NEURAL NETWORKS FOR NOVICES

TREVOR YU & CARTER DEMARS

TODAY'S AGENDA

- Examine deep learning concepts such as tensors, tensor operations, gradient descent, and backpropagation.
- Define and discuss hyperparameters in the context of deep learning models, including learning rate, batch size, epochs, layers, hidden units, optimizers, and activation functions.
- Interpret loss curves to identify overfitting during the training process.
- Apply this knowledge to a real-world dataset using TensorFlow.

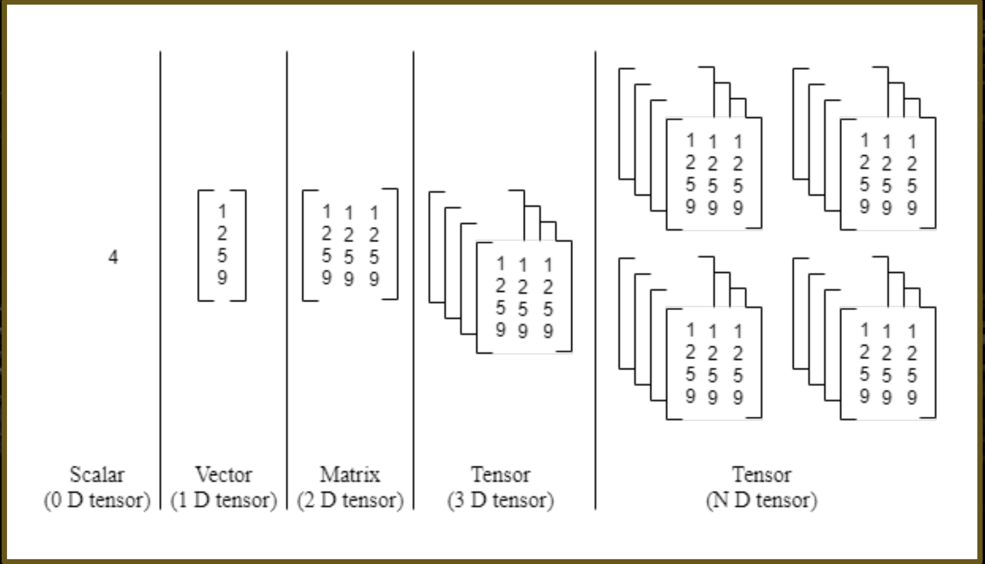
WHY LEARN ABOUT DEEP LEARNING?

- When is deep learning most useful?
    - Enough labelled training data is available
    - Access to proper compute infrastructure
    - Performing complex tasks (image classification, NLP, speech recognition)
    - Can often achieve high accuracy than other techniques

- When is it less useful?
    - Insufficient data quantity
    - No access to compute (GPUs, TPUs)
    - Explainability and interpretability are a priority
    - More challenging to run in production
    - Not always the simplest solution: when faced with several methods that give roughly equivalent performance, pick the simplest!

TENSORS

- A tensor is a homogenous, n-dimensional array of numbers
  - 0-D: Scalar
  - 1-D: Vector
  - 2-D: Matrix
  - 3-D+: Tensor

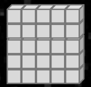
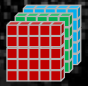


TENSOR DIMENSIONS

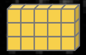
- In deep learning, we use the following letters to represent certain dimensions
    - N for Number of examples, B for Batch size
    - W for Width, horizontal spatial dimension of an image
    - H for Height, vertical spatial dimension of an image
    - C for Channels (images or signals), D for feature Dimension (most other data)
- The batch dimension is often first / on the left
- The feature dimension is often last / on the right

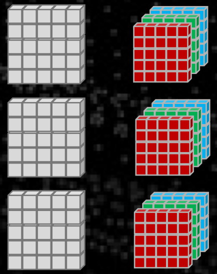
TYPICAL TENSORS SHAPES

SINGLE DATA ELEMENTS

- Scalar (1,) 
- Row vector (1, D) 
- Greyscale image (W, H, 1) 
- RGB image (W, H, 3) 

BATCHES OF DATA ELEMENTS

- Column vector (B, 1) 
- Feature matrix (B, D) 
- Greyscale image (B, W, H, 1)
- RGB image (B, W, H, 3)

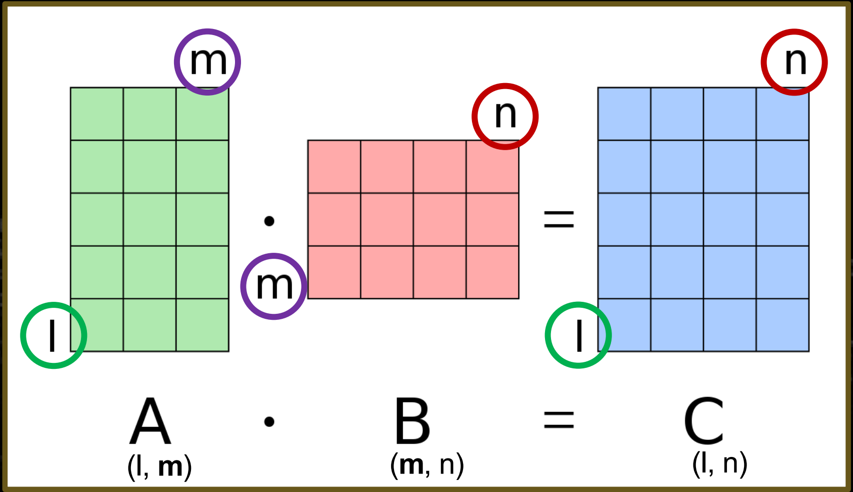


RESHAPING TENSOR OPERATIONS

- Alter the shape / layout of the tensor without adding or modifying the elements
- Useful for making inputs compatible with neural network
- Examples:
    - `tf.reshape(a, new_shape)`: General reshape, specify a tuple of the new shape
    - `tf.squeeze(a)`: Remove singleton dims
    - `tf.expand_dims(a, axis)`: Add a singleton dim at specified axis
    - `tf.keras.layers.Flatten()`: Unravel all non-batch dims into a vector
    - `tf.transpose(a, perm)`: Permute order of dims

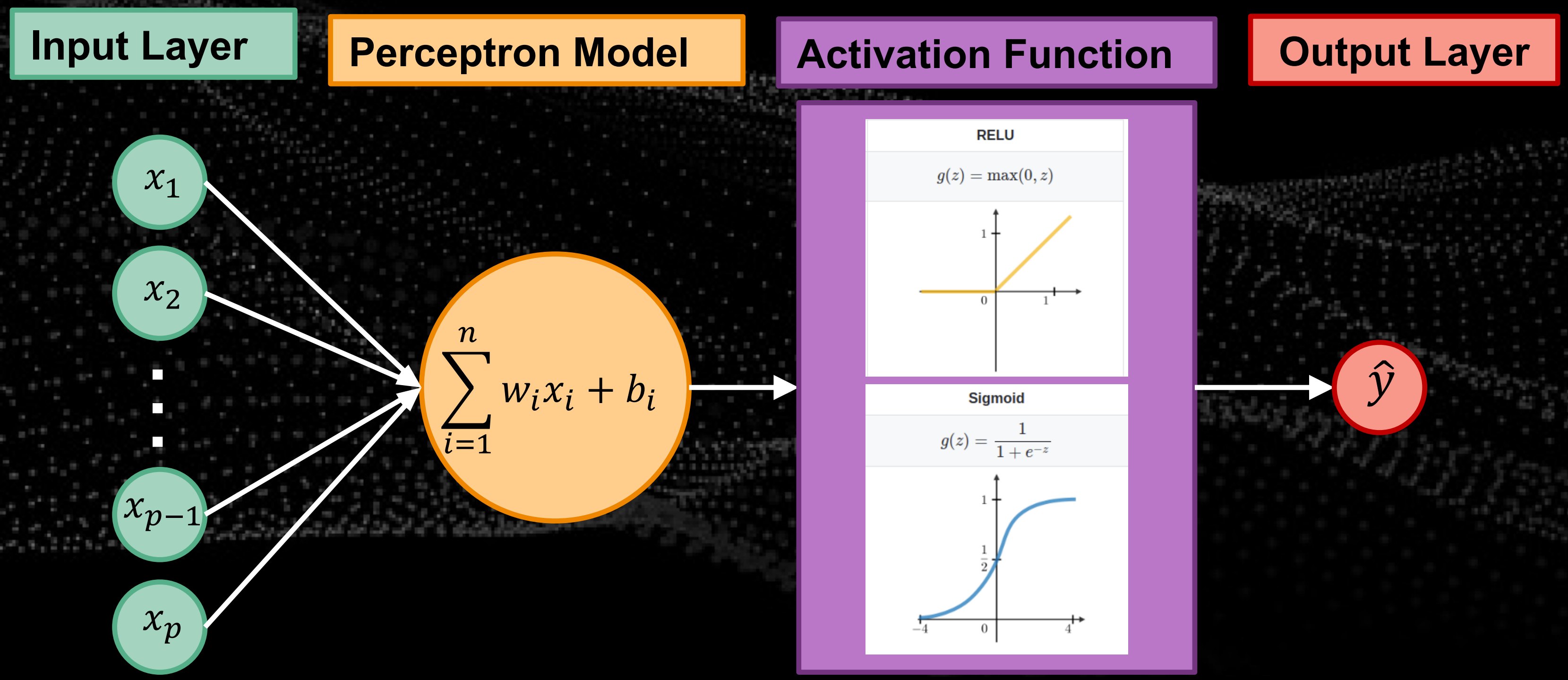
MATRIX MULTIPLICATION

- Also known as dot product or inner product
- Left operand (A) is at least 2-D tensor, right operand (B) is a 2-D tensor (matrix)
- Matmul is a linear transformation on A from m-dimensional vector space to n-dimensional vector space
- Used in Dense layers



- Last dimension of A must be same size as second-last dimension of B
- C maintains same shape of A, but with last dimension the same as last dimension of B

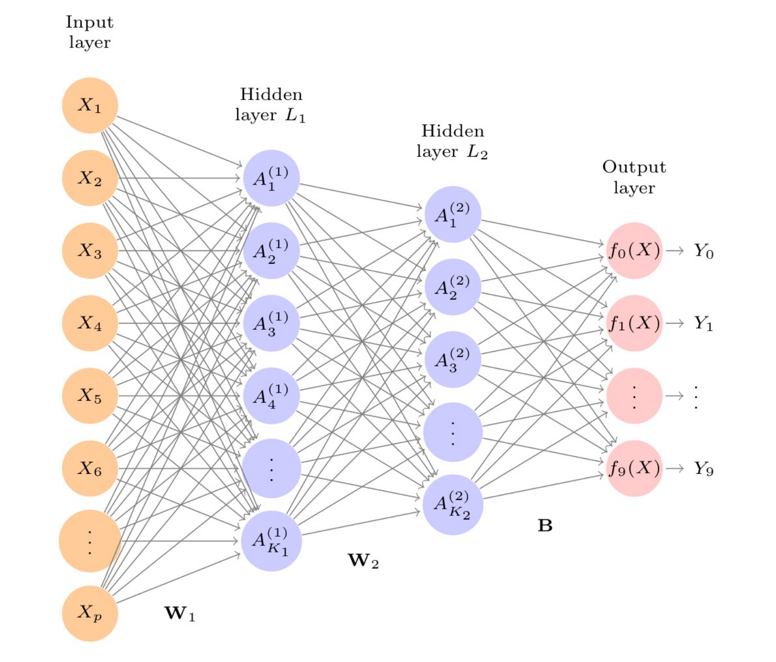
NEURAL NETWORKS: THE PERCEPTRON



NEURAL NETWORKS: THE PERCEPTRON

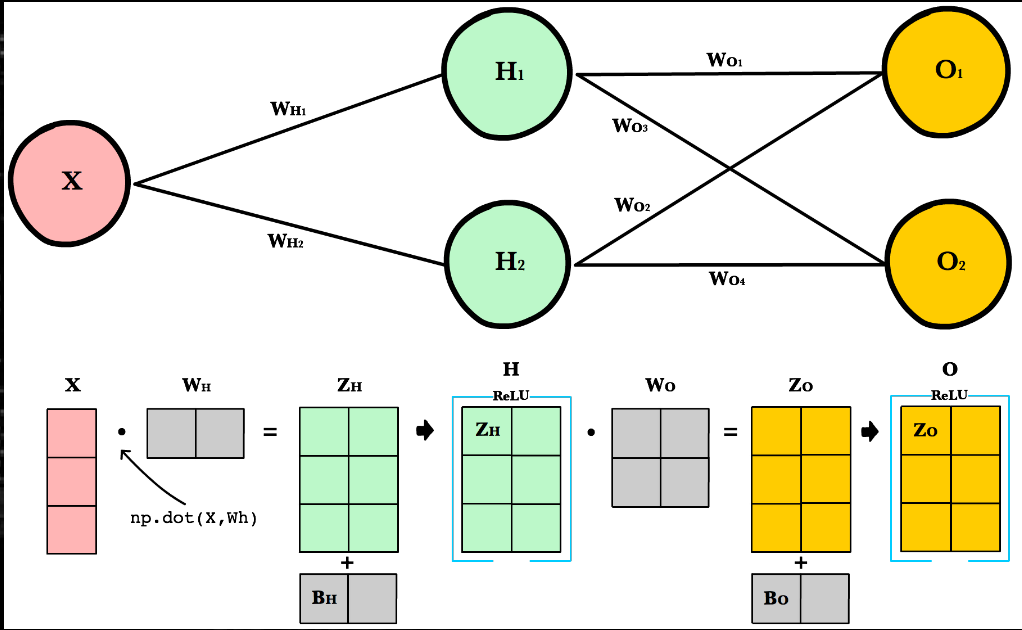
- Modern neural networks generally have many layers of perceptrons
  - Input Layer
  - Hidden layer(s)
  - Output Layer

- The operations that are performed at each node will differ based on the model architecture, but the general idea holds.
  - Fully connected layers
  - Convolutional layers
  - Recurrent layers
  - Attention layers



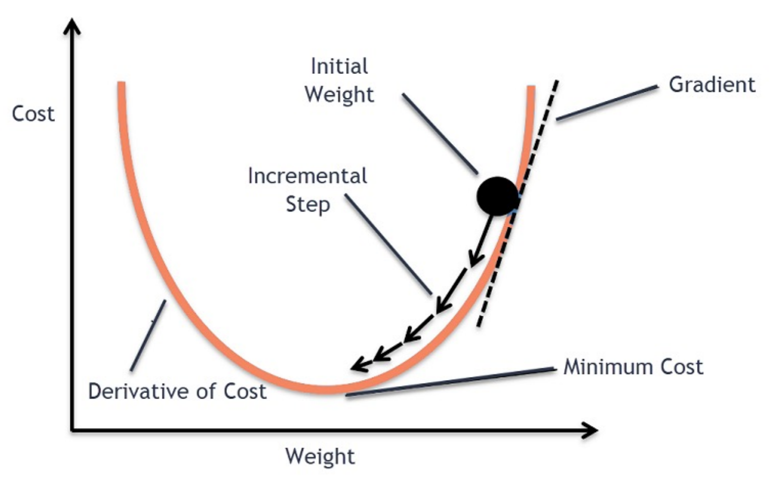
NEURAL NETWORKS: FORWARD PROPAGATION

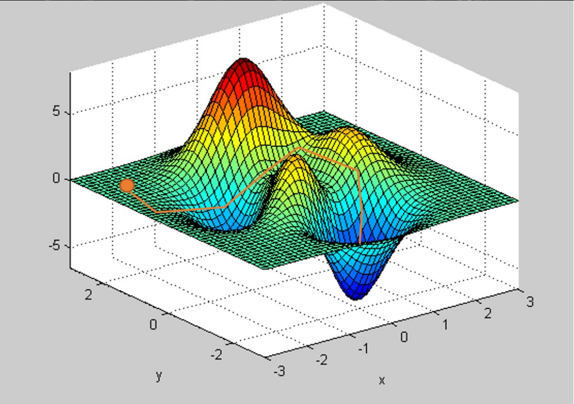
- Initialize weights and biases
- At each hidden layer compute
  - Preactivations (linear transformation of inputs)
  - Activations (apply activation function)
- At the final layer, evaluate the output using a cost function



NEURAL NETWORKS: OPTIMIZATION

- "Training" a neural network involves using stochastic gradient descent optimization (SGD) to adjust the network's weights such that they minimize a cost function
    - Cost (loss) functions are differentiable and relate to the task
- Regular GD requires gradients (derivatives) over the entire dataset. We can't do this if our dataset is too big.
- SGD splits the dataset into random "mini-batches". Iterating through all mini-batches in the dataset is called an "epoch"
- NN optimization is highly non-convex





NEURAL NETWORKS: LOSS FUNCTIONS

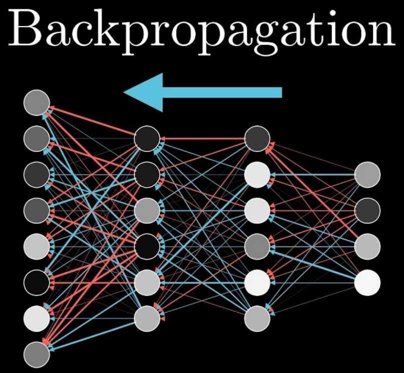
- Loss functions are a metric used in training that measures how good model estimates are compared to true labels
  - Differentiable, so can compute gradients through them
  - Surrogate metric, they are not exactly the measurement of "real world success"
- Final layer activations convert convert raw outputs to sensible values
  - For classification, we convert log odds (logit) scores into probabilities between (0, 1)
  - For regression, raw outputs should represent target variables
- Regression uses mean squared error (MSE)
- Classification uses cross entropy (CE)
  - CE measures difference between two probability distributions

NEURAL NETWORKS: LOSS FUNCTIONS

Task	Last layer activation	Loss Function	Typical Real-World Metrics
Binary classification	y = sigmoid(a)	Binary cross entropy $$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log(1 - \hat{Y}_i))$$	Accuracy, precision, recall
Multi-class classification	y = softmax(a)	Cross entropy $$L_{\mathrm{CE}} = -\sum_{i=1}^{n} t_i \log(p_i),$$	Categorical accuracy, top-k accuracy
Regression	y = a	Mean squared error $$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2,$$	Root mean squared error, mean absolute error

NEURAL NETWORKS: BACKWARD PROPAGATION

- We want to use GD to update parameters, but how to compute derivatives?
- Backpropagation is an algorithm to compute the derivative of the loss wrt parameters using the chain rule
- DL packages like Tensorflow and PyTorch do this automatically (autograd)
- (Interested in math behind backpropagation? 3B1B video series)

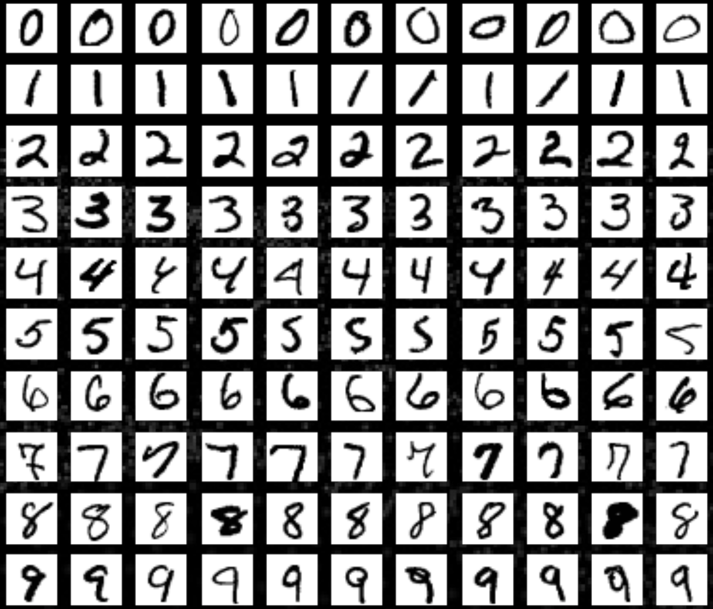


TRAINING A NETWORK

1. Obtain dataset
2. Convert data to tensors
3. Define model architecture
4. Select hyperparameters
5. Fit model with gradient descent
6. Evaluate on test dataset

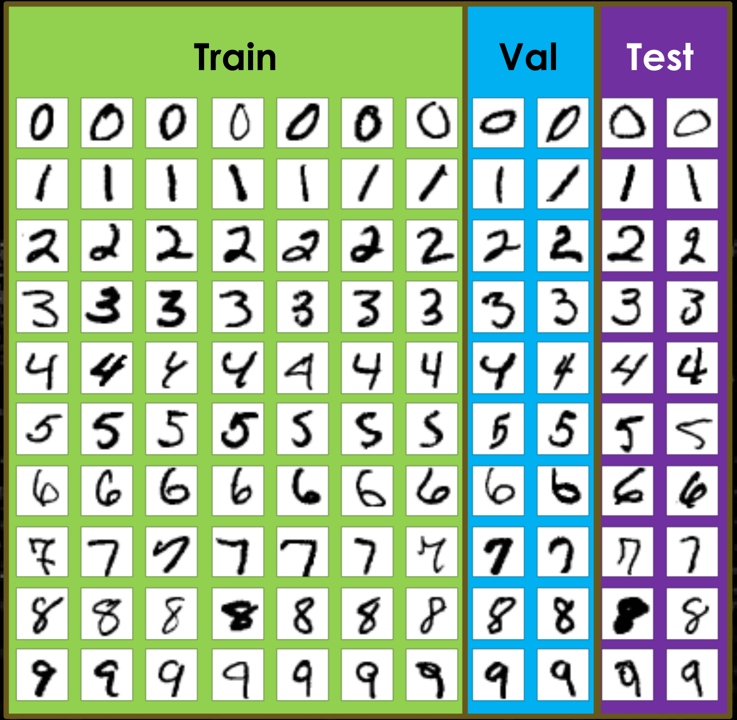
MNIST DATASET

- Modified National Institute of Standards and Technology database of handwritten digits
- Originally collected for automatically reading US ZIP codes for mail sorting
- “Hello world” problem of deep learning
- Ten categories (digits 0-9)
- 60,000 training examples
- 10,000 testing examples



TRAINING SPLITS

- During neural network training, we want to monitor for overfitting using a validation dataset
  - Split of the training data that will not be used for parameter updates
  - Separate from the testing data that is evaluated at the end of training procedure
- Use validation data for tuning hyperparameters, early stopping, but not for performance claims



HYPERPARAMETERS

- Hyperparameters are parameters of a model setup that are set by an ML Engineer and not updated during gradient descent optimization

- Changing hyperparameters will directly affect the model's performance, important to get them right

- Packages like Optuna can be used to automatically find good hyperparameters

HYPERPARAMETERS

- Model architecture
  - Number of layers, number of neurons per layer
  - Convolution kernel size
  - Model block structure

- Data preprocessing steps
  - Feature engineering, feature selection

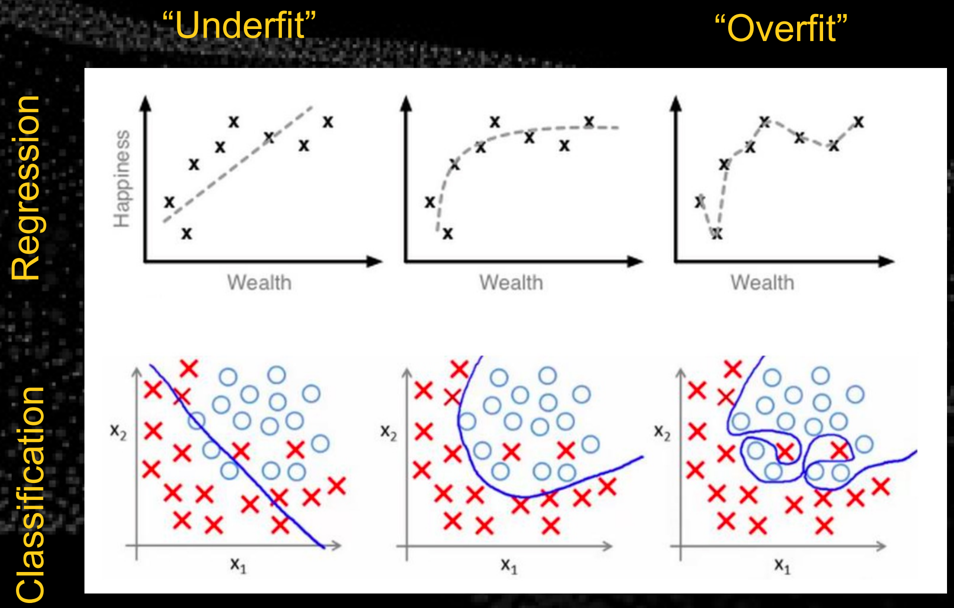
- Optimization setup
  - Optimizer choice, optimizer parameters
  - Learning rate, learning rate scheduler
  - Loss penalties (L2 regularization)
  - Batch size, number of epochs
  - Loss function

HYPERPARAMETERS AND OVERFITTING

- Some hyperparameters can have a predictable effect on overfitting
- Overfitting increases when model complexity increases
- Selecting an optimal learning rate is challenging

Hyperparameter	Increases overfitting when
Number of model parameters (depth, layer size, convolution kernel size)	More parameters
Number of data features	More extraneous features
Batch size	Decreased
Number of epochs	Increased

OVERFITTING & UNDERFITTING - REVISITED



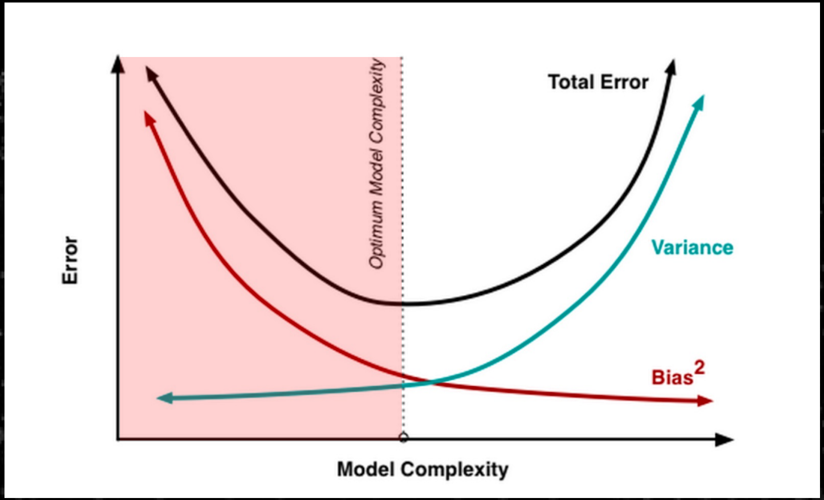
UNDERFITTING

HOW CAN WE TELL?

- High bias, low variance
- High error on the training set

POSSIBLE CULPRITS

- Training set is too small
- Model is too simple
- Training data is too noisy



POSSIBLE SOLUTIONS

- Increase model complexity
- Increase the number of features using feature engineering
- Remove noise from the data
- Increase the number of epochs or training duration

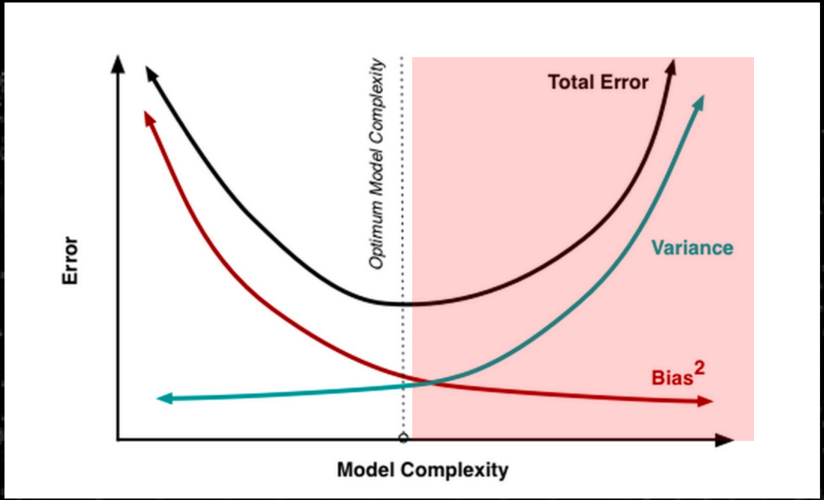
OVERFITTING

HOW CAN WE TELL?

- High variance, low bias
- High error on the validation set

POSSIBLE CULPRITS

- Model is too complex
- Training data is too simple or fails to represent the validation/test data

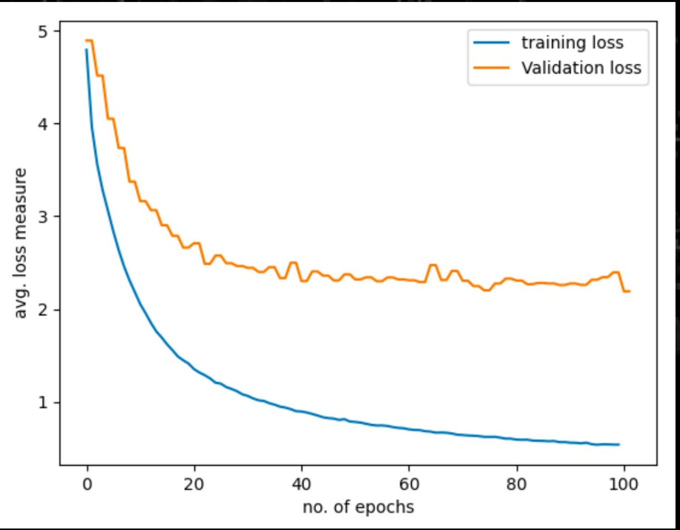


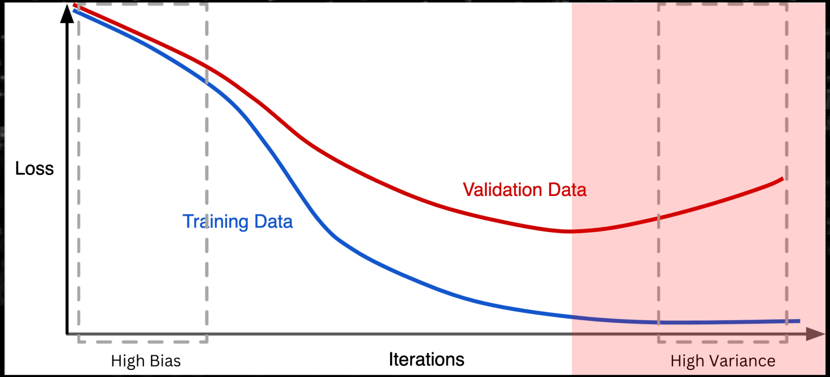
POSSIBLE SOLUTIONS

- Decrease model complexity
- Find more training data!
- Use techniques such as early stopping, regularization, dropout

INTERPRETING LOSS CURVES

- We plot the number of iterations against the loss
- Training loss < validation loss, since we optimize on training data
- If validation loss starts to increase, indicates we fit too much to random noise in the training data and not generalizing well to validation data





INTERACTIVE ACTIVITY

- The model shown in the code-along activity is definitely overfit on the training data and has a gap in performance on the testing data
- Your job is to try out different hyperparameters to make a model that achieves the best performance on the held-out testing data
- Submit scores here: https://keepthescore.com/board/qrqqhjgqwhxbe/

UPCOMING EDUCATION SESSIONS

- Dive into Deep Learning (Wednesday at 5pm)
- Recording and slides will be posted to Github
- More resources will be posted on our discord
    - https://discord.gg/46KUMNGE8J