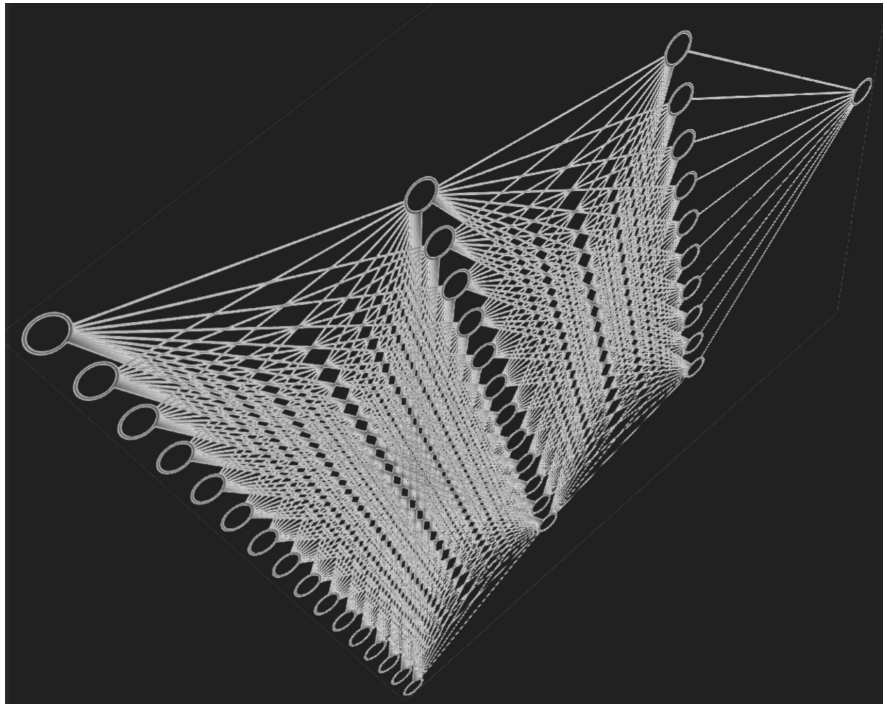
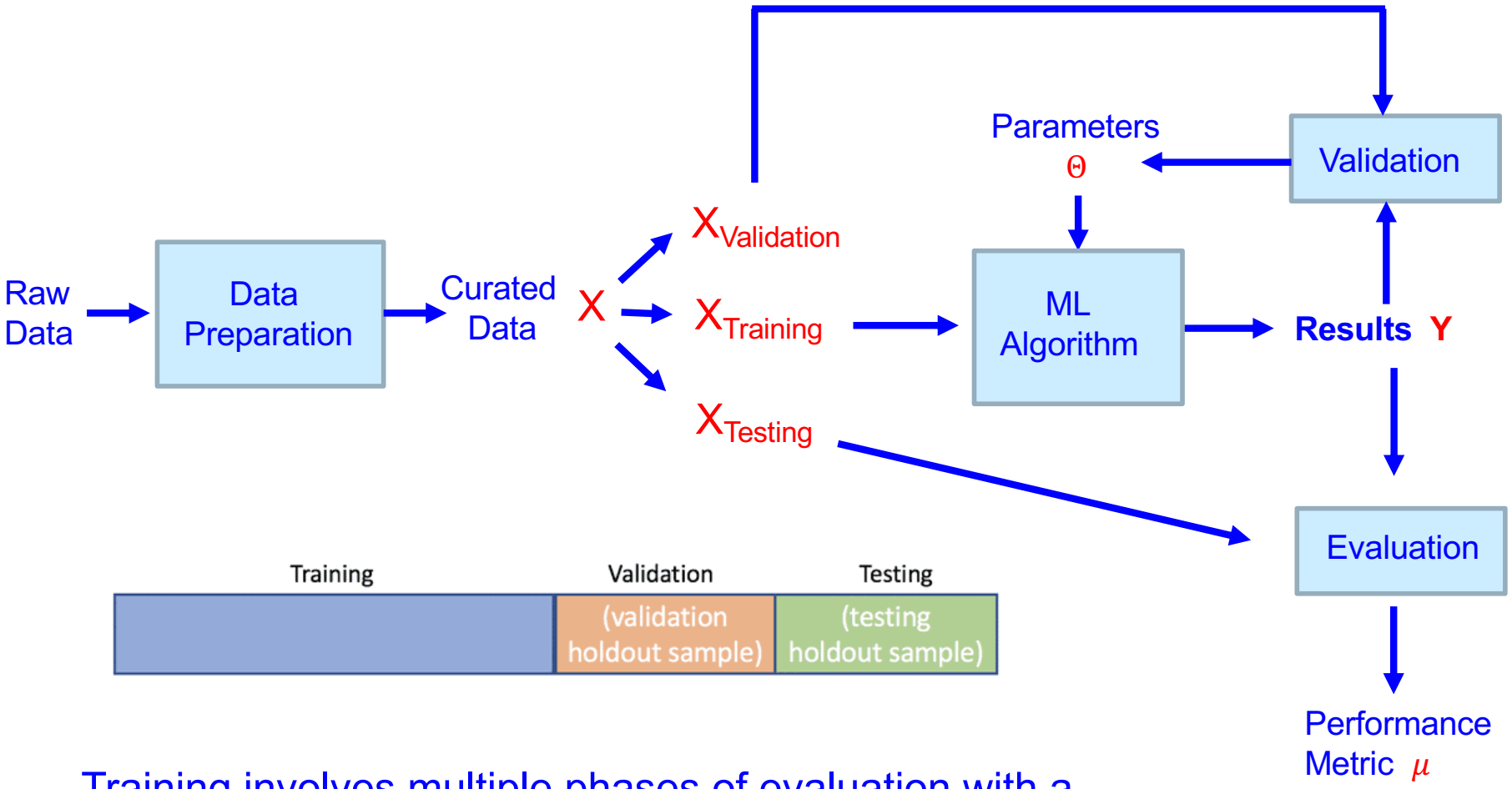
CS 505: Introduction to Natural Language Processing

Wayne Snyder
Boston University

Lecture 12: **Deep Learning:** Evaluating Classifiers; Generalization



Recall: Supervised Machine Learning Workflow

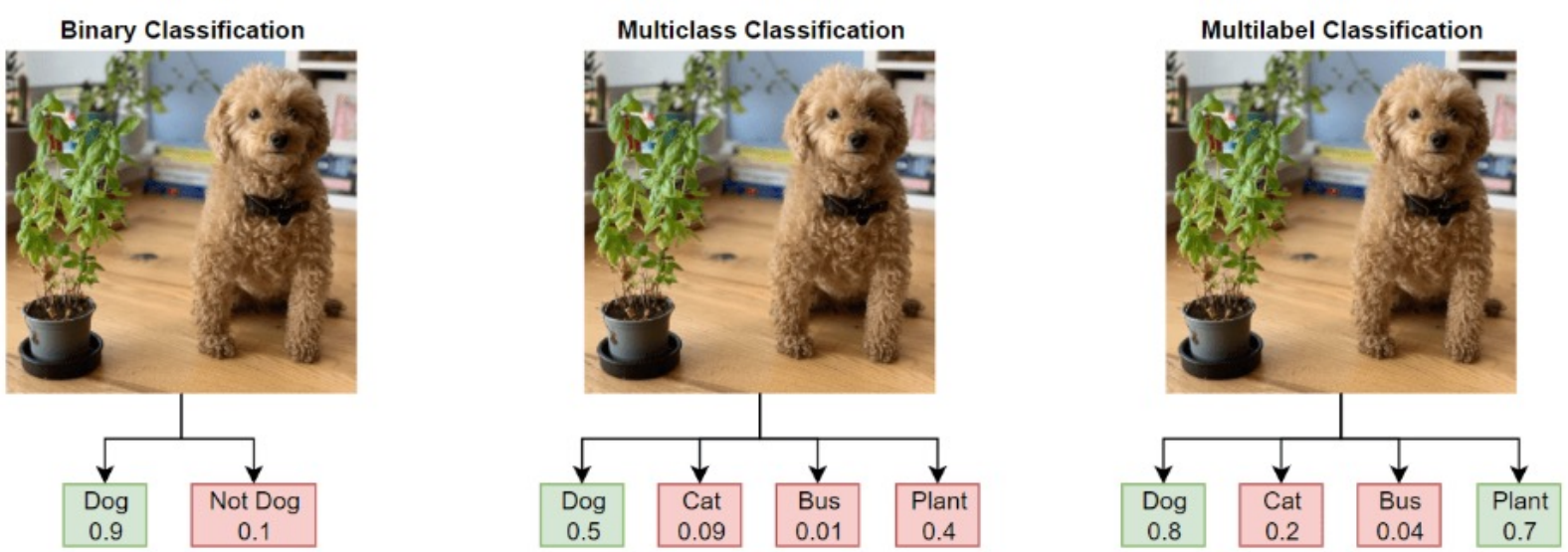


Training involves multiple phases of evaluation with a validation set to find optimal values for the hyperparameters.

Multiclass and Multilabel Classification

In Multiclass Classification, we have more than 2 labels, and our task is to assign a single label to each sample:

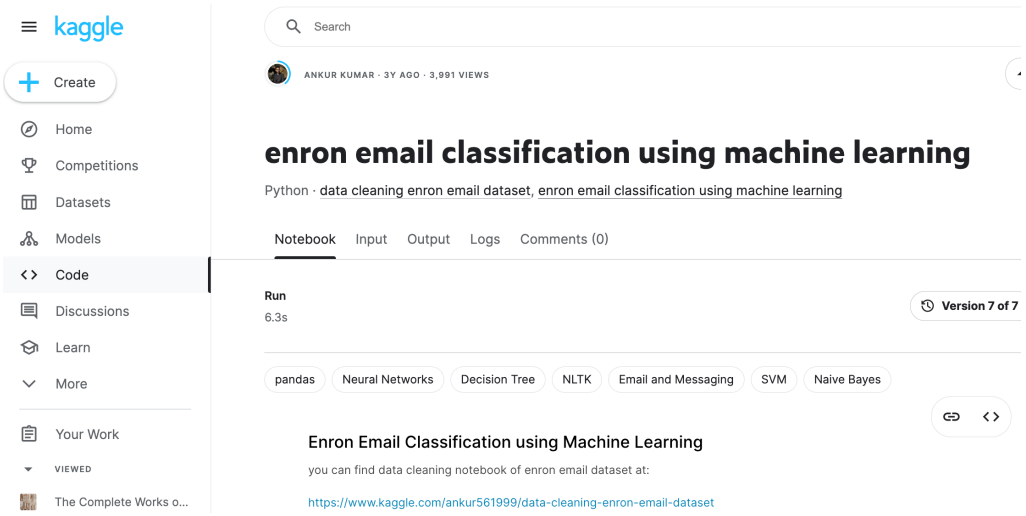
In Multilabel Classification, we have more than 2 labels, and our task is to assign any appropriate labels (not just one):



Let's look at an example of multiclass classification, using the MNIST handwritten digits database....

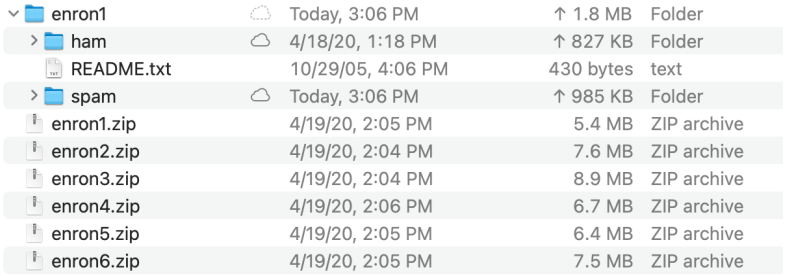
Binary Classification

Example: Spam or Not Spam?



Dataset	N°of Legitimate	N°of Spam	Total
Enron 1	3,672	1,500	5,172
Enron 2	4,361	1,496	5,857
Enron 3	4,012	1,500	5,512
Enron 4	1,500	4,500	6,000
Enron 5	1,500	3,675	5,175
Enron 6	1,500	4,500	6,000
Total	**16,545**	**17,171**	**33,716**

Labels: **1 = Spam** **0 = Not Spam**

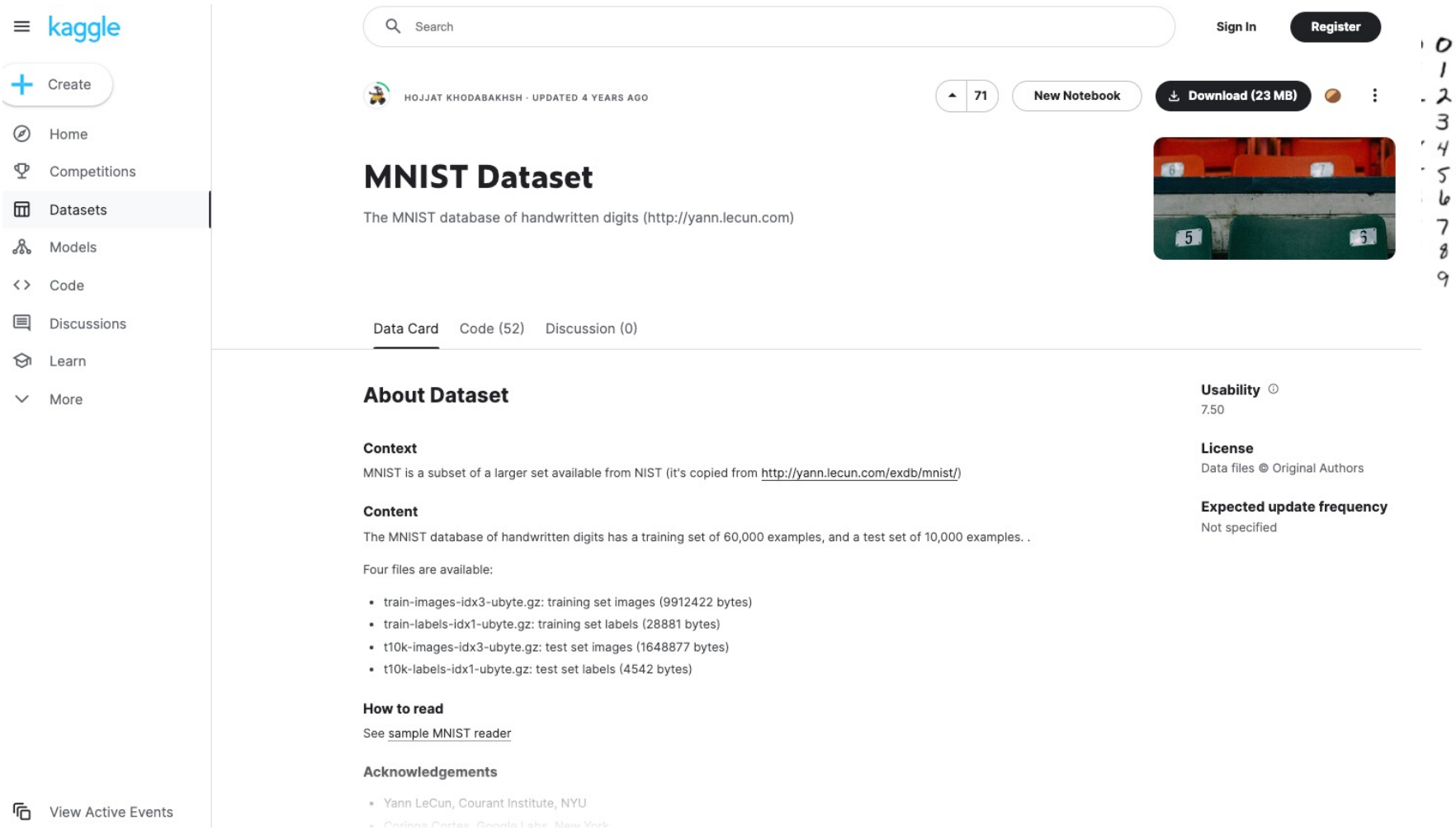


Notice that this data set is fairly well balanced:

51% Spam vs. 49% Not Spam

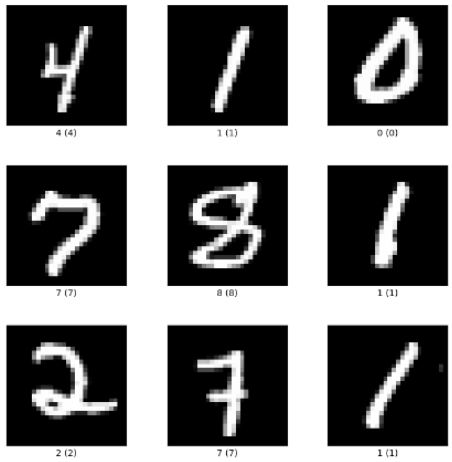
Multiclass Classification

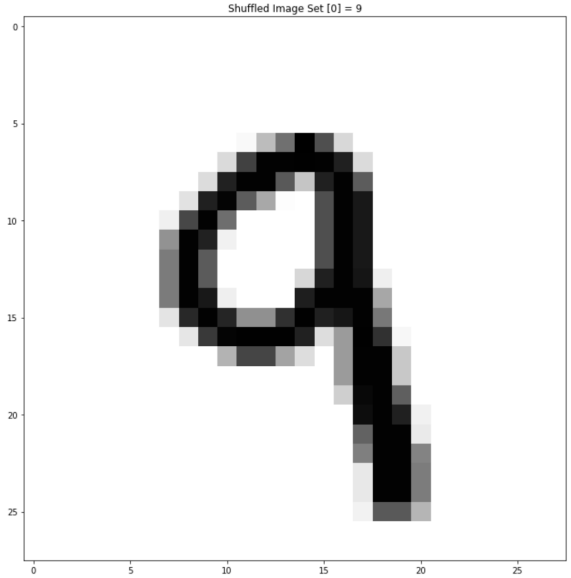
The well-known MNIST dataset of handwritten digits is a classic multiclass problem.



Multiclass Classification

The MNIST digit database consists of 70,000 28x28 BW pixel images, stored as 28*28=784 floating-point numbers in the range [0..1]; labels are integers 0..9:






```
In [33]: print(training_images[0])
```

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.05882353 0.08235294 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.05882353
 0.54901963 0.49019608 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.16862746 0.98039216 0.49019608
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.7176471  0.98039216 0.49019608 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.69411767 0.9882353
 0.98039216 0.49019608 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.08627451 0.7176471  0.9882353  0.99607843 0.93333334 0.4117647
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.11372549 0.65882355 0.98039216
 0.98039216 0.93333334 0.43529412 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.08627451 0.65882355 0.98039216 0.98039216 0.98039216 0.4117647
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.7176471  0.98039216
 0.98039216 0.98039216 0.98039216 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.9882353  0.98039216 0.98039216 0.98039216
 0.98039216 0.         0.         0.         0.         0.
 0.         0.         0.41960785 0.27450982 0.         0.
 0.         0.         0.         0.         0.         0.
```

Etc.

Note that the array is sparse, it is mostly 0's.

```
In [36]: print(train_labels[:100])
```

```
[5 0 4 1 9 2 1 3 1 4 3 5 3 6 1 7 2 8 6 9 4 0 9 1 1 2 4 3 2 7 3 8 6 9 0 5 6
 0 7 6 1 8 7 9 3 9 8 5 9 3 3 0 7 4 9 8 0 9 4 1 4 4 6 0 4 5 6 1 0 0 1 7 1 6
 3 0 2 1 1 7 9 0 2 6 7 8 3 9 0 4 6 7 4 6 8 0 7 8 3 1]
```

Multilabel Classification

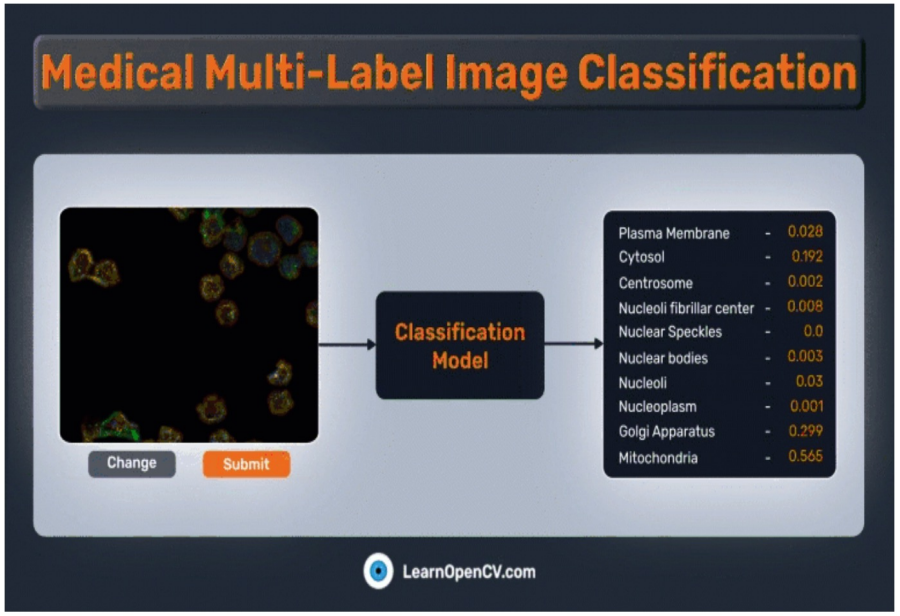
Multilabel classification is a very important problem in medical diagnosis and object recognition in images:

Enhancing Medical Multi-Label Image Classification Using PyTorch & Lightning

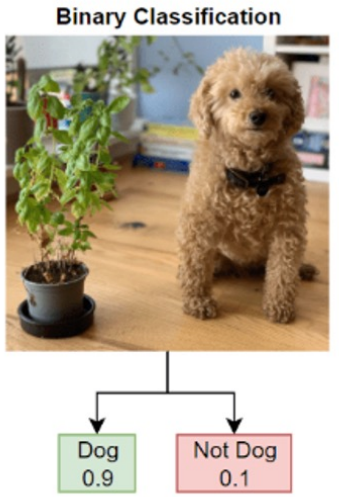
Vaibhav Singh

JULY 11, 2023 — LEAVE A COMMENT

CNN | Deep Learning | Image Classification | PyTorch | PyTorch-Lightning



Evaluation of Classifiers: "It's complicated!"



Precise evaluation of classifier performance is complex, even for the simplest case of binary classification:

We have four different outcomes for predictions:

True Label	Prediction	Type of Outcome	
Dog	Dog	True Positive (TP)	
Not Dog	Not Dog	True Negative (TN)	
Dog	Not Dog	False Negative (FN)	[Type II error]
Not Dog	Dog	False Positive (FP)	[Type I error]

		Predicted Class: Positive	Predicted Class: Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

Intuitively, these make good sense:

- **Accuracy** is the percentage of *all* predictions which were correct;
- **Precision** is the percentage of *positive* predictions which were correct;
- **Sensitivity** (or Recall or "true positive rate") is the percentage of the actual positives identified correctly; and
- **Specificity** (or "true negative rate") is the percentage of the actual negatives identified correctly.

Evaluation of Binary Classifiers (still complicated!)

The idea of accuracy is fairly straight-forward: how many did it get correct? This is the most common in NLP.

However, the others are useful in many circumstances where the cost of errors may be unacceptable:

		Predicted Class: Positive	Predicted Class: Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	**Sensitivity** $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

- If positive = the patient has cancer, then a FN is disasterous, since you have missed diagnosing a deadly disease, so we want to increase *sensitivity*;
- If positive = the patient does NOT have cancer, then a FP is disasterous, since again you have missed diagnosing a deadly disease, so we want to increase *precision*;
- If you are querying a database of documents, and positive = document retrieved, and you want to retrieve all possible documents, then you need to increase *recall* (hence the name);

Precision and recall often in conflict: increasing one will decrease the other; therefore a composite measure is often used, the "harmonic mean" of prediction and recall, which attempts to equalize the error between these two:

$$F_1 \text{ score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

For a very complete listing of metrics for confusion matrices, see:
https://en.wikipedia.org/wiki/Precision_and_recall#Definition_(classification_context)

Evaluation of Binary Classifiers (still complicated!)

But even accuracy is not that simple!

Consider two classifiers with accuracy metrics as follows:

A. Accuracy of 4-way classification of blobs: 90%

B. Accuracy of 10-way classification of digits: 85%

Which is the more accurate classifier?

Evaluation of Binary Classifiers (still complicated!)

Consider two classifiers with accuracy metrics as follows:

A. Accuracy of 4-way classification of blobs: 90%

B. Accuracy of 10-way classification of digits: 85%

Which is the more accurate classifier?

Answer 1: 90% > 85% **A is more accurate**

But wait! A baseline classifier which chooses randomly gets

25% accuracy for A and 10% accuracy for B !!

[Assuming equal distribution of classes – else it is the percentage of biggest class.]

Punchline:

Always compare your model with a suitable baseline model,
for example a random model which always chooses the largest class.

Alternately: Your baseline model is your model **before training**, assuming parameters are initialized randomly.

Evaluation of Binary Classifiers (still complicated!)

Example: Suppose you get the following results:

A. Accuracy of 4-way classification of blobs: 90%
B. Accuracy of 10-way classification of digits: 85%

Ok, how to precisely compare a baseline model?

Answer 2 (most common in ML): Measure improvement above baseline:

A improves on random (25%) by +65%;
B improves on random (10%) by +75%, so B is a better model.

Answer 3: Measure improvement normalized to inaccuracy of baseline model:

Cohen's Kappa: $$\kappa = \frac{\text{accuracy} - \text{baseline accuracy}}{1.0 - \text{baseline accuracy}}$$

Now A wins: $$\kappa(A) = \frac{0.9 - 0.25}{1.0 - 0.25} = 0.87 \qquad \kappa(B) = \frac{0.85 - 0.1}{1.0 - 0.1} = 0.83$$

Evaluation of Binary Classifiers (still complicated!)

Which is correct? Well, "it depends"! Statisticians have been arguing about the right way to do this for at least a century, here is a recent paper which compares the principal methods:

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 20, 2021, accepted May 21, 2021, date of publication May 26, 2021, date of current version June 3, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3084050

The Matthews Correlation Coefficient (MCC) is More Informative Than Cohen's Kappa and Brier Score in Binary Classification Assessment

DAVIDE CHICCO[1], MATTHIJS J. WARRENS[2], AND GIUSEPPE JURMAN[3]
[1]Institute of Health Policy, Management and Evaluation, University of Toronto, Toronto, ON, Canada
[2]Groningen Institute for Educational Research, University of Groningen, Groningen, The Netherlands
[3]Data Science for Health Unit, Fondazione Bruno Kessler, Trento, Italy
Corresponding author: Davide Chicco (davidechicco@davidechicco.it)

$$\mathrm{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)\cdot(TP+FN)\cdot(TN+FP)\cdot(TN+FN)}}$$

$$(\text{worst value} = -1; \text{best value} = +1)$$

ML people seem to like the simplicity of Answer 2 : percent improvement over baseline model which always chooses the largest class.

Evaluation of Binary Classifiers (still complicated!)

Quick Check:

Suppose you have a spam/not spam binary classification task, but your data set is heavily unbalanced: 30% is spam, and 70% is not spam.

You train two different models, A gets 88% accuracy and B gets 92%.

How much better is B than A?

Evaluation of Binary Classifiers (still complicated!)

Quick Check:

Suppose you have a spam/not spam binary classification task, but your data set is heavily unbalanced: 30% is spam, and 70% is not spam.

You train two different models, A gets 88% accuracy and B gets 92%.

How much better is B than A?

A baseline model would simply say that all samples are not spam, and would have 70% accuracy.

Standard answer: B is 22% above the baseline, and A is 18% above.

This doesn't sound that great, compared with 92% for your best model!

You can also see why balanced data sets are preferred!

Evaluation of Multiclass Classifiers (yup, complicated!)

Large Confusion Matrices

If you are doing multiclass classification, you can extend the confusion matrix to have as many rows and columns as the number of classes.

Here is a simple example of a large Confusion Matrix showing how two strings match up (as if each character were a label!):

```
>>> reference = 'This is the reference data.  Testing 123.  aoaeoeoe'
>>> test =      'Thos iz_the rifirenci data.  Testeng 123.  aoaeoeoe'
>>> print(ConfusionMatrix(reference, test))
   |   . 1 2 3 T _ a c d e f g h i n o r s t z |
---+-----------------------------------------+
   |<8>. . . . 1 . . . . . . . . . . . . . . |
.  | .<2>. . . . . . . . . . . . . . . . . . |
1  | . .<1>. . . . . . . . . . . . . . . . . |
2  | . . .<1>. . . . . . . . . . . . . . . . |
3  | . . . .<1>. . . . . . . . . . . . . . . |
T  | . . . . .<2>. . . . . . . . . . . . . . |
_  | . . . . . .<.>. . . . . . . . . . . . . |
a  | . . . . . .<4>. . . . . . . . . . . . . |
c  | . . . . . .<1>. . . . . . . . . . . . . |
d  | . . . . . .<1>. . . . . . . . . . . . . |
e  | . . . . . .<6>. . . 3 . . . . . . . . . |
f  | . . . . . .<1>. . . . . . . . . . . . . |
g  | . . . . . .<1>. . . . . . . . . . . . . |
h  | . . . . . .<2>. . . . . . . . . . . . . |
i  | . . . . . 1 . . .<1>. 1 . . . . . . . . |
n  | . . . . . .<2>. . . . . . . . . . . . . |
o  | . . . . . .<3>. . . . . . . . . . . . . |
r  | . . . . . .<2>. . . . . . . . . . . . . |
s  | . . . . . .<2>. 1 . . . . . . . . . . . |
t  | . . . . . .<3>. . . . . . . . . . . . . |
z  | . . . . . .<.>. . . . . . . . . . . . . |
---+-----------------------------------------+
(row = reference; col = test)
```

The same measures are used as in binary classification:

Accuracy = % correct

Sensitivity and specificity are the average for each row.

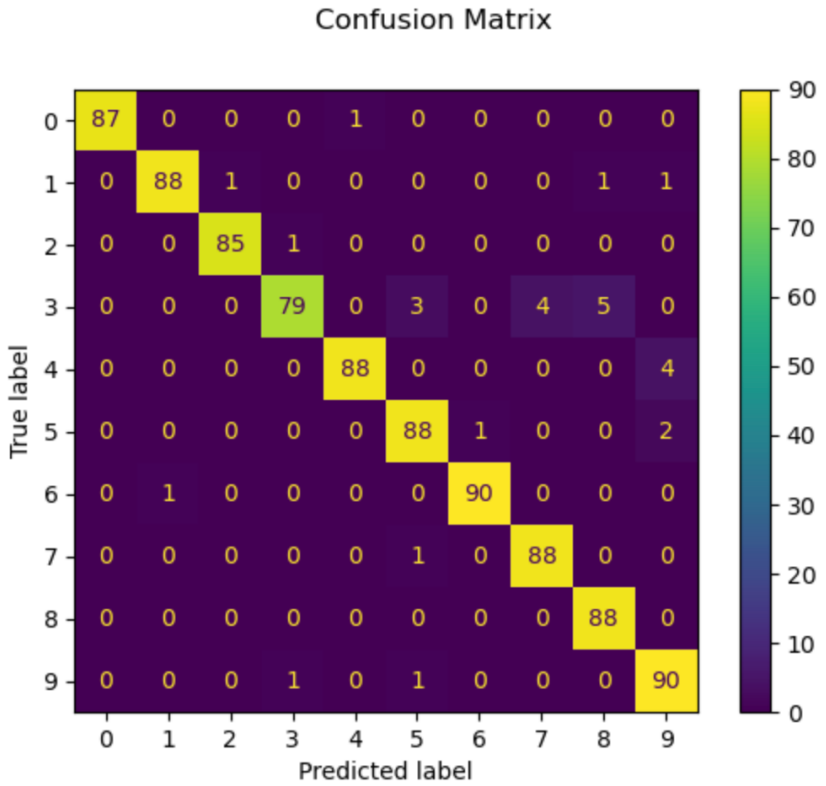
Recall and NPV are the average for each column.

Evaluation of Multiclass Classifiers (yup, complicated!)

Large Confusion Matrices

These displays can help you to understand your data and how your model is performing with respect to individual samples.

Here is a confusion matrix for the MNIST digit-recognition task:

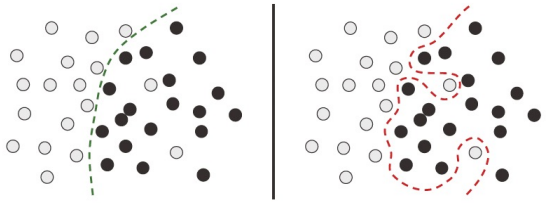


Classification report rebuilt from confusion matrix:

	precision	recall	f1-score	support
0	1.00	0.99	0.99	88
1	0.99	0.97	0.98	91
2	0.99	0.99	0.99	86
3	0.98	0.87	0.92	91
4	0.99	0.96	0.97	92
5	0.95	0.97	0.96	91
6	0.99	0.99	0.99	91
7	0.96	0.99	0.97	89
8	0.94	1.00	0.97	88
9	0.93	0.98	0.95	92
accuracy			0.97	899
macro avg	0.97	0.97	0.97	899
weighted avg	0.97	0.97	0.97	899

Generalization and Overfitting

Generalization – the ability of a NN to learn the patterns in a data set so as to perform well on data it has never seen – is the most important goal in developing deep learning models.



The problem is overfitting – the NN is starting to "memorize" the training set without learning the most important patterns which characterize the essential information present in the data.

Overfitting can be seen when the training loss goes down, but the validation loss goes up. In general, you will see the validation accuracy peak at some epoch and then goes down (generally not as noticably as the rise in the validation loss):

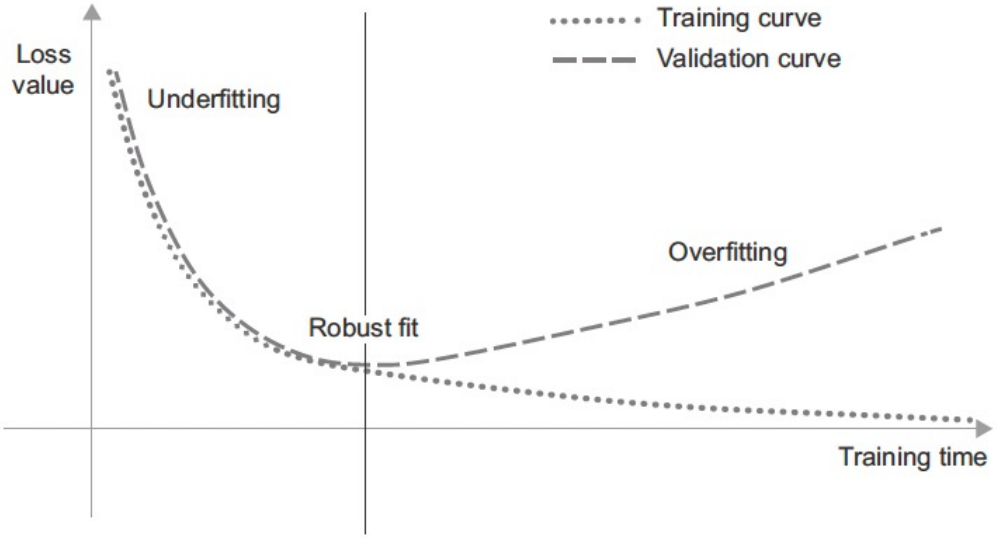
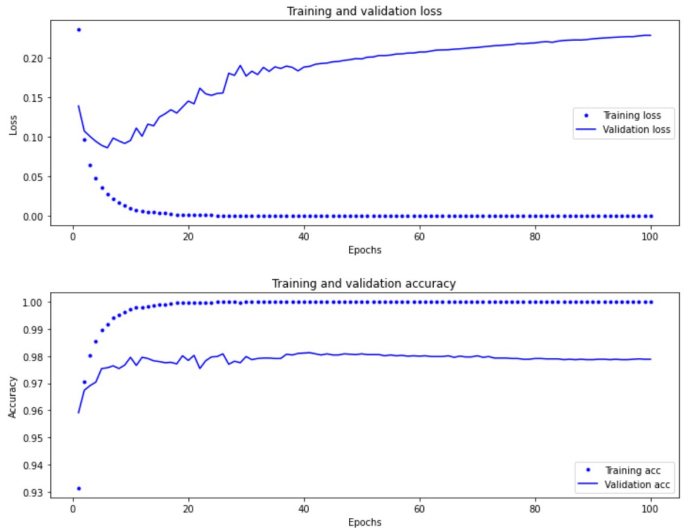


Figure 5.1 Canonical overfitting behavior



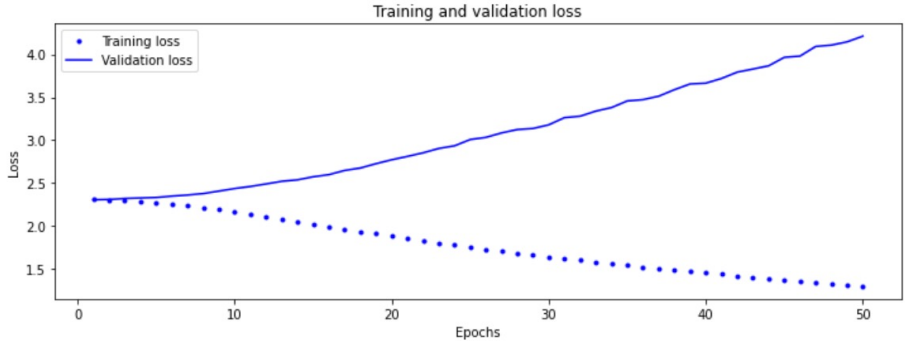

```
Best Validation Accuracy: 0.9813 at epoch  40
```

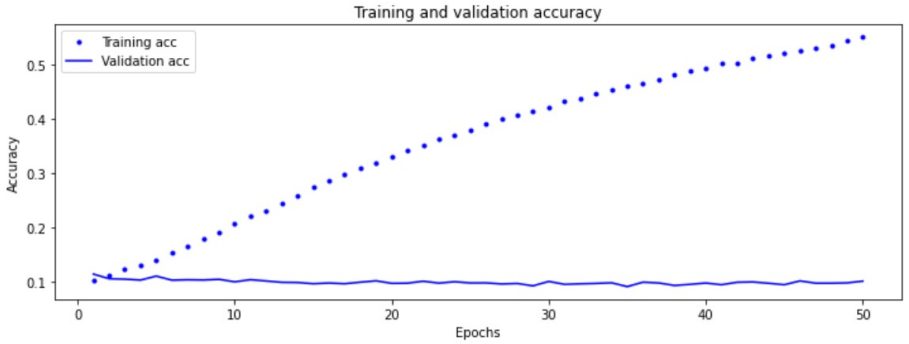
Generalization: Overfitting

The problem is that a NN can learn ANY data set you give it, essentially by memorizing the exact training set. Here is a dramatic example: we randomly permute the labels, so that there is no correspondence between data and labels. The model continues to "learn" the training set, but the validation accuracy remains around the baseline of 10%.

```
shuffled_indices = np.random.permutation(num_data_instances)
shuffled_indices2 = np.random.permutation(num_data_instances)

images = images[shuffled_indices]
labels = labels[shuffled_indices2]
```






```
In [37]: model.evaluate(testing_images, randint(10,size=len(testing_labels)), batch_size = 128)
110/110 [==============================] - 0s 2ms/step - loss: 4.1812 - accuracy: 0.1021
Out[37]: [4.181239128112793, 0.10207142680883408]
```

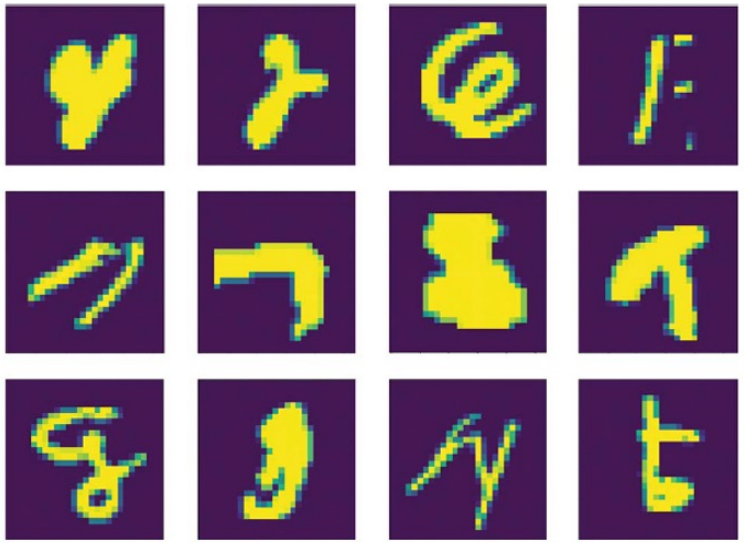
```
Best Validation Accuracy: 0.1139 at epoch  0
```

Generalization: Overfitting

Overfitting is often due to data which is

- Noisy (non-data, ambigious, or outliers)
- Mislabeled
- Or has rare features or spurious correlations.

Noisy Data in MNIST:



Mislabeled data in MNIST:

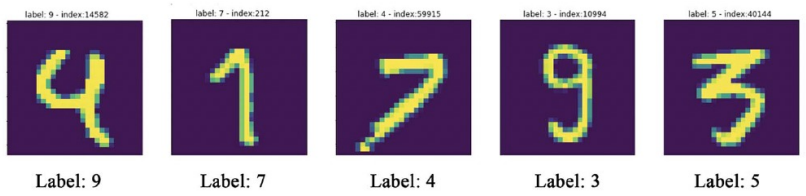


Figure 5.3 Mislabeled MNIST training samples

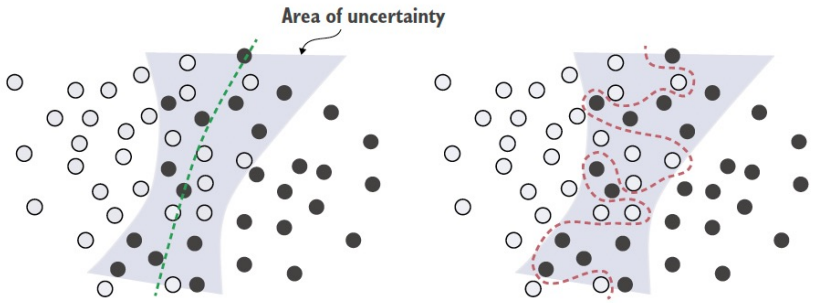


Figure 5.5 Robust fit vs. overfitting giving an ambiguous area of the feature space

Generalization: Overfitting

Overfitting is often due to data which is

- Noisy (non-data, ambigious, or outliers)
- Mislabeled
- Or has rare features or spurious correlations.

Rare features

If your data contains "one-off" features (e.g., a "Getty Images" logo in one image, or a unique or misspelled word in an email), the NN will learn to associate that feature with its label – it is overfitting!

Spurious Correlations

This is actually worse—and more common—than rare features. A word may occur 100s of time in movie reviews, but by a statistical fluke, it occurs in 58% of the positive reviews, and 42% of the negative reviews. The NN will give this word undue weight in learning the data set, and **it won't generalize well**.

Generalization: A Deep Dive into the Matrix..

The Manifold Hypothesis

A manifold in an N dimensional space is a set of points which is isomophic to a lower-dimensional space that is Euclidean, i.e., is continuous and has a notion of "distance."

Ex 1: A curved line is literally in 2 D, but can be mapped 1-to-1 (isomophic) to a 1 D line:

Ex 2: A crumpled piece of paper is 3 D, but is isomophic to a 2D (flat) piece of paper:

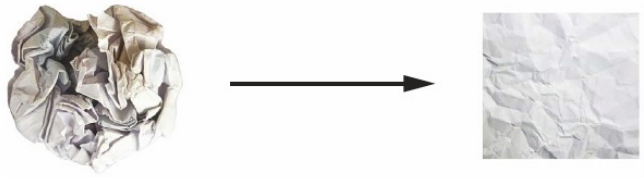


Figure 5.9 Uncrumpling a complicated manifold of data

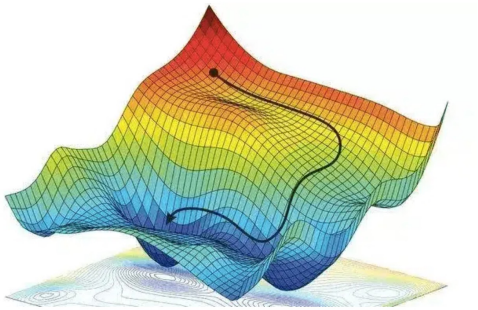
Ex 3:

Möbius strip

Generalization: A Deep Dive into the Matrix..

The Manifold Hypothesis is "that many high-dimensional data sets that occur in the real world actually lie along low-dimensional latent manifolds inside that high-dimensional space. As a consequence of the manifold hypothesis, many data sets that appear to initially require many variables to describe, can actually be described by a comparatively small number of variables, likened to the local coordinate system of the underlying manifold. It is suggested that this principle underpins the effectiveness of machine learning algorithms in describing high-dimensional data sets by considering a few common features." -Wikipedia

Your model is searching in a high-dimensional space (= number of parameters attached as weights to neurons) for a representation of the data (lower dimensional manifold). The spaces are continuous and have a notion of distance, which are intrinsic to the gradient descent algorithm:



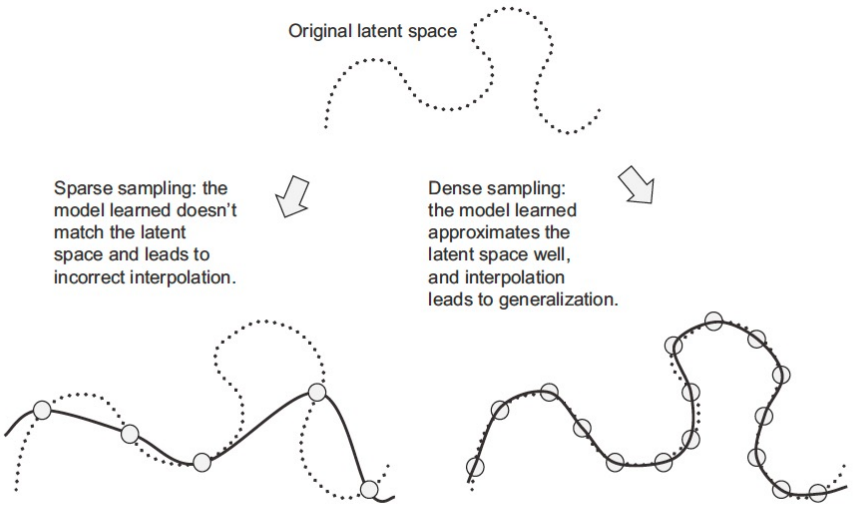


Figure 5.11 A dense sampling of the input space is necessary in order to learn a model capable of accurate generalization.

Generalization: A Deep Dive into the Matrix..

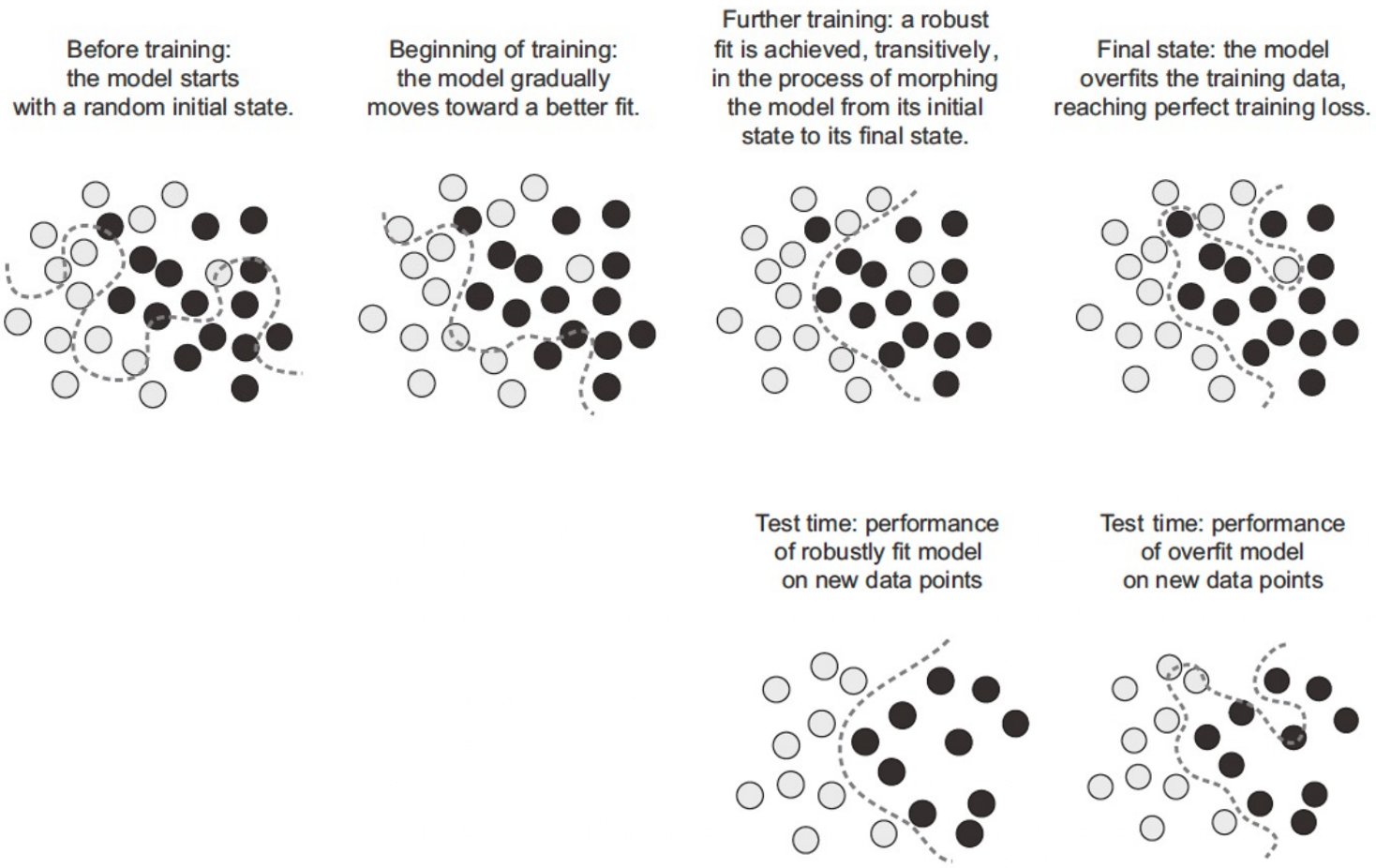


Figure 5.10 Going from a random model to an overfit model, and achieving a robust fit as an intermediate state

Generalization: Underfitting and Overfitting

Overfitting is not a sign that something is wrong with your model, in fact, it shows that your model has sufficient power to represent the patterns that characterize the true "meaning" of the data. You just have to find ways to control this **awesome power**.

Chollet, p.138: "The first big milestone of a machine learning project: getting a model that has some generalization power (it can beat a trivial baseline) and that is able to overfit." p.141: "Remember that it should always be possible to overfit."

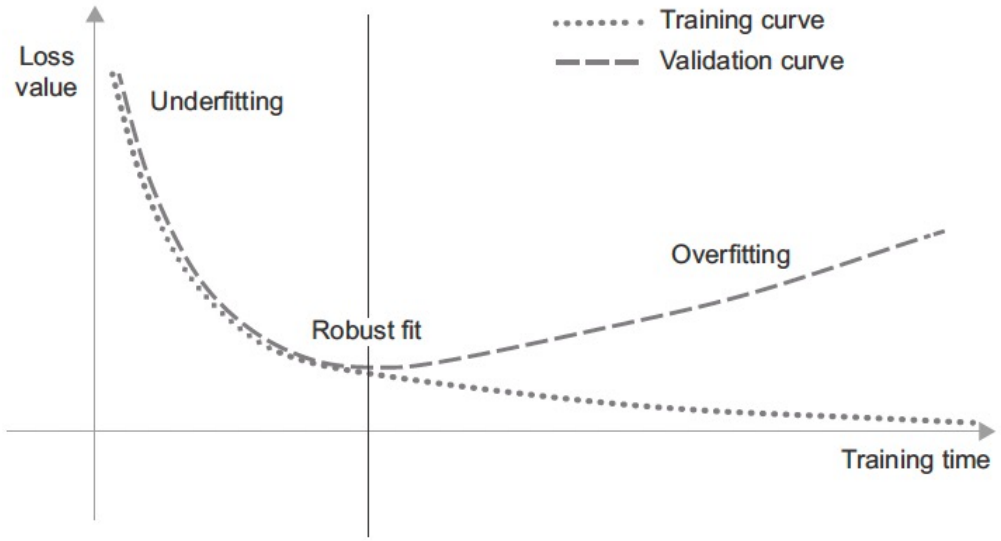


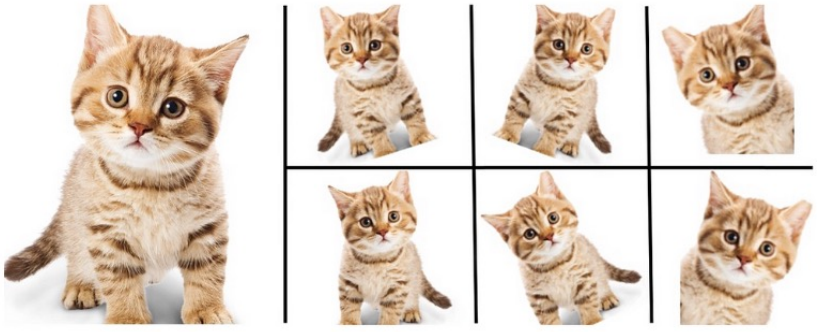
Figure 5.1 Canonical overfitting behavior

Improving Generalization

Improving generalization can be accomplished by various techniques.

Getting more data, improving your data: more data is almost always better; make sure there are minimal labeling errors, reconsider your data normalization.

If you can not get more data, consider data augmentation: manipulating your existing data in ways that produce different samples with the *same essential information*.



Unfortunately, data augmentation is a little tricky in NLP: how do you create new documents or text that has the same "essential information"?

Typical approaches: replace words by synonyms, translate to another language, then back, etc. (not very satisfying).

Improving Generalization

Reconsider your choice of architecture: Add more layers, or fewer, or of different widths. Consider starting with wider layers, and getting narrower as you go deeper. Consider different kinds of layers better suited to your data (this works better with images than in NLP). Google around to see what others have done successfully with similar data.

Tuning hyperparameters: Play with the hyperparameters, including type of optimizer, the learning rate, and the batch size.

Better feature engineering: Use domain knowledge about the data, and experience with the model you are using to better represent the data. Tools can help with feature selection (find out which features are making the most difference).

Example: What kind of word vector?

TF? TF-IDF? DIY embeddings?

Glove embeddings? etc. ....

In some simple cases, we can observe which features were most important; here is a Naive Bayes classifier for identifying gender for names:

```
Most Informative Features
                suffix2 = 'na'           female : male   =    100.3 : 1.0
                suffix2 = 'la'           female : male   =     77.0 : 1.0
                suffix2 = 'ia'           female : male   =     41.3 : 1.0
                suffix3 = 'son'            male : female =     36.5 : 1.0
                suffix2 = 'ld'             male : female =     35.9 : 1.0
                suffix2 = 'sa'           female : male   =     34.8 : 1.0
                suffix2 = 'us'             male : female =     29.6 : 1.0
                suffix3 = 'ana'          female : male   =     25.7 : 1.0
                suffix3 = 'tta'          female : male   =     25.7 : 1.0
                suffix2 = 'ta'           female : male   =     25.7 : 1.0
                suffix2 = 'rd'             male : female =     25.2 : 1.0
                suffix2 = 'ra'           female : male   =     25.1 : 1.0
                suffix2 = 'do'             male : female =     23.8 : 1.0
                suffix2 = 'rt'             male : female =     22.5 : 1.0
                suffix3 = 'ard'            male : female =     21.0 : 1.0
                suffix2 = 'os'             male : female =     20.4 : 1.0
                suffix3 = 'nne'          female : male   =     20.1 : 1.0
```

Improving Generalization

Early Stopping: Stop training when a robust fit is achieved. This can often be done automatically by setting a parameter in your model. An elegant method is to save the best model after every K epoches, then refer back after you've gone too far....

Here is a naive example of early stopping, which does not do so well:

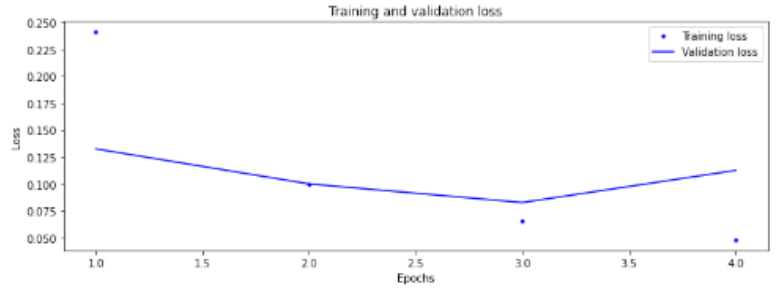
```
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping()
custom_early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=8,
    min_delta=0.001,
    mode='max'
)


model = keras.Sequential([
    layers.Dense(512,activation="relu"),
    layers.Dense(10,activation='softmax')
])

model.compile( optimizer = 'rmsprop',
               loss = 'sparse_categorical_crossentropy',
               metrics = ['accuracy'] )

history = model.fit(training_images,
                    training_labels,
                    verbose= 0,
                    epochs=50,
                    batch_size=64,
                    callbacks=[early_stopping],
                    validation_data=(validation_images,validation_labels))

display_results(history)
model.evaluate(testing_images, testing_labels,batch_size = 128)
```




```
Best Validation Accuracy: 0.9749 at epoch  2
110/110 [==============================] - 0s 1ms/step - loss: 0.1168 - accuracy: 0.9689

[0.11679539084434509, 0.9688571691513062]
```

Improving Generalization

Early Stopping: Stop training when a robust fit is achieved. This can often be done automatically by setting a parameter in your model.

Tuning the early stopping callback results in better results:

In [37]:

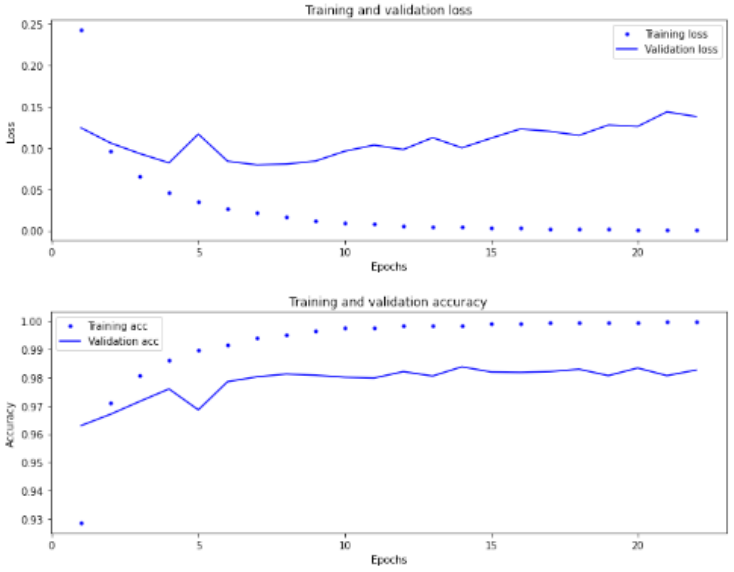
```
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping()
custom_early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=8,
    min_delta=0.001,
    mode='max'
)


model = keras.Sequential([
    layers.Dense(512,activation="relu"),
    layers.Dense(10,activation='softmax')
])

model.compile( optimizer = 'rmsprop',
               loss = 'sparse_categorical_crossentropy',
               metrics = ['accuracy'] )

history = model.fit(training_images,
                    training_labels,
                    verbose= 0,
                    epochs=50,
                    batch_size=64,
                    callbacks=[custom_early_stopping],
                    validation_data=(validation_images,validation_la

display_results(history)
model.evaluate(testing_images, testing_labels,batch_size = 128)
```




```
Best Validation Accuracy: 0.9839 at epoch  13
```

In [36]:

```
model.evaluate(testing_images, testing_labels,batch_size = 128)
```

```
110/110 [==============================] - 0s 1ms/step - loss: 0.1475 - accuracy: 0.9807
```

Out[36]:

```
[0.14754530787467957, 0.9807142615318298]
```

Improving Generalization

Regularization: Various techniques which "actively impede the model's ability to fit perfectly to the training data, with the goal of making the model perform better during validation." The model is simpler, more "regular."

Reduce model size (but not too small):

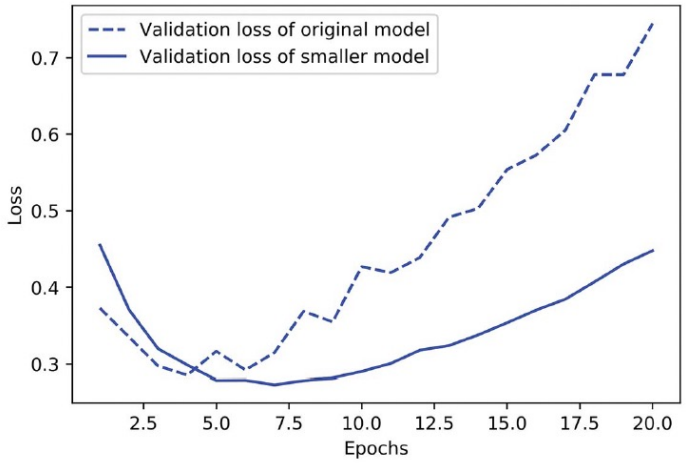


Figure 5.17 Original model vs. smaller model on IMDB review classification

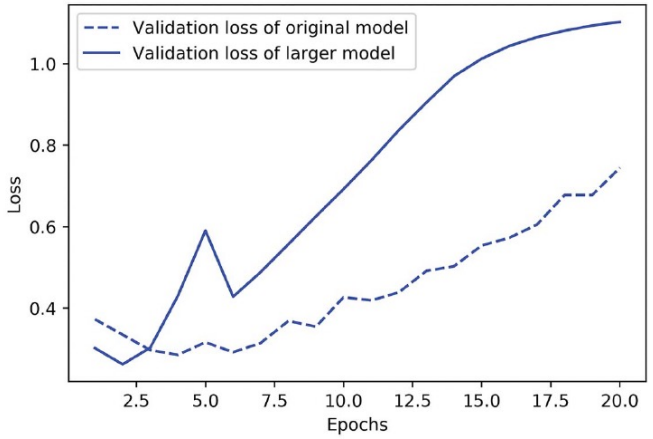


Figure 5.18 Original model vs. much larger model on IMDB review classification

Improving Generalization

Regularization: Various techniques which "actively impede the model's ability to fit perfectly to the training data, with the goal of making the model perform better during validation." The model is simpler, more "regular."

Weight Regularization: Place limits on how large the weights in the model can become, so that the model is forced to be simpler (having fewer possibilities of weights).
There are two flavors:

- *L1 regularization*—The cost added is proportional to the *absolute value of the weight coefficients* (the *L1 norm* of the weights).
- *L2 regularization*—The cost added is proportional to the *square of the value of the weight coefficients* (the *L2 norm* of the weights). L2 regularization is also called *weight decay* in the context of neural networks. Don't let the different name confuse you: weight decay is mathematically the same as L2 regularization.

Listing 5.13 Adding L2 weight regularization to the model

```
from tensorflow.keras import regularizers
model = keras.Sequential([
    layers.Dense(16,
                 kernel_regularizer=regularizers.l2(0.002),
                 activation="relu"),
    layers.Dense(16,
                 kernel_regularizer=regularizers.l2(0.002),
                 activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

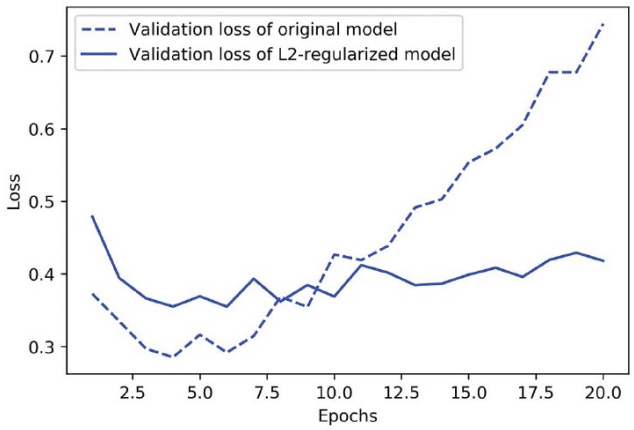


Figure 5.19 Effect of L2 weight regularization on validation loss

Improving Generalization

Regularization: Various techniques which "actively impede the model's ability to fit perfectly to the training data, with the goal of making the model perform better during validation." The model is simpler, more "regular."

Adding Dropout: Dropout is applied to a layer, and is very simple: with some probability p, set each parameter in a layer to 0.0:

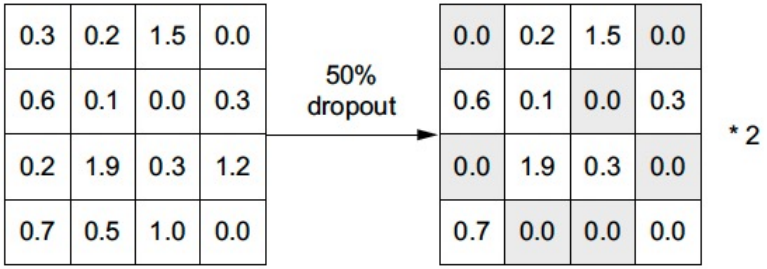


Figure 5.20 Dropout applied to an activation matrix at training time, with rescaling happening during training. At test time the activation matrix is unchanged.

This is one of the weirdest great ideas in Deep Learning: it seems like it can't possibly help, but it is one of the most effective and most common ways to regularize your model.

Listing 5.15 Adding dropout to the IMDB model

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
history_dropout = model.fit(
    train_data, train_labels,
    epochs=20, batch_size=512, validation_split=0.4)
```

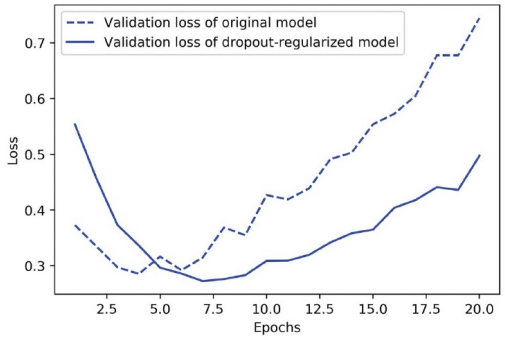


Figure 5.21 Effect of dropout on validation loss