Training Process Improvements: Part 2

Lecture 8

Subir Varma

Improved SGD

Learning Rate Adaptation

- AdaGrad
- RMSProp

Faster Convergence

- Momentum
- Nesterov Momentum

Combination Technique

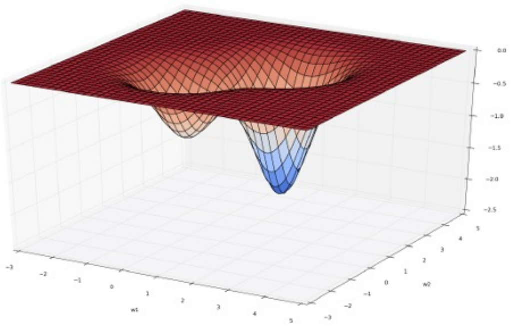
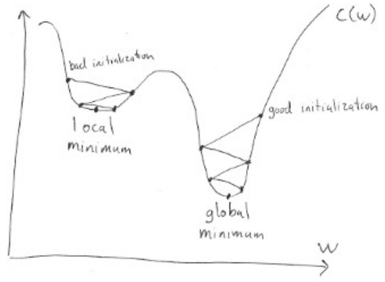
- Adam

https://cs231n.github.io/neural-networks-3/

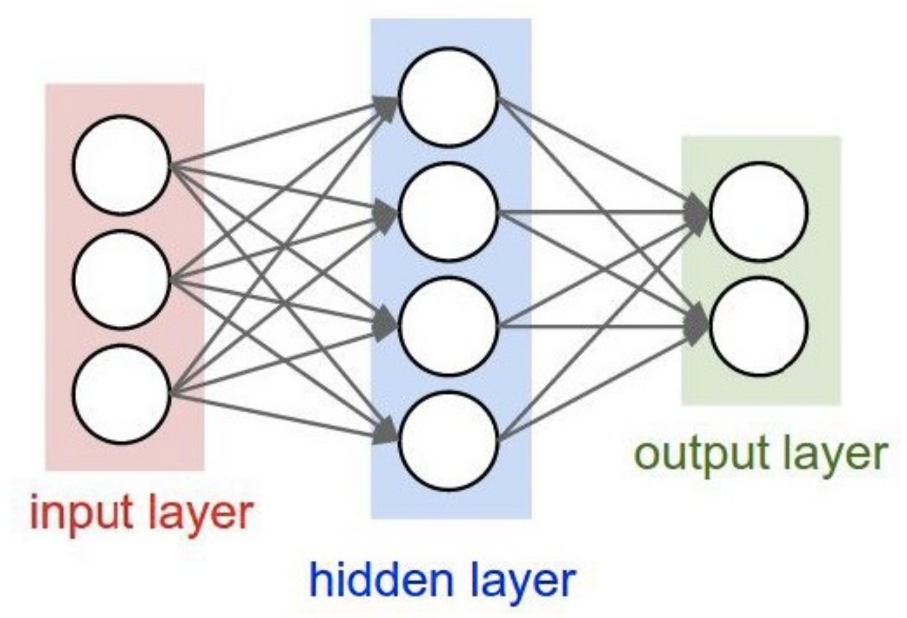
Weight Initialization

Weight Initialization

- A very important topic
- One of the reasons early Neural Networks did not work very well was due to lack of understanding on how to initialize the weights

Initializing all Weights to the Same Value



All nodes compute to the same value, even after backprop
- Need to break symmetry

$$a_j = \sum_{j=1}^{N} w_{ij} z_j + b_j$$

$$\acute{\delta}_j^{(r)} = f'(a_j^{(r)})(\sum_k w_{kj}^{(r+1)} \delta_k^{(r+1)})$$

Random Initialization

$$a_j = \sum_{j=1}^{N} w_{ij} z_j + b_j$$

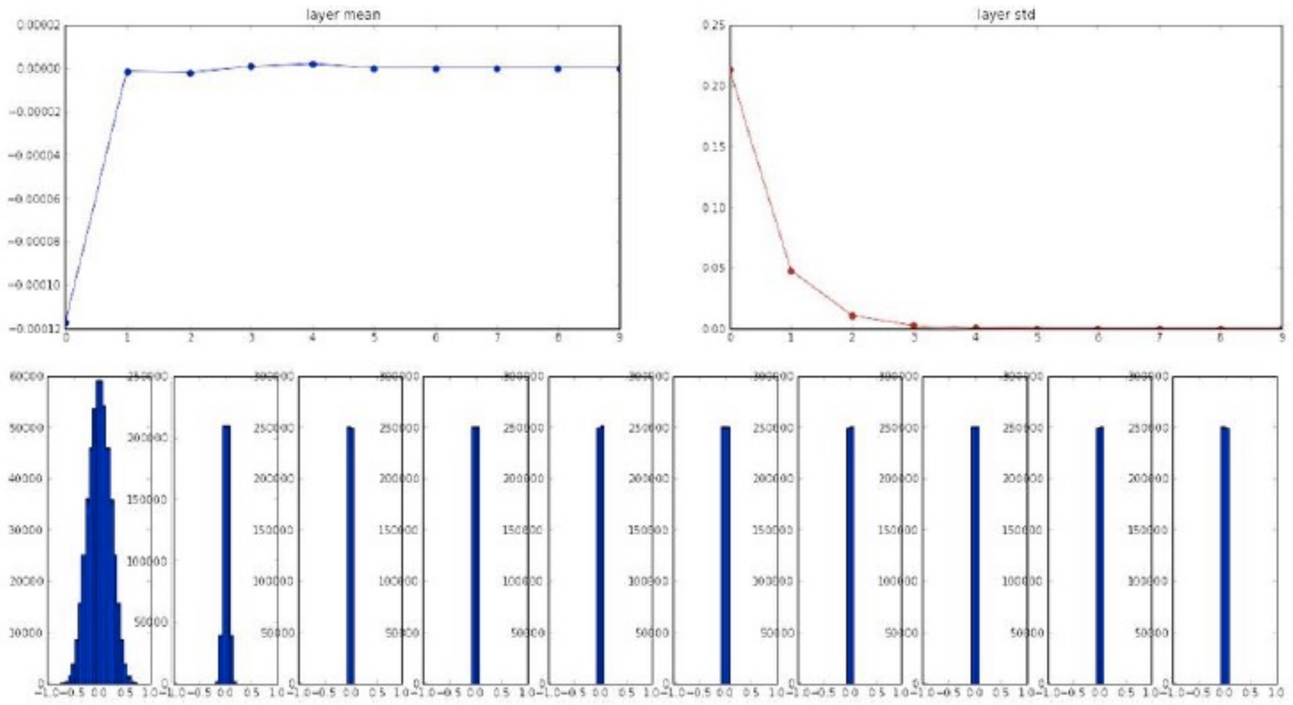
$$A^{n+1} = W A^n + B$$
$$= W^n A \quad + ..$$

- Initialize using the Gaussian distribution

$$w_{ij} \sim N(0,0.01)$$

- Works ok for smaller networks, but as we add more layers,
  - The activations converge to 0
  - The gradients and weights converge to 0

tanh Activations



Random Initialization with Larger Variance

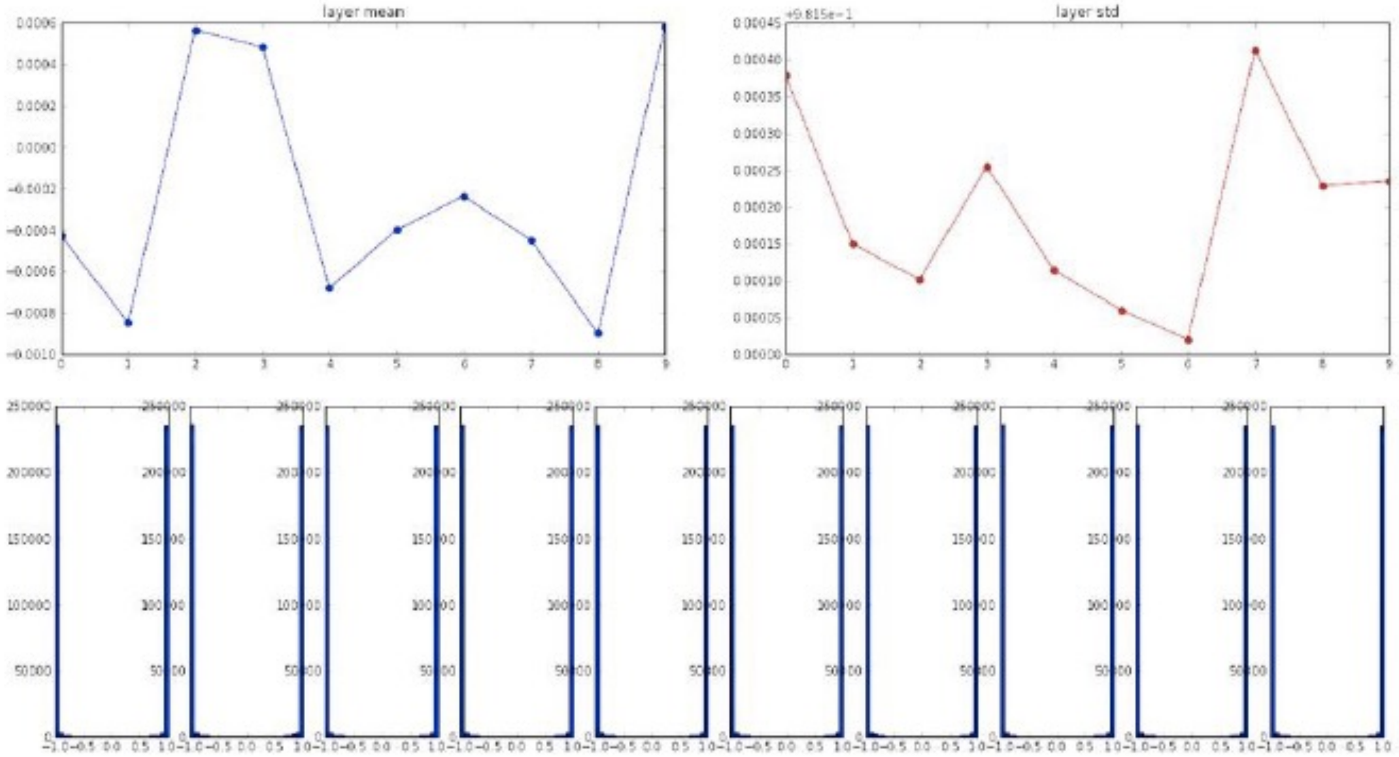
$$a_j = \sum_{j=1}^{N} w_{ij} z_j + b_j$$

- Initialize using the Gaussian distribution

$$w_{ij} : N(0,1)$$

- All the tanh units become saturated at +1 or −1, gradients at zero

tanh Activations



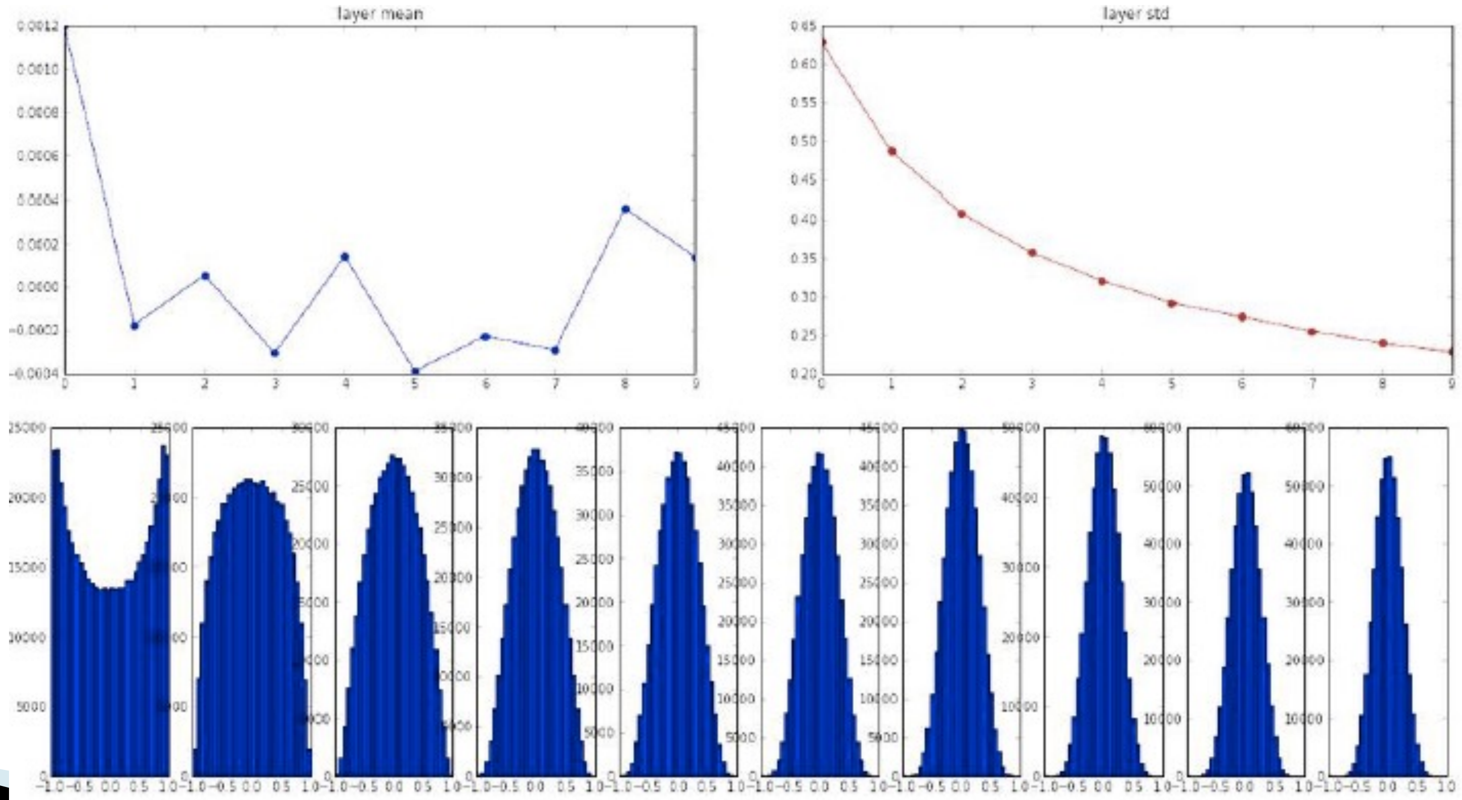
Xavier Initialization

$$a_j = \sum_{j=1}^{N} w_{ij} z_j + b_j$$

- Initialize using the Gaussian distribution

$$w_{ij} : N(0, \frac{1}{\sqrt{n_{in}}})$$

tanh Activations



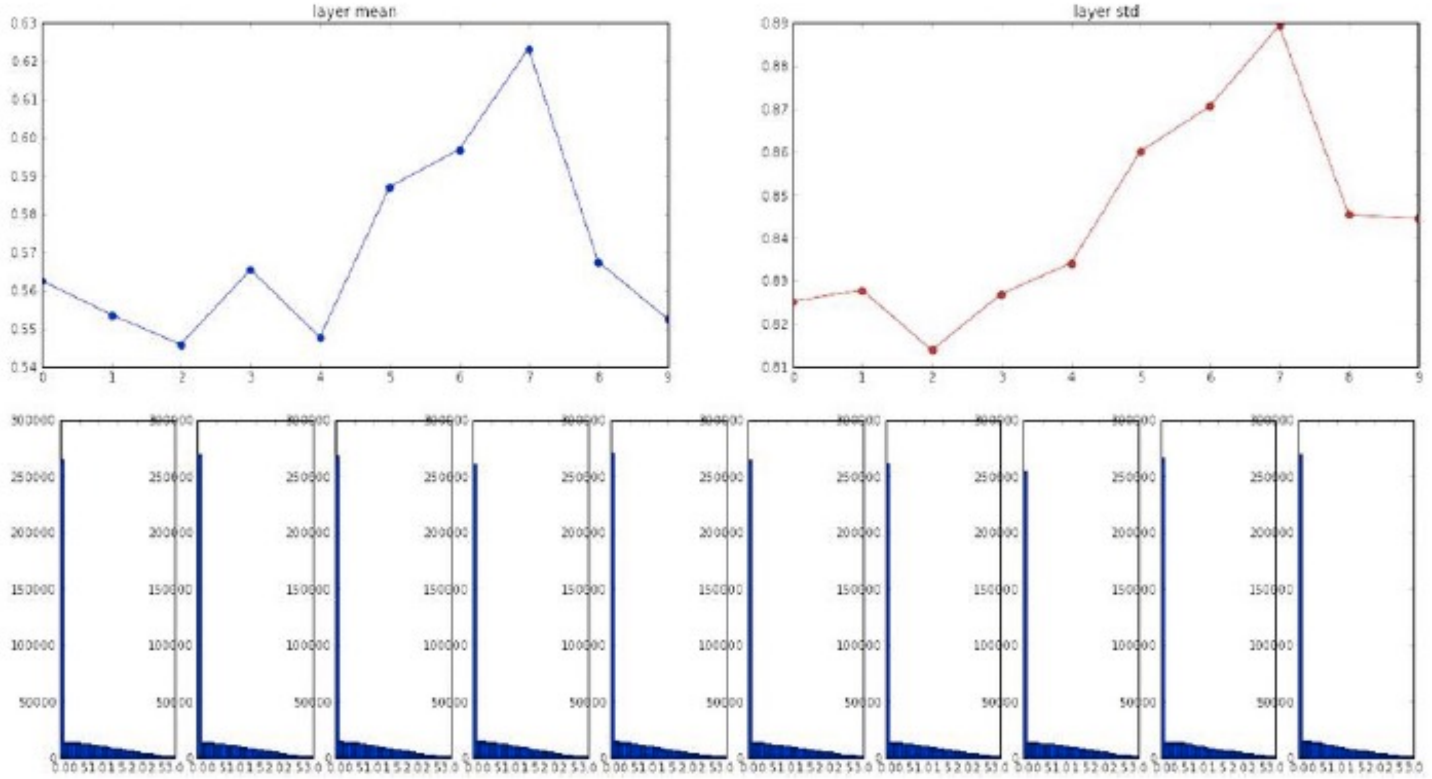
Xavier–He Initialization: Works for ReLU

- Initialize using the Gaussian Distribution

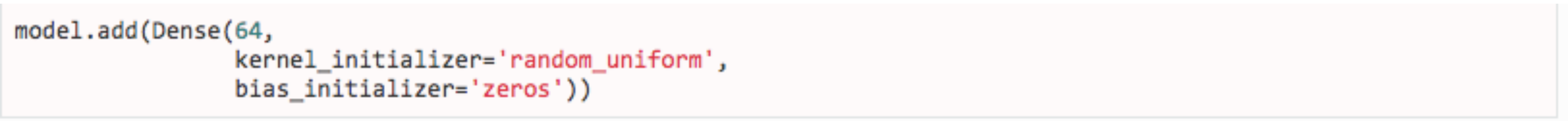
$$w_{ij} : N(0, \frac{1}{\sqrt{n_{in}/2}})$$

ReLU halves the variance.

In order to compensate, increase the incoming variances by a factor of 2



Weight Initialization in Keras



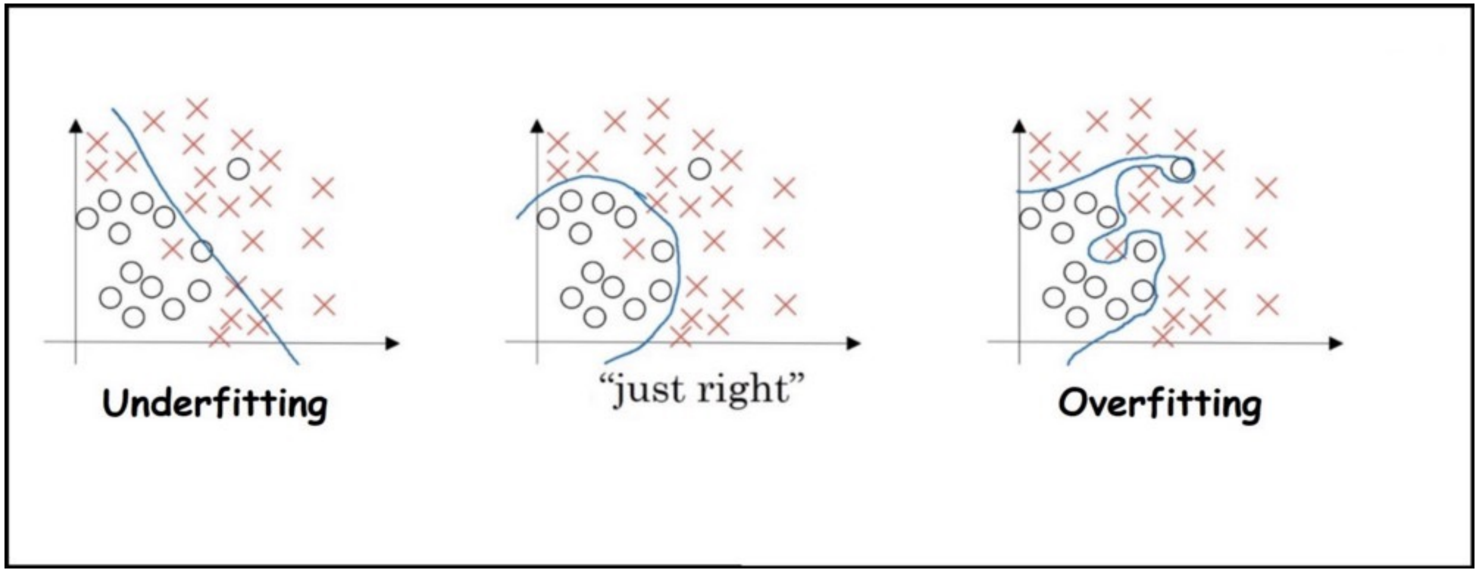
```
model.add(Dense(64,
                kernel_initializer='random_uniform',
                bias_initializer='zeros'))
```

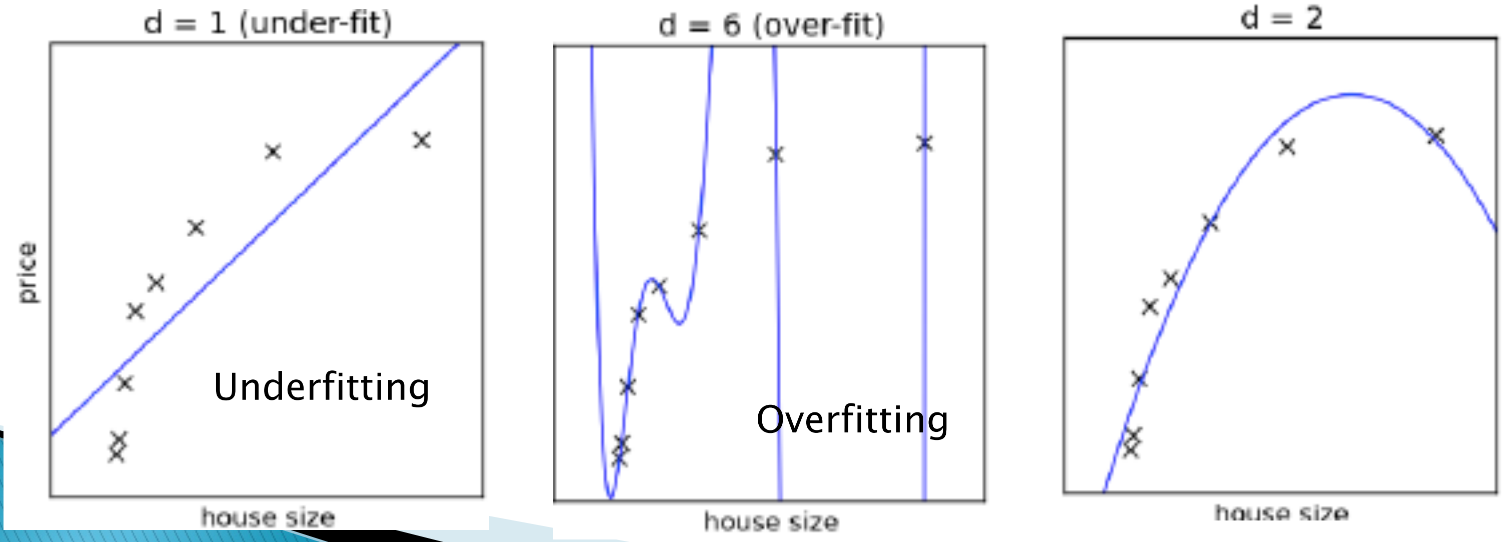
Information available at:

https://keras.io/initializers/

Model Underfitting and Overfitting

Underfitting and Overfitting



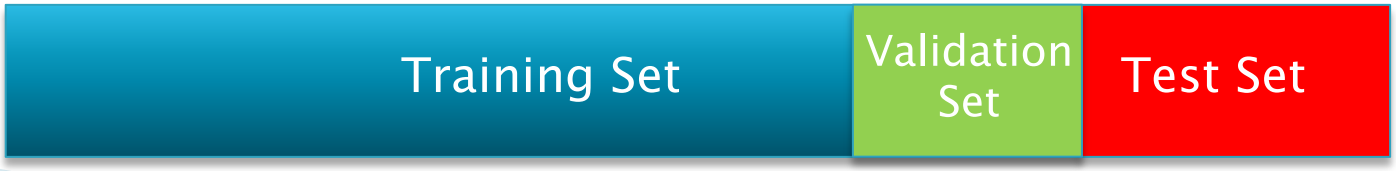


Underfitting and Overfitting

Once the DLN model has been trained, its true test is how well it is able to classify inputs that it has not seen before (i.e. Test Data), which is also known as its Generalization Ability.

There are two kinds of problems that can afflict ML models in general:

1. Even after the model has been fully trained such that its training error is small, it exhibits a high test error rate. This is known as the problem of Overfitting
2. The training error fails to come down in-spite of several epochs of training. This is known as the problem of Underfitting.



Data Complexity and Model Capacity

Data Complexity: Degree of non-linearity in the Data

Model Capacity

- Degree of non-linearity that the model can capture
- Model Capacity is proportional to the number of layers (and nodes per layer). An increase in model non-linearity increases capacity.

Data Complexity and Model Capacity

Ideally: Data Complexity = Model Capacity

<u>Data Complexity</u>: Degree of non-linearity in the Data

<u>Model Capacity</u>

- Degree of non-linearity that model can capture
- Model Capacity is proportional to the number of layers (and nodes per layer)

Causes of Underfitting and Overfitting

Underfitting:

Data Complexity > Model Capacity

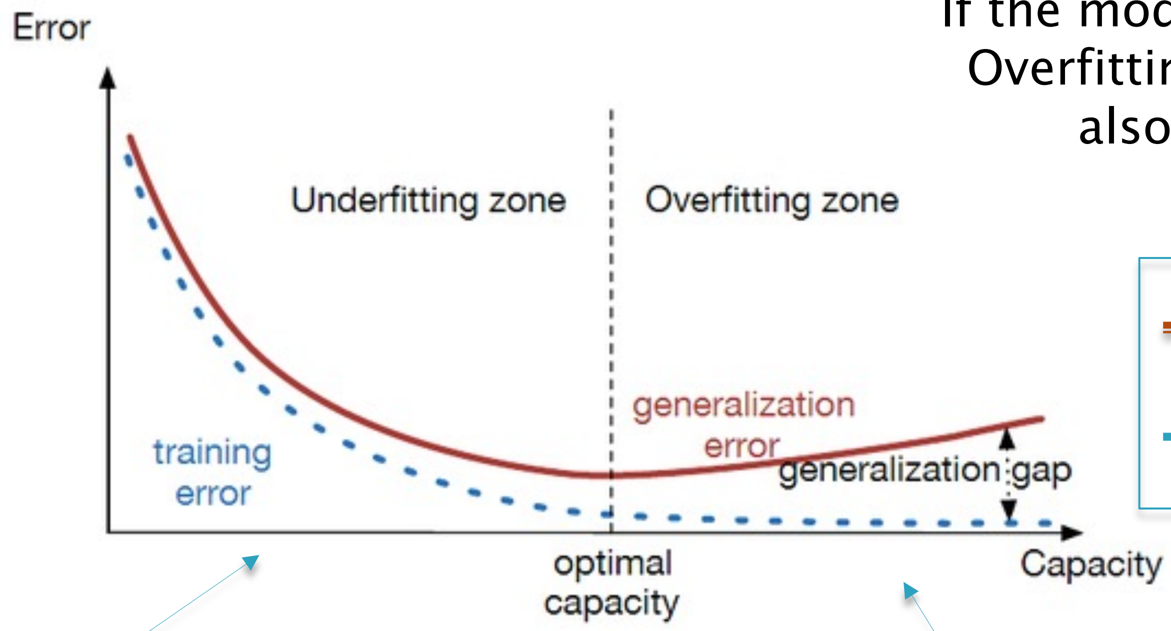
Overfitting:

Data Complexity < Model Capacity, OR

In-sufficient Training Data

Overfitting and Underfitting

If the model never enters the Overfitting Zone, then that also is a problem



▬	Validation
– – –	Training

Model hasn't modeled all the relevant patterns in the Training Data

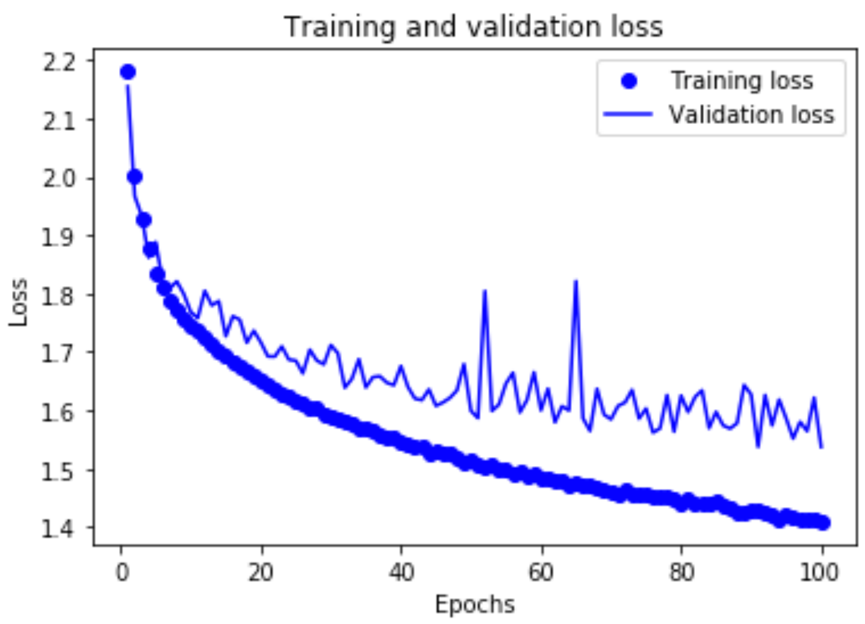
Model is learning patterns that are specific to the Training Data but irrelevant to the Test Data

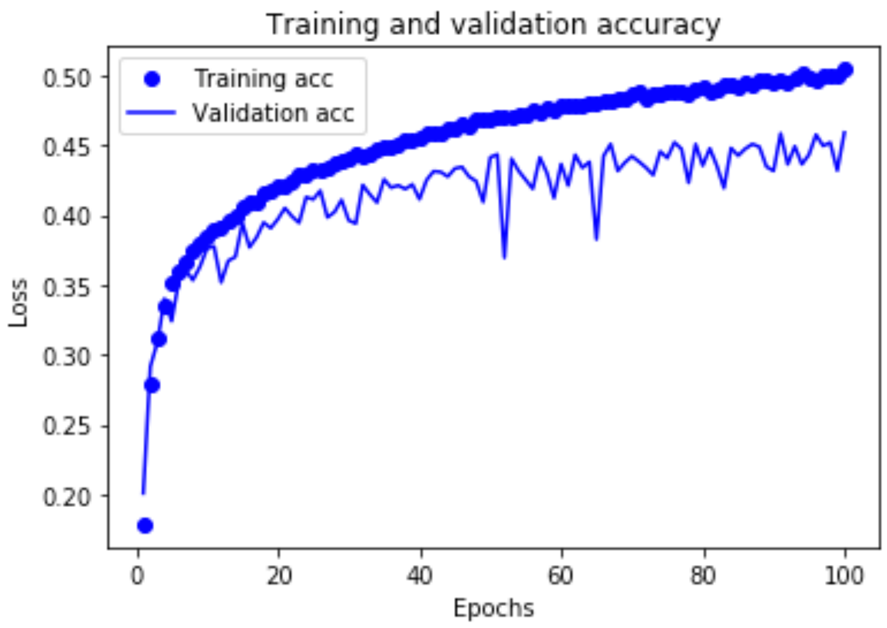
The Underfitting Problem

- Symptom: Model never enters the Overfitting Zone OR Low Training Data Accuracy and/or High Loss, even after multiple epochs of training.
- Cause: The degree of non-linearity in the training data is higher than the amount of non-linearity the Network is capable of capturing.
- Solution: The modeler can increase the model capacity by increasing the number of hidden layers and/or adding more nodes per hidden layer.

If these steps fail to solve the problem, then it points to bad quality/mis-labeled training data.

Detecting Underfitting

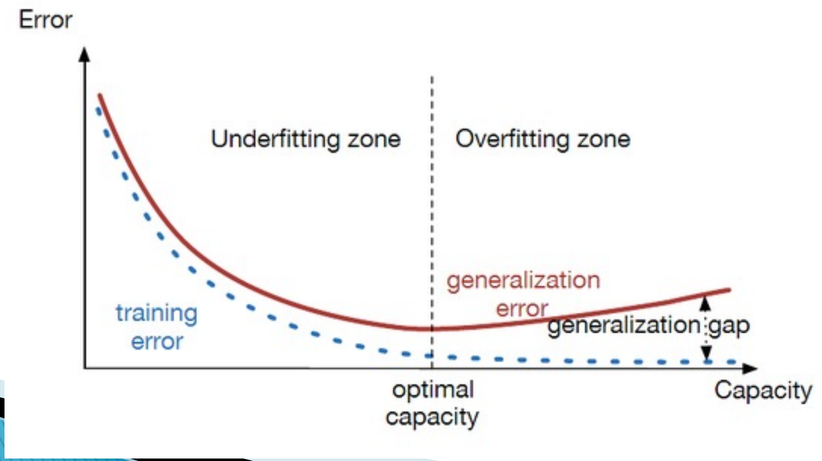




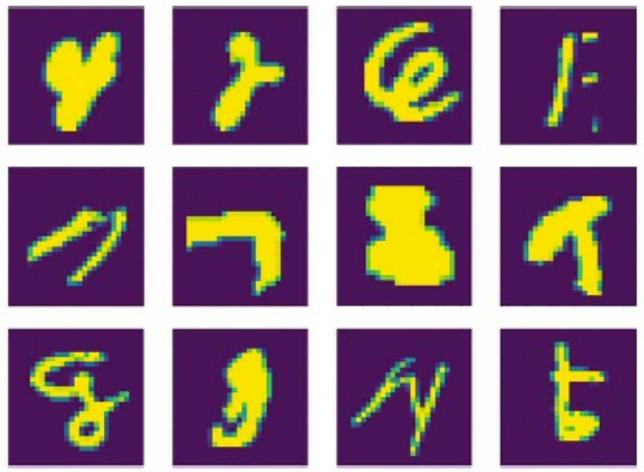
CIFAR-10 Dataset using a Dense Feed Forward Model

Overfitting

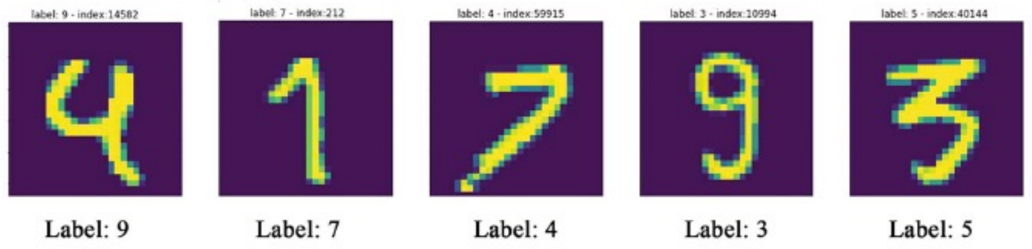
- Overfitting is one of the major problems that plagues ML models.
- When this problem occurs, the model fits the training data very well, but fails to make good predictions in situations it hasn't been exposed to before, i.e. the test data.
- It can be triggered by causes such:
  1. Lack of training data, or
  2. The model being too complex for the given amount of training data.



Overfitting due to Ambiguous or Mis-Labeled Training Data

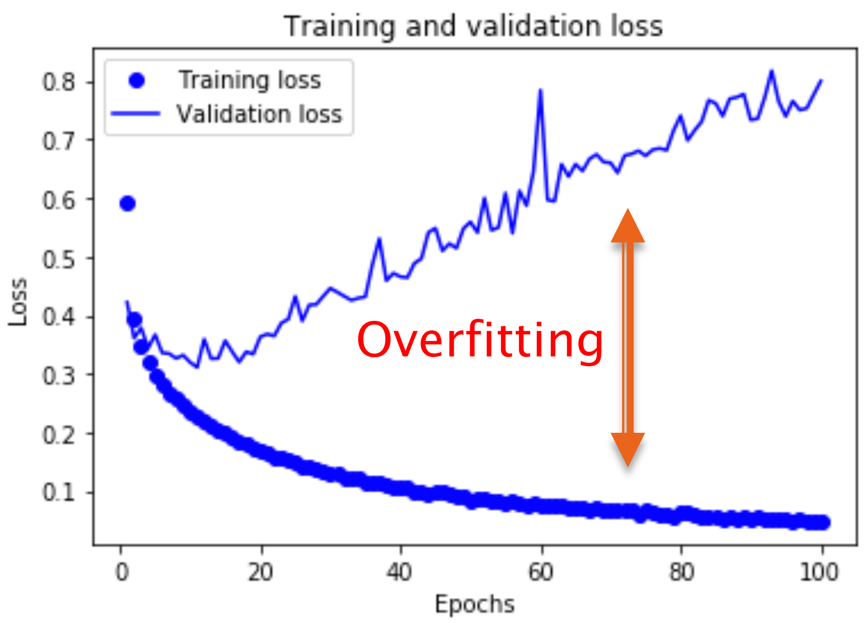


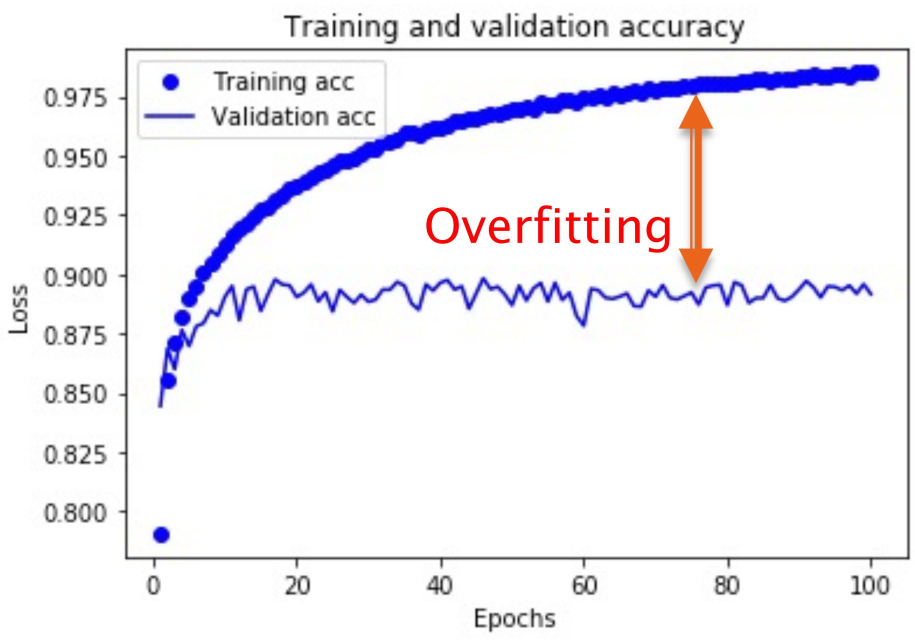
Ambiguous MNIST Training Data



Mis-Labeled MNIST Training Data

Detecting Overfitting



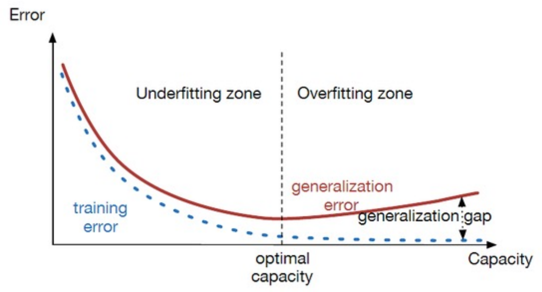


Fashion Dataset using a Dense Feed Forward Network
Single Hidden Layer with 512 nodes

Finding Better Models: Pushing Out the Overfitting Threshold

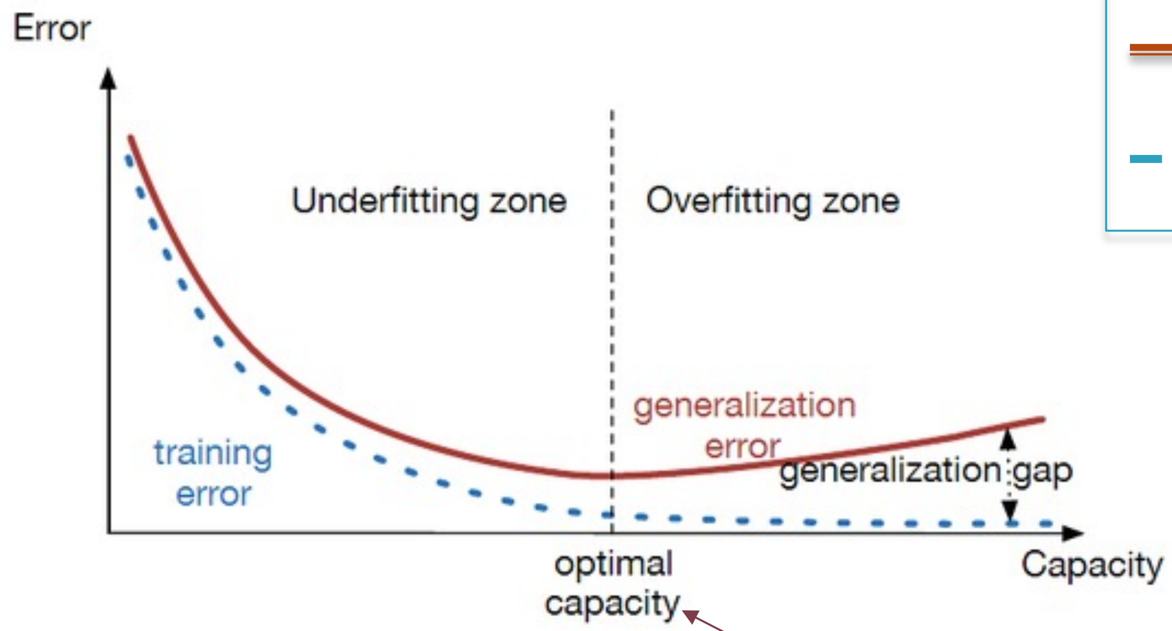
Solutions

1. Decrease Model Capacity
2. Regularization: This is the most common, and also a very effective technique to combat Overfitting.
3. Increasing the amount and quality of training data.
   If this not possible, then
4. Data Augmentation: Increase the amount of training data synthetically by doing various transforms.



Avoiding an Overfitted Model

Early Stopping: Use the Validation Data Set to compute the classification accuracy at the end of each training epoch. Once the accuracy stops increasing, stop the training.

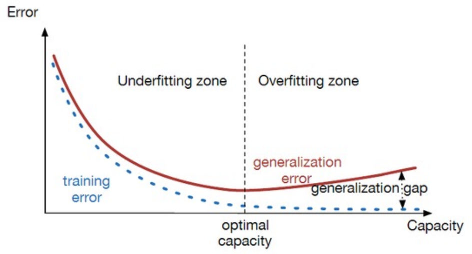


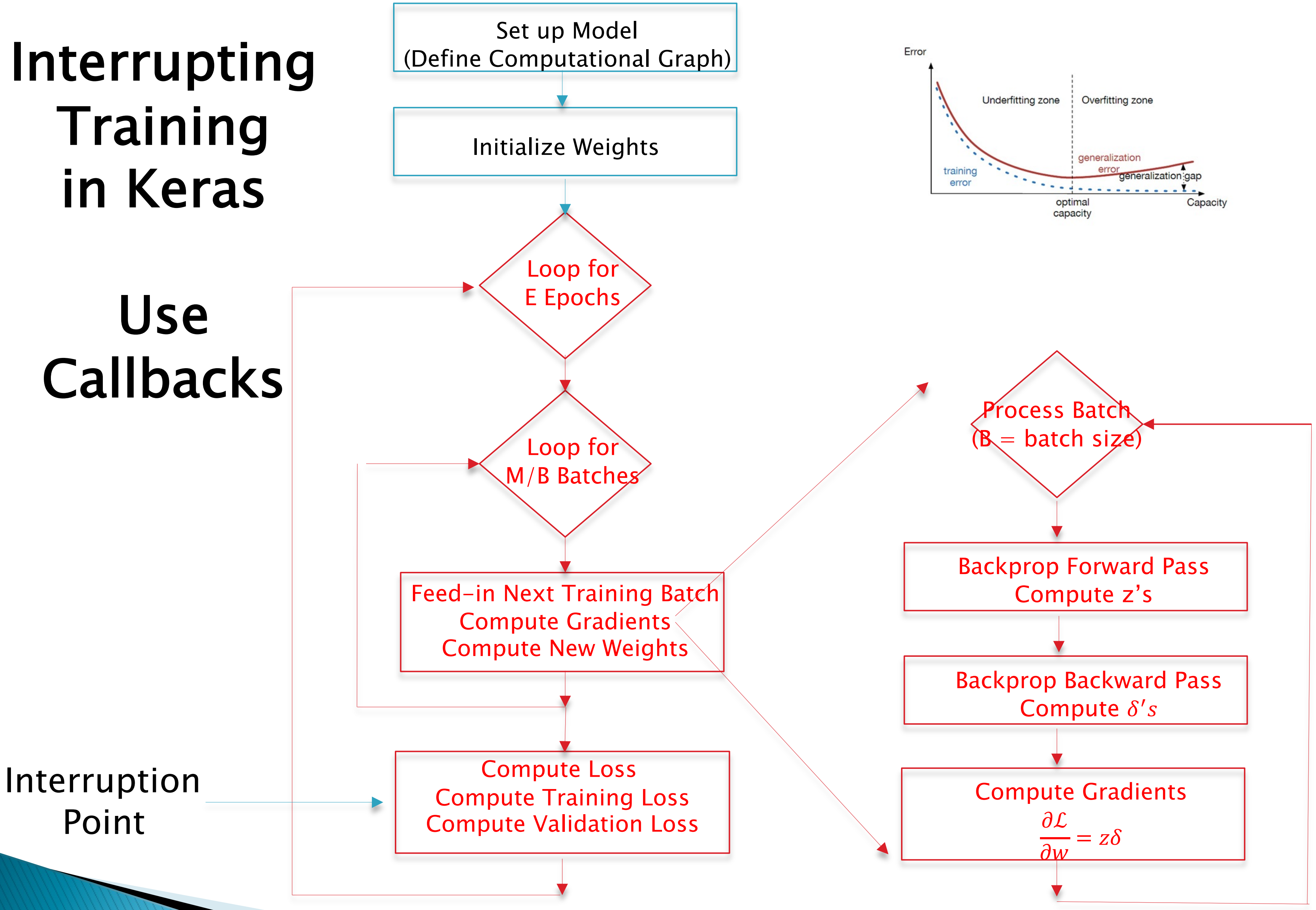
▬	Validation
- - -	Training

Stop the training here

Interrupting Training in Keras

Use Callbacks





```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

The Early Stopping Callback

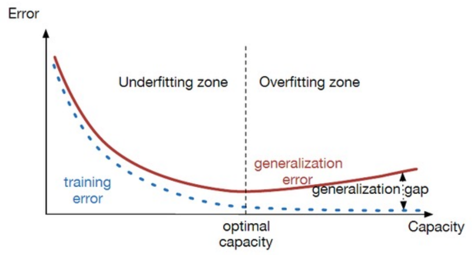
```
Callbacks_list =    [
         keras.callbacks.EarlyStopping(
                    monitor = ‘acc’,
                    patience = 1,
         )
         keras.callbacks.ModelCheckpoint(
                    filepath =‘my_model.h5’,
                    monitor = ‘val_loss’,
                    save_best_only = True
            )
         ]

model.compile(optimizer = ‘rmsprop’,
              loss = ‘binary_crossentropy’,
              metrics = [‘acc’],)

model.fit(x,y,
         epochs = 10,
         batch_size = 32,
         callbacks = callbacks_list,
         validation_data = (x_val, y_val))
```

Interrupt training when accuracy improvement stops

Interrupt training when accuracy stops improving for more than one epoch

Save the model weights after every epoch

Don’t override the saved model file unless val_loss has improved, i.e., keep the weights of the best model seen during training



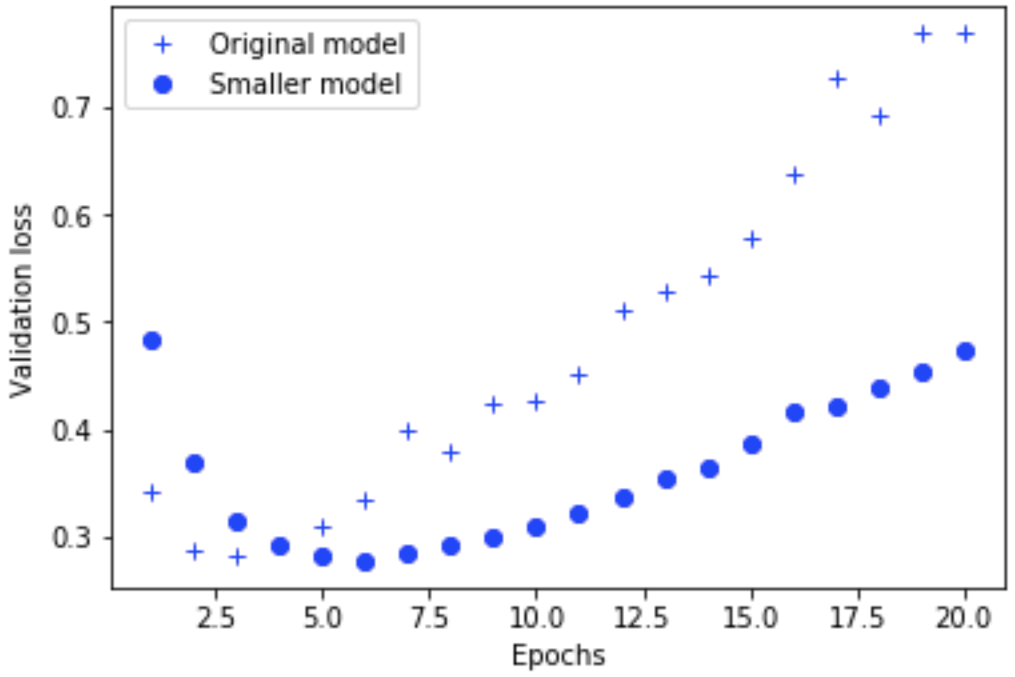
Regularization

What is Regularization

A set of techniques to improve Model Generalization Ability → Move the Overfitting Threshold Further Out.

Prevents the Model from fitting the Training Data too well, in the hope that it will work better with the Test Data

Reducing Model Capacity to Improve Generalization



Original Model: 2 Hidden Layers with 16 nodes each

Smaller Model: 2 Hidden Layers with 4 nodes each

Introducing Regularization

Choose a model with High Capacity,
and then prevent overfitting by doing Regularization

↓

Regularization reduces Model Capacity

Ways to Reduce Model Capacity with Regularization

Regularization algorithms that reduce model complexity by penalizing large weights

Examples: L2 or L1 Regularization

Regularization algorithms that introduce some randomness during the training process, thus preventing the model from fitting the training data too well.

Examples: Dropout Regularization, Batch Normalization, Dropconnect

L2/L1 were inherited from older ML systems and work for smaller models
Dropout and Batch Normalization were designed specifically for Deep Learning systems

L2 Regularization

λ: Regularization Parameter

- L2 Regularization

$$\mathcal{L}_R = \mathcal{L}(cross\ entropy) + \frac{\lambda}{2}\sum_{r=1}^{R+1}\sum_{j=1}^{n_{r-1}}\sum_{i=1}^{n_r}(w_{ij}^{(r)})^2$$

- The effect of regularization is to make the network prefer to learn smaller weights
- The weight update rule becomes:

$$w \leftarrow (1-\eta\lambda)w - \eta\frac{\partial\mathcal{L}}{\partial w}$$

Reduces Model Capacity

What is the effect of λ on Model Capacity?

Effect of L2 Regularization

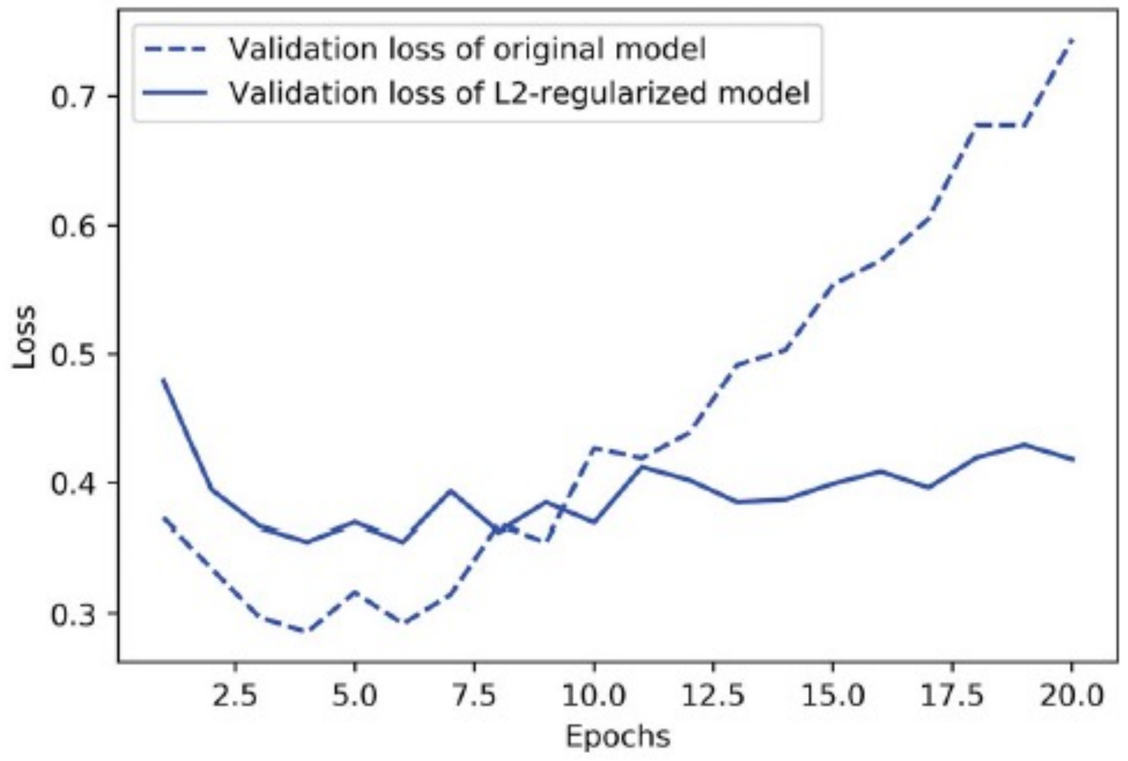


Figure 5.19 Effect of L2 weight regularization on validation loss

L1 Regularization

- L1 Regularization

$$\mathcal{L}_R = \mathcal{L}(cross\ entropy) + \lambda \sum_{r=1}^{R+1} \sum_{j=1}^{n_{r-1}} \sum_{i=1}^{n_r} |w_{ij}^{(r)}|$$

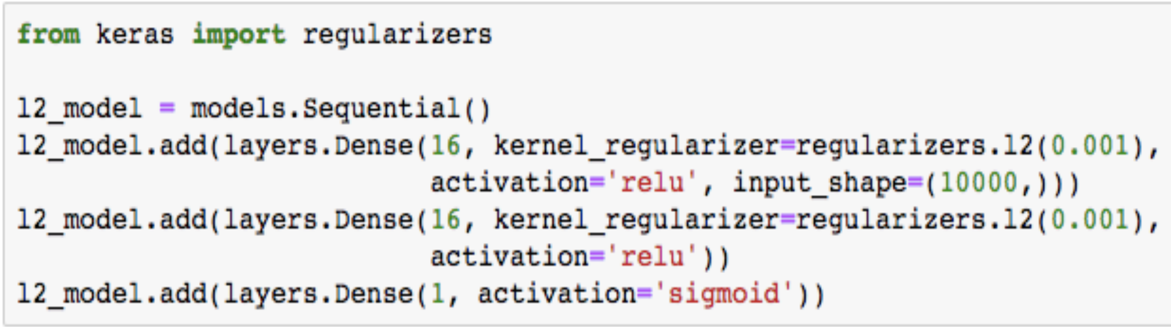
results in the gradient update rule:

$$w \leftarrow w - \eta\lambda\, sgn(w) - \eta \frac{\partial \mathcal{L}}{\partial w}$$

- Results in networks in which the weights are concentrated in a relatively small number of high importance connections, while the other weights are driven towards zero.

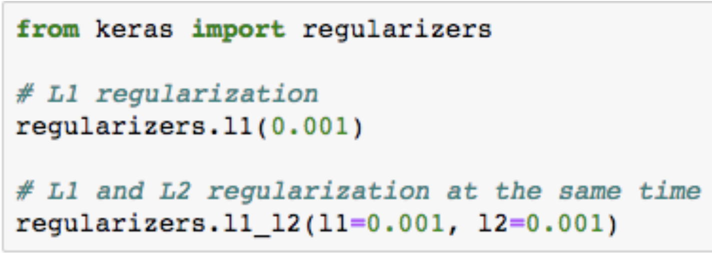
L1/L2 Regularizations work for smaller models

Adding Regularization in Keras



```
from keras import regularizers

l2_model = models.Sequential()
l2_model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                          activation='relu', input_shape=(10000,)))
l2_model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                          activation='relu'))
l2_model.add(layers.Dense(1, activation='sigmoid'))
```

L2 Regularization



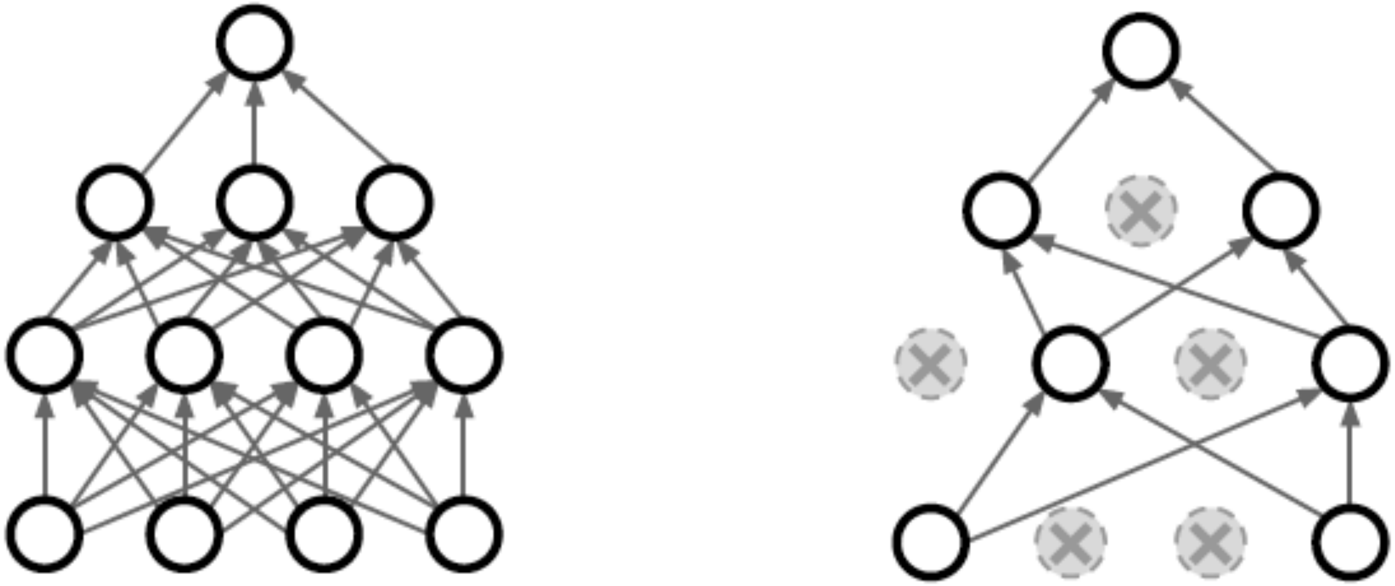
```
from keras import regularizers

# L1 regularization
regularizers.l1(0.001)

# L1 and L2 regularization at the same time
regularizers.l1_l2(l1=0.001, l2=0.001)
```

L1 Regularization

L1 + L2 Regularization

Dropout Regularization

In each forward pass, randomly set some neurons to zero
Probability of retaining is a hyperparameter; 0.5 is common



A different subset of nodes is erased in each iteration of training

Dropout Regularization: Test Time

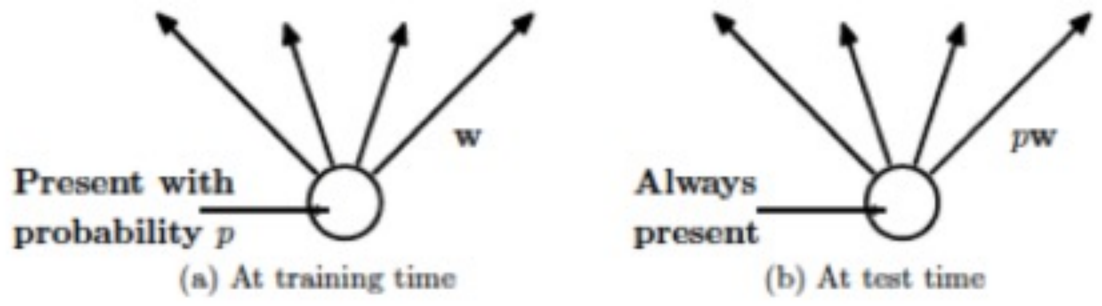
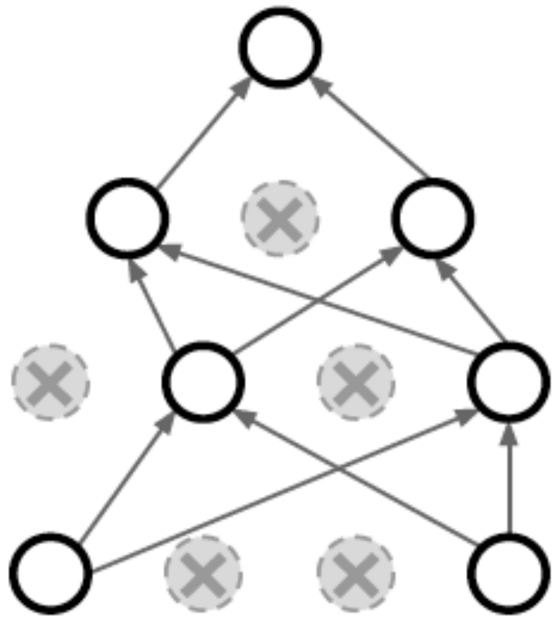


Figure 8.8: Weight Adjustment at Test time

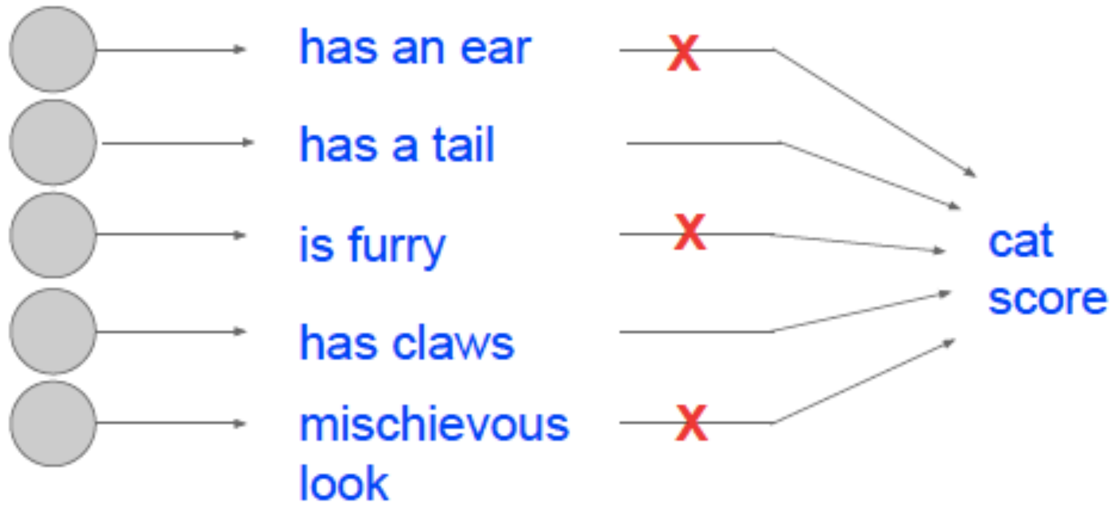
At test time all neurons are active always
=> We must scale the activations so that for each neuron:
<u>output at test time</u> = <u>expected output at training time</u>

Dropout Regularization: Interpretation 1

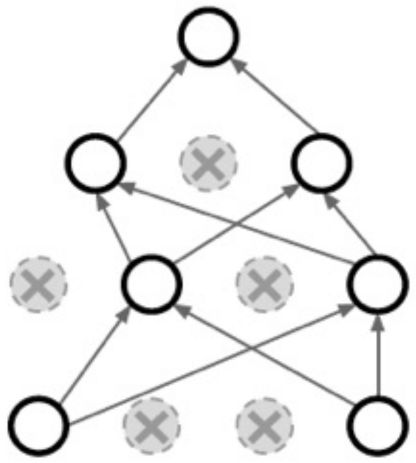
How can this possibly be a good idea?



Forces the network to have a redundant representation;
Prevents co-adaptation of features

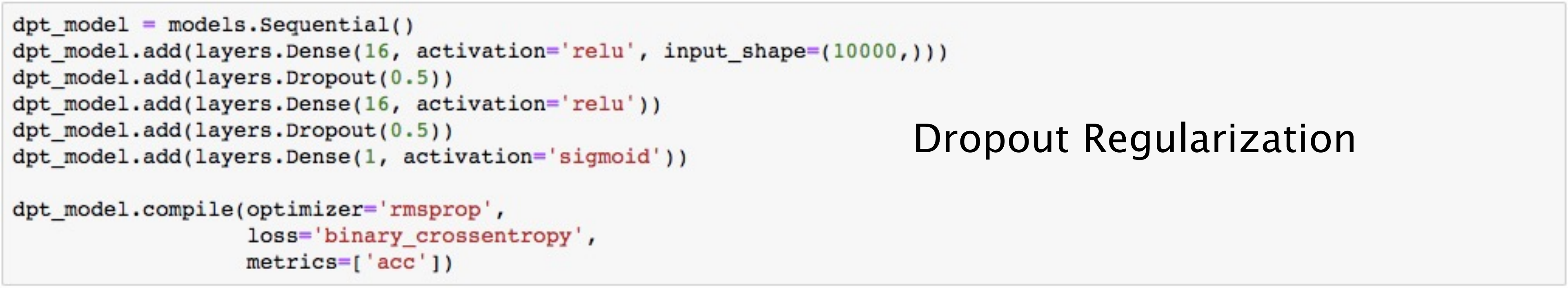


Dropout Regularization: Interpretation 2



- Heuristically when we dropout different sets of neurons, its like we are testing different neural networks
- Hence the dropout procedure is like averaging the effect of a very large number of different networks
- Different networks will overfit in different ways, so the net effect of dropout is to reduce overfitting

Adding Dropout Regularization in Keras



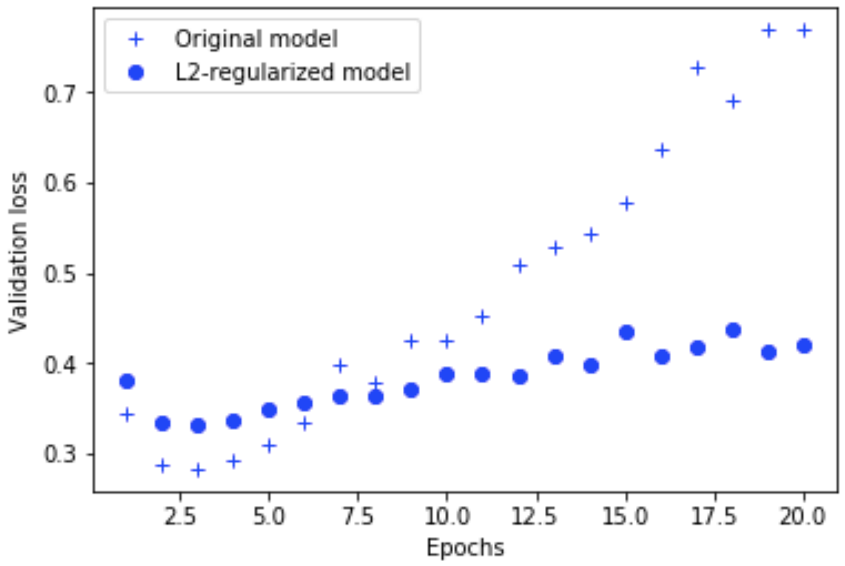
```
dpt_model = models.Sequential()
dpt_model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
dpt_model.add(layers.Dropout(0.5))
dpt_model.add(layers.Dense(16, activation='relu'))
dpt_model.add(layers.Dropout(0.5))
dpt_model.add(layers.Dense(1, activation='sigmoid'))

dpt_model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['acc'])
```

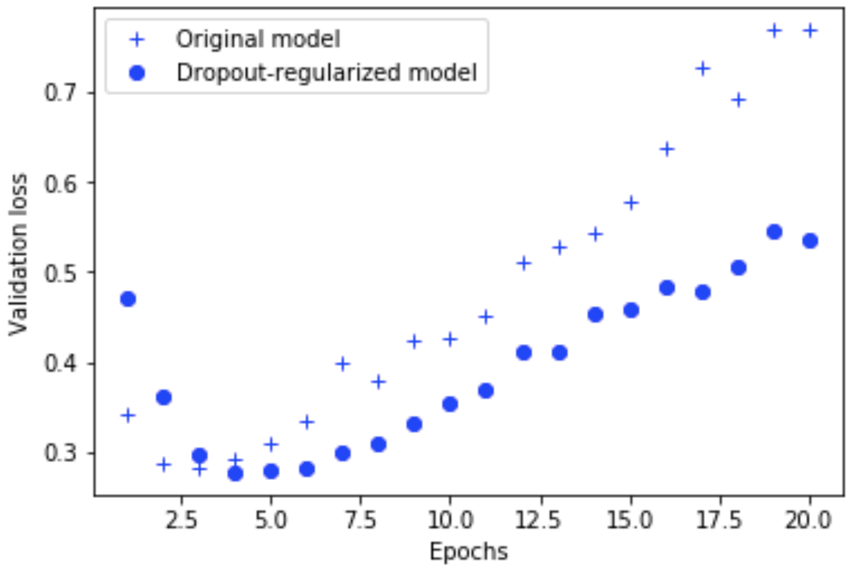
Dropout Regularization

Model Results

L2 Regularization

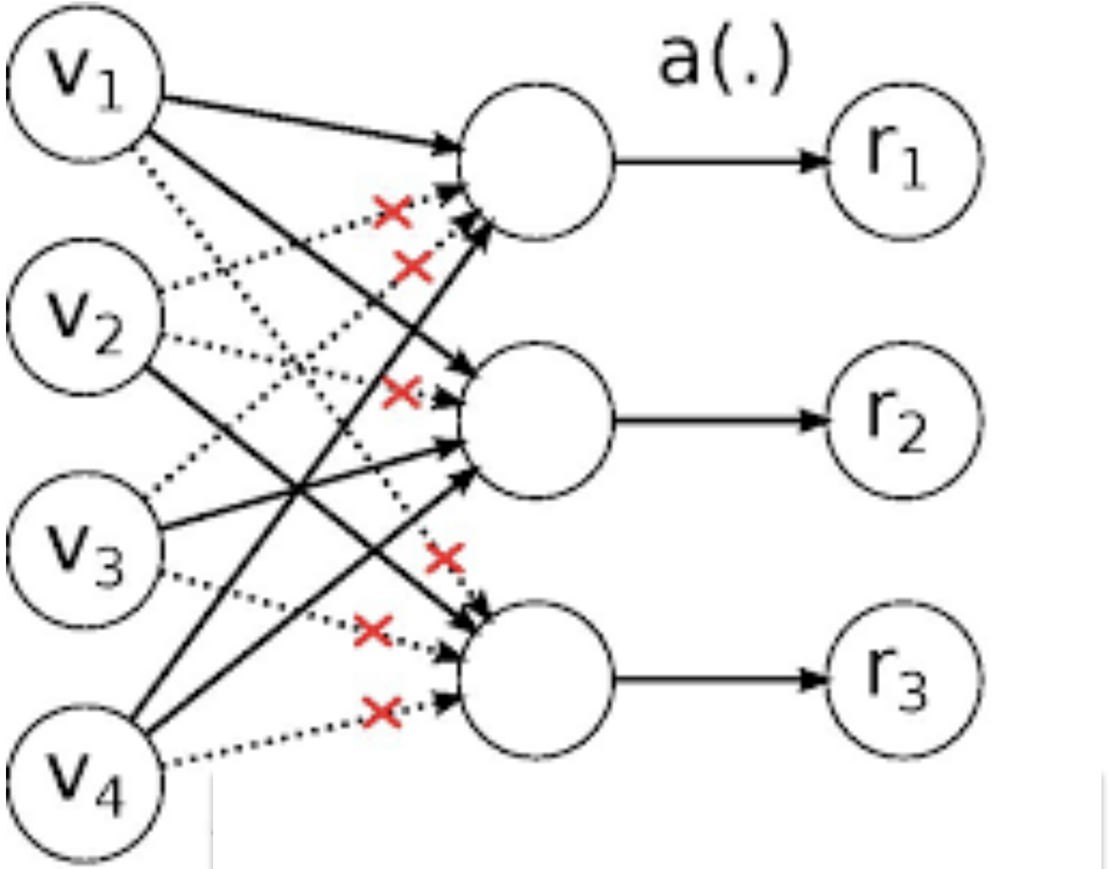
Dropout





IMDB Input, 2 Hidden Layers with 16 Nodes each

Drop Connect Regularization



Regularization: A Common Pattern

Training: Add random noise
Testing: Marginalize over the noise

Examples:
Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth

Further Reading

- Chapters 8: ImprovingModelGeneralization
- Chapter 5 of Chollet