



BST 261: Data Science II

Lecture 11

RNNs Continued

Heather Mattie
Harvard T.H. Chan School of Public Health
Spring 2 2021



Recipe of the Day!

Dominique Ansel's Peanut Butter Chocolate Crunch Cake





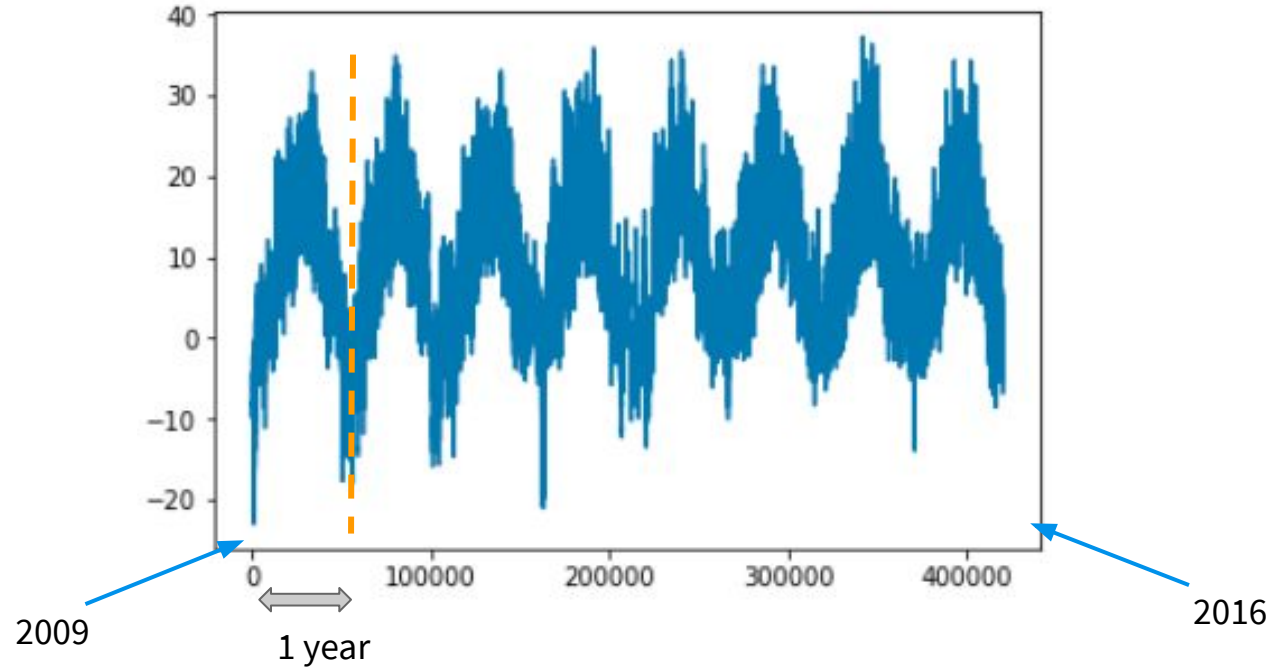
RNNs for Time Series

Example: temperature forecasting

- ◎ RNNs can be applied to any type of sequence data, not just text
- ◎ We will be using a **weather timeseries** dataset recorded at the [Weather Station at Max Planck Institute for Biochemistry](#) in Jena, Germany

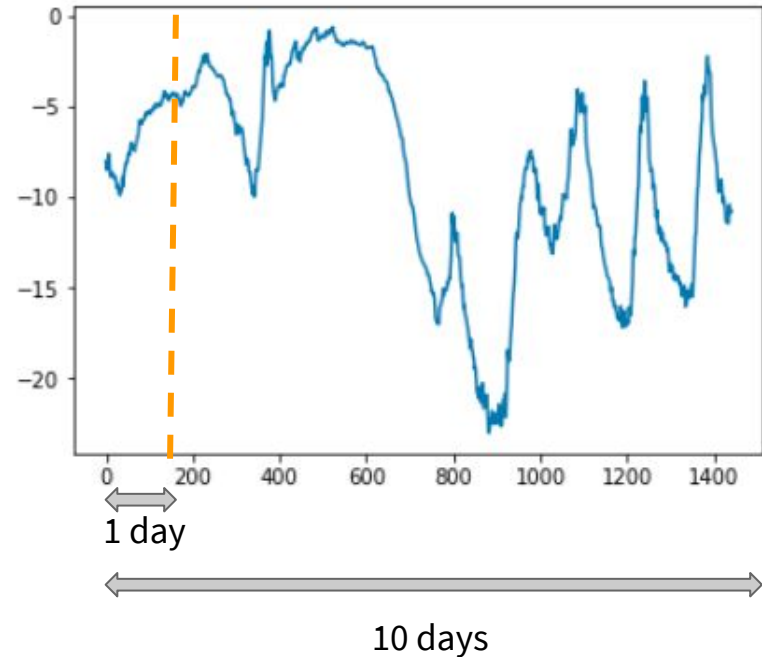


Temperature over time

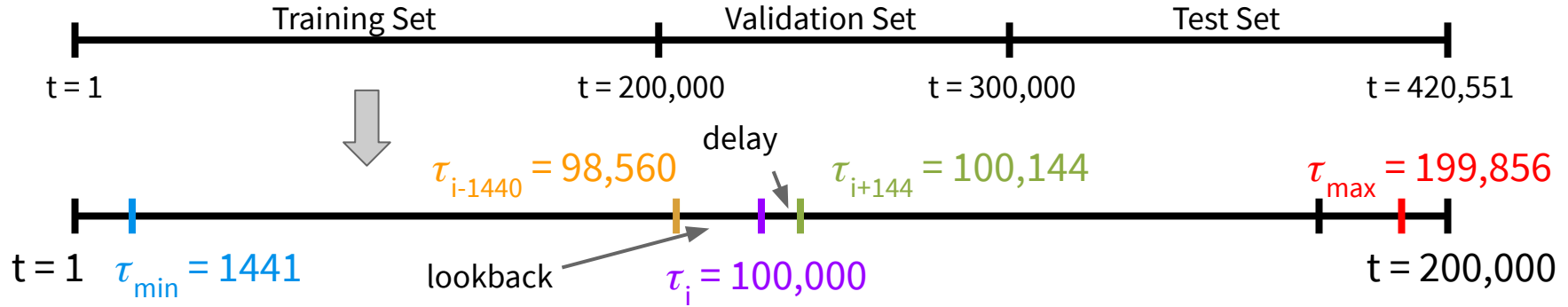


Temperature over time

- Let's plot the temperature over time (a few days)
- Notice that there is periodicity present, but that it isn't as consistent as the last plot - this will make predicting the weather in the next 24 hours using data from a few days beforehand more challenging



Timeline



τ_{\min} : we need all 10 days worth of past data to predict the temperature for the next time point. Thus, the minimum value τ_i can take is 1400 (10 days of previous data) + 1 (time point) = 1441. If we choose $\tau_i < 1441$, we won't have enough prior data to make a prediction.

τ_{\max} : we need 24 hours worth of data after this point in order to have a point to make a prediction for. Thus, the maximum value τ_i can take is $200,000 - 144 = 199,856$. If we choose $\tau_i > 199,856$, we won't have a data point to make a prediction for.

Steps:

1. Randomly sample a point in time, τ_i , between τ_{\min} and τ_{\max}
2. Keep 10 days of data prior to τ_i and 24 hours after τ_i .
3. Repeat this process multiple times
4. Split training examples into batches
5. Feed into the network
6. Repeat similar process for validation and test sets

Whether to shuffle
points in time or
not.

While true,
meaning while
there are still
examples to
include in a batch

Colab Notebook

```
1 def generator(data, lookback, delay, min_index, max_index,  
2               shuffle=False, batch_size=128, step=6):  
3     if max_index is None:  
4         max_index = len(data) - delay - 1  
5     i = min_index + lookback  
6     while 1:  
7         if shuffle:  
8             rows = np.random.randint(  
9                 min_index + lookback, max_index, size=batch_size)  
10        else:  
11            if i + batch_size >= max_index:  
12                i = min_index + lookback  
13            rows = np.arange(i, min(i + batch_size, max_index))  
14            i += len(rows)  
15  
16            samples = np.zeros((len(rows),  
17                               lookback // step,  
18                               data.shape[-1]))  
19            targets = np.zeros((len(rows),))  
20            for j, row in enumerate(rows):  
21                indices = range(rows[j] - lookback, rows[j], step)  
22                samples[j] = data[indices]  
23                targets[j] = data[rows[j] + delay][1]  
24            yield samples, targets
```

For the training set -
randomly choose
points in time

For the validation
and test sets -
choose batches of
timesteps (in
chronological
order)

Floor division

One batch of
input data

Corresponding target
temperatures

Use the generator function to instantiate three generators, one for training, one for validation and one for testing.

Each will look at different temporal segments of the original data: the training generator looks at the first 200,000 timesteps, the validation generator looks at the following 100,000, and the test generator looks at the remainder.

```
1 lookback = 1440
2 step = 6
3 delay = 144
4 batch_size = 128
5
6 train_gen = generator(float_data,
7                       lookback=lookback,
8                       delay=delay,
9                       min_index=0,
10                      max_index=200000,
11                      shuffle=True,
12                      step=step,
13                      batch_size=batch_size)
14 val_gen = generator(float_data,
15                    lookback=lookback,
16                    delay=delay,
17                    min_index=200001,
18                    max_index=300000,
19                    step=step,
20                    batch_size=batch_size)
21 test_gen = generator(float_data,
22                     lookback=lookback,
23                     delay=delay,
24                     min_index=300001,
25                     max_index=None,
26                     step=step,
27                     batch_size=batch_size)
28
29 # This is how many steps to draw from `val_gen`
30 # in order to see the whole validation set:
31 val_steps = (300000 - 200001 - lookback) // batch_size
32
33 # This is how many steps to draw from `test_gen`
34 # in order to see the whole test set:
35 test_steps = (len(float_data) - 300001 - lookback) // batch_size
36
37 print(val_steps)
38 print(test_steps)
```

Temperature Forecasting

- ⊙ We need to come up with a baseline benchmark to beat
- ⊙ Common-sense approach: always predict that the temperature 24 hours from now will be equal to the temperature now
- ⊙ We'll use mean absolute error (MAE) to measure loss

```
1 def evaluate_naive_method():
2     batch_maes = []
3     for step in range(val_steps):
4         samples, targets = next(val_gen)
5         preds = samples[:, -1, 1]
6         mae = np.mean(np.abs(preds - targets))
7         batch_maes.append(mae)
8     print(np.mean(batch_maes))
9
10 evaluate_naive_method()
```

0.2897359729905486

We get MAE = 0.29.

Since our temperature data has been normalized to be centered at 0 and have a standard deviation of 1, this number is not immediately interpretable. It translates to an average absolute error of 0.29 * temperature_std degrees Celsius, i.e. 2.57°C.

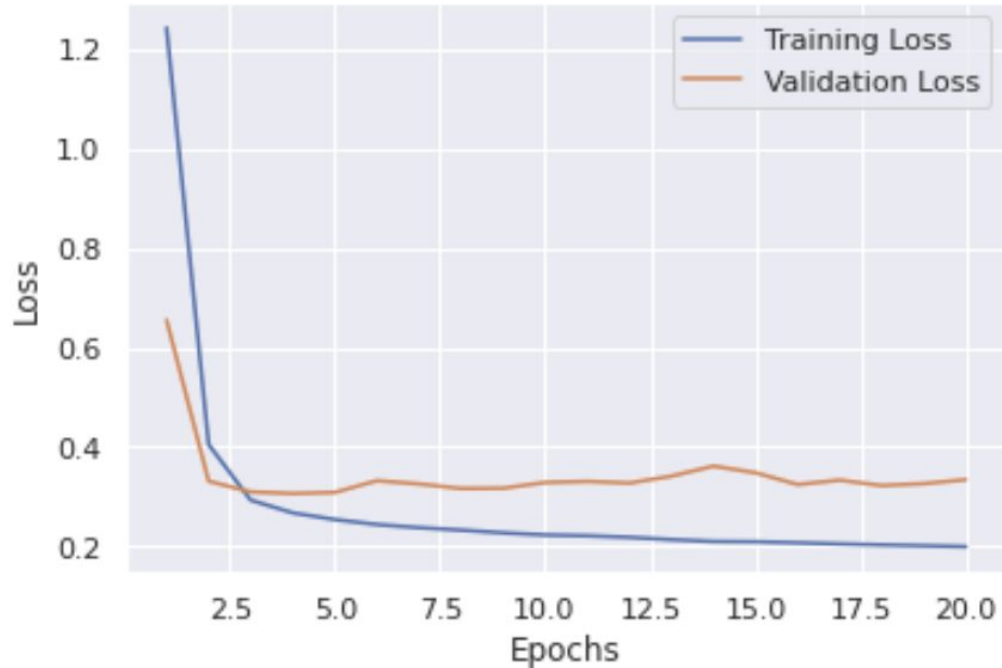
That's a fairly large average absolute error – now the task is to leverage our knowledge of deep learning to do better.

Temperature Forecasting - Simple Model

- Let's first try a simple model (MLP) before developing a more complex one
- In general it's best to start with a basic model and then work your way up in complexity

```
1 model = tf.keras.models.Sequential([
2     tf.keras.layers.Flatten(input_shape=(lookback // step, float_data.shape[-1])),
3
4     tf.keras.layers.Dense(32, activation='relu'),
5     tf.keras.layers.Dense(1)
6 ])
7
8 model.compile(optimizer = tf.keras.optimizers.RMSprop(),
9               loss='mae')
10
11 history = model.fit(train_gen,
12                     steps_per_epoch=500,
13                     epochs=20,
14                     validation_data=val_gen,
15                     validation_steps=validation_steps)
```

Temperature Forecasting - Simple Model



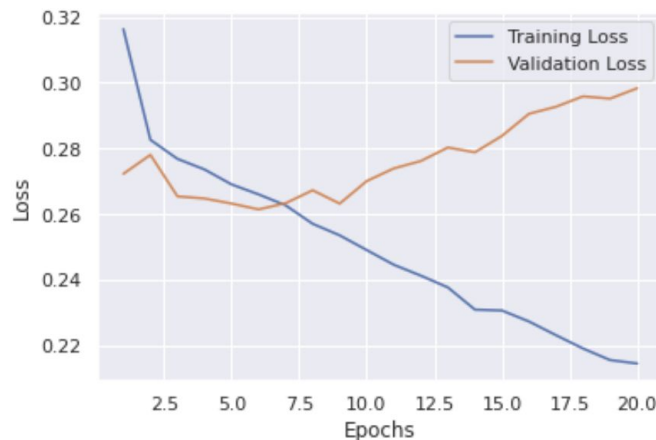
We get MAEs above 0.3 for the validation loss - worse than our benchmark.

This shows that the simple model isn't complex enough for our data and task (it's not taking time into account), and that some benchmarks can be difficult to beat.

Temperature Forecasting - RNN

This model is better than the previous simple model and the common-sense baseline. There is evidence of overfitting, so let's try dropout next.

```
1 model = tf.keras.models.Sequential([
2     tf.keras.layers.GRU(32, input_shape=(None, float_data.shape[-1])),
3
4     tf.keras.layers.Dense(1)
5 ])
6
7 model.compile(optimizer = tf.keras.optimizers.RMSprop(),
8               loss='mae')
9
10 history = model.fit(train_gen,
11                     steps_per_epoch=500,
12                     epochs=20,
13                     validation_data=val_gen,
14                     validation_steps=val_steps)
```



The background of the slide is a light gray network diagram. It consists of numerous small circular nodes, some of which are solid gray and others are hollow with a gray outline. These nodes are interconnected by a web of thin, light gray lines, creating a complex, interconnected pattern that resembles a neural network or a data graph.

Recurrent Dropout

Recurrent Dropout

- ◎ It turns out that the classic technique of dropout we saw in earlier lectures can't be applied in the same way for recurrent layers
 - Applying dropout before a recurrent layer impedes learning rather than helping to implement regularization
- ◎ The proper way to apply dropout with a recurrent network was discovered in 2015
 - Yarin Gal, "[Uncertainty in Deep Learning \(PhD Thesis\)](#),"
 - **The same pattern of dropped units should be applied at every timestep**

Recurrent Dropout

- ◎ This allows the network to properly propagate its learning error rate through time - a temporally random dropout pattern would disrupt the error signal and hinder the learning process
- ◎ Yarin's mechanism has been built into Keras
- ◎ Every recurrent layer has 2 dropout-related arguments:
 - **dropout**: a float number specifying the dropout rate for input units of the layer
 - **recurrent_dropout**: a float number specifying the dropout rate of the recurrent units

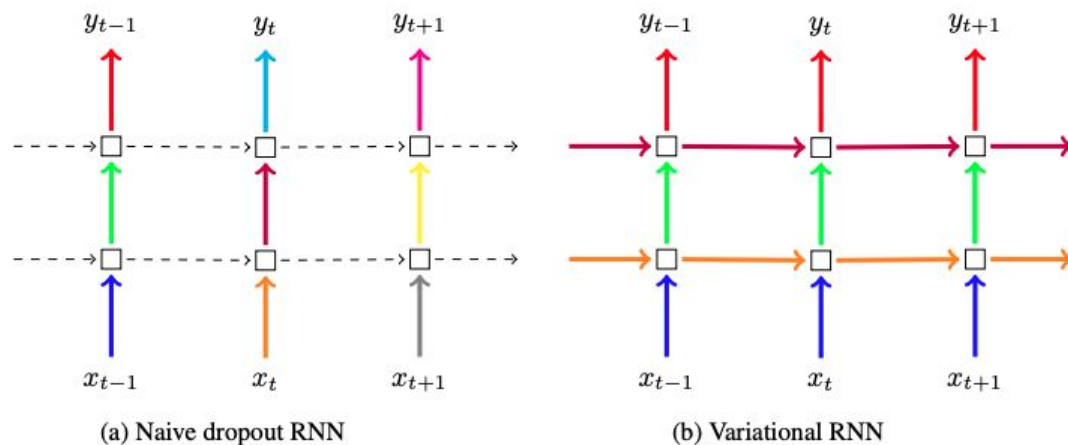
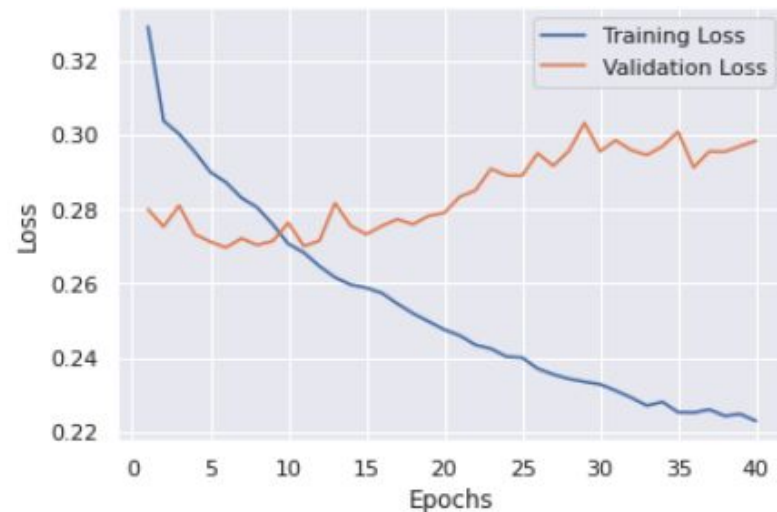


Figure 1: **Depiction of the dropout technique following our Bayesian interpretation (right) compared to the standard technique in the field (left).** Each square represents an RNN unit, with horizontal arrows representing time dependence (recurrent connections). Vertical arrows represent the input and output to each RNN unit. Coloured connections represent dropped-out inputs, with different colours corresponding to different dropout masks. Dashed lines correspond to standard connections with no dropout. Current techniques (naive dropout, left) use different masks at different time steps, with no dropout on the recurrent layers. The proposed technique (Variational RNN, right) uses the same dropout mask at each time step, including the recurrent layers.

Source: <https://arxiv.org/pdf/1512.05287.pdf>

Recurrent Dropout in Keras

```
1 model = tf.keras.models.Sequential([
2     tf.keras.layers.GRU(32,
3         dropout = 0.2,
4         recurrent_dropout=0.2,
5         input_shape=(None, float_data.shape[-1])),
6
7     tf.keras.layers.Dense(1)
8 ])
9
10 model.compile(optimizer = tf.keras.optimizers.RMSprop(),
11               loss='mae')
12
13 history = model.fit(train_gen,
14                     steps_per_epoch=500,
15                     epochs=40,
16                     validation_data=val_gen,
17                     validation_steps=val_steps)
```



This helps a little with overfitting - increasing the dropout percentage might help more.

We have more stable evaluation scores, but our best scores are not much lower than they were previously



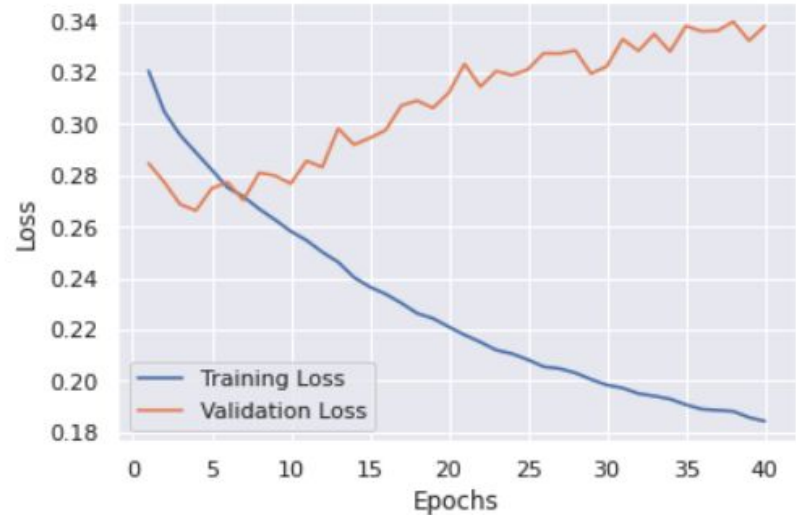
Stacking Recurrent Layers

Stacking Recurrent Layers

- ◎ Because we have hit a performance bottleneck, we should consider increasing the capacity of the network - make the model more complex
- ◎ Increasing network capacity is typically done by increasing the number of units in the layers or adding more layers.
- ◎ Recurrent layer stacking is a classic way to build more-powerful recurrent networks: for instance, what currently powers the Google Translate algorithm is a stack of 8 large LSTM layers—that's huge!

Stacking Recurrent Layers in Keras

```
1 model = tf.keras.models.Sequential([
2     tf.keras.layers.GRU(32,
3         dropout = 0.1,
4         recurrent_dropout=0.5,
5         return_sequences=True,
6         input_shape=(None, float_data.shape[-1])),
7     tf.keras.layers.GRU(64, activation='relu',
8         dropout = 0.1,
9         recurrent_dropout=0.5),
10    tf.keras.layers.Dense(1)
11 ])
12
13 model.compile(optimizer = tf.keras.optimizers.RMSprop(),
14     loss='mae')
15
16 history = model.fit(train_gen,
17     steps_per_epoch=500,
18     epochs=40,
19     validation_data=val_gen,
20     validation_steps=val_steps)
```



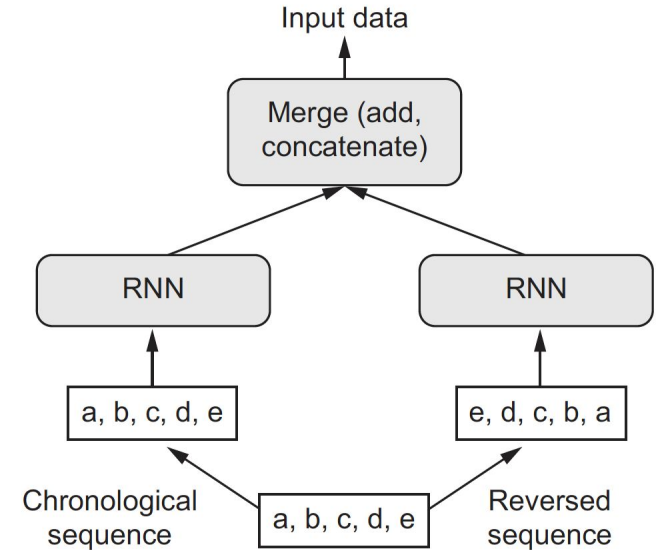
- The overfitting becomes worse, signaling the network capacity is too high, i.e. the model is too complex and has too many parameters.
- It would probably be best to drop the added layer and increase the number of nodes in the first GRU layer

The background of the slide is a light gray network diagram. It consists of numerous small circular nodes, some of which are highlighted with a darker gray or blue fill. These nodes are interconnected by a web of thin, light gray lines, creating a complex, interconnected pattern that resembles a neural network or a data graph. The overall aesthetic is clean and technical.

Bidirectional Recurrent Layers

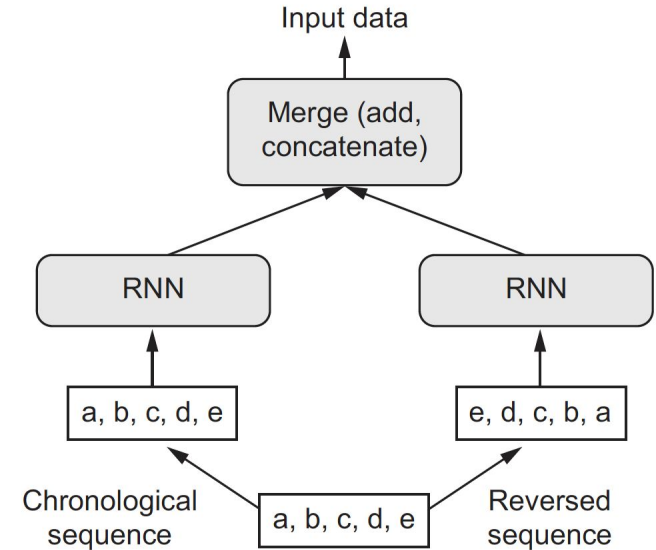
Bidirectional RNNs

- ◎ A bidirectional RNN (BRNN) can offer greater performance on certain tasks
- ◎ Frequently used in natural-language processing (NLP)
- ◎ BRNNs exploit the order sensitivity of RNNs



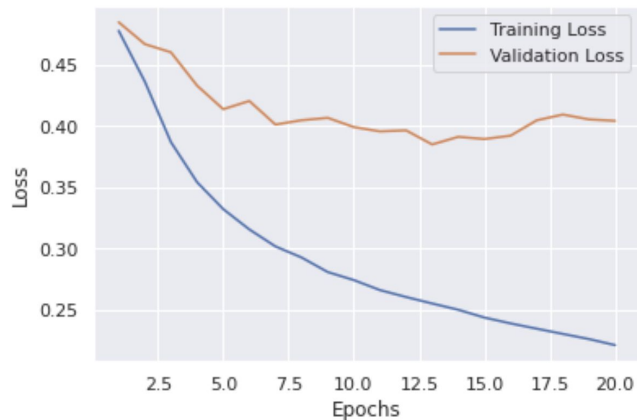
Bidirectional RNNs

- Uses 2 regular RNNs, each of which processes the input sequence in one direction (chronologically and anti chronologically), and then merges their representations
- Catches patterns that may be overlooked by a regular RNN



Temperature Forecasting with a BRNN

It performs about as well as the regular GRU layer. It's easy to understand why: all of the predictive capacity must be coming from the chronological half of the network, since the anti-chronological half is known to be severely underperforming on this task (again, because the recent past matters much more than the distant past in this case).



```
1 model = keras.Sequential([
2     layers.Bidirectional(tf.keras.layers.GRU(32),
3                           input_shape=(None, float_data.shape[-1])),
4
5     layers.Dense(1)
6 ])
7
8 model.compile(optimizer = tf.keras.optimizers.RMSprop(),
9               loss='mae')
10 history = model.fit(train_gen,
11                     steps_per_epoch=500,
12                     epochs=40,
13                     validation_data=val_gen,
14                     validation_steps=val_steps)
```

Summary

- ◎ There are several other things you can try to improve performance
 - Change the number of units in each recurrent layer
 - Try using LSTM layers instead of GRU layers
 - Change the learning rate used by the RMSprop optimizer (or any optimizer)
 - Try a bigger densely connected classifier on top of the recurrent layers

A background network diagram consisting of numerous light blue circular nodes connected by thin, light gray lines, creating a complex web-like pattern across the entire slide.

1D Convolution for Sequence Data

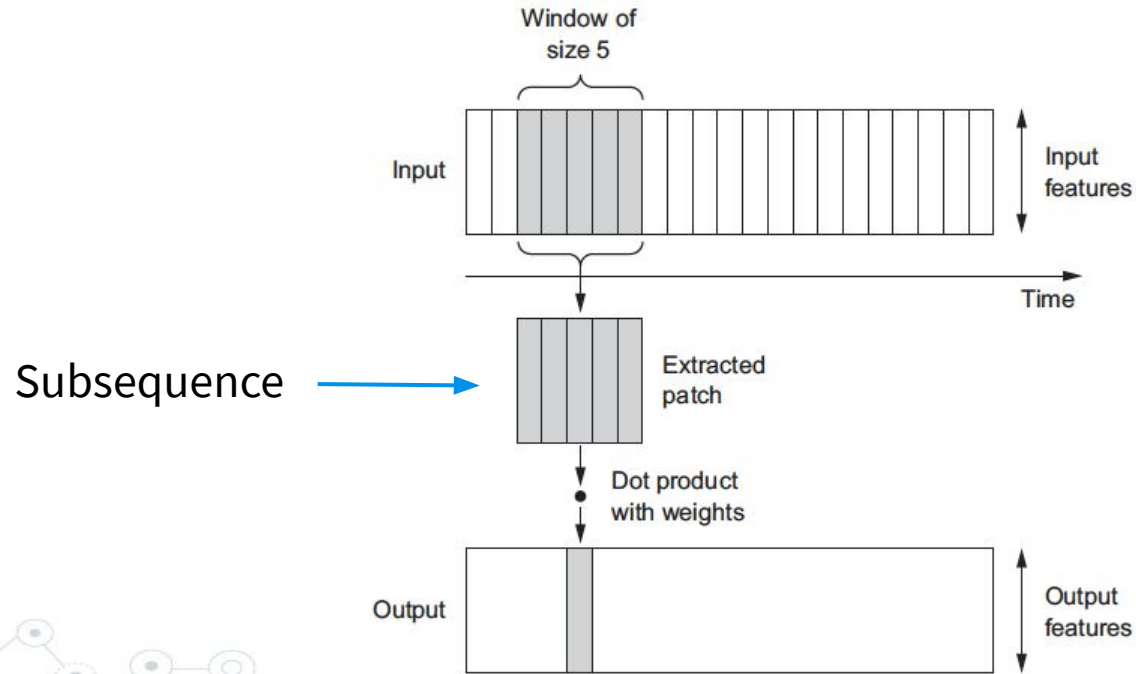
1D Convolution for Sequence Data

- ◎ Recall that CNNs can extract features from local input patches and then recognize them anywhere
- ◎ If we think of time as a spatial dimension, we can use 1D CNNs for sequence data
- ◎ Great for audio generation and machine translation
- ◎ Faster to run than RNNs

1D Convolution for Sequence Data

- ◎ **1D CNNs extract subsequences (patches)** from sequences and perform the same transformation on each subsequence
 - A pattern learned at a specific position of a sequence can be recognized at a different position (translation invariant)
- ◎ Pooling in this case is similar to what we have seen before - output the maximum or average value of a subsequence

1D Convolution for Sequence Data

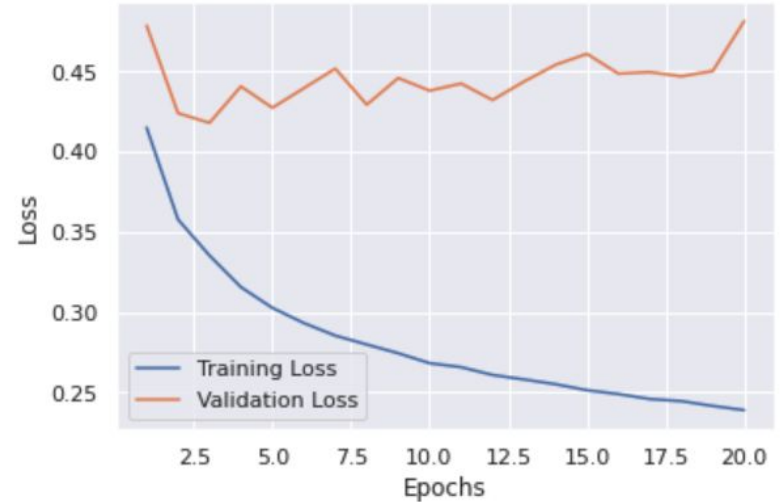


1D Convolution for Sequence Data

- ◎ In Keras, use the **Conv1D** layer
 - Takes as input (samples, time, features)
 - Returns 3D tensors
- ◎ Combine Conv1D layer with a **MaxPooling1D** layer
- ◎ End with a **GlobalMaxPooling** or **Flatten** layer
- ◎ Can use larger windows for 1D CNNs - typically windows of size 7 or 9
 - In a 2D convolution layer, a 3 x 3 filter contains 9 feature vectors
 - A 1D convolution layer with a window of size 3 contains only 3 vectors

Stacking 1D CNN layers

```
1 model = keras.Sequential([
2     layers.Conv1D(32, 5, activation='relu',
3                   input_shape=(None, float_data.shape[-1])),
4     layers.MaxPooling1D(3),
5     layers.Conv1D(32, 5, activation='relu'),
6     layers.MaxPooling1D(3),
7     layers.Conv1D(32, 5, activation='relu'),
8     layers.GlobalMaxPooling1D(),
9     layers.Dense(1)
10 ])
11
12 model.compile(optimizer = tf.keras.optimizers.RMSprop(),
13              loss = 'mae')
14
15 history = model.fit(train_gen,
16                    steps_per_epoch = 500,
17                    epochs = 20,
18                    validation_data=val_gen,
19                    validation_steps=val_steps)
```

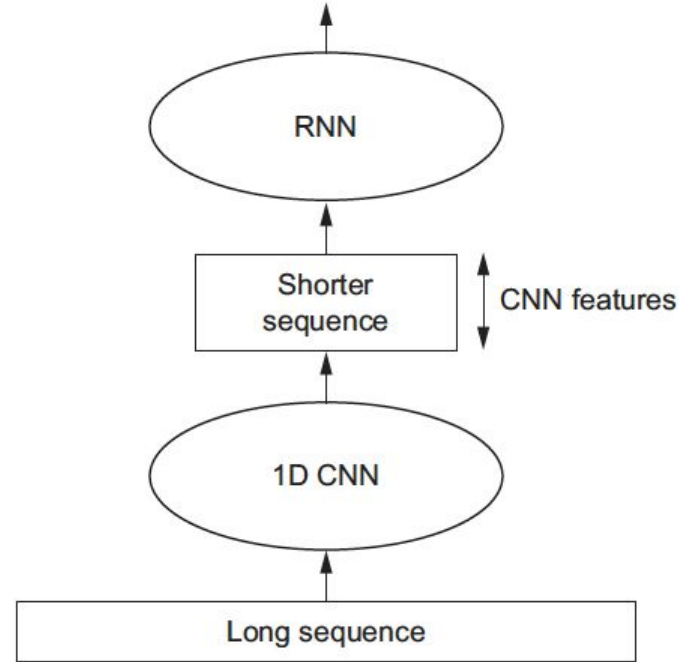


Stacking 1D CNN layers

- ◎ 1D CNNs process subsequences independently and aren't sensitive to the order of the timesteps - thus, they don't perform well when faced with long sequences
- ◎ Could try stacking 1D CNN layers, but still doesn't induce order sensitivity
- ◎ Doesn't beat the common-sense baseline
 - Due to the fact that 1D CNNs aren't time sensitive
- ◎ Let's try combining CNNs and RNNs

Combining CNNs and RNNs

- ◎ To combine the speed of 1D CNNs with the time sensitivity of RNNs, could do the following:
 - Use the 1D CNN as a preprocessing step
 - Feed this output into an RNN
- ◎ The CNN turns long sequences into much shorter sequences



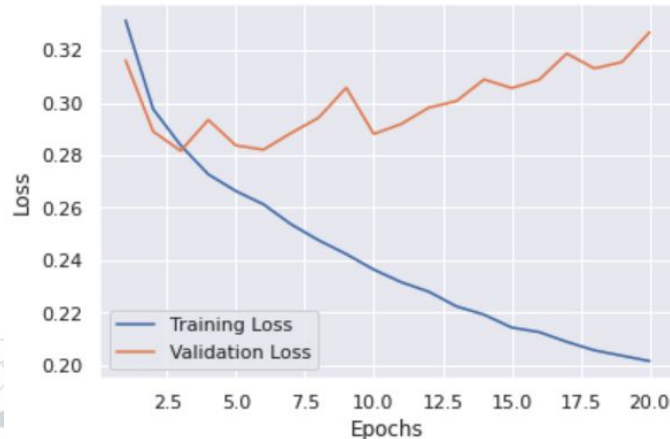
Temperature Forecasting

- ◎ With the combination of a CNN and RNN we can now either look at data from longer ago (increase the lookback parameter), or look at high-resolution timeseries (decrease the step parameter)
- ◎ Let's decrease the step parameter by half - this gives us sequences that are twice as long
- ◎ Temperature data is now sampled at a rate of 1 point per 30 minutes

```
1 model = keras.Sequential([
2     layers.Conv1D(32, 5, activation='relu',
3                   input_shape=(None, float_data.shape[-1])),
4     layers.MaxPooling1D(3),
5     layers.Conv1D(32, 5, activation='relu'),
6     layers.GRU(32, dropout=0.1, recurrent_dropout=0.5),
7     layers.Dense(1)
8 ])
9
10 model.summary()
11
12 model.compile(optimizer = tf.keras.optimizers.RMSprop(),
13              loss = 'mae')
14
15 history = model.fit(train_gen,
16                    steps_per_epoch = 500,
17                    epochs = 20,
18                    validation_data=val_gen,
19                    validation_steps=val_steps)
```

This model doesn't perform as well as the regularized GRU model, but it does run a lot faster.

This model also looks at twice as much data as the other model, but that doesn't seem to help with accuracy in this case - it could for other data sets.



Summary

- ◎ In the same way that 2D convnets perform well for processing visual patterns in 2D space, 1D convnets perform well for processing temporal patterns. They offer a faster alternative to RNNs on some problems, in particular natural language processing tasks.
- ◎ Typically, 1D convnets are structured much like their 2D equivalents from the world of computer vision: they consist of stacks of Conv1D layers and Max-Pooling1D layers, ending in a global pooling operation or flattening operation.

Summary

- Because RNNs are extremely expensive for processing very long sequences, but 1D convnets are cheap, it can be a good idea to use a 1D convnet as a preprocessing step before an RNN, shortening the sequence and extracting useful representations for the RNN to process.

The background of the slide features a complex, light gray network pattern. It consists of numerous small circles, some of which are solid and others are hollow, connected by thin, intersecting lines that form a web-like structure across the entire page.

Text Generation with LSTM

Generative Deep Learning

- ◎ Generative deep learning methods using RNNs and CNNs have been around for awhile, but have recently been getting a lot of attention
 - 2002: Douglas Eck applied LSTM to music generation - he is now at Google Brain and started a research group called Magenta to use deep learning to create engaging music
 - 2013: Alex Graves applies recurrent mixture density networks to generate human-like handwriting
 - Many more, and one that we'll talk about today
- ◎ Many researchers in this field have said that “generating sequential data is the closest computers get to **dreaming**”

Generative RNNs for Text

- ◎ RNNs have been successfully used for
 - Music generation
 - Dialogue generation
 - Image generation
 - Speech synthesis
 - Molecule design
- ◎ Main idea for text generation: train a model to predict the next token or next few tokens in a sequence

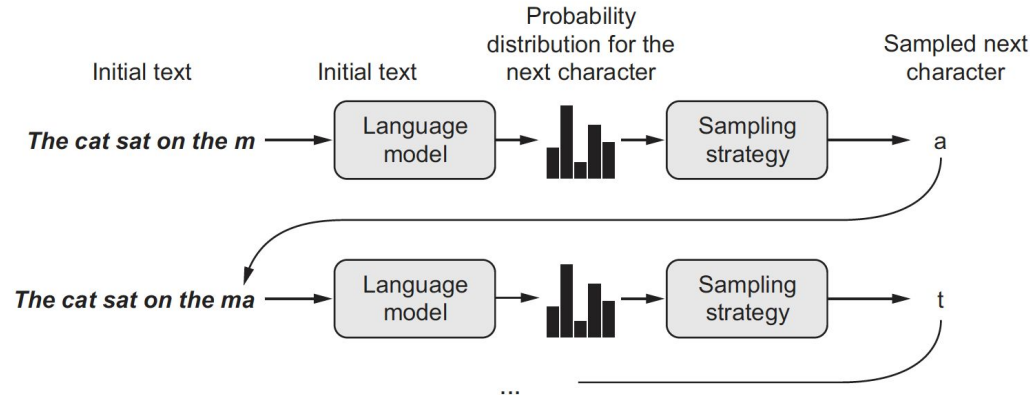
Generative RNNs for Text

- ◎ **Language Model:** any network that can model the probability of the next token given the previous ones
 - Captures the latent space of language - its statistical structure
 - Once it is trained, you can sample from it to generate new sequences

Generative RNNs for Text

◎ Process

- 1. Feed it an initial string of text (called conditioning data)
- 2. Ask the model to generate the next character or word
- 3. Add the generated output back to the input data
- 4. Repeat many times



Generative RNNs for Text

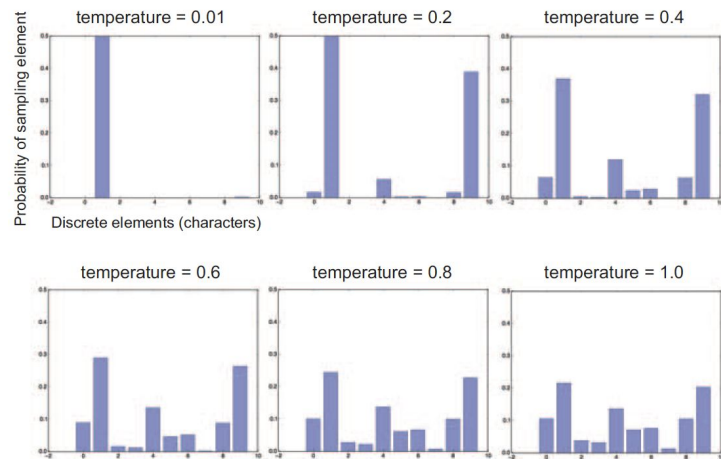
- ◎ We can choose the next character or word in different ways - some are better than others
- ◎ A naive approach is greedy sampling - always choosing the most likely next character or word
 - This results in repetitive, predictable strings and not very coherent language

Generative RNNs for Text

- ◎ Better approach is stochastic sampling
 - Sample next characters or words with specific probability from a probability distribution
 - Allows even unlikely characters or words to be sampled at times, generating more interesting and creative sentences
 - Doesn't offer a way of controlling the randomness in the sampling process

Generative RNNs for Text

- ◎ New parameter to tune: **softmax temperature**
 - Controls the amount of randomness
 - More randomness = similar probability for every character or word and results in more interesting output
 - Less randomness = higher probability for just one or a few characters or words and results in repetitive output
 - Can change the amount of randomness via the temperature value
 - ◎ **Higher** temperature = more **randomness**
 - ◎ **Lower** temperature = more **deterministic**



Character-level LSTM Generation

- ◎ Need a lot of data to train from
- ◎ Can choose from many sources, referred to as a **corpus**
 - Wikipedia
 - The Lord of the Rings
 - The writings of Nietzsche translated into English
- ◎ Let's see an example with the writings of Nietzsche as our corpus
 - [Colab notebook](#)

Character-level LSTM Generation

- ◎ Training the language model and sampling from it:
- ◎ Given a trained model and a seed text snippet, do the following repeatedly
 - 1. Draw from the model a probability distribution for the next character, given the generated text available so far
 - 2. Reweight the distribution to a certain temperature
 - 3. Sample the next character at random according to the reweighted distribution
 - 4. Add the new character at the end of the available text

Character-level LSTM Generation

- ◎ Random seed:
 - “new faculty, and the jubilation reached its climax when kant”
- ◎ Output at epoch 20 with temperature = 0.2:

“new faculty, and the jubilation reached its climax when kant and such a man in the same time the spirit of the surely and the such the such as a man is the sunligh and subject the present to the superiority of the special pain the most man and strange the subjection of the special conscience the special and nature and such men the subjection of the special men, the most surely the subjection of the special intellect of the subjection of the same things and”

Character-level LSTM Generation

- ◎ Random seed:
 - “new faculty, and the jubilation reached its climax when kant”
- ◎ Output at epoch 20 with temperature = 0.5:

“new faculty, and the jubilation reached its climax when kant in the eterned and such man as it's also become himself the condition of the experience of off the basis the superiority and the special morty of the strength, in the langus, as which the same time life and "even who discless the mankind, with a subject and fact all you have to be the stand and lave no comes a troveration of the man and surely the conscience the superiority, and when one must be w ”

Character-level LSTM Generation

- ◎ Random seed:
 - “new faculty, and the jubilation reached its climax when kant”
- ◎ Output at epoch 20 with temperature = 1.0:

“new faculty, and the jubilation reached its climax when kant, as a periling of manner to all definites and transpects it it so hicable and ont him artiar resull too such as if ever the proping to makes as cnecience. to been juden, all every could coldiciousnike hother aw passife, the plies like which might thiod was account, indifferent germin, that everythery certain destrution, intellect into the deteriorablen origin of moralian, and a lessority o”

Character-level LSTM Generation

- ◎ Random seed:
 - “new faculty, and the jubilation reached its climax when kant”
- ◎ Output at epoch 60 (model has now fully converged) with temperature = 0.2:

“cheerfulness, friendliness and kindness of a heart are the sense of the spirit is a man with the sense of the sense of the world of the self-end and self-concerning the subjection of the strengthorixes–the subjection of the subjection of the subjection of the self-concerning the feelings in the superiority in the subjection of the subjection of the spirit isn’t to be a man of the sense of the subjection and said to the strength of the sense of the”

Character-level LSTM Generation

- ◎ Random seed:
 - “new faculty, and the jubilation reached its climax when kant”
- ◎ Output at epoch 60 with temperature = 0.5:

“cheerfulness, friendliness and kindness of a heart are the part of the soul who have been the art of the philosophers, and which the one won’t say, which is it the higher the and with religion of the frences. the life of the spirit among the most continuess of the strengther of the sense the conscience of men of precisely before enough presumption, and can mankind, and something the conceptions, the subjection of the sense and suffering and the”

Character-level LSTM Generation

- ◎ Random seed:
 - “new faculty, and the jubilation reached its climax when kant”
- ◎ Output at epoch 60 with temperature = 1.0:

“cheerfulness, friendliness and kindness of a heart are spiritual by the ciuture for the entalled is, he astraged, or errors to our you idstood–and it needs, to think by spars to whole the amvives of the newoatly, prefectly raals! it was name, for example but voludd atu-especity”–or rank onee, or even all "solett increessic of the world and implussional tragedy experience, transf, or insiderar,–must hast if desires of the strubction is be stronges”

Character-level LSTM Generation

- ◎ Low temperature results in repetitive and predictable text, but local structure is highly realistic
- ◎ Higher temperatures result in more interesting, surprising and creative text, sometimes creating new words - but the local structure breaks down and most words are strings of random characters
- ◎ Generally, somewhere in the middle (around 0.5) creates the most interesting text - but this depends on the corpus and the human reading the results