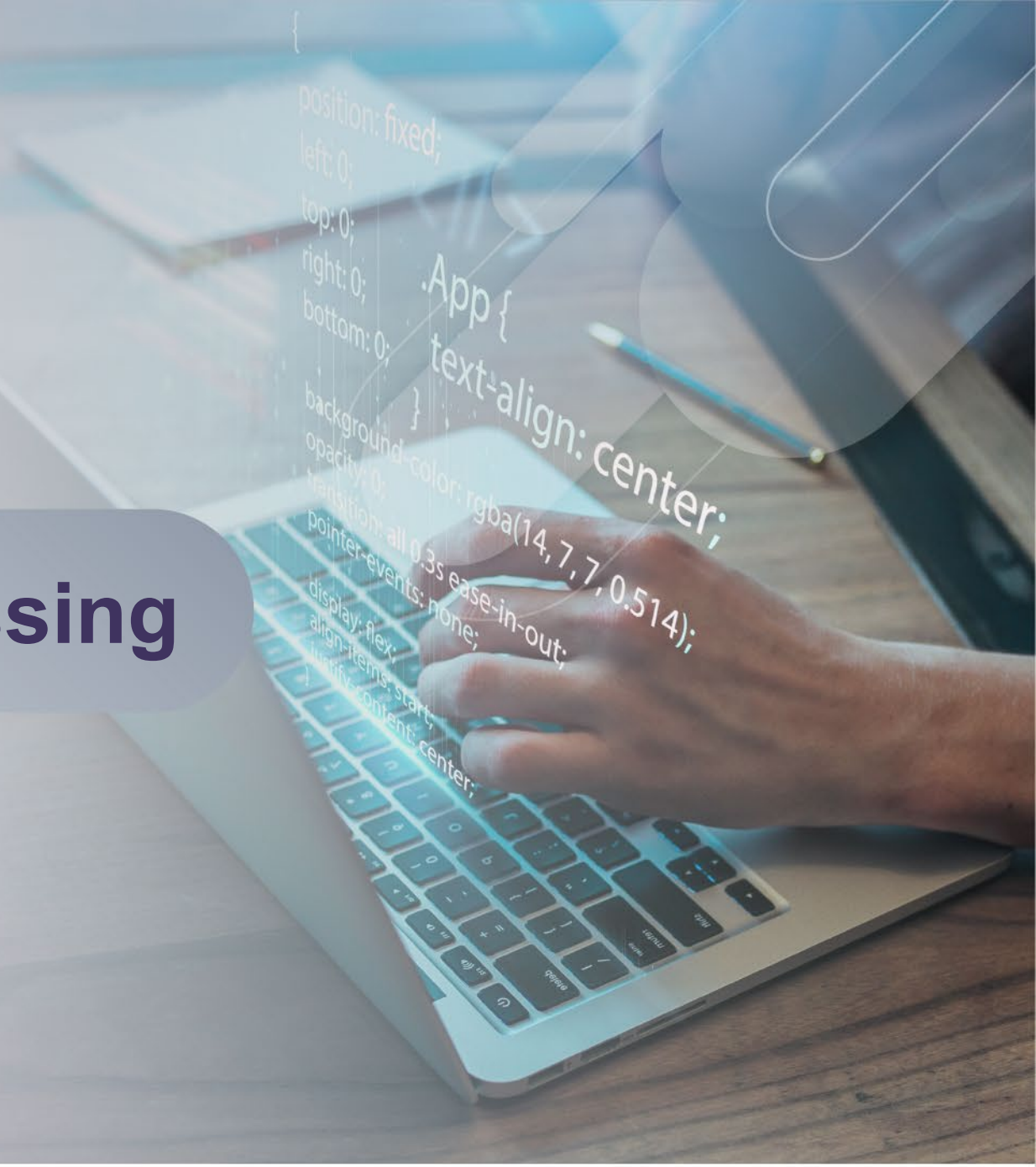
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

EE6405

Natural Language Processing



Dr. S. Supraja
NTU Electrical and Electronic Engineering

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

HyperParameter Tuning

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

HyperParameter Tuning

- Hyperparameters are **external configurations** that influence the training process and performance of NLP models.
- Hyperparameters **control the learning process** and determine the values of model parameters that a learning algorithm ends up learning.
- Proper tuning is crucial for optimising model performance in tasks like text classification, sentiment analysis, and language generation.
- We **cannot know the best value for a model hyperparameter** on a given problem. We may use rules of thumb, copy values used on other problems, or search for the best value by trial and error.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

K-Fold Cross Validation

- Cross-validation is a **resampling procedure** used to evaluate machine learning models on a limited data sample.
- The procedure has a **single parameter** called **k** that refers to the number of groups that a given data sample is to be split into.
- When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.
- Generally results in a **less biased** or **less optimistic** estimate of the model skill **than other methods**, such as a simple train/test split.
- Note that k-fold cross-validation is to **evaluate the model design**, not a particular training. Because you re-trained the model of the same design with different training sets.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

K-Fold

- The general procedure is as follows:

1
Shuffle the dataset **randomly**.

2
Split the dataset into **k groups**.

3
For each unique group:
- **Take the group** as a hold out or **test data set.**
- **Take the remaining groups** as **a training data set.**
- **Fit a model** on the training set and evaluate it on the test set.
- **Retain** the **evaluation score** and **discard the model.**

4
Summarise the skill of the model using the sample of model evaluation scores.

- **Each observation** in the data sample is **assigned to an individual group** and stays in that group for the duration of the procedure.
- **Each sample** is given the opportunity to be **used** in the **hold out** set **1 time** and used to **train** the model **k-1 times**.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

K-Fold

- A poorly chosen value for k may result in a mis-representative idea of the skill of the model, such as a score with a high variance (that may change a lot based on the data used to fit the model), or a high bias, (such as an overestimate of the skill of the model).
- **Three common tactics for choosing a value for k are as follows:**

Representative

The value for k is chosen such that each train/test group of data samples is large enough to be statistically representative of the broader dataset.

k=10

The value for k is fixed to 10, a value that has been found through experimentation to generally result in a model skill estimate with low bias a modest variance.

k=n

The value for k is fixed to n, where n is the size of the dataset to give each test sample an opportunity to be used in the hold out dataset. This approach is called leave-one-out cross-validation.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

K-Fold – Worked Example

- We can split a given dataset using the stratified K-Fold library in SkLearn.
- Here, we set k=5, meaning we will have 5 folds.
- 5 models are trained and evaluated with each fold given a chance to be the held out test set.
- E.g. Model 1: Trained on Fold1+2+3+4, Tested on Fold 5
- We create a list to hold the skill scores for each model (in this case, we will simply use accuracy).

```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score

# Set the number of folds
num_folds = 5
seed = 42

# Initialize StratifiedKFold (or KFold) for cross-validation
kf = StratifiedKFold(n_splits=num_folds, shuffle=True, random_state=seed)

# Initialize an empty list to store the cross-validation scores
cv_scores = []
```


© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

K-Fold – Worked Example

- We train the 5 models using the following loop.
- The models are then discarded after they are evaluated as they have served their purpose.
- The skill scores are collected for each model and summarised for use.

```
X=training_set['review'].values
y=training_set['sentiment'].values
for train_index, test_index in kf.split(X, y):
    X_train, X_val = X[train_index], X[test_index]
    y_train, y_val = y[train_index], y[test_index]

    # Compile the model
    lstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    # Train the model on the training data
    lstm.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val), batch_size=32, verbose=0)

    # Evaluate the model on the validation data
    y_pred = (lstm.predict(X_val) > 0.5).astype("int32")
    acc = accuracy_score(y_val, y_pred)

    # Store the cross-validation score
    cv_scores.append(acc)
```


© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

K-Fold – Worked Example

- As shown, the accuracy score on a K-Fold is significantly more strict than a simple train test split.

```
import statistics
statistics.mean(cv_scores)
```

```
0.5184
```

Mean accuracy score of K-Fold validation

```
accuracy: 0.9005
```

Accuracy score on simple train-test split

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Optimisers

- Optimisers are algorithms that **adjust the model's parameters** during training to **minimise a loss function**.
- They enable neural networks to **learn** from data by **iteratively** updating weights and biases.
- Common Optimisers include Stochastic Gradient Descent (SGD), Adam, and RMSprop.
- Each optimiser has specific **update rules, learning rates, and momentum** to find optimal model parameters for improved performance.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

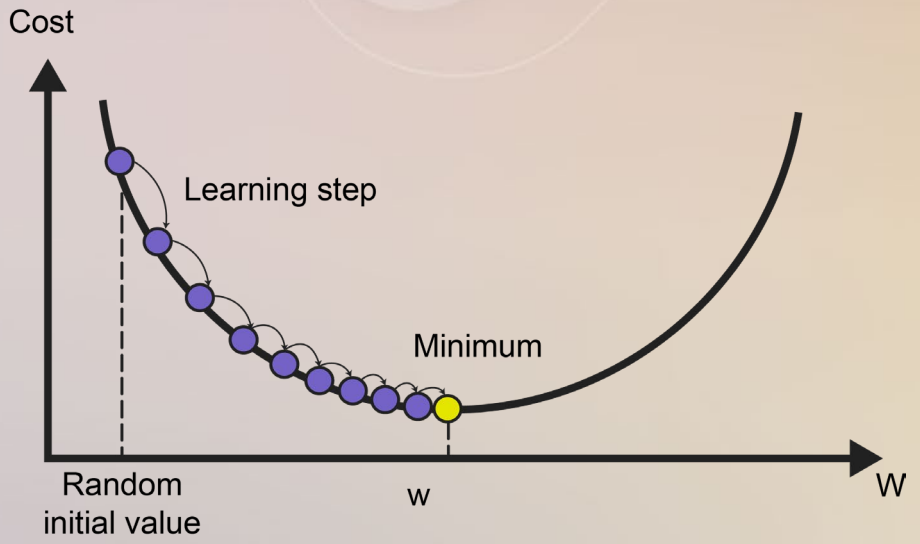
Optimisers – Gradient Descent

- This optimisation algorithm modifies the parameters consistently and to achieve the local minimum.
- **The formula is given as follows:**

$$x_{new} = x - \alpha \cdot f'(x)$$

Where α is the step size that represents how far to move against each gradient with each iteration.

- **Gradient descent works as follows:**
  1. It starts with some coefficients, sees their cost, and searches for cost value lesser than what it is now.
  2. It moves towards the lower weight and updates the value of the coefficients.
  3. The process repeats until the local minimum is reached. A local minimum is a point beyond which it can not proceed.



© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Optimisers – Gradient Descent

- Gradient descent gets too **computationally expensive** as the dataset grows in size.
- To tackle the problem, we have stochastic gradient descent.
- In stochastic gradient descent, instead of taking the whole dataset for each iteration, we r**andomly select the batches of data.**
- The procedure is first to select the initial parameters w and learning rate n. Then **randomly shuffle** the data at each iteration to reach an approximate minimum.
- **The formula is given as:**

$$w := w - \eta \nabla Q_i(w)$$

- Since we are not using the whole dataset but the batches of it for each iteration, the algorithm **experiences** much **more noise** than traditional gradient descent.
- Hence, SGD uses a **higher number of iterations** to reach the local minima.
- Although a higher number of iterations are used to converge upon the local minima, this is still less computationally expensive than traditional gradient descent.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Optimisers – ADAM

- ADAM optimiser, short for Adaptive Moment Estimation optimiser is an extension of the stochastic gradient descent (SGD) algorithm and is designed to update the weights of a neural network during training.
- Unlike SGD, which maintains a single learning rate throughout training, ADAM optimiser dynamically computes individual learning rates based on the past gradients and their second moments.
- ADAM includes a momentum term similar to the one used in SGD with Momentum. This helps accelerate optimisation by accumulating a running average of past gradients.
- The momentum term allows the optimiser to keep moving in a certain direction, even when the gradient has become small, helping to overcome local minima and reach convergence faster.
- In the early iterations, the momentum and the adaptive learning rates may be biased towards zero. To counteract this bias, ADAM includes a bias correction step, especially for the first few iterations.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Optimisers – ADAM

- The **ADAM formula** is as follows:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta w_t}\right]$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta w_t}\right]^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$w_{t+1} = w_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where: m_t is the first-order moment(mean of gradients), v_t is the second-order moment of the gradients (uncentered variance of the gradients), β_1 and β_2 are decay rates for the first and second moments, L is the loss function, and w_t represents model parameters at iteration t.

α is the learning rate and ϵ is a small constant to prevent division by zero.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Optimisers – Gradient Descent

- RMS Prop focuses on accelerating the optimisation process by decreasing the number of function evaluations to reach the local minimum.
- The algorithm keeps the moving average of squared gradients for every weight and divides the gradient by the square root of the mean square.

$$(w,t) \coloneqq \gamma v(w,t-1) + (1-\gamma)\left(\nabla Q_i(w)\right)^2$$ Where γ is the forgetting factor

- The weights are updated by:

$$w \coloneqq w - \frac{\eta}{\sqrt{v(w,t)}}\nabla Q_i(w)$$

- In simpler terms, if there exists a parameter due to which the cost function oscillates a lot, we want to penalise the update of this parameter.
- The algorithm requires lesser tuning than gradient descent algorithms and their variants.
- However, the learning rate has to be defined manually, and the suggested value doesn't work for every application.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Optimisers

- The optimiser for a neural model in Keras can be specified when compiling the model.

```
lstm.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
             optimizer=tf.keras.optimizers.Adam(1e-4),
             metrics=['accuracy'])
```

- Available Optimisers include:

ADAM
Adaptive Moment Estimator

RMSProp
Root Mean Square Propagation

SGD
Stochastic Gradient Descent

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Loss Functions

- A loss or cost function (sometimes also called an error function) is a function that maps an event or values of one or more variables onto a real number intuitively representing some “cost” associated with the event.
- The loss function is a method of evaluating how well your algorithm is modeling your dataset.
- The goal during the training process is to minimize this loss function, effectively improving the model's ability to make accurate predictions.
- The choice of a suitable loss function depends on the nature of the task (e.g., regression, classification, etc.) and the specific requirements of the problem.
- **For classification tasks, two common loss functions are:**

Binary Cross Entropy/Log Loss

Categorical Cross Entropy

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Loss Functions – Binary Cross Entropy

- As the name suggests, this loss function is used for binary classification tasks.
- Binary cross entropy compares each of the predicted probabilities to the actual class output which can be either 0 or 1.
- It then calculates the score that penalizes the probabilities based on the distance from the expected value.
- **The formula is given as:**

$$BCE(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

where: y are the target values and $y\hat{}$ represent the predicted values

- The negative sign and the division by N at the beginning are there to compute the average loss over the entire batch.
- Cross-entropy loss increases as the predicted probability diverges from the actual label.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Loss Functions – Categorical Cross Entropy

- Categorical cross entropy is a commonly used loss function in multiclass classification problems.
- Similar to BCE, it measures the dissimilarity between the true distribution of the classes and the predicted distribution.
- **The formula for CCE is:**

$$CCE(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} \cdot \log(\hat{y}_{ij})$$

where: y are the target values and $\hat{y}$ represent the predicted values, N represents the number of samples in the batch and C represents the number of classes.

- CCE is commonly used with softmax activation in the output layer of a neural network for multiclass classification tasks.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Loss Functions

- The loss function can be computed during model compilation as shown below.

```
lstm.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
             optimizer=tf.keras.optimizers.Adam(clipvalue=0.5),
             metrics=['accuracy'])
```


© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Batch Size

- The batch size represents the number of training samples utilized in one iteration. It is a crucial hyperparameter during the training process.
- Larger batch sizes often lead to faster training times because the model updates its weights less frequently. However, this might result in a less accurate model.
- Smaller batch sizes lead to more frequent weight updates, potentially providing a more accurate model but increasing training time.
- Larger batch sizes are commonly used when training on powerful hardware (GPUs) to take advantage of parallel processing.
- Smaller batch sizes are favored when working with limited computational resources or when using online learning.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Batch Size

- Batch size can be specified using the batch_size parameter while training the model.

```
historyLSTM = lstm.fit(training_set['review'],training_set['sentiment'], epochs=10,
                       validation_data=(test_set['review'],test_set['sentiment']),
                       batch_size=32,
                       validation_steps=30)
```


© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Learning Rate

- The learning rate determines the size of the steps taken during optimisation. It controls the amount by which the model's weights are updated.
- A higher learning rate can result in faster convergence but may lead to overshooting the optimal weights and cause the model to oscillate or diverge.
- A lower learning rate may improve stability but often requires a longer training time.
- Common learning rates range from 0.1 to 0.0001. It's often beneficial to perform a grid search or random search to find an optimal learning rate for a specific model and dataset.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Learning Rate

- The learning rate can also be specified while fitting the model.

```
lstm2.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4, clipvalue=0.5),
              metrics=['accuracy'])
```


© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Epochs

- An "epoch" refers to one complete pass through the entire training dataset during the training of a model.
- During each epoch, the model processes and learns from every example in the training dataset once.
- The number of epochs is a hyperparameter that determines how many times the learning algorithm will work through the entire training dataset.
- Training for too few epochs may result in underfitting, while training for too many epochs may lead to overfitting.
- Early stopping is a technique where training is halted if the model's performance on a validation set stops improving, preventing overfitting.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Epochs

- The early stopping function can be defined as follows:

```
#To implement early Stopping
from keras import callbacks
earlystopping = callbacks.EarlyStopping(monitor ="accuracy",
                                        mode ="max", patience = 5,
                                        restore_best_weights = True)
```

- It can then be included in the model fitting stage:

```
historyLSTM = lstm.fit(training_set['review'],training_set['sentiment'], epochs=10,
                   validation_data=(test_set['review'],test_set['sentiment']),
                   learning_rate=0.001,
                   batch_size=32,
                   validation_steps=30
                   callbacks=[earlystopping])
```


© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Gradient Clipping

- Gradient clipping is a technique to prevent exploding gradients during training by imposing a threshold on the gradients.
- In deep learning, especially in recurrent neural networks (RNNs), exploding gradients can occur, leading to numerical instability and ineffective learning.
- Gradient clipping helps stabilise training by limiting the magnitude of gradients.
- The clipping threshold is a hyperparameter that should be set based on empirical testing. A common value is around 1.0.
- **Gradient clipping can be applied in two common ways:**

Clipping by value

Clipping by norm

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Gradient Clipping

Clipping by value:

- Clipping the gradient by value involves defining a minimum and a maximum threshold. If the gradient goes above the maximum value it is capped to the defined maximum. Similarly, if the gradient goes below the minimum it is capped to the stated minimum value.

Gradient Clipping-by-norm:

- Clipping the gradient by norm ensures that the gradient of every weight is clipped such that its norm won't be above the specified value.
- Where a norm is a function that accepts as input a vector from our vector space V and spits out a real number that tells us how big that vector is.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Gradient Clipping

- Gradient clipping can be specified as part of the model's optimiser:

```
lstm2.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4, clipvalue=0.5),
              metrics=['accuracy'])
```

- "clipvalue" implements clipping by value, in this case limiting all gradients >0.5 to 0.5 and gradients <-0.5 to -0.5.
- Similarly, "clipnorm" implements gradient clipping by norm.
- **The updated gradient vector is given by:**

$$\hat{g} = \frac{clip\ value}{norm} \cdot g$$

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Regularisation

- Regularisation is a set of techniques used in machine learning to prevent overfitting.
- Regularisation methods introduce additional constraints or penalties to the model during training.
- Discouraging overly complex models and promoting simpler ones that are more likely to generalise well.
- Regularisation, significantly reduces the variance of the model, without substantial increase in its bias.
- **Two commonly used regularisation techniques include:**

Dropout:

Where, during training, randomly selected neurons (units) in the neural network are ignored or "dropped out" with a
certain probability.

Weight Decay:

Weight decay, also known as L2 regularisation, involves adding a penalty term to the loss function that discourages large weights in the model.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Regularisation

- Dropout can be carried out as follows:

1. During each training iteration, a random subset of neurons is "dropped out" by setting their outputs to zero.
2. The network is trained on the modified data, effectively creating an ensemble of different sub-networks.
3. During inference (testing or making predictions), all neurons are used.

- Dropout prevents the co-adaptation of neurons and encourages robust representations. It acts as a form of ensemble learning, where the model learns from multiple sub-networks, reducing the risk of overfitting.

When building our model, we can introduce a dropout layer as shown here, with a probability of 0.5.

```
lstm = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=64,
        # Use masking to handle the variable sequence lengths
        mask_zero=True),
    tf.keras.layers.LSTM(64), #Replacing the simple RNN with a LSTM
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1)
])
```


© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Regularisation

- Weight Decay can be carried out as follows:

1. A regularisation term is added to the loss function, penalising large weights.
2. The regularisation term is controlled by a hyperparameter (λ), which determines the strength of the penalty.

- Weight decay discourages the model from relying too much on any single input feature, preventing extreme weight values and promoting a smoother, more generalised model

When building our model, we can introduce weight decay within the layer's hyperparameters.

```
lstm = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=64,
        # Use masking to handle the variable sequence lengths
        mask_zero=True),
    tf.keras.layers.LSTM(64, kernel_regularizer=keras.regularizers.l2(0.01)), #Replacing the simple RNN with
    tf.keras.layers.Dense(1)
])
```


© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

Summary

Key points discussed this week:

- K-Fold cross validation, a resampling technique to assess a model's design suitability.
- Loss Functions, which quantify the 'error' of a model.
- Optimisers, which aim to minimize the loss functions.
- Batch Size, which controls training speed and accuracy.
- Learning Rate, which determines the size of steps taken during optimisation.
- Gradient Clipping, which prevents exploding/vanishing gradients.

© 2023 Nanyang Technological University, Singapore. All Rights Reserved.

No part of this video shall be filmed, recorded, downloaded, reproduced, distributed, republished or transmitted in any form or by any means without written approval from the University.