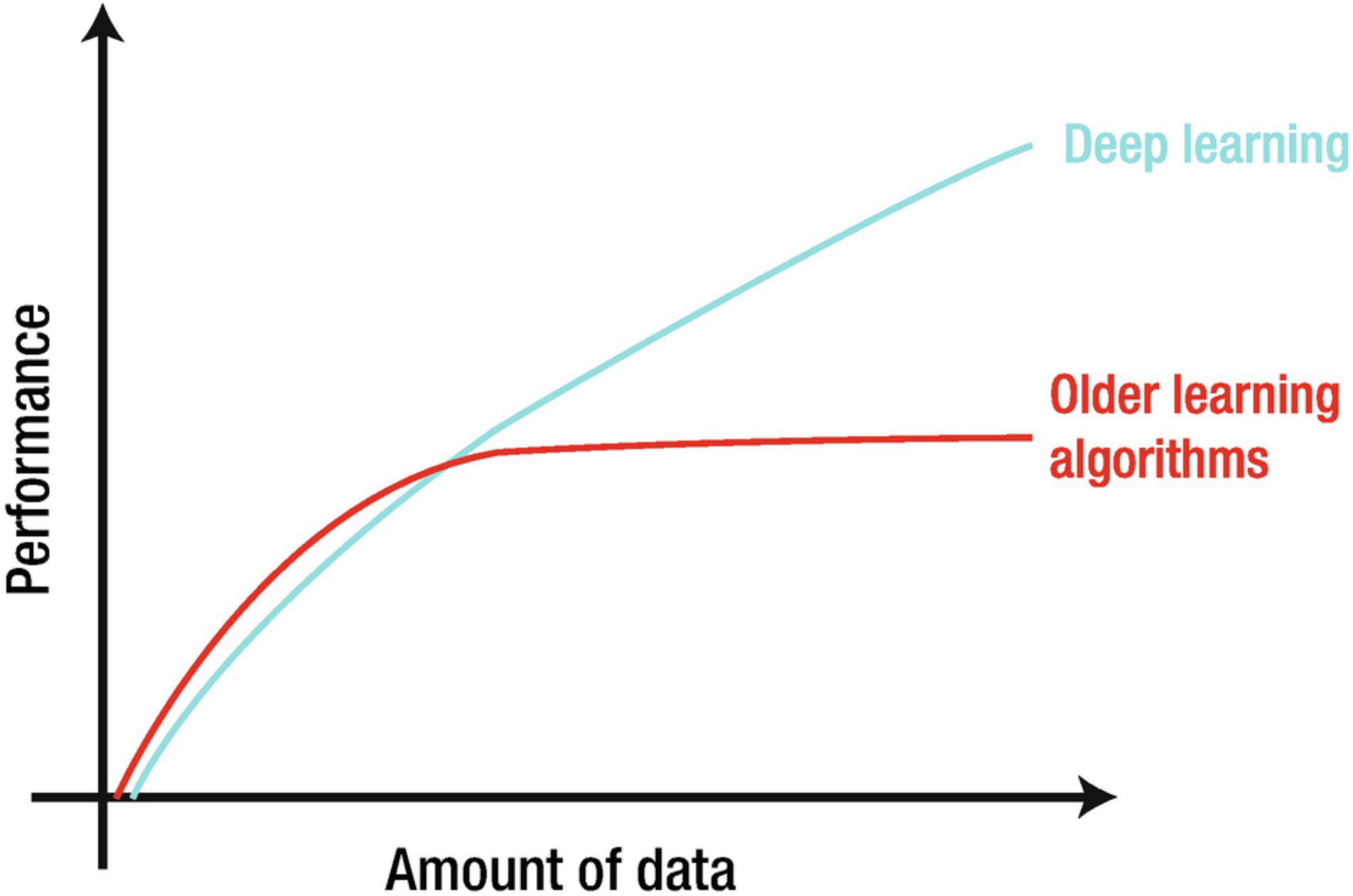
Improving Deep Neural Networks

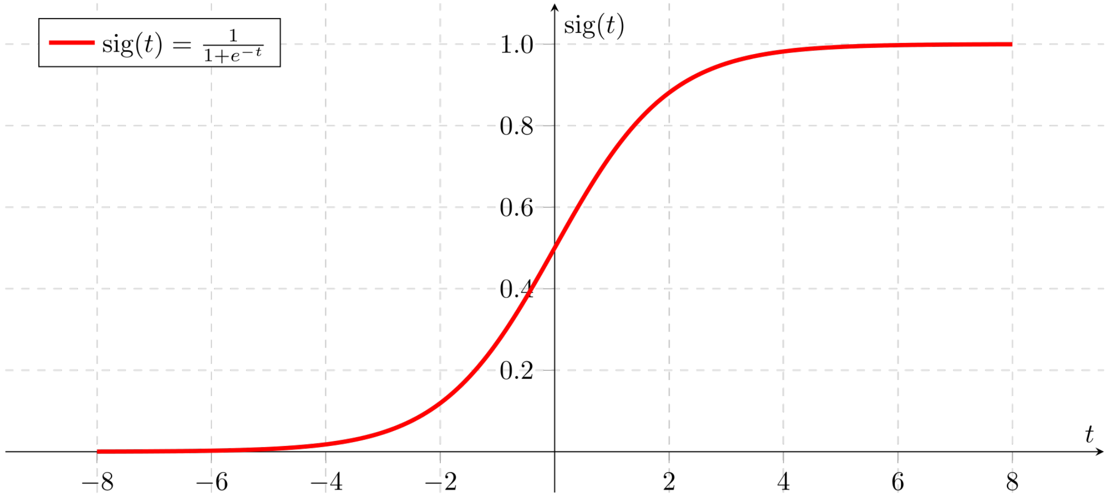
Il-Youp Kwak, PhD

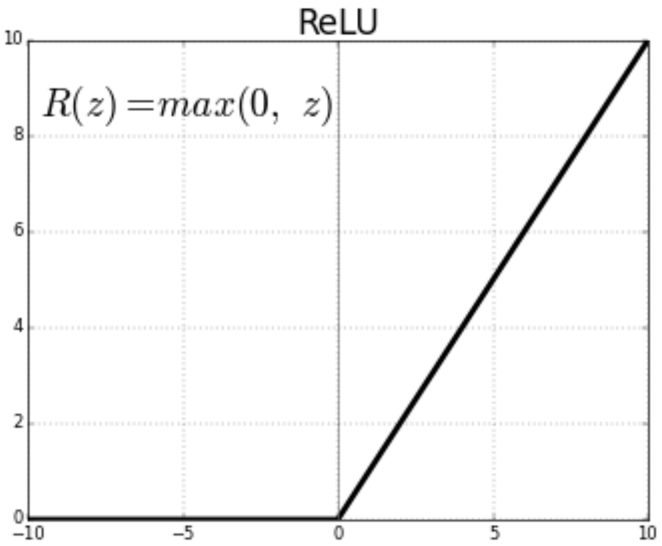
Why deep learning?



Why deep learning taking off?

- Firstly proposed in 1943
- Originally had problem in computational speed
- Development in hardware for computing (GPUs) and algorithm itself





https://playground.tensorflow.org

Binary Classification

- **We are Classifying Cat or Dog** $f : \mathbf{x} \xrightarrow{f_\theta} \mathbb{R}_{[0,1]}$

- **Dimension for x is 64*64*3 = 12288**

- **Data:** $(\mathbf{x}, y) \qquad \mathbf{x} \in \mathbb{R}^{n_x}, y \in \{0, 1\}$

  m training examples $\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})\}$

  $X = [\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}] \quad Y = [y^{(1)}, \ldots, y^{(m)}]$

Logistic Regression

- Given $\mathbf{x} \in \mathbb{R}^{n_x}$, want $\hat{y} = P(y = 1|\mathbf{x}) \in \mathbb{R}^{[0,1]}$
- Parameters: $\mathbf{w} \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$
- Output: $\hat{y} = \sigma(\mathbf{w}^t\mathbf{x} + b)$, where $\sigma(z) = 1/(1 + e^{-z})$

Loss function

- **Squared error loss:** $L(\hat{y}, y) = (\hat{y} - y)^2$

- **Cross entropy loss:**

$$L(\hat{y}, y) = -(ylog(\hat{y}) + (1 - y)log(1 - \hat{y}))$$

- **Cost for Logistic regression: Use Cross entropy loss**

$$C(W, b) = \sum_{i=1}^{m} L(\hat{y_i}, y_i)$$

Constructing Logistic regression

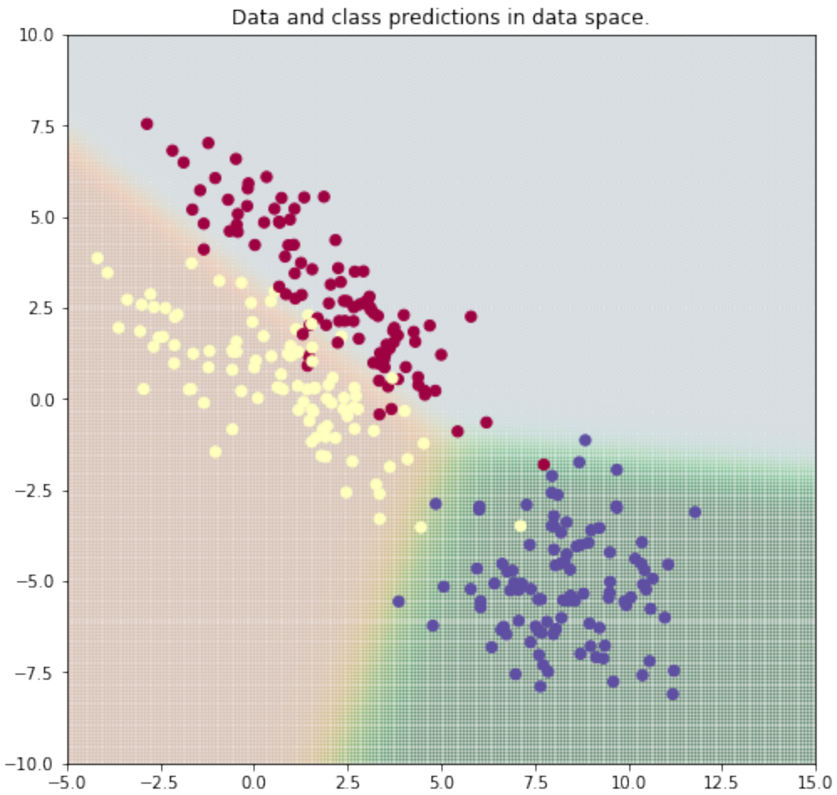
```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(2, activation='sigmoid')
])
```

```
[17] model.compile(optimizer='adam',
                   loss='binary_crossentropy',
                   metrics=['accuracy'])
```

Softmax Regression

- $\hat{\mathbf{z}} = e^{(W\mathbf{x}+\mathbf{b})}$ $\quad t = \sum_i \hat{z}_i$

- Then, $\hat{\mathbf{y}} = \hat{\mathbf{z}}/t$ represent probability for each item



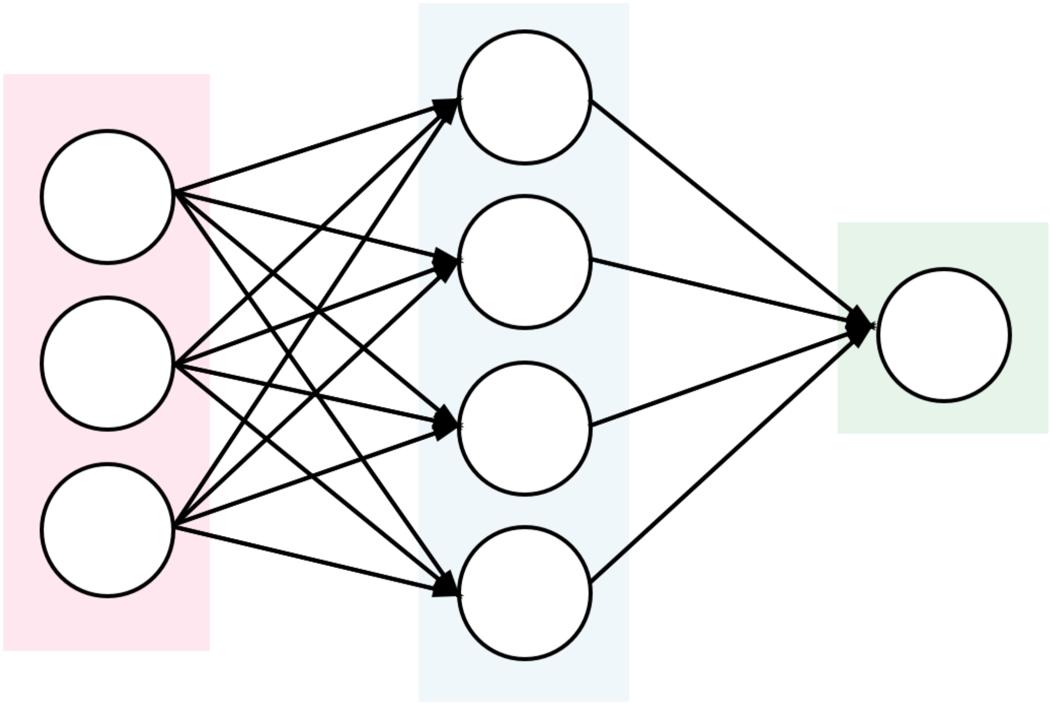
Constructing softmax regression

```
[14] model = keras.Sequential([
         keras.layers.Flatten(input_shape=(28, 28)),
         keras.layers.Dense(2, activation='softmax')
     ])
```

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

Artificial Neural Networks with one hidden layer

- Output: $\hat{\mathbf{y}} = \sigma(W_2 \text{relu}(W_1 \mathbf{x} + \mathbf{b\_1}) + \mathbf{b\_2})$



Constructing ANN

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(50, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

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

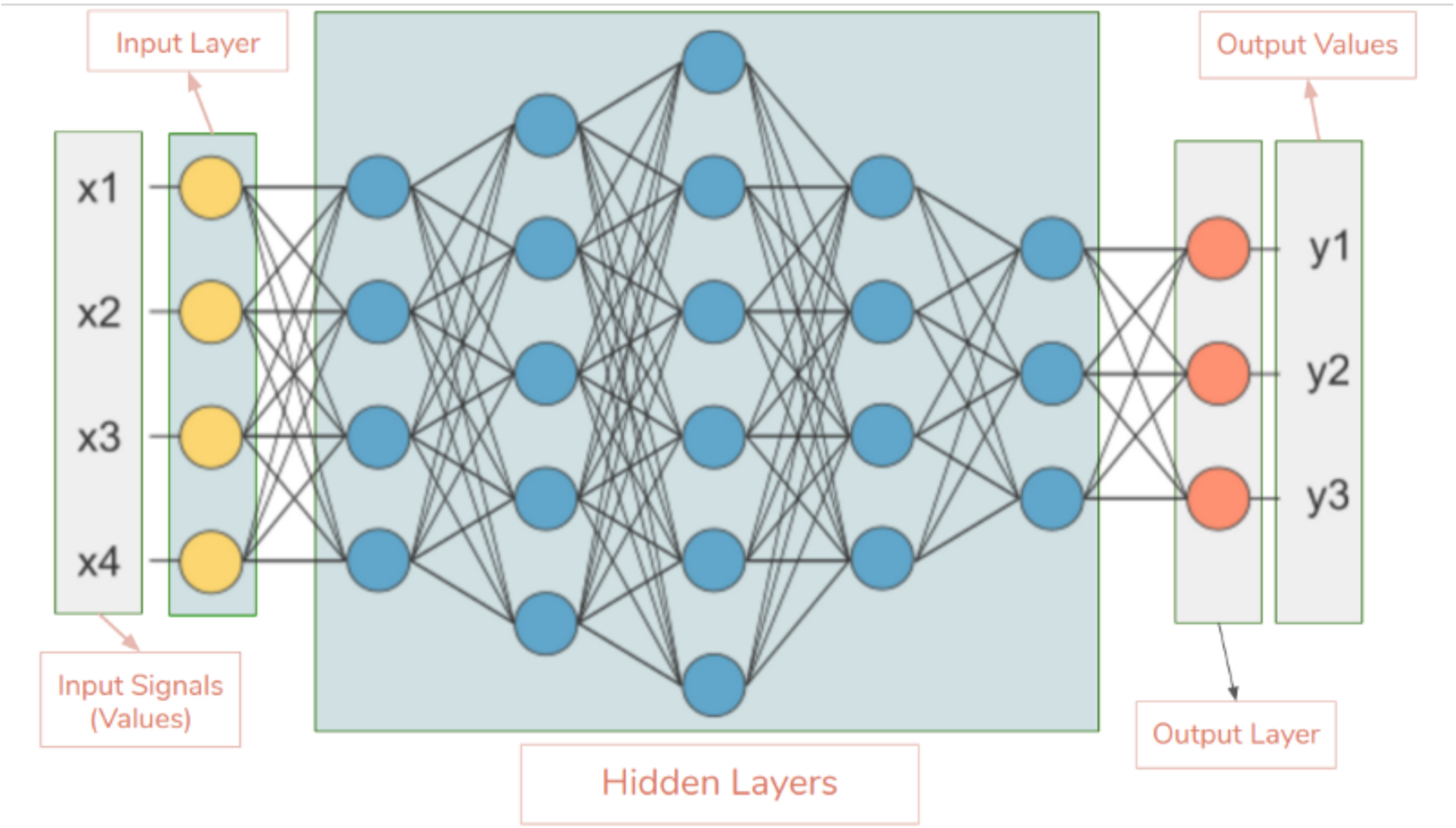
Check model with model.summary()

```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_4 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_6 (Dense)              (None, 50)                39250
_________________________________________________________________
dense_7 (Dense)              (None, 10)                510
=================================================================
Total params: 39,760
Trainable params: 39,760
Non-trainable params: 0
_________________________________________________________________
```

ANN with multiple hidden layers

- **Ex)** $y = f_1(f_2(f_3(f_4(f_5(x)))))$



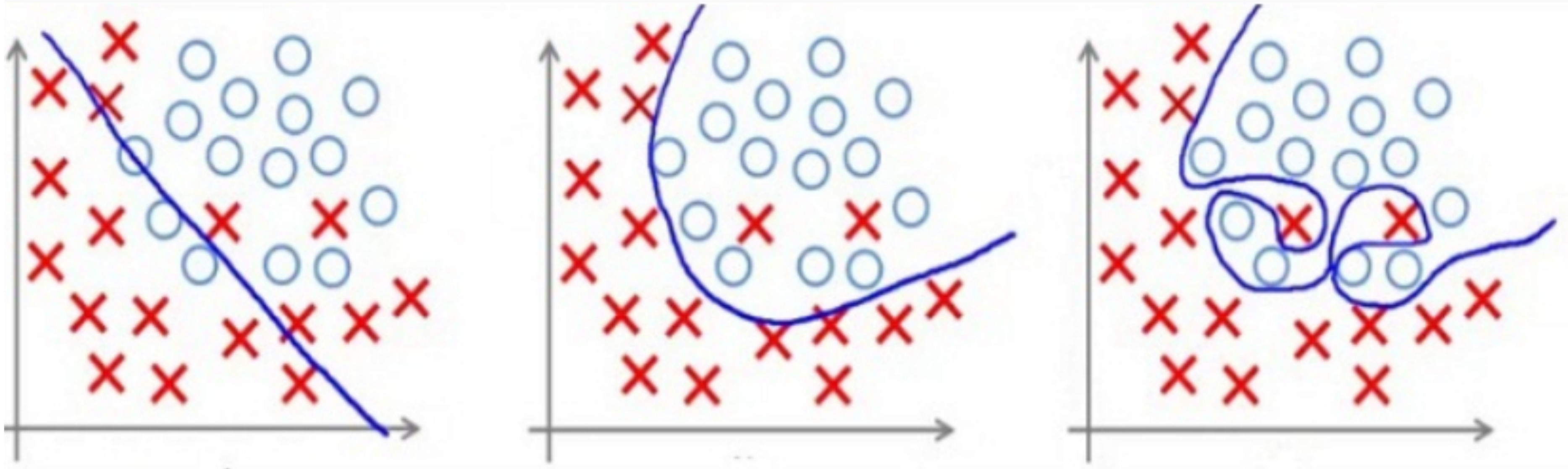
Practice

Train / Dev / Test set

- Traditionally, 7:3 for train and dev or 6:2:2
- Or, 6:2:2
- With big data, 98:1:1 or use even larger train set
- It is important to use independent, separate Dev, Test set with different configuration for real evaluation

https://www.youtube.com/watch?v=1waHlpKiNyY&list=PLkDaE6sCZn6Hn0vK8co82zjQtt3T2Nkqc&index=1

Overfitting / underfitting



Under-fitting (too simple to explain the variance)

Appropriate-fitting

Over-fitting (forcefitting -- too good to be true)

Check accuracy on train / dev

- Too good on train and low on dev imply overfitting
- Try to minimize accuracy(or EER, AUC, F1) on dev
- Check whether you have balanced or unbalanced data (train and dev, consider cost-sensitive learning when unbalanced)

Regularization

- **Consider regularization when overfitted**

- **L1, L2**

Ex) in logistic regression:

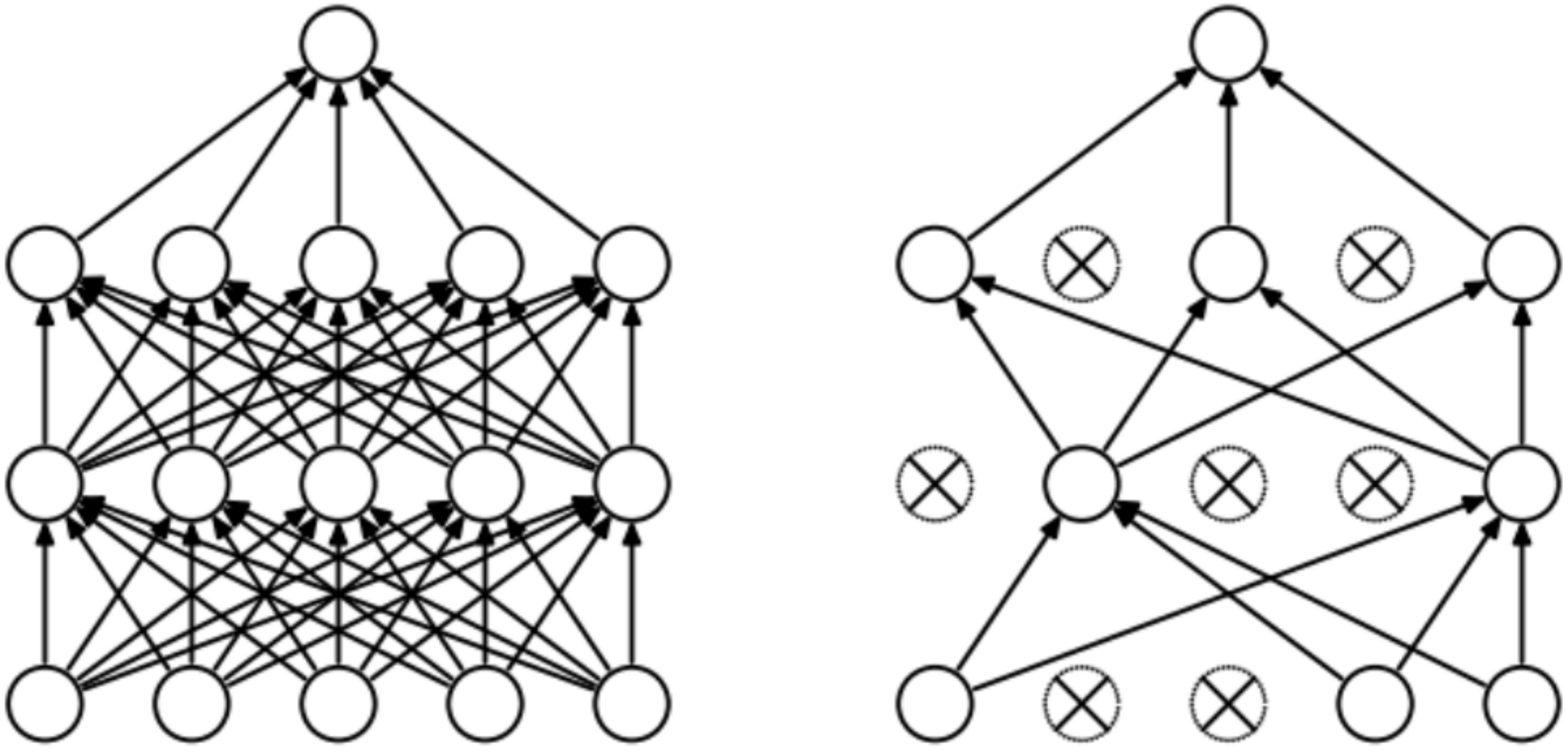
L1: $C(\mathbf{w}, b) = \sum_{i=1}^{m} L(\hat{y}_i, y_i) + \frac{\lambda}{2m} ||\mathbf{w}||^2$ $\quad ||\mathbf{w}||^2 = \sum_{j=1}^{n_x} w_j^2$

L2: $C(\mathbf{w}, b) = \sum_{i=1}^{m} L(\hat{y}_i, y_i) + \frac{\lambda}{2m} ||\mathbf{w}||$ $\quad ||\mathbf{w}|| = \sum_{j=1}^{n_x} |w_j|$

- **Dropout**

Dropout Regularization

- Reduce high dependency on few nodes (act like random forest)



(a) Standard Neural Net (b) After applying dropout.

Codes for Regularization

- **L2 :**

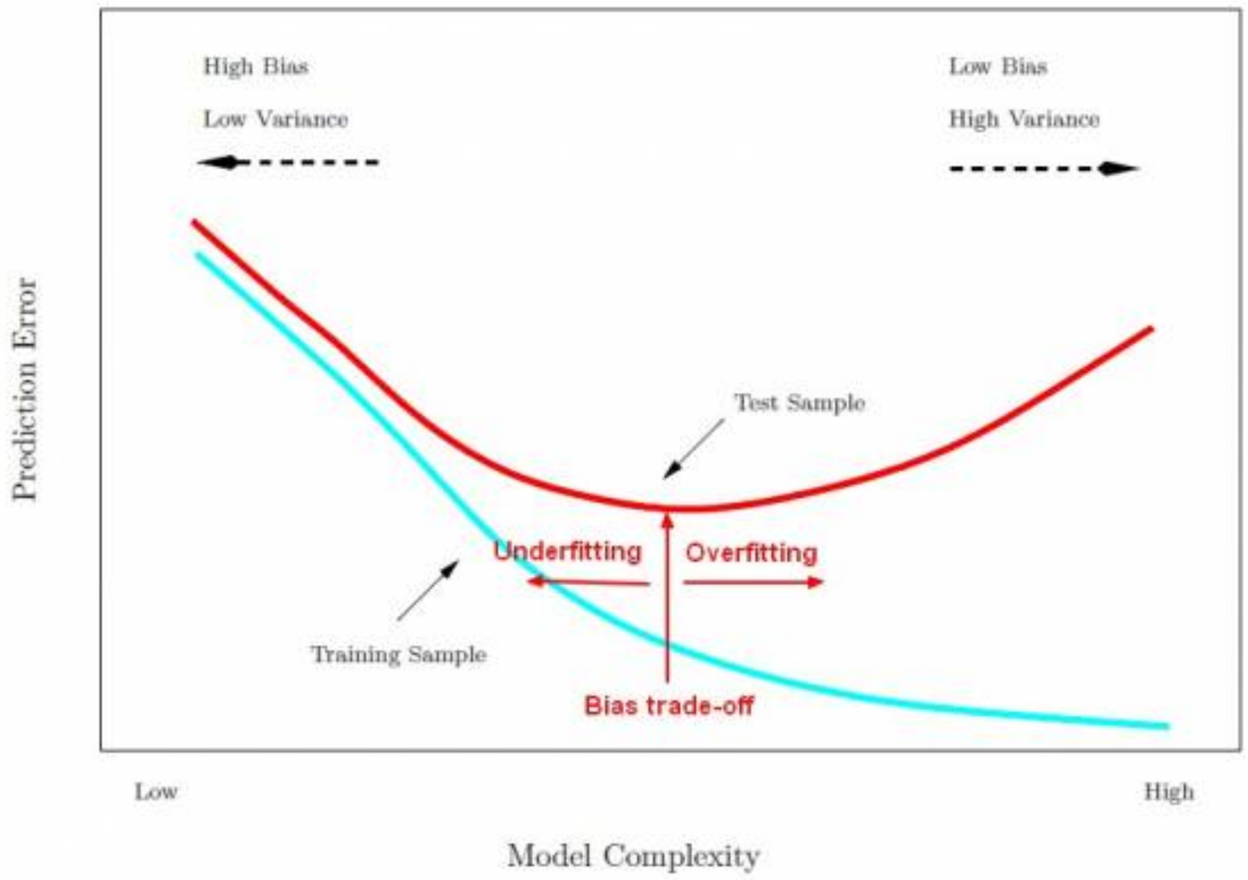
```
l2_model = keras.models.Sequential([
    keras.layers.Dense(16, kernel_regularizer=keras.regularizers.l2(0.001),
                       activation='relu', input_shape=(NUM_WORDS,)),
    keras.layers.Dense(16, kernel_regularizer=keras.regularizers.l2(0.001),
                       activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

- **Dropout :**

```
dpt_model = keras.models.Sequential([
    keras.layers.Dense(16, activation='relu', input_shape=(NUM_WORDS,)),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(16, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1, activation='sigmoid')
])
```

Early Stopping

- Reduce high dependency on few nodes (act like random forest)



Codes for Early Stopping

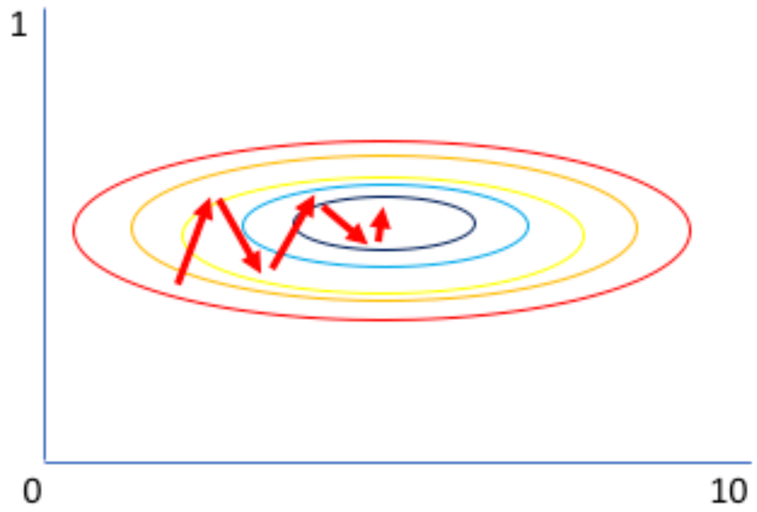
```
from keras.callbacks import EarlyStopping
earlystop= EarlyStopping(monitor='val_acc', patience=3)
```

- **monitor** denotes the quantity that needs to be monitored and ‘val_err’ denotes the validation error.
- **Patience** denotes the number of epochs with no further improvement after which the training will be stopped.

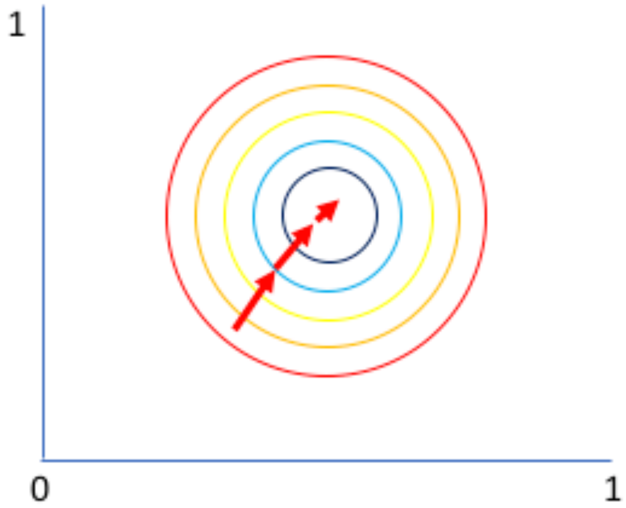
```
model.fit(train_images, train_labels, epochs=10,
          validation_data = (test_images, test_labels),
          callbacks=[earlystop] )
```

Normalizing inputs

Why normalize?



Gradient of larger parameter dominates the update

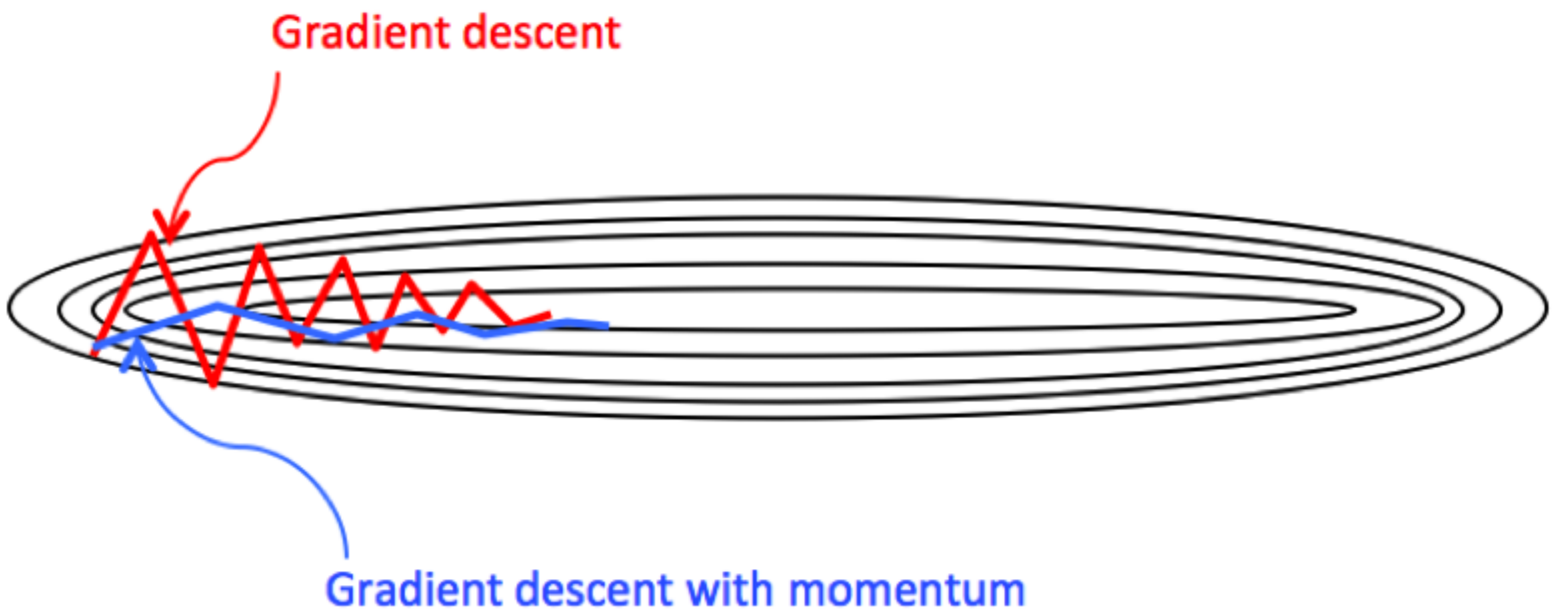


Both parameters can be updated in equal proportions

Practice

https://www.tensorflow.org/tutorials/keras/classification

Gradient Decent with Momentum



- RMSprop, and Adam optimizer

Learning rate decay / adaptive learning rate

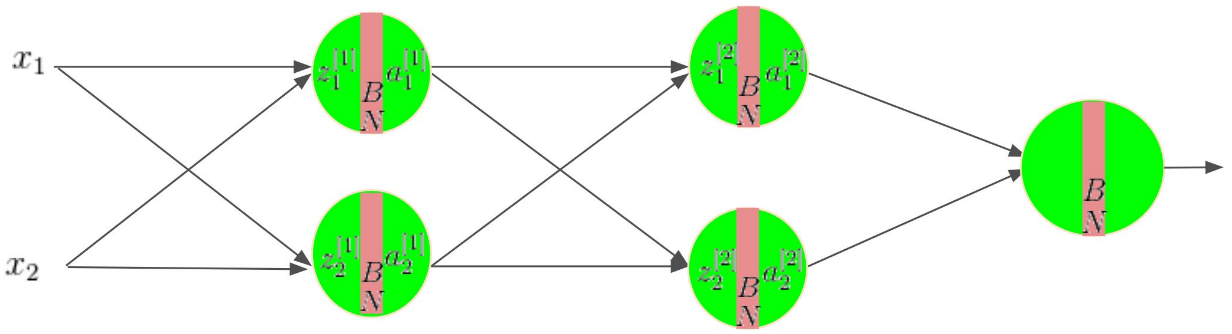
- It is often useful to reduce learning rate as the training progresses

- Use **learning rate schedules** or **adaptive learning rate methods**

https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1

Batch Normalization

- Normalizing inputs to speed up learning



$$z^{[l]} = W^{[l]} a^{[l-1]} \longrightarrow$$

$$\mu^{[l]} = \frac{1}{m} \sum_i z^{[l](i)}$$

$$\sigma^{[l]2} = \frac{1}{m} \sum_i (z^{[l](i)} - \mu^{[l]})^2$$

$$z_{norm}^{[l](i)} = \frac{z^{[l](i)} - \mu^{[l]}}{\sqrt{\sigma^{[l]2} + \epsilon}}$$

$$\tilde{z}^{[l](i)} = \gamma^{[l]} z_{norm}^{[l](i)} + \beta^{[l]}$$

$$\longrightarrow a^{[l]} = g^{[l]}(\tilde{z}^{[l]})$$

- Use keras.layer.BatchNormalization()

Practice

https://www.tensorflow.org/tutorials/keras/classification

Thank you!
Q & A