Classification

ACTL3143 & ACTL5111 Deep Learning for Actuaries

Patrick Laub

Lecture Outline

- **Example 1: Binary Classification**
- Example 2: Multiclass Classification
- Summary

Stroke Prediction Data description

1. `id`: unique identifier
2. `gender`: “Male”, “Female” or “Other”
3. `age`: age of the patient
4. `hypertension`: 0 or 1 if the patient has hypertension
5. `heart_disease`: 0 or 1 if the patient has any heart disease
6. `ever_married`: “No” or “Yes”
7. `work_type`: “children”, “Govt_jov”, “Never_worked”, “Private” or “Self-employed”
8. `Residence_type`: “Rural” or “Urban”
9. `avg_glucose_level`: average glucose level in blood
10. `bmi`: body mass index
11. `smoking_status`: “formerly smoked”, “never smoked”, “smokes” or “Unknown”
12. `stroke`: 0 or 1 if the patient had a stroke

Source: Kaggle, Stroke Prediction Dataset.

Load up the (pre-)preprocessed data

```
PROCESSED_DATA_DIR = Path("stroke/processed")

X_train = pd.read_csv(PROCESSED_DATA_DIR / "x_train.csv")
X_val= pd.read_csv(PROCESSED_DATA_DIR / "x_val.csv")
X_test = pd.read_csv(PROCESSED_DATA_DIR / "x_test.csv")
y_train = pd.read_csv(PROCESSED_DATA_DIR / "y_train.csv")
y_val = pd.read_csv(PROCESSED_DATA_DIR / "y_val.csv")
y_test = pd.read_csv(PROCESSED_DATA_DIR / "y_test.csv")

X_train
```

	gender_Female	gender_Male	ever_married_No	ever_married_Yes	Residence_type_Rural	Residence_type_U
0	0.0	1.0	0.0	1.0	1.0	0.0
1	0.0	1.0	1.0	0.0	1.0	0.0
2	0.0	1.0	1.0	0.0	1.0	0.0
...	...	...	...	...	...	...
3063	1.0	0.0	0.0	1.0	1.0	0.0
3064	1.0	0.0	0.0	1.0	1.0	0.0
3065	0.0	1.0	1.0	0.0	0.0	1.0

3066 rows × 20 columns

Target variable

```
1 y_train
```

	stroke
0	0
1	0
2	0
...	...
3063	0
3064	0
3065	0

3066 rows × 1 columns

```
1 import numpy as np
2 classes, counts = np.unique(y_train.values.ravel(), return_counts=True)
3 print("Classes:", classes)
4 print("Counts:", counts)
```

```
Classes: [0 1]
Counts: [2909  157]
```

Setup a binary classification model

```python
def create_model(seed=42):
    random.seed(seed)
    model = Sequential()
    model.add(Input(X_train.shape[1:]))
    model.add(Dense(32, "leaky_relu"))
    model.add(Dense(16, "leaky_relu"))
    model.add(Dense(1, "sigmoid"))
    return model
```

```python
model = create_model()
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	672
dense_1 (Dense)	(None, 16)	528
dense_2 (Dense)	(None, 1)	17

Total params: 1,217 (4.75 KB)
Trainable params: 1,217 (4.75 KB)
Non-trainable params: 0 (0.00 B)

Fit the model

```
model = create_model()
model.compile("adam", "binary_crossentropy")
model.fit(X_train, y_train, epochs=5, verbose=2)
```

```
Epoch 1/5
96/96 - 0s - 3ms/step - loss: 0.2734
Epoch 2/5
96/96 - 0s - 2ms/step - loss: 0.1753
Epoch 3/5
96/96 - 0s - 2ms/step - loss: 0.1665
Epoch 4/5
96/96 - 0s - 3ms/step - loss: 0.1619
Epoch 5/5
96/96 - 0s - 2ms/step - loss: 0.1595
```

```
<keras.src.callbacks.history.History at 0×12359acf0>
```

Track accuracy as the model trains

```
model = create_model()
model.compile("adam", "binary_crossentropy", metrics=["accuracy"])
model.fit(X_train, y_train, epochs=5, verbose=2)
```

```
Epoch 1/5
96/96 - 0s - 2ms/step - accuracy: 0.9204 - loss: 0.2711
Epoch 2/5
96/96 - 0s - 3ms/step - accuracy: 0.9488 - loss: 0.1766
Epoch 3/5
96/96 - 0s - 3ms/step - accuracy: 0.9488 - loss: 0.1667
Epoch 4/5
96/96 - 0s - 3ms/step - accuracy: 0.9488 - loss: 0.1623
Epoch 5/5
96/96 - 0s - 3ms/step - accuracy: 0.9488 - loss: 0.1595
```

```
<keras.src.callbacks.history.History at 0×12186a0d0>
```

Run a long fit

```
model = create_model()
model.compile("adam", "binary_crossentropy", metrics=["accuracy"])
%time hist = model.fit(X_train, y_train, epochs=500, validation_data=(X_val, y_val), verb
```

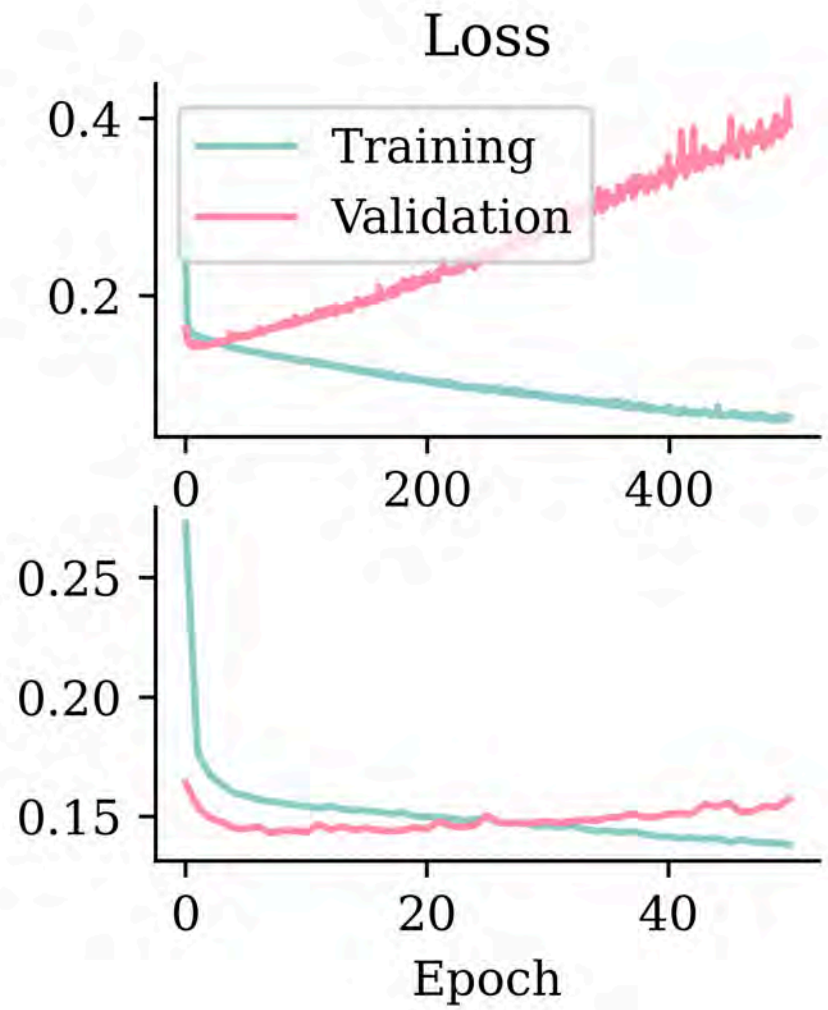
```
CPU times: user 2min 31s, sys: 3.46 s, total: 2min 34s
Wall time: 2min 31s
```

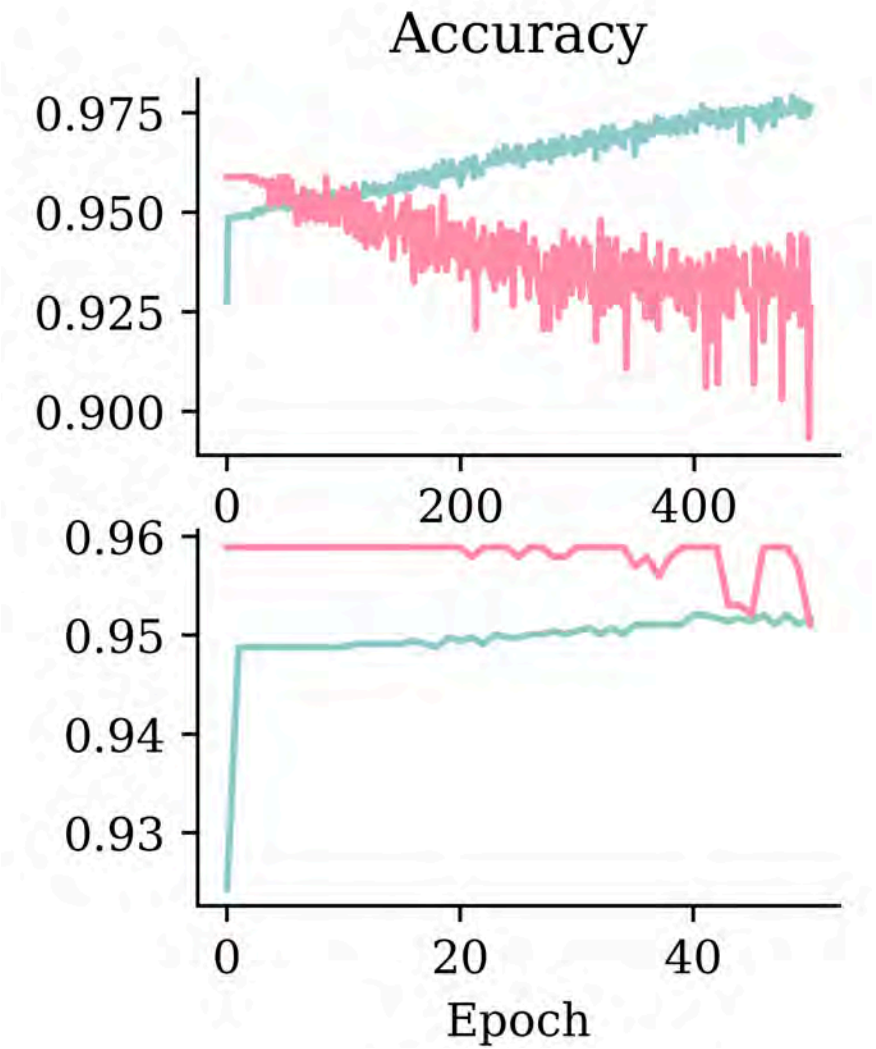
Add early stopping

```
model = create_model()
model.compile("adam", "binary_crossentropy", metrics=["accuracy"])
es = EarlyStopping(restore_best_weights=True, patience=50, monitor="val_accuracy")
%time hist_es = model.fit(X_train, y_train, epochs=500, validation_data=(X_val, y_val), c
print(f"Stopped after {len(hist_es.history['loss'])} epochs.")
```

```
CPU times: user 15.4 s, sys: 347 ms, total: 15.7 s
Wall time: 15.4 s
Stopped after 51 epochs.
```

Fitting metrics





Add metrics, compile, and fit

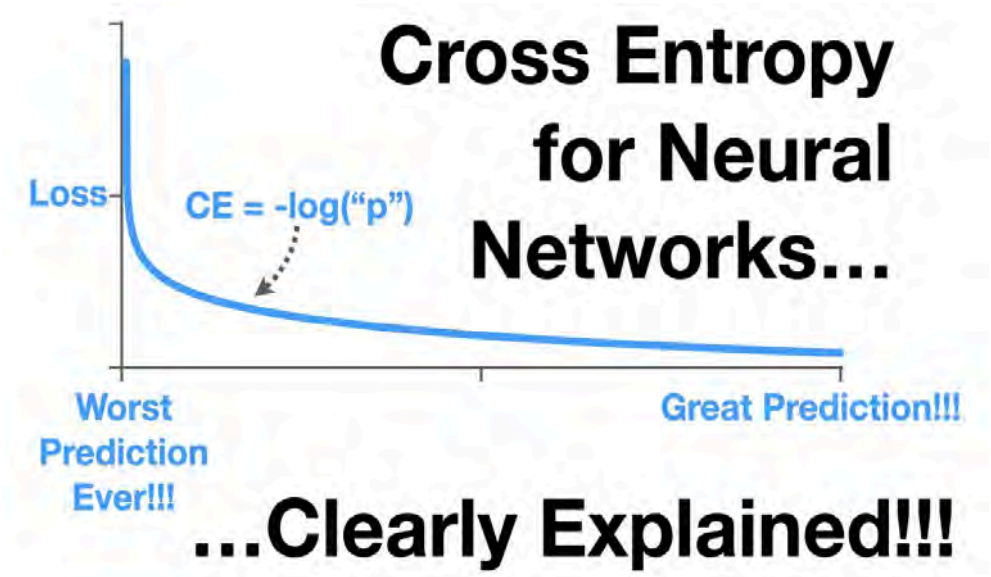
```
model = create_model()

pr_auc = keras.metrics.AUC(curve="PR", name="pr_auc")
model.compile(optimizer="adam", loss="binary_crossentropy",
    metrics=[pr_auc, "accuracy", "auc"])

es = EarlyStopping(patience=50, restore_best_weights=True,
    monitor="val_pr_auc", verbose=1)
model.fit(X_train, y_train, callbacks=[es], epochs=1_000, verbose=0,
  validation_data=(X_val, y_val));
```

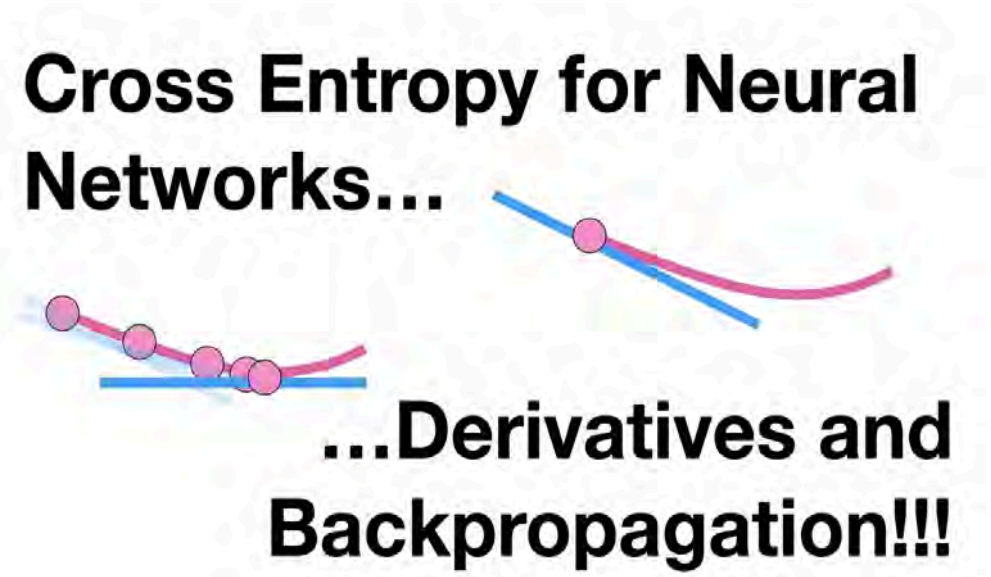
```
Epoch 81: early stopping
Restoring model weights from the end of the best epoch: 31.
```

```
model.evaluate(X_val, y_val, verbose=0)
```

```
[0.14898666739463806,
 0.12857568264007568,
 0.9569471478462219,
 0.8119411468505859]
```

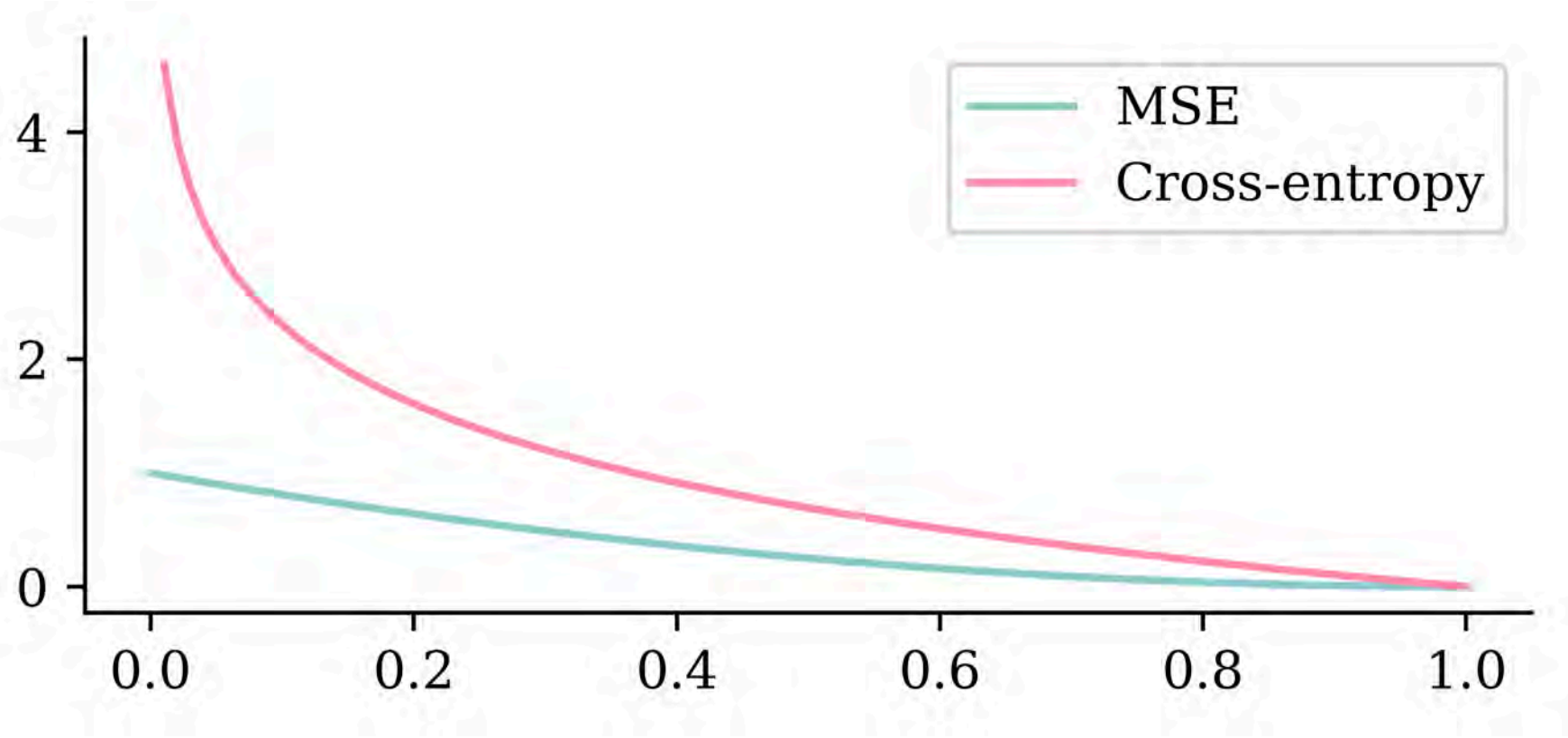
Cross-entropy loss: ELI5





Why use cross-entropy loss?

```
1  p = np.linspace(0, 1, 100)
2  plt.plot(p, (1 - p) ** 2)
3  plt.plot(p, -np.log(p))
4  plt.legend(["MSE", "Cross-entropy"]);
```



Overweight the minority class

```
model = create_model()

pr_auc = keras.metrics.AUC(curve="PR", name="pr_auc")
model.compile(optimizer="adam", loss="binary_crossentropy",
    metrics=[pr_auc, "accuracy", "auc"])

es = EarlyStopping(patience=50, restore_best_weights=True,
    monitor="val_pr_auc", verbose=1)
model.fit(X_train, y_train.to_numpy(), callbacks=[es], epochs=1_000, verbose=0,
  validation_data=(X_val, y_val), class_weight={0: 1, 1: 10});
```

```
Epoch 64: early stopping
Restoring model weights from the end of the best epoch: 14.
```

```
model.evaluate(X_val, y_val, verbose=0)
```

```
[0.3523019552230835,
 0.13380154967308044,
 0.7896282076835632,
 0.8259596824645996]
```

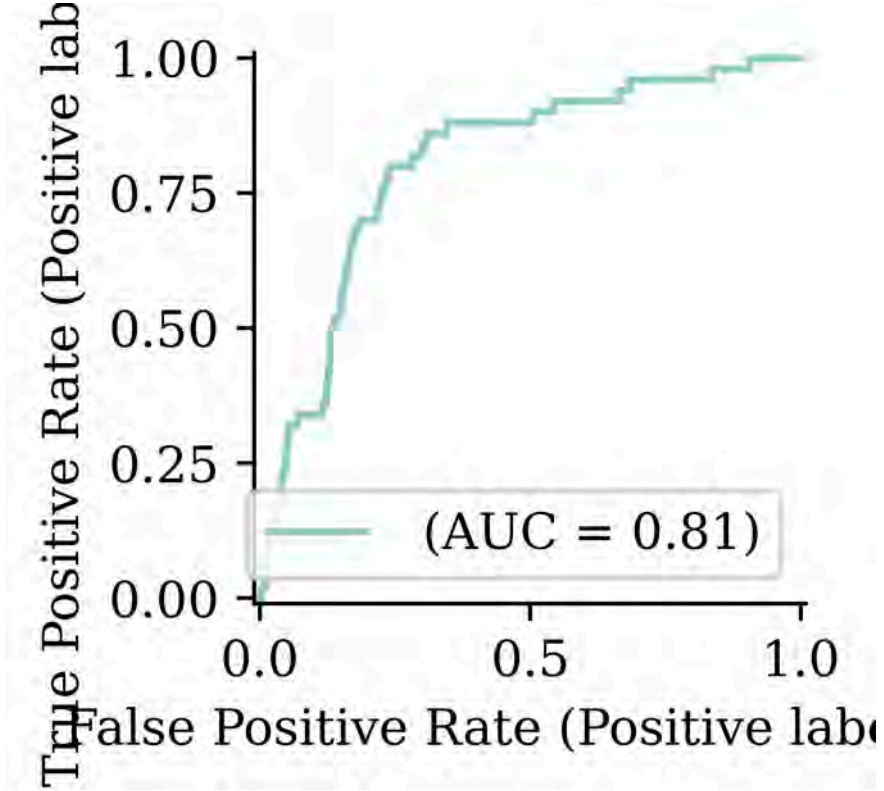
```
model.evaluate(X_test, y_test, verbose=0)
```

```
[0.36996063590049744,
 0.15842117369174957,
 0.7954990267753601,
 0.8060390949249268]
```

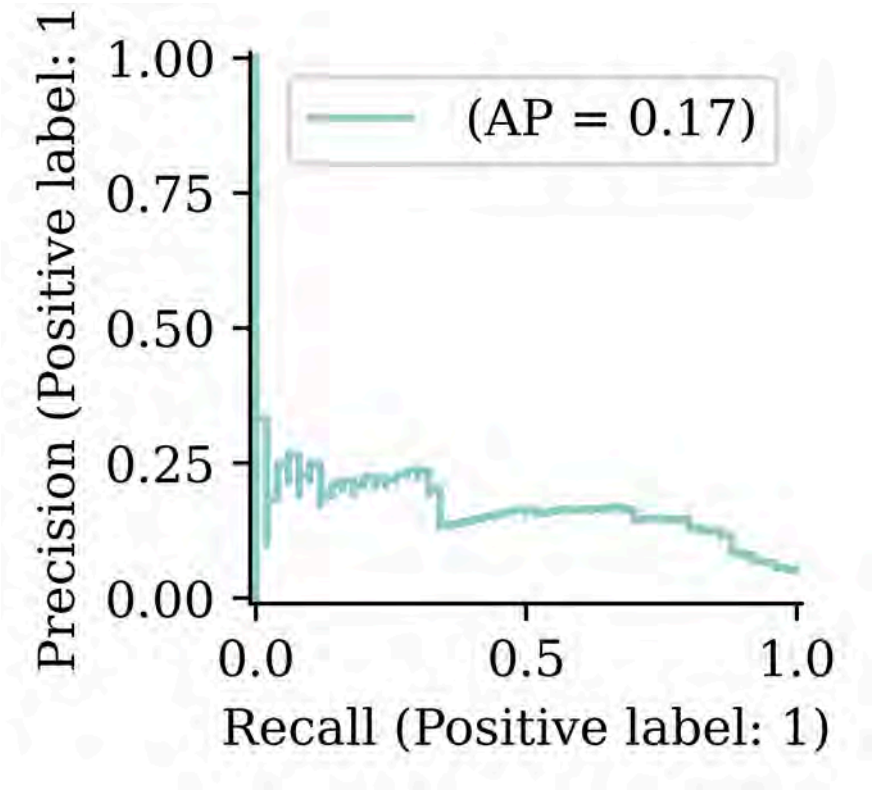
Classification Metrics

```
from sklearn.metrics import confusion_matrix, RocCurveDisplay, PrecisionRecallDisplay
y_pred = model.predict(X_test, verbose=0)
```

```
RocCurveDisplay.from_predictions(y_test, y_pred, name="");
```




```
PrecisionRecallDisplay.from_predictions(y_test, y_pred, name
```




```
y_pred_stroke = y_pred > 0.5
confusion_matrix(y_test, y_pred_stroke)
```

```
array([[778, 194],
```

```
y_pred_stroke = y_pred > 0.3
confusion_matrix(y_test, y_pred_stroke)
```

```
array([[647, 325],
```

UNSW SYDNEY

Lecture Outline

- Example 1: Binary Classification
- **Example 2: Multiclass Classification**
- Summary

Iris dataset

```
1 from sklearn.datasets import load_iris
2 iris = load_iris()
3 names = ["SepalLength", "SepalWidth", "PetalLength", "PetalWidth"]
4 features = pd.DataFrame(iris.data, columns=names)
5 features
```

	SepalLength	SepalWidth	PetalLength	PetalWidth
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
...	...	...	...	...
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Target variable

```
1 iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'],
dtype='<U10')
```

```
1 iris.target[:8]
```

```
array([0, 0, 0, 0, 0, 0, 0, 0])
```

```
1 target = iris.target
2 target = target.reshape(-1, 1)
3 target[:8]
```

```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0]])
```

```
1 classes, counts = np.unique(
2         target,
3         return_counts=True
4 )
5 print(classes)
6 print(counts)
```

```
[0 1 2]
[50 50 50]
```

```
1 iris.target_names[
2   target[[0, 30, 60]]
3 ]
```

```
array([['setosa'],
       ['setosa'],
       ['versicolor']], dtype='<U10')
```

UNSW SYDNEY

Split the data into train and test

```
1 X_train, X_test, y_train, y_test = train_test_split(features, target, random_state=24)
2 X_train
```

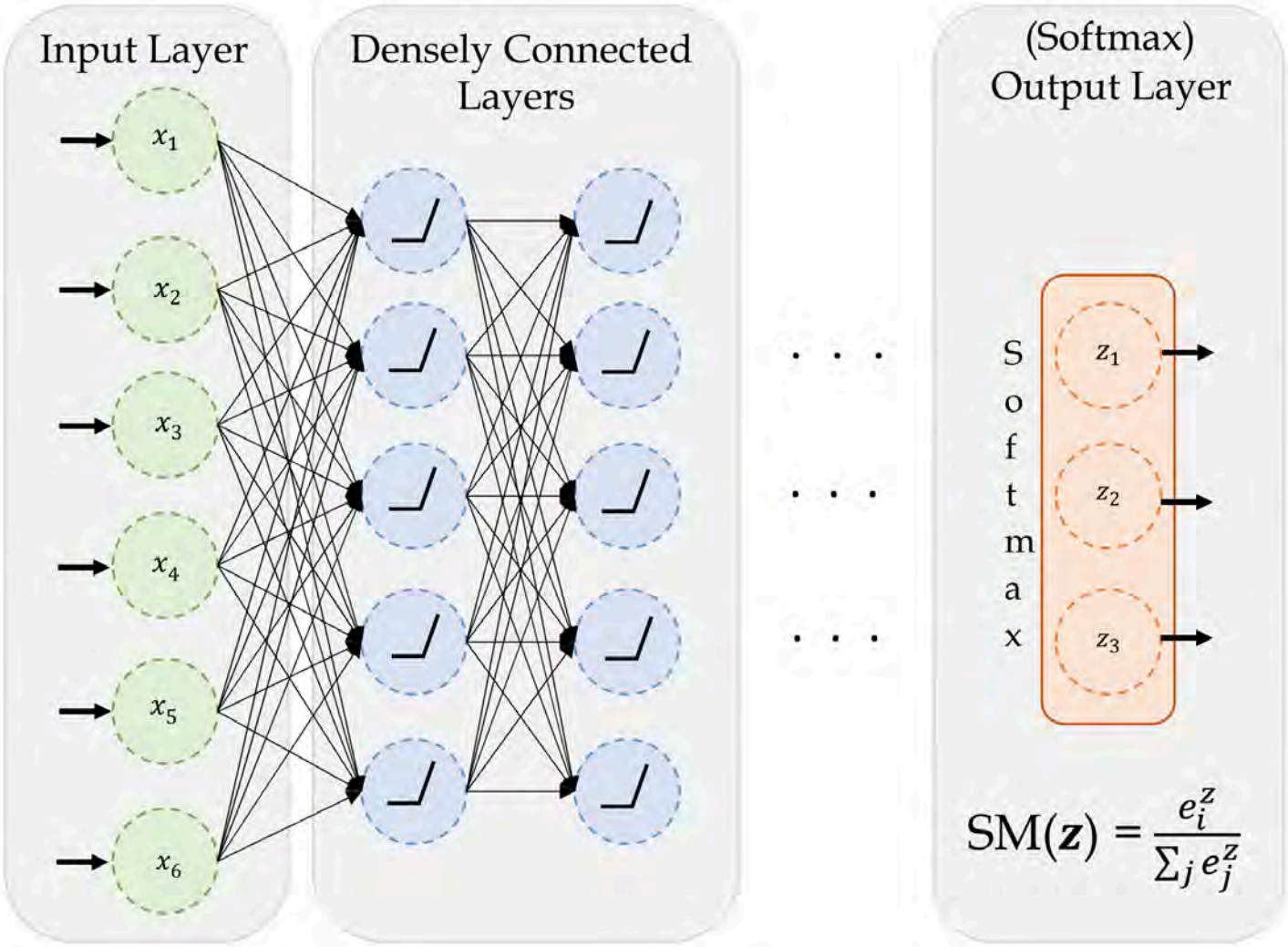
	SepalLength	SepalWidth	PetalLength	PetalWidth
53	5.5	2.3	4.0	1.3
58	6.6	2.9	4.6	1.3
95	5.7	3.0	4.2	1.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
87	6.3	2.3	4.4	1.3
131	7.9	3.8	6.4	2.0

112 rows × 4 columns

```
1 X_test.shape, y_test.shape
```

```
((38, 4), (38, 1))
```

A basic classifier network



A basic network for classifying into three categories.

Source: Marcus Lautier (2022).

Create a classifier model

```
NUM_FEATURES = len(features.columns)
NUM_CATS = len(np.unique(target))

print("Number of features:", NUM_FEATURES)
print("Number of categories:", NUM_CATS)
```

```
Number of features: 4
Number of categories: 3
```

Make a function to return a Keras model:

```
def build_model(seed=42):
    random.seed(seed)
    return Sequential([
        Dense(30, activation="relu"),
        Dense(NUM_CATS, activation="softmax")
    ])
```

Fit the model

```
1 model = build_model()
2 model.compile("adam", "sparse_categorical_crossentropy")
3
4 model.fit(X_train, y_train, epochs=5, verbose=2);
```

```
Epoch 1/5
4/4 - 0s - 2ms/step - loss: 1.3920
Epoch 2/5
4/4 - 0s - 2ms/step - loss: 1.2912
Epoch 3/5
4/4 - 0s - 2ms/step - loss: 1.2196
Epoch 4/5
4/4 - 0s - 2ms/step - loss: 1.1576
Epoch 5/5
4/4 - 0s - 2ms/step - loss: 1.1084
```

Track accuracy as the model trains

```
model = build_model()
model.compile("adam", "sparse_categorical_crossentropy", metrics=["accuracy"])
model.fit(X_train, y_train, epochs=5, verbose=2);
```

```
Epoch 1/5
4/4 - 0s - 2ms/step - accuracy: 0.2857 - loss: 1.3930
Epoch 2/5
4/4 - 0s - 2ms/step - accuracy: 0.2857 - loss: 1.2970
Epoch 3/5
4/4 - 0s - 2ms/step - accuracy: 0.2857 - loss: 1.2203
Epoch 4/5
4/4 - 0s - 2ms/step - accuracy: 0.2946 - loss: 1.1596
Epoch 5/5
4/4 - 0s - 2ms/step - accuracy: 0.3393 - loss: 1.1067
```

Run a long fit

```
model = build_model()
model.compile("adam", "sparse_categorical_crossentropy", \
        metrics=["accuracy"])
%time hist = model.fit(X_train, y_train, epochs=500, \
        validation_split=0.25, verbose=False)
```

```
CPU times: user 5.19 s, sys: 495 ms, total: 5.68 s
Wall time: 5.36 s
```

Evaluation now returns both *loss* and *accuracy*.

```
model.evaluate(X_test, y_test, verbose=False)
```

```
[0.08639740198850632, 0.9736841917037964]
```

Add early stopping

```
model_es = build_model()
model_es.compile("adam", "sparse_categorical_crossentropy", \
        metrics=["accuracy"])

es = EarlyStopping(restore_best_weights=True, patience=50,
        monitor="val_accuracy")
%time hist_es = model_es.fit(X_train, y_train, epochs=500, \
        validation_split=0.25, callbacks=[es], verbose=False);

print(f"Stopped after {len(hist_es.history['loss'])} epochs.")
```

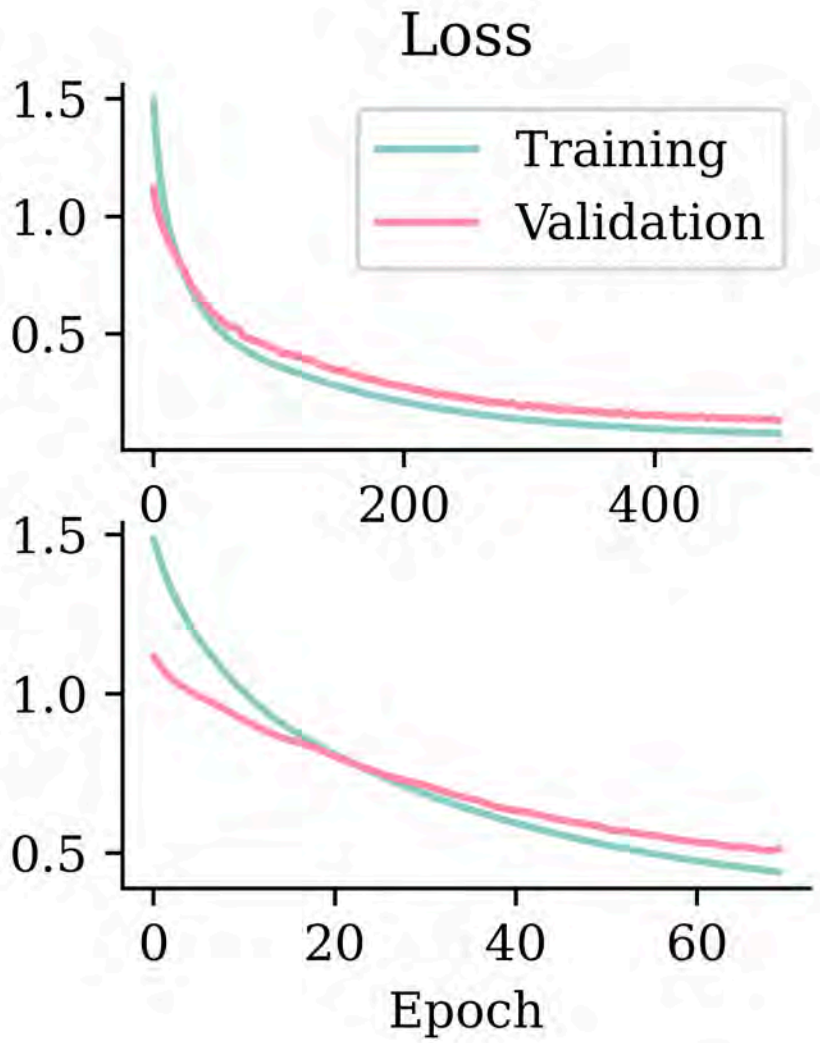
```
CPU times: user 748 ms, sys: 69 ms, total: 817 ms
Wall time: 772 ms
Stopped after 70 epochs.
```

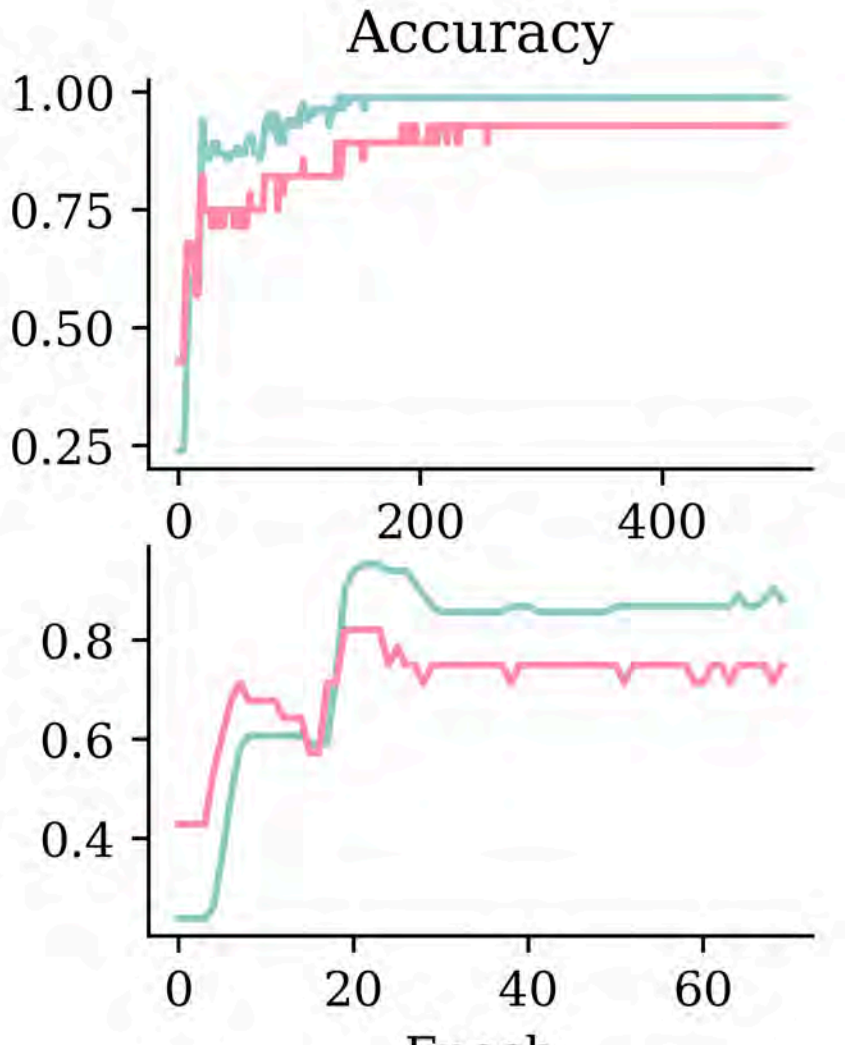
Evaluation on test set:

```
model_es.evaluate(X_test, y_test, verbose=False)
```

```
[0.8077937960624695, 0.9210526347160339]
```

Fitting metrics





What is the softmax activation?

It creates a "probability" vector: $\text{Softmax}(\boldsymbol{x}) = \frac{\mathrm{e}_i^x}{\sum_j \mathrm{e}_j^x}$.

In NumPy:

```
1 out = np.array([5, -1, 6])
2 (np.exp(out) / np.exp(out).sum()).round(3)
```

```
array([0.27, 0.  , 0.73])
```

In Keras:

```
1 out = keras.ops.convert_to_tensor([[5.0, -1.0, 6.0]])
2 keras.ops.round(keras.ops.softmax(out), 3)
```

```
tensor([[0.2690, 0.0010, 0.7310]])
```

Prediction using classifiers

```
y_test[:4]
```

```
array([[2],
       [2],
       [1],
       [1]])
```

```
y_pred = model.predict(X_test.head(4), verbose=0)
y_pred
```

```
array([[2.02e-06, 7.64e-02, 9.24e-01],
       [1.86e-07, 1.62e-03, 9.98e-01],
       [1.44e-02, 9.76e-01, 1.00e-02],
       [2.80e-03, 8.50e-01, 1.48e-01]], dtype=float32)
```

```
# Add 'keepdims=True' to get a column vector.
np.argmax(y_pred, axis=1)
```

```
array([2, 2, 1, 1])
```

```
iris.target_names[np.argmax(y_pred, axis=1)]
```

```
array(['virginica', 'virginica', 'versicolor', 'versicolor'], dtype='<U10')
```

Lecture Outline

- Example 1: Binary Classification
- Example 2: Multiclass Classification
- **Summary**

Classification models in Keras

If the number of classes is c, then:

Target	**Output Layer**	**Loss Function**
Binary ($c = 2$)	1 neuron with `sigmoid` activation	Binary Cross-Entropy
Multi-class ($c > 2$)	c neurons with `softmax` activation	Categorical Cross-Entropy

Optionally output logits

If the number of classes is c, then:

Target	**Output Layer**	**Loss Function**
Binary ($c = 2$)	1 neuron with `linear` activation	Binary Cross-Entropy (`from_logits=True`)
Multi-class ($c > 2$)	c neurons with `linear` activation	Categorical Cross-Entropy (`from_logits=True`)

Code examples

Binary

```
model = Sequential([
  # Skipping the earlier layers
  Dense(1, activation="sigmoid")
])
model.compile(loss="binary_crossentropy")
```

Multi-class

```
model = Sequential([
  # Skipping the earlier layers
  Dense(n_classes, activation="softmax")
])
model.compile(loss="sparse_categorical_crossentropy")
```

Binary (logits)

```
from keras.losses import BinaryCrossentropy
model = Sequential([
  # Skipping the earlier layers
  Dense(1, activation="linear")
])
loss = BinaryCrossentropy(from_logits=True)
model.compile(loss=loss)
```

Multi-class (logits)

```
from keras.losses import SparseCategoricalCrossentropy

model = Sequential([
  # Skipping the earlier layers
  Dense(n_classes, activation="linear")
])
loss = SparseCategoricalCrossentropy(from_logits=True)
model.compile(loss=loss)
```

Package Versions

```
1 from watermark import watermark
2 print(watermark(python=True, packages="keras,matplotlib,numpy,pandas,seaborn,scipy,torch"
```

```
Python implementation: CPython
Python version       : 3.14.3
IPython version      : 9.13.0

keras     : 3.14.1
matplotlib: 3.10.9
numpy     : 2.4.4
pandas    : 3.0.2
seaborn   : 0.13.2
scipy     : 1.17.1
torch     : 2.11.0
```

Glossary

- accuracy
- classification problem
- confusion matrix
- cross-entropy loss
- metrics
- sigmoid activation function
- softmax activation