Deep Learning

Lecture Topic:
Fundamentals of Machine Learning

Anxiao (Andrew) Jiang

Learning Objectives:

1. Understand underfitting and overfitting.

2. Understand regularization techniques.

3. Understand basic learning paradigms.

Roadmap of this lecture:

1. Under-fitting and over-fitting.

2. Regularizations: L1, L2, dropout.

3. Basic learning paradigms.

Underfitting and overfitting

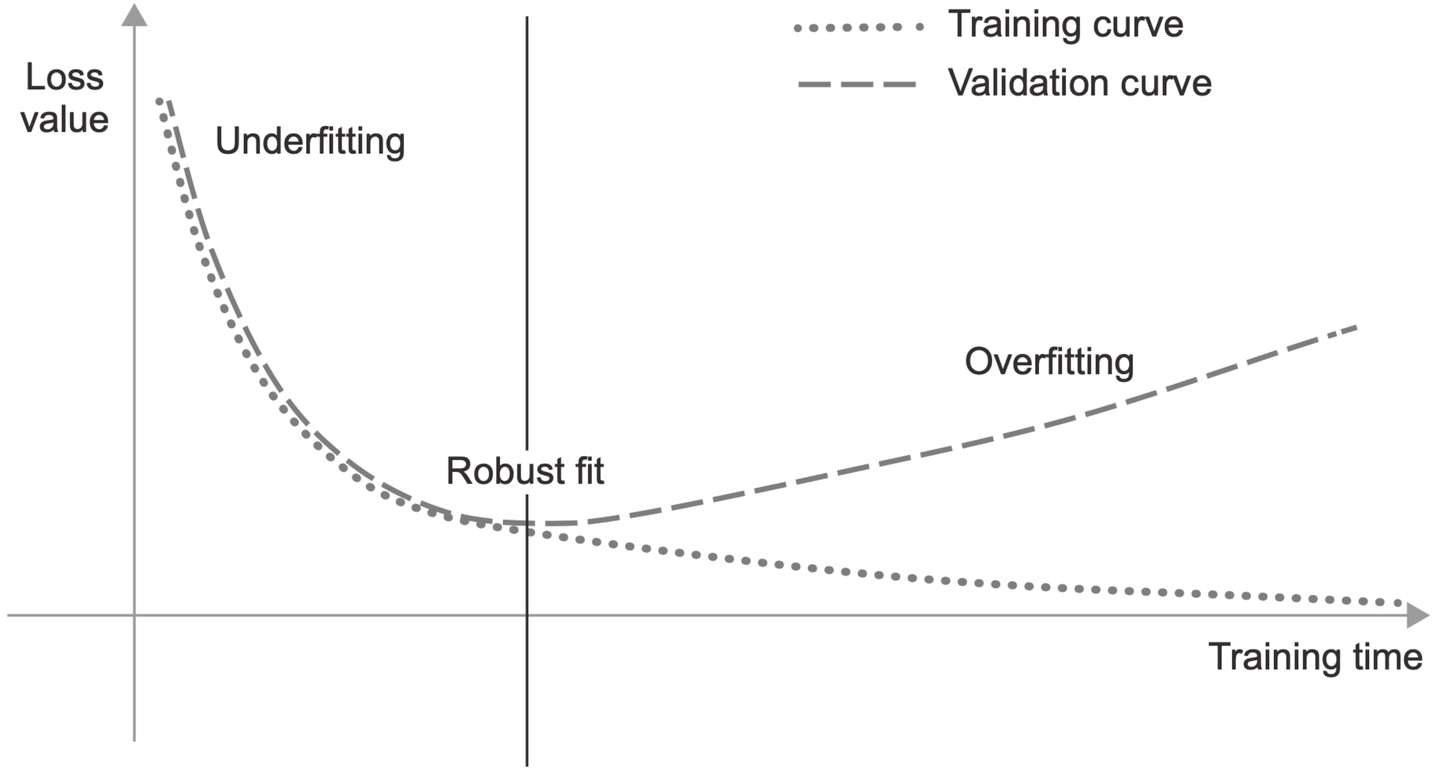


Figure 5.1 Canonical overfitting behavior

Noisy Training Data

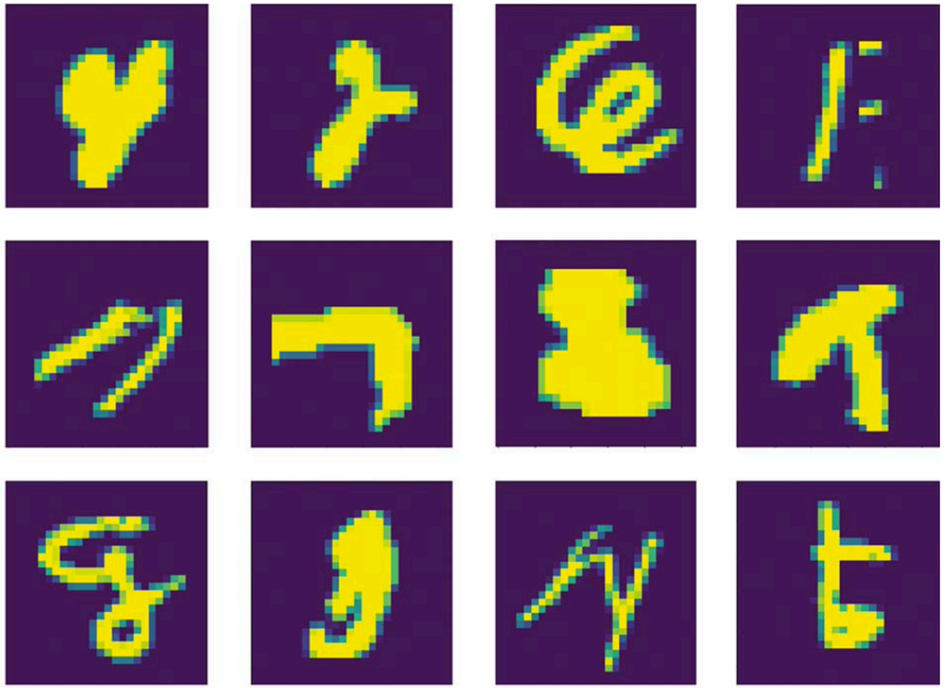


Figure 5.2 Some pretty weird MNIST training samples

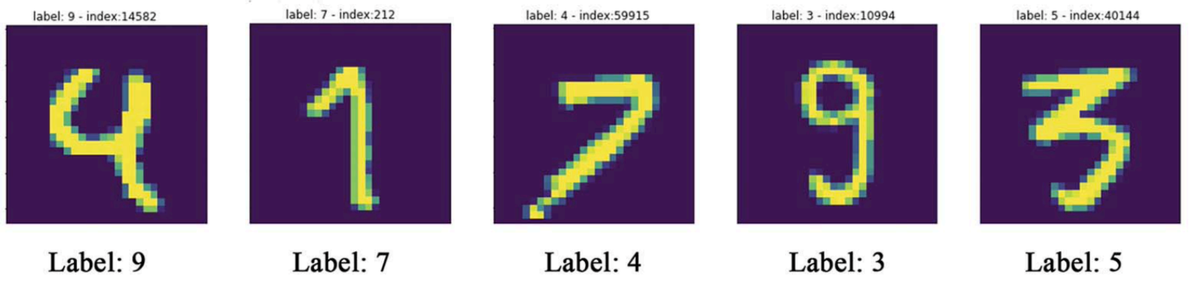


Figure 5.3 Mislabeled MNIST training samples

Robust fit vs. overfitting

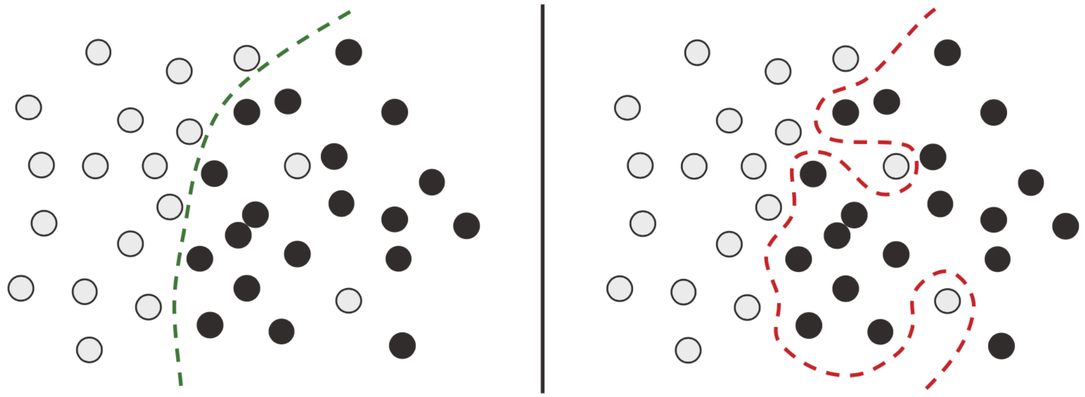


Figure 5.4 Dealing with outliers: robust fit vs. overfitting

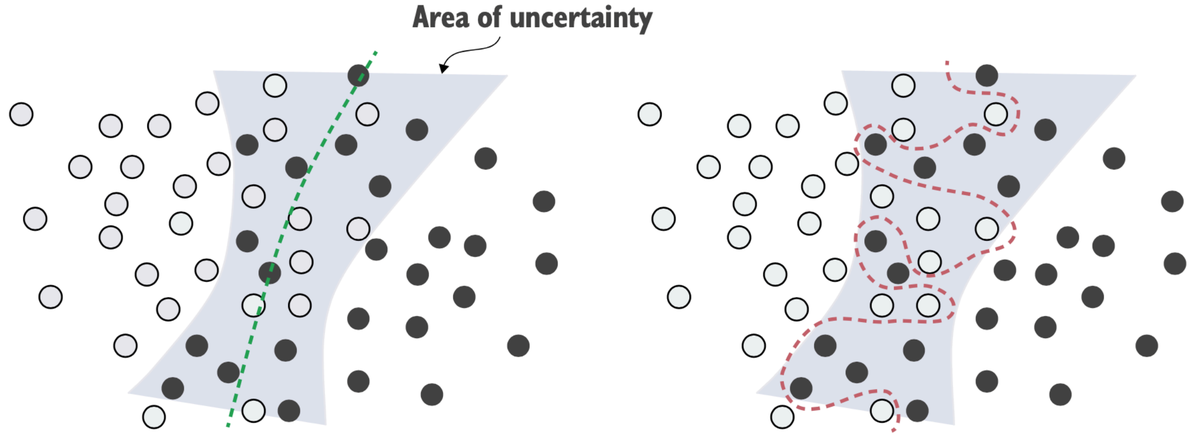


Figure 5.5 Robust fit vs. overfitting giving an ambiguous area of the feature space

Training data is paramount

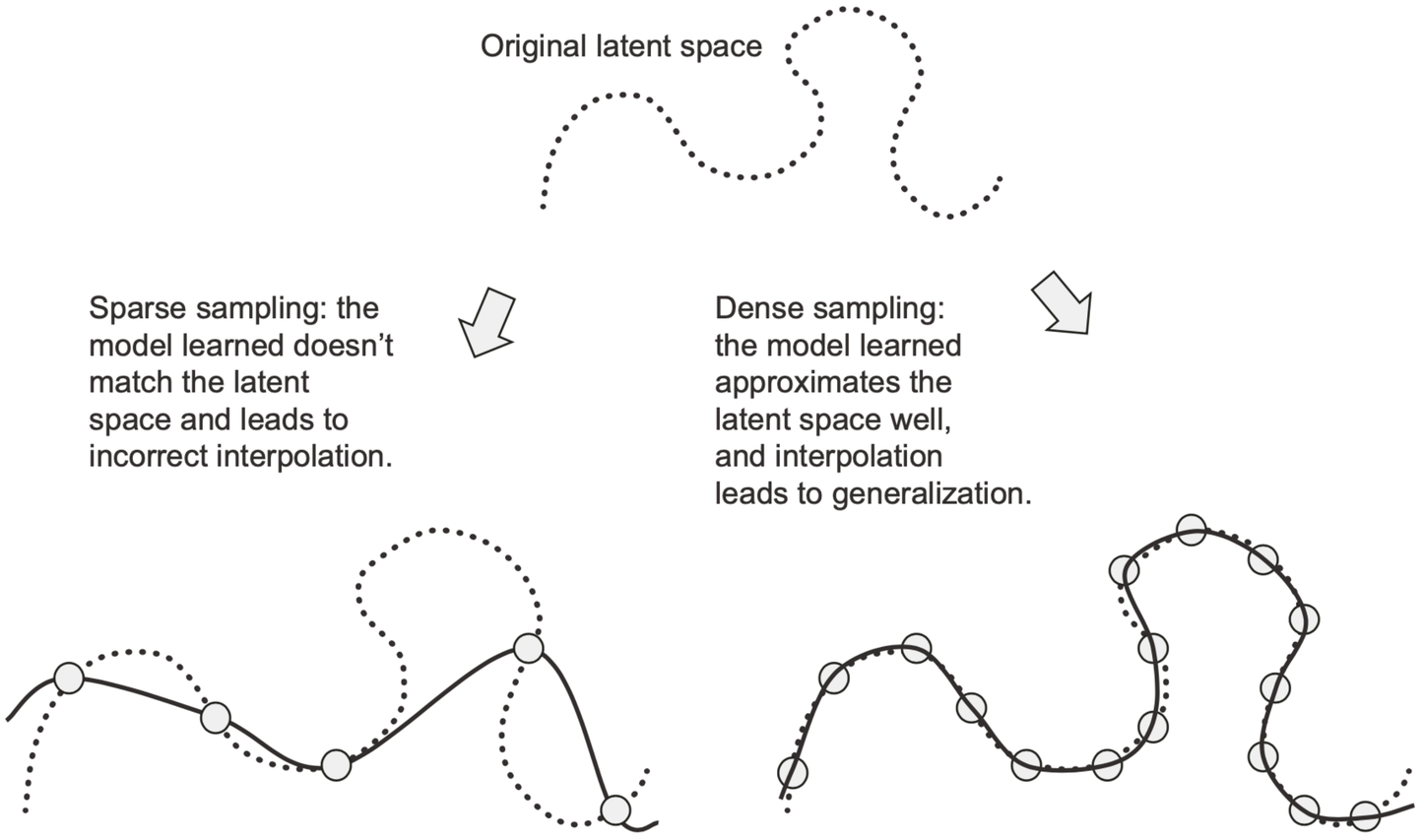


Figure 5.11 A dense sampling of the input space is necessary in order to learn a model capable of accurate generalization.

Underfitting: training performance is bad

- This is the first thing we should worry about.

Overfitting: training performance is good, but test performance is bad

- This is the next thing we should worry about.

To prevent underfitting: change our model or make it general (to include the correct function in our solution set). Then train our NN well to find that correct function.

To prevent overfitting: after we have found a NN with good training performance, simplify the NN model (e.g., reduce NN size, or use regularization techniques). Or get more data.

Quiz questions:

1. What is under-fitting?

2. What is over-fitting?

Roadmap of this lecture:

1. Under-fitting and over-fitting.

2. Regularizations: L1, L2, dropout.

3. Basic learning paradigms.

Regularization techniques

- Weight regularization: add a function of weights to the loss function, to prevent the weights from becoming too large.

L2 regularization $\quad \text{new loss function} = \text{old loss function} + \lambda \sum_i w_i^2$

L1 regularization $\quad \text{new loss function} = \text{old loss function} + \lambda \sum_i |w_i|$

A reason for weight regularization: large weight can make the model more sensitive to noise/variance in data.

L2 regularization: it tends to make all weights small.

L1 regularization: it tends to make weights sparser (namely, more 0s).

Add weight 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"])
```

Add weight regularization

```
history_l2_reg = model.fit(
    train_data, train_labels,
    epochs=20, batch_size=512, validation_split=0.4)
```

In the preceding listing, `l2(0.002)` means every coefficient in the weight matrix of the layer will add `0.002 * weight_coefficient_value ** 2` to the total loss of the model. Note that because this penalty is *only added at training time*, the loss for this model will be much higher at training than at test time.

Add weight regularization

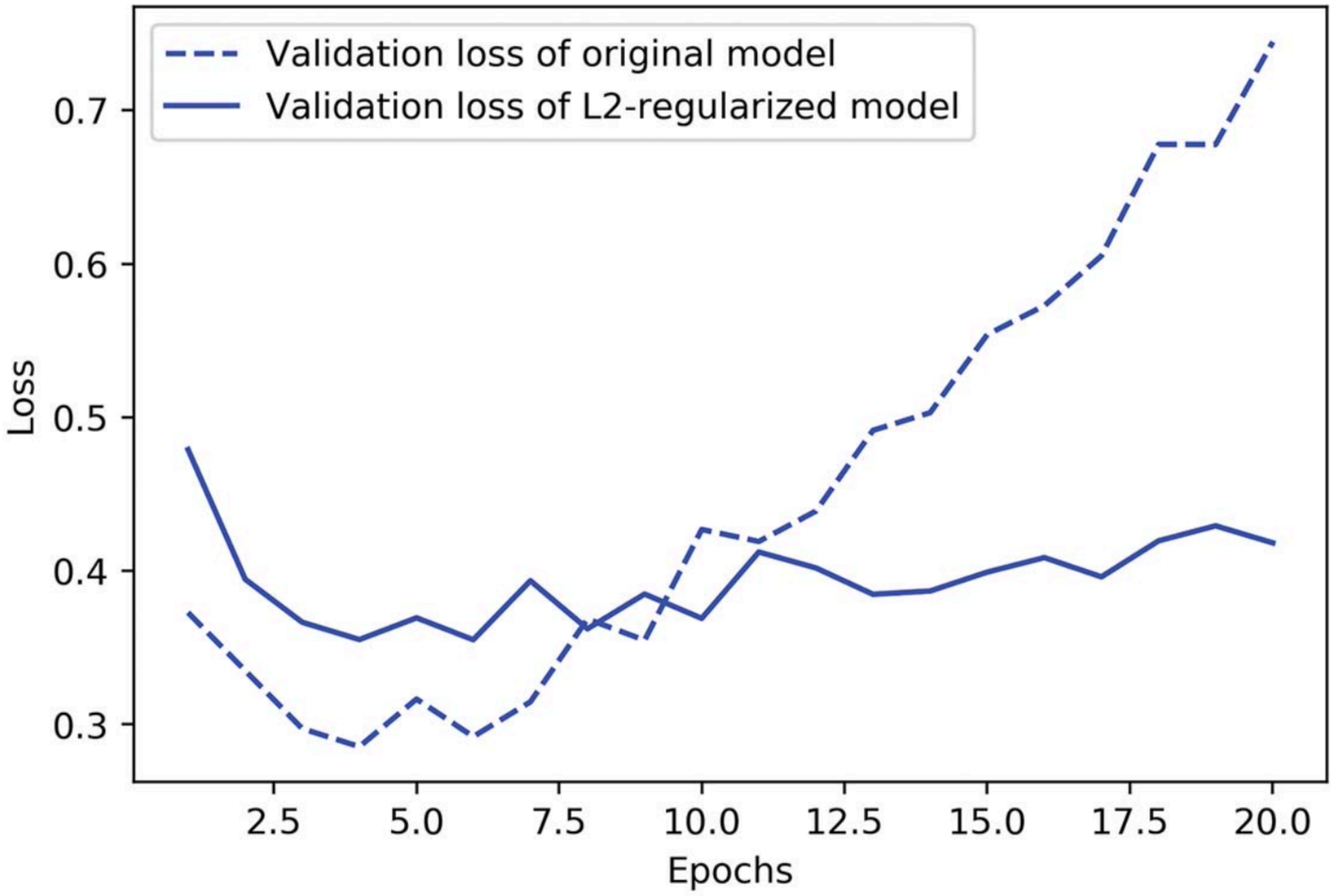


Figure 5.19 Effect of L2 weight regularization on validation loss

Add weight regularization

As an alternative to L2 regularization, you can use one of the following Keras weight regularizers.

Listing 5.14 Different weight regularizers available in Keras

```
from tensorflow.keras import regularizers
regularizers.l1(0.001)
regularizers.l1_l2(l1=0.001, l2=0.001)
```

L1 regularization

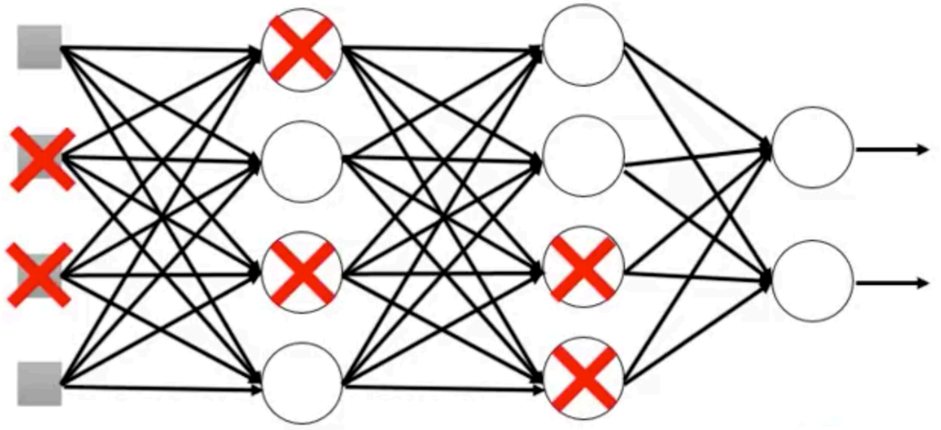
Simultaneous L1 and L2 regularization

Note that weight regularization is more typically used for smaller deep learning models. Large deep learning models tend to be so overparameterized that imposing constraints on weight values hasn't much impact on model capacity and generalization. In these cases, a different regularization technique is preferred: dropout.

Another regularization technique: Dropout

Dropout

Training:



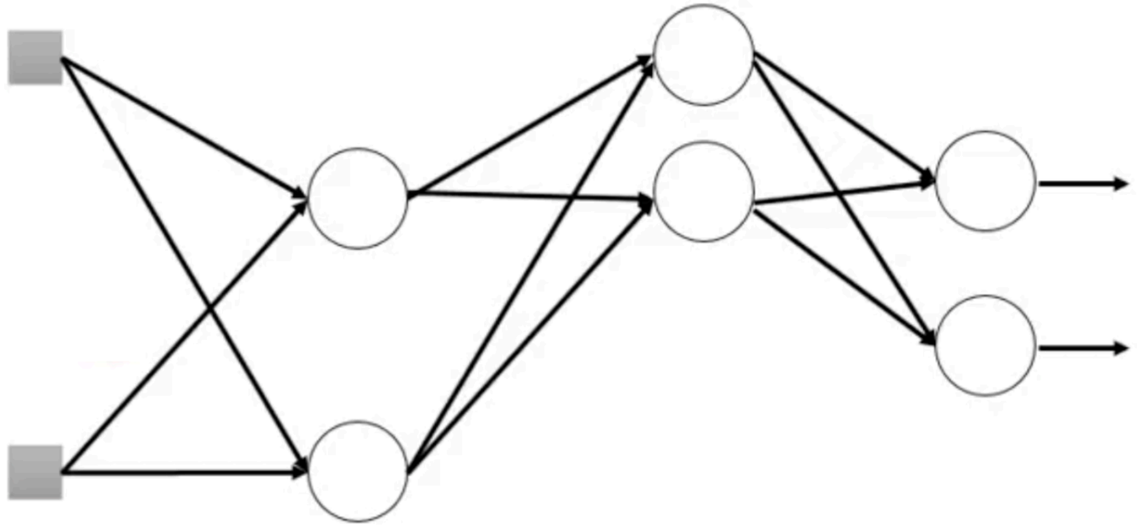
- **Each time before updating the parameters**
  - Each neuron has p% to dropout

When a node drops out, its output and its outgoing edges are all removed for this weight update.

The following several slides were adapted from a public lecture by Dr. Hung-yi Lee.

Dropout

Training:



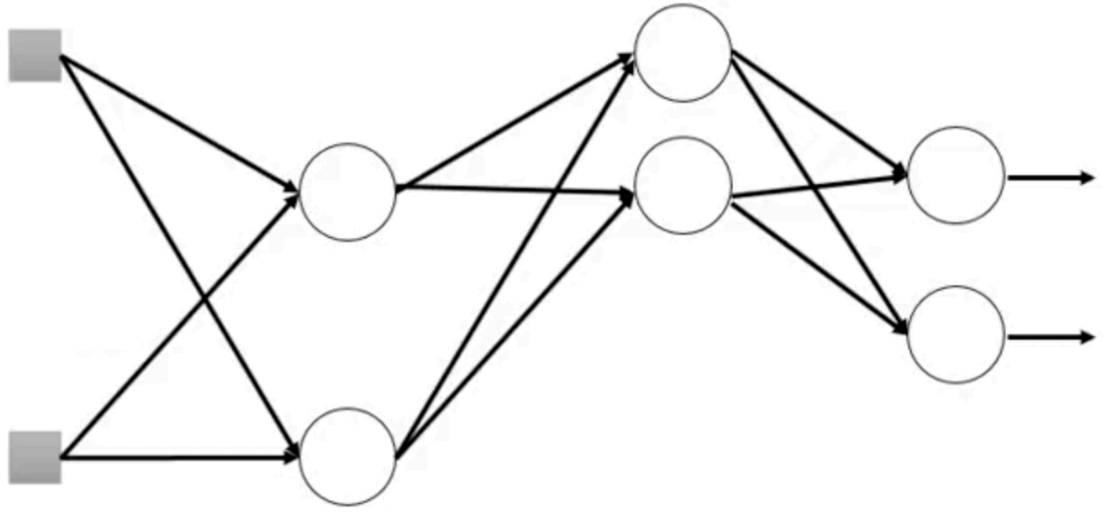
- **Each time before updating the parameters**
  - Each neuron has p% to dropout

    → **The structure of the network is changed.**

Use the new network to train its weights, for this mini-batch of data.

Dropout

Training:



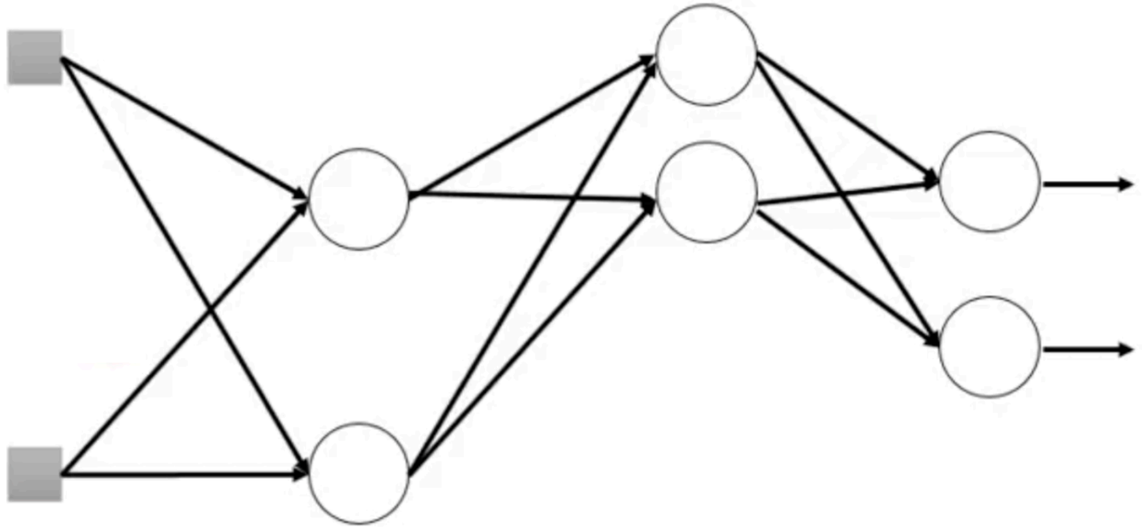
- **Each time before updating the parameters**
  - Each neuron has p% to dropout

    → **The structure of the network is changed.**

For each mini-batch, we re-sample the nodes to drop out.

Dropout

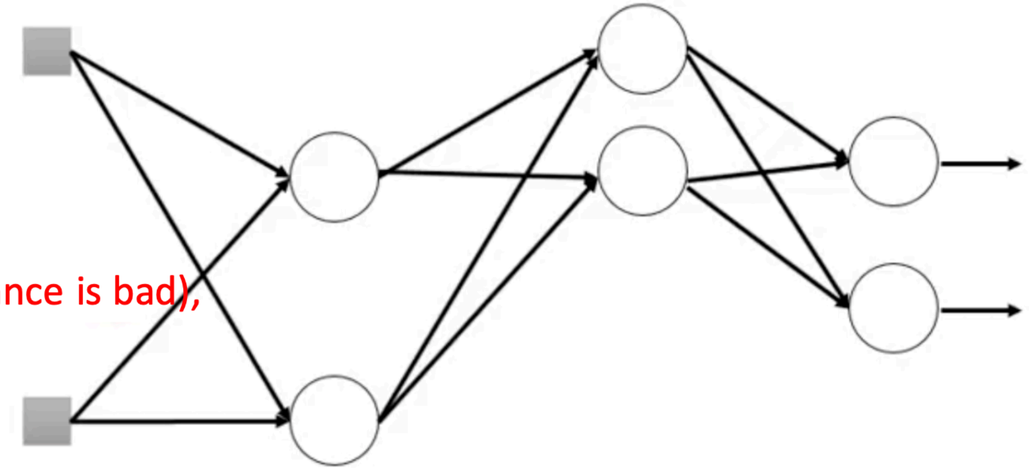
Training:



With dropout, training performance will worsen.
(But we hope the test performance will become better due to less overfitting).

Dropout

Training:

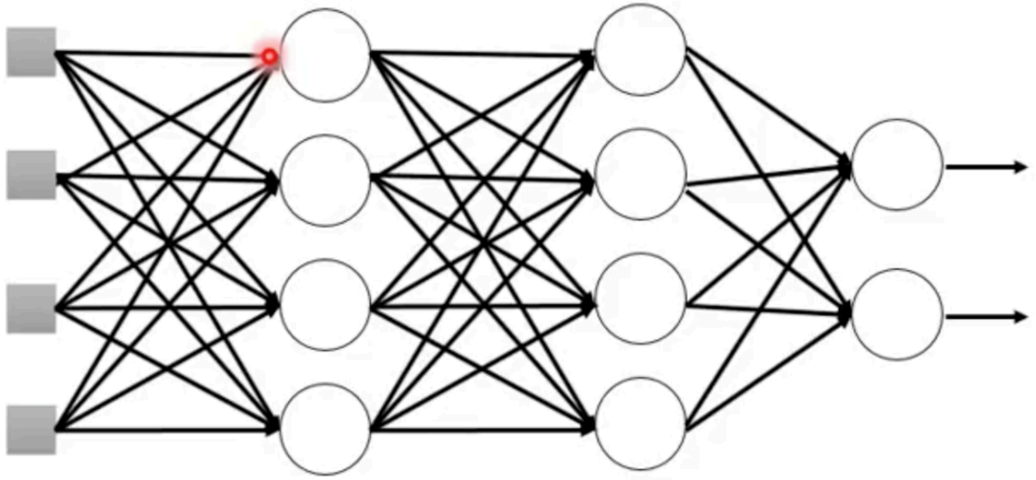


If the NN is underfitting
(namely, training performance is bad),
don't use dropout.

With dropout, training performance will worsen.
(But we hope the test performance will become better due to less overfitting).

Dropout

Testing:



- **No dropout**
  - If the dropout rate at training is p%, all the weights times (1-p)%

Keras will take care of it automatically for us. So don't worry about it.

Add Dropout

In Keras, you can introduce dropout in a model via the `Dropout` layer, which is applied to the output of the layer right before it. Let's add two `Dropout` layers in the IMDB model to see how well they do at reducing overfitting.

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)
```

Add Dropout

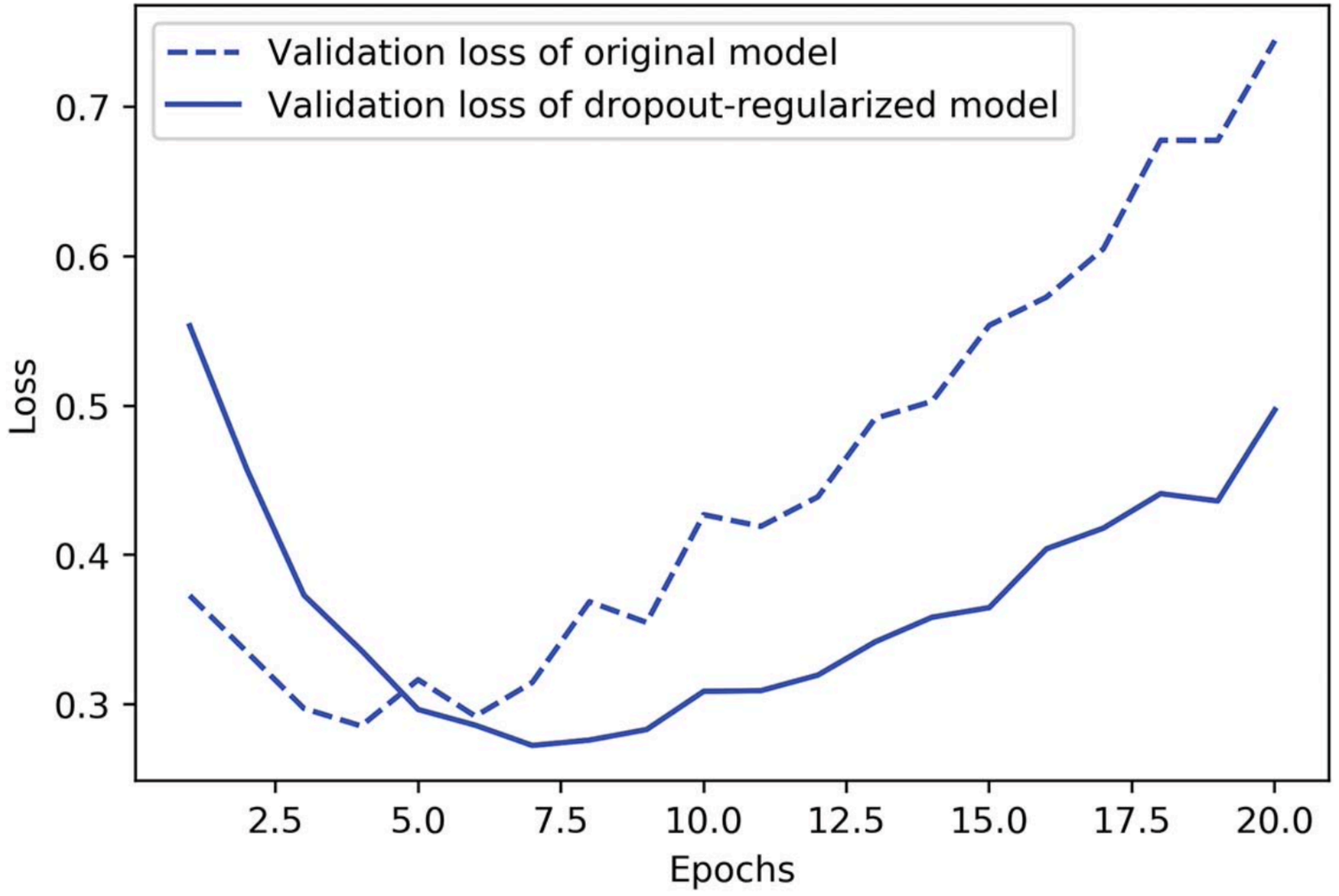


Figure 5.21 Effect of dropout on validation loss

Intuitions for dropout

- Weights have less chance of “collusion” for overfitting.
- Each weight “trains harder” to capture a feature, since other weights may dropout during training. (If my teammates do not work hard, then I have to work harder.)

Quiz questions:

1. What is the difference between L1/L2 regulation and dropout?

2. When should we use regulation?

Roadmap of this lecture:

1. Under-fitting and over-fitting.

2. Regularizations: L1, L2, dropout.

3. Basic learning paradigms.

Supervised Learning

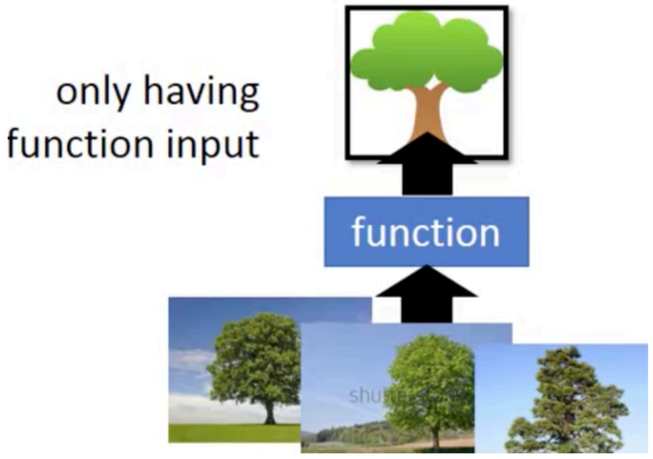
- Input and output are both known. Just learn the function.

- The four applications introduced so far in our class are all supervised learning.

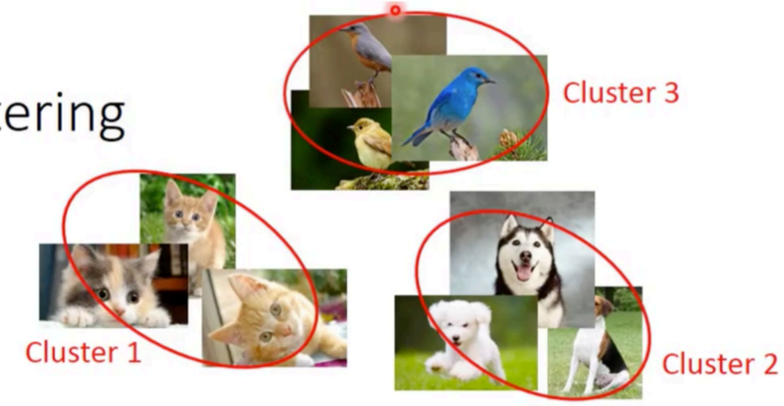
Unsupervised Learning

- **Output is unknown. Learn the relationship between data.**

- Dimension Reduction



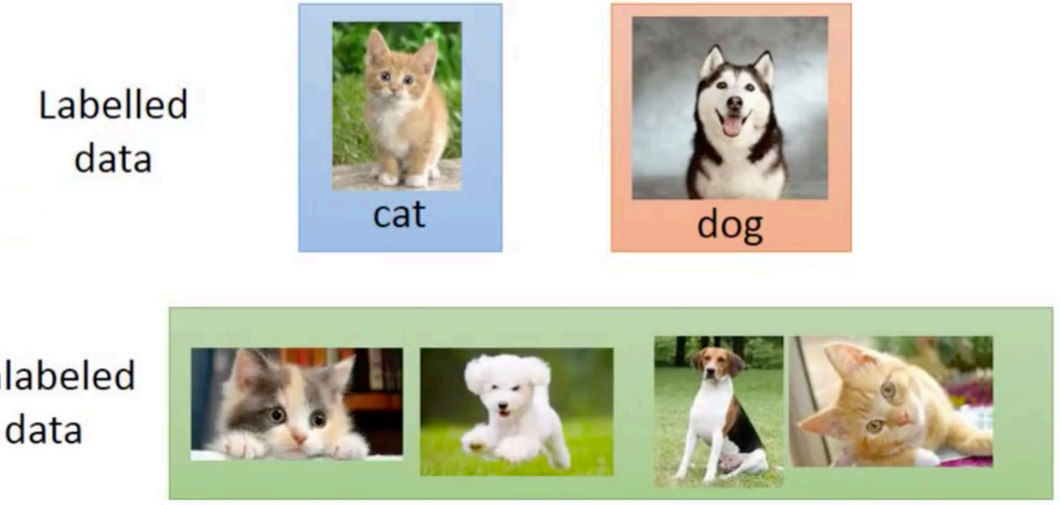
Clustering



Figures by Prof. Hung-yi Lee.

Semi-supervised Learning

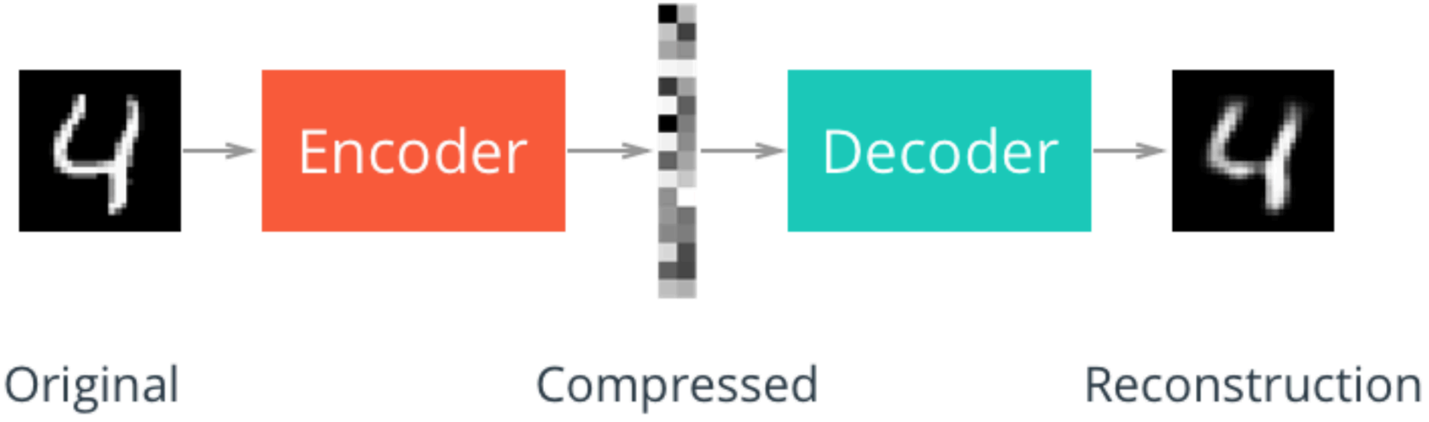
- Some outputs are known, but not all. (Most data are unlabeled.)



Figures by Prof. Hung-yi Lee.

Self-supervised Learning

- Output is generated from input data, without human help.
- Example: auto-encoder



Reinforcement Learning

- Learn from feedback (penalty or reward) from environment.
- But the environment does not tell what to do.

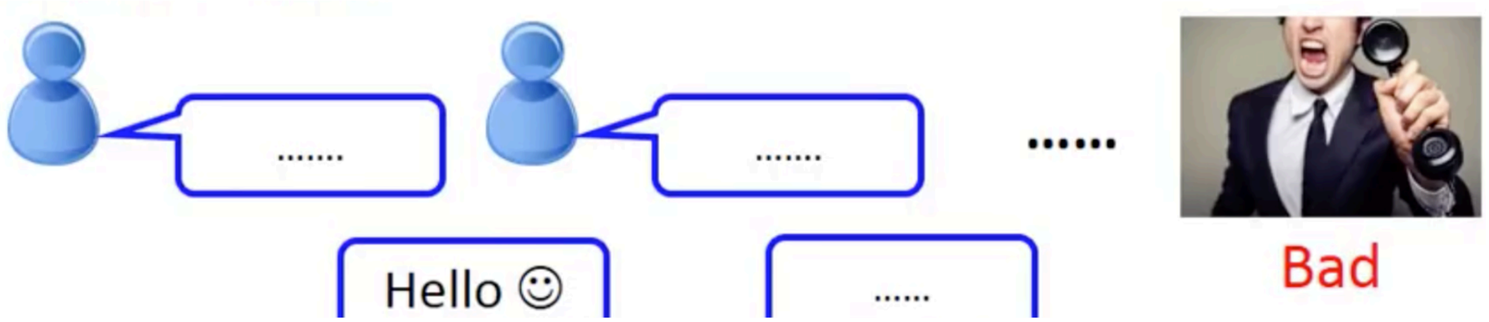


Figure by Prof. Hung-yi Lee.

Handle missing values

You may sometimes have missing values in your data. For instance, in the house-price example, the first feature (the column of index 0 in the data) was the per capita crime rate. What if this feature wasn't available for all samples? You'd then have missing values in the training or test data.

You could just discard the feature entirely, but you don't necessarily have to.

- If the feature is categorical, it's safe to create a new category that means "the value is missing." The model will automatically learn what this implies with respect to the targets.
- If the feature is numerical, avoid inputting an arbitrary value like `"0"`, because it may create a discontinuity in the latent space formed by your features, making it harder for a model trained on it to generalize. Instead, consider replacing the missing value with the average or median value for the feature in the dataset. You could also train a model to predict the feature value given the values of other features.

Quiz questions:

1. What are the different learning paradigms? When should we use them, respectively?

2. How to handle missing values in data?