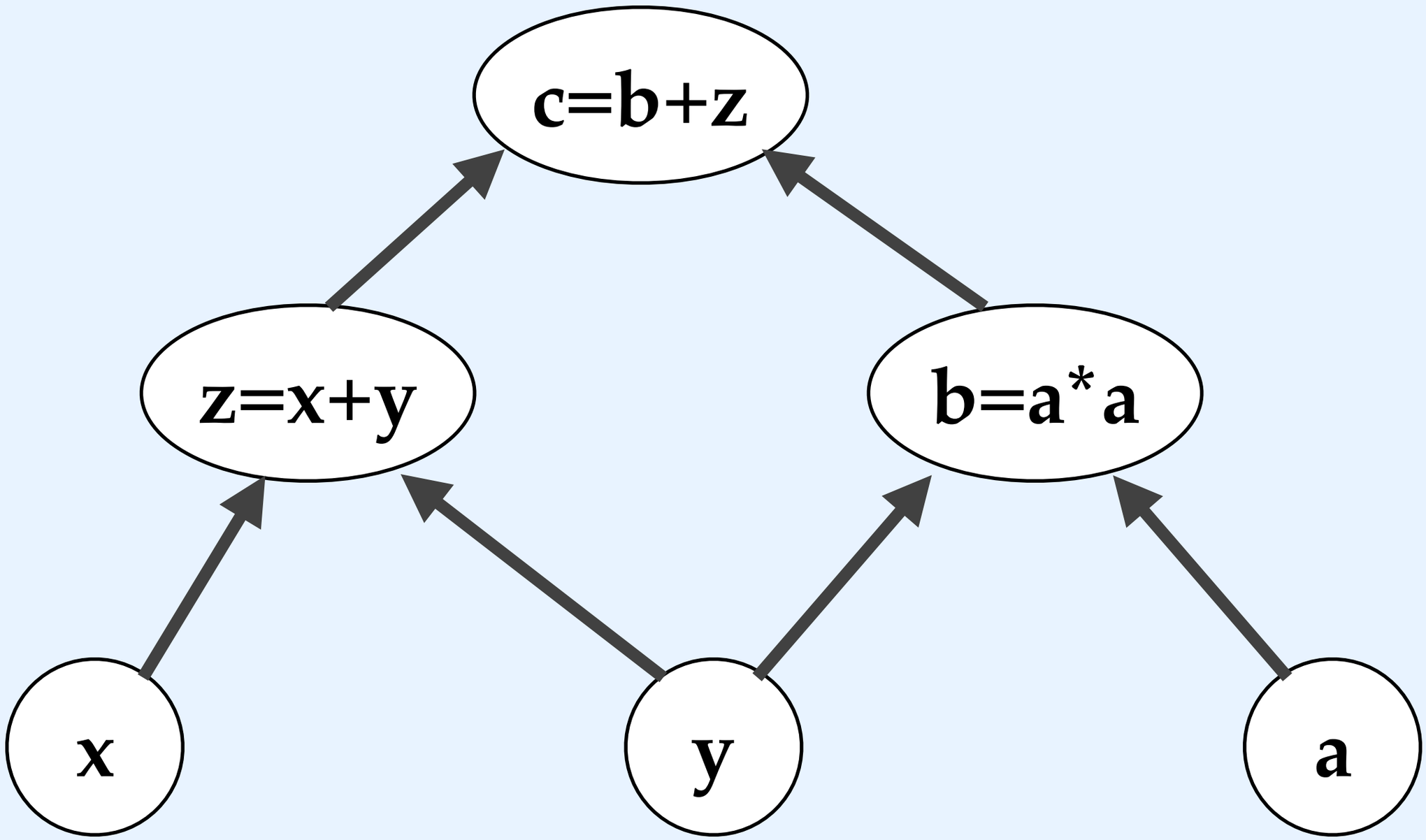
Computation Graph Languages

November 29, 2018

Computation Graph

A computation graph is a directed graph where nodes represent operations and variables and edges define the order of computation.

Computation Graph



Tensorflow

Tensorflow is a deep learning library for Python.

It is a computation graph language: A Tensorflow program defines a computation graph (and methods for running the graph).

Tensor

A tensor is a multidimensional array.

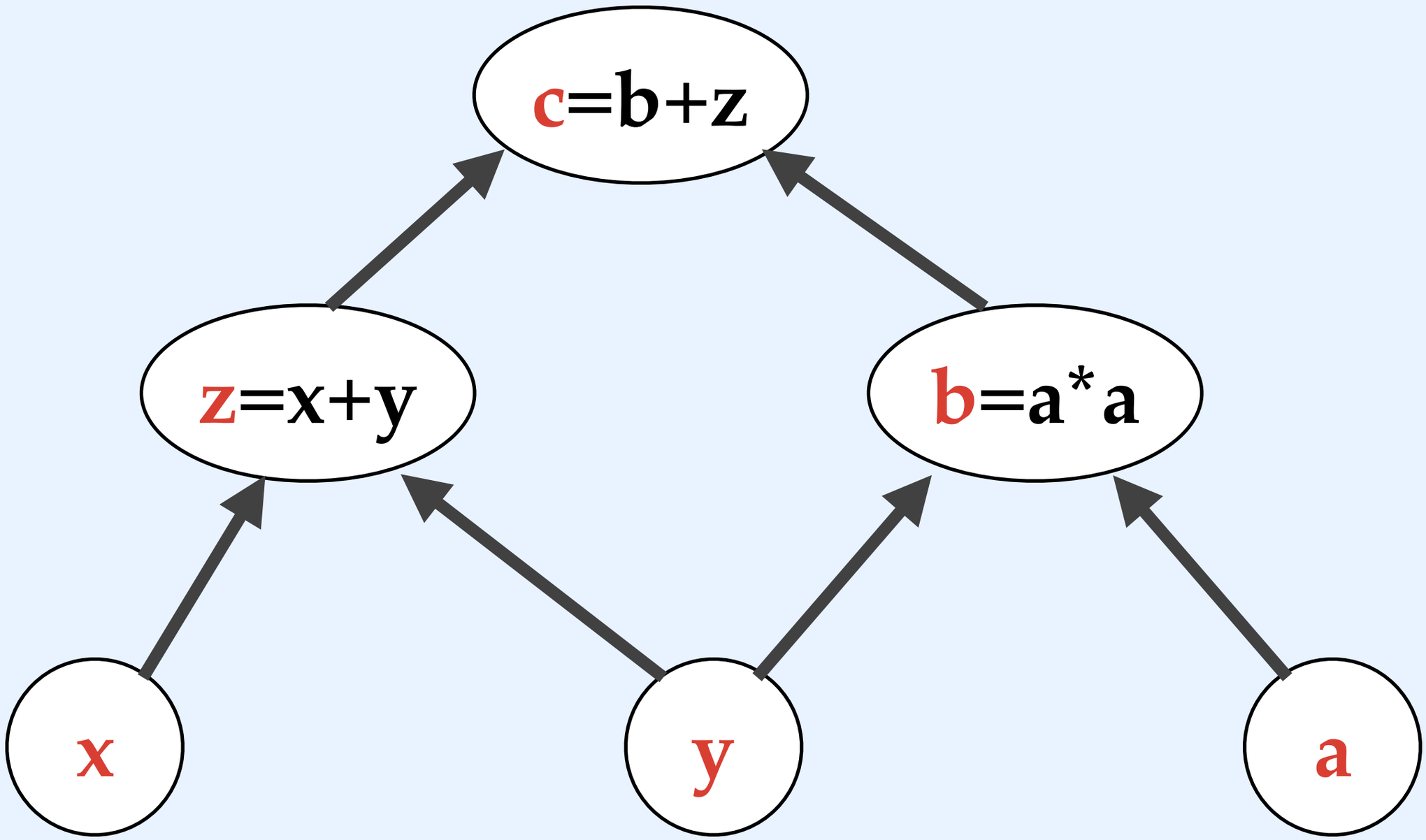
Vectors, scalars, and matrices are all tensors.

Computation Graph as Program

A Tensorflow program defines a computation graph (as well as methods for running it).

Some of the nodes in the graph are operators; others are place-holders.

Computation Graph as Program



Placeholders

In order to know the value of a placeholder, it is necessary to run the computation graph (or some part of it).

Placeholders

Computation graph languages are designed for probabilistic problems: they are not guaranteed to produce the same result every time they are run.

Because of this, the value of a placeholder is only known for a particular session (run of the computation graph).

Session

A session is a particular run through the computation graph. Any probabilistic inputs are initialized randomly each session.

Different sessions can produce different results.

Linear Regression

Imagine that we want to predict cat scores at the American Cat Fanciers Association annual cat show.

We have three kinds of information for each cat: age, length of whiskers, and glossiness.

Linear Regression

In linear regression, we try to model the output as a linear combination of the inputs (age, whisker length, and glossiness).

$$\text{score} = w_1 * \text{age} + w_2 * \text{whisker} + w_3 * \text{gloss}$$

Linear Regression

We have the scores and the input values. Our job is to find the combination of weights that best fits the data.

$$\text{score} = w_1 * \text{age} + w_2 * \text{whisker} + w_3 * \text{gloss}$$

Linear Regression

Once we learn weights w_1, w_2, and w_3, we can use them to predict unknown scores (given input values).

$$\text{score} = w_1 * \text{age} + w_2 * \text{whisker} + w_3 * \text{gloss}$$

Learning the weights

How do we learn the weights?

We initialize them randomly and then make small updates based on observations of our training data.

score = w_1*age + w_2*whisker + w_3*gloss

Learning the weights

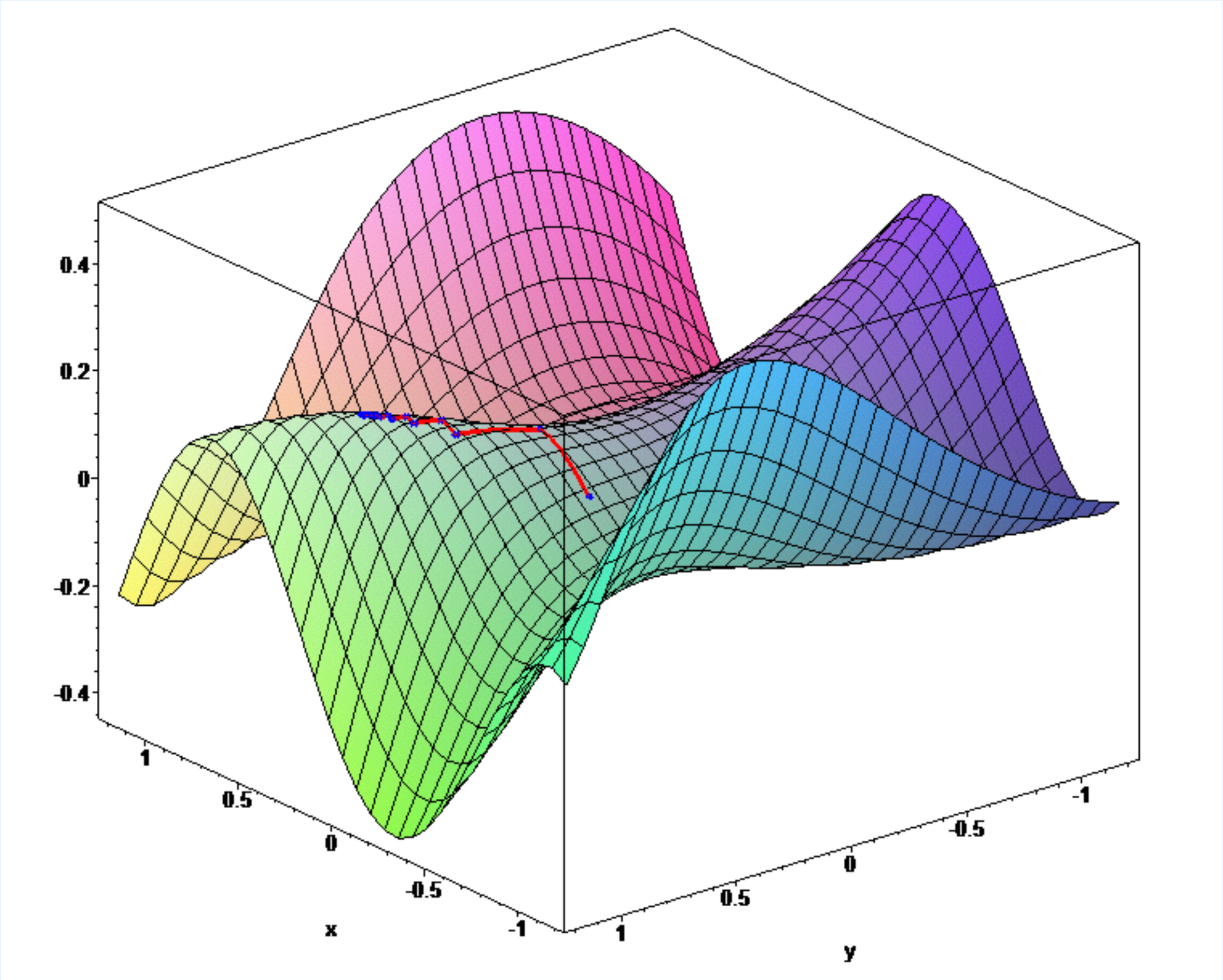
Based on our current guesses for the weights, we make a prediction about the score for a given set of inputs.

If this score differs from the actual score, we update the weights accordingly.

Gradient Descent

How do we know how to adjust the weights?

We use gradient descent to portion out the blame for the error.



Backpropagation

We run a computation graph forwards to make a prediction.

Then we calculate the difference between our prediction and the actual output.

The error (the difference between the predicted and observed output) is backpropagated through the graph.

Computation graph training

In general, training computation graphs involves the following steps:

1. Forward propagation of values to make a prediction.

2. Error calculation via the loss function.

3. Backward propagation of the error to update the weights.

Up next: more computation graphs!

Next week we'll look at an example of another kind of programming languages that define computation graphs: probabilistic programming languages.