The Mathematical Building Blocks of Neural Networks

By

Professor M. Shahidur Rahman

DoCSE, SUST

Data Representations for Neural Networks: Tensor

- A tensor is a container for data—usually numerical data
- Tensors are fundamental to the field—so fundamental that TensorFlow was named after them

Tensors

- Scalars (rank-0 tensors)
  - A tensor that contains only one number is called a scalar (or scalar tensor, or rank-0 tensor, or 0D tensor).
  - Ex. A float32 or float64 number

```
import numpy as np
>>> x = np.array(12)
>>> x
array(12)
>>> x.ndim
0
```

Tensors…

- An array of numbers is called a vector, or rank-1 tensor, or 1D tensor. A rank-1 tensor has exactly one axis.

```
>>> x = np.array([12, 3, 6, 14, 7])
>>> x
array([12, 3, 6, 14, 7])
>>> x.ndim
 1
```

- An array of vectors is a matrix, or rank-2 tensor, or 2D tensor. A matrix has two axes.

```
>>> x = np.array([[5, 78, 2, 34, 0],
                  [6, 79, 3, 35, 1],
                  [7, 80, 4, 36, 2]])
>>> x.ndim
2
```

- Rank-3 and higher-rank tensors are also frequently used

Rank-3 and higher-rank Tensors

- If we pack such matrices in a new array, we obtain a rank-3 tensor (or 3D tensor), which can be visually interpreted as a cube of numbers.

```
>>> x = np.array([[[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]]])
>>> x.ndim
3
```

- By packing rank-3 tensors in an array, we can create a rank-4 tensor, and so on. In deep learning, we'll generally manipulate tensors with ranks 0 to 4, although we may go up to 5 if we process video data.

Tensors: Key Attributes

A tensor is defined by three key attributes:

- Number of axes (rank)
- Shape - how many dimensions the tensor has along each axis
  - The previous matrix example has shape (3, 5),
  - A vector has a shape with a single element, such as (5,),
  - A scalar has an empty shape, ()
- Data type (usually called dtype in Python libraries)—
  - Type of the data contained in the tensor
  - For instance, a tensor's type could be float16, float32, float64, uint8, and so on.

Manipulating Tensors in NumPy

- Selecting specific elements in a tensor is called tensor slicing.

```
>>> my_slice = train_images[10:100]
>>> my_slice.shape
(90, 28, 28)
```

or [more detailed],

```
>>> my_slice = train_images[10:100, :, :]
>>> my_slice.shape
(90, 28, 28)
```

or,

```
>>> my_slice = train_images[10:100, 0:28, 0:28]
>>> my_slice.shape
(90, 28, 28)
```

- Selects bottom-right corner of all images

```
my_slice = train_images[:, 14:, 14:]
```

- Selects patches of 14 × 14 pixels centered in the middle (uses negative index)

```
my_slice = train_images[:, 7:-7, 7:-7]
```

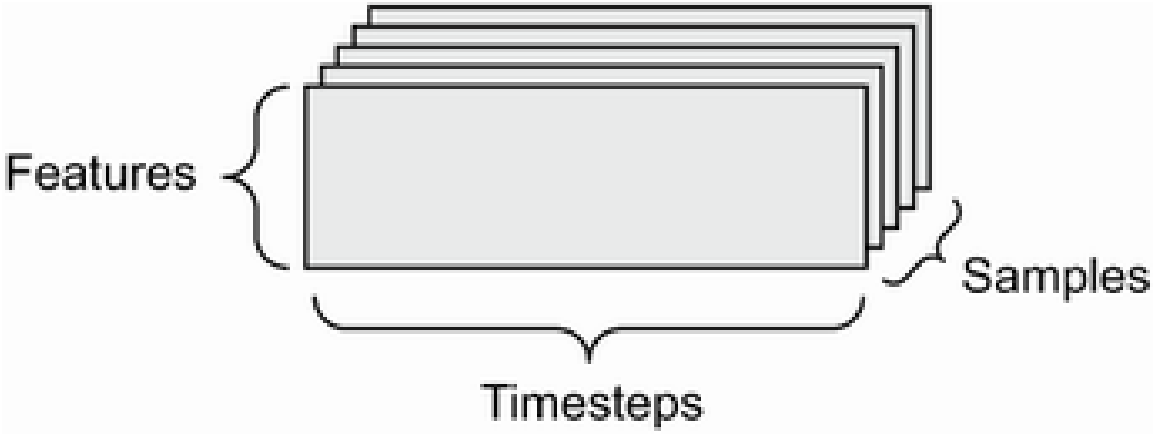
Notion of Data Batches

- In general, the first axis (axis 0) in all data tensors in deep learning will be the samples axis (or samples dimension).
- Deep learning models don't process an entire dataset at once; rather, they break the data into small batches.
  - For example, one batch of MNIST digits, with a batch size of 128: **batch = train_images[:128]**
  - And here's the next batch: **batch = train_images[128:256]**
  - And the nth batch: **batch = train_images[128 * n:128 * (n + 1)]**
- When considering such a batch tensor, the first axis (axis 0) is called the batch axis or batch dimension.

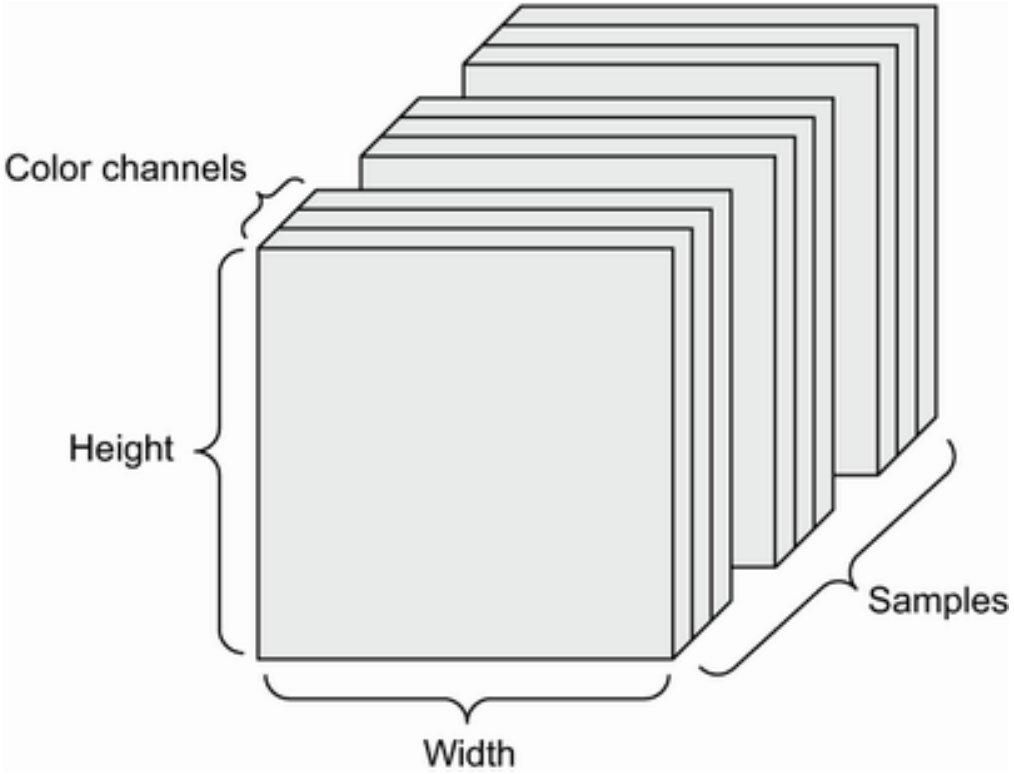
Real-world Examples of Data Tensors

- Vector data
  - Rank-2 tensors of shape (samples, features)
  - each sample is a vector of numerical attributes (“features”)
- Timeseries data or sequence data
  - Rank-3 tensors of shape (samples, timesteps, features)
  - each sample is a sequence (of length timesteps) of feature vectors
- Images
  - Rank-4 tensors of shape (samples, height, width, channels)
  - each sample is a 2D grid of pixels, and each pixel is a vector of values (“channels”)
- Video
  - Rank-5 tensors of shape (samples, frames, height, width, channels)
  - each sample is a sequence (of length frames) of images

Visualization of the Tensors



A rank-3 timeseries data tensor
[shape (250, 390, 3)]



A rank-4 image data tensor
[shape (128, 256, 256, 3)]

Gears of Neural Networks: Tensor operations

- ***Element-wise operations***

```
import numpy as np
z = x + y
z = np.maximum(z, 0.)
```

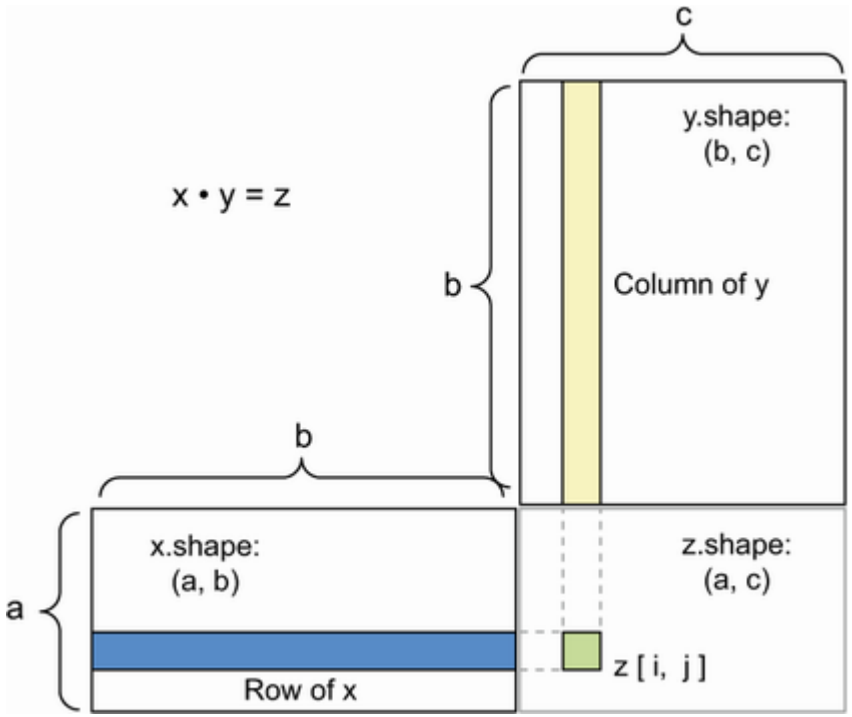
Element-wise addition

Element-wise relu

Tensor Product

- The *tensor product*, or *dot product* is one of the most useful tensor operations.

```
x = np.random.random((32,))
y = np.random.random((32,))
z = np.dot(x, y)
```



Matrix dot-product box diagram

Broadcasting

- What happens with addition when the shapes of the two tensors being operated differ?
  - When possible, and if there's no ambiguity, the smaller tensor will be *broadcast* to match the shape of the larger tensor.
- Broadcasting consists of two steps:
  - Axes (called *broadcast axes*) are added to the smaller tensor to match the *ndim* of the larger tensor.
  - The smaller tensor is repeated alongside these new axes to match the full shape of the larger tensor.

Broadcasting: an Example

- Consider X with shape (32, 10) and y with shape (10,):

```
import numpy as np
X = np.random.random((32, 10))          # shape(32, 10)
y = np.random.random((10,))             # shape(10,)
```

- First, we add an empty first axis to y, whose shape becomes (1, 10):

```
y = np.expand_dims(y, axis=0)           # shape(1, 10)
Y = np.concatenate([y] * 32, axis=0)    # shape(32, 10)
```

Tensor Reshaping

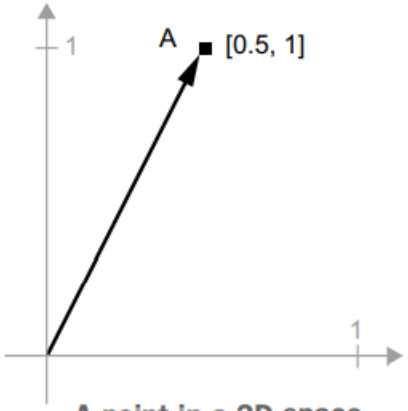
- Reshaping a tensor means rearranging its rows and columns to match a target shape.

```
>>> x = np.array([[0., 1.],
                  [2., 3.],
                  [4., 5.]])
>>> x.shape
(3, 2)
>>> x = x.reshape((6, 1))
>>> x
array([[ 0.],
       [ 1.],
       [ 2.],
       [ 3.],
       [ 4.],
       [ 5.]])
```

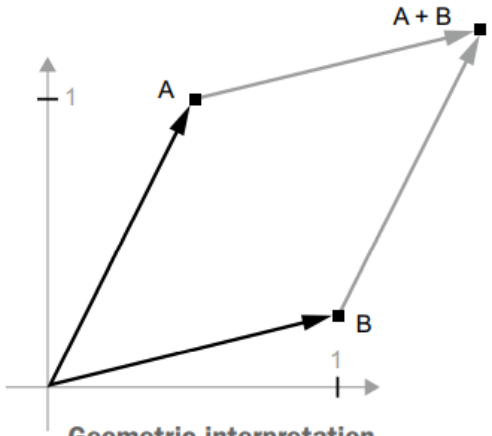
```
>>> x = x.reshape((2, 3))
>>> x
array([[ 0.,  1.,  2.],
       [ 3.,  4.,  5.]])
```

Geometric Interpretation of Tensor Operations

- Tensors manipulated by tensor operations can be interpreted as coordinates of points in some geometric space
- Tensor addition, for example, represents the action of translating an object (moving the object without distorting it) by a certain amount in a certain direction.
- Elementary geometric operations such as translation, rotation, scaling, skewing, and so on can be expressed as tensor operations.



A point in a 2D space pictured as an arrow



Geometric interpretation of the sum of two vectors

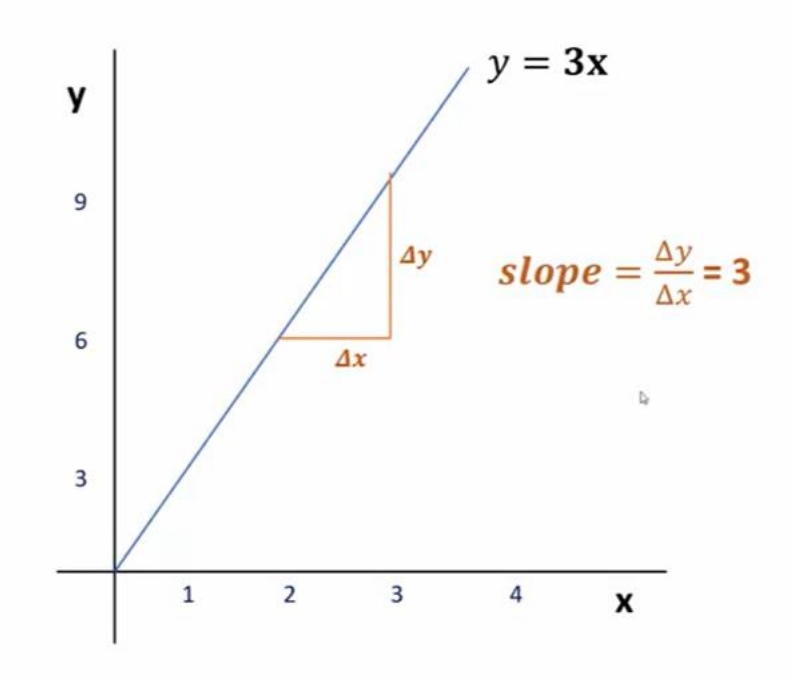
Geometric Interpretation of Deep Learning

- What a neural network is meant to do is figure out a transformation of the paper ball that would uncrumple it, so as to make the two classes cleanly separable.
- Implemented as a series of simple transformations of the 3D space, such as those you could apply on the paper ball with your fingers, one movement at a time.
- Each layer in a deep network applies a transformation that disentangles the data a little, and a deep stack of layers makes tractable an extremely complicated disentanglement process.

Uncrumpling a complicated manifold of data

The Engine of Neural Networks: Gradient-based Optimization

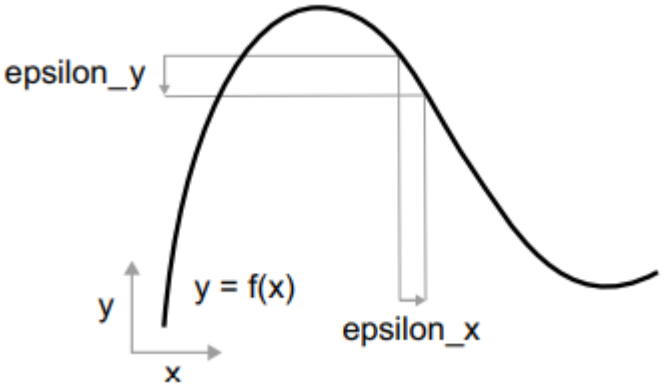
- What's a slope?
  - when f(x) is linear
  - It's constant

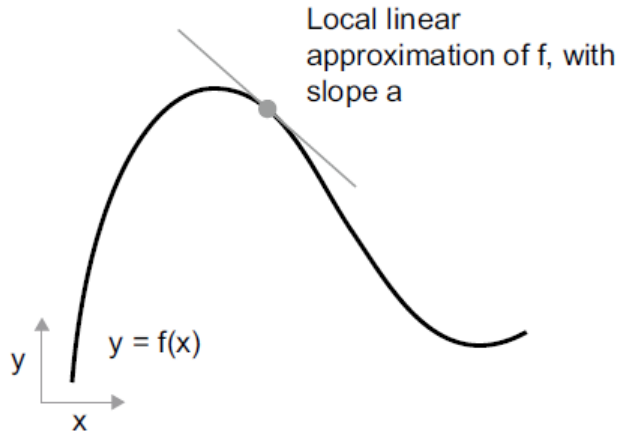


Gradient-based Optimization…

- What's a derivative?
  - With a smooth continuous function, a small change in **x** results in a small change in **y**.
  - Around a point p, f can be approximated as a linear function of slope a, so that epsilon_y becomes a * epsilon_x:

    f(x + epsilon_x) = y + a * epsilon_x
  - The slope **a** is called the ***derivative*** of **f** in p.
  - If a is negative, a small increase in x around p will result in a decrease of f(x)
  - If a is positive, a small increase in x will result in an increase of f(x).
  - The absolute value of a tells how quickly this increase or decrease will happen.





Gradient-based Optimization…

- What is a gradient?
  - The (previous) function turned a scalar value x into another scalar value y, that can be plotted as a curve in a 2D plane
  - A function can turn a tuple of scalars (x, y) into a scalar value z, that can be plotted as a 2D surface in a 3D space (indexed by coordinates x, y, z)
  - Likewise, functions can take matrices, rank-3 tensors as inputs…
  - The derivative of a tensor operation (or tensor function) is called a *gradient*.
  - Like slope in the previous case, the gradient of a tensor function represents the curvature of the multidimensional surface described by the function

Gradient-based Optimization: An Example

- An input vector, *x* (a sample in a dataset)
- A matrix, *W* (the weights of a model)
- A target, *y_true* (what the model should learn to associate to *x*)
- A loss function, *loss* (the gap between the model's current predictions and y_true)

```
y_pred = dot(W, x)
loss_value = loss(y_pred, y_true)
```

We use the model weights, W, to make a prediction for x.

We estimate how far off the prediction was.

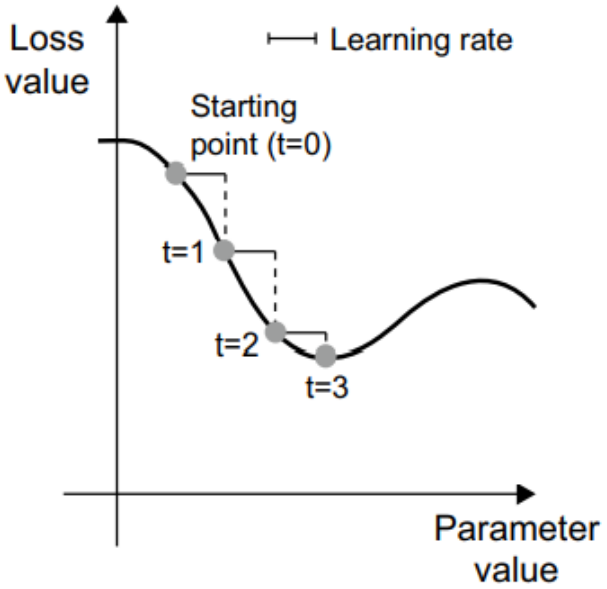
```
loss_value = f(W)
```

f describes the curve (or high-dimensional surface) formed by loss values when W varies.

Gradient-based Optimization: An Example…

- What does grad(loss_value, W0) represent?
  - The derivative of a function f(x) of a single coefficient can be interpreted as the slope of the curve of f.
    - f(x) is reduced by moving x a little in the opposite direction from the derivative
  - Likewise, grad(loss_value, W0) can be interpreted as the tensor describing the direction of steepest ascent of loss_value = f(W) around W0.
    - With a function f(W) of a tensor, we can reduce loss_value = f(W) by moving W in the opposite direction from the gradient:

      W1 = W0 - step * grad(f(W0), W0) (here, step (learning rate) is a small scaling factor).

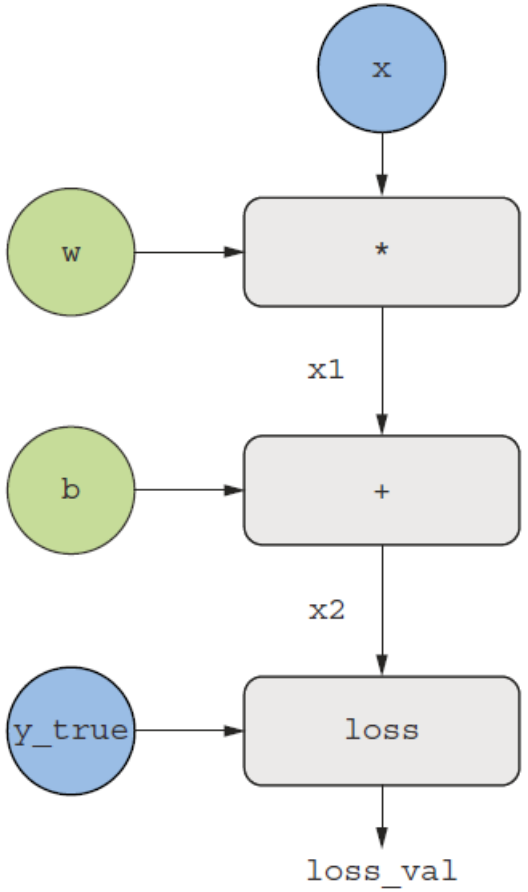


Chaining derivatives: Backpropagation Algorithm

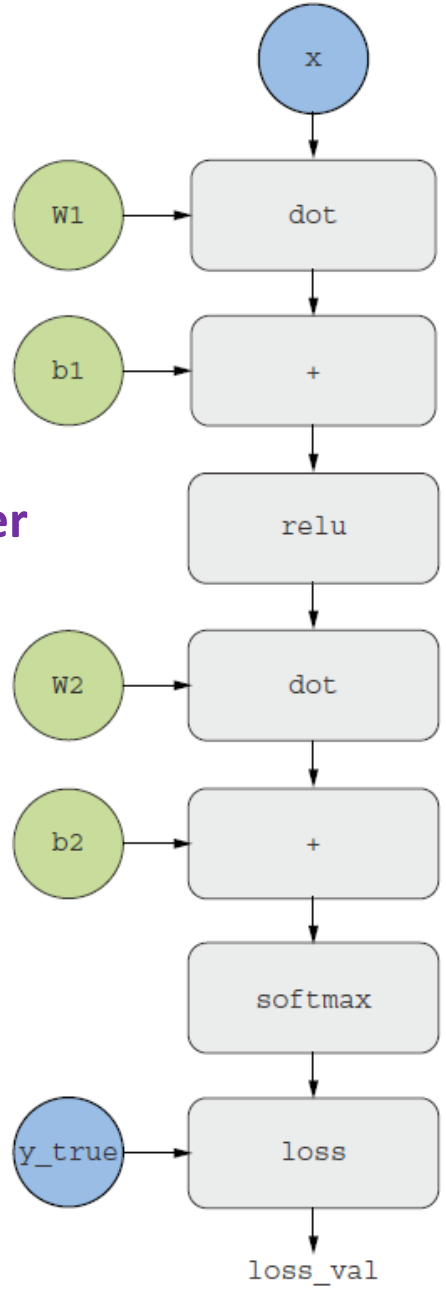
- Backpropagation is a way to use the derivatives of simple operations (such as addition, relu, or tensor product) to easily compute the gradient of arbitrarily complex combinations of these atomic operations.
- The **chain rule** states that grad(y, x) == grad(y, x1) * grad(x1, x).
- Applying the chain rule to the computation of the gradient values of a neural network gives rise to an algorithm called ***backpropagation***.
- A useful way to think about backpropagation is in terms of ***computation graphs***.
- *Computation graph* is a directed acyclic graph of operations—in our case, tensor operations.

Computation Graphs

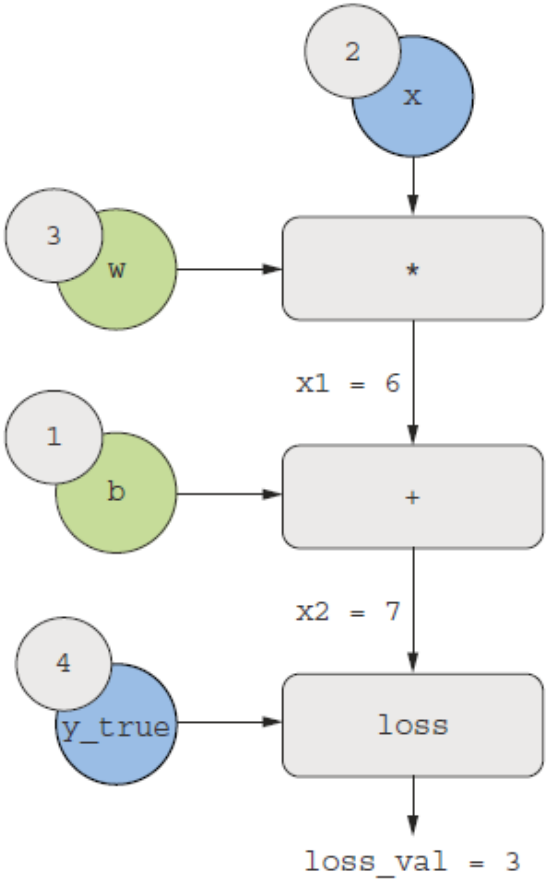
Basic Computation Graph



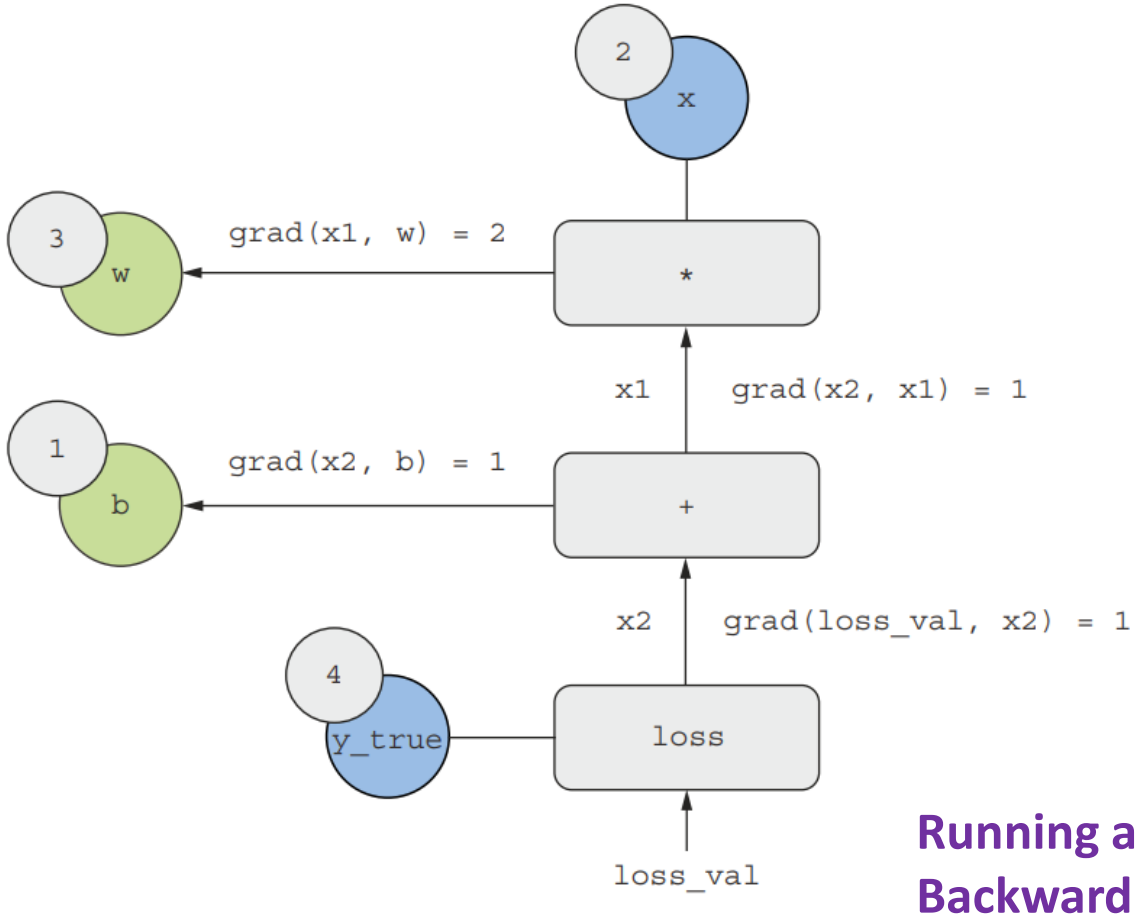
Computation Graph of 2-layer Model



Computation Graphs of Forward & Backward Pass

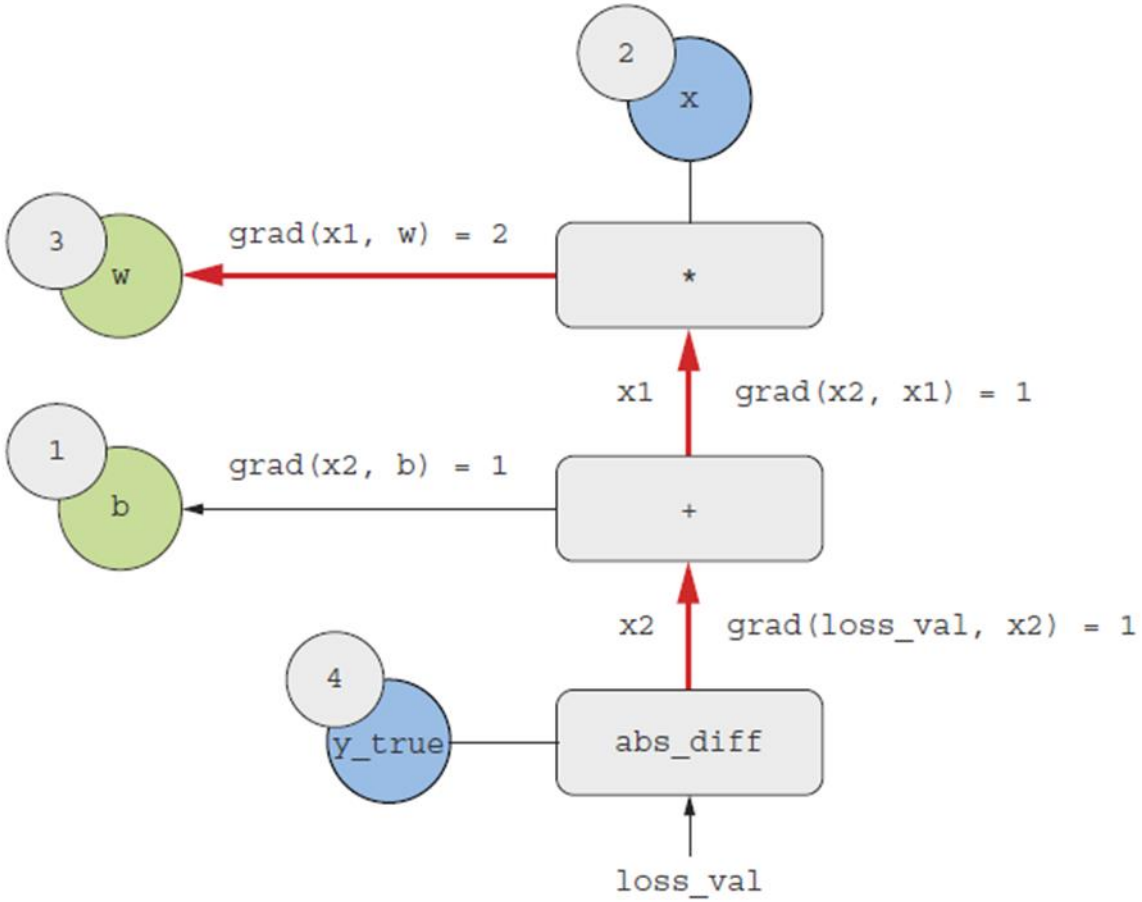


Running a Forward Pass



Running a Backward Pass

Path from loss_val to w in the backward graph



What is Obtained with Computation Graph?

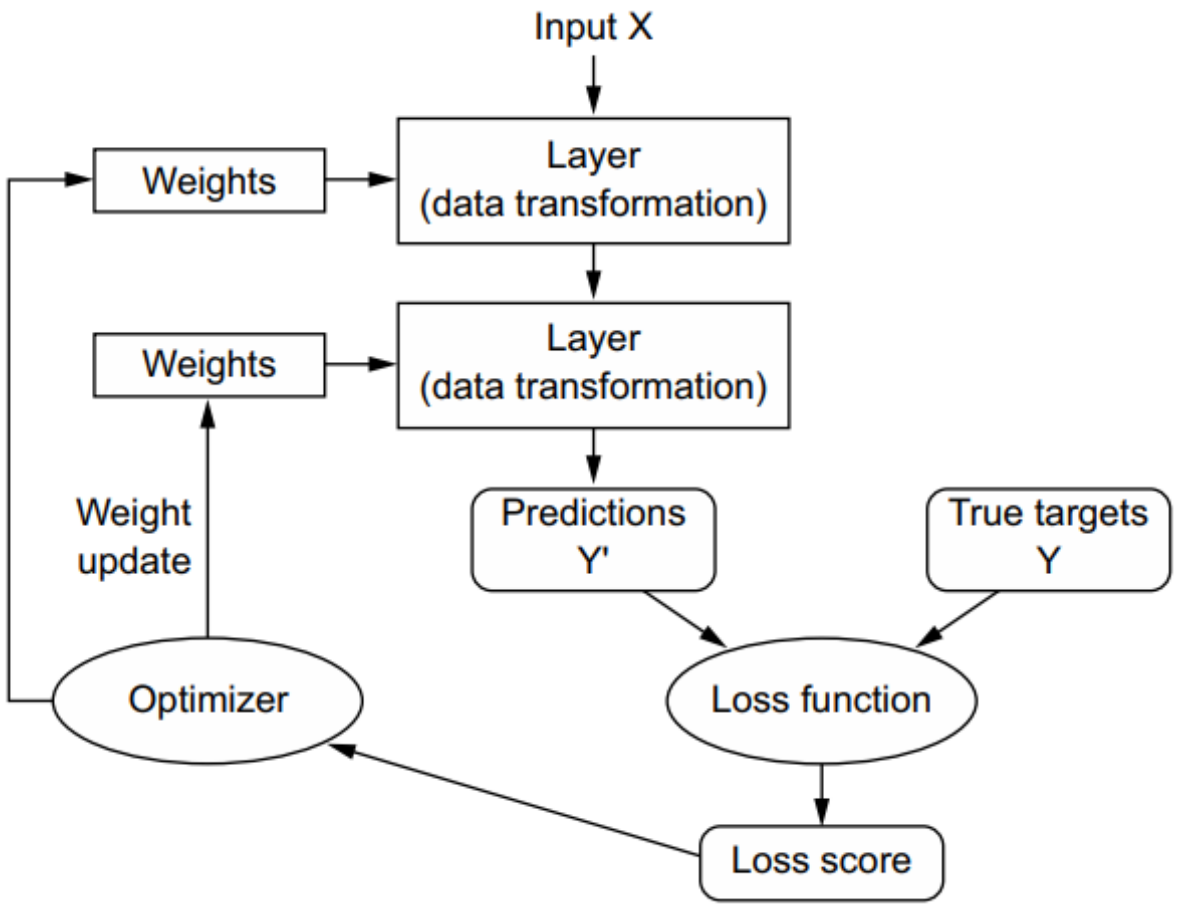
We have the following:

- grad(loss_val, x2) = 1, because as x2 varies by an amount epsilon, loss_val = abs(4 - x2) varies by the same amount.
- grad(x2, x1) = 1, because as x1 varies by an amount epsilon, x2 = x1 + b = x1 + 1 varies by the same amount.
- grad(x2, b) = 1, because as b varies by an amount epsilon, x2 = x1 + b = 6 + b varies by the same amount.
- grad(x1, w) = 2, because as w varies by an amount epsilon, x1 = x * w = 2 * w varies by 2 * epsilon.

Automatic Differentiation with Computation Graphs

- What the chain rule says about this backward graph is that you can obtain the derivative of a node with respect to another node by *multiplying the derivatives for each edge along the path linking the two nodes*.
  - For instance, grad(loss_val, w) = grad(loss_val, x2) * grad(x2, x1) * grad(x1, w)
- By applying the chain rule to our graph, we obtain what we were looking for:
  - grad(loss_val, w) = 1 * 1 * 2 = 2
  - grad(loss_val, b) = 1 * 1 = 1
- Modern frameworks, such as TensorFlow, are capable of *automatic differentiation*, which is implemented with the kind of computation graphs.

The network, layers, loss function, and optimizer



Summary

- Tensors form the foundation of modern machine learning systems. They come in various flavors of **dtype**, **rank**, and **shape**.
- You can manipulate numerical tensors via tensor operations (such as addition, tensor product, or element-wise multiplication), which can be interpreted as encoding geometric transformations (in general).
- Deep learning models consist of chains of simple tensor operations, parameterized by weights, which are themselves tensors. **The weights of a model are where its "knowledge" is stored.**
- Learning means finding a set of values for the model's weights that minimizes a loss function for a given set of training data samples and their corresponding targets.

Summary…

- Learning happens by drawing random batches of data samples and their targets, and computing the gradient of the model parameters with respect to the loss on the batch.
- The entire learning process is made possible by the fact that all tensor operations in neural networks are differentiable, and thus it's possible to apply the chain rule of derivation to find the gradient function mapping the current parameters and current batch of data to a gradient value. This is called **backpropagation**.
- The two things (loss and optimizers) you need to define before you begin feeding data into a model.
  - The loss is the quantity you'll attempt to minimize during training, so it should represent a measure of success for the task you're trying to solve.
  - The optimizer specifies the exact way in which the gradient of the loss will be used to update parameters: for instance, it could be the RMSProp optimizer, SGD with momentum, and so on.