Training Deep Neural Nets

Loss Function, Chain Rule, and Backpropagation

N. Rich Nguyen, PhD

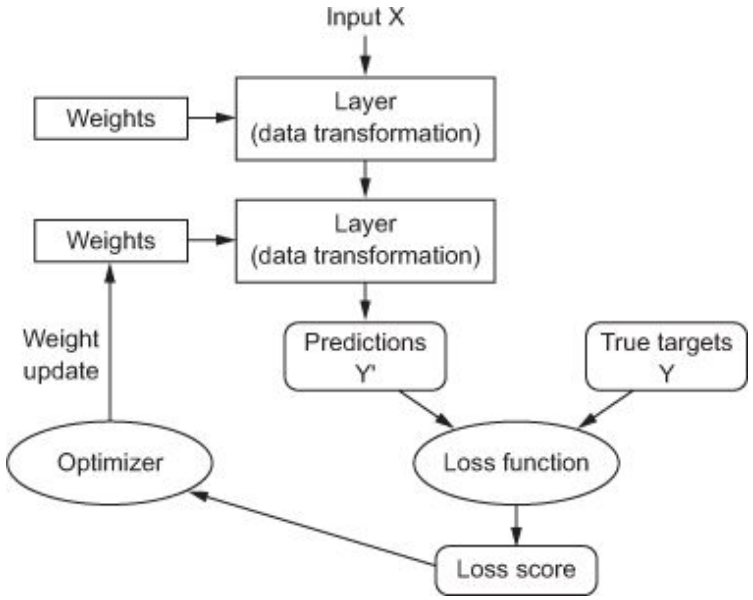
SYS 6016

1. Training Deep Neural Nets

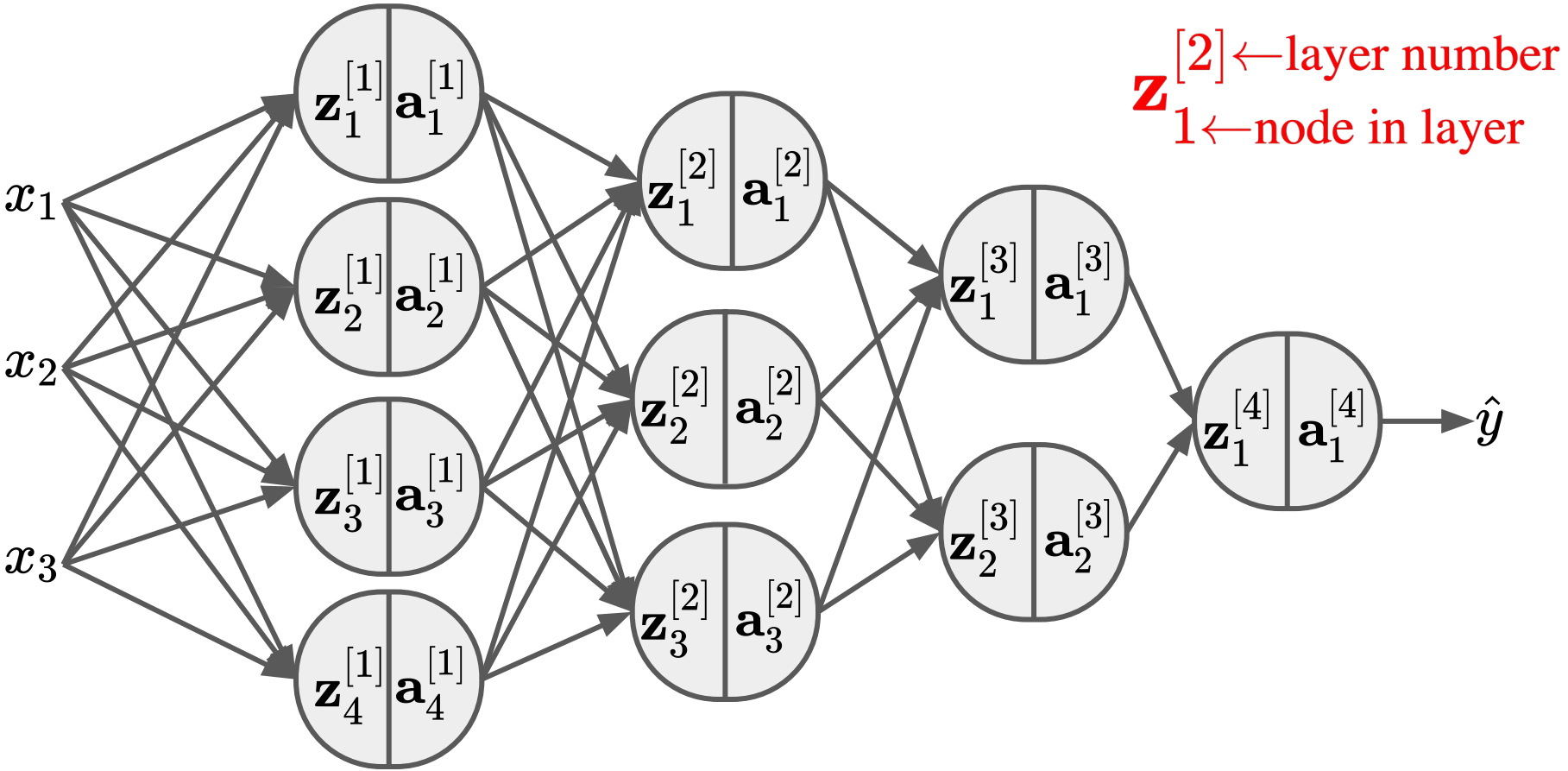
Training a neural network

Training a neural net revolves around the following 4 objects:

1. **Input data**, which includes features and corresponding labels
2. **Layers**, which are combined into a network
3. **Loss function**, which defines the feedback signal used for training
4. **Optimizer**, which determines how the learning proceeds



Layers: from the previous video



Loss function *aka. cost function, objective function, error function*

It is the quantity that measures how well a specific algorithm models the data and will be ***minimized*** during training (square loss, log loss, hinge loss, ect.)

For regression task, we can use **MSE (square loss)**: $\mathcal{L}_{\text{mse}} = \frac{1}{2}\sum_i (y_i - \hat{y}_i)^2$

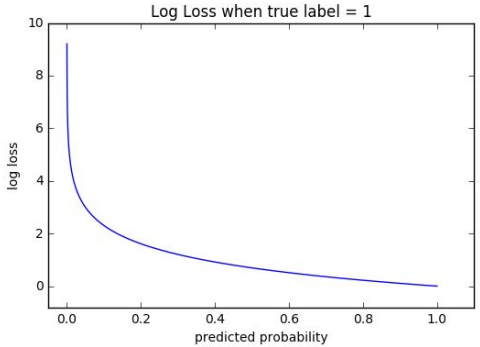
For classification task, we can use **Cross-Entropy (log loss)** to evaluate the network output as a probability distribution obtained by the **softmax** function

- In binary classification, cross-entropy is calculated as:

$$\mathcal{L}_{\text{ce}} = -y\log(p) + (1-y)\log(1-p)$$

- In multiclass classification with one-hot labels:

$$\mathcal{L}_{\text{ce}} = -\sum_{k=1}^{K} y_k \log(p_k)$$



Cross-entropy increases as the predicted probability diverges from the label.

Optimizer

Is used to learn the network parameter to minimize the error/loss function

Determines how the network will be updated based on the loss function

As-seen-before: Gradient Descent (GD), Mini-batch GD, Quadratic Programming

For neural networks: GD and its variants

- Momentum
- RMSProp
- Adam

Today, we look into how to use a type of GD algorithm, called **backpropagation**, to compute the gradients of the network parameters

2. Backpropagation algorithm

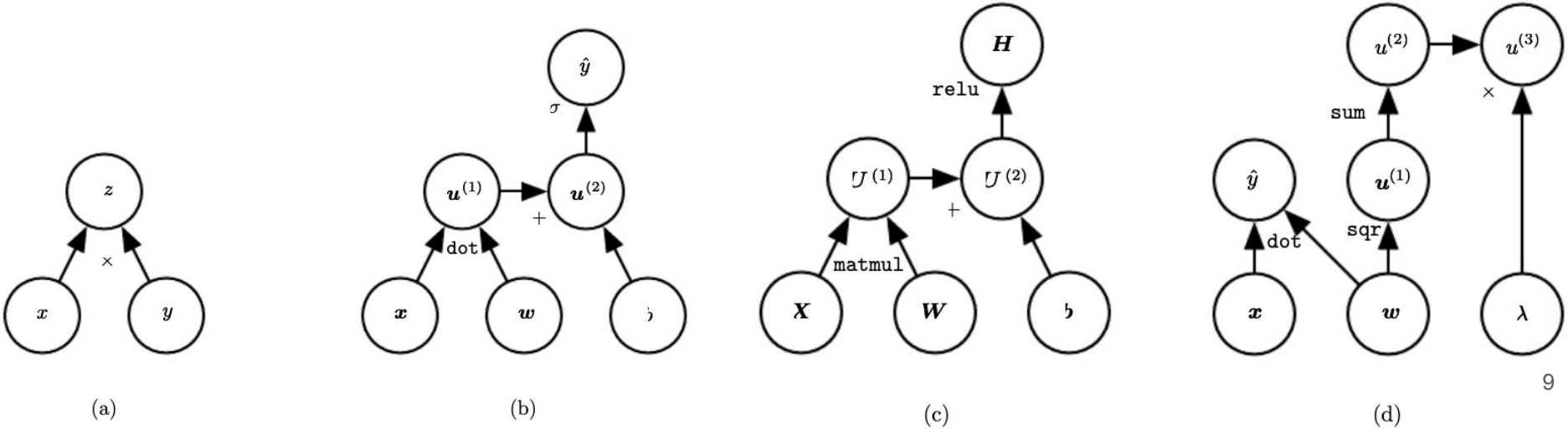
Backpropagation for training a neural network

Proposed in a 1986 groundbreaking article by **Rumelhart** et al, back-propagation (or **backprop**) algorithm consist of 3 steps:

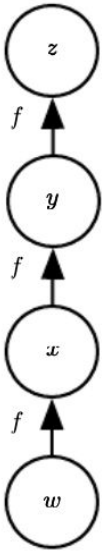
1. **Forward Pass:** Feed a training instance to the network, compute the output signal of every neuron, and measure the network error via a loss function
2. **Backward Pass:** From the network error at the output layer, measure (*propagate*) the error contributions coming from each neuron in previous layers
3. **Gradient Descent:** Run gradient descent on all connection weights using the measured error from step 2.

Computational Graphs

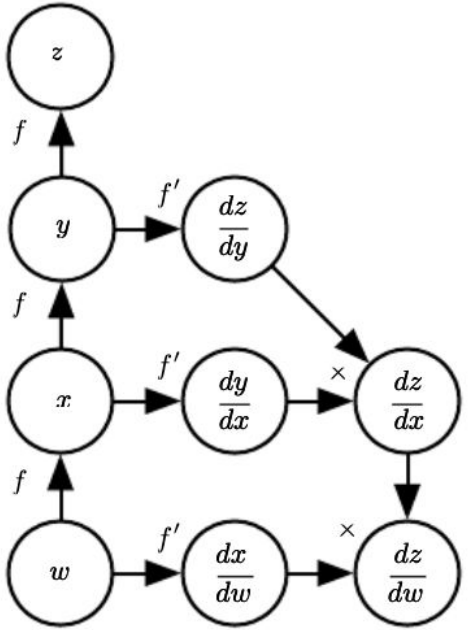
- A **computational graph** language will help describe the back-propagation algorithm more precisely
- An **operation** of a graph is a simple function of one or more variables.
- **Functions** more complicated than the operations may be described by composing many operation together.



Symbol-to-Symbol Derivatives



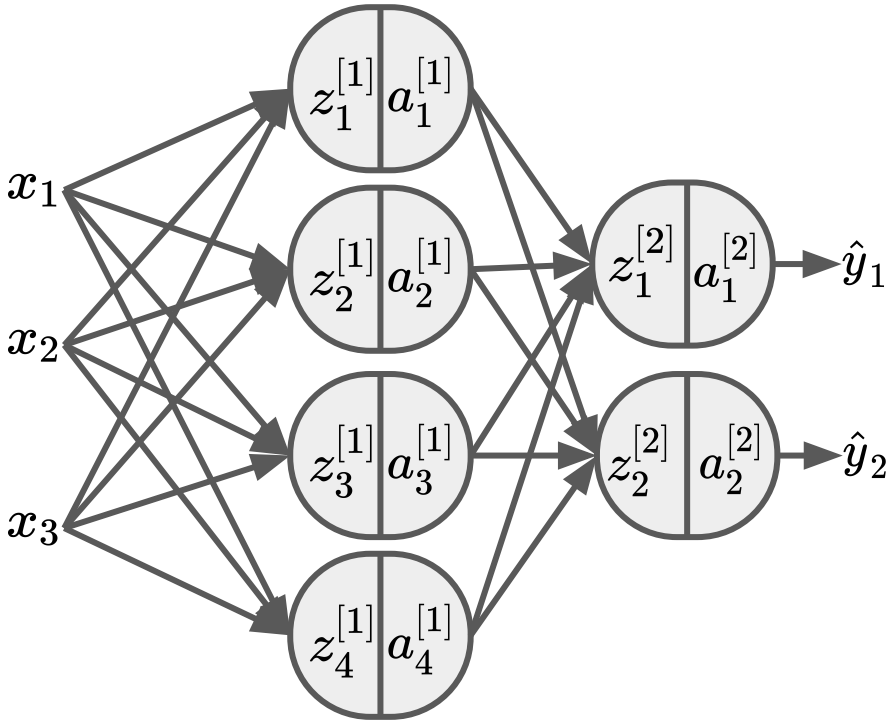
$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} = f'(y)f'(x)f'(w) = f'(f(f(w)))f'(f(w))f'(w).$$



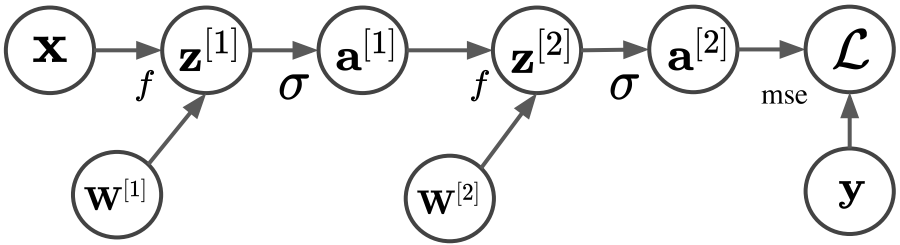
- Notice some repeated subexpressions `f(w)` in computing the derivatives
- We only need to compute `f(w)` once and store it in a variable for multiple usage.
- Using the symbol-to-symbol approach, we can describe the derivative without specifying exactly **when** each operation should be computed

Example: training a neural network

Graphical Representation

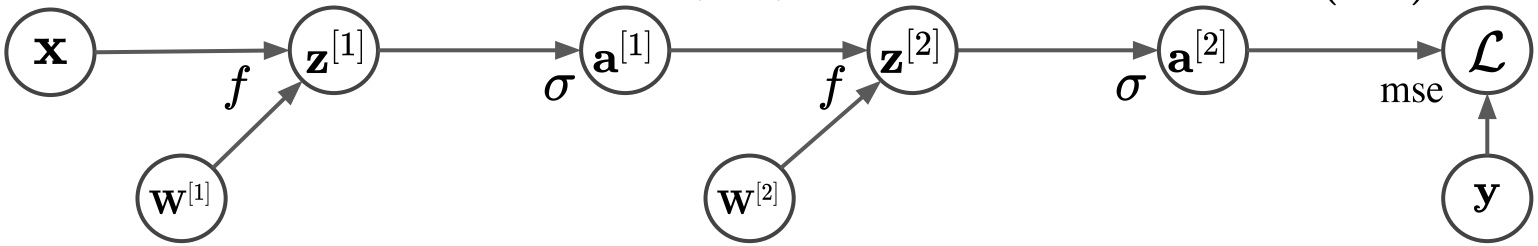


Computational Graph

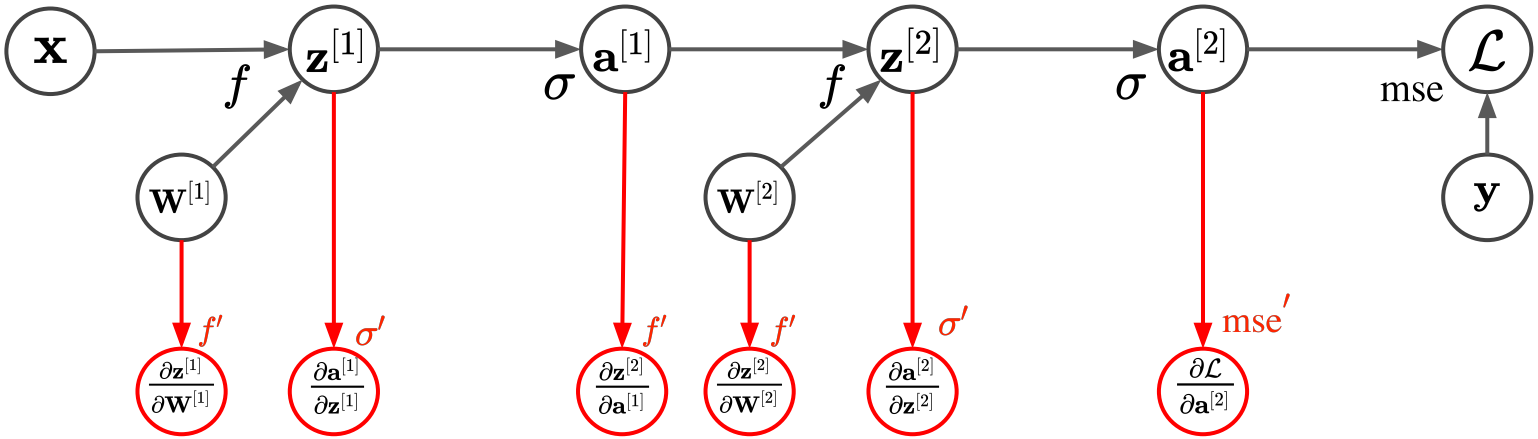


Forward Pass: Computing Loss Function

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} \quad \mathbf{a}^{[1]} = \sigma(\mathbf{z}^{[1]}) \quad \mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} \quad \mathbf{a}^{[2]} = \sigma(\mathbf{z}^{[2]}) \quad \mathcal{L}_{\text{mse}} = \frac{1}{2}||\mathbf{y} - \mathbf{a}^{[2]}||^2$$



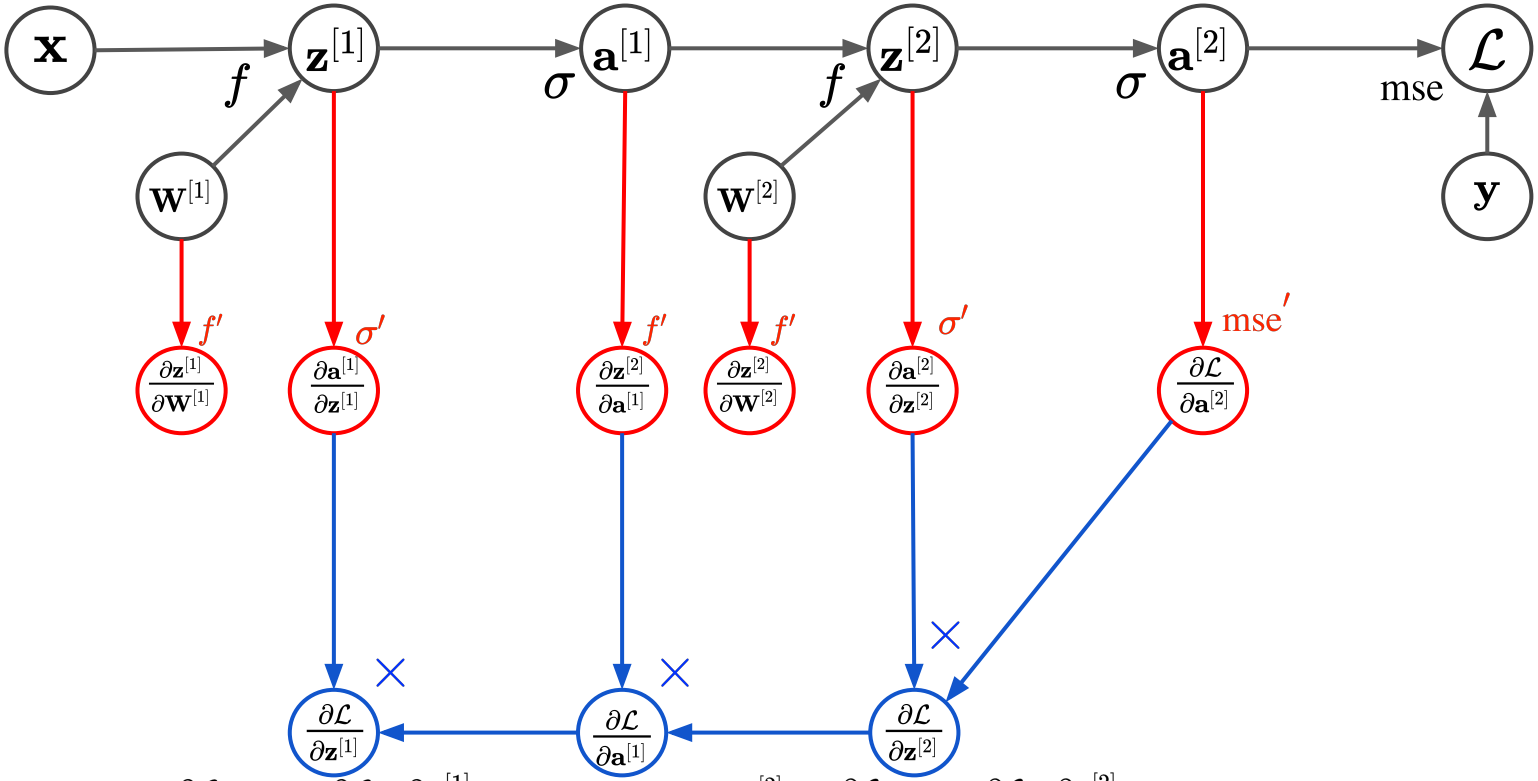
Backward Pass: Taking Partial Derivatives



$$\frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}} = \mathbf{x} \quad \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} = \sigma'(\mathbf{z}^{[1]}) \quad \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} = \mathbf{W}^{[2]} \quad \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} = \sigma'(\mathbf{z}^{[2]}) \quad \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[2]}} = \frac{\partial}{\partial \mathbf{a}^{[2]}} \frac{1}{2} ||\mathbf{y} - \mathbf{a}^{[2]}||^2 = \mathbf{y} - \mathbf{a}^{[2]}$$

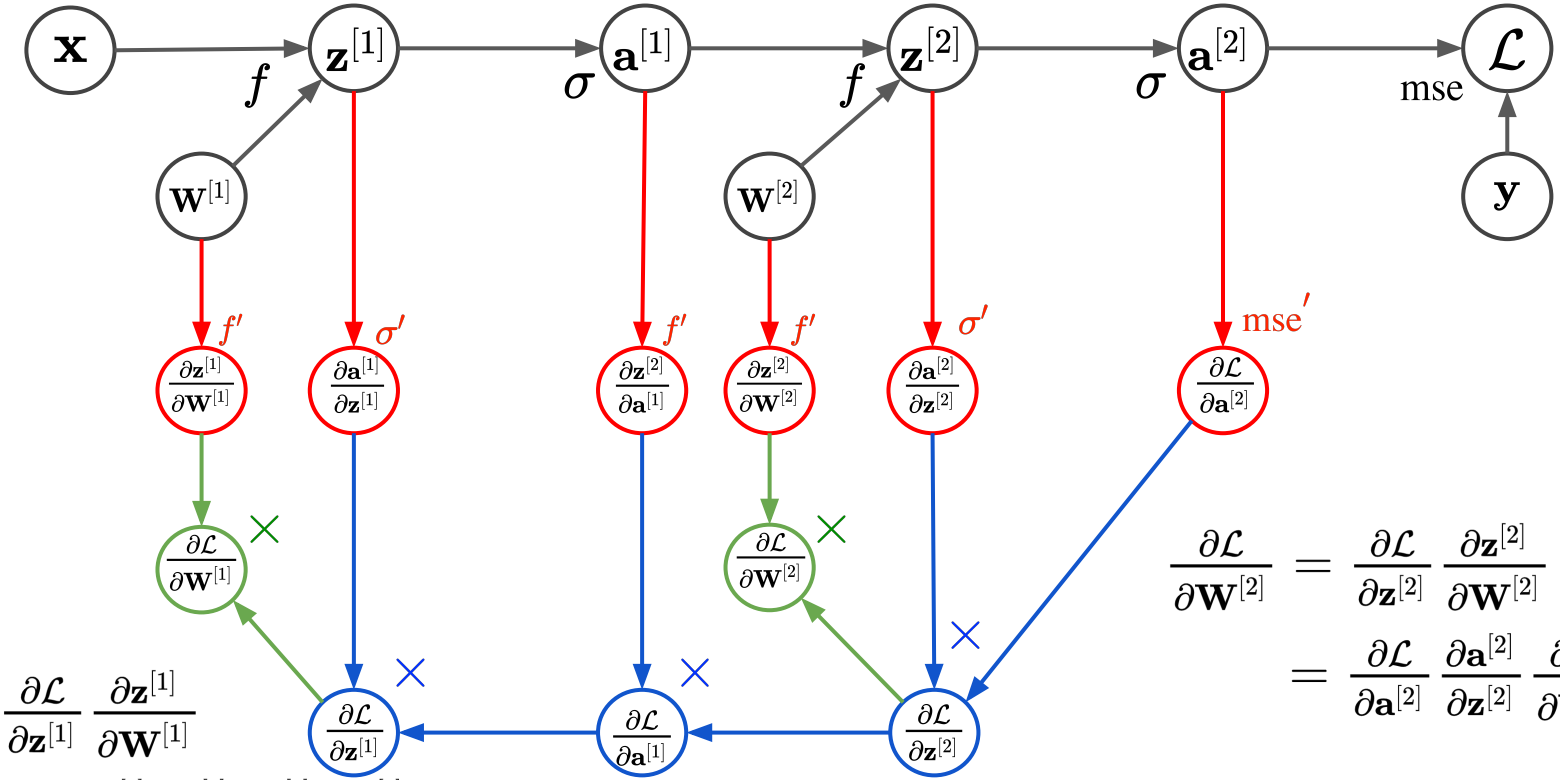
$$\frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}} = \mathbf{a}^{[1]}$$

Backward Pass: Applying Chain Rule



$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \quad \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \quad \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}}$$

Backward Pass: Getting Gradient of Parameters



$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}}$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}}$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}}$$

Backprop Algorithm

Initialize network parameters: $\mathbf{W}^{[1]}, \mathbf{W}^{[2]}$

Repeat until converge:

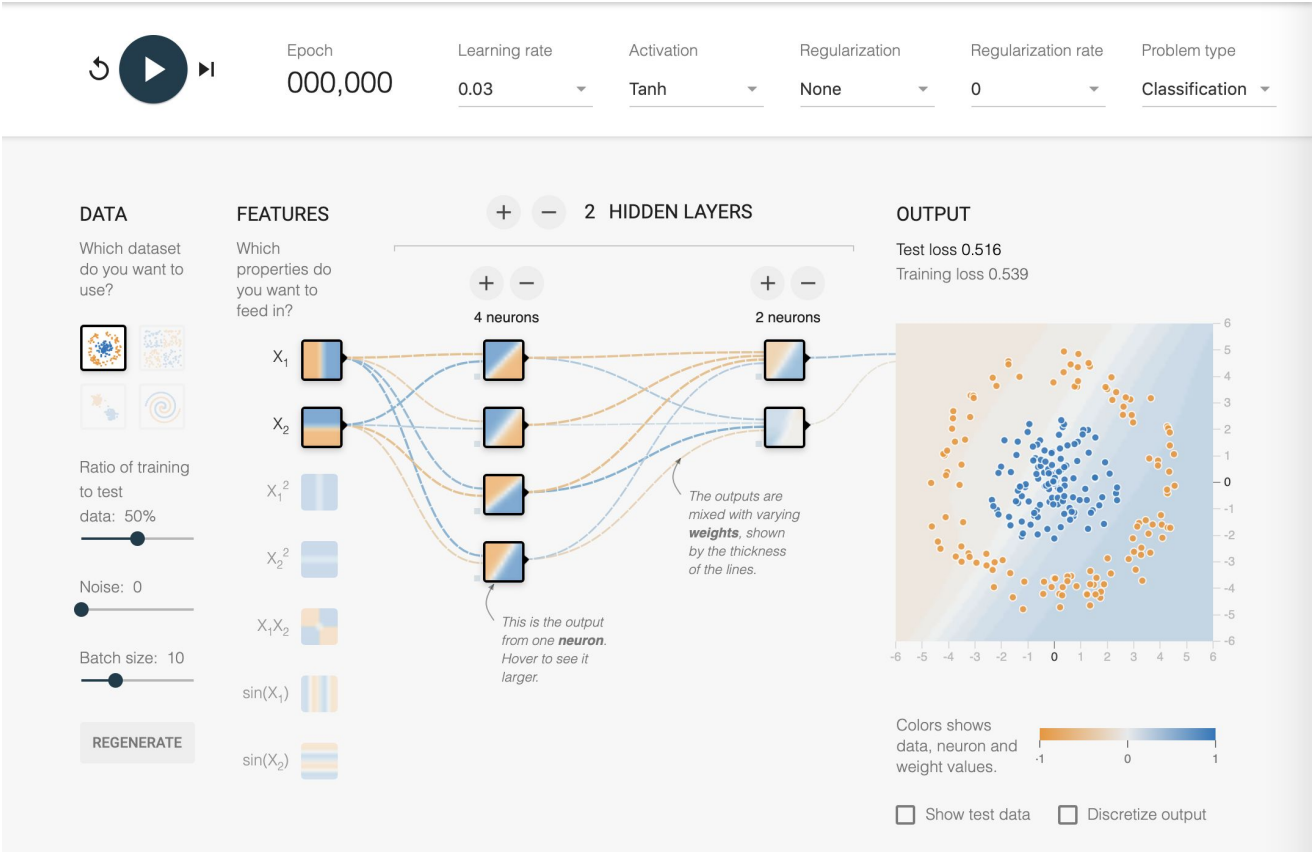
Forward pass to compute loss function $\mathcal{L}(\mathbf{a}^{[2]}, \mathbf{y})$

Backward pass to compute gradients $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1]}}, \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[2]}}$

Update network parameters: $\mathbf{W}^{[1]} = \mathbf{W}^{[1]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1]}}$

$$\mathbf{W}^{[2]} = \mathbf{W}^{[2]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[2]}}$$

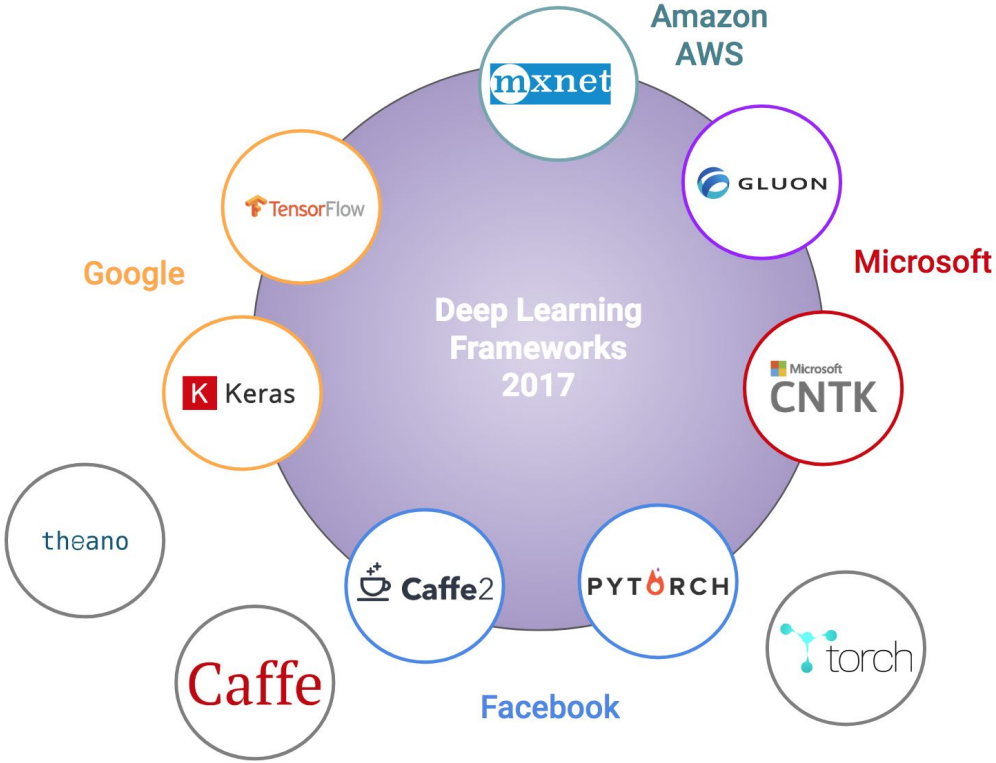
TensorFlow Playground



https://playground.tensorflow.org

3. Tensorflow + Keras API

Open source Deep Learning Libraries



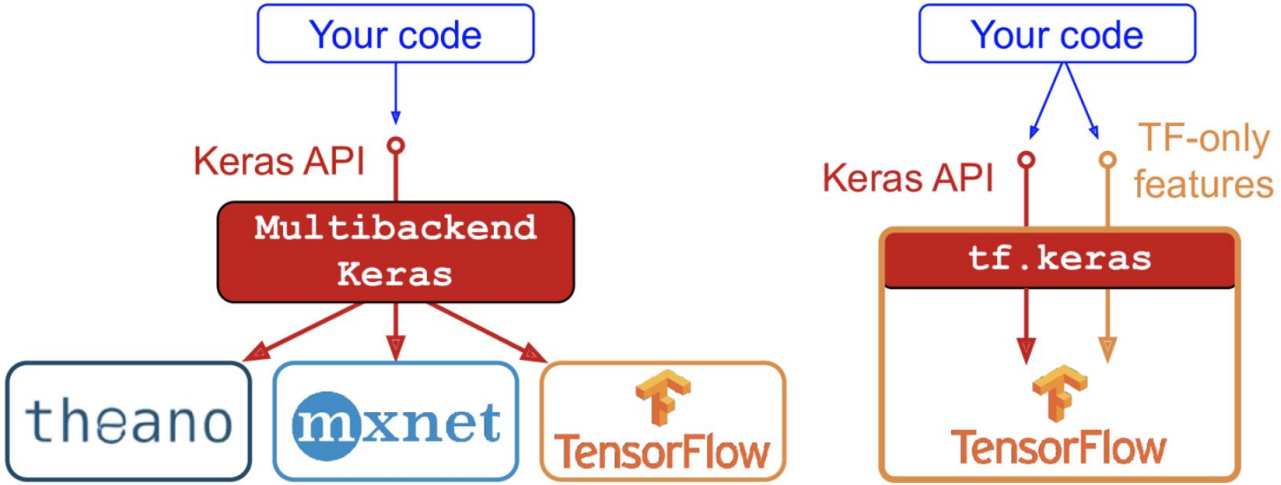
TensorFlow highlights

- Runs on Windows/Linux/MacOS/iOS/Android
- Provides simple Python API called tensorflow compatible with Scikit-Learn
- Includes highly efficient C++ implementations of ML operations
- Optimizes nodes to search for parameters that minimize a cost function
- Comes with great visualization tool called TensorBoard
- Runs on Google Cloud Platform
- Growing community of developers to contribute to improving

Keras key features

- Allows the same code to run seamlessly on CPU or GPU
- Has a user-friendly keras API to make quick prototype ML models
- Has built-in support for a variety of neural networks or their combination
- Supports flexible architectures (essentially any deep learning models).
- Is distributed under the permissive MIT license, which means it can be freely used in commercial projects.
- Has well over 200,000 users from both academics and industries

Getting started on Tensorflow and Keras




```
>>> import tensorflow as tf
>>> from tensorflow import keras
>>> tf.__version__
'2.0.0'
>>> keras.__version__
'2.2.4-tf'
```

Code Sample

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

```
>>> history = model.fit(X_train, y_train, epochs=30,
...                     validation_data=(X_valid, y_valid))
```

```
>>> model.evaluate(X_test, y_test)
10000/10000 [==========] - 0s 29us/sample - loss: 0.3340 - accuracy: 0.8851
[0.3339798209667206, 0.8851]
```

Bonus Content

Review: Chain Rule of Calculus

Chain rule is used to compute the derivative of a function formed by composing other functions whose derivatives are known.

Suppose that $y = g(x)$ and $z = f(y) = f(g(x))$, then $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$.

Suppose $\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n, \mathbf{y} = g(\mathbf{x})$ and $z = f(\mathbf{y})$, then $\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}$.

This is equivalently written in vector notation as $\nabla_{\mathbf{x}} z = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \times \nabla_{\mathbf{y}} z$

where $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is the $m \times n$ matrix of partial derivatives (Jacobian) of g.

If we use the shorthand notation $d\mathbf{x} = \nabla_{\mathbf{x}} \mathcal{L}, \Rightarrow d\mathbf{x} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \times d\mathbf{y}$

Fashion MNIST Dataset

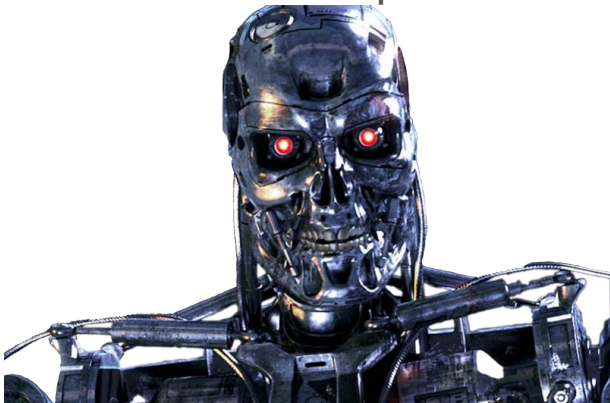
- Same format as MNIST (70,000 grayscale images of 28x28 pixels, with 10 classes)
- Images represent fashion items rather than handwritten digits
- More challenging than MNIST

Summary: Loss functions and optimizers

Loss (objective) function: the quantity that will be minimized during the training process. It represents a measure of success for the task at hand

Optimizer: determines how the network will be updated based on the loss function (ie. Stochastic Gradient Descent)

Choosing the right objective function is extremely important! If the objective function doesn't fully correlate with the task at hand, your network will end up doing things you may not have wanted...



Backward Pass: Taking partial derivatives

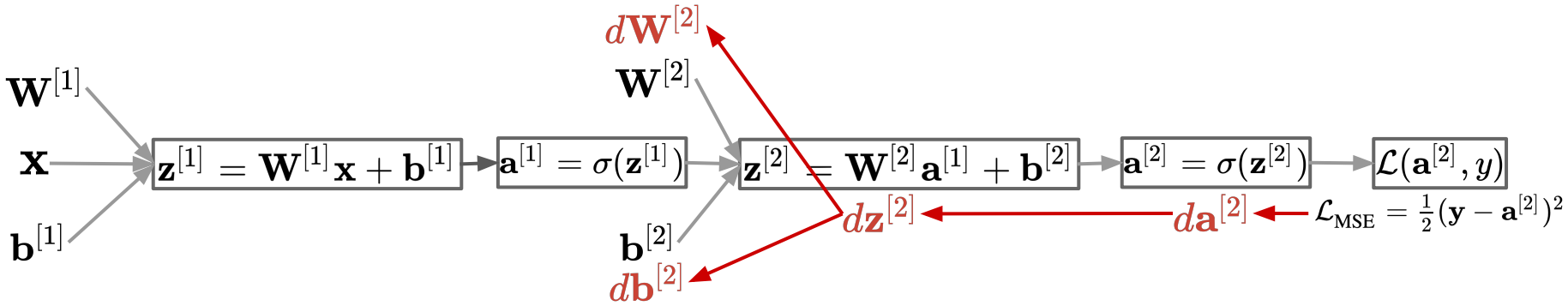
$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x}$$
$$\mathbf{a}^{[1]} = \sigma(\mathbf{z}^{[1]})$$
$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]}$$
$$\mathbf{a}^{[2]} = \sigma(\mathbf{z}^{[2]})$$
$$\mathcal{L}_{\text{mse}} = \frac{1}{2}||\mathbf{y} - \mathbf{a}^{[2]}||^2$$

$$\frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{x}} = \mathbf{W}^{[1]} \qquad \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}} = \mathbf{x}$$
$$\frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} = \sigma'(\mathbf{z}^{[1]})$$
$$\frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} = \mathbf{W}^{[2]} \qquad \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}} = \mathbf{a}^{[1]}$$
$$\frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} = \sigma'(\mathbf{z}^{[2]})$$
$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[2]}} = \frac{\partial}{\partial \mathbf{a}^{[2]}} \frac{1}{2}||\mathbf{y} - \mathbf{a}^{[2]}||^2 = \mathbf{y} - \mathbf{a}^{[2]}$$

*use square loss for simplicity

Backward Pass

$$d\mathbf{x} = \nabla_{\mathbf{x}}\mathcal{L} \Rightarrow d\mathbf{x} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \times d\mathbf{y}$$



$$d\mathbf{a}^{[2]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[2]}} = \mathbf{y} - \mathbf{a}^{[2]}$$

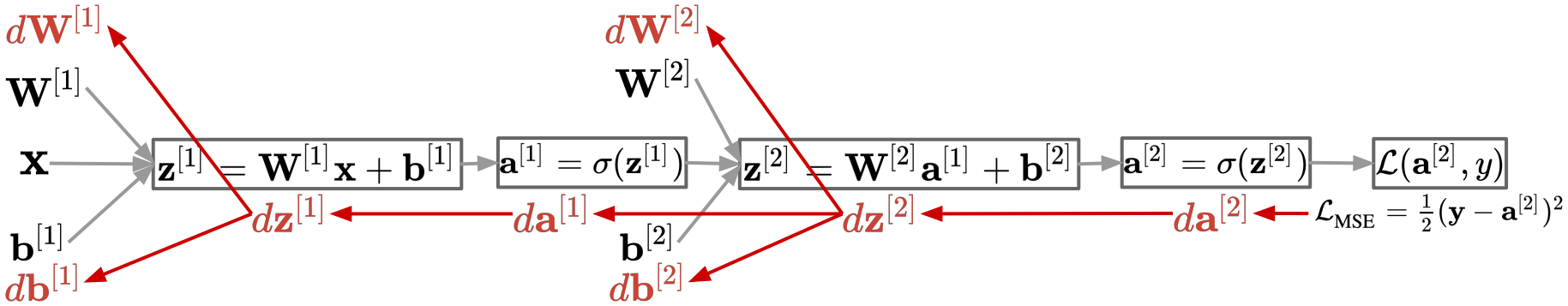
$$d\mathbf{z}^{[2]} = \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \times d\mathbf{a}^{[2]} = \sigma'(\mathbf{z}^{[2]}) \times d\mathbf{a}^{[2]}$$

$$d\mathbf{W}^{[2]} = \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}} \times d\mathbf{z}^{[2]} = \mathbf{a}^{[1]} \times d\mathbf{z}^{[2]}$$

$$d\mathbf{b}^{[2]} = \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{b}^{[2]}} \times d\mathbf{z}^{[2]} = \mathbf{I} \times d\mathbf{z}^{[2]}$$

Backward Pass (cont)

$$d\mathbf{x} = \nabla_{\mathbf{x}}\mathcal{L} \Rightarrow d\mathbf{x} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \times d\mathbf{y}$$



$$d\mathbf{a}^{[1]} = \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \times d\mathbf{z}^{[2]} = \mathbf{W}^{[2]} \times d\mathbf{z}^{[2]}$$

$$d\mathbf{z}^{[1]} = \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \times d\mathbf{a}^{[1]} = \sigma'(\mathbf{z}^{[1]}) \times d\mathbf{a}^{[1]}$$

$$d\mathbf{W}^{[1]} = \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}} \times d\mathbf{z}^{[1]} = \mathbf{x} \times d\mathbf{z}^{[1]}$$

$$d\mathbf{b}^{[1]} = \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{b}^{[1]}} \times d\mathbf{z}^{[1]} = \mathbf{I} \times d\mathbf{z}^{[1]}$$

Computational Graph for Training