Welcome to SEAS Online at George Washington University

Class will begin shortly

Audio: To eliminate background noise, please be sure your audio is muted. To speak, please click the hand icon at the bottom of your screen (**Raise Hand**). When instructor calls on you, click microphone icon to unmute. When you've finished speaking, ***be sure to mute yourself again***.

Chat: Please type your questions in Chat.

Recordings: As part of the educational support for students, we provide downloadable recordings of each class session to be used exclusively by registered students in that particular class. Releasing these recordings is strictly prohibited.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

SEAS 8520 – Lecture 4: ANN Optimization and Regularization

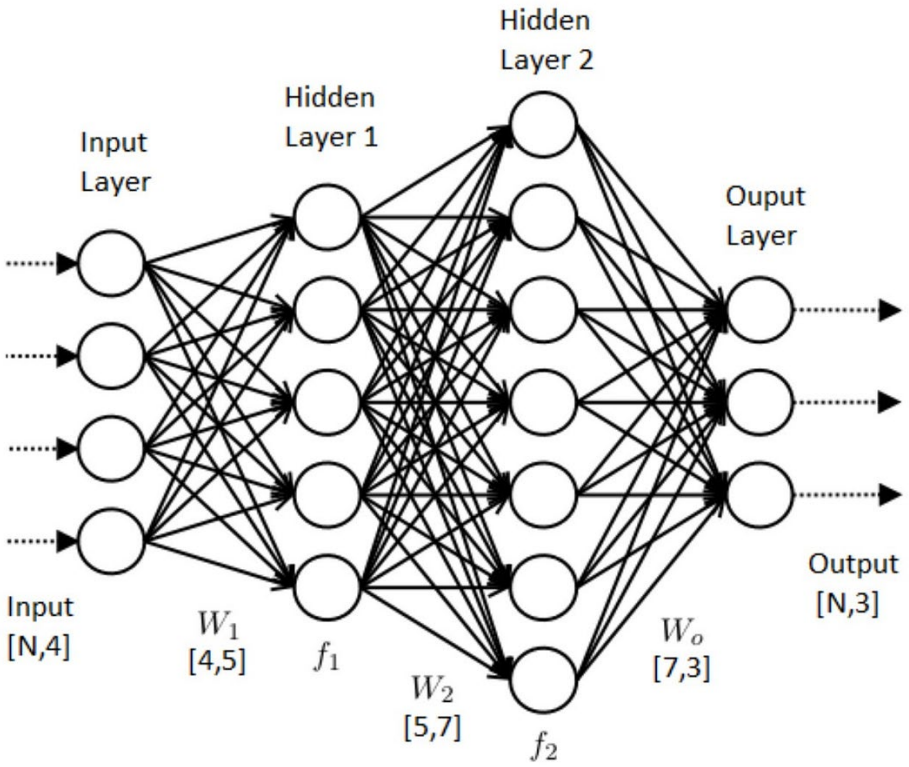
Walid Hassan, M.B.A, D.Eng.

Agenda

- Data Management
- Overfitting and Underfitting
- Regularization Techniques
- Learning Rate Optimization Algorithms
- Feature Scaling
- ANN for a Binary Classification Problem

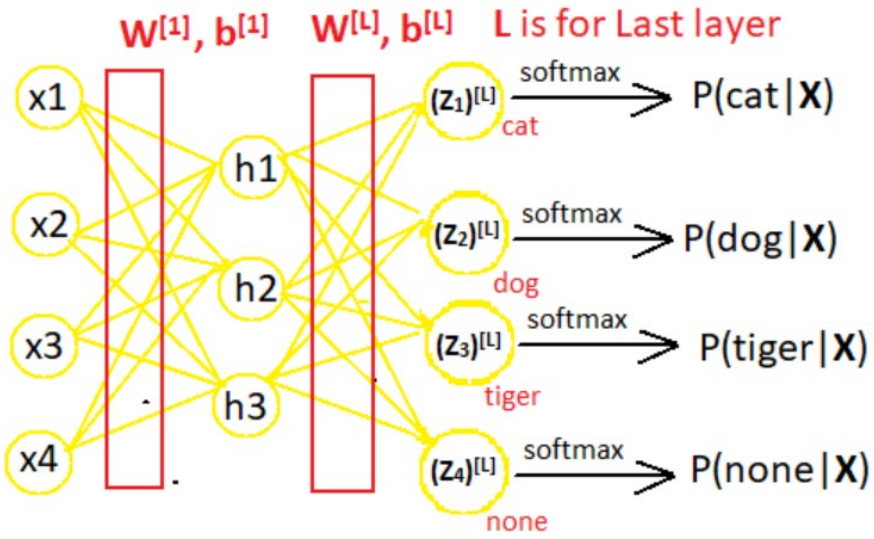
Multi-class Classification & Softmax

- Multiple classes, typically multiple outputs
- Score *should* reflect confidence …



Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
https://medium.com/@srijaneogi31/exploring-multi-class-classification-using-deep-learning

Multi-class Classification



$$softmax(Z_i) = \frac{\exp(Z_i)}{\sum \exp(Z_i)}$$

- It is the mathematical function that converts the vector of numbers into the vector of the probabilities. The softmax activation function is commonly used as an activation function in the case of multi-class classification problems in machine learning. The output of the softmax is interpreted as the probability of getting each class.

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
https://vidyasheela.com/post/softmax-activation-function-in-neural-network-formula-included

Multi-class Classification

$$Z^{[L]} = \begin{bmatrix} Z_1{}^{[L]} = 1.25 \\ Z_2{}^{[L]} = 2.44 \\ Z_3{}^{[L]} = 0.78 \\ Z_4{}^{[L]} = 0.12 \end{bmatrix}$$

$$e^{Z_1{}^{[L]}} = e^{1.25} = 3.49\,,$$

$$e^{Z_2{}^{[L]}} = e^{2.44} = 11.47,$$

$$e^{Z_3{}^{[L]}} = e^{0.78} = 2.18,$$

$$e^{Z_4{}^{[L]}} = e^{0.12} = 1.27.$$

$$\sum e^{Z^{[L]}} = e^{Z_1{}^{[L]}} + e^{Z_2{}^{[L]}} + e^{Z_3{}^{[L]}} + e^{Z_4{}^{[L]}}$$
$$= 3.49 + 11.47 + 2.18 + 1.27 = 18.41$$

$$P(cat|\boldsymbol{X}) = \frac{e^{Z_1{}^{[L]}}}{\sum e^{Z^{[L]}}} = \frac{3.49}{18.41} = 0.189$$

$$P(dog|\boldsymbol{X}) = \frac{e^{Z_2{}^{[L]}}}{\sum e^{Z^{[L]}}} = \frac{11.47}{18.41} = 0.623$$

$$P(tiger|\boldsymbol{X}) = \frac{e^{Z_3{}^{[L]}}}{\sum e^{Z^{[L]}}} = \frac{2.18}{18.41} = 0.1184$$

$$P(none|\boldsymbol{X}) = \frac{e^{Z_4{}^{[L]}}}{\sum e^{Z^{[L]}}} = \frac{1.27}{18.41} = 0.0689$$

- . The output of the softmax is interpreted as the probability of getting each class.

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
https://vidyasheela.com/post/softmax-activation-function-in-neural-network-formula-included

Universal Approximation Theorem

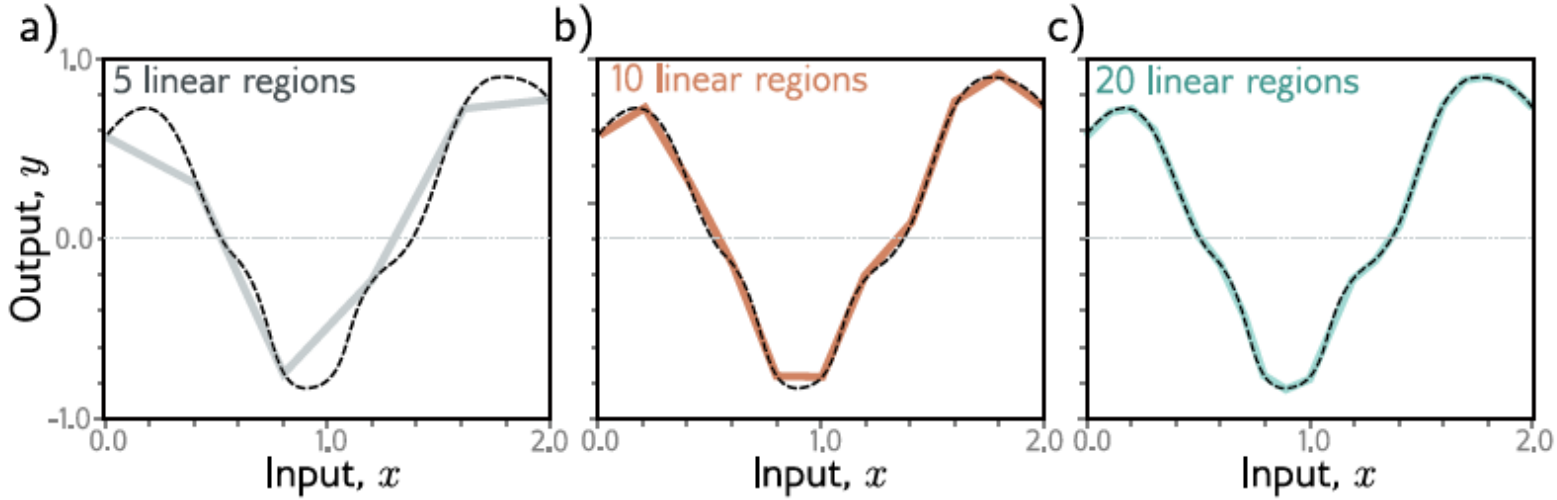
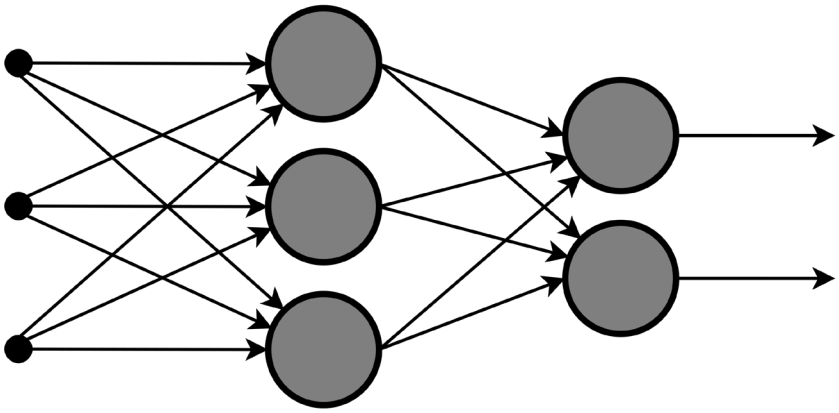


Figure 3.5 Approximation of a 1D function (dashed line) by a piecewise linear model. a–c) As the number of regions increases, the model becomes closer and closer to the continuous function. A neural network with a scalar input creates one extra linear region per hidden unit. The universal approximation theorem proves that, with enough hidden units, there exists a shallow neural network that can describe any given continuous function defined on a compact subset of $\mathbb{R}^{D_i}$ to arbitrary precision.

Source: Simon Prince.



Neural Networks

Data Management

1. Total Dataset: Your collected data before any splits.

2. Features and Labels: The inputs (X) and outputs (y).

3. Data Quality: Ensure your data is clean and well-preprocessed.

- **Definition**: The data used to fit the model.
- **Size**: Usually 60-80% of the total dataset.
- **Purpose**: To enable the model to learn the underlying patterns.
- **Example**: 7,000 samples in a 10,000-sample dataset would form a 70% training set.

Note that not all data splits are equal. The nature of the problem and the data itself might dictate specific splitting strategies.

- **Definition**: Data set aside for model tuning.
- **Size**: Typically, 10-20% of the dataset.
- **Purpose**: Hyperparameter tuning, model comparison, early stopping.
- **Example**: 1,500 samples in a 10,000-sample dataset would form a 15% validation set.

- **Definition**: Data used to evaluate the model.
- **Size**: About 10-20% of the dataset.
- **Purpose**: To assess how well the model generalizes to new data.
- **Example**: 1,500 samples in a 10,000-sample dataset would form a 15% test set.

K-Fold Testing

- Select different sets for Validation , Training, Testing (k-times) and average your metrics

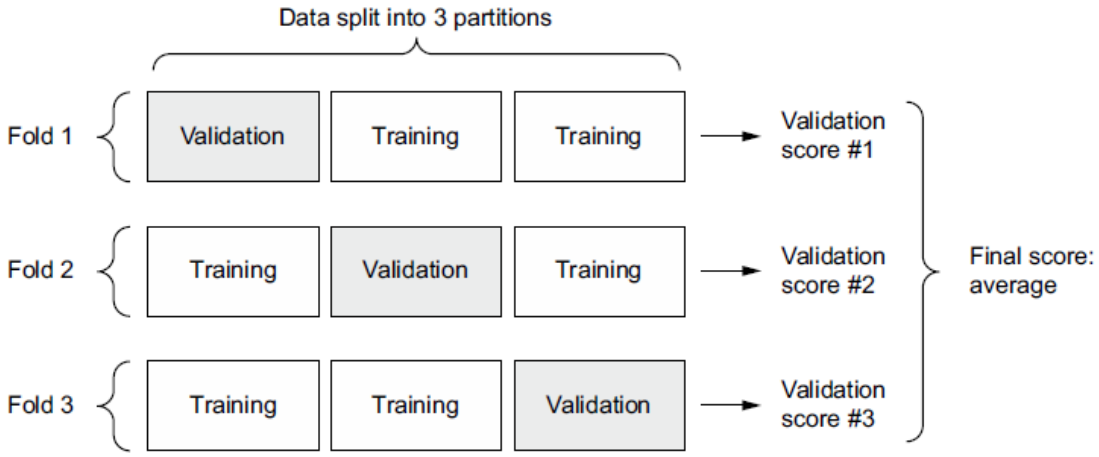
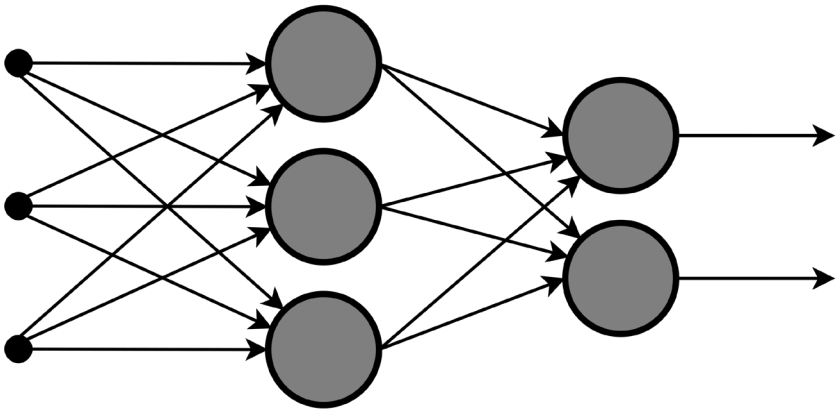


Figure 4.8 K-fold cross-validation with K=3

Src: Deep Learning with Python Ch: 4



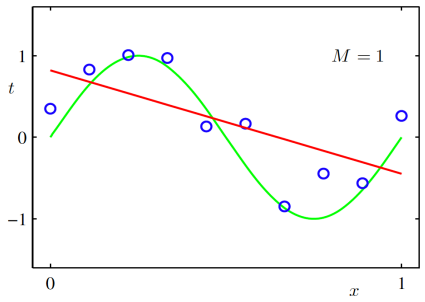
Neural Networks

Overfitting and Underfitting

- Generalization ability refers to an algorithm's ability to give accurate predictions for new, previously unseen data.
- Assumption: Future unseen data will have the same properties as the current training sets.
- Models that are too complex for the amount of training data available are said to overfit and are not likely to generalize well to new examples.
- Models that are too simple, that don't even do well on the training data, are said to underfit and also not likely to generalize well.

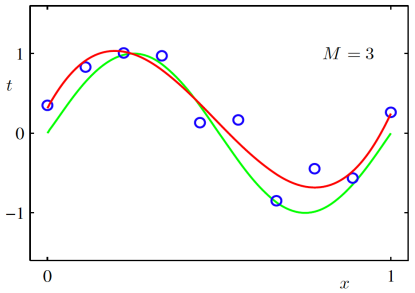
Model Generalization

f is linear



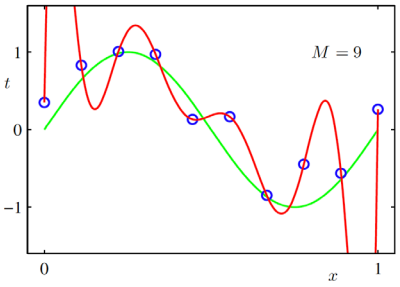
Loss is high

f is cubic



Loss is low

f is a polynomial of degree 9

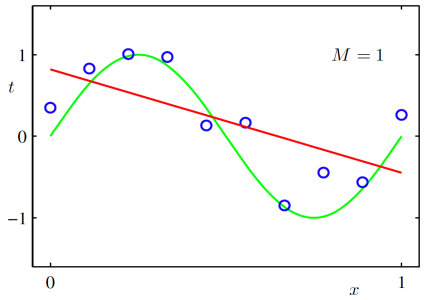


Loss is zero!

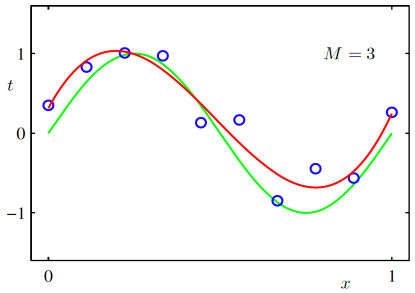
Underfitting

Overfitting

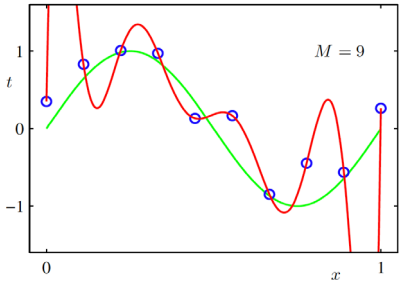
f is linear



f is cubic

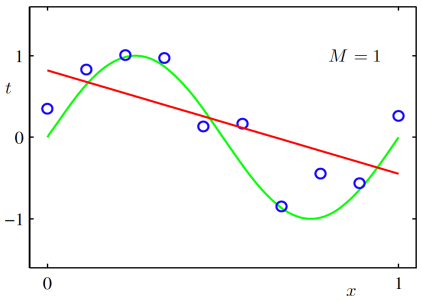


f is a polynomial of degree 9

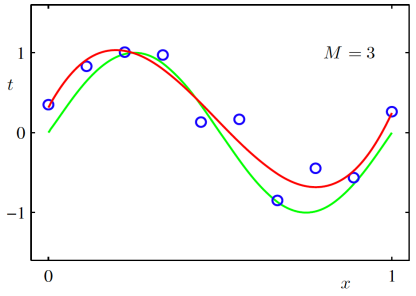


- The linear model suffers from "underfitting" (also called as high bias)
- The model is limited in terms of its free parameters and cannot adapt to a complex target function "f"
- Both in-sample (train) and test errors will be high

f is linear



f is cubic

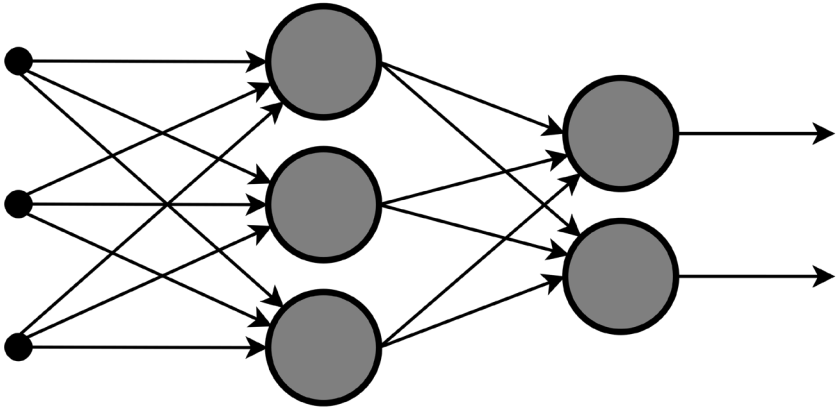


f is a polynomial of degree 9

f is linear

- The ninth -order model (model-3) is performing what is known as "overfitting" (also known as high variance)
- The model does not do well as it has more free parameters and fits the data more than is warranted. Essentially, the model is fitting to the noise in the training set
- In-sample (train) error will be low, but test error will be high

Neural Networks



Regularization Techniques

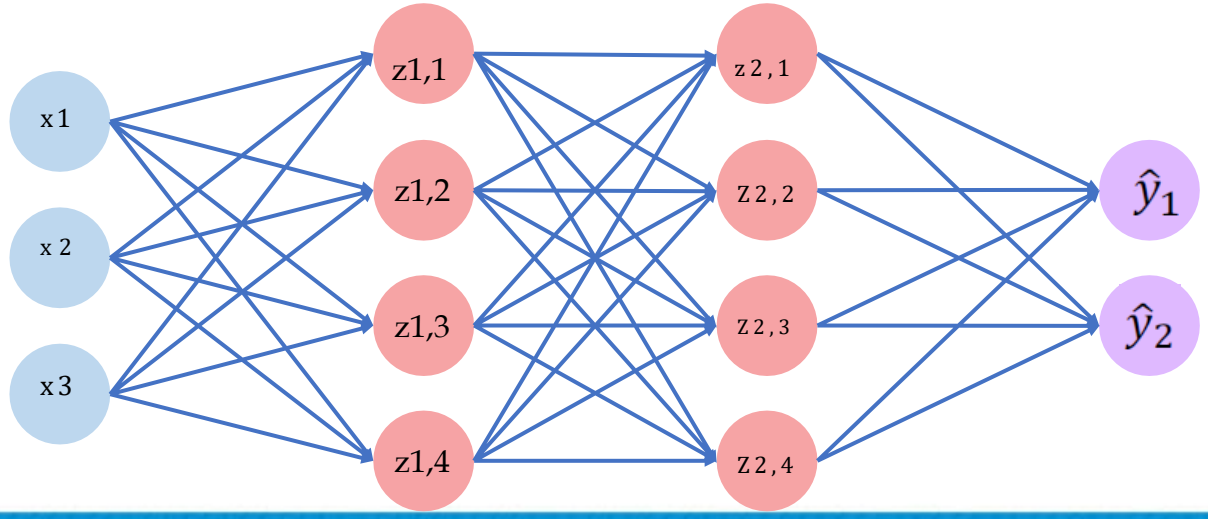
- Regularization is a technique used to prevent overfitting in ML models.
- Overfitting occurs when a model learns the training data too well, capturing noise as if it were a real pattern.
- Overfit models perform poorly on unseen data.
- The goal of regularization is to constrain or regularize the learning process to improve the model's generalization to new, unseen data.

Regularization Techniques

Technique	Description	Equation	When to Use
Dropout	Randomly sets a fraction of input units to 0 at each update during training, which helps prevent overfitting.	N/A	When dealing with neural networks and preventing overfitting is critical.
L1 Regularization (Lasso)	Adds the sum of the absolute values of the weights to the loss function. Encourages sparsity, meaning it encourages the model to have fewer parameters.	$L+\lambda\sum\|w_i\|$	When the model is suspected of having irrelevant or redundant features.
L2 Regularization (Ridge)	Adds the sum of the squares of the weights to the loss function. It discourages large weights and encourages small weights, helping in preventing overfitting.	$L+\lambda\sum w_i^2$	When you want to penalize large weights to avoid overfitting
Early Stopping	Halts training when the model's performance on a validation set stops improving to prevent overfitting.	N/A	Best used in scenarios with large datasets where model training time is to be optimized.

Regularization 1: Dropout

- During training, randomly set some activations to 0

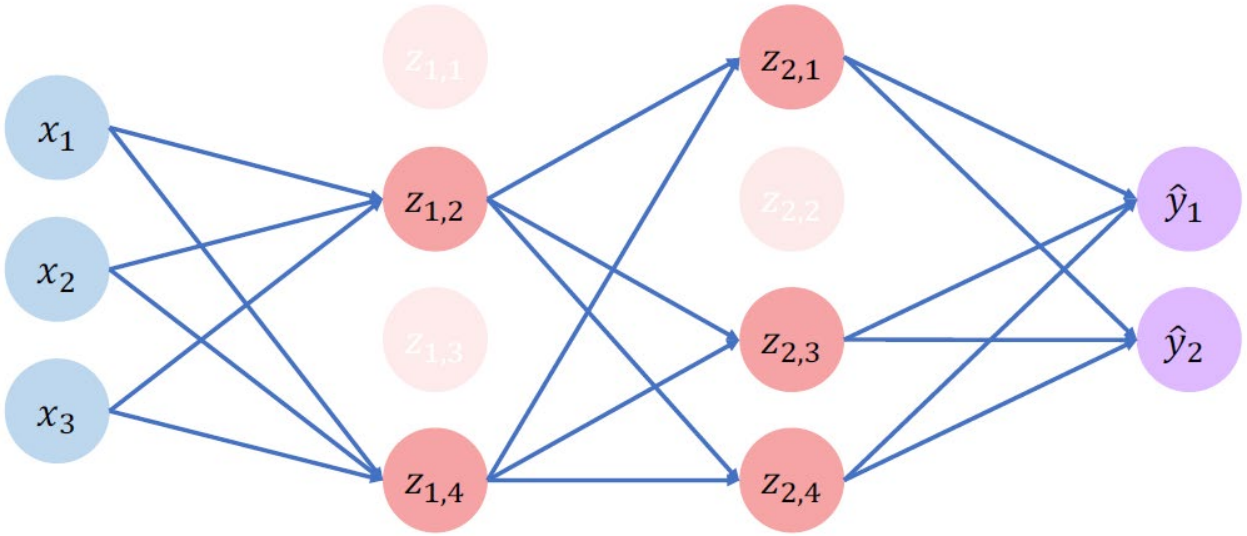


Source: introtodeeplearning.com

Regularization 1: Dropout

- During training, randomly set some activations to 0
  - Typically 'drop' 50% of activations in layer
  - Forces network to not rely on any 1 node

```
tf.keras.layers.Dropout(p=0.5)
```

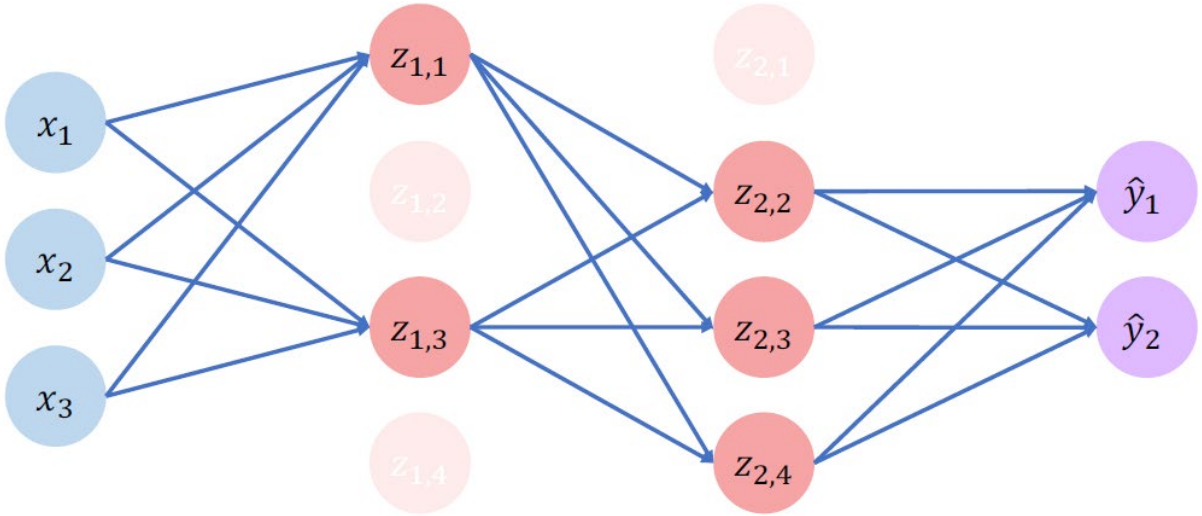


Source: introtodeeplearning.com

Regularization 1: Dropout

- During training, randomly set some activations to 0
  - Typically 'drop' 50% of activations in layer
  - Forces network to not rely on any 1 node

```
tf.keras.layers.Dropout(p=0.5)
```



Source: introtodeeplearning.com

- Early stopping is a form of regularization used to avoid overfitting.
- It tracks the model's performance on a validation set during training.
- If the validation performance stops improving, training is halted to avoid overfitting.
- Typically, the model state with the best validation performance is saved and restored.
- By stopping before overfitting, early stopping helps the model generalize better to unseen data.

- Patience Parameter: A hyperparameter that sets the number of epochs to wait for improvement in validation loss, preventing early stopping due to minor fluctuations.
  - Too small a value could lead to early stopping due to noise
  - Too large a value might prevent early stopping when it would be beneficial.

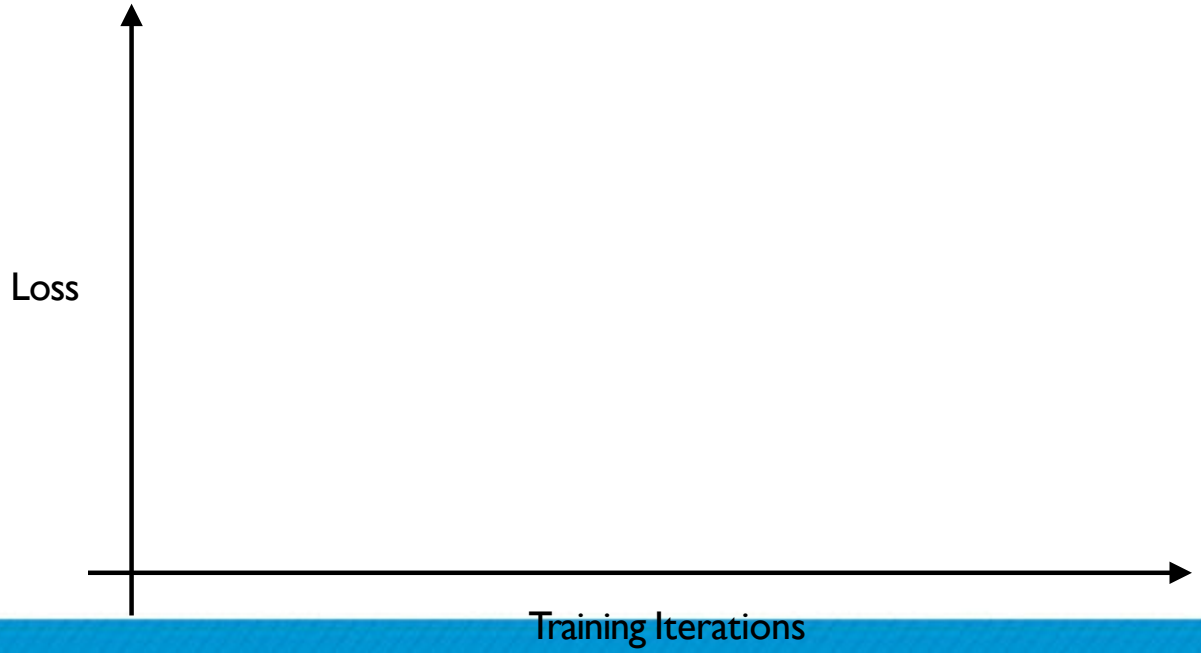
- Min_Delta Parameter: Specifies the minimum change in validation loss to be considered as an improvement.

- Together, patience and min_delta balance training speed and model accuracy.

- Both patience and min_delta require tuning to match the specific dataset and model architecture.

Regularization 2: Early Stopping

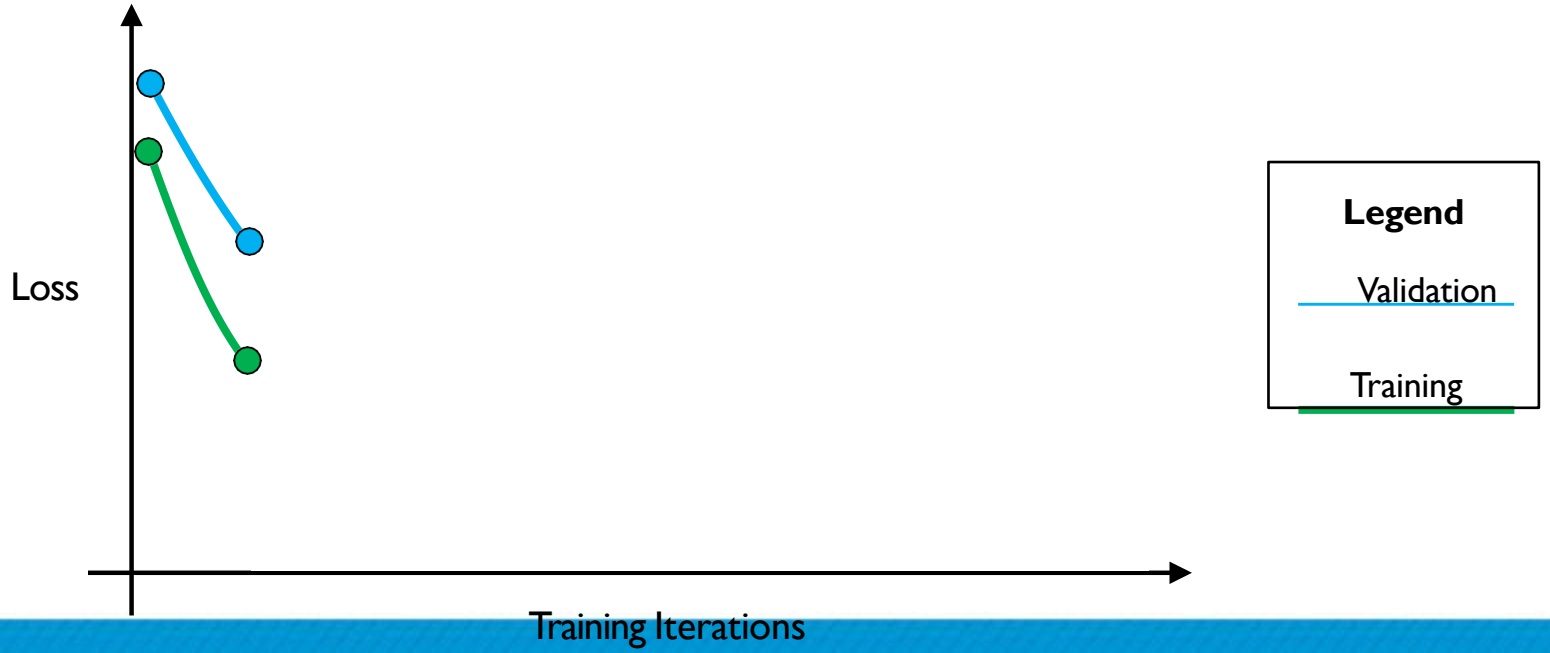
- Stop training before we have a chance to overfit



Source: introtodeeplearning.com

Regularization 2: Early Stopping

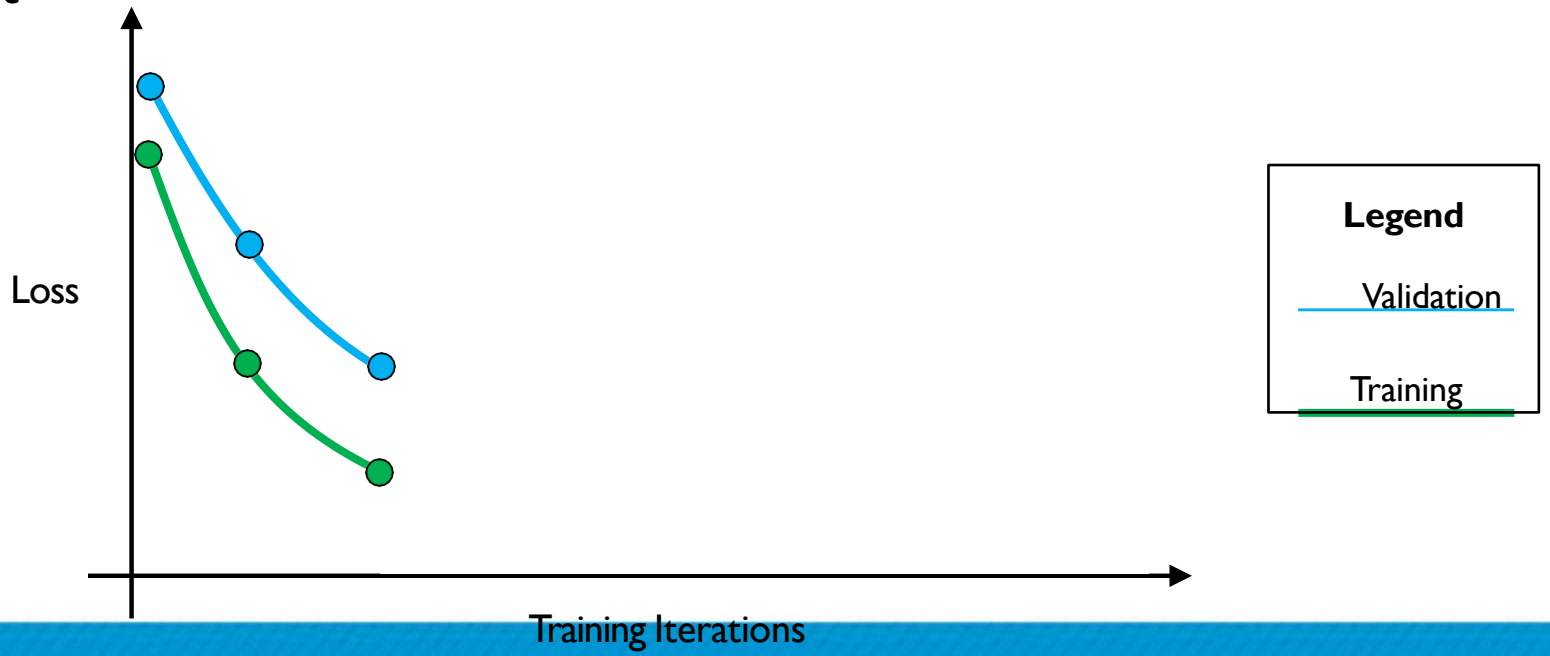
- Stop training before we have a chance to overfit



Source: introtodeeplearning.com

Regularization 2: Early Stopping

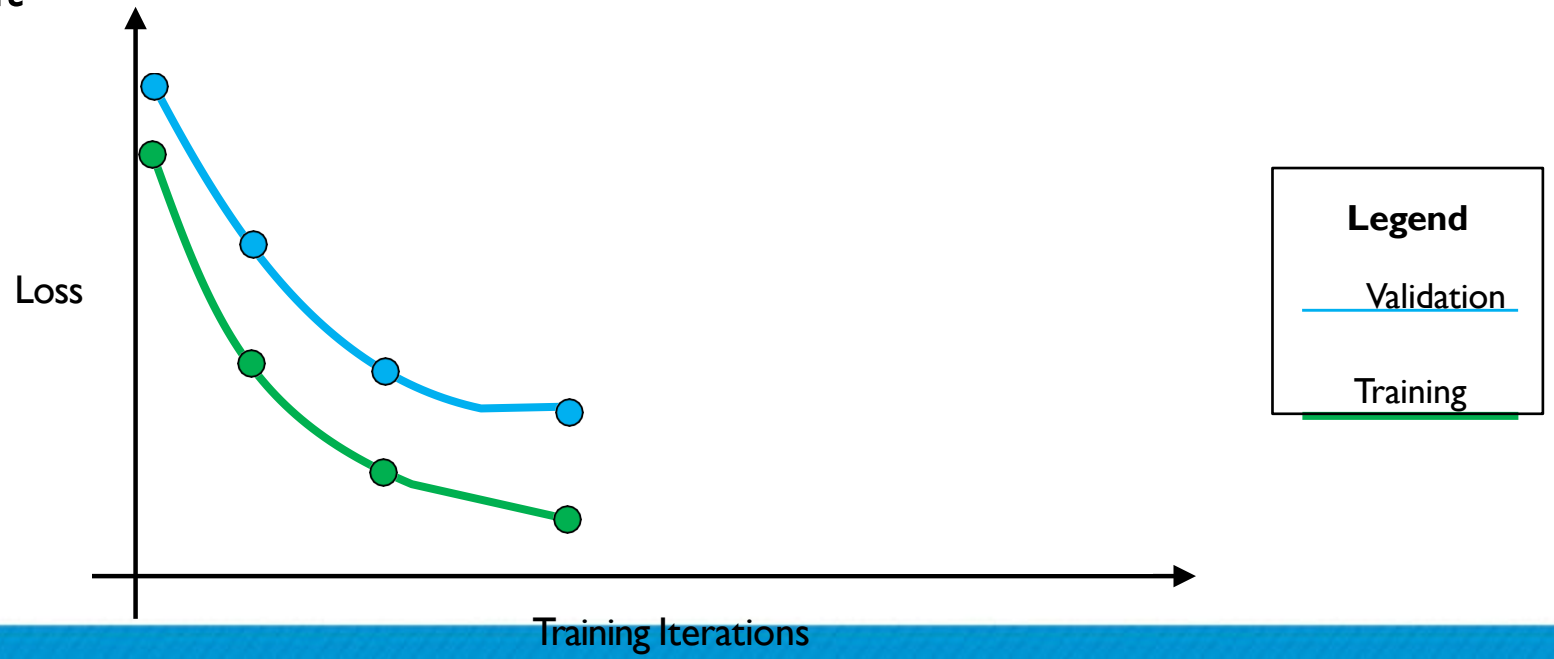
- Stop training before we have a chance to overfit



Source: introtodeeplearning.com

Regularization 2: Early Stopping

- Stop training before we have a chance to overfit



Source: introtodeeplearning.com

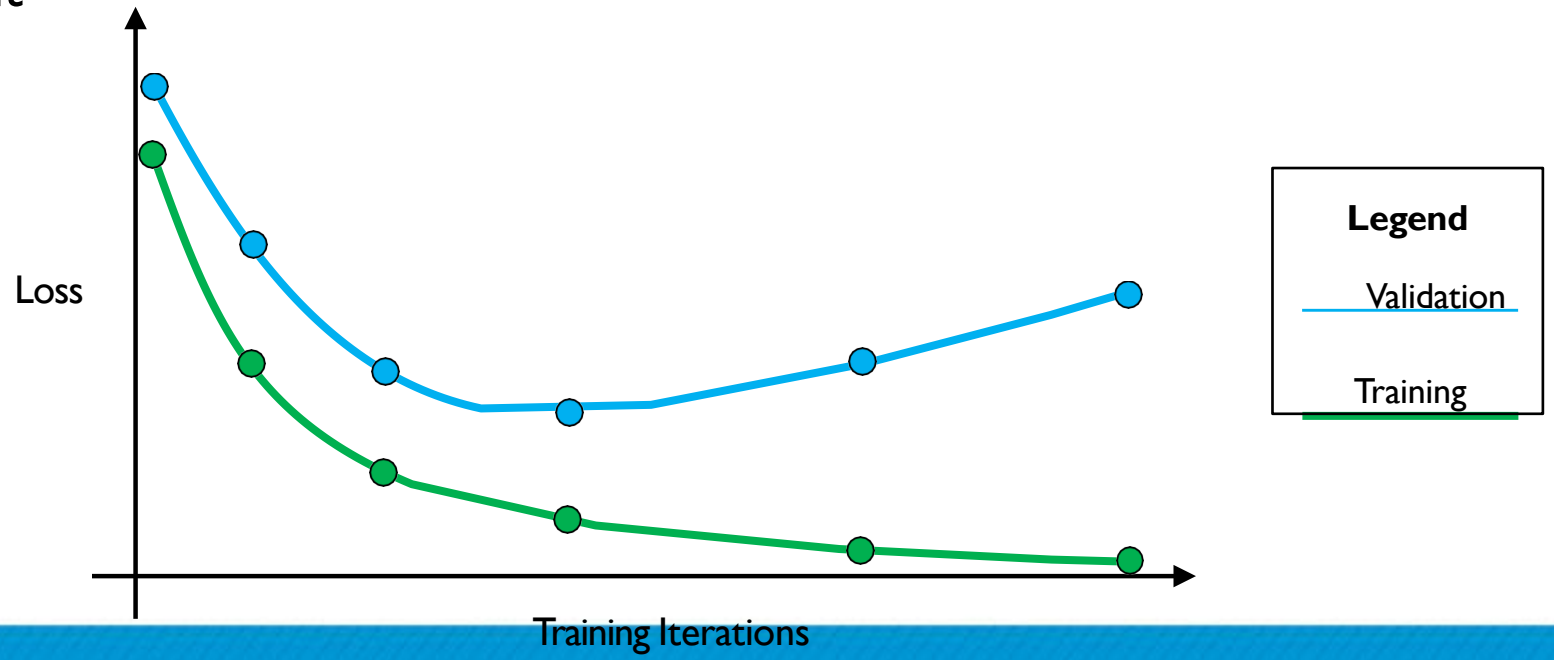
Regularization 2: Early Stopping

- Stop training before we have a chance to overfit

Source: introtodeeplearning.com

Regularization 2: Early Stopping

- Stop training before we have a chance to overfit



Source: introtodeeplearning.com

Regularization 2: Early Stopping

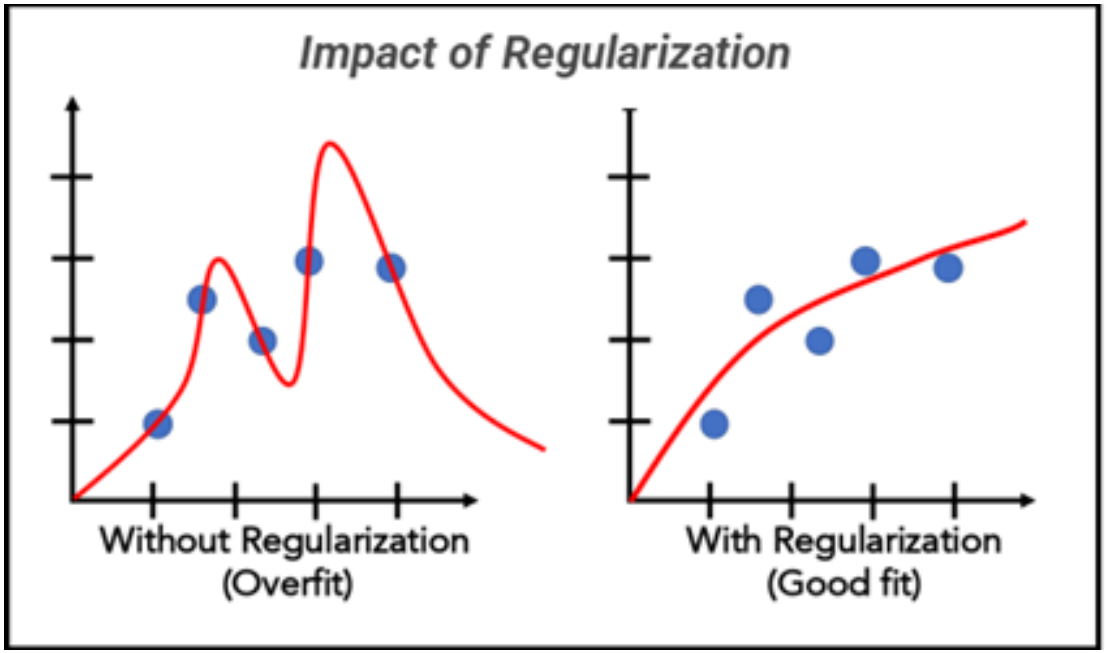
- Stop training before we have a chance to overfit

Source: introtodeeplearning.com

Regularization 2: Early Stopping

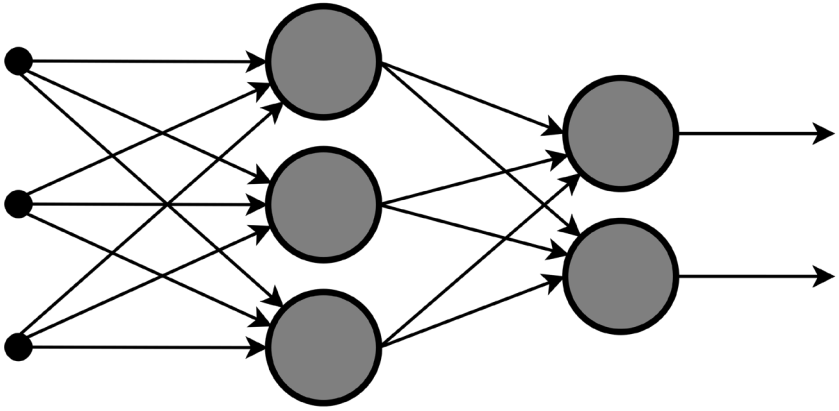
- Stop training before we have a chance to overfit

Source: introtodeeplearning.com



https://www.analyticsvidhya.com/blog/2021/11/study-of-regularization-techniques-of-linear-model-and-its-roles/

Neural Networks



Learning Rate Optimization Algorithms

Loss Functions Can Be Difficult to Optimize

Remember:

Optimization through gradient descent

$$w = w - \alpha \frac{\partial J(w,b)}{\partial w}$$

Loss Functions Can Be Difficult to Optimize

Remember:

Optimization through gradient descent

$$w = w - \alpha \frac{\partial J(w, b)}{\partial w}$$

How can we set the learning rate?

Setting the Learning Rate

- Small learning rate converges slowly and gets stuck in false local minima
- Large learning rates overshoot, become unstable and diverge
- Stable learning rates converge smoothly and avoid local minima

How to deal with this?

Idea 1:

Try lots of different learning rates and see what works "just right"

How to deal with this?

Idea 1:

Try lots of different learning rates and see what works "just right"

Idea 2:

Do something smarter!
Design an adaptive learning rate that "adapts" to the landscape

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...

Popular Learning Rate: Momentum, Adam

- **Purpose:** Momentum modifies the update rule by introducing a velocity component, v, which is a weighted sum of the current gradient and the previous velocity.

- **Formula:** At each time step t, the gradient of the loss function, $\nabla_w J(w)$, is computed based on current values.

Velocity Update: $v_t = \gamma v_{t-1} + \alpha \nabla_w J(w)$

Parameter Update: $w = w - v_t$

γ is the momentum coefficient (usually 0.9).

α is the learning rate.

- The velocity, v, is initially set to zero: $v_0 = 0$

- **Advantages:** Overcomes shallow local minima and reduces oscillations.

Consider a simplified scenario to illustrate how momentum affects the update rule in gradient descent:

Assume a single weight w, with an initial value of 0.

The learning rate (α) is set to 0.1.

The momentum coefficient (γ) is 0.9.

The gradients at three consecutive steps are 0.5, 0.4, and -0.3 (simplified for demonstration).

We'll calculate the updates for these three steps with and without momentum to demonstrate the effect.

Without Momentum

The update is simply the learning rate multiplied by the gradient:

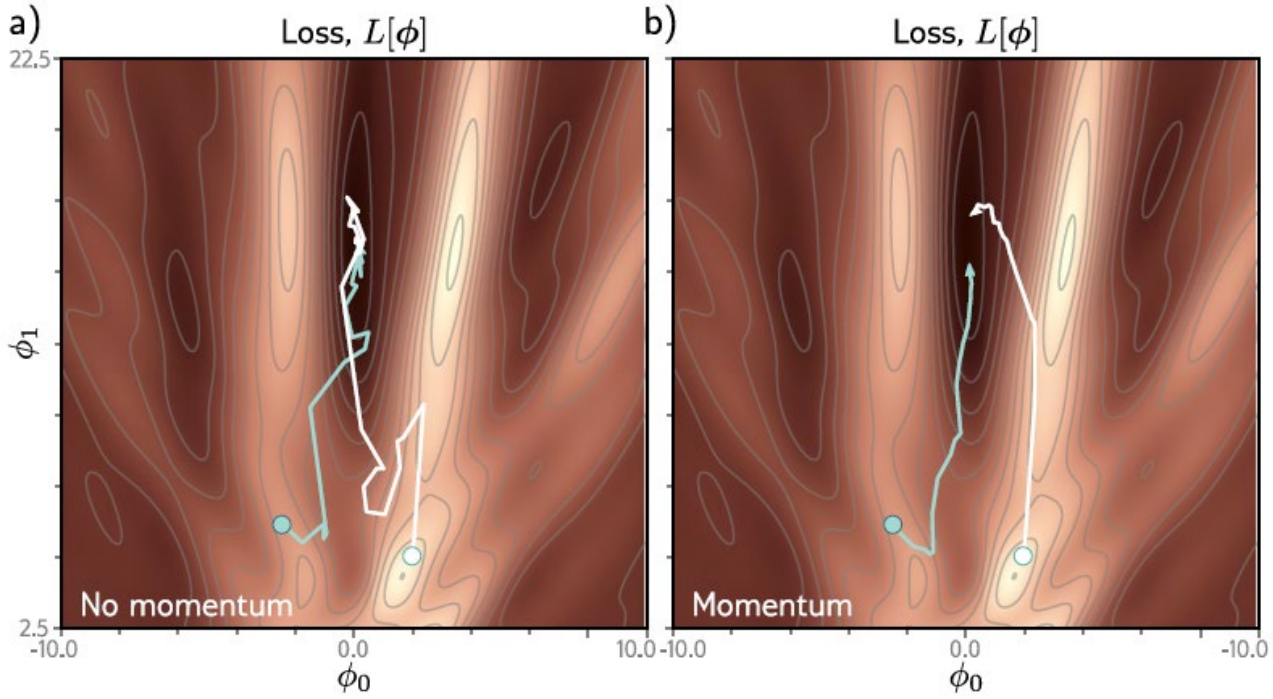
1. Step 1: $w = w - \alpha * 0.5 = 0 - 0.1*0.5 = -0.05$
2. Step 2: $w = w - \alpha *0.4 = -0.05 - 0.1*0.4 = -0.09$
3. Step 3: $w = w - \alpha*(-0.3) = -0.09 + 0.1*0.3 = -0.06$

With momentum, we also consider the velocity from the previous step:

1. **Step 1:**
    1. $v_0 = 0$ (initial velocity)
    2. $v_1 = \gamma v_0 + \alpha * 0.5 = 0.9 \cdot 0 + 0.1 * 0.5 = 0.05$
    3. $w = w - v_1 = 0 - 0.05 = -0.05$
2. **Step 2:**
    1. $v_2 = \gamma v1 + \alpha * 0.4 = 0.9 * 0.05 + 0.1 * 0.4 = 0.085$
    2. $w = w - v_2 = -0.05 - 0.085 = -0.135$
3. **Step 3:**
    1. $v_3 = \gamma v_2 + \alpha * (-0.3) = 0.9 * 0.085 + 0.1 * (-0.3) = 0.0465$
    2. $w = w - v_3 = -0.135 - 0.0465 = -0.1815$

- Momentum accelerates the optimization process by aggregating velocity from previous gradients. This means if the optimization is heading in a consistent direction towards the minimum, momentum will accumulate, increasing the size of the updates. While this is beneficial for traversing flat landscapes quickly or escaping shallow local minima, it can become a double-edged sword when approaching steep valleys or the global minimum.

- Learning Rate Scheduling: Adjusting the learning rate over time (e.g., reducing it as the optimization progresses) can help control the speed of convergence and minimize the risk of overshooting. As the learning rate decreases, the impact of accumulated momentum is also reduced, allowing for finer adjustments as the optimization process nears the minimum.

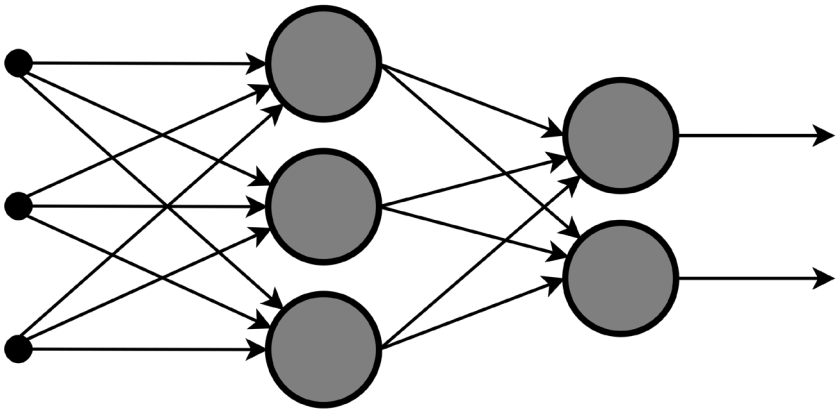


No Momentum

With Momentum

Adam is an Adaptive learning rate that adjusts the learning rate for each parameter individually, taking into account the historical gradients. This means that parameters with large gradients (indicating steep loss landscape) will have their learning rate reduced to prevent overshooting, while parameters with small gradients (indicating a flat loss landscape) will have their learning rate increased to speed up convergence.

Use Cases: Effective across various applications, notably in settings with sparse gradients like NLP and Computer Vision.



Neural Networks

Feature scaling

Introduction to Feature Scaling

1. **What is Feature Scaling?**
    1. Standardizing the range of independent variables or features in the data.

2. **Why Feature Scaling?**
    1. Ensures each feature contributes equally to the distance-based calculations in algorithms.

3. **Common Types**:
    1. Min-Max Scaling
    2. Z-Score Normalization

Feature scaling (Min-max scaling)

Get every feature into a $0 \leq x_i \leq 1$

$$x_i = \frac{x_i - min}{s_i}$$

Range $(max - min)$ of x_i in training set

Pros and Cons:

1. **Pros**: Simple, bounded range.
2. **Cons**: Sensitive to outliers.

Feature scaling (Standardization)

Standardization is different from min-max: first it subtracts the mean value (so standardized values always have a zero mean), and then it divides by the standard deviation so that the resulting distribution has unit variance.

$$x_i = \frac{x_i - \mu_i}{\sigma}$$

Pros and Cons:

1. **Pros**: Less sensitive to outliers.
2. **Cons**: Not Bounded Range.

Example - Linear Regression with Gradient Descent

- **Context**: Predicting house prices using two features - size (in square feet) and number of b $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
- **Scale Disparity**: Size ranges from 1000 to 2000 square feet, while the number of bedrooms ranges from 1 to 5.
- **Problem**: Without scaling, the larger scale of size dominates the gradient, affecting the optimization process.

Artificial Neural Network (ANN) for a Binary Classification Problem

Objective:

The objective is to help the bank identify customers who are likely to leave in the future so that they can take necessary actions to retain them, such as offering them special deals, incentives, or improved services.

Data:

The dataset is in a file called 'Churn_Modelling.csv' and contains various features that could influence customer churn.

1) Data Preprocessing

The dataset is read into a DataFrame.

Features and target labels are separated.

Categorical variables like "Gender" and "Geography" are encoded to numerical types.

2) Data Splitting

The data is split into a training set (60% of the data), validation set(20% of the data) and a test set (20% of the data).

3) Feature Scaling

Features are scaled to ensure that no variable dominates others, which could cause the model to perform poorly.

4) Model Building

An ANN with two hidden layers is created using TensorFlow/Keras.

5) Training

The model is trained on the training set.

6) Validation

The model is run on the validation set and accuracy as well as loss results are reported.

7) Testing

The model is run on the testing set and accuracy results are reported.

Experimentation with ANN Architectures

Objective: Understand the impact of neural network architecture complexity on model performance.

Scenarios to Explore:

Architecture: Create an ANN with two hidden layers containing 6 neurons each.

Simpler Architecture: Create an ANN with one hidden layer containing 3 neurons to demonstrate potential underfitting.

More Complex Architecture: Add two additional hidden layers to the initial model, each with 10 neurons, to explore a more complex architecture.

Run one of the scenarios without feature scaling.

Customer Churn Prediction for a Bank: Experimentation

Model Evaluation: For each scenario, record the following metrics:

Training Loss

Validation Loss

Training Accuracy

Validation Accuracy

Testing Accuracy on unseen data

Epoch at which Training Stopped (if early stopping is used)

Reflection: Consider how changes in model complexity may affect overfitting or underfitting and the model's ability to generalize.

Steps for Experimentation:

Use the provided base ANN code as the starting point.

Adjust the model architecture according to the given scenarios.

Train the model and capture the required metrics.

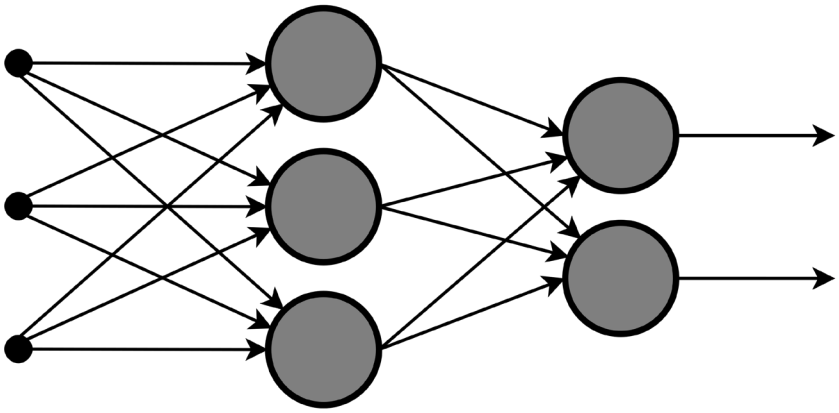
Perform the experiments with/without early stopping.

References

In addition to the references in each slide, the following are leveraged throughout this lecture:

DeepLearning.AI(https://www.deeplearning.ai/)

Backup



Neural Networks

Keras Tuner

- What is Hyperparameter Tuning?

- In machine learning, a model's performance can often be improved by tuning its hyperparameters. Unlike model parameters which are learned during training, hyperparameters are set before the training process begins.

- Common examples include learning rate, number of units in a neural network layer, dropout rate, and kernel size in a convolutional layer.

- Improved Performance: By searching through various hyperparameter combinations, you can find a set that offers the best performance on your validation set.

- Combat Overfitting: Some hyperparameters can help prevent overfitting. For example, adding dropout or reducing the complexity of the model.

- Faster Convergence: Proper learning rates or optimizers can help a model converge faster during training.

Grid Search:

- Exhaustive search through a manually specified subset of the hyperparameter space.
- Provides high accuracy but computationally expensive.

Random Search:

- Samples hyperparameter settings randomly from the predefined space.
- More efficient and effective compared to grid search in high-dimensional spaces.

Hyperband:

- Based on the idea that if a set of hyperparameters is promising in the early stages of training, it will likely be promising at the end. So, it allocates more resources to promising configurations.

- First, you define a model-building function where you also define the search space for hyperparameters.
- Next, you choose a tuner (Random Search, Hyperband, etc.) and pass in the model-building function.
- Finally, you call the search method on the tuner object to start the tuning process.

Important Libraries

Library	Description	Deep Learning Use Case(s)	Key Features in Deep Learning Context
NumPy (np)	A library for numerical operations that's often used for mathematical computations in deep learning.	Data preprocessing, feature extraction, weight initialization, etc.	- Efficient multi-dimensional array operations -Linear algebra statistical, and mathematical functions -Useful for initializing weights, etc.
Pandas (pd)	Frequently used in the data preprocessing steps that are crucial in deep learning workflows.	Data loading, cleaning, and preprocessing for feeding into neural networks.	- Handling tabular data with mixed types (numerical, categorical).
TensorFlow (tf)	A go-to library for constructing and training deep learning models.	Building, training, and deployment of deep learning models.	- High-level APIs like Keras for rapid development - Low-level APIs for fine-grained control - Efficiently utilizes hardware (GPUs/TPUs)

1. **Root Mean Square Propagation:** Adaptively changes the learning rate during training.

2. **Learning Rate:** Typically requires less tuning and is more robust to the choice of initial learning rate.

3. **Use Cases:** Especially effective for problems characterized by sparse data.

4. **Consideration:** Can sometimes be sensitive to the choice of its hyperparameter values.

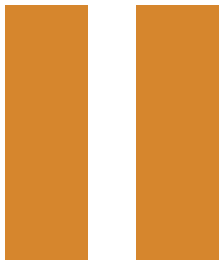
The core idea behind momentum-based optimization is akin to a ball rolling down a hill. The ball (representing the optimization process) accumulates velocity as it rolls downhill, allowing it to move faster towards the bottom (the minimum of the loss function) and to overcome small bumps (local minima or irregularities in the loss landscape) along the way.

1. **Root Mean Square Propagation:** Adaptively changes the learning rate during training.

2. **Learning Rate:** Typically requires less tuning and is more robust to the choice of initial learning rate.

3. **Use Cases:** Especially effective for problems characterized by sparse data.

4. **Consideration:** Can sometimes be sensitive to the choice of its hyperparameter values.



BREAK

Please come back @

2:25 PM EST

1:25 PM CST

Backup

Hands On…..

References

In addition to the references in each slide the below are leveraged throughout this course.

Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.

VanderPlas, J. (2017). Python Data Science Handbook. O'Reilly.
Wolff, S. G. (2018). Less is more: optimizing classification performance through feature selection in a very-high-resolution remote sensing object-based urban application. GIScience & Remote Sensing. doi:10.1080/15481603.2017.1408892

Prince, S. J. (January 28, 2024). Understanding Deep Learning. MIT Press.