CMSC 373 Artificial Intelligence
Fall 2025
14-Training

Deepak Kumar
Bryn Mawr College

Review

Adam
Backpropagation
Bias
Binary Cross Entropy
Categorical Cross Entropy
Epochs
Exponential
Forward Pass
Full Batch SGD
Gradient Descent
Hyperparameters
Labelled Dataset
Learning Rule
Loss Function
Mean Absolute Error
Mean Squared Error
Mini Batch SGD
Model
Optimizer
Parameters
Relu
RMSProp
SGD
Sigmoid
Softmax
Tanh
True SGD

Backpropagation Network (Updated)

- **Net Input**

$$\mathrm{I} = \sum_{i=1}^{i=n} w_i x_i + \overline{\boldsymbol{b}}$$

- **Activation Functions**

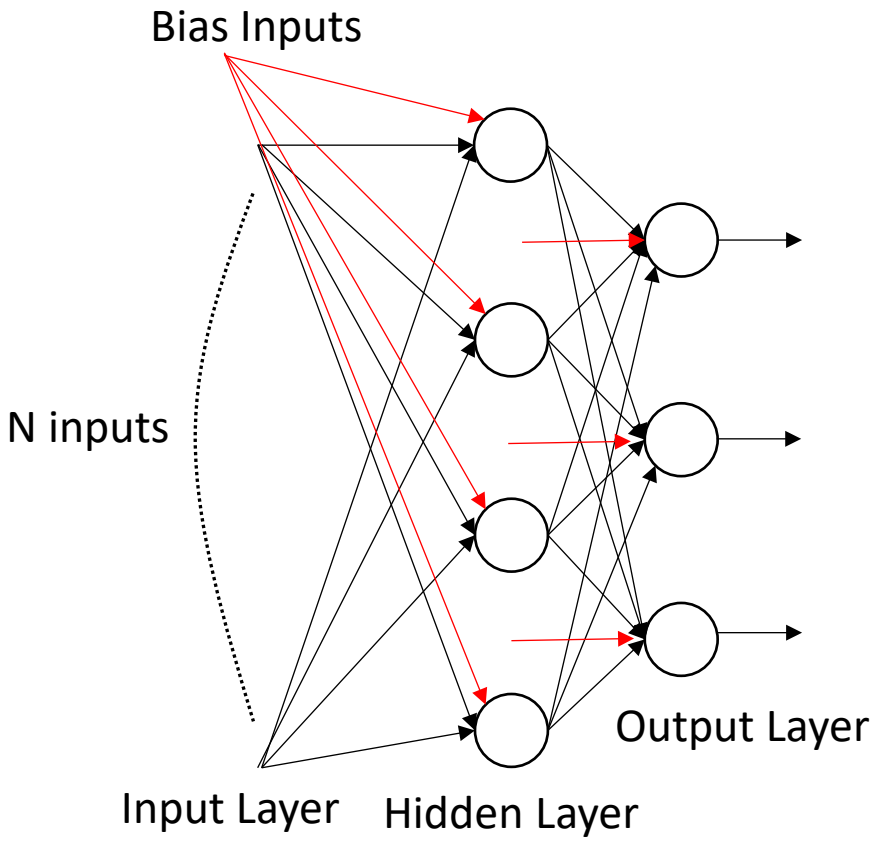
  Sigmoid, Relu, Softmax, tanh, exponential, etc.

- **Loss Functions**

  Binary Cross Entropy, Categorical Cross Entropy, Poisson, KL Divergence, Mean Squared Error, Mean Absolute Error, Cosine Similarity, etc.

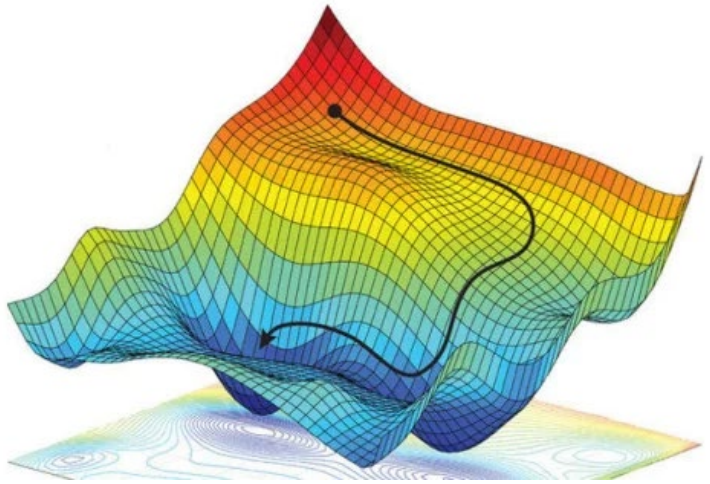
- **Optimizer (Learning Rule)**

  SGD, RMSprop, AdaGrad, Adam, Adadelta, Adamax, Namad, etc.

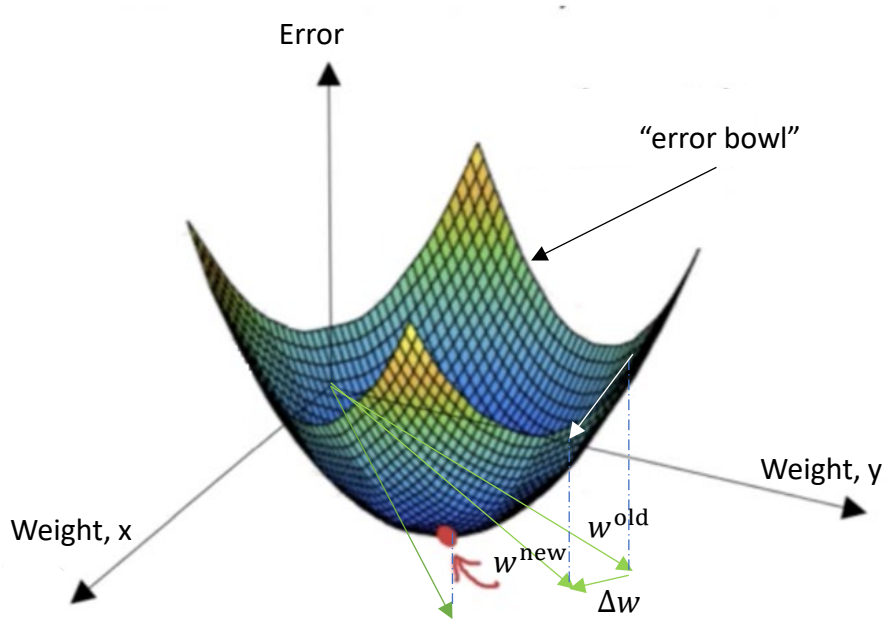


Backpropagation: Gradient Descent

- In higher dimensional weight vectors (typical ML situations), the error surface can be quite complex.

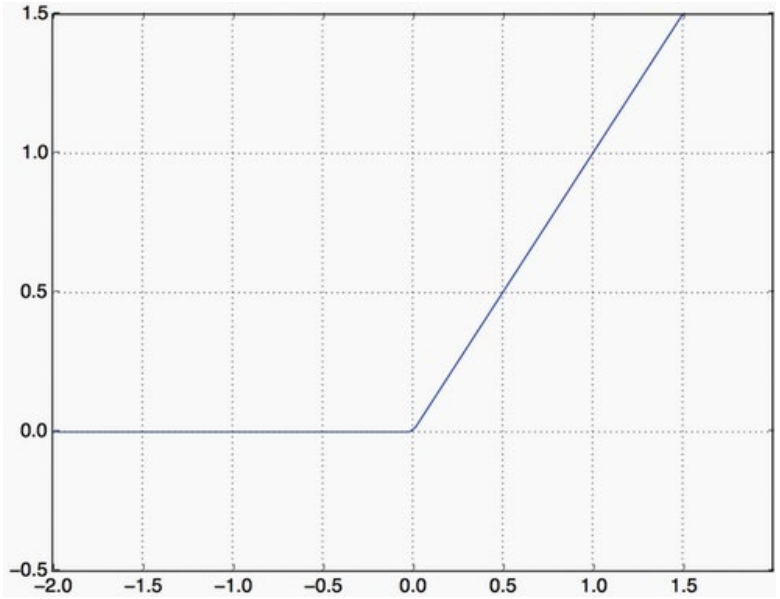


From: https://poissonisfish.com/2023/04/11/gradient-descent/

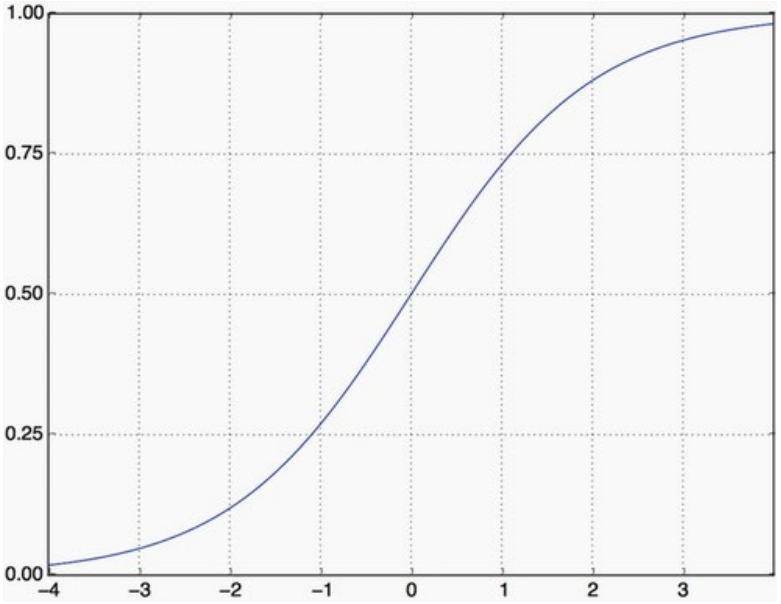


From: https://builtin.com/data-science/gradient-descent

Popular Activation Functions



Relu – Rectified Linear Unit: $f(I) = \max(0, I)$



Sigmoid: $\mathrm{f(I)} = \frac{1}{1+e^{\mathrm{I}}}$

Both are non-linear.
Relu is easier to compute than Sigmoid. Hence commonly used in Deep networks.
Sigmoid constrains activation, Relu doesn't.
Networks with Relu activations tend to show better convergence over Sigmoid.
Too many zero activations in Relu networks can be a problem.

Popular Activation Functions

- **Softmax**

  Converts a vector of values to a vector of probabilities (i.e. a probability distribution).
  Elements of the output vector are in range (0,1) and sum to 1.
  Typically used as a last layer in a classification network.

Computation(for an Output vector $\vec{\boldsymbol{o}}$ of length, n, with inputs I:

$$o_i = \frac{e^{I_i}}{\sum_{j=1}^{n} e^{I_j}}$$

- Enables use of a **cross-entropy loss function** (since the outputs are a probability distribution).

Backpropagation Network (Updated)

- **Net Input**

$$\mathrm{I} = \sum_{i=1}^{i=n} w_i x_i + \overline{\boldsymbol{b}}$$

- **Activation Functions**

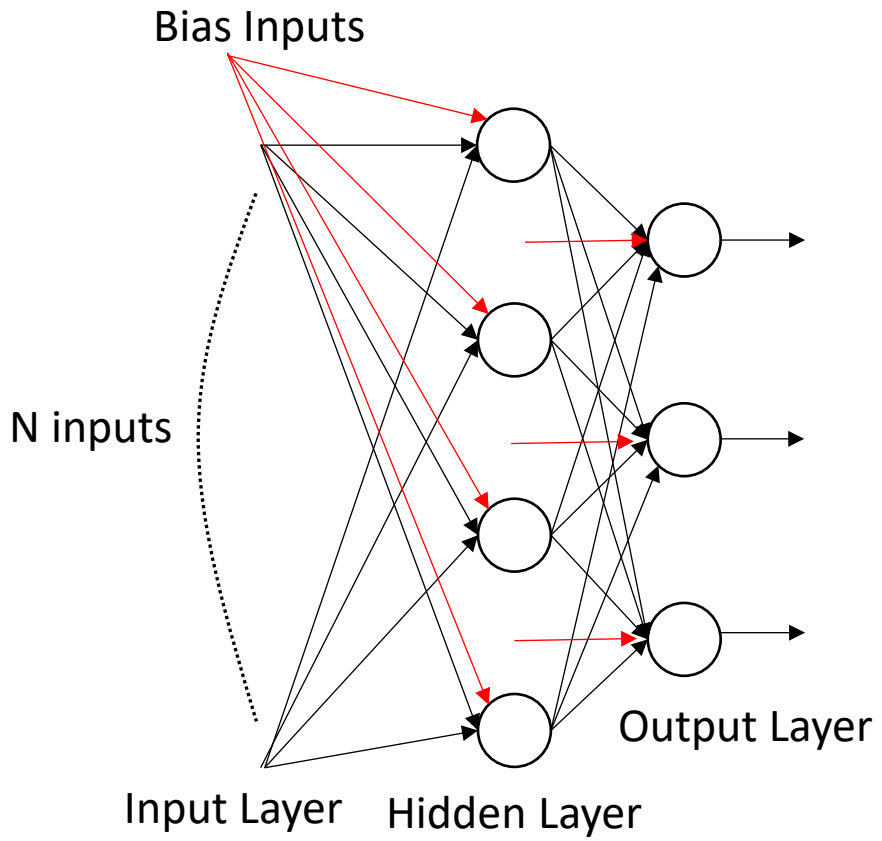
  Sigmoid, **Relu**, **Softmax**, tanh, exponential, etc.

- **Loss Functions**

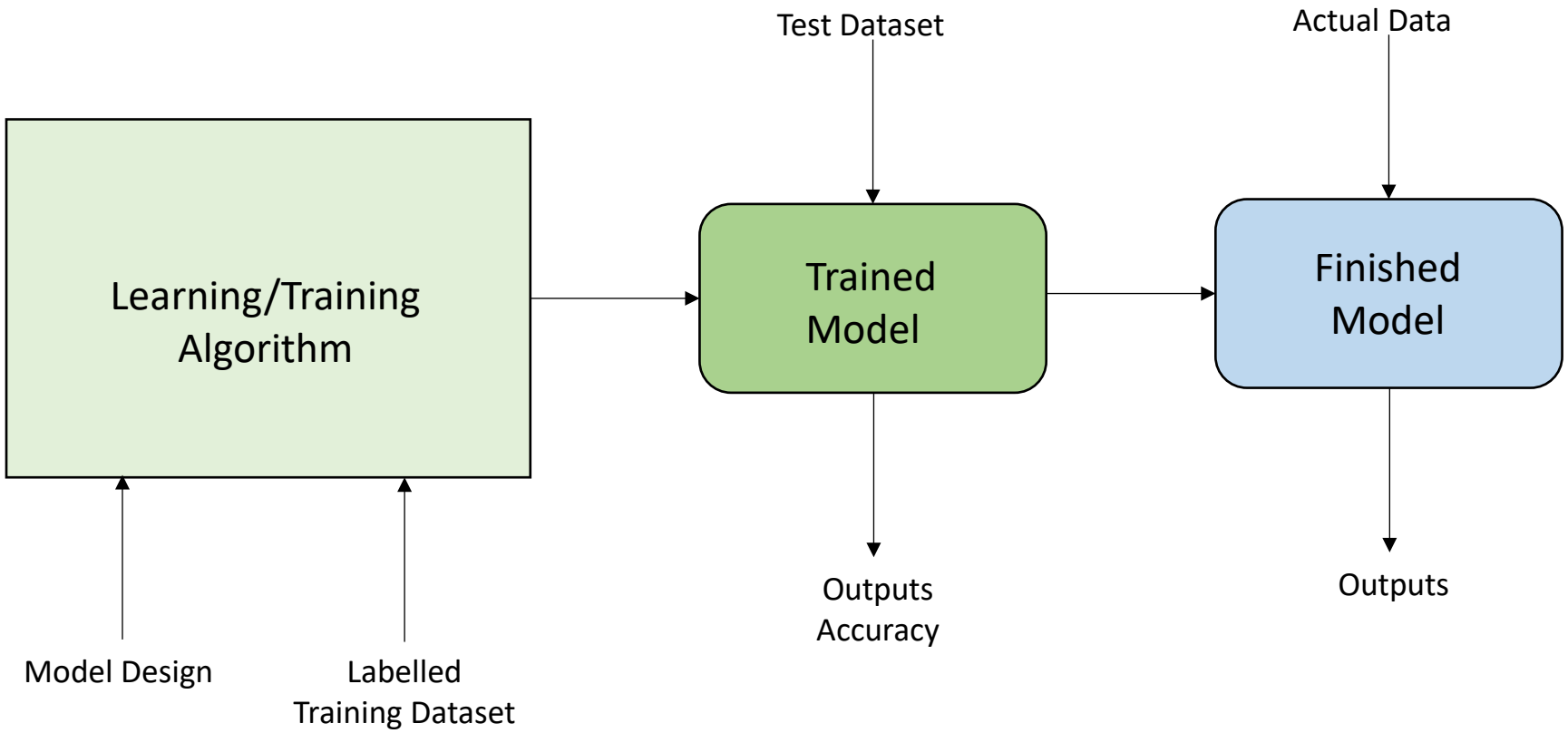
  Binary Cross Entropy, **Categorical Cross Entropy**, Poisson, KL Divergence, **Mean Squared Error**, Mean Absolute Error, Cosine Similarity, etc.

- **Optimizer (Learning Rule)**

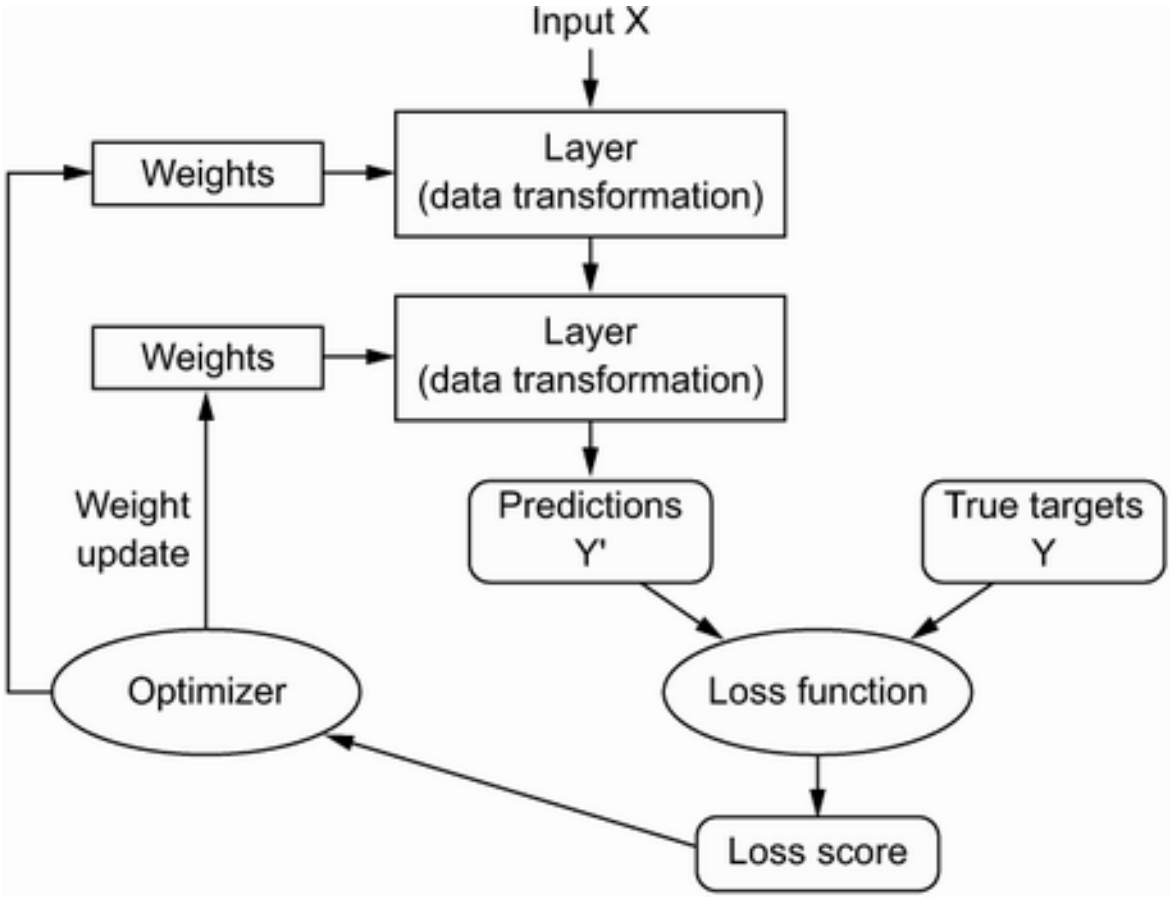
  SGD, **RMSprop**, AdaGrad, Adam, Adadelta, Adamax, Namad, etc.
  See: Optimizers in Deep Learning.



The Learning Paradigm

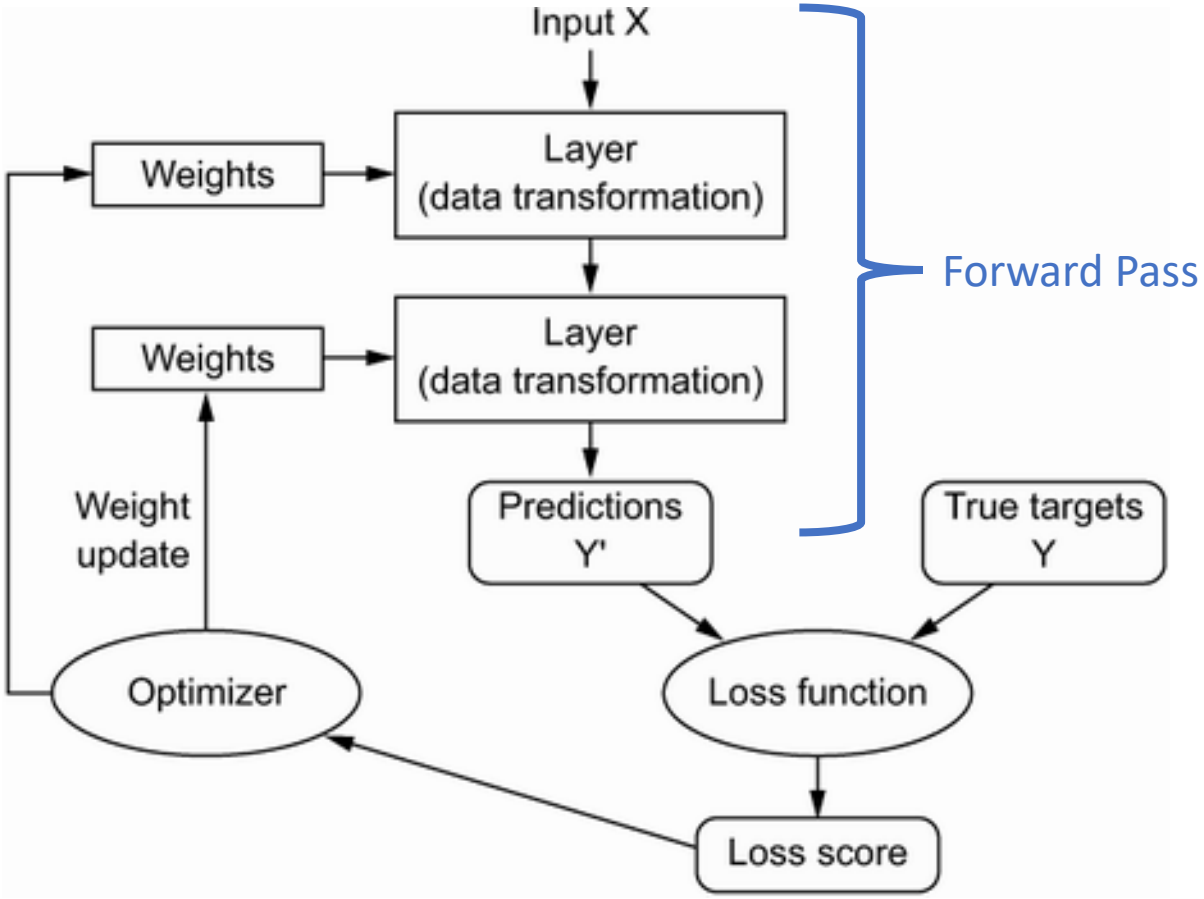


How NN Learning Works



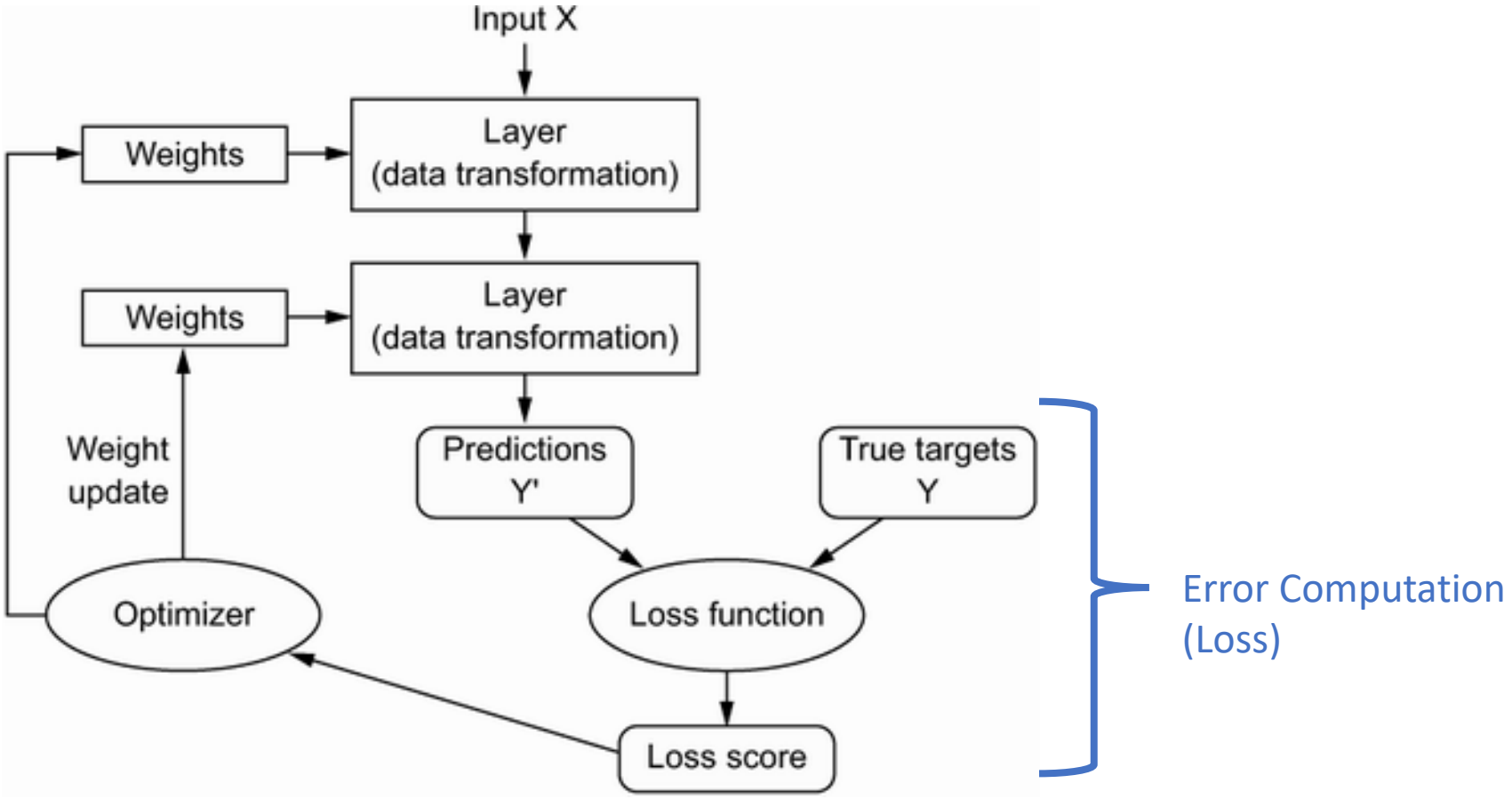
From: Chollet, 2021.

How NN Learning Works



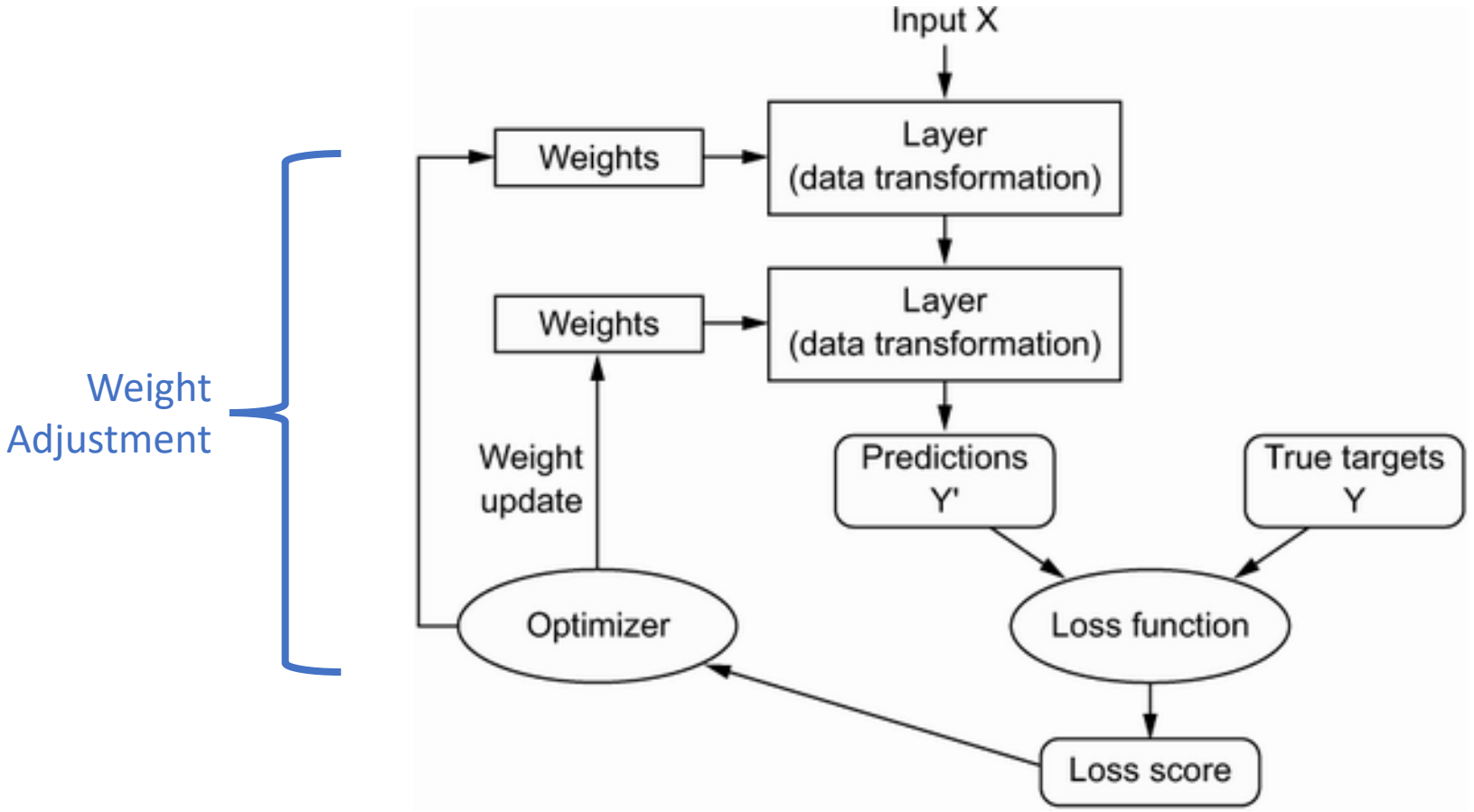
From: Chollet, 2021.

How NN Learning Works



From: Chollet, 2021.

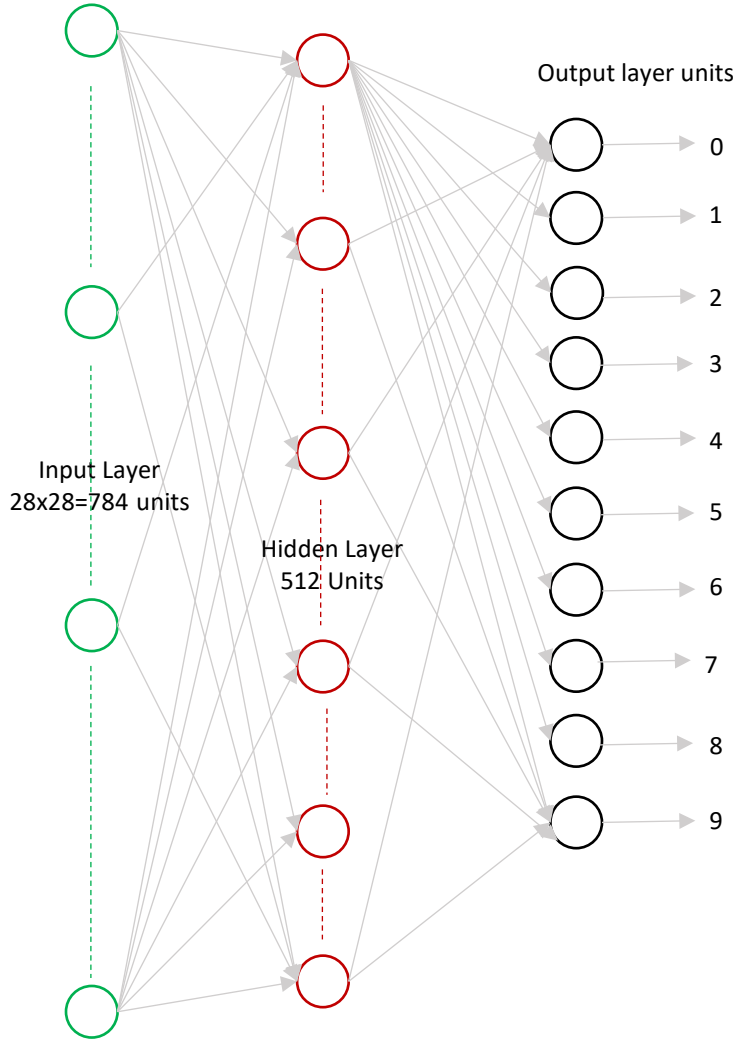
How NN Learning Works



From: Chollet, 2021.

MNIST Digit Recognition: The Design

- **A 3-layer network:** input, hidden, and output layers
- **Input** will be 28x28=784 units
- 10 **outputs**, one for each digit. 512 units in **hidden layer**.
- **Parameters**
  784x512 + 512 (bias) + 512x10 + 10 (Bias) = 407,050
- **Hyperparameters**
  3 Layers
  784 Units in input layer
  512 units in hidden layer (Sigmoid/Relu)
  10 Units in output layer (Softmax)
  Loss: Mean Squared Error/Categorical Cross-Entropy
  Optimizer: (SGD, RMSProp),
  Learning Rate: β (decided by the optimizer)
  Accuracy Metric: Accuracy (% Correct)
  # epochs (? Start with a max of 10)



Commonsense Baseline Accuracy

- Before we begin, we should always estimate a baseline accuracy.

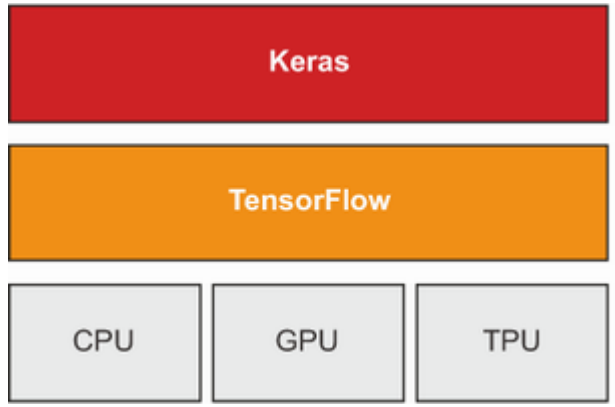
  That is, given any data set, if we were to randomly assign an output, what would be the accuracy?

  For binary classification, the baseline is 50% accuracy.

  For MNIST Digits classification, the baseline is 10% accuracy.

- Any neural network model we train should be able to perform far better!

Introducing Keras



From: Chollet, 2021.

- Deep Learning API for Python (2016-17)
- Built on top of TensorFlow (2015)
- Can run on a typical CPU, or can be accelerated with specialized hardware, if available. GPUs (Graphics Processing Units), TPUs (Tensor Processing Units)
- Makes Neural Network design, implementation, and exploration akin to building with LEGOs!

Typical Keras Workflow

- **Acquire, prepare, and load the dataset**
  Keras has a few predefined datasets available: MNIST Digits, CIFAR10, CIFAR100, IMDB Reviews for sentiment classification, Reuters Newswire classification, Fashion MNIST, Boston Housing price regression (see https://keras.io/api/datasets/)
- **Design and Build the Model**
  How many layers to use? How many units in each later? What activation function to use? (see https://keras.io/api/layers/activations/)
- **Compile the Model**
  Decide which **optimizer** to use, **loss function**, **accuracy metric**
- **Train/Fit the Model**
  Provide the training data and its labels, number of epochs to train, batch size
- **Test/Validate the Model**
  Use the test data to test how well the trained model performs

Over to Colab…

- See Lab for Recognizing Handwritten Digits

  (https://colab.research.google.com/drive/1fmKbmOnDpS4D_WMYEj1XjRjPt6fJ5GWz?usp=sharing)

Reflection from Colab Work

- We defined our network design and all the hyperparameters.
- We used **Sigmoid** for hidden layer activations and **Softmax** for output layer activations.
- We used **Sparse Categorical Cross Entropy** as our loss function, **RMSProp** as our optimizer and **accuracy** (% correct) as our accuracy metric.
- We trained the model for **10 epochs** using **mini batch SGD**.
  Accuracy obtained: XXX%
  Time for 10 epochs: XXX seconds
- We tested the trained model for 5 test inputs to examine the results for correctness.
  # Correct/5
- We checked the image of one of the test digits to confirm.
- We performed an **evaluation of the model** on the test dataset.
  Loss: XXX
  Accuracy: XXX
  Time Taken: XXX seconds

Testing & Validation

- Unlike the Perceptron, a NN training may not learn 100% of the training data correctly. You must evaluate and decide on an acceptable accuracy level for the model.
- Keras facilitates an examination of the model during and after training.

```
>>> model.fit(x, y, epochs=20, batch_size=128)
Epoch 1/20
469/469 [==============================] - 7s 4ms/step - loss: 0.3627 - accuracy: 0.8951
Epoch 2/20
469/469 [==============================] - 2s 3ms/step - loss: 0.2128 - accuracy: 0.9366
. . .
Epoch 20/20
469/469 [==============================] - 2s 3ms/step - loss: 0.0844 - accuracy: 0.9747
```

- As you can see, The loss is decreasing, and accuracy is increasing as the epochs progress.
- **Training Loss & Training Accuracy**
  The value of the loss function during training is **Training Loss**. The accuracy during training is **Training Accuracy**.

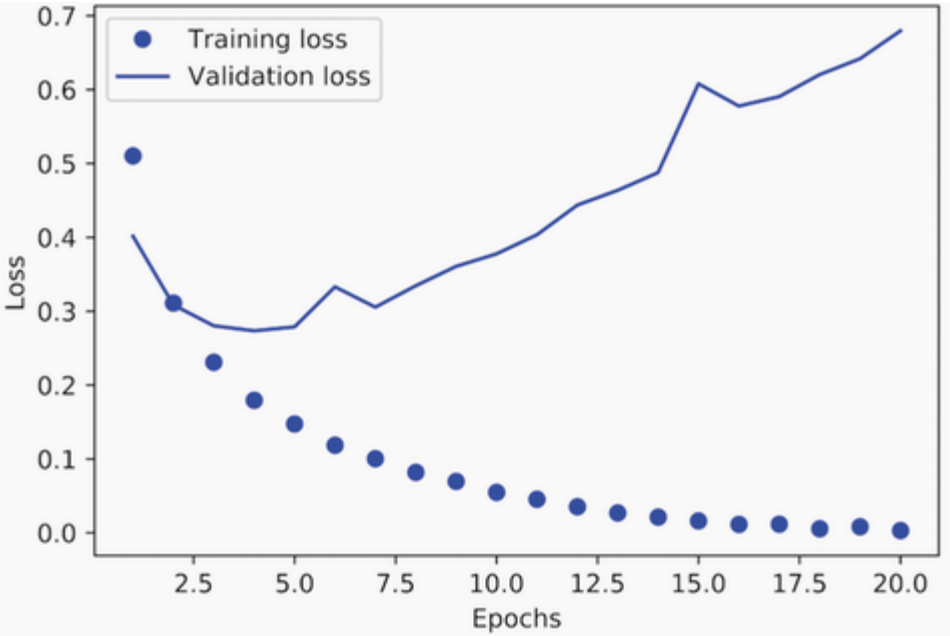
- **Validation Loss & Validation Accuracy**
  The value of the loss function during testing is **Validation Loss**. The accuracy during testing is **Validation Accuracy**.

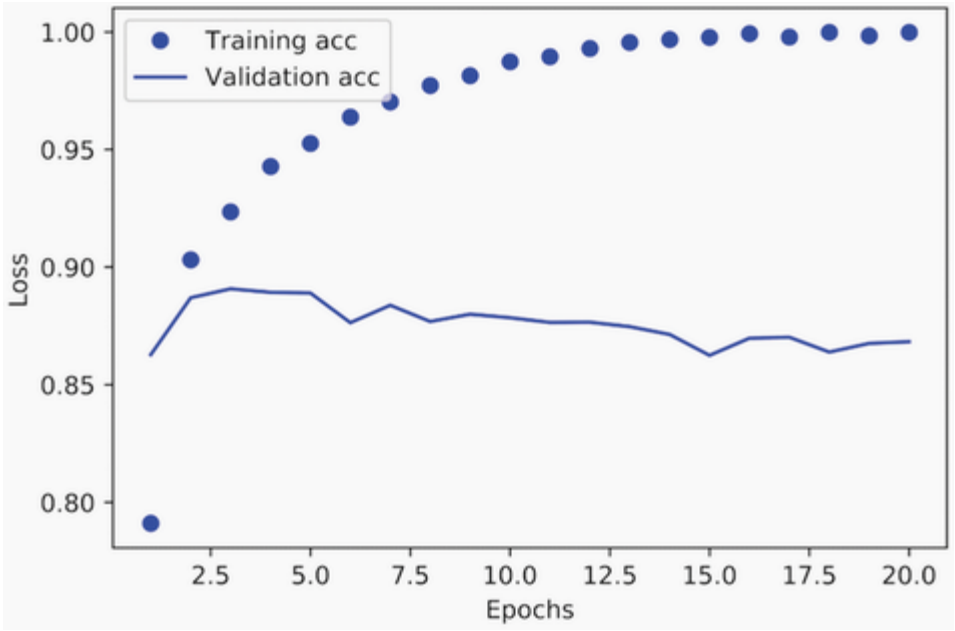
Over to Colab…

- Testing the MNIST Model

  (https://colab.research.google.com/drive/1upz1hi9It1hiGZcqIqUEDDbtVV2jjmWR#scrollTo=1AKjUpcMP97q)

Training & Validation Loss and Accuracy

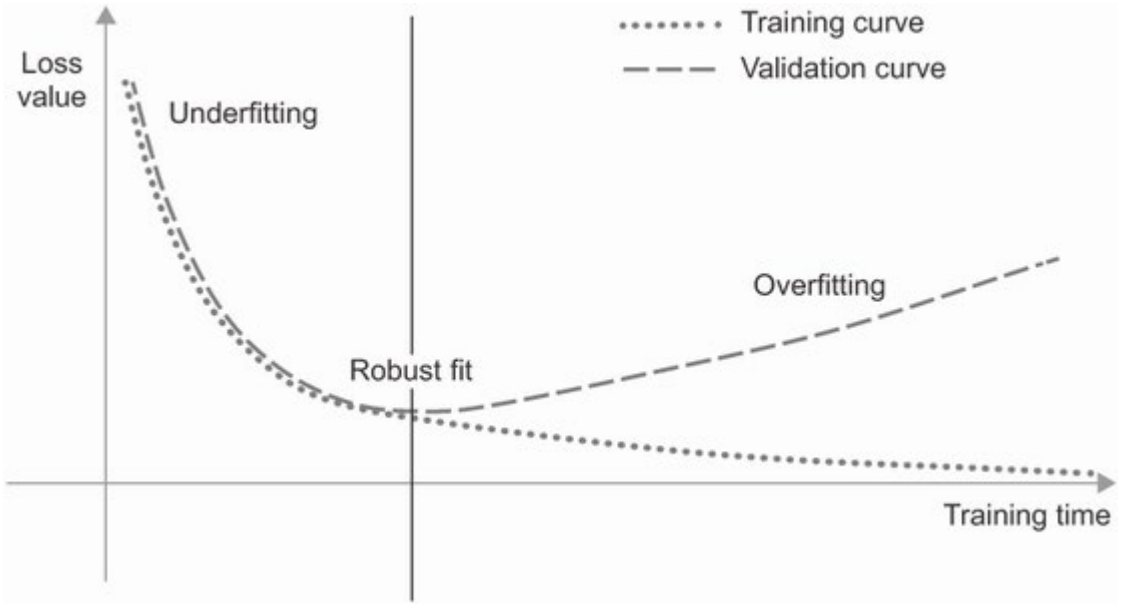




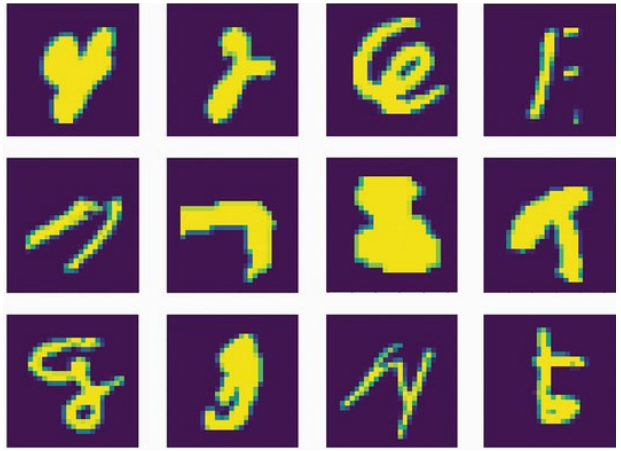
- Notice that the validation loss and validation accuracy both diverge after ~ 4th epoch.
- The model performs better on training data doesn't necessarily do well on the testing data.
- This is called **overfitting**.

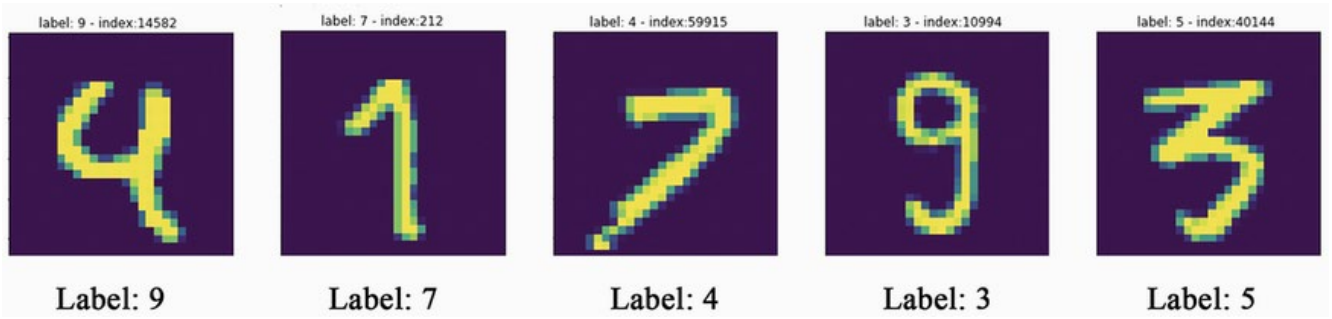
Underfitting and Overfitting

- Initially, as the training proceeds, the lower the loss on training data, the lower the loss on test data. This is **underfitting**. The network hasn't yet modeled the all the patterns in the training data.
- As training proceeds further, the testing stops improving and starts degrading: This is **overfitting**. The network is starting to learn patterns specific to the training data.
- Overfitting can occur when the training data is noisy, ambiguous, or involves uncertainty.

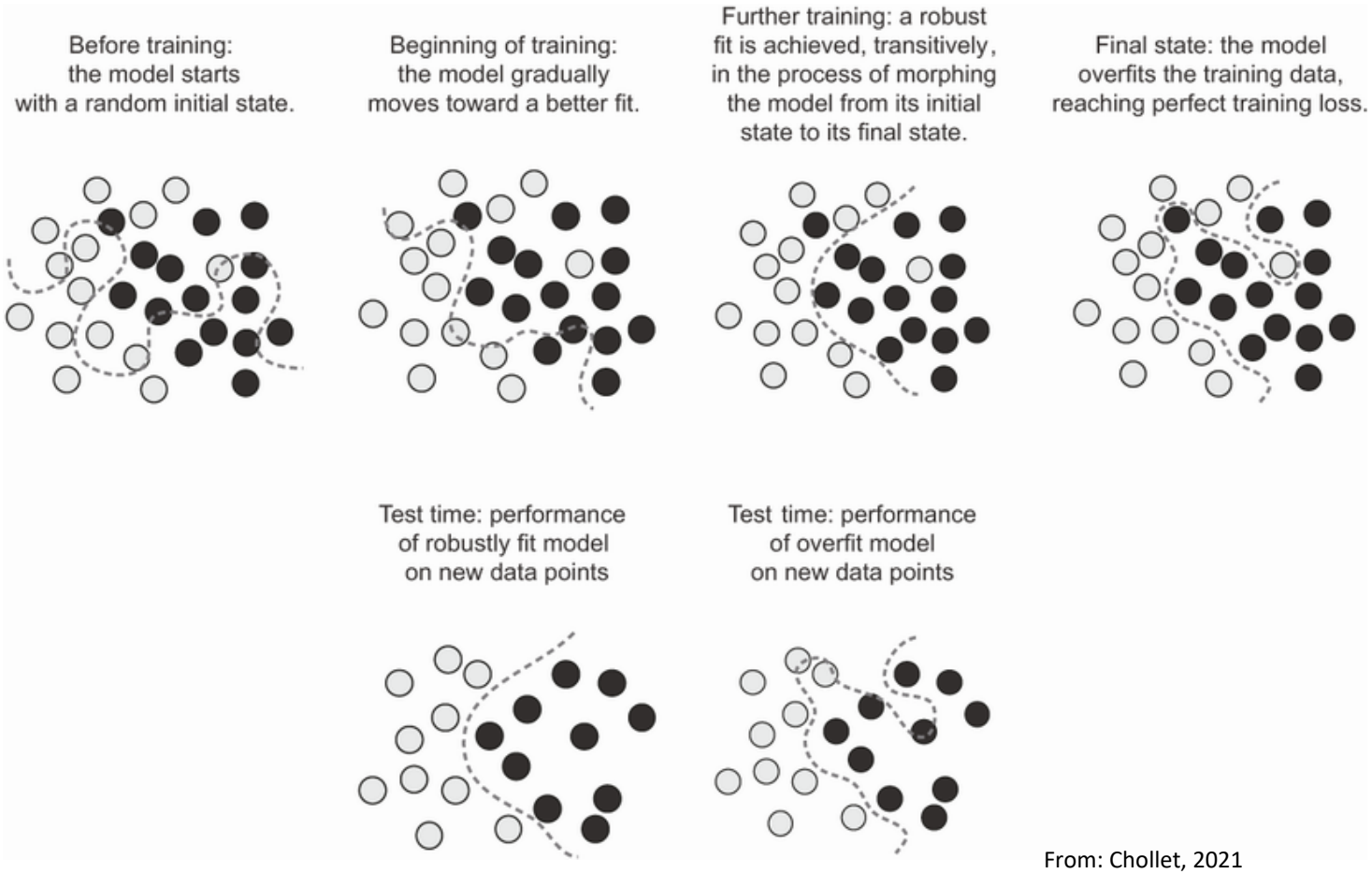


Noisy Data



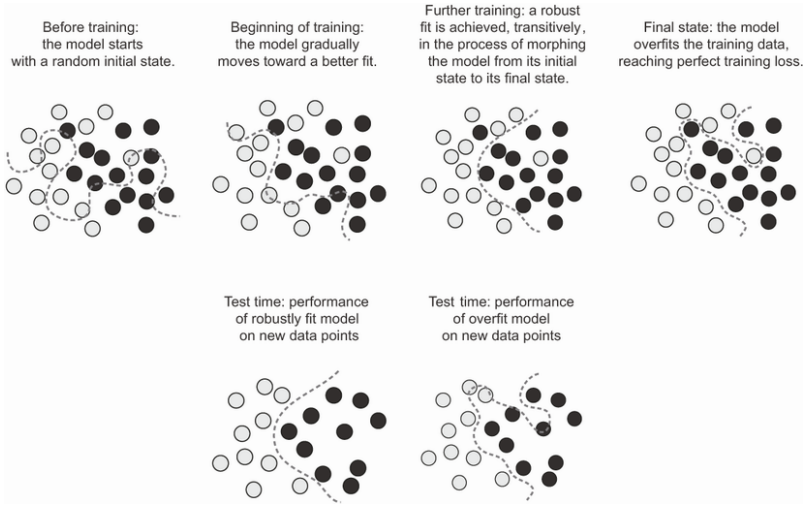

label: 9 - index:14582
label: 7 - index:212
label: 4 - index:59915
label: 3 - index:10994
label: 5 - index:40144
Label: 9
Label: 7
Label: 4
Label: 3
Label: 5

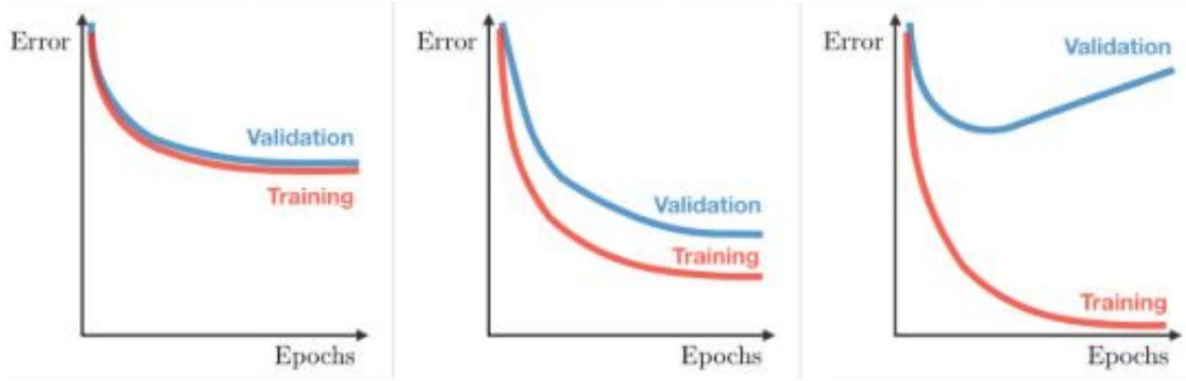
From a Random Model to Overfitting or Robust Fit



From: Chollet, 2021

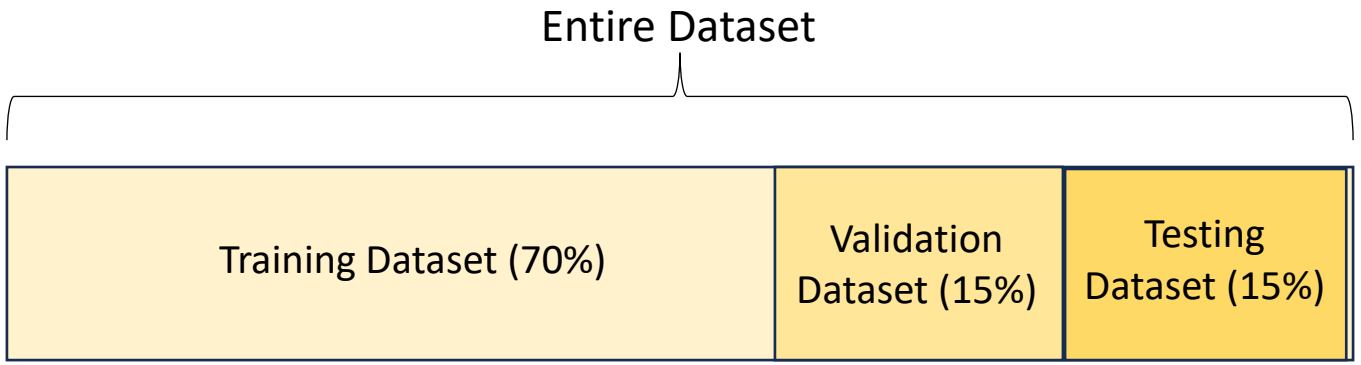
From a Random Model to Overfitting or Robust Fit





From: Chollet, 2021, and
https://medium.com/@datascienceeurope/do-you-know-overfitting-and-underfitting-f27f87ac2f37

Training Data, Validation Data, Testing Data



- **Training dataset** is for use during training
- **Validation dataset** is to estimate loss/accuracy of the model to tune the hyperparameters
- **Testing dataset** is for evaluating the model after training. (how well does it generalize?)

Review

Adam
Backpropagation
Bias
Binary Cross Entropy
Categorical Cross Entropy
Commonsense Baseline Accuracy
Epochs
Exponential
Forward Pass
Full Batch SGD
Gradient Descent
Hyperparameters
Keras
Labelled Dataset
Learning Rule
Loss Function
Mean Absolute Error
Mean Squared Error
Mini Batch SGD
Model

Optimizer
Overfitting
Parameters
Relu
RMSProp
Scikit-Learn
SGD
Sigmoid
Softmax
Tanh
Testing Data
Training Accuracy
Training Data
Training Loss
True SGD
Underfitting
Validation Accuracy
Validation Data
Validation Loss

References

- M. Caudill and C. Butler: Understanding Neural Networks, Volume 1, MIT Press, 1993.
- F. Chollet: *Deep Learning with Python, Second Edition*, Manning2021.
- A Geron: *Hands-on Machine Learning with SciKit-Learn, Keras and TensorFlow*, Oreilly, 2019.
- M. Mitchell: *Artificial Intelligence: A Guide For Thinking Humans*, Farrar, Strouss, Giroux, 2019.
- M. Wooldridge: *A Brief History of Artificial Intelligence*. Flatiron Books, 2020.