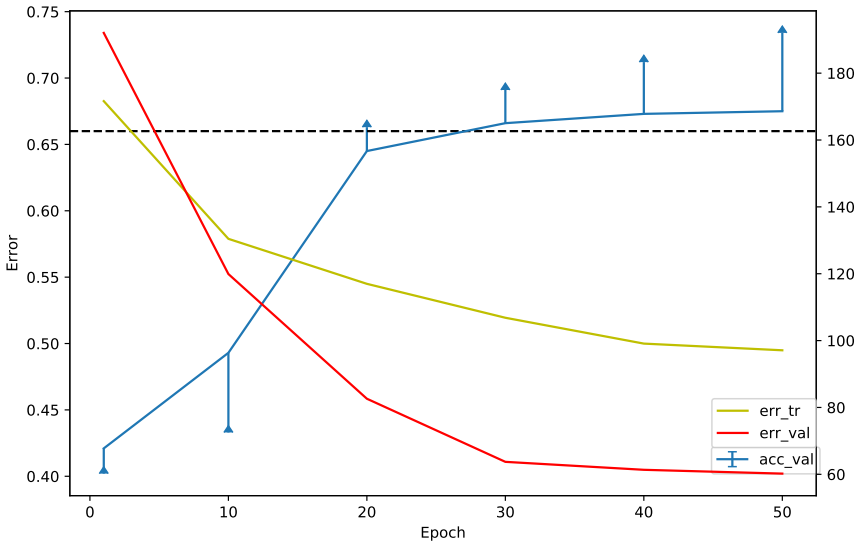


Figure 24: Training performance of triple-layered bi-directional GRU with type3(fig 23c) outputtings for the all time-step error measure. Black line is the latest maximum validation set accuracy reached by GRU-CNN embeddings model and unidirectional double-layered GRU for all time-steps look ups(table 22).

Finally, we inject the new error measurement into our suggested model(fig 14 and apply our energy measure the performance, which is summarised in table 22 below. On combining the

CNN embedding and GRU with all time-step looks up, additional 1% accuracy is achieved from the performance of each.

Model specification	$Acc_{Val}(Acc_{Tr})$	Comment
$GRU2^{uni}_{n_h=32}$	0.64(0.70)	The baseline found in section 2.c
$GRU3^{uni}_{n_h=32} - CNN2_{noPooling}$	0.66(0.74)	Opt. CNN embeddings in section 5.a.3
$GRU2^{uni}_{n_h=64}$-all time-steps	0.66(0.72)	Optimise with outputs in all time-steps. Performance declination phenomenon is observed
$GRU2^{bi(type3)}_{n_h=64}$-all time-steps	0.67(0.82)	Bidirectional layer with sensible mix of forward/backward output sequence(fig 23c) shows the improvement on the model by sequential enforcement
$GRU3^{bi(type3)} - CNN2$ -all time-steps	0.68(0.84)	Optimal combination of CNN embeddings(sec 5.a) and sequential enforcement(sec 5.b) by coarse search on hyperparameters.[38]

Table 22: Summary of learning performance of suggested model with regards to cell stated enforcement for the classification task. Each model is trained for 50 epochs.

5.b.4 Summary

In this subsection we directly and indirectly enforce the states$\{(c, m)\}_t$ overall temporal space. It was assumed that the learning over the past would be transferred to make the inference on the classification label. Sequential autoencoder, which is deemed as an effective representation learning pre-trainer, was found to be not effective non MSD10 classification task. Global look up on the temporal space via attention mechanism did not help as well.

Subsequently, we alter the injection material from the future input to the classification label itself because the cells need to learn how to classify in the end. By taking the classification label from the outputs in all the temporal space, we found an improvement of the performance over the GRU baseline suggested in section 2.c. However, uncontrollable over-fitting which resulting unstable declination of the model's performance arises. Manipulation of the error profiles over the temporal space does not work. After finding the root of the declination problem as the speed gap of learning between the first and the last output state in the uni-directional GRU - hence the last output state diminishes its contribution to the backpropagation gradient, bi-directional GRU is revisited with suggestion on the sensible outputting method. The suggested bi-directional GRU works on both performance improvement and diminishing contribution problem of the classifier output at the last time-step.

Lastly, transfer of improved representation for the MSD10 made by CNN embeddings to the GRU structure is again observed with bidirectionality and the all time-step classification strategy, by little margin.

[38]For CNN, double-layer with [64,32] for the first and second layer's output channel, [2,3,5]*[1.,0.5] temporal, feature dimension kernels are fixed. CNN is linked directly to triple-layered GRU of n_hidden=64. The optimal dropout for CNN is found to be [0.1,0.3*2] from the input to the top convolutional layer, while GRU's dropout is changed to 30%.

6 Conclusion

Motivated from sequential inference modeling in machine learning, we investigated Recurrent Neural Network, a neural network model that realises chain model of hidden variables by time. Gate mechanism resolves the issue of vanilla RNN on lack of long-term state reference with diminishing gradients in backpropagation through time. Long-Short Term Memory and Gated Recurrent Unit cell structure are investigated as the examples of the gated RNN in aspects of layer depth and size, over-fitting control, gradient-clipping, and over-all performance stability. There we empirically confirmed that the sequential inference model with temporal states of cell and memory tutples boosts the classification capability of the neural network model over the simple fully-connected layered feed-forward network.

Going in deeper, two issues of applying gRNN the classification task for the sequential data are raised: (1)gRNN must better to infer for smallest time-step(fig 15) so that each cell state looks over very narrow temporal space and (2)only the final output state is referred for the genre prediction work so that the past outputs are ignored for their importance in direct use.

To challenge on the first problem(hyp 3), CNN embeddings with sensible specification on parameter space size and regularisation magnitude can be a better source for GRU to interpret the Million Song Dataset 10, resulting in marginally better(table 18: $\sim 2\%$ and may be more if fractional feature dimensional kernels could be computed) classification performance than the GRU baseline without the CNN embedding. Hence the hypothesis 3 is empirically supported by this report. First and foremost, by excluding the pooling layer at CNN's temporal axis, we secure the full temporal dimension of the input to the embeddings which results in best performance on MSD10 data. This is mainly due to the dependency of performance of the GRU to the dimension of the temporal space(fig 15). Sensible combination of kernels in both temporal and feature space improves the quality of CNN embedding, hence the performance of the model. Lastly, bridging part between CNN and RNN has found to be no use for the model.

For the second problem(hyp 2), we implemented sequential auto-encoding pretraining and all outputs look up for error measure. Sequential AE is not found to be helpful for classifying the data hence the representations of mirroring do not transfer to the classification representations for MSD10. Attention mechanism as a facilitator to look over the global temporal space is neither proven to be useful for that transfer learning. Rather than enforcing the transfer of useful representations, we inject the cross-entropy measure of all outputs over the temporal space to the objective function and observed the rise of performance of the gRNN baseline. There we empirically confirm the validity of sequential state(hyp 2) with enforcement with direct curriculum as the proof. This error injection enforcement reinforces our findings in section 2.d that the classification task refers to the final output just because it is better than other states, but is not guaranteed to be the best. Progressing further, we dealt on the diminishing contribution issue(fig 21) of the last time-step output - which is better to be the classifier output by 2.d). Clipping on error distribution or dropout themselves do not deal with the fast overfitting&declining performance issue. Bidirectional gRNN structure erases the stiff time-step on the classification error curve as we can fill up the whole output space with long-term trained(at least half of the temporal dimension).

On the course of investigations for this report, one question arises - what does 'Deep' neural network give us on machine learning task? Deep network gives much freedom of constructing theories into practices, and by which we can leave the deep understanding of the features to the model from us. Iterations of hypothesis make ups and real measurements this report attempted also gives marginal improvements on making the model deep and complex. However, the deep learning also requires vast and sophisticated investigations on the model specification. In many occasions, the model easily becomes computationally overwhelming but less performs unless very

neat specification is captured. One cannot even confirm the validity of the hypothesis in many cases unless the time-consuming process finishes out. The performance and empirical results of the deep model hugely depends on the learning task and the nature of data. Eventhough one can bare much higher computational cost and even more demanding specification optimisation process, the transfer of theories to improvement of performance are not usually transferred and in most of other cases marginal - at least in this report. Giving this experience, the reporter finally reaffirms the importance of probabilistic aspect of deep neural network as the beautiful artifacts of local representation learners(RNN, CNN) gives the most huge quantum jump he could observe during the investigation - to really know what he thinks of making sense actually makes sense by the machine.

7 Further direction

The most missing thing during the course of the investigation was that fractional feature dimensional kernels performed but could not be investigated due to the computing power. Combination of kernels with multiple temporal step and feature dimension would be powerful and would give us to explore the nature of acoustic data. Also, we could not find the usefulness of attention mechanism on our classification task from loads of experiments conducted for this paper, which is also a miss. Sensible way of applying them in terms of computational cost(both methods are unrealistically costly), and better learning performance.

In the section 5.b.1, we discussed about sequential auto-encoding boosted classifier via recurrent neural network. We expected to see a good proof of transfer learning(from AE task to context analysis task, which is classification) working on the sequential data. During the course of auto-encoding, it is essentially work as flexible N-gram models that generate identity vectors in NLP as we generate(or re-represent) the input space from the noisy-channeled sequential inference model. This means that we may be able to apply other variants of such N-gram task as alternatives to auto-encoding task. For example, we can apply Skip-gram model[27] to our model where our auto-encoder outputs window of input $\{\underline{\tilde{x}}_i\}$ around $\underline{x}_t$, $i \in [t - t_{window}, t + t_{window}]$, rather than $\underline{\tilde{x}}_t$. By encoding neighbourhoods not itself, the RNN memories may be overlook temporal dependencies as how the Skip-gram model benefited language models. Again, attention mechanism and sequential AE did not perform during the investigation and it is a disappointing fact. Exploration on their proper use should be desirable.

Stability of GRU over LSTM is also the thing bluntly discussed in this report. Equivalence or differences of gated mechanism among models were not the main topic in this report so we simple took advantage of GRU however, it is worth to investigate how the GRU works well even without gradient control or other typical requirements for other models. It is even not clear how the difference in gate structure affects to the cell memories in terms of their resolution to sequential representation.

Lastly, hyper-parameter optimisation is not thoroughly done in this paper. This is because our deep model requires so many hyper-parameters so that it is nearly impossible to look up each of them concretely. It is often tricky to rely on heuristic approach, as one hyper-parameter can shift anothers hugely for new optima(ex: Dropout probability sudden takes huge importance when CNN becomes deeper in section 5.a and 5.a.2). Effectiveness of dropout for gated-RNNs is not thoroughly investigated in this report after section 2.b.3 by relying on GRU's stability. However, if we stack GRU deeper such about 4 layers, then we may need to consider it as an important factor. In addition, various kernel size on each time-step of input is one of the thing needed to explore where we only used full-length on this($Dim(Kernel)_{input|time} = 25$ for the first convolution layer, and 1 for upper convolutions) to get our experiments done on time.

References

[1] D.E., Rumelhart. et al.
Learning Internal Representations by Error Propagation.
ACM, 1986.

[2] Ian, Goodfellow.
Deep Learning.
MIT press, 2016.

[3] Yann, Lecun.
Efficient BackProp.
yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

[4] Razvan, Pascanu. et al.
On the diculty of training recurrent neural networks.
JMLR, 2013.

[5] Sepp, Hochreiter. et al.
Long Short-Term Memory.
Neural Computation, 1997.

[6] Martens, James. et al.
Learning recurrent neural networks with hessian-free optimization.
ICML, 2011.

[7] Geoffrey E., Hinton. et al.
Improving neural networks by preventing co-adaptation of feature detectors.
Arxiv, 2012.

[8] Wojciech, Zaremba. et al.
Recurrent Neural Network Regularization.
ICLR, 2015.

[9] Michiel, Hermans. Benjamin, Schrauwen.
Training and Analyzing Deep Recurrent Neural Networks NIPS, 2013.

[10] Alex, Graves. et al.
Hybrid Speech Recognition with Deep Bidirectional LSTM.
ASRU, 2013.

[11] Kyunghyun, Cho. et al.
On the properties of neural machine translation: Encoder-decoder approaches.
Arxiv, 2014.

[12] Junyoung, Chung. Kyunghyun, Cho. et al.
Gated Feedback Recurrent Neural Networks.
Arxiv, 2015.

[13] Jie, Zhou. et al.
End-to-end Learning of Semantic Role Labeling Using Recurrent Neural Networks.
ACL, 2015.

[14] Dzmitry, Bahdanau. et al.
Neural Machine Translation by Jointly Learning to Align and Translate.
ICLR, 2015.

[15] Ming Thang, Luong. et al.
Effective Approaches to Attention-based Neural Machine Translation.
ACL, 2015.

[16] Nitish, Srivastava. et al.
Unsupervised Learning of Video Representations using LSTMs.
Arxiv, 2016.

[17] Jason, Lee. et al.
Fully Character-Level Neural Machine Translation without Explicit Segmentation.
Arxiv, 2016.

[18] Rupesh K., Srivastava. et al.
Training Very Deep Networks.
Arxiv, 2015.

[19] Kaiming, He. et al. zz *Deep Residual Learning for Image Recognition.*
Arxiv, 2015.

[20] Marta R., Costa-Jussa. et al.
Character-based neural machine translation.
ACL, 2016.

[21] Yoon, Kim., et al.
Character-Aware Neural Language Models.
Arxiv, 2015.

[22] Caglar, Gulcehre., et al.
Learned-Norm Pooling for Deep Feedforward and Recurrent Neural Networks.
Arxiv, 2014.

[23] Yann, Lecun.
Signal recovery from Pooling Representations.
Arxiv, 2013.

[24] Manaal, Faruqui. et al.
Problems With Evaluation of Word Embeddings Using Word Similarity Tasks.
ACL, 2016.

[25] Yann, Lecun.
Learning Invariant Feature Hierarchies.
Arxiv, 2014.

[26] Geoffrey, Hinton.
Reducing the Dimensionality of Data with Neural Networks.
Nature, 2006.

[27] Tomas, Mikolov. et al.
Distributed Representations of Words and Phrases and their Compositionality.
NIPS, 2013.