

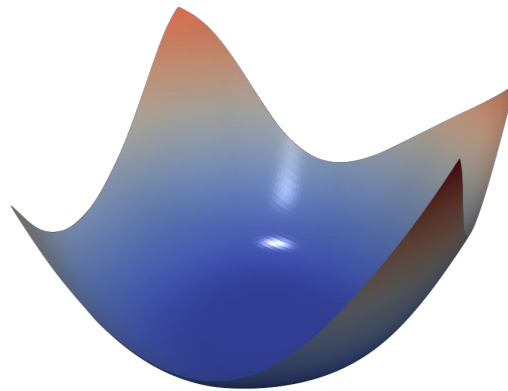
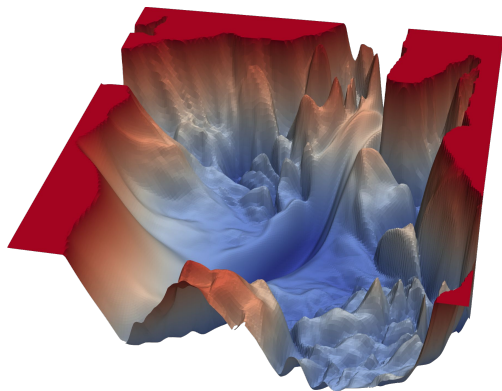
ICME Summer Workshops 2020

Introduction to Deep Learning

Session 2: 10:45—12:00 PM

Instructor: Sherrie Wang

icme-workshops.github.io/deep-learning



Workshop Schedule

Session 1 (9:00—10:30 AM)

- Introduction
- Current state-of-the-art in deep learning
- Math review
- Fully connected neural networks

Session 2 (10:45—12:00 PM)

- Loss functions
- Gradient descent
- Backpropagation
- Overfitting and underfitting

Lunch (12:00—2:00 PM)

Session 3 (2:00—3:15 PM)

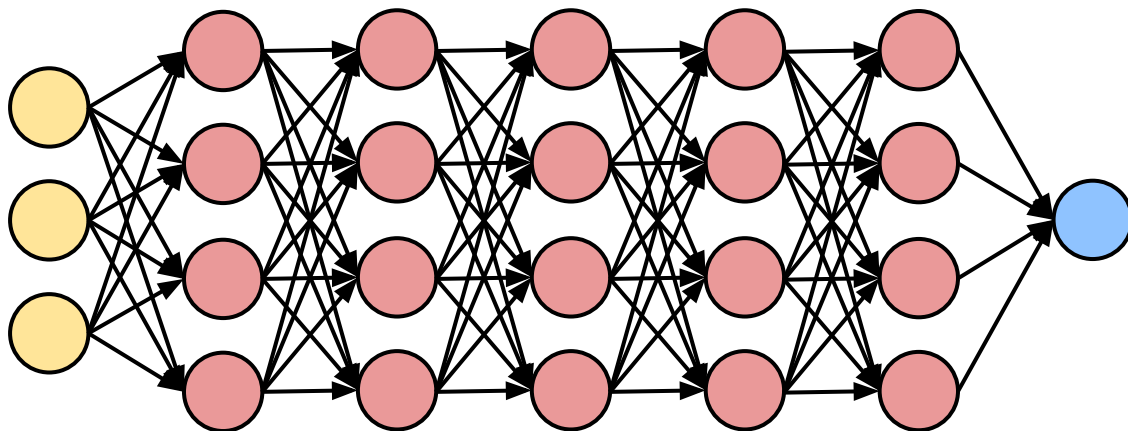
- Convolutional neural networks
- Recurrent neural networks
- Other architectures
- Deep learning libraries
- Hands-on coding session in Tensorflow

Session 4 (3:30—4:45 PM)

- Hands-on coding session in Keras
- Hands-on coding session on transfer learning
- Failures of deep learning

Loss Functions

Training neural networks



$$g \left(\begin{pmatrix} 2 & 4 & 2 \\ 1 & 3 & 7 \\ 0 & 7 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix} + \begin{pmatrix} -3 \\ 0 \\ 2 \end{pmatrix} \right)$$

Trainable parameters

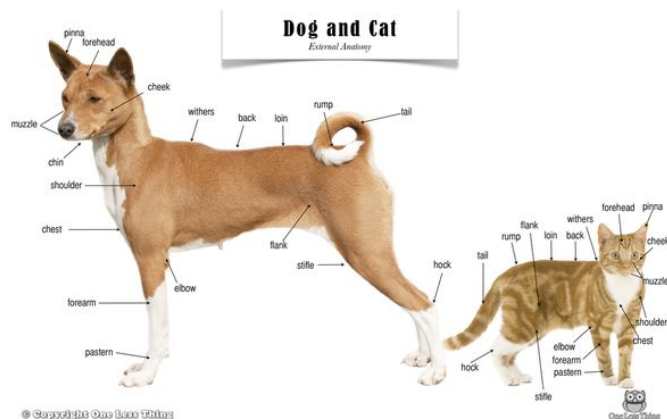
Trained
how?

Training neural networks

Machine learning: builds a model based on “training data” in order to make predictions without being explicitly programmed to do so

Not machine learning

Human says: “dogs are like this, and cats are like this...”



Machine learning

“Here’s a bunch of pictures of cats and dogs. Figure out how to sort them.”



<https://mc.ai/deep-learning-vs-machine-learning-a-simple-explanation/>

Training neural networks

Dataset



training

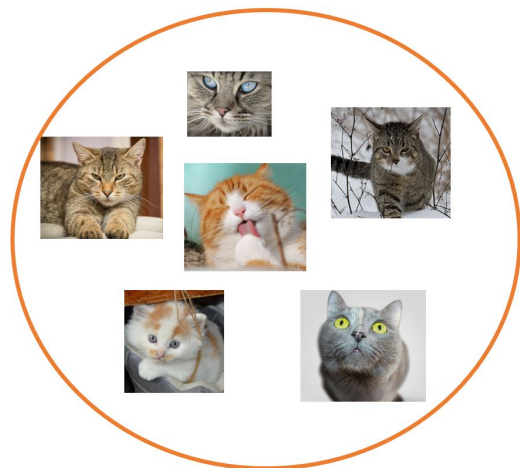


Weights & biases

$$\begin{pmatrix} 2 & 4 & 2 \\ 1 & 3 & 7 \\ 0 & 7 & 3 \end{pmatrix} \begin{pmatrix} -3 \\ 0 \\ 2 \end{pmatrix}$$

Supervised learning

Dataset has labels



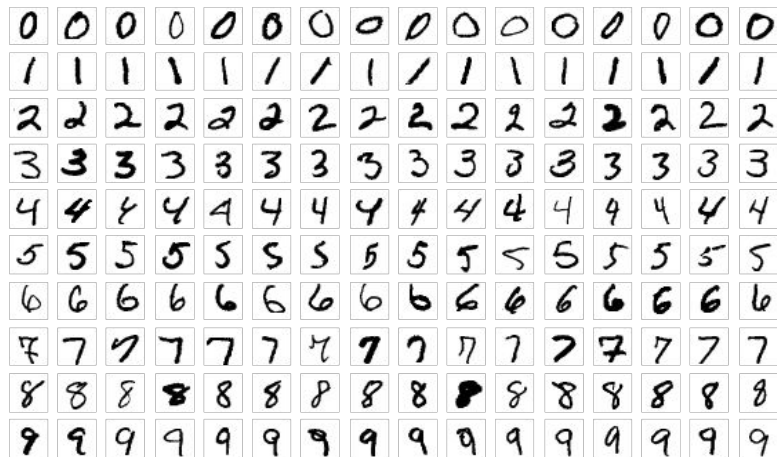
Cat



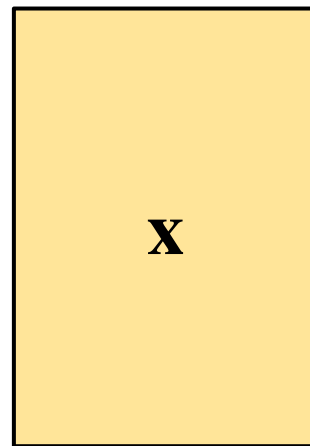
Dog

Supervised learning

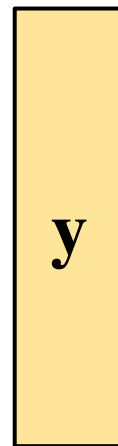
MNIST Dataset (Handwritten digits)



Each image is 28 x 28 pixels

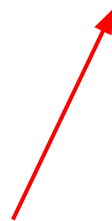


60,000 x 784



60,000 x 10

One-hot
encoding



The goal of training

Intuitively, we want to tweak the weights of the neural network until the network predicts the correct output for most of the examples in the training set.

Neural network #1

$$\text{softmax} \left(\begin{pmatrix} 2 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) = \begin{pmatrix} 0.98 \\ 0.02 \end{pmatrix}$$

Weights Input Prediction

Neural network #2

$$\text{softmax} \left(\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) = \begin{pmatrix} 0.12 \\ 0.88 \end{pmatrix}$$

Weights Input Prediction

True label $\mathbf{y} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

We prefer network #1 to #2

Optimization formulation

We will frame this as an optimization problem:

$$\max_{\mathbf{W}_i, \mathbf{b}_i} \text{accuracy}(\text{DNN}(\mathbf{W}_i, \mathbf{b}_i))$$

Accuracy is the fraction of training examples that the model predicts correctly

Optimization formulation

Accuracy is not a continuous function, so it is hard to directly optimize for maximum accuracy.

So we use **loss functions**.

Think of losses as continuous proxies to accuracy, so that we can formulate it as an optimization problem and solve it.

Loss functions

$$\text{True label } \mathbf{y} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \text{Predicted probabilities} = \begin{pmatrix} 0.2 \\ 0.7 \\ 0.1 \end{pmatrix}$$

In English, we want to **reward the neural network** for predicting class 2 with higher probability than classes 1 and 3, while incentivizing it to become even better at making this distinction

Cross entropy loss

$$\text{Predicted probabilities} = \begin{pmatrix} 0.2 \\ 0.7 \\ 0.1 \end{pmatrix}$$

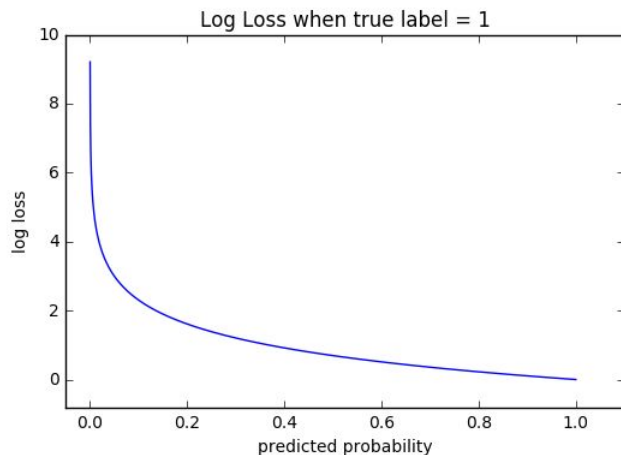
$$-\log(0.7) = 0.35$$

$$\text{Predicted probabilities} = \begin{pmatrix} 0.6 \\ 0.1 \\ 0.3 \end{pmatrix}$$

$$-\log(0.1) = 2.3$$

Cross entropy loss

$$\text{CE}(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_i y_i \log \hat{y}_i$$



Cost function

Cost = Average loss over
training examples

$$\min_{\mathbf{W}_i, \mathbf{b}_i} \text{cost}(\text{DNN}(\mathbf{W}_i, \mathbf{b}_i))$$

Often written $\min_{\mathbf{W}_i, \mathbf{b}_i} \mathcal{J}(\mathbf{W}_i, \mathbf{b}_i)$

The objective function is now a continuous function of the weights and biases

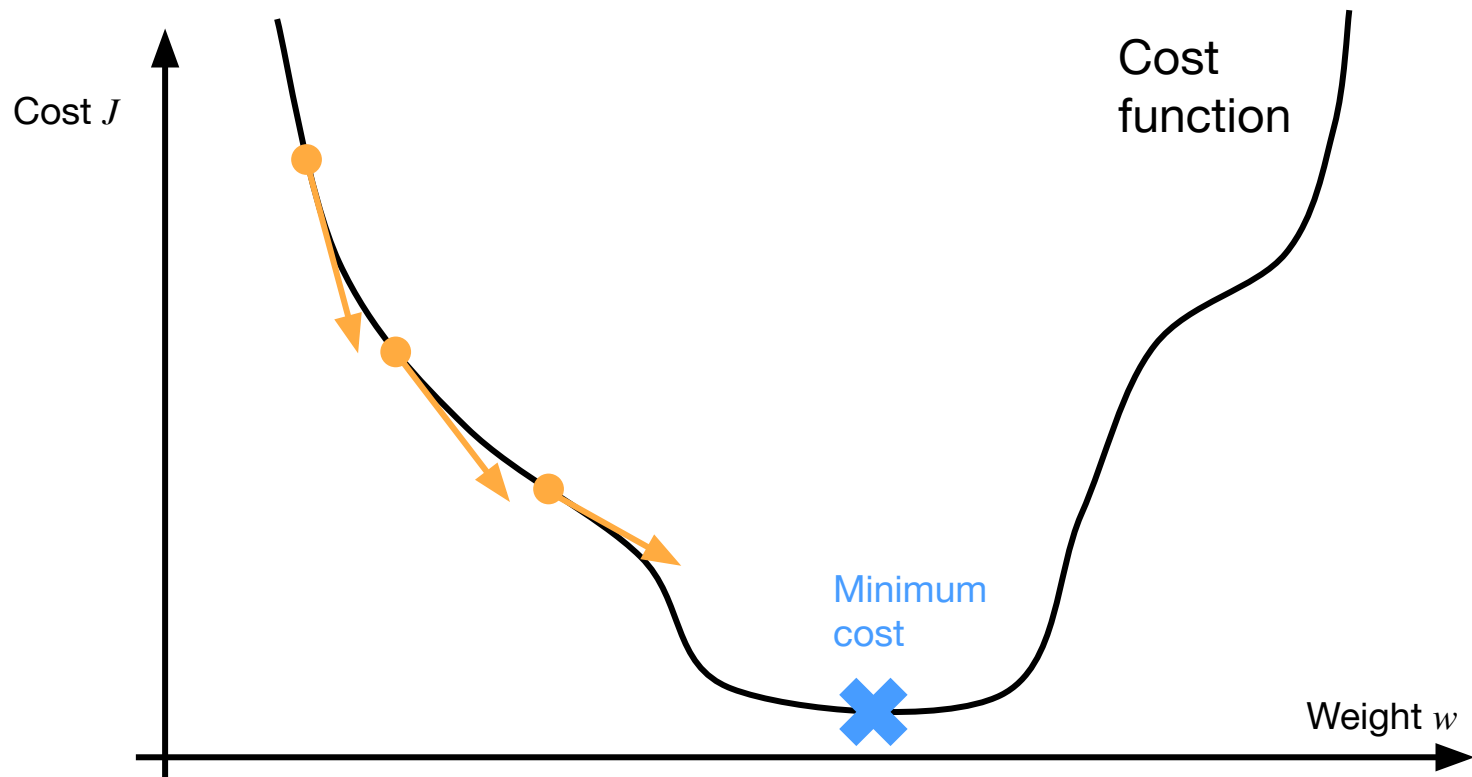
Gradient Descent

How do we solve the optimization problem?

The purpose of reformulating the problem in terms of a cost function is so that we can use gradient-based methods

1. Start with random weights
2. Compute the gradient with respect to the cost function
3. Update the weights so that the cost function decreases
4. Repeat steps 2-3

Gradient descent

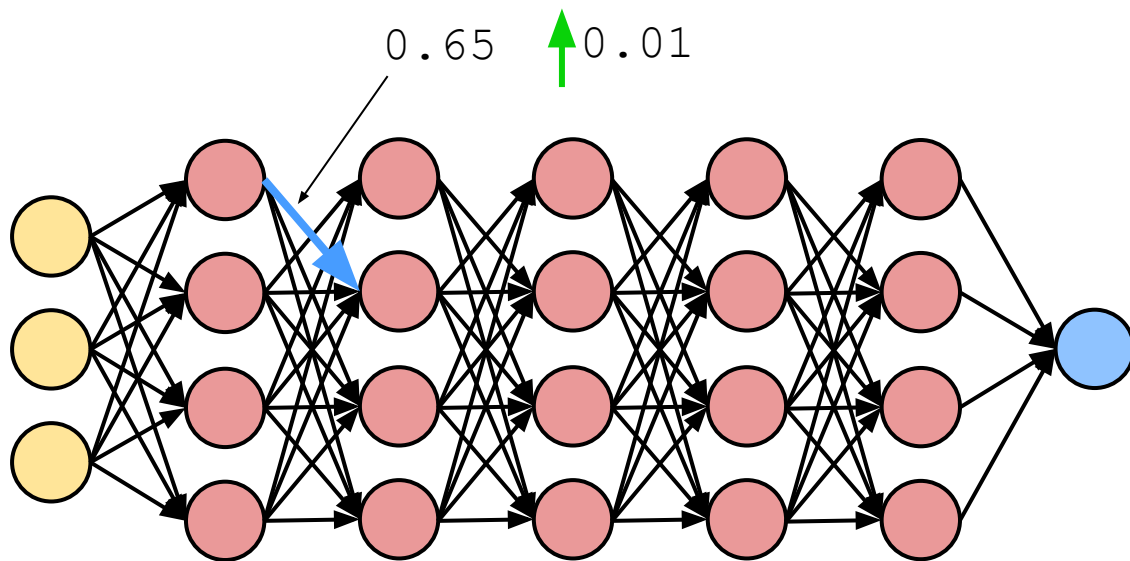


What are gradients?

1. It is a vector whose elements are the partial derivatives associated with every weight and bias in the neural network
2. The partial derivative on a weight = how many units the cost function will change by if the weight is changed by one unit
3. This assumes that the unit of change is small (technically infinitesimally small)

$$\nabla f(x, y, z) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{bmatrix}$$

What are gradients?



$$\text{Gradient} = \frac{\partial \mathcal{J}}{\partial w_i} = \frac{-0.05}{0.01} = -5$$

What are gradients?

Suppose you have a function $F(x,y,z)$

Then we have partial derivatives F_x , F_y , and F_z

$$F(x + \Delta x, y, z) \approx F(x, y, z) + F_x \Delta x$$

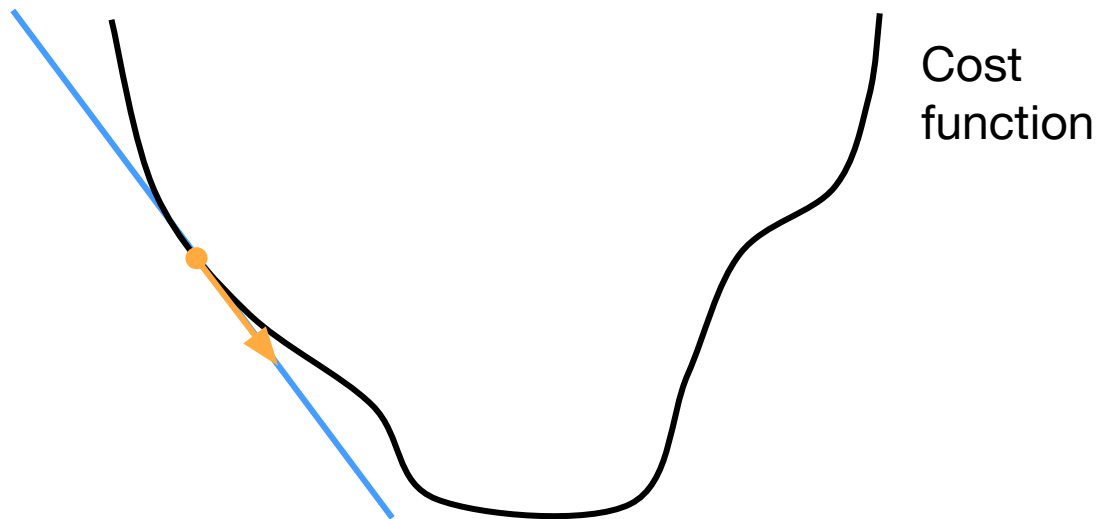
$$F(x, y + \Delta y, z) \approx F(x, y, z) + F_y \Delta y$$

$$F(x, y, z + \Delta z) \approx F(x, y, z) + F_z \Delta z$$

Linear approximation to the cost function

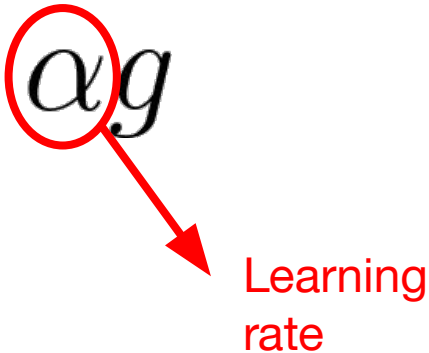
We can build a linear approximation to the function $F(x,y,z)$ using the gradients

$$F(x + \Delta x, y + \Delta y, z + \Delta z) \approx F(x, y, z) + F_x \Delta x + F_y \Delta y + F_z \Delta z$$



Gradient descent

1. Go downhill according to the linear approximation of the cost function
2. But not by too much, as the approximation may not be accurate far away
3. This means updating each weight in proportion to the gradient:

$$w' = w - \alpha g$$


Learning
rate

Backpropagation

How to compute gradients

- Numerically approximating gradients is too slow and computationally expensive
- The neural network is a well-defined mathematical function, so we should be able to differentiate it and calculate the derivatives using the **Chain Rule**



...like this?

$$\begin{aligned} & \frac{d}{dx} \sqrt{(x-1)(x+2)} + 1 \\ &= \frac{d}{dx} \left\{ [(x-1)(x+2)]^{\frac{1}{2}} + 1 \right\} \\ &= \frac{1}{2} \left\{ [(x-1)(x+2)]^{\frac{1}{2}} + 1 \right\}^{\frac{1}{2}-1} \frac{d}{dx} \left\{ [(x-1)(x+2)]^{\frac{1}{2}} + 1 \right\} \\ &= \frac{1}{2} \left\{ [(x-1)(x+2)]^{\frac{1}{2}} + 1 \right\}^{\frac{1}{2}-1} \frac{1}{2} [(x-1)(x+2)]^{\frac{1}{2}-1} \frac{d}{dx} [(x-1)(x+2)] \\ &= \frac{1}{2} \left\{ [(x-1)(x+2)]^{\frac{1}{2}} + 1 \right\}^{\frac{1}{2}-1} \frac{1}{2} [(x-1)(x+2)]^{\frac{1}{2}-1} \left\{ \left[\frac{d}{dx} (x-1) \right] (x+2) + (x-1) \left[\frac{d}{dx} (x+2) \right] \right\} \end{aligned}$$

- This also too tedious, so we use an algorithm called **backpropagation**

Computational graph

A way to organize mathematical expressions

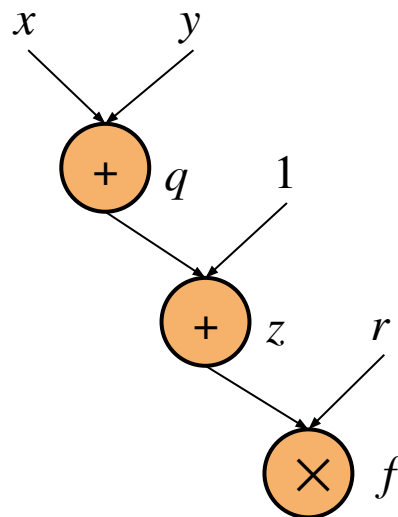
Mathematical expressions

$$x + y = q$$

$$q + 1 = z$$

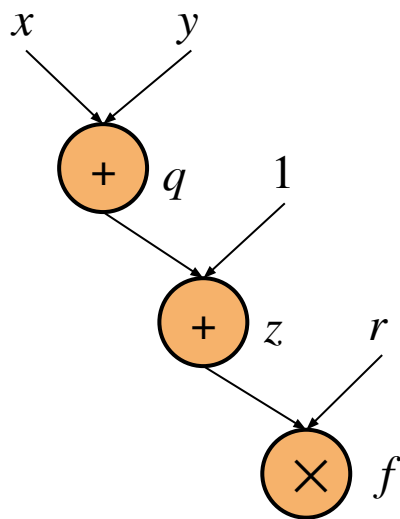
$$z \times r = f$$

Computational graph



Each circle denotes an operation

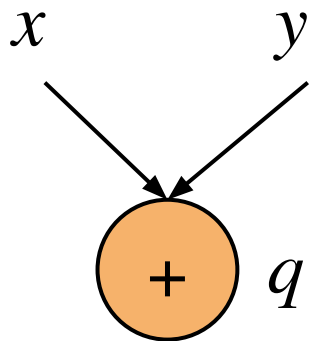
Gradients on computational graphs



Suppose we want to know $\frac{\partial f}{\partial x}$.

We can find this by chaining together derivatives at each operation.

Gradients on computational graphs

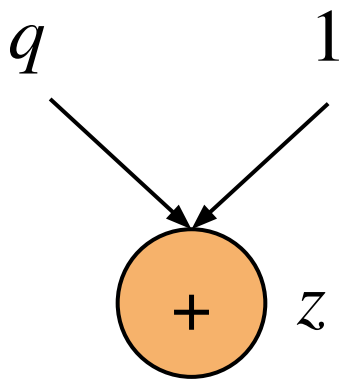


Let's take one piece of the computational graph at a time.

$$x + y = q$$

$$\frac{\partial q}{\partial x} = 1 \quad \frac{\partial q}{\partial y} = 1$$

Gradients on computational graphs



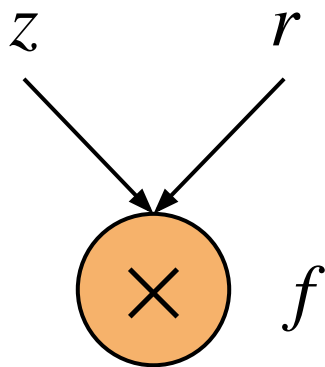
$$q + 1 = z$$

$$\frac{dz}{dq} = 1$$

**Chain
Rule**

$$\frac{\partial z}{\partial x} = \frac{dz}{dq} \frac{\partial q}{\partial x} = 1$$

Gradients on computational graphs



$$z \times r = f$$

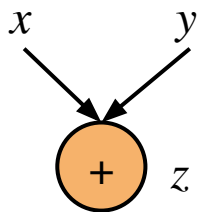
$$\frac{\partial f}{\partial z} = r \quad \frac{\partial f}{\partial r} = z$$

Chain Rule

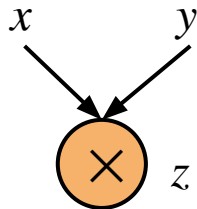
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial z} \frac{dz}{dq} \frac{\partial q}{\partial x} = r$$

In backprop, gradients are highly local

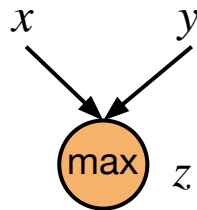
Three common operations in neural networks:



$$\frac{\partial f}{\partial x} = 1 \quad \frac{\partial f}{\partial y} = 1$$



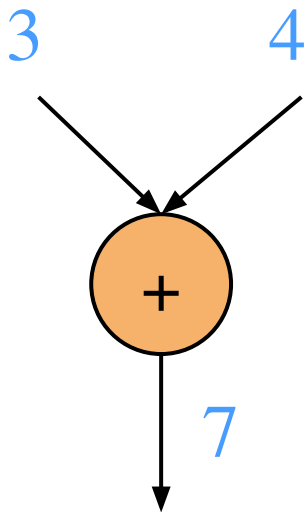
$$\frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x$$



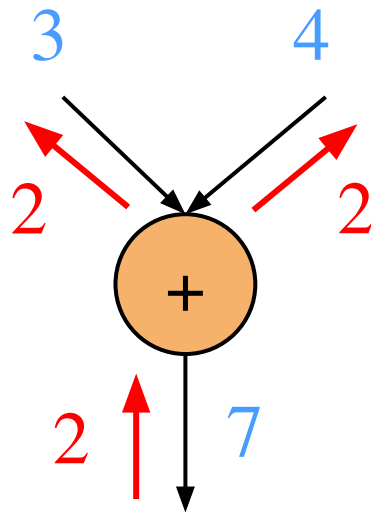
$$\frac{\partial f}{\partial x} = \mathbf{1}\{x \geq y\}$$
$$\frac{\partial f}{\partial y} = \mathbf{1}\{y \geq x\}$$

A small example with numbers

Forward pass

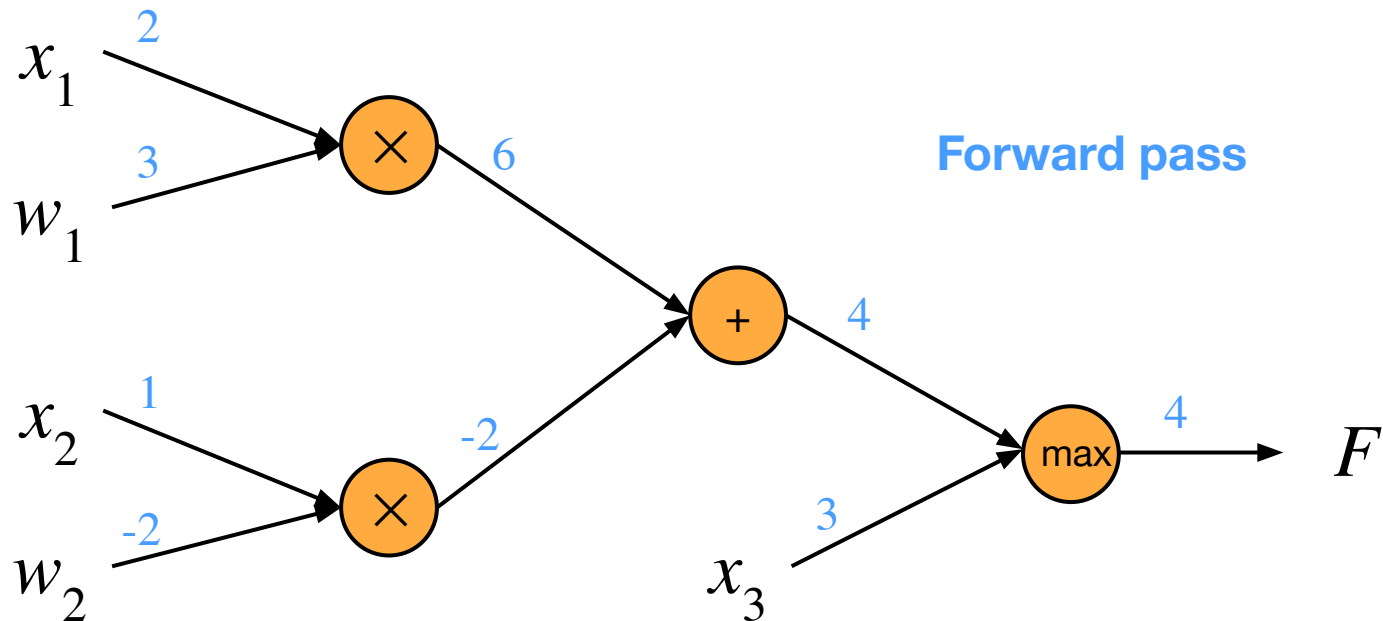


Backward pass



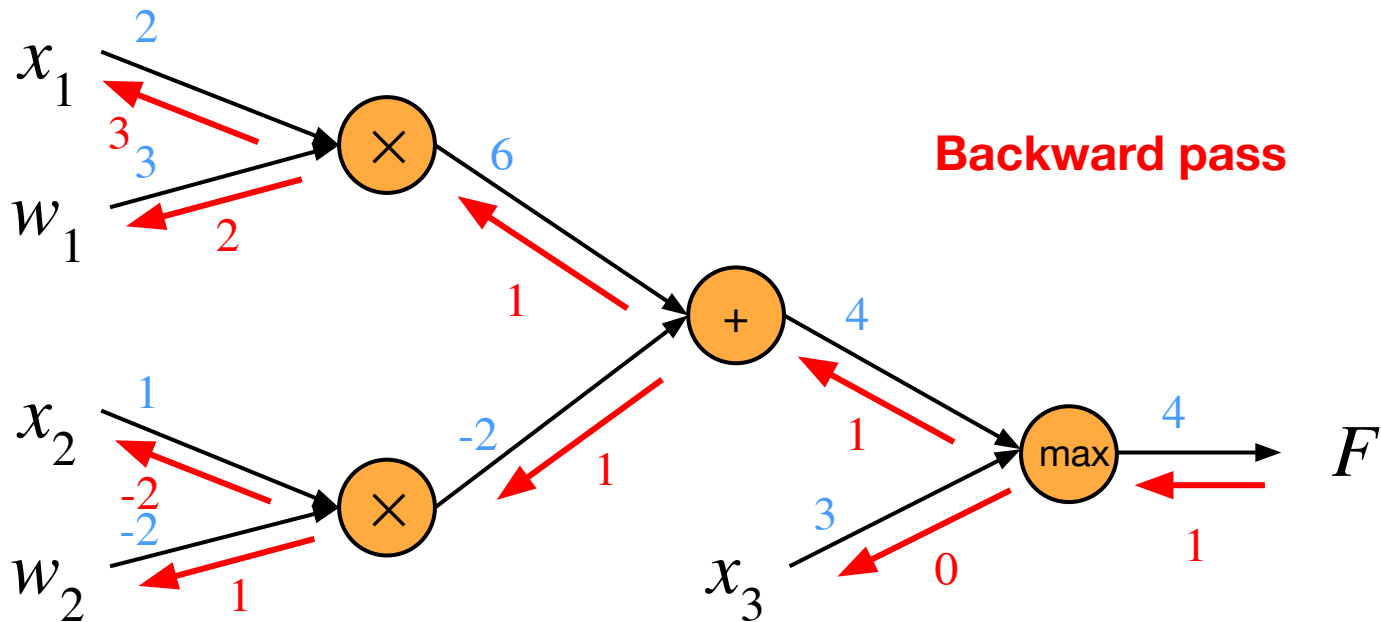
A larger example with numbers

$$F(x_1, x_2, x_3) = \max(w_1x_1 + w_2x_2, x_3)$$



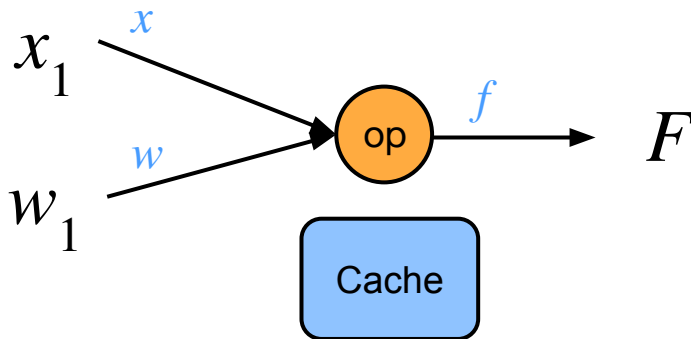
A larger example with numbers

$$F(x_1, x_2, x_3) = \max(w_1x_1 + w_2x_2, x_3)$$

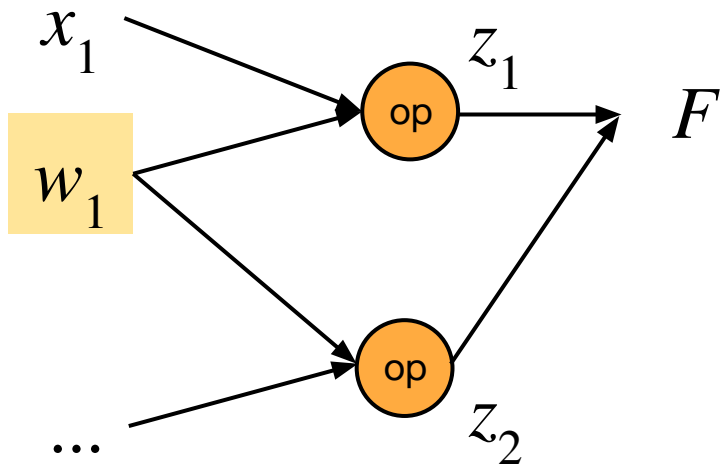


Note 1: Cache forward pass variables

During the backward pass, variable values from the forward pass are often needed



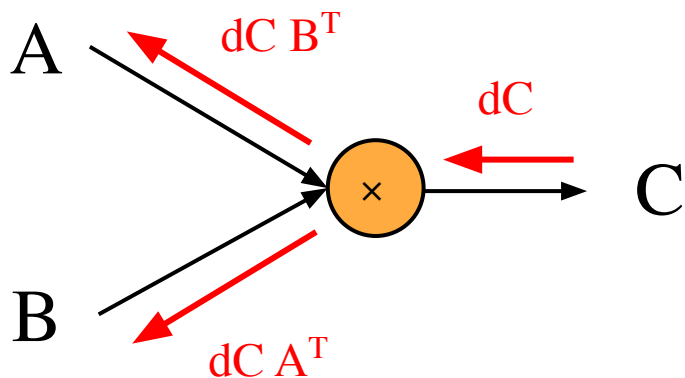
Note 2: Gradients add when a variable appears multiple times



$$\frac{\partial F}{\partial w_1} = \frac{\partial F}{\partial z_1} \frac{\partial z_1}{\partial w_1} + \frac{\partial F}{\partial z_2} \frac{\partial z_2}{\partial w_1}$$

Multivariable Chain Rule

Note 3: Backprop through matrix multiplication



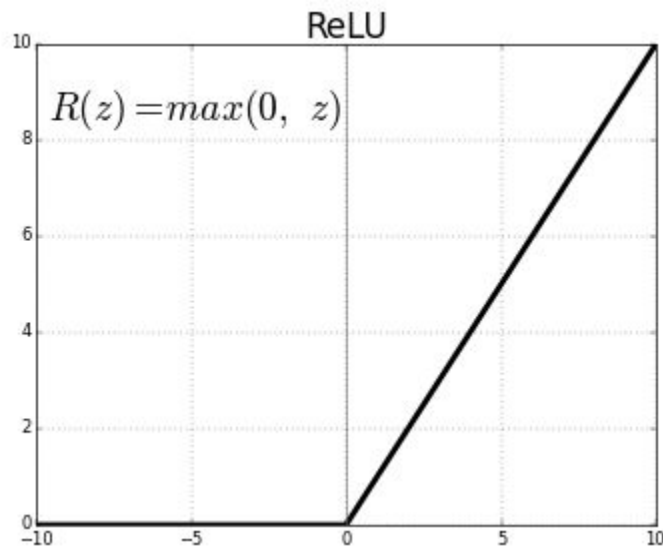
$$AB = C$$

$$dA = dC \times B^T$$

$$dB = dC \times A^T$$

Check matrix dimensions to verify

Exercise: What happens when the gradient passes through the ReLU function?



To answer, please type
into Zoom chat

Gradient descent

$$w' = w - \alpha g$$

The cost function is the average of the loss function over all training examples.

This means we have to forward and backward pass through all training examples before we take one step of gradient descent.

Stochastic gradient descent

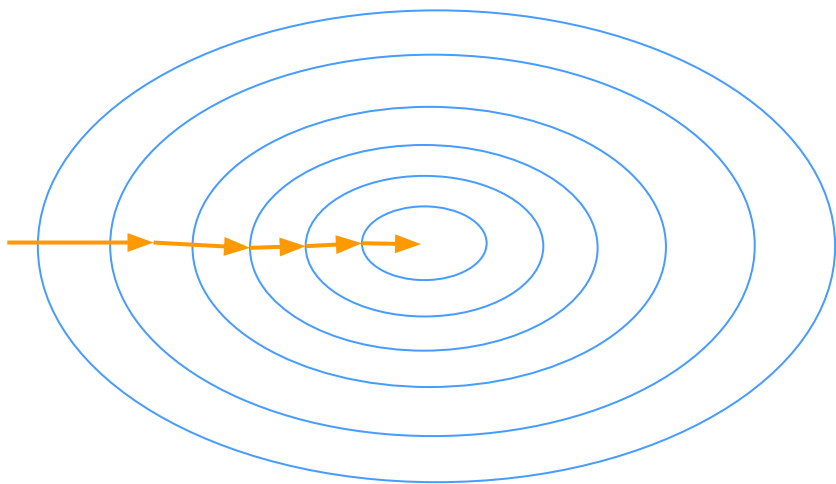
In SGD, we estimate the gradient by evaluating it over only a small subset of the training examples.

The size of this subsample is the **batch size**, which becomes another hyperparameter.

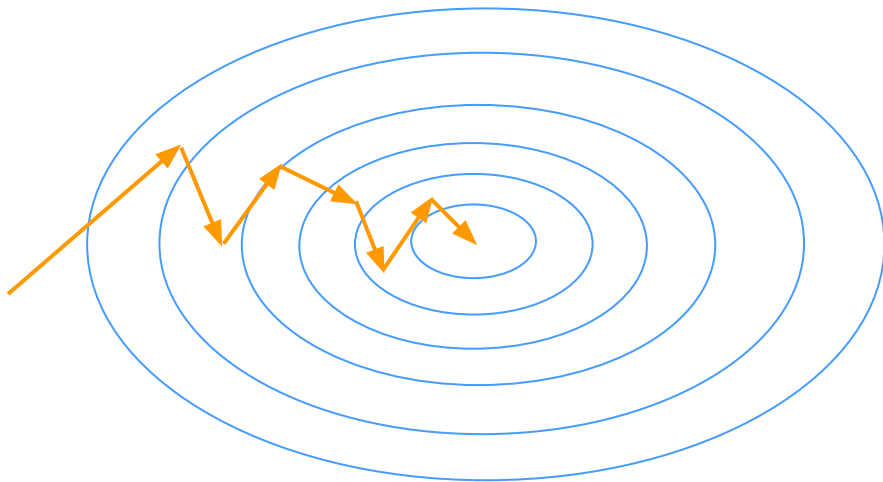
We use a different subsample for each gradient step, so as to go through all the training examples. Going through the entire training set once is called one **epoch** of training.

Stochastic gradient descent

Gradient descent



Stochastic gradient descent



Training, validation, and test sets

To evaluate the performance of the model, set aside a portion of the dataset.

Training set: used to train the weights of the neural network.

Validation set: used to tune the network hyperparameters and perform architecture search.

Test set: reserved until the end, used only once to evaluate the final weights and hyperparameters of the model.



Poll: Data splits

Go to PollEv.com/dlworkshop2020

Suppose I want to try training my neural network with different learning rates $\{0.1, 0.01, 0.001, 0.0001\}$. Which split should I use to evaluate which learning rate is best?

A. Train

B. Validation

C. Test

Training loop

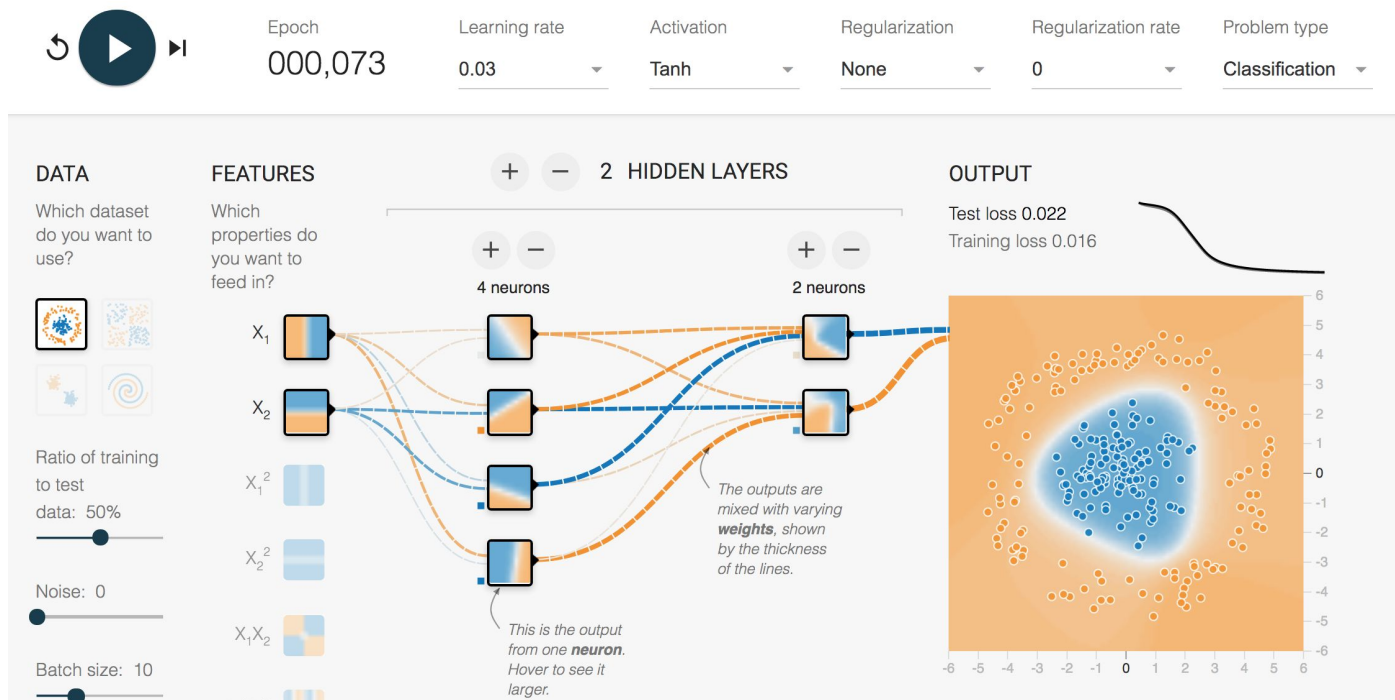
1. Sample a batch of data
2. Forward pass through all examples in the batch
3. Compute loss
4. Backpropagate through all examples
5. Compute final gradient on batch and update weights
6. Repeat

Workflow

1. Split your data into train/val/test
2. Choose architecture
3. Choose hyperparameters: learning rate, batch size, etc
4. Train neural network until convergence, evaluating on validation set each epoch
5. Check if training is overfitting or underfitting
6. Modify architecture and hyperparameters as needed, repeat

TensorFlow playground

<https://playground.tensorflow.org>

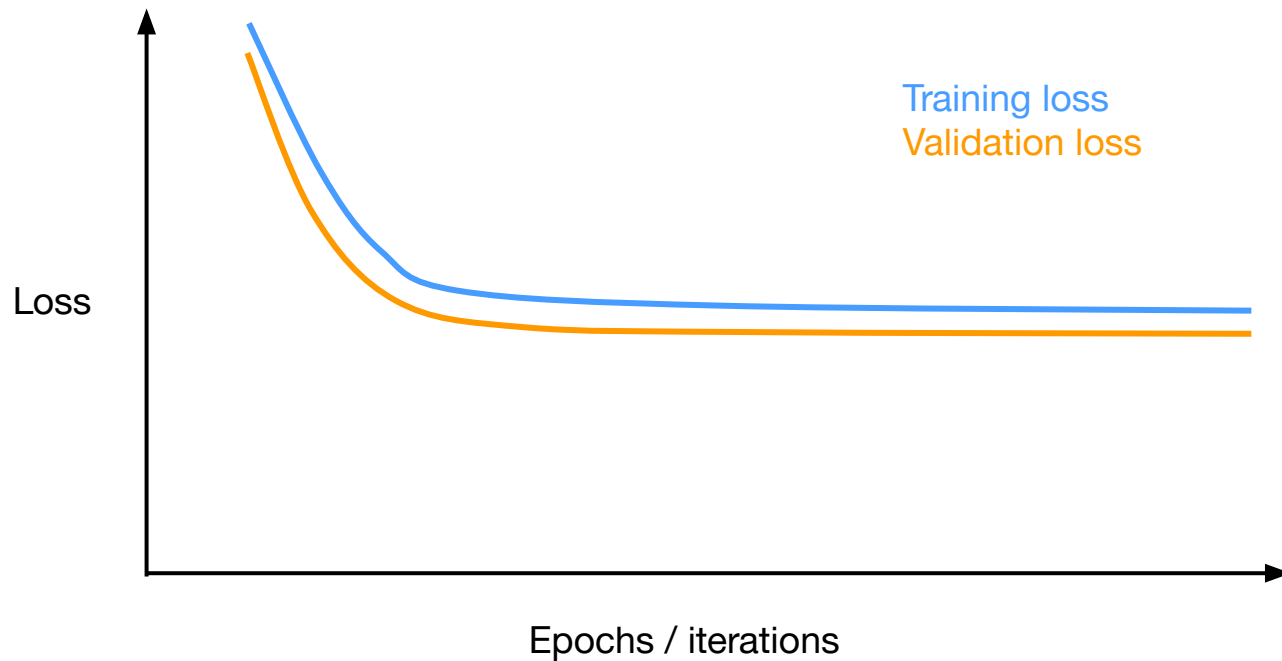


Overfitting and Underfitting

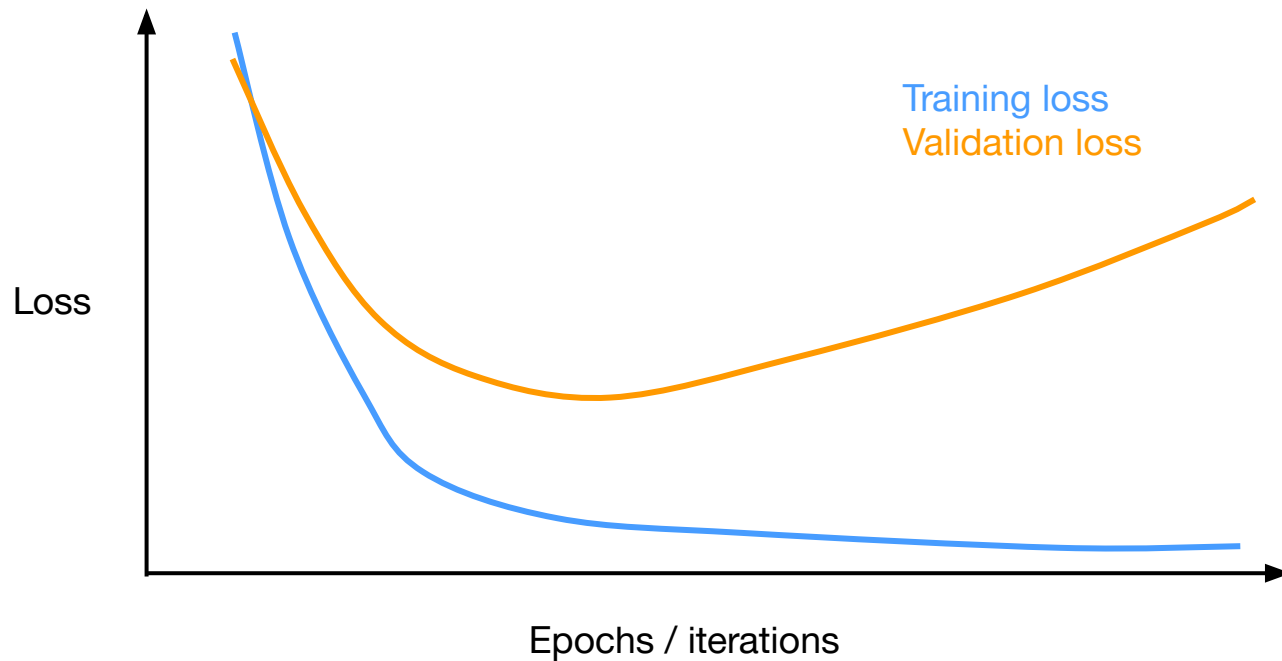
Overfitting and underfitting

- **Underfitting:** when your model is not complex enough to capture the structure in the data. The capacity of the model is insufficient. Both training loss and validation loss are high.
- **Overfitting:** when your model has so much capacity that it memorizes your training data. It usually indicates insufficient training data relative to your model complexity. Training loss is low, but validation loss is high.

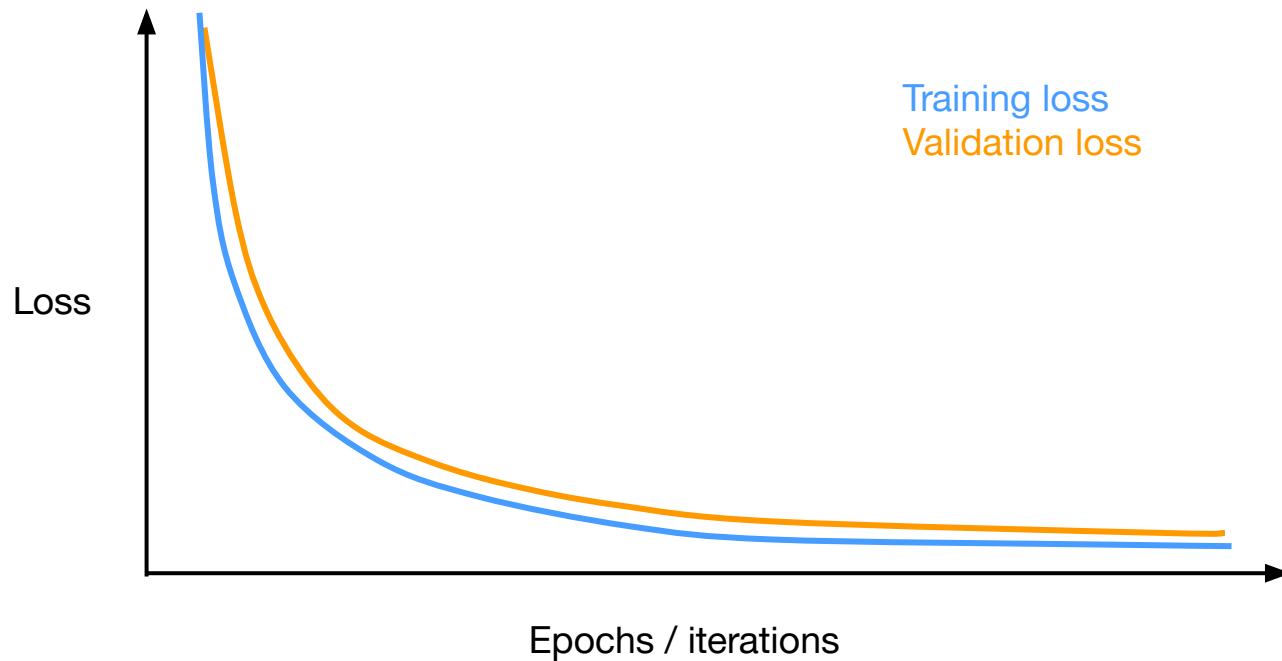
Underfitting



Overfitting

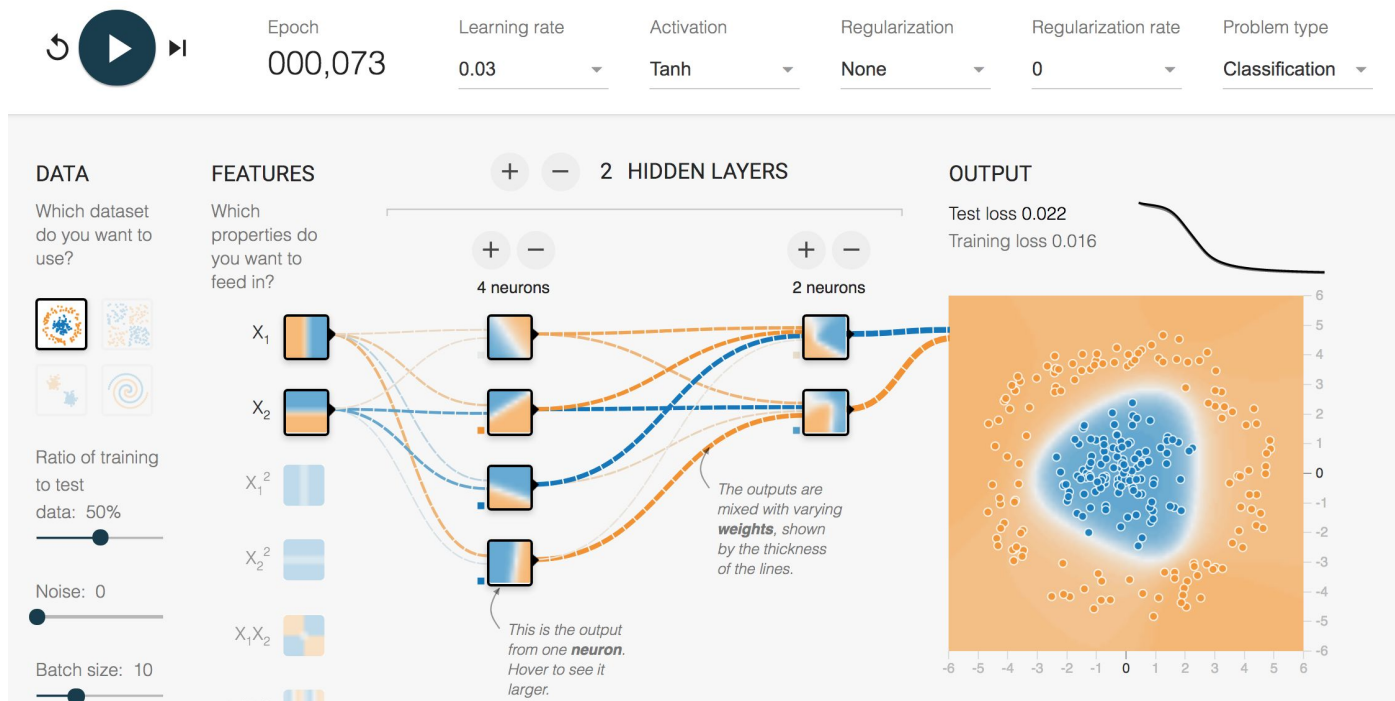


Good training



Exercise: Underfit a dataset in TF Playground

<https://playground.tensorflow.org>



Why do we need a test set?

Since we use the validation set to choose architecture and hyperparameters, it is possible to overfit to the validation set.


The test data set is used to give us an unbiased estimate of out-of-sample performance.

This underscores the importance of using the test set *only once*.

Regularization

One way to combat overfitting is to add a regularization term to the cost function.

This constrains the capacity of the model so that it has a lower chance of overfitting.

$$J(W, b) + \lambda R(W, b)$$

$$\|W\|^2$$

Final tips and tricks

1. Try other machine learning methods first
2. If you have trouble with training, first try to overfit a small subset of the dataset
3. If loss does not decrease smoothly, try to increase the number of hidden neurons in each layer
4. Play around with the learning rate; try to increase it as well as decrease it over time
5. Check online for others who have tried to solve your problem before attempting your own solution