// Protocol messages for describing input data Examples for machine learning

// model training or inference.

syntax = "proto3";

package tensorflow;

import "tensorflow/core/example/feature.proto";

option cc\_enable\_arenas = true;

option java\_outer\_classname = "ExampleProtos";

option java\_multiple\_files = true;

option java\_package = "org.tensorflow.example";

option go\_package = "github.com/tensorflow/tensorflow/tensorflow/go/core/example";

// LINT.IfChange

// An Example is a mostly-normalized data format for storing data for

// training and inference. It contains a key-value store (features); where

// each key (string) maps to a Feature message (which is oneof packed BytesList,

// FloatList, or Int64List). This flexible and compact format allows the

// storage of large amounts of typed data, but requires that the data shape

// and use be determined by the configuration files and parsers that are used to

// read and write this format. That is, the Example is mostly \*not\* a

// self-describing format. In TensorFlow, Examples are read in row-major

// format, so any configuration that describes data with rank-2 or above

// should keep this in mind. For example, to store an M x N matrix of Bytes,

// the BytesList must contain M\*N bytes, with M rows of N contiguous values

// each. That is, the BytesList value must store the matrix as:

// .... row 0 .... .... row 1 .... // ........... // ... row M-1 ....

//

// An Example for a movie recommendation application:

// features {

// feature {

// key: "age"

// value { float\_list {

// value: 29.0

// }}

// }

// feature {

// key: "movie"

// value { bytes\_list {

// value: "The Shawshank Redemption"

// value: "Fight Club"

// }}

// }

// feature {

// key: "movie\_ratings"

// value { float\_list {

// value: 9.0

// value: 9.7

// }}

// }

// feature {

// key: "suggestion"

// value { bytes\_list {

// value: "Inception"

// }}

// }

// # Note that this feature exists to be used as a label in training.

// # E.g., if training a logistic regression model to predict purchase

// # probability in our learning tool we would set the label feature to

// # "suggestion\_purchased".

// feature {

// key: "suggestion\_purchased"

// value { float\_list {

// value: 1.0

// }}

// }

// # Similar to "suggestion\_purchased" above this feature exists to be used

// # as a label in training.

// # E.g., if training a linear regression model to predict purchase

// # price in our learning tool we would set the label feature to

// # "purchase\_price".

// feature {

// key: "purchase\_price"

// value { float\_list {

// value: 9.99

// }}

// }

// }

//

// A conformant Example data set obeys the following conventions:

// - If a Feature K exists in one example with data type T, it must be of

// type T in all other examples when present. It may be omitted.

// - The number of instances of Feature K list data may vary across examples,

// depending on the requirements of the model.

// - If a Feature K doesn't exist in an example, a K-specific default will be

// used, if configured.

// - If a Feature K exists in an example but contains no items, the intent

// is considered to be an empty tensor and no default will be used.

message Example {

Features features = 1;

}

// A SequenceExample is an Example representing one or more sequences, and

// some context. The context contains features which apply to the entire

// example. The feature\_lists contain a key, value map where each key is

// associated with a repeated set of Features (a FeatureList).

// A FeatureList thus represents the values of a feature identified by its key

// over time / frames.

//

// Below is a SequenceExample for a movie recommendation application recording a

// sequence of ratings by a user. The time-independent features ("locale",

// "age", "favorites") describing the user are part of the context. The sequence

// of movies the user rated are part of the feature\_lists. For each movie in the

// sequence we have information on its name and actors and the user's rating.

// This information is recorded in three separate feature\_list(s).

// In the example below there are only two movies. All three feature\_list(s),

// namely "movie\_ratings", "movie\_names", and "actors" have a feature value for

// both movies. Note, that "actors" is itself a bytes\_list with multiple

// strings per movie.

//

// context: {

// feature: {

// key : "locale"

// value: {

// bytes\_list: {

// value: [ "pt\_BR" ]

// }

// }

// }

// feature: {

// key : "age"

// value: {

// float\_list: {

// value: [ 19.0 ]

// }

// }

// }

// feature: {

// key : "favorites"

// value: {

// bytes\_list: {

// value: [ "Majesty Rose", "Savannah Outen", "One Direction" ]

// }

// }

// }

// }

// feature\_lists: {

// feature\_list: {

// key : "movie\_ratings"

// value: {

// feature: {

// float\_list: {

// value: [ 4.5 ]

// }

// }

// feature: {

// float\_list: {

// value: [ 5.0 ]

// }

// }

// }

// }

// feature\_list: {

// key : "movie\_names"

// value: {

// feature: {

// bytes\_list: {

// value: [ "The Shawshank Redemption" ]

// }

// }

// feature: {

// bytes\_list: {

// value: [ "Fight Club" ]

// }

// }

// }

// }

// feature\_list: {

// key : "actors"

// value: {

// feature: {

// bytes\_list: {

// value: [ "Tim Robbins", "Morgan Freeman" ]

// }

// }

// feature: {

// bytes\_list: {

// value: [ "Brad Pitt", "Edward Norton", "Helena Bonham Carter" ]

// }

// }

// }

// }

// }

//

// A conformant SequenceExample data set obeys the following conventions:

//

// Context:

// - All conformant context features K must obey the same conventions as

// a conformant Example's features (see above).

// Feature lists:

// - A FeatureList L may be missing in an example; it is up to the

// parser configuration to determine if this is allowed or considered

// an empty list (zero length).

// - If a FeatureList L exists, it may be empty (zero length).

// - If a FeatureList L is non-empty, all features within the FeatureList

// must have the same data type T. Even across SequenceExamples, the type T

// of the FeatureList identified by the same key must be the same. An entry

// without any values may serve as an empty feature.

// - If a FeatureList L is non-empty, it is up to the parser configuration

// to determine if all features within the FeatureList must

// have the same size. The same holds for this FeatureList across multiple

// examples.

// - For sequence modeling, e.g.:

// http://colah.github.io/posts/2015-08-Understanding-LSTMs/

// https://github.com/tensorflow/nmt

// the feature lists represent a sequence of frames.

// In this scenario, all FeatureLists in a SequenceExample have the same

// number of Feature messages, so that the ith element in each FeatureList

// is part of the ith frame (or time step).

// Examples of conformant and non-conformant examples' FeatureLists:

//

// Conformant FeatureLists:

// feature\_lists: { feature\_list: {

// key: "movie\_ratings"

// value: { feature: { float\_list: { value: [ 4.5 ] } }

// feature: { float\_list: { value: [ 5.0 ] } } }

// } }

//

// Non-conformant FeatureLists (mismatched types):

// feature\_lists: { feature\_list: {

// key: "movie\_ratings"

// value: { feature: { float\_list: { value: [ 4.5 ] } }

// feature: { int64\_list: { value: [ 5 ] } } }

// } }

//

// Conditionally conformant FeatureLists, the parser configuration determines

// if the feature sizes must match:

// feature\_lists: { feature\_list: {

// key: "movie\_ratings"

// value: { feature: { float\_list: { value: [ 4.5 ] } }

// feature: { float\_list: { value: [ 5.0, 6.0 ] } } }

// } }

//

// Conformant pair of SequenceExample

// feature\_lists: { feature\_list: {

// key: "movie\_ratings"

// value: { feature: { float\_list: { value: [ 4.5 ] } }

// feature: { float\_list: { value: [ 5.0 ] } } }

// } }

// and:

// feature\_lists: { feature\_list: {

// key: "movie\_ratings"

// value: { feature: { float\_list: { value: [ 4.5 ] } }

// feature: { float\_list: { value: [ 5.0 ] } }

// feature: { float\_list: { value: [ 2.0 ] } } }

// } }

//

// Conformant pair of SequenceExample

// feature\_lists: { feature\_list: {

// key: "movie\_ratings"

// value: { feature: { float\_list: { value: [ 4.5 ] } }

// feature: { float\_list: { value: [ 5.0 ] } } }

// } }

// and:

// feature\_lists: { feature\_list: {

// key: "movie\_ratings"

// value: { }

// } }

//

// Conditionally conformant pair of SequenceExample, the parser configuration

// determines if the second feature\_lists is consistent (zero-length) or

// invalid (missing "movie\_ratings"):

// feature\_lists: { feature\_list: {

// key: "movie\_ratings"

// value: { feature: { float\_list: { value: [ 4.5 ] } }

// feature: { float\_list: { value: [ 5.0 ] } } }

// } }

// and:

// feature\_lists: { }

//

// Non-conformant pair of SequenceExample (mismatched types)

// feature\_lists: { feature\_list: {

// key: "movie\_ratings"

// value: { feature: { float\_list: { value: [ 4.5 ] } }

// feature: { float\_list: { value: [ 5.0 ] } } }

// } }

// and:

// feature\_lists: { feature\_list: {

// key: "movie\_ratings"

// value: { feature: { int64\_list: { value: [ 4 ] } }

// feature: { int64\_list: { value: [ 5 ] } }

// feature: { int64\_list: { value: [ 2 ] } } }

// } }

//

// Conditionally conformant pair of SequenceExample; the parser configuration

// determines if the feature sizes must match:

// feature\_lists: { feature\_list: {

// key: "movie\_ratings"

// value: { feature: { float\_list: { value: [ 4.5 ] } }

// feature: { float\_list: { value: [ 5.0 ] } } }

// } }

// and:

// feature\_lists: { feature\_list: {

// key: "movie\_ratings"

// value: { feature: { float\_list: { value: [ 4.0 ] } }

// feature: { float\_list: { value: [ 5.0, 3.0 ] } }

// } }

message SequenceExample {

Features context = 1;

FeatureLists feature\_lists = 2;

}

// LINT.ThenChange(

// https://www.tensorflow.org/code/tensorflow/python/training/training.py)