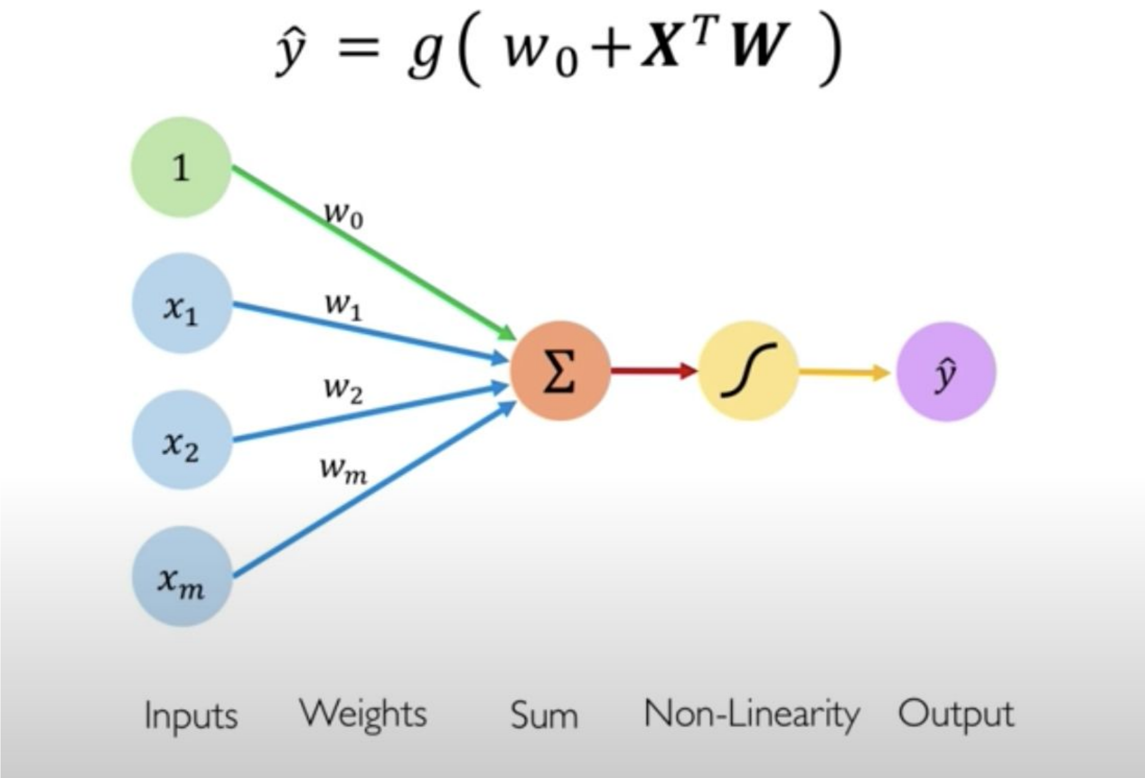
Pytorch Tutorial 3

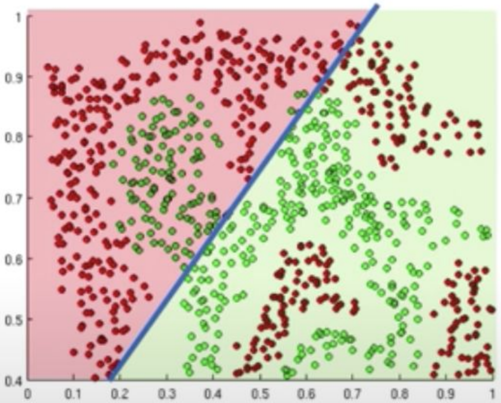
The perceptron



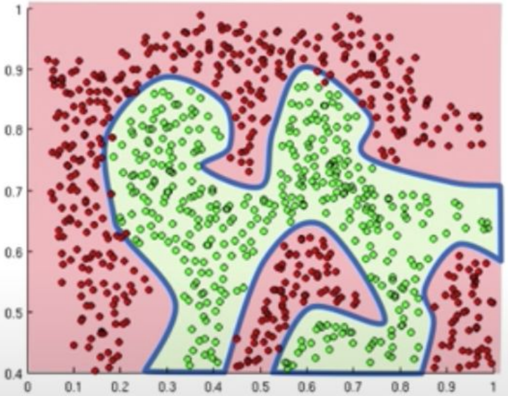
From: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Importance of Activation Functions

The purpose of activation functions is to ***introduce non-linearities*** *into the network*



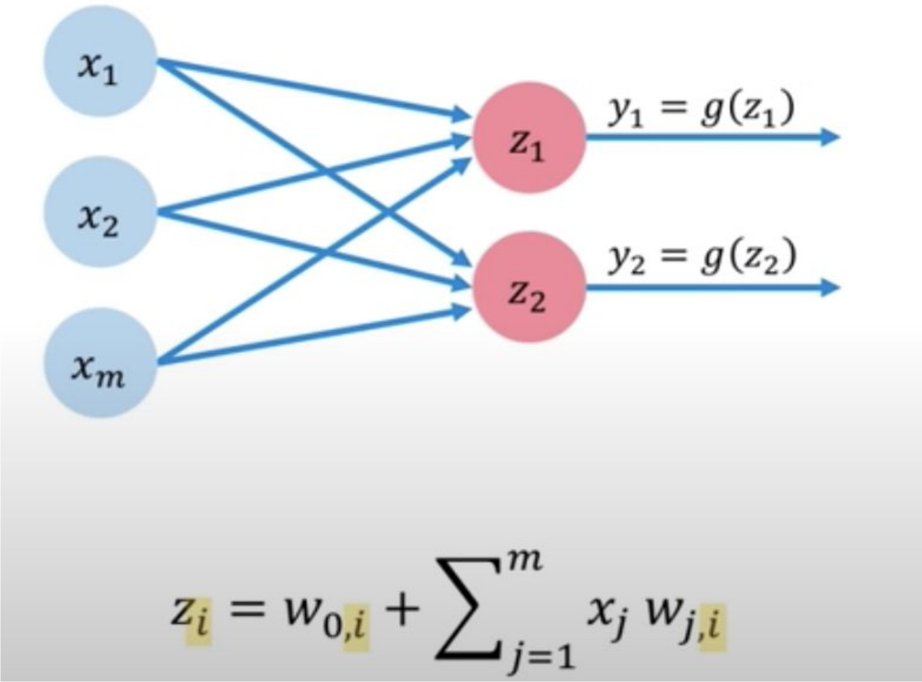
Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

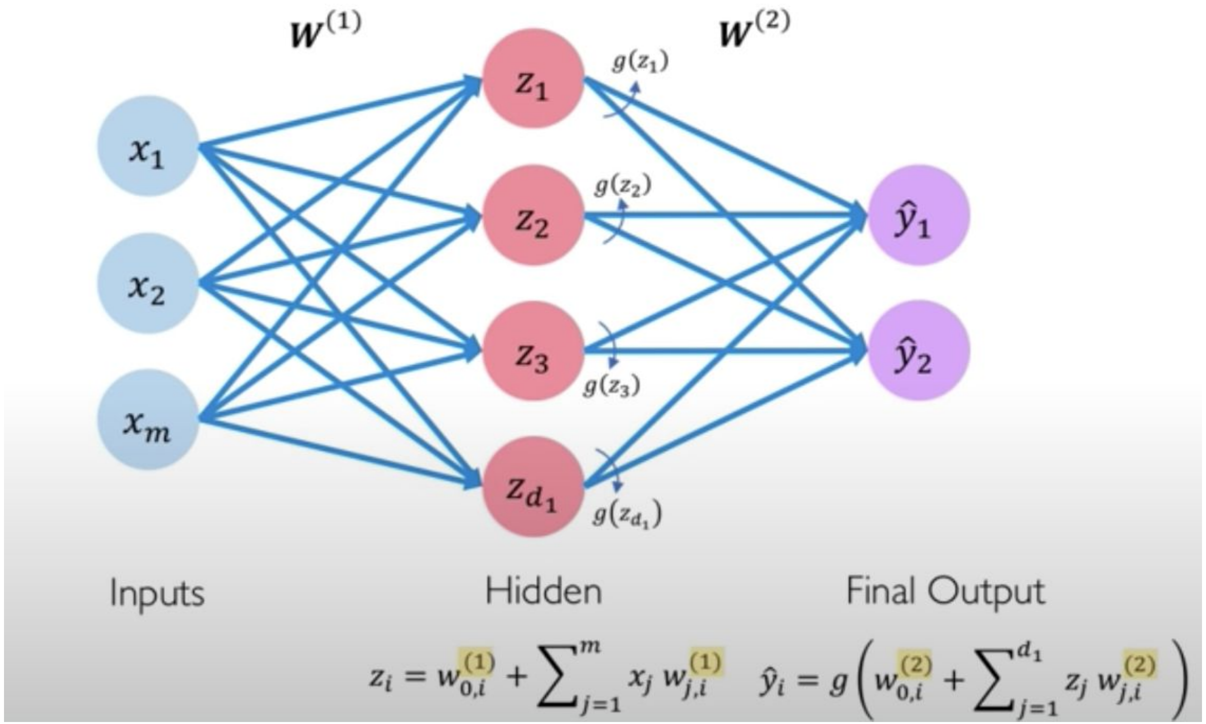
From: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Multi Output Perceptron



From: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Single Layer Neural Network



From: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Diving Deep into Supervised Training

★ Example: Supervised training for a perceptron and a binary classification

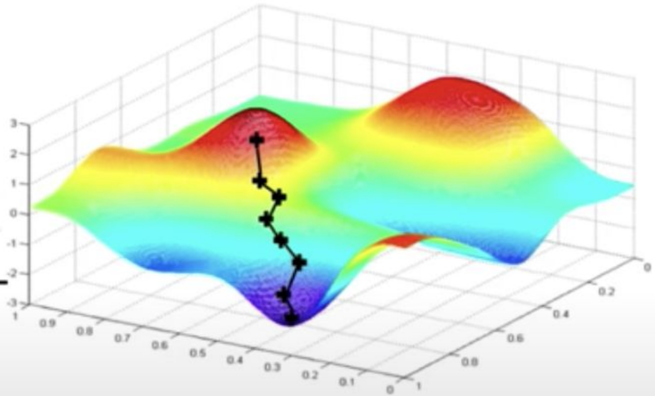
Example: Supervised training for a perceptron and a binary classification

1) Choosing a model
   a) E.g. Perceptron
2) Choosing a loss function
3) Choosing an optimizer
   a) E.g. SGD, Adam

Stochastic Gradient Descent

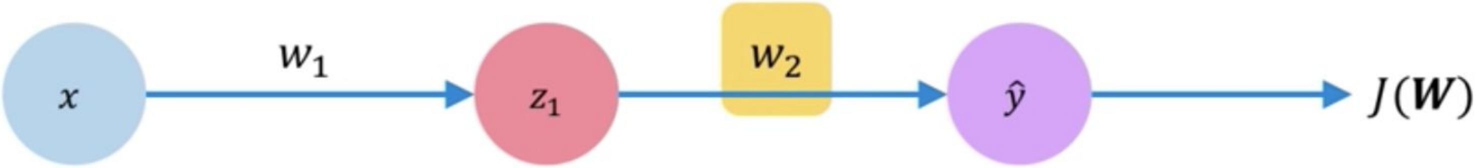
Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



From: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

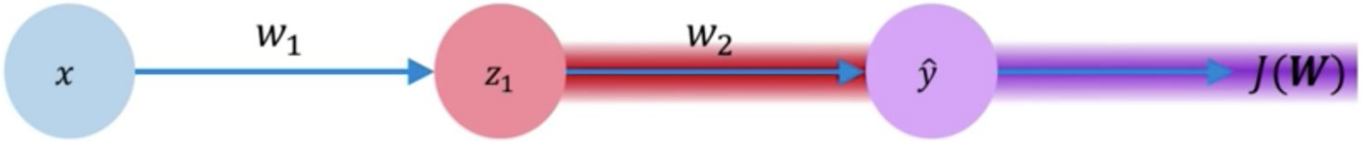
Computing Gradients: Backpropagation



How does a small change in one weight (ex. $\mathbf{w_2}$) affect the final loss $\boldsymbol{J}(\boldsymbol{W})$?

From: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Computing Gradients: Backpropagation

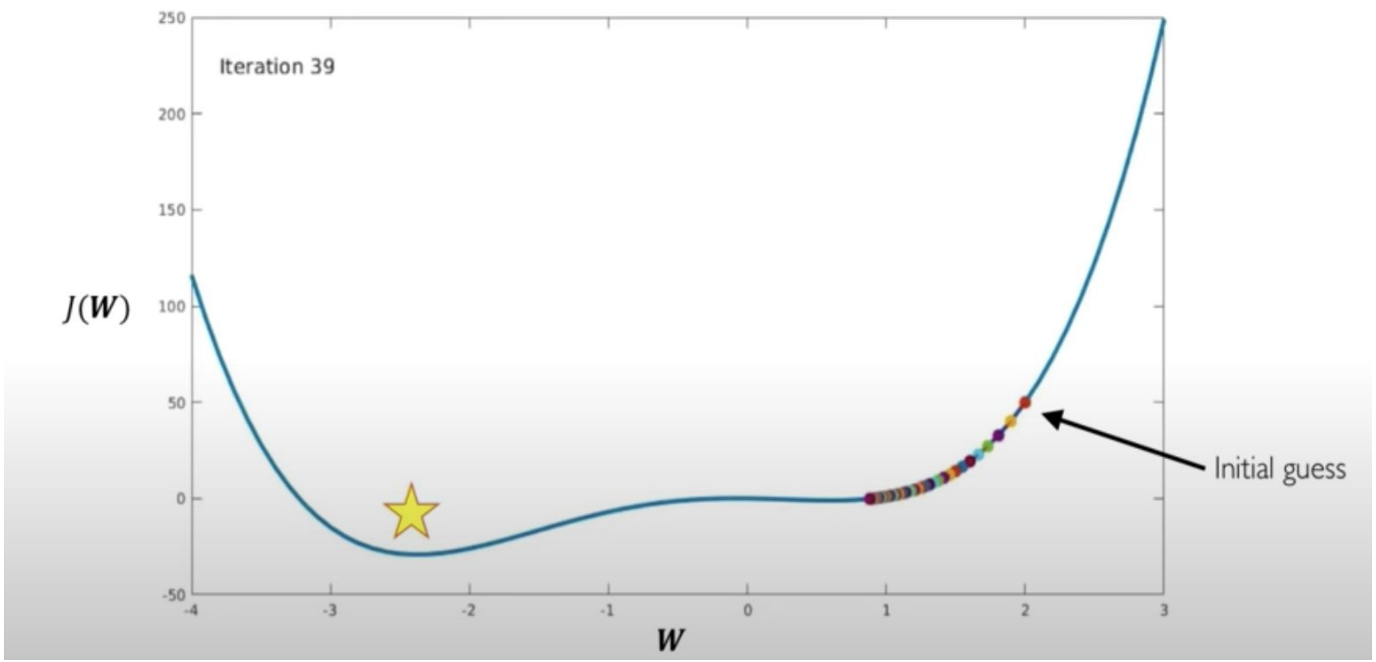


$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

From: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Setting the Learning Rate

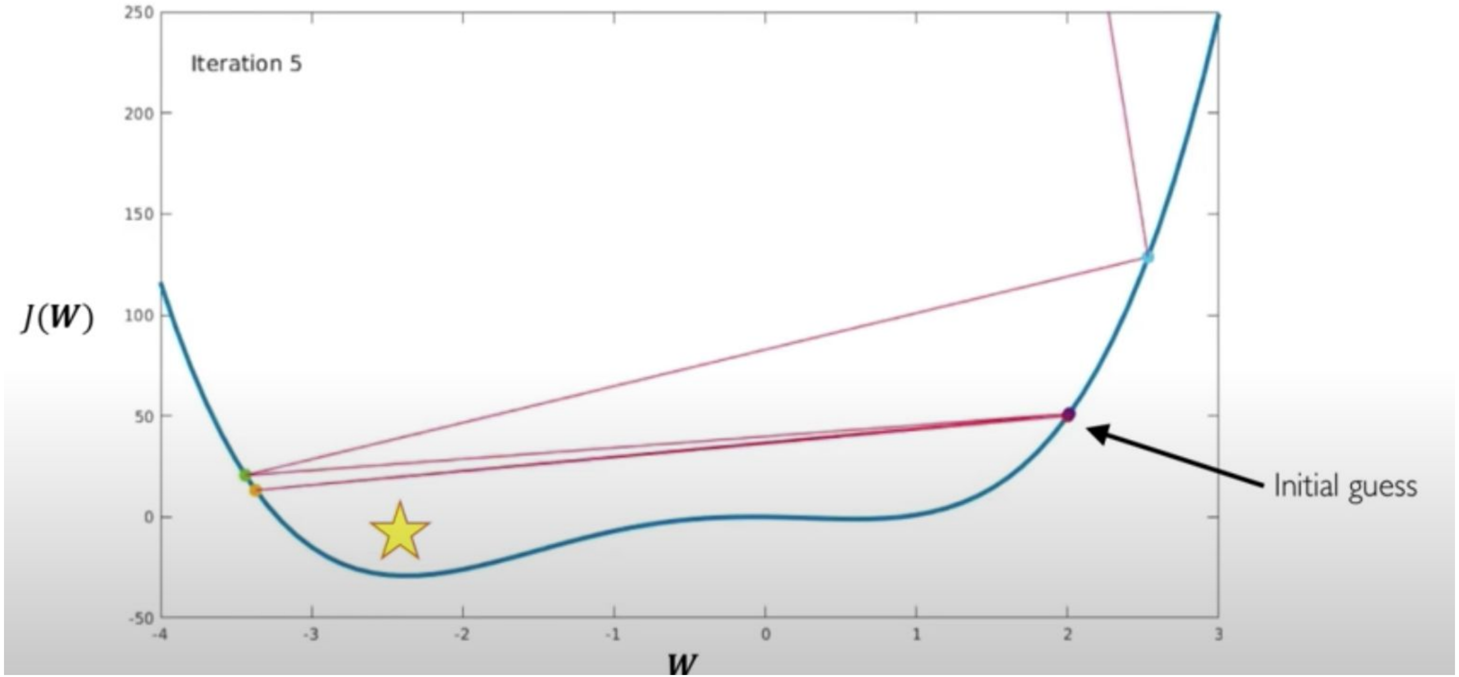
Small learning rate *converges slowly and gets stuck in false local minima*



From: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Setting the Learning Rate

Large learning rates *overshoot, become unstable and diverge*



From: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Adaptive Learning Rates

- Learning rates are no longer fixed
- Can be made larger or smaller depending on:
  - how large gradient is
  - how fast learning is happening
  - size of particular weights
  - etc...

E.g. Adam

From: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Example 3-10 Instantiating the Adam Optimizer

```
import torch.nn as nn
import torch.optim as optim

input_dim = 2
lr = 0.001

perceptron = Perceptron(input_dim=input_dim)
bce_loss = nn.BCELoss()
optimizer = optim.Adam(params=perceptron.parameters(), lr=lr)
```

Example 3-11 A supervised training loop for a perceptron and binary classification

```
#each epoch is a complete pass over the training data
for epoch_i in range(n_epochs):
  #the inner loop is over the batches in the dataset
  for batch_i in range(n_batches):

    #Step 0: Get the data
    x_data, y_target = get_toy_data(batch_size)

    #Step 1: Clear the gradients
    perceptron.zero_grad()

    #Step 2:Compute the forward pass of the model
    y_pred = perceptron(x_data, apply_sigmoid=True)

    #Step 3: Compute the loss value that we wish to optimize
    loss = bce_loss(y_pred, y_target)

    #Step 4: Propagate the loss signal backard
    loss.backward()

    #Step 5: Trigger the optmizer to perform one update
    optimizer.step()
```

Auxiliary Training concepts

Auxiliary Training concepts: Evaluation Metrics

- **Accuracy**, precision, recall, F1, etc.

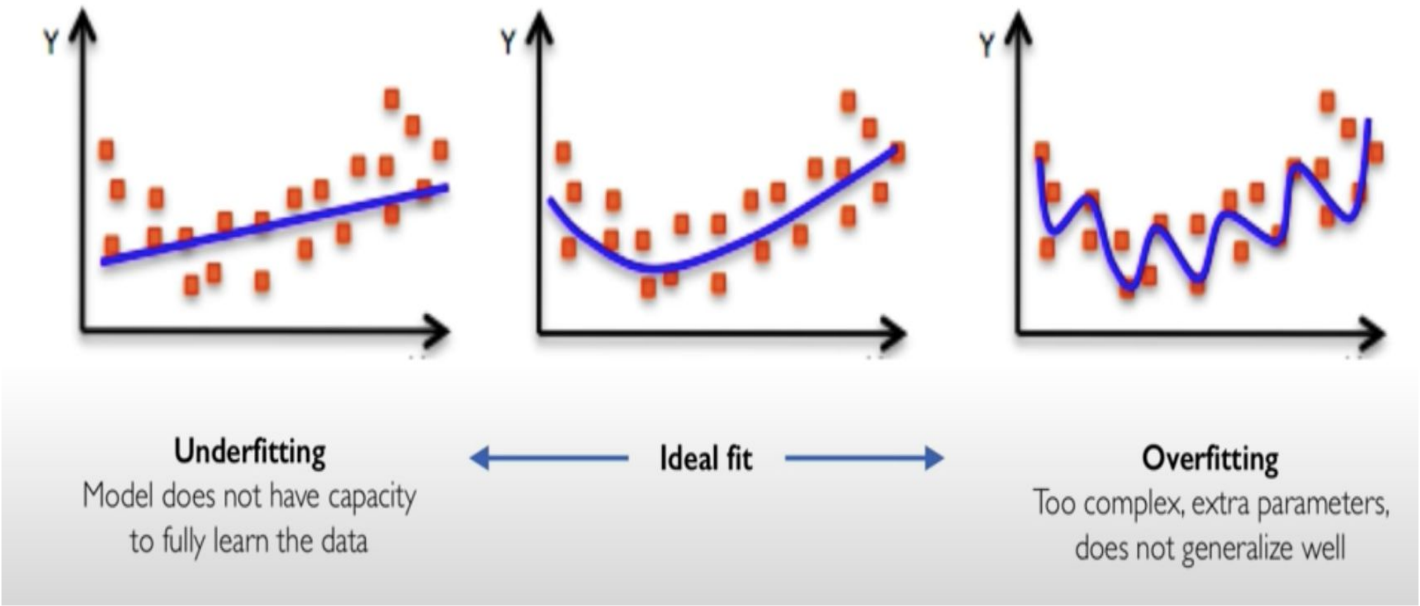
Auxiliary Training concepts: Splitting the dataset

- Standard practice: training, validation and test splitting
- K-fold cross validation (for small datasets)
- Common split percentage: **70%** training, **15%** validation, and **15%** for testing
- For benchmark tasks: predefined training, validation, and test split might exist
  - E.g. Glue benchmark

Auxiliary Training concepts: Knowing when to stop training

From: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

The problem of overfitting



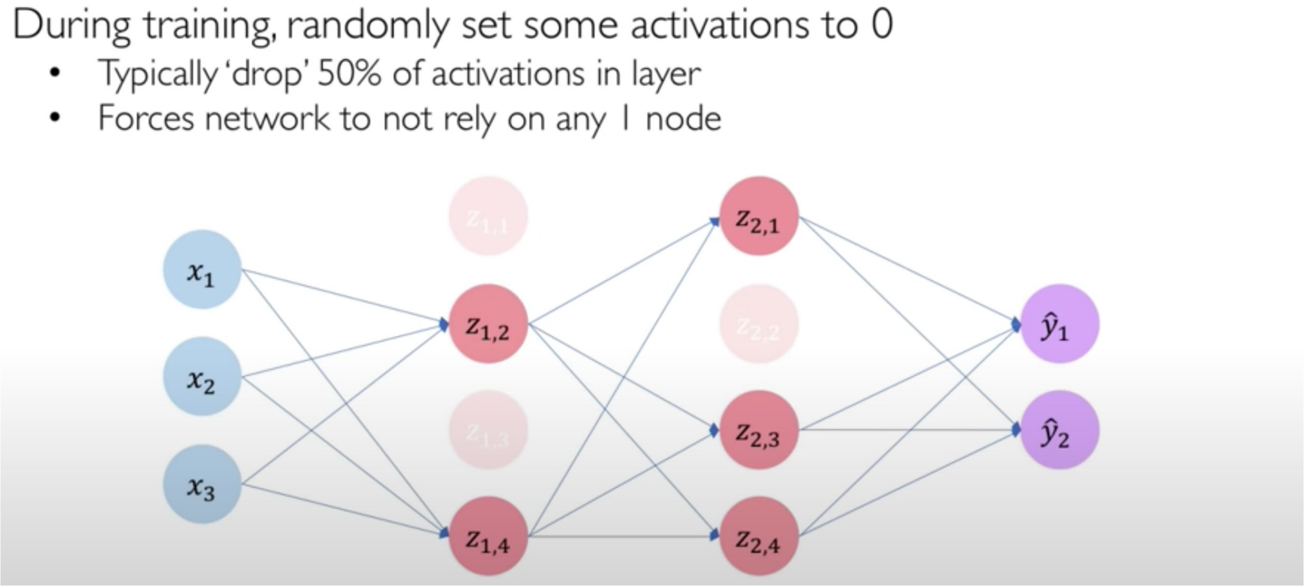
From: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Auxiliary Training techniques: Regularization

- Constrains optimization problem to discourage complex models
- Improves generalization on unseen data

Auxiliary Training concepts: Regularization

❖ Dropout



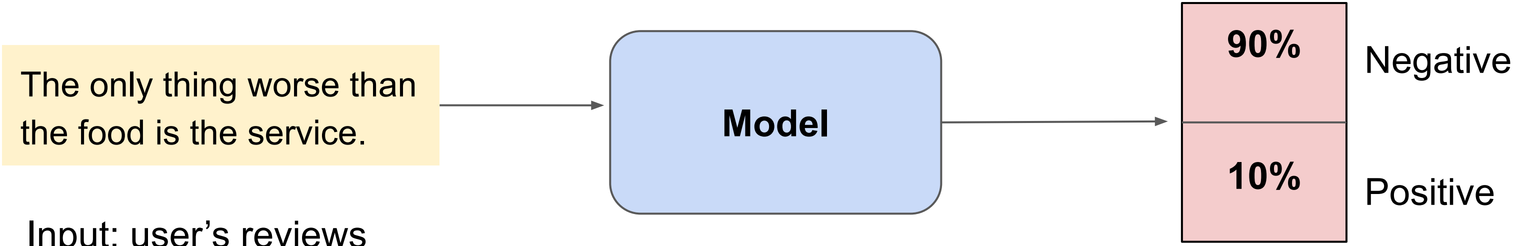
From: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Auxiliary Training concepts: Regularization

- L2 regularization (weight-decay), L1 regularization
- Data augmentation

Example: Classifying Sentiment of Restaurant Reviews

- Task: Classify reviews of a restaurant
  - Positive or negative



Input: user’s reviews

Output: sentiment

Yelp dataset simplified version : http://xzh.me/docs/charconvnet.pdf

Example: Classifying Sentiment of Restaurant Reviews

⇒ Data preprocessing notebook

⇒ Classifying reviews notebook