|  |
| --- |
|  |
| # Examples |
|  |  |
|  | ```python |
|  | from keras\_contrib.layers import CRF |
|  | from keras\_contrib.losses import crf\_loss |
|  | from keras\_contrib.metrics import crf\_viterbi\_accuracy |
|  |  |
|  | model = Sequential() |
|  | model.add(Embedding(3001, 300, mask\_zero=True)(X) |
|  |  |
|  | # use learn\_mode = 'join', test\_mode = 'viterbi', |
|  | # sparse\_target = True (label indice output) |
|  | crf = CRF(10, sparse\_target=True) |
|  | model.add(crf) |
|  |  |
|  | # crf\_accuracy is default to Viterbi acc if using join-mode (default). |
|  | # One can add crf.marginal\_acc if interested, but may slow down learning |
|  | model.compile('adam', loss=crf\_loss, metrics=[crf\_viterbi\_accuracy]) |
|  |  |
|  | # y must be label indices (with shape 1 at dim 3) here, |
|  | # since `sparse\_target=True` |
|  | model.fit(x, y) |
|  |  |
|  | # prediction give onehot representation of Viterbi best path |
|  | y\_hat = model.predict(x\_test) |
|  | ``` |
|  |  |
|  | The following snippet shows how to load a persisted |
|  | model that uses the CRF layer: |
|  |  |
|  | ```python |
|  | from keras.models import load\_model |
|  | from keras\_contrib.losses import import crf\_loss |
|  | from keras\_contrib.metrics import crf\_viterbi\_accuracy |
|  |  |
|  | custom\_objects={'CRF': CRF, |
|  | 'crf\_loss': crf\_loss, |
|  | 'crf\_viterbi\_accuracy': crf\_viterbi\_accuracy} |
|  |  |
|  | loaded\_model = load\_model('<path\_to\_model>', |
|  | custom\_objects=custom\_objects) |
|  | ``` |
|  |  |
|  | # Arguments |
|  | units: Positive integer, dimensionality of the output space. |
|  | learn\_mode: Either 'join' or 'marginal'. |
|  | The former train the model by maximizing join likelihood while the latter |
|  | maximize the product of marginal likelihood over all time steps. |
|  | One should use `losses.crf\_nll` for 'join' mode |
|  | and `losses.categorical\_crossentropy` or |
|  | `losses.sparse\_categorical\_crossentropy` for |
|  | `marginal` mode. For convenience, simply |
|  | use `losses.crf\_loss`, which will decide the proper loss as described. |
|  | test\_mode: Either 'viterbi' or 'marginal'. |
|  | The former is recommended and as default when `learn\_mode = 'join'` and |
|  | gives one-hot representation of the best path at test (prediction) time, |
|  | while the latter is recommended and chosen as default |
|  | when `learn\_mode = 'marginal'`, |
|  | which produces marginal probabilities for each time step. |
|  | For evaluating metrics, one should |
|  | use `metrics.crf\_viterbi\_accuracy` for 'viterbi' mode and |
|  | 'metrics.crf\_marginal\_accuracy' for 'marginal' mode, or |
|  | simply use `metrics.crf\_accuracy` for |
|  | both which automatically decides it as described. |
|  | One can also use both for evaluation at training. |
|  | sparse\_target: Boolean (default False) indicating |
|  | if provided labels are one-hot or |
|  | indices (with shape 1 at dim 3). |
|  | use\_boundary: Boolean (default True) indicating if trainable |
|  | start-end chain energies |
|  | should be added to model. |
|  | use\_bias: Boolean, whether the layer uses a bias vector. |
|  | kernel\_initializer: Initializer for the `kernel` weights matrix, |
|  | used for the linear transformation of the inputs. |
|  | (see [initializers](../initializers.md)). |
|  | chain\_initializer: Initializer for the `chain\_kernel` weights matrix, |
|  | used for the CRF chain energy. |
|  | (see [initializers](../initializers.md)). |
|  | boundary\_initializer: Initializer for the `left\_boundary`, |
|  | 'right\_boundary' weights vectors, |
|  | used for the start/left and end/right boundary energy. |
|  | (see [initializers](../initializers.md)). |
|  | bias\_initializer: Initializer for the bias vector |
|  | (see [initializers](../initializers.md)). |
|  | activation: Activation function to use |
|  | (see [activations](../activations.md)). |
|  | If you pass None, no activation is applied |
|  | (ie. "linear" activation: `a(x) = x`). |
|  | kernel\_regularizer: Regularizer function applied to |
|  | the `kernel` weights matrix |
|  | (see [regularizer](../regularizers.md)). |
|  | chain\_regularizer: Regularizer function applied to |
|  | the `chain\_kernel` weights matrix |
|  | (see [regularizer](../regularizers.md)). |
|  | boundary\_regularizer: Regularizer function applied to |
|  | the 'left\_boundary', 'right\_boundary' weight vectors |
|  | (see [regularizer](../regularizers.md)). |
|  | bias\_regularizer: Regularizer function applied to the bias vector |
|  | (see [regularizer](../regularizers.md)). |
|  | kernel\_constraint: Constraint function applied to |
|  | the `kernel` weights matrix |
|  | (see [constraints](../constraints.md)). |
|  | chain\_constraint: Constraint function applied to |
|  | the `chain\_kernel` weights matrix |
|  | (see [constraints](../constraints.md)). |
|  | boundary\_constraint: Constraint function applied to |
|  | the `left\_boundary`, `right\_boundary` weights vectors |
|  | (see [constraints](../constraints.md)). |
|  | bias\_constraint: Constraint function applied to the bias vector |
|  | (see [constraints](../constraints.md)). |
|  | input\_dim: dimensionality of the input (integer). |
|  | This argument (or alternatively, the keyword argument `input\_shape`) |
|  | is required when using this layer as the first layer in a model. |
|  | unroll: Boolean (default False). If True, the network will be |
|  | unrolled, else a symbolic loop will be used. |
|  | Unrolling can speed-up a RNN, although it tends |
|  | to be more memory-intensive. |
|  | Unrolling is only suitable for short sequences. |
|  |  |
|  | # Input shape |
|  | 3D tensor with shape `(nb\_samples, timesteps, input\_dim)`. |
|  |  |
|  | # Output shape |
|  | 3D tensor with shape `(nb\_samples, timesteps, units)`. |
|  |  |
|  | # Masking |
|  | This layer supports masking for input data with a variable number |
|  | of timesteps. To introduce masks to your data, |
|  | use an [Embedding](embeddings.md) layer with the `mask\_zero` parameter |
|  | set to `True`. |
|  |  |
|  | """ |
|  |  |