LECTURE 21

Convolutional Neural Networks (CNNs)

Building CNNs in Using Keras

Data Science, Fall 2024 @ Knowledge Stream

Sana Jabbar

Review

Lecture 21

- Introduction to Neural Networks
- Neural network forward pass
- Activation functions
- Back Propagation

- Introduction to Neural Networks
- Neural network forward pass
- Activation functions
- **Back Propagation**

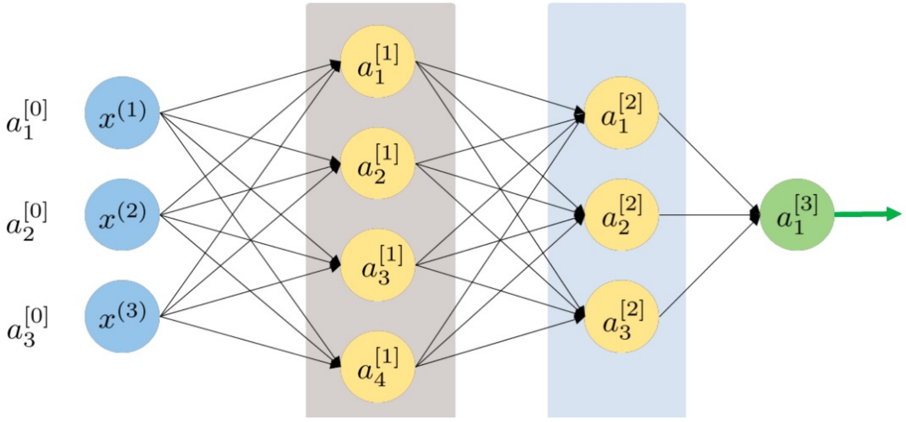
Back Propagation

Lecture 21

Learning Weights

Given the training data, we want to learn the weights (weight matrices+bias vectors) for hidden layers and output layer.

Example:



$\mathbf{W}^{[1]}$ $\mathbf{W}^{[2]}$ $\mathbf{W}^{[3]}$

$\mathbf{b}^{[1]}$ $\mathbf{b}^{[2]}$ $\mathbf{b}^{[3]}$

Parameters need to be learned!

Notation Revisit

- L - number of layers.
- $\mathbf{a}^{[\ell]} = \mathbf{x}$ input layer.
- $\mathbf{a}^{[L]} = y$ output layer.
- Number of nodes in the ℓ-th layer, $m^{[\ell]}$
- $\mathbf{a}^{[\ell]}$ - vector of outputs of ℓ-th node.
- $a_i^{[\ell]}$ denotes the output of i-th node in the ℓ-th layer.

Learning Weights

- We assume we have training data D given by

$$D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$

- Given our prior knowledge, output y is a composite function of input $\mathbf{x}$. Therefore, it is continuous and differentiable and we can use chain rule to compute the gradient.

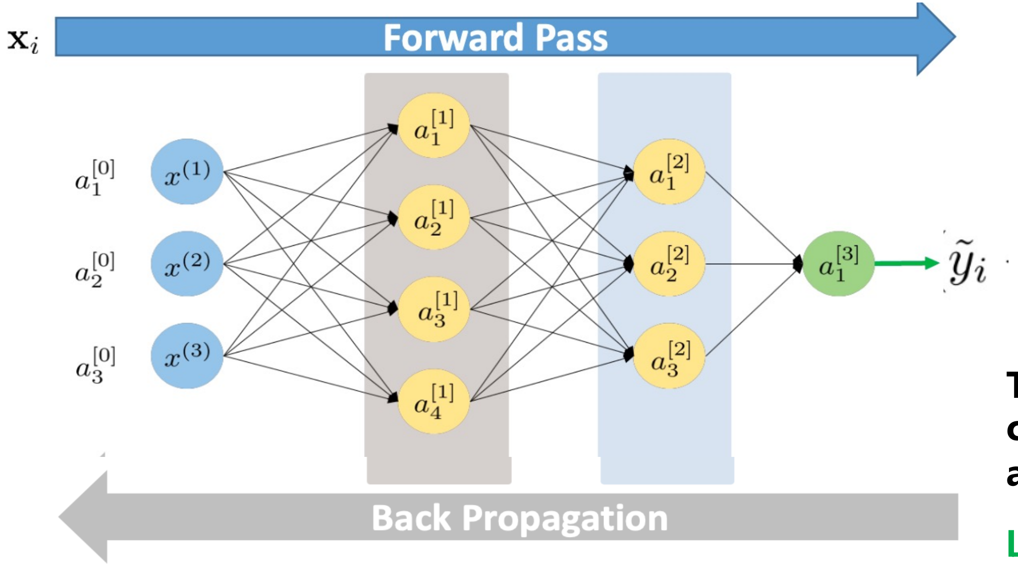
where $\tilde{y}_i$ denotes the output of the neural network for i-th input.

$$\mathcal{L} = \frac{1}{2} \sum_{i=1}^{n} \left(\tilde{y}_i - y_i\right)^2$$

- We can use gradient descent to learn the weight matrices and bias vectors.

We use a method called 'Back Propagation' to implement the chain rule for the computation the gradient.

Learning Weights



$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{n}\left(\tilde{y}_i - y_i\right)^2$$

$$w_{i,j}^{[\ell]} = w_{i,j}^{[\ell]} - \alpha\frac{\partial \mathcal{L}}{\partial w_{i,j}^{[\ell]}}$$

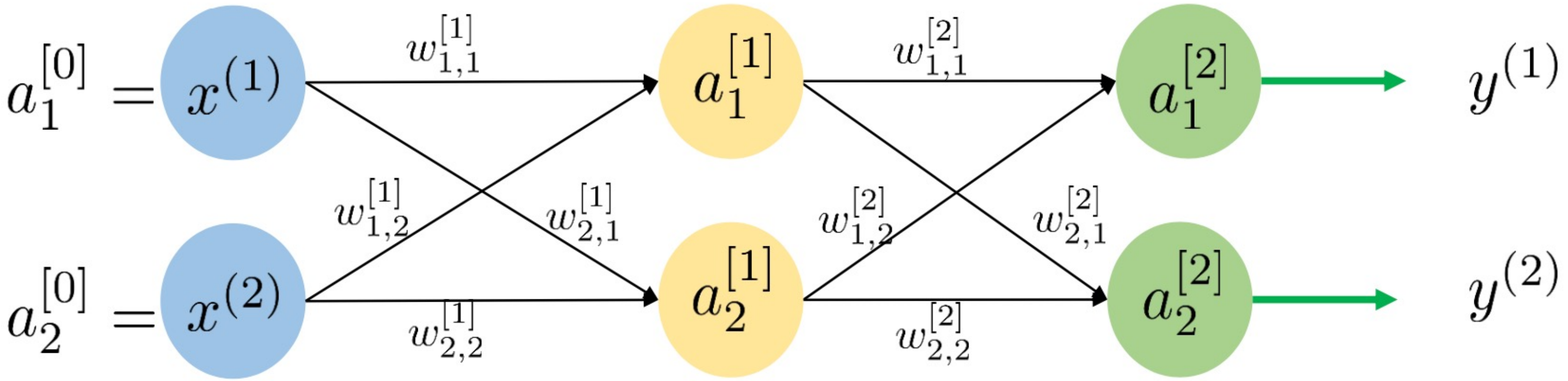
The weights are the only parameters that can be modified to make the loss function as low as possible.

Learning problem reduces to the question of calculating gradient (partial derivatives) of loss function.

- We compute the derivate by propagating the total loss at the output node back into the neural network to determine the contribution of every node in the loss.

Learning Weights

- 2 layer with 2 neurons in the hidden layer , 2 inputs, 2 outputs network.
- Assuming sigmoid as activation function, that is, $g(z) = \sigma(z)$.



- Given training data

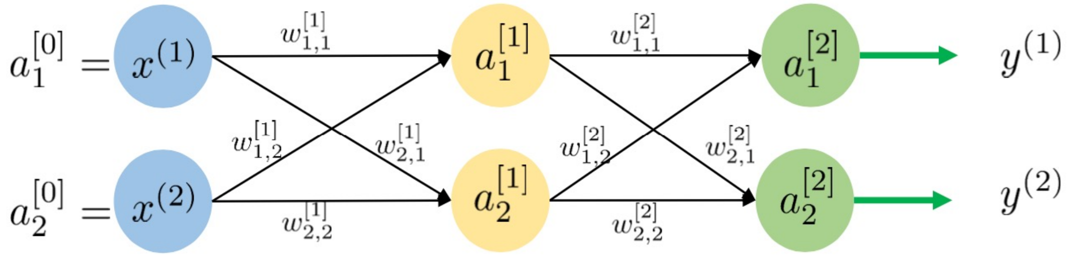
$$x^{(1)} = 0.05, \quad x^{(2)} = 0.1, \quad y^{(1)} = 0.01, \quad y^{(2)} = 0.99$$

- Initial values of weights and biases:

$$w_{1,1}^{[1]} = 0.15,\ w_{1,2}^{[1]} = 0.2,\ w_{2,1}^{[1]} = 0.25,\ w_{2,2}^{[1]} = 0.3,\ b_1^{[1]} = 0.35,\ b_2^{[1]} = 0.35.$$

$$w_{1,1}^{[2]} = 0.4,\ w_{1,2}^{[2]} = 0.45,\ w_{2,1}^{[2]} = 0.5,\ w_{2,2}^{[2]} = 0.55,\ b_1^{[1]} = 0.6,\ b_2^{[1]} = 0.6.$$

Learning Weights



- Loss function (noting output is a vector):

$$\mathcal{L} = \frac{1}{2}\|(\tilde{y}^{(1)} - y^{(1)})^2 - (\tilde{y}^{(2)} - y^{(2)})^2\|^2$$

$$\mathcal{L} = \frac{1}{2}\|(0.01, 0.99) - (0.7514, 0.7729)\|^2 = 0.2984$$

Forward Pass

$$a_1^{[1]} = g(z_1^{[1]}), \quad z_1^{[1]} = \mathbf{w}_1^{[1]^T}\mathbf{x} + b_1^{[1]}$$

$$a_2^{[1]} = g(z_2^{[1]}), \quad z_2^{[1]} = \mathbf{w}_2^{[1]^T}\mathbf{x} + b_2^{[1]}$$

$$a_1^{[2]} = g(z_1^{[2]}), \quad z_1^{[2]} = \mathbf{w}_1^{[2]^T}\mathbf{x} + b_1^{[2]}$$

$$a_2^{[2]} = g(z_2^{[2]}), \quad z_2^{[2]} = \mathbf{w}_2^{[2]^T}\mathbf{x} + b_2^{[2]}$$

$$z_1^{[1]} = w_{1,1}^{[1]}x^{(1)} + w_{1,2}^{[1]}x^{(2)} + b_1^{[1]} = 0.3775, \quad a_1^{[1]} = g(0.3775) = 0.5933$$

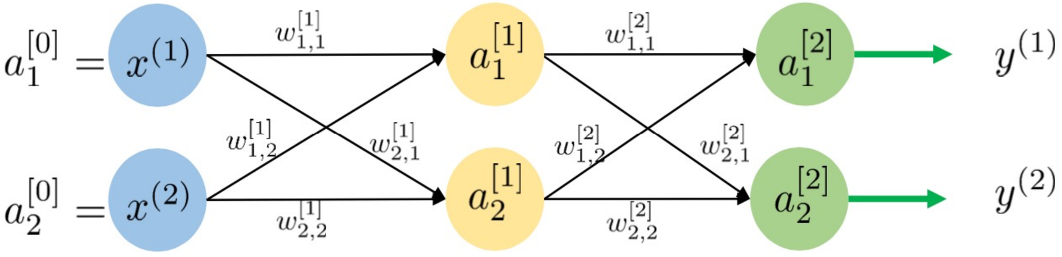
$$z_2^{[1]} = \mathbf{w}_2^{[1]^T}\mathbf{x} + b_2^{[1]} = 0.3925, \quad a_2^{[1]} = g(0.3925) = 0.5969$$

$$z_1^{[2]} = \mathbf{w}_1^{[2]^T}\mathbf{x} + b_1^{[2]} = 1.106, \quad a_1^{[2]} = g(1.106) = 0.7514 = \tilde{y}^{(1)}$$

$$z_2^{[2]} = \mathbf{w}_2^{[2]^T}\mathbf{x} + b_2^{[2]} = 1.225, \quad a_2^{[2]} = g(1.225) = 0.7729 = \tilde{y}^{(2)}$$

Nothing fancy so far, we have computed the output and loss by traversing neural network. Let's compute the contribution of loss by each node; back propagate the loss.

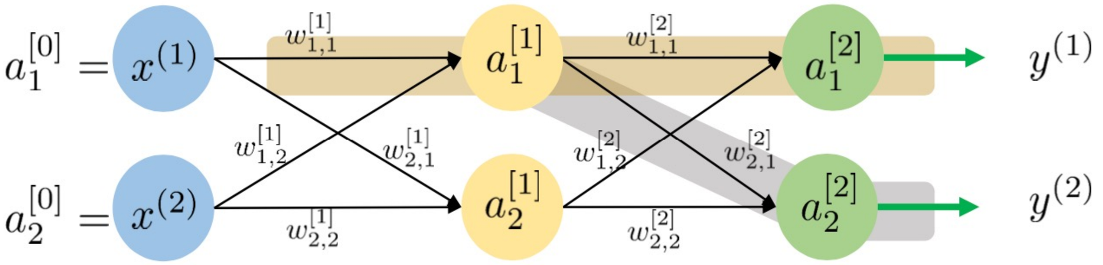
Learning Weights



- Consider a case when we want to compute $\dfrac{\partial \mathcal{L}}{\partial w_{1,1}^{[2]}}$
- Traverse the path from the loss function back to the weight $w_{1,1}^{[2]}$:

$$\left.\begin{aligned}
\mathcal{L} &= \frac{1}{2}\|(\tilde{y}^{(1)} - y^{(1)})^2 - (\tilde{y}^{(2)} - y^{(2)})^2\|^2 \\
\tilde{y}^{(2)} &= \sigma(z_1^{[2]}) \\
z_1^{[2]} &= w_{1,1}^{[2]} a_1^{[1]} + w_{1,2}^{[2]} a_2^{[1]} + b_1^{[2]}
\end{aligned}\right\}
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_{1,1}^{[2]}} &= \frac{\partial \mathcal{L}}{\partial \tilde{y}^{(1)}} \frac{\partial \tilde{y}^{(1)}}{\partial z_1^{[2]}} \frac{\partial z_1^{[2]}}{\partial w_{1,1}^{[2]}} \\
&= 0.0821
\end{aligned}
\left\{\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \tilde{y}^{(1)}} &= \tilde{y}^{(1)} - y^{(1)} = 0.7414 \\
\frac{\partial \tilde{y}^{(1)}}{\partial z_1^{[2]}} &= \sigma(\partial z_1^{[2]})\Big(1 - \sigma(\partial z_1^{[2]})\Big) = 0.1868 \\
\frac{\partial z_1^{[2]}}{\partial w_{1,1}^{[2]}} &= a_1^{[a]} = 0.5933
\end{aligned}\right.$$

Learning Weights



$$\mathcal{L} = \frac{1}{2}\|(\tilde{y}^{(1)} - y^{(1)})^2 - (\tilde{y}^{(2)} - y^{(2)})^2\|^2$$

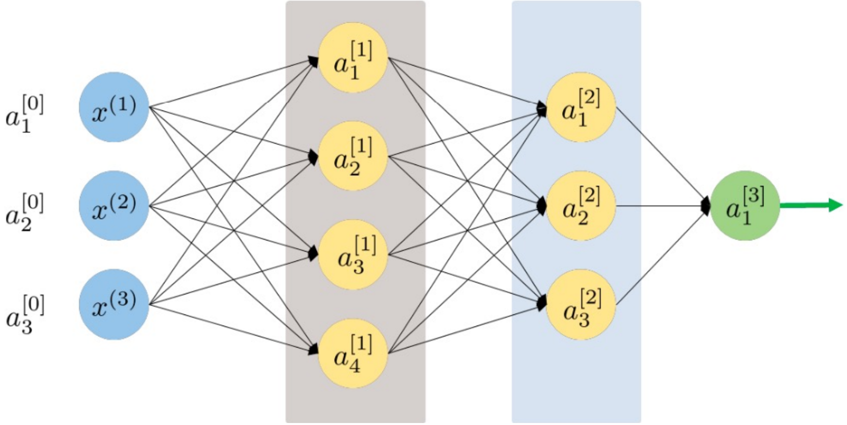
- Consider a case when we want to compute $\dfrac{\partial \mathcal{L}}{\partial w_{1,1}^{[1]}}$
- Traverse the path from the loss function back to the weight $w_{1,1}^{[1]}$. There are two paths from the output to the weight $w_{1,1}^{[1]}$. In other words, $w_{1,1}^{[1]}$ is contributing to both the outputs.

$$\frac{\partial \mathcal{L}}{\partial w_{1,1}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \tilde{y}^{(1)}} \frac{\partial \tilde{y}^{(1)}}{\partial z_1^{[2]}} \frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \frac{\partial z_1^{[1]}}{\partial w_{1,1}^{[1]}} + \frac{\partial \mathcal{L}}{\partial \tilde{y}^{(2)}} \frac{\partial \tilde{y}^{(2)}}{\partial z_2^{[2]}} \frac{\partial z_2^{[2]}}{\partial a_1^{[1]}} \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \frac{\partial z_1^{[1]}}{\partial w_{1,1}^{[1]}}$$

- Looking tedious but the concept is very straightforward. I encourage you to write one partial derivative using the same approach to strengthen the concept.

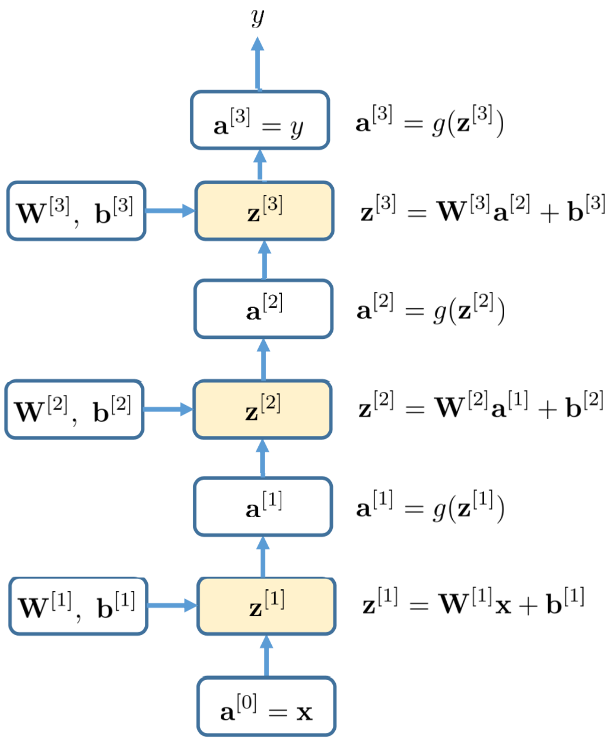
Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.



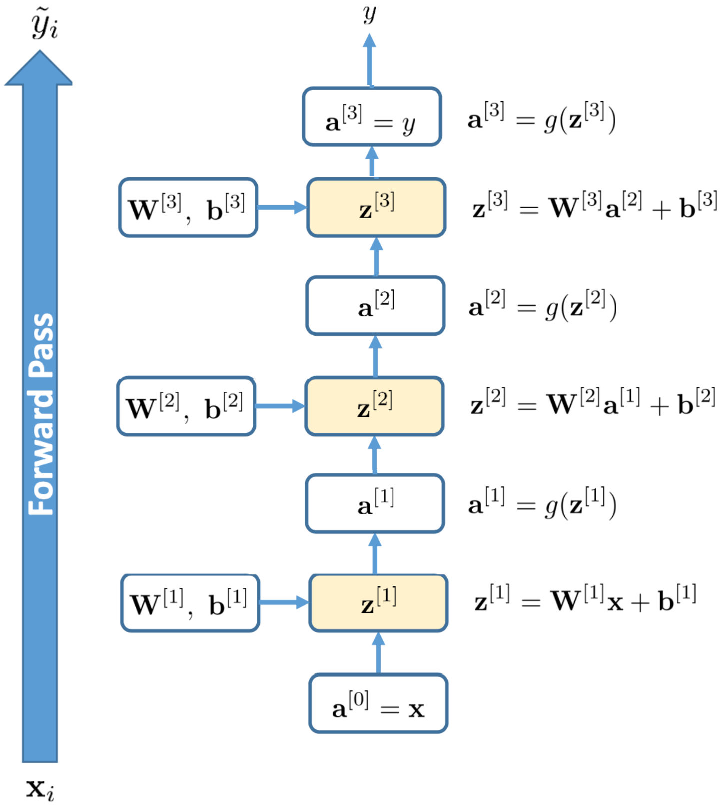
Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.



Back Propagation: Vectorization

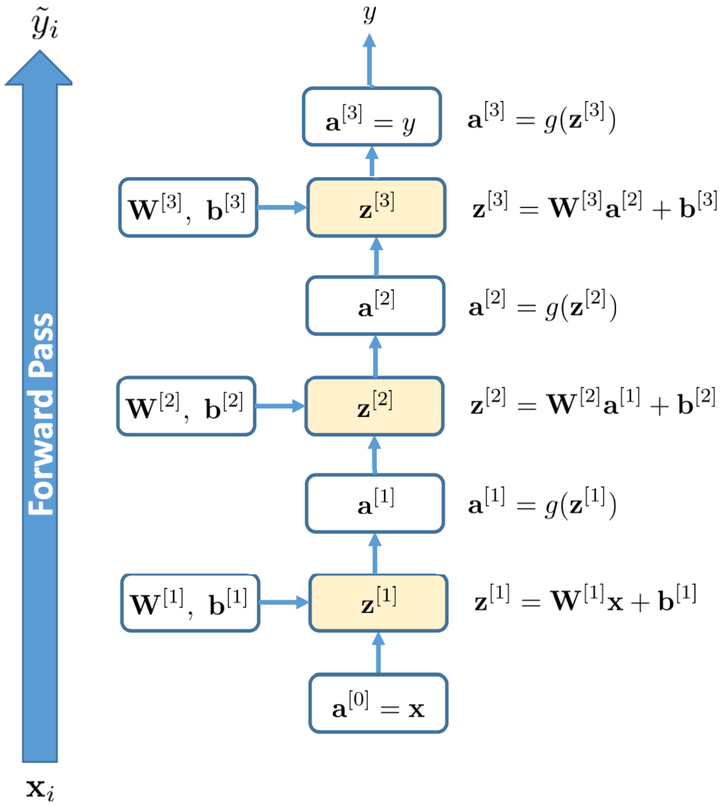
- We compute loss function $\mathcal{L}$ using forward pass.



Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.
- We update $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ using gradient descent as:

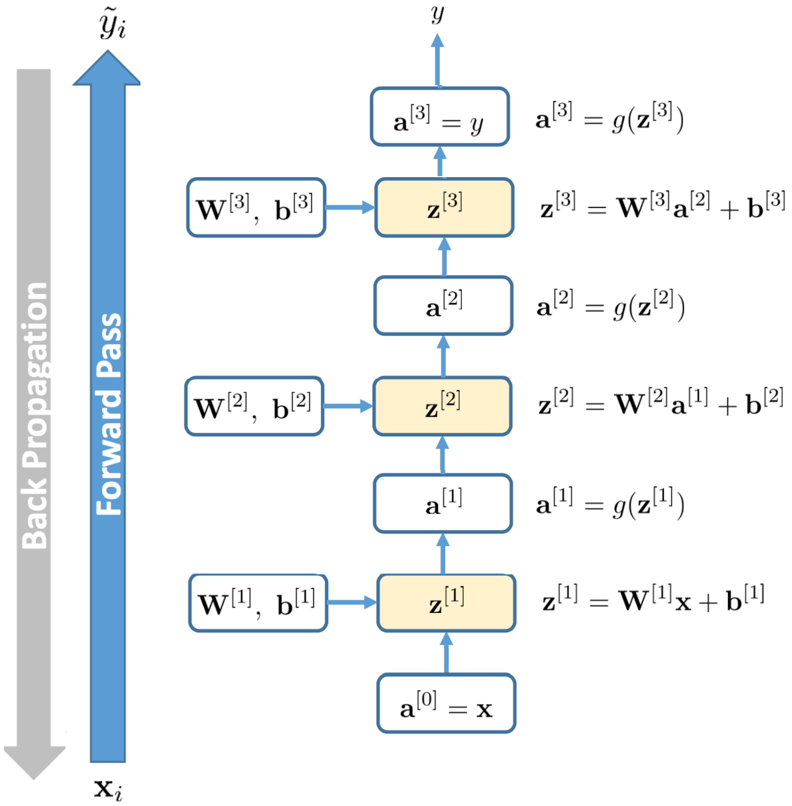
$$\mathbf{W}^{[\ell]} = \mathbf{W}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[\ell]}} \qquad \mathbf{b}^{[\ell]} = \mathbf{b}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[\ell]}}$$



Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.
- We update $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ using gradient descent as:

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



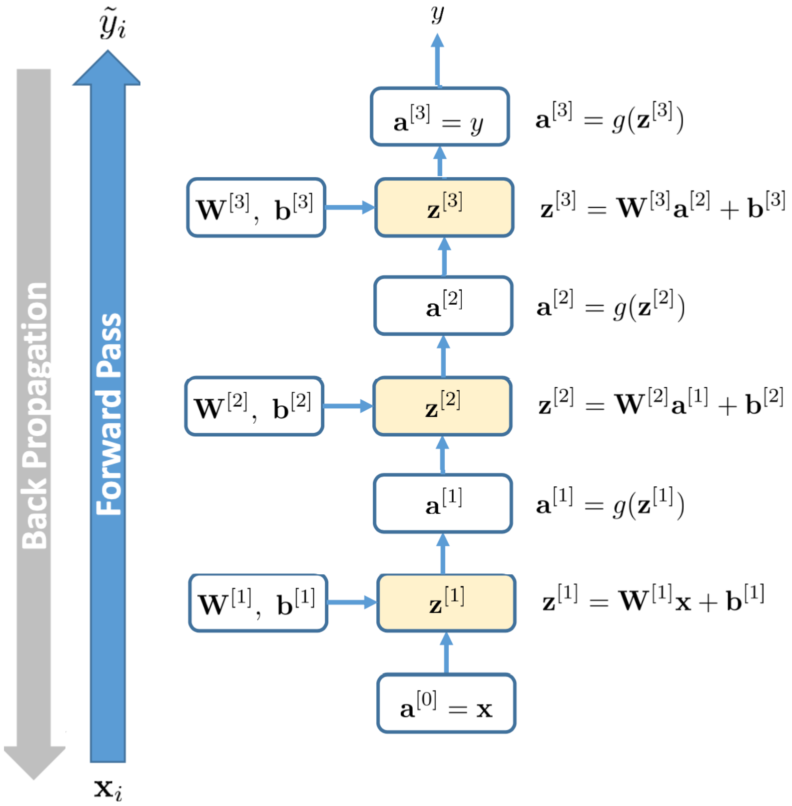
Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.
- We update $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ using gradient descent as:

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

Partial Derivatives:

$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{W}^{[3]}} \qquad \frac{\partial \mathcal{L}}{\mathbf{b}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{b}^{[3]}}$$



Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.

- We update $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ using gradient descent as:

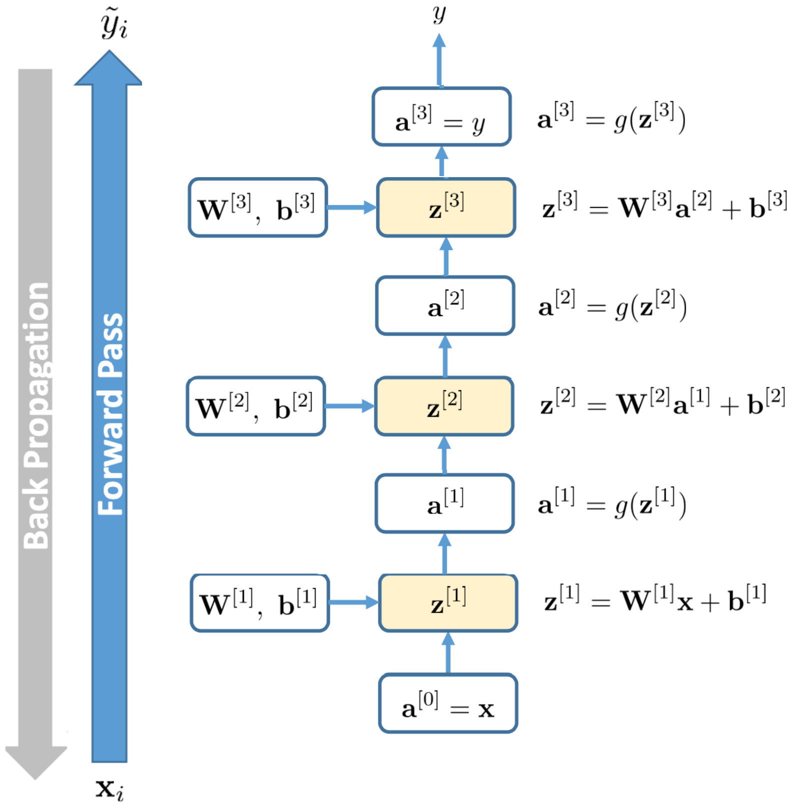
$$\mathbf{W}^{[\ell]} = \mathbf{W}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[\ell]}} \qquad \mathbf{b}^{[\ell]} = \mathbf{b}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[\ell]}}$$

Partial Derivatives:

$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{W}^{[3]}} \qquad \frac{\partial \mathcal{L}}{\mathbf{b}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{b}^{[3]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{b}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{b}^{[2]}}$$



Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.
- We update $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ using gradient descent as:

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

Partial Derivatives:

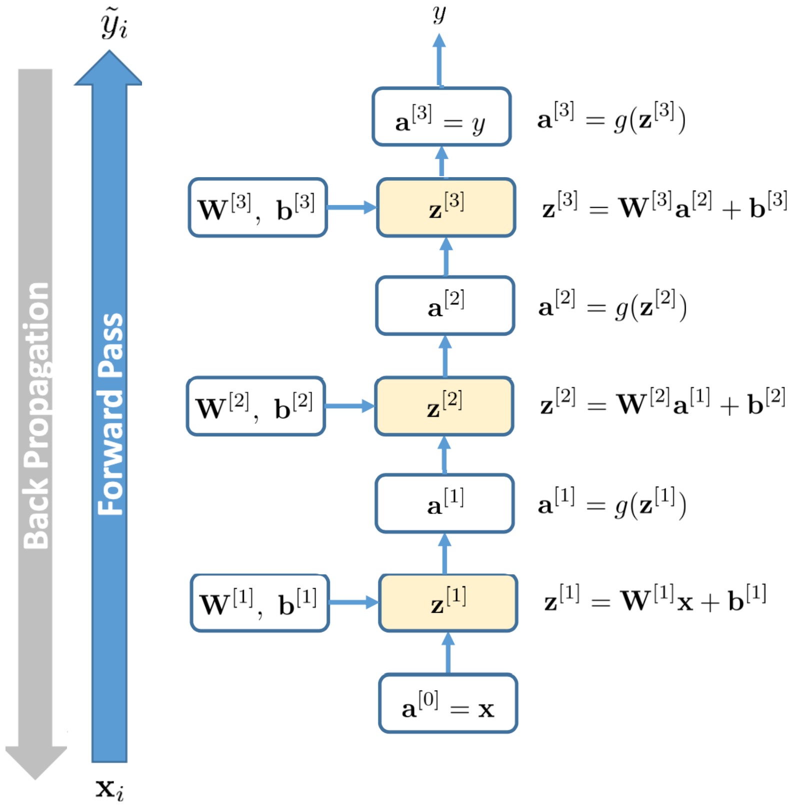
$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{W}^{[3]}} \qquad \frac{\partial \mathcal{L}}{\mathbf{b}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{b}^{[3]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{b}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{b}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{b}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{b}^{[1]}}$$



Neural Networks In Keras

```
• from keras.models import Sequential
• from keras.layers import Dense

• # built model
• model = Sequential([
• Dense(32, activation='relu', input_shape=(10,)),
• Dense(32, activation='relu'),
• Dense(1, activation='sigmoid'),
• ])

• # compile your model
• model.compile(optimizer='sgd',
• loss='binary_crossentropy',
• metrics=['accuracy'])
```

Neural Networks In Keras

- `# now train your model using fit`
- `hist = model.fit(X_train, Y_train,`
- `batch_size=32, epochs=100,`
- `validation_data=(X_val, Y_val))`

- `# evaluate your model using model.evaluate`
- `model.evaluate(X_test, Y_test)[1]`

Convolutional Networks

Convolutional Neural Networks

- Automatic feature extraction.
- Highly accurate at image recognition & classification.
- Weight sharing.
- Minimizes computation.
- Ability to handle large datasets.
- Hierarchical learning