Deep Learning Principles & Applications

Chapter 3 – Shallow Neural Network

Sudarsan N.S. Acharya (sudarsan.acharya@manipal.edu)

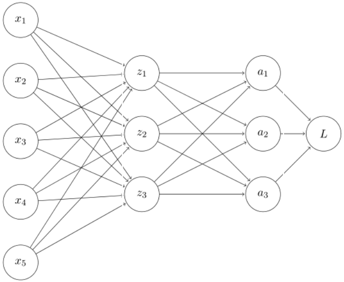
Softmax classifier as a zero hidden layer neural network

Softmax classifier as a zero hidden layer neural network

A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:

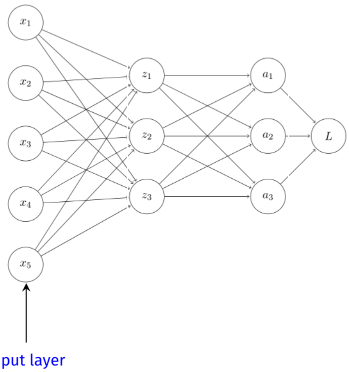
Softmax classifier as a zero hidden layer neural network

A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:



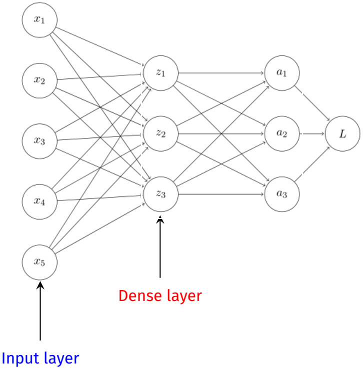
Softmax classifier as a zero hidden layer neural network

A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:



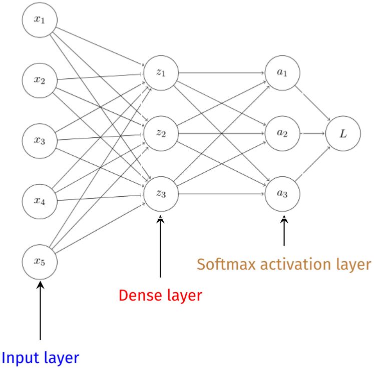
Softmax classifier as a zero hidden layer neural network

A *layered visualization* of applying the softmax classifier to a sample x with 5 features and correct output label y from 3 possible output labels:



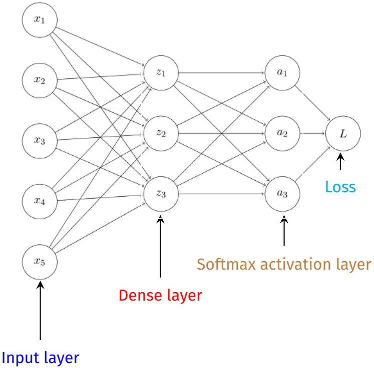
Softmax classifier as a zero hidden layer neural network

A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:



Softmax classifier as a zero hidden layer neural network

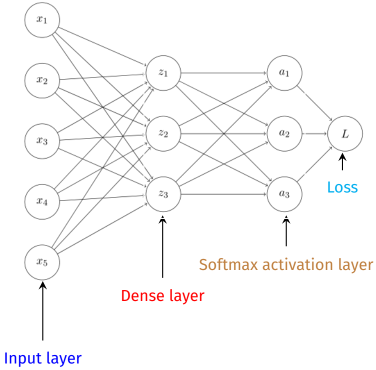
A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:



Softmax classifier as a zero hidden layer neural network

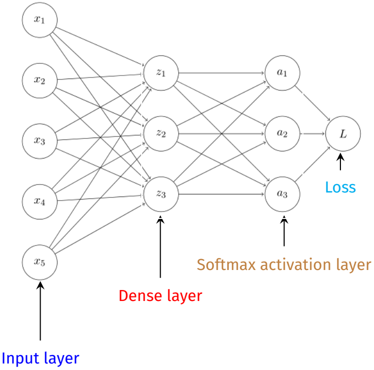
A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:

- Bias feature 1 is excluded for clarity.



Softmax classifier as a zero hidden layer neural network

A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:

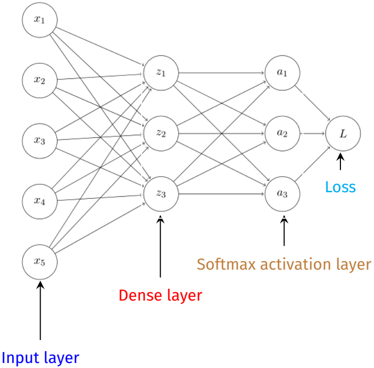


- Bias feature 1 is excluded for clarity.
- Input layer:

Softmax classifier as a zero hidden layer neural network

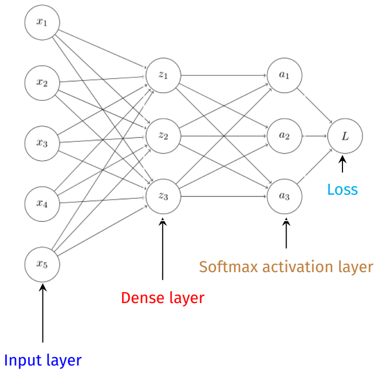
A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:

- Bias feature 1 is excluded for clarity.
- Input layer: the sample vector $\mathbf{x}$.



Softmax classifier as a zero hidden layer neural network

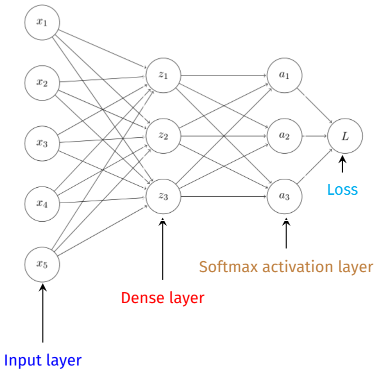
A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:



- Bias feature 1 is excluded for clarity.
- Input layer: the sample vector $\mathbf{x}$.
- Dense layer:

Softmax classifier as a zero hidden layer neural network

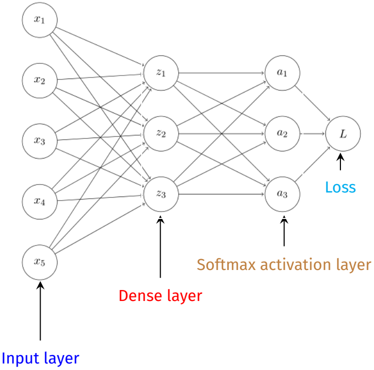
A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:



- Bias feature 1 is excluded for clarity.
- Input layer: the sample vector $\mathbf{x}$.
- Dense layer: owns the 3×5-weights matrix $\mathbf{W}$ and calculates the raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x}$.

Softmax classifier as a zero hidden layer neural network

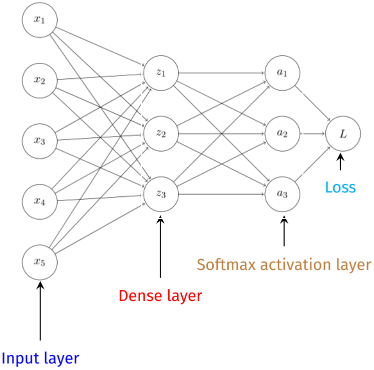
A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:



- Bias feature 1 is excluded for clarity.
- Input layer: the sample vector $\mathbf{x}$.
- Dense layer: owns the 3×5-weights matrix $\mathbf{W}$ and calculates the raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x}$.
- Softmax activation layer:

Softmax classifier as a zero hidden layer neural network

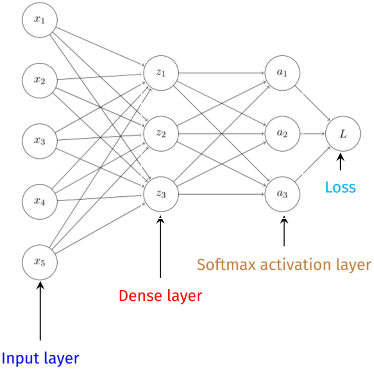
A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:



- Bias feature 1 is excluded for clarity.
- Input layer: the sample vector $\mathbf{x}$.
- Dense layer: owns the 3×5-weights matrix $\mathbf{W}$ and calculates the raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x}$.
- Softmax activation layer: activates the raw scores vector as $\mathbf{a} = \text{softmax}(\mathbf{z})$.

Softmax classifier as a zero hidden layer neural network

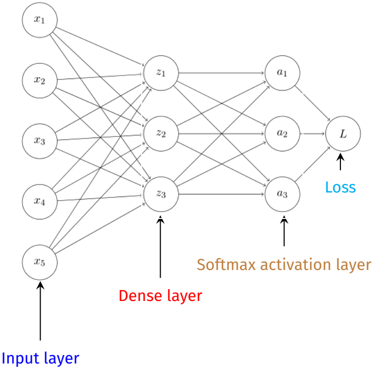
A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:



- Bias feature 1 is excluded for clarity.
- Input layer: the sample vector $\mathbf{x}$.
- Dense layer: owns the 3×5-weights matrix $\mathbf{W}$ and calculates the raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x}$.
- Softmax activation layer: activates the raw scores vector as $\mathbf{a} = \text{softmax}(\mathbf{z})$.
- Loss:

Softmax classifier as a zero hidden layer neural network

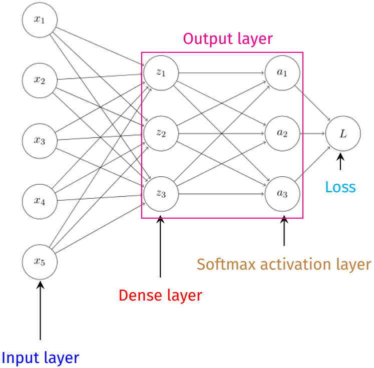
A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:



- Bias feature 1 is excluded for clarity.
- Input layer: the sample vector $\mathbf{x}$.
- Dense layer: owns the 3×5-weights matrix $\mathbf{W}$ and calculates the raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x}$.
- Softmax activation layer: activates the raw scores vector as $\mathbf{a} = \text{softmax}(\mathbf{z})$.
- Loss: $L = -\log\left([\mathbf{a}]_y\right)$

Softmax classifier as a zero hidden layer neural network

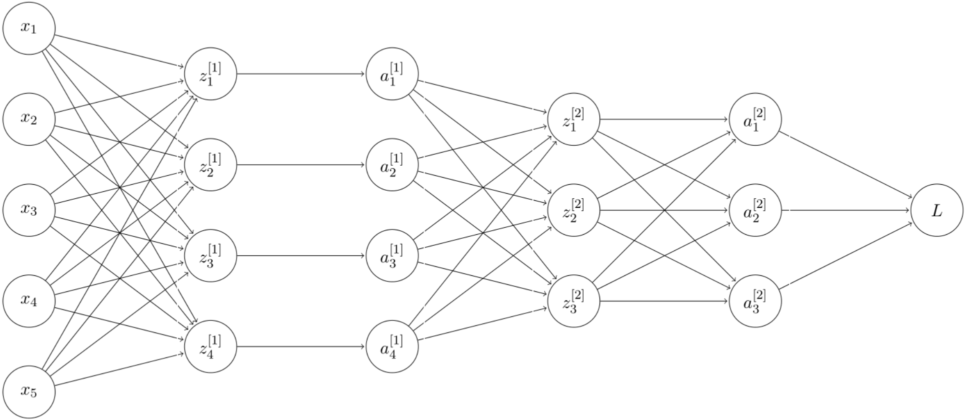
A *layered visualization* of applying the softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels:



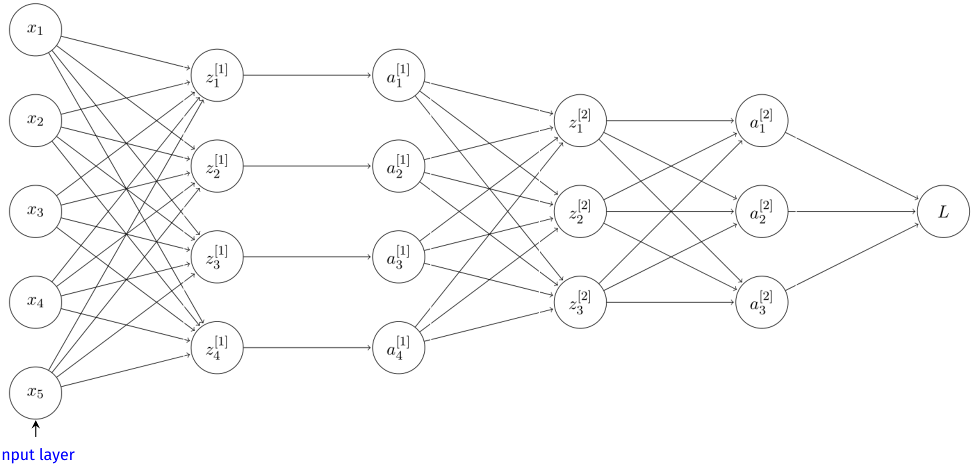
- Bias feature 1 is excluded for clarity.
- Input layer: the sample vector $\mathbf{x}$.
- Dense layer: owns the 3×5-weights matrix $\mathbf{W}$ and calculates the raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x}$.
- Softmax activation layer: activates the raw scores vector as $\mathbf{a} = \text{softmax}(\mathbf{z})$.
- Loss: $L = -\log\left([\mathbf{a}]_y\right)$
- Dense + Softmax activation layers put together is called the output layer.

Single hidden layer neural network: architecture

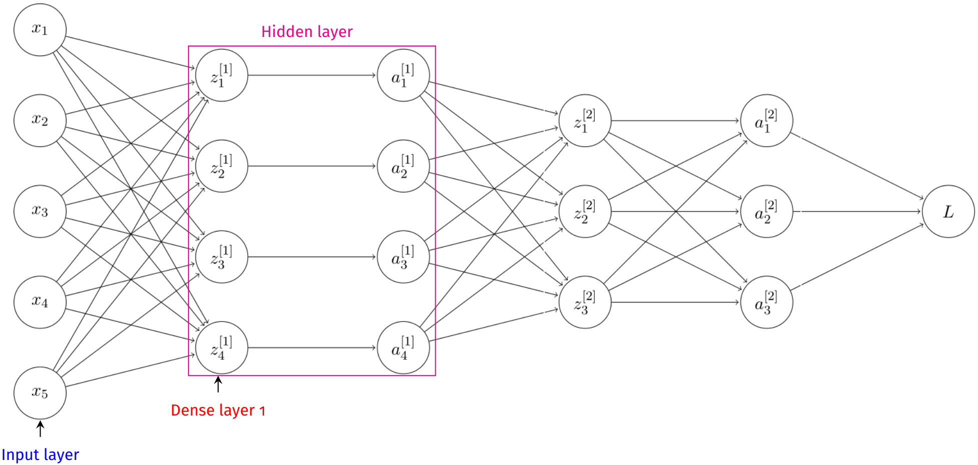
Single hidden layer neural network: architecture



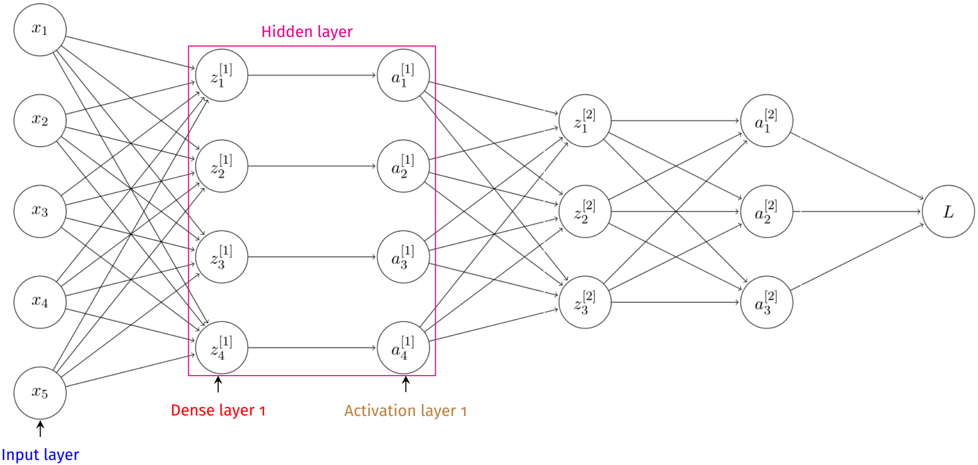
Single hidden layer neural network: architecture



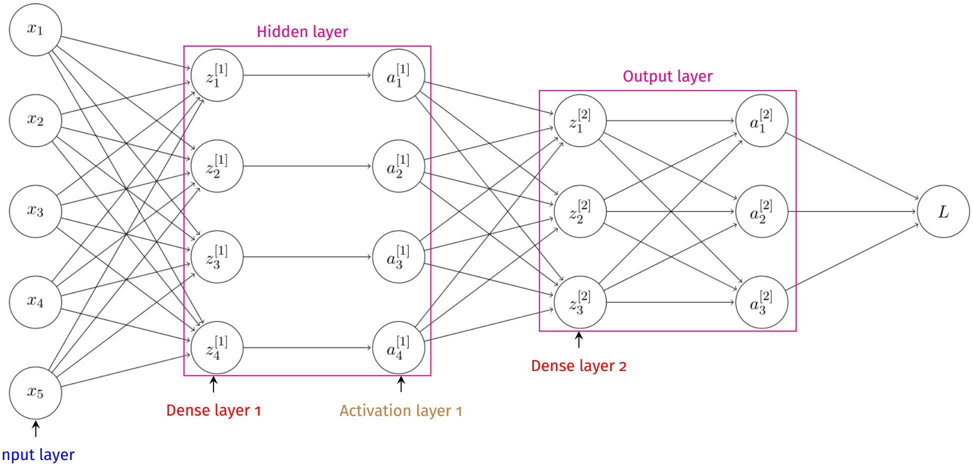
Single hidden layer neural network: architecture



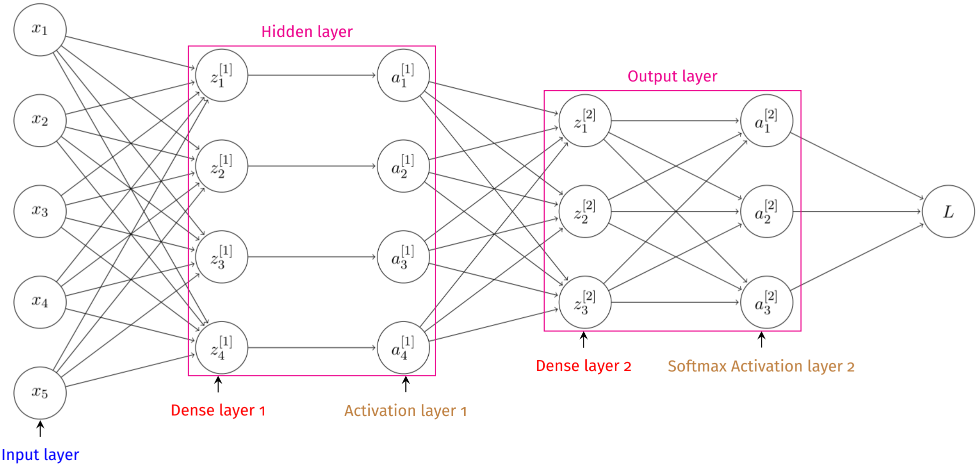
Single hidden layer neural network: architecture



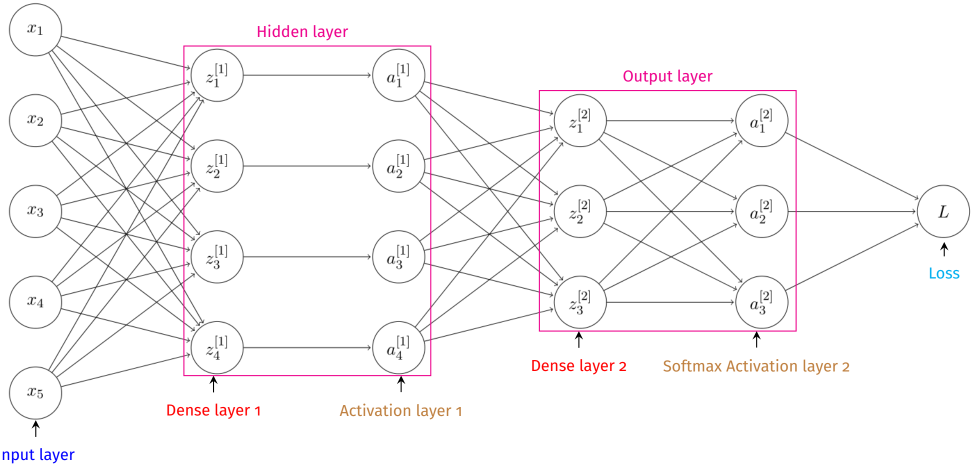
Single hidden layer neural network: architecture



Single hidden layer neural network: architecture



Single hidden layer neural network: architecture



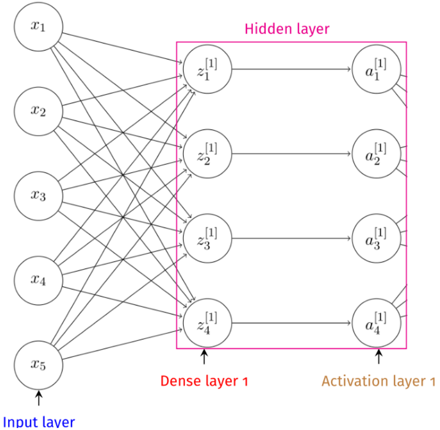
Single hidden layer neural network: notation

Single hidden layer neural network: notation

Focus on the input layer (5 nodes) and the hidden layer (4 nodes):

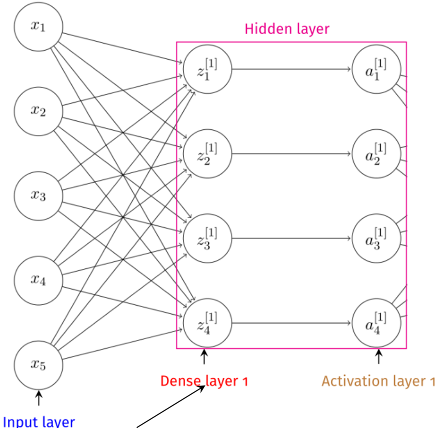
Single hidden layer neural network: notation

Focus on the input layer (5 nodes) and the hidden layer (4 nodes):



Single hidden layer neural network: notation

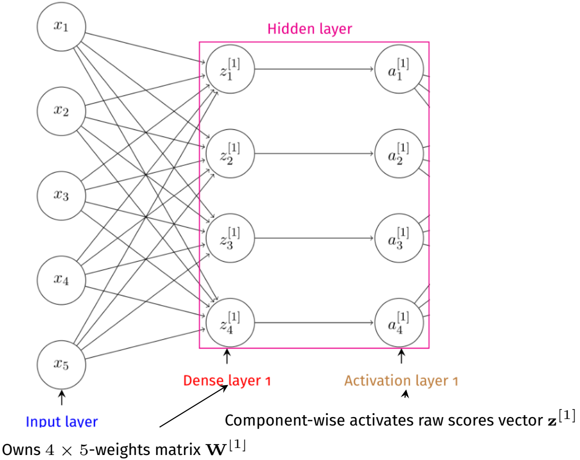
Focus on the input layer (5 nodes) and the hidden layer (4 nodes):



Owns 4×5-weights matrix $\mathbf{W}^{[1]}$

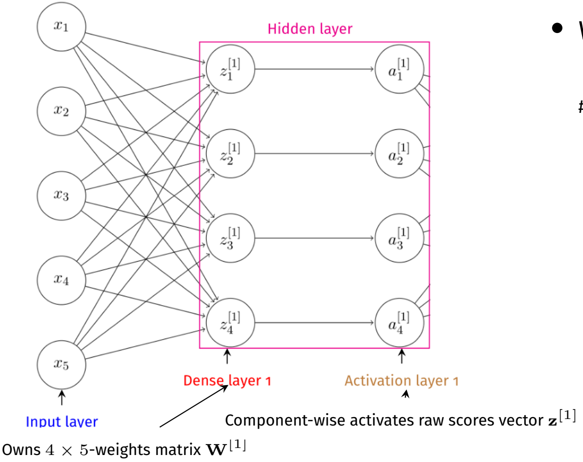
Single hidden layer neural network: notation

Focus on the input layer (5 nodes) and the hidden layer (4 nodes):



Single hidden layer neural network: notation

Focus on the input layer (5 nodes) and the hidden layer (4 nodes):

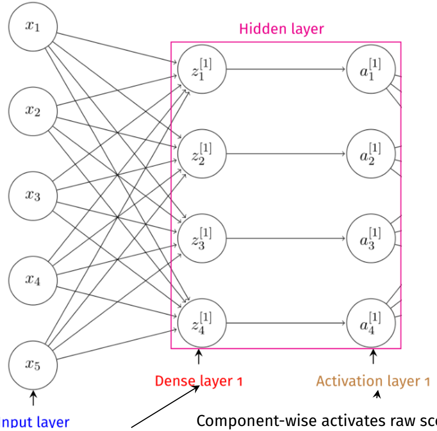


- Weights matrix $\mathbf{W}^{[1]}$ has shape

$$\underbrace{4}_{\text{\# output raw scores in dense layer 1}} \times \underbrace{5}_{\text{\# features in input layer}}$$

Single hidden layer neural network: notation

Focus on the input layer (5 nodes) and the hidden layer (4 nodes):



Component-wise activates raw scores vector $\mathbf{z}^{[1]}$

Owns 4×5-weights matrix $\mathbf{W}^{[1]}$

- Weights matrix $\mathbf{W}^{[1]}$ has shape

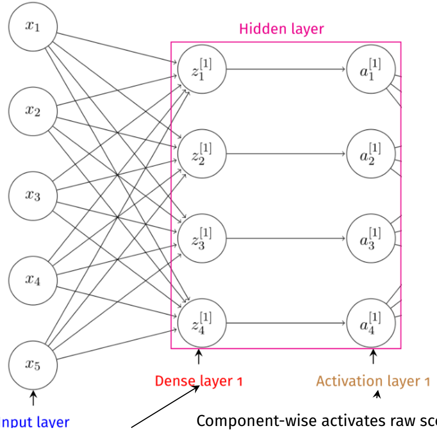
$$\underbrace{4}_{\text{\# output raw scores in dense layer 1}} \times \underbrace{5}_{\text{\# features in input layer}}$$

- Raw scores vector for dense layer 1 is

$$\mathbf{z}^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}.$$

Single hidden layer neural network: notation

Focus on the input layer (5 nodes) and the hidden layer (4 nodes):



Component-wise activates raw scores vector $\mathbf{z}^{[1]}$

Owns 4×5-weights matrix $\mathbf{W}^{[1]}$

- Weights matrix $\mathbf{W}^{[1]}$ has shape

$$\underbrace{4}_{\text{\# output raw scores in dense layer 1}} \times \underbrace{5}_{\text{\# features in input layer}}$$

- Raw scores vector for dense layer 1 is

$$\mathbf{z}^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}.$$

- Activated scores for activation layer 1 is $\mathbf{a}^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \begin{bmatrix} g\left(z_1^{[1]}\right) \\ g\left(z_2^{[1]}\right) \\ g\left(z_3^{[1]}\right) \\ g\left(z_4^{[1]}\right) \end{bmatrix}$, where g is the *layer's activation function*.

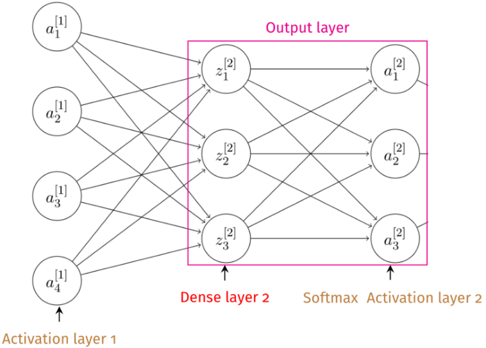
Single hidden layer neural network: notation – continued

Single hidden layer neural network: notation – continued

Focus on activation layer 1 (4 nodes) and the output layer (3 nodes):

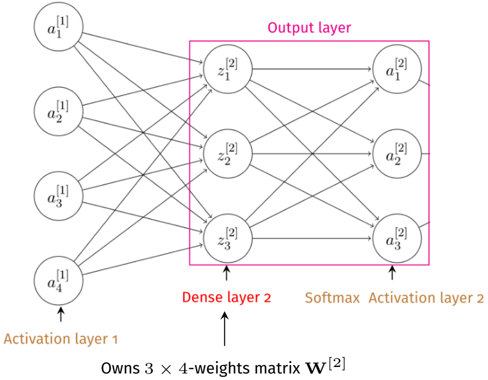
Single hidden layer neural network: notation – continued

Focus on activation layer 1 (4 nodes) and the output layer (3 nodes):



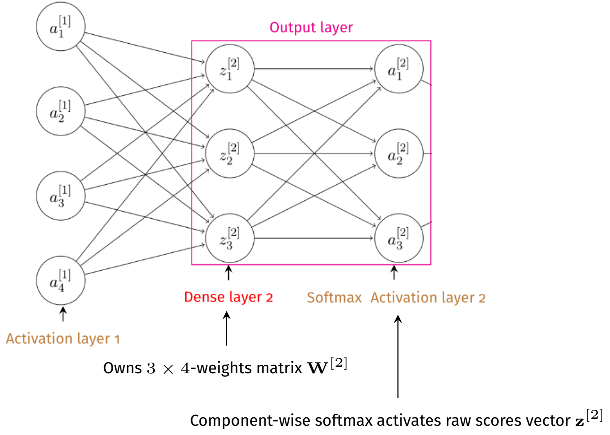
Single hidden layer neural network: notation – continued

Focus on activation layer 1 (4 nodes) and the output layer (3 nodes):



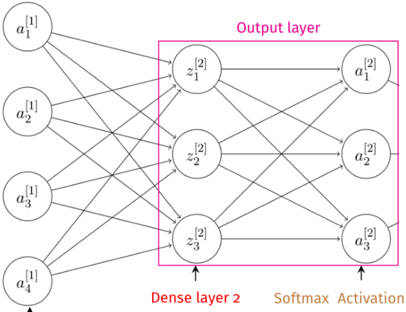
Single hidden layer neural network: notation – continued

Focus on activation layer 1 (4 nodes) and the output layer (3 nodes):



Single hidden layer neural network: notation – continued

Focus on activation layer 1 (4 nodes) and the output layer (3 nodes):



Dense layer 2 Softmax Activation layer 2

Activation layer 1

Owns 3×4-weights matrix $\mathbf{W}^{[2]}$

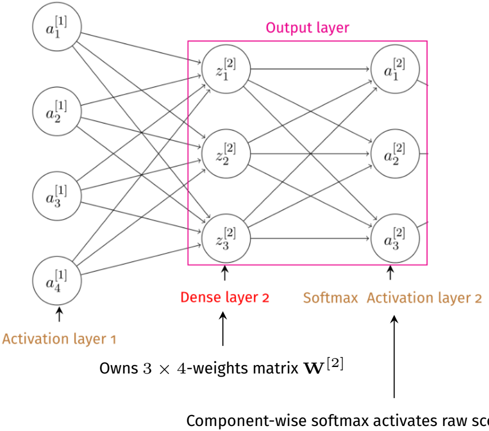
Component-wise softmax activates raw scores vector $\mathbf{z}^{[2]}$

- Weights matrix $\mathbf{W}^{[2]}$ has shape

$$\underbrace{3}_{\text{\# output raw scores in dense layer 2}} \times \underbrace{4}_{\text{\# inputs in activation layer 1}}$$

Single hidden layer neural network: notation – continued

Focus on activation layer 1 (4 nodes) and the output layer (3 nodes):



- Weights matrix $\mathbf{W}^{[2]}$ has shape

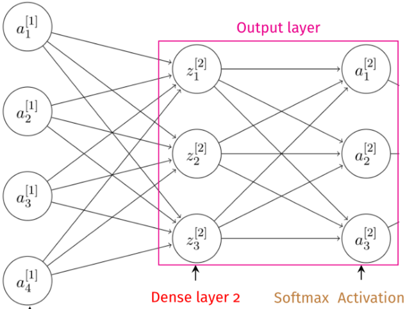
$$\underbrace{3}_{\text{\# output raw scores in dense layer 2}} \times \underbrace{4}_{\text{\# inputs in activation layer 1}}$$

- Raw scores vector for dense layer 2 is

$$\mathbf{z}^{[2]} = \begin{bmatrix} z_1^{[2]} \\ z_2^{[2]} \\ z_3^{[2]} \end{bmatrix}.$$

Single hidden layer neural network: notation – continued

Focus on activation layer 1 (4 nodes) and the output layer (3 nodes):



- Weights matrix $\mathbf{W}^{[2]}$ has shape

$$\underbrace{3}_{\#\text{ output raw scores in dense layer 2}} \times \underbrace{4}_{\#\text{ inputs in activation layer 1}}$$

- Raw scores vector for dense layer 2 is

$$\mathbf{z}^{[2]} = \begin{bmatrix} z_1^{[2]} \\ z_2^{[2]} \\ z_3^{[2]} \end{bmatrix}.$$

- Activated scores for softmax activation layer 2 is

$$\mathbf{a}^{[2]} = \begin{bmatrix} a_1^{[2]} \\ a_2^{[2]} \\ a_3^{[2]} \end{bmatrix} = \text{softmax}\left(\mathbf{z}^{[2]}\right) = \text{softmax}\left(\begin{bmatrix} z_1^{[2]} \\ z_2^{[2]} \\ z_3^{[2]} \end{bmatrix}\right).$$

Activation functions: need and types

Activation functions: need and types

- Calculating raw scores at each dense layer can be related to weighting the features.

Activation functions: need and types

- Calculating raw scores at each dense layer can be related to weighting the features.
- Activating those raw scores can be related to how each neuron's (node's) raw score *fires* it so that relevant information is retained and pushed further by the neuron.

Activation functions: need and types

- Calculating raw scores at each dense layer can be related to weighting the features.
- Activating those raw scores can be related to how each neuron's (node's) raw score *fires* it so that relevant information is retained and pushed further by the neuron.
- A simple activation function is $g(z) = z$ called the identity or linear activation function;

Activation functions: need and types

- Calculating raw scores at each dense layer can be related to weighting the features.
- Activating those raw scores can be related to how each neuron's (node's) raw score *fires* it so that relevant information is retained and pushed further by the neuron.
- A simple activation function is $g(z) = z$ called the identity or linear activation function; that is, it simply pushes the input forward.

Activation functions: need and types

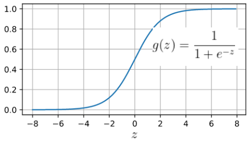
- Calculating raw scores at each dense layer can be related to weighting the features.
- Activating those raw scores can be related to how each neuron's (node's) raw score *fires* it so that relevant information is retained and pushed further by the neuron.
- A simple activation function is $g(z) = z$ called the identity or linear activation function; that is, it simply pushes the input forward.
- However, the identity activation function does not lead to *nonlinear* learning that can capture potential nonlinear relationship between the input and output.

Activation functions: need and types

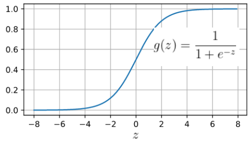
- Calculating raw scores at each dense layer can be related to weighting the features.
- Activating those raw scores can be related to how each neuron's (node's) raw score *fires* it so that relevant information is retained and pushed further by the neuron.
- A simple activation function is $g(z) = z$ called the identity or linear activation function; that is, it simply pushes the input forward.
- However, the identity activation function does not lead to *nonlinear* learning that can capture potential nonlinear relationship between the input and output.
- *Nonlinear learning* is achieved using nonlinear activation functions.

Activation functions: need and types – continued

Activation functions: need and types – continued

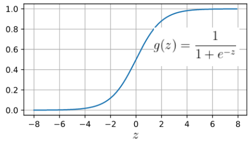


Activation functions: need and types – continued



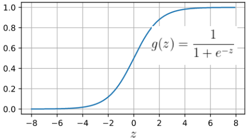
- Sigmoid activation function.

Activation functions: need and types – continued



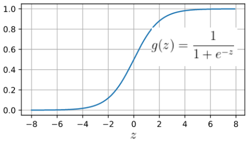
- Sigmoid activation function.
- Activated output between 0 and 1.

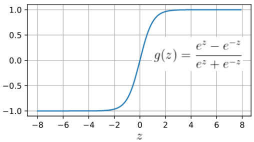
Activation functions: need and types – continued



- Sigmoid activation function.
- Activated output between 0 and 1.
- Almost linear behavior for small inputs around 0.

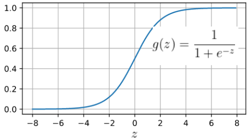
Activation functions: need and types – continued



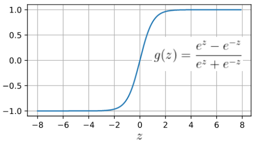


- Sigmoid activation function.
- Activated output between 0 and 1.
- Almost linear behavior for small inputs around 0.

Activation functions: need and types – continued

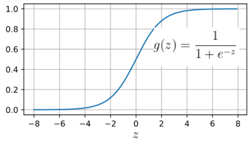


- Sigmoid activation function.
- Activated output between 0 and 1.
- Almost linear behavior for small inputs around 0.

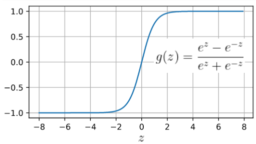


- tanh (hyperbolic tangent) activation function.

Activation functions: need and types – continued

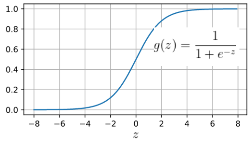


- Sigmoid activation function.
- Activated output between 0 and 1.
- Almost linear behavior for small inputs around 0.

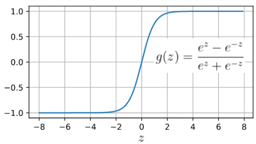


- tanh (hyperbolic tangent) activation function.
- Activated output between -1 and 1 (centered around 0).

Activation functions: need and types – continued

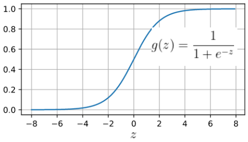


- Sigmoid activation function.
- Activated output between 0 and 1.
- Almost linear behavior for small inputs around 0.

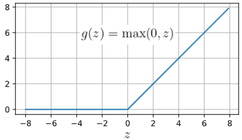


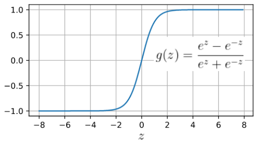
- tanh (hyperbolic tangent) activation function.
- Activated output between -1 and 1 (centered around 0).
- Almost linear behavior for small inputs around 0.

Activation functions: need and types – continued



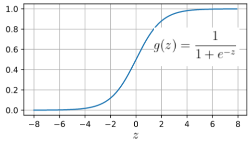
- Sigmoid activation function.
- Activated output between 0 and 1.
- Almost linear behavior for small inputs around 0.



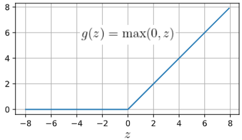


- tanh (hyperbolic tangent) activation function.
- Activated output between -1 and 1 (centered around 0).
- Almost linear behavior for small inputs around 0.

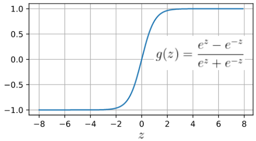
Activation functions: need and types – continued



- Sigmoid activation function.
- Activated output between 0 and 1.
- Almost linear behavior for small inputs around 0.

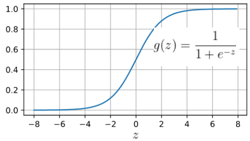


- ReLU (rectified linear unit) activation function.

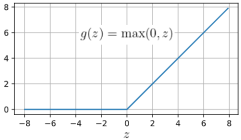


- tanh (hyperbolic tangent) activation function.
- Activated output between -1 and 1 (centered around 0).
- Almost linear behavior for small inputs around 0.

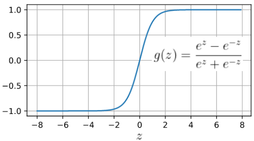
Activation functions: need and types – continued



- Sigmoid activation function.
- Activated output between 0 and 1.
- Almost linear behavior for small inputs around 0.

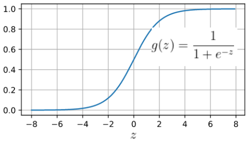


- ReLU (rectified linear unit) activation function.
- Negative inputs clipped to 0, non-negative ones transmitted as such.

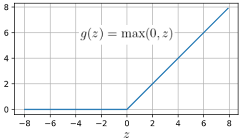


- tanh (hyperbolic tangent) activation function.
- Activated output between -1 and 1 (centered around 0).
- Almost linear behavior for small inputs around 0.

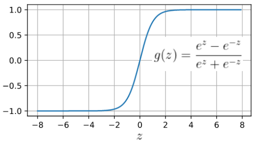
Activation functions: need and types – continued



- Sigmoid activation function.
- Activated output between 0 and 1.
- Almost linear behavior for small inputs around 0.

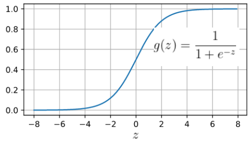


- ReLU (rectified linear unit) activation function.
- Negative inputs clipped to 0, non-negative ones transmitted as such.
- Simple and effective.

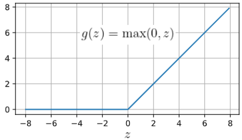


- tanh (hyperbolic tangent) activation function.
- Activated output between -1 and 1 (centered around 0).
- Almost linear behavior for small inputs around 0.

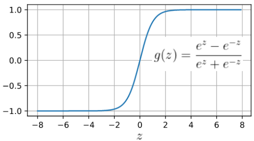
Activation functions: need and types – continued



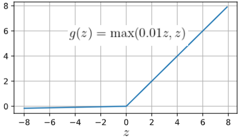
- Sigmoid activation function.
- Activated output between 0 and 1.
- Almost linear behavior for small inputs around 0.



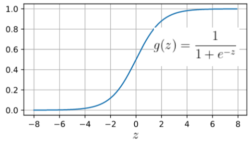
- ReLU (rectified linear unit) activation function.
- Negative inputs clipped to 0, non-negative ones transmitted as such.
- Simple and effective.



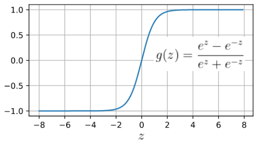
- tanh (hyperbolic tangent) activation function.
- Activated output between -1 and 1 (centered around 0).
- Almost linear behavior for small inputs around 0.



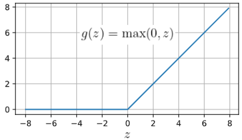
Activation functions: need and types – continued



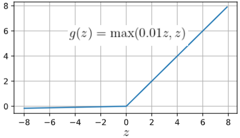
- Sigmoid activation function.
- Activated output between 0 and 1.
- Almost linear behavior for small inputs around 0.



- tanh (hyperbolic tangent) activation function.
- Activated output between -1 and 1 (centered around 0).
- Almost linear behavior for small inputs around 0.

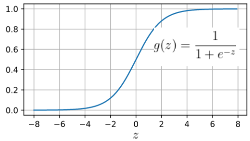


- ReLU (rectified linear unit) activation function.
- Negative inputs clipped to 0, non-negative ones transmitted as such.
- Simple and effective.

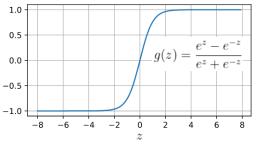


- Leaky ReLU (leaky rectified linear unit) activation function.

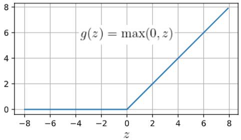
Activation functions: need and types – continued



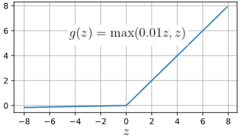
- Sigmoid activation function.
- Activated output between 0 and 1.
- Almost linear behavior for small inputs around 0.



- tanh (hyperbolic tangent) activation function.
- Activated output between -1 and 1 (centered around 0).
- Almost linear behavior for small inputs around 0.

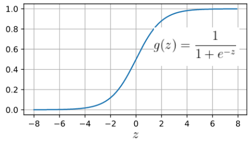


- ReLU (rectified linear unit) activation function.
- Negative inputs clipped to 0, non-negative ones transmitted as such.
- Simple and effective.

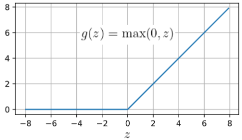


- Leaky ReLU (leaky rectified linear unit) activation function.
- Negative inputs transmitted with a small multiplicative factor while non-negative ones transmitted as such just like ReLU.

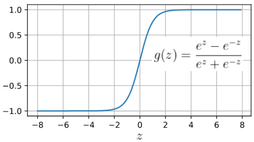
Activation functions: need and types – continued



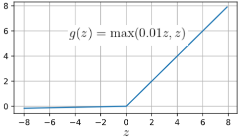
- Sigmoid activation function.
- Activated output between 0 and 1.
- Almost linear behavior for small inputs around 0.



- ReLU (rectified linear unit) activation function.
- Negative inputs clipped to 0, non-negative ones transmitted as such.
- Simple and effective.



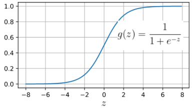
- tanh (hyperbolic tangent) activation function.
- Activated output between -1 and 1 (centered around 0).
- Almost linear behavior for small inputs around 0.



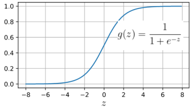
- Leaky ReLU (leaky rectified linear unit) activation function.
- Negative inputs transmitted with a small multiplicative factor while non-negative ones transmitted as such just like ReLU.
- Just like ReLU, simple and effective.

Gradients of activation functions

Gradients of activation functions

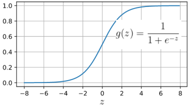


Gradients of activation functions



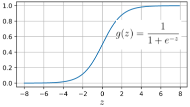
- Sigmoid activation function.

Gradients of activation functions



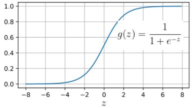
- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{\left(1+e^{-z}\right)^2} = g(z)(1 - g(z)).$

Gradients of activation functions



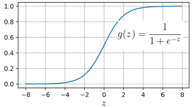
- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{\left(1+e^{-z}\right)^2} = g(z)(1-g(z))$.
- Gradient close to zero (*vanishes*) when input raw score z has large magnitude;

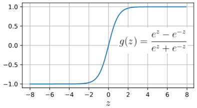
Gradients of activation functions



- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{\left(1+e^{-z}\right)^2} = g(z)(1 - g(z)).$
- Gradient close to zero (*vanishes*) when input raw score z has large magnitude; as $z \to \infty,\ g(z) \to 1$ and as $z \to -\infty,\ g(z) \to 0.$

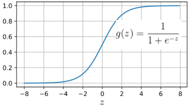
Gradients of activation functions



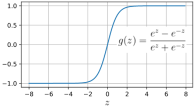


- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{\left(1+e^{-z}\right)^2} = g(z)(1 - g(z)).$
- Gradient close to zero (*vanishes*) when input raw score z has large magnitude; as $z \to \infty$, $g(z) \to 1$ and as $z \to -\infty$, $g(z) \to 0$.

Gradients of activation functions

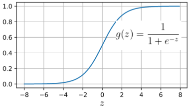


- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{\left(1+e^{-z}\right)^2} = g(z)(1-g(z))$.
- Gradient close to zero (*vanishes*) when input raw score z has large magnitude; as $z \to \infty,\ g(z) \to 1$ and as $z \to -\infty,\ g(z) \to 0$.

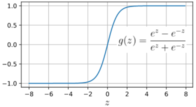


- tanh (hyperbolic tangent) activation function.

Gradients of activation functions

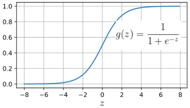


- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{\left(1+e^{-z}\right)^2} = g(z)(1-g(z))$.
- Gradient close to zero (*vanishes*) when input raw score z has large magnitude; as $z \to \infty,\ g(z) \to 1$ and as $z \to -\infty,\ g(z) \to 0$.

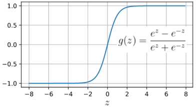


- tanh (hyperbolic tangent) activation function.
- $\nabla_z(g(z)) = 1 - \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)^2 = 1 - g(z)^2$.

Gradients of activation functions

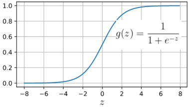


- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{\left(1+e^{-z}\right)^2} = g(z)(1-g(z))$.
- Gradient close to zero (*vanishes*) when input raw score z has large magnitude; as $z \to \infty,\ g(z) \to 1$ and as $z \to -\infty,\ g(z) \to 0$.

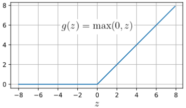


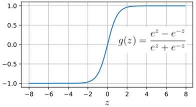
- tanh (hyperbolic tangent) activation function.
- $\nabla_z(g(z)) = 1 - \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)^2 = 1 - g(z)^2$.
- Has similar vanishing gradient behavior like ReLU for large magnitude input raw score.

Gradients of activation functions



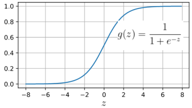
- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{\left(1+e^{-z}\right)^2} = g(z)(1-g(z))$.
- Gradient close to zero (*vanishes*) when input raw score z has large magnitude; as $z \to \infty$, $g(z) \to 1$ and as $z \to -\infty$, $g(z) \to 0$.



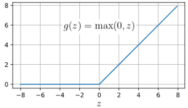


- tanh (hyperbolic tangent) activation function.
- $\nabla_z(g(z)) = 1 - \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)^2 = 1 - g(z)^2$.
- Has similar vanishing gradient behavior like ReLU for large magnitude input raw score.

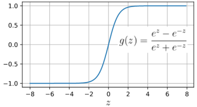
Gradients of activation functions



- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{\left(1+e^{-z}\right)^2} = g(z)(1-g(z))$.
- Gradient close to zero (*vanishes*) when input raw score z has large magnitude; as $z \to \infty,\ g(z) \to 1$ and as $z \to -\infty,\ g(z) \to 0$.

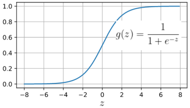


- ReLU (rectified linear unit) activation function.

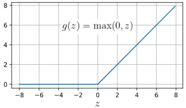


- tanh (hyperbolic tangent) activation function.
- $\nabla_z(g(z)) = 1 - \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)^2 = 1 - g(z)^2$.
- Has similar vanishing gradient behavior like ReLU for large magnitude input raw score.

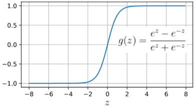
Gradients of activation functions



- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{\left(1+e^{-z}\right)^2} = g(z)(1-g(z))$.
- Gradient close to zero (*vanishes*) when input raw score z has large magnitude; as $z \to \infty$, $g(z) \to 1$ and as $z \to -\infty$, $g(z) \to 0$.

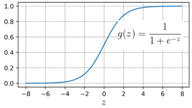


- ReLU (rectified linear unit) activation function.
- $\nabla_z(g(z)) = \begin{cases} 0 & \text{if } z < 0, \\ 1 & \text{if } z > 0. \end{cases}$

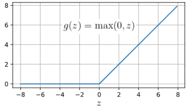


- tanh (hyperbolic tangent) activation function.
- $\nabla_z(g(z)) = 1 - \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)^2 = 1 - g(z)^2$.
- Has similar vanishing gradient behavior like ReLU for large magnitude input raw score.

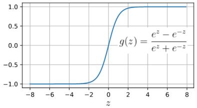
Gradients of activation functions



- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{\left(1+e^{-z}\right)^2} = g(z)(1-g(z))$.
- Gradient close to zero (*vanishes*) when input raw score z has large magnitude; as $z \to \infty,\ g(z) \to 1$ and as $z \to -\infty,\ g(z) \to 0$.

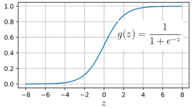


- ReLU (rectified linear unit) activation function.
- $\nabla_z(g(z)) = \begin{cases} 0 & \text{if } z < 0, \\ 1 & \text{if } z > 0. \end{cases}$
- Gradient vanishes for negative input raw score and is undefined at $z = 0$.

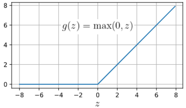


- tanh (hyperbolic tangent) activation function.
- $\nabla_z(g(z)) = 1 - \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)^2 = 1 - g(z)^2$.
- Has similar vanishing gradient behavior like ReLU for large magnitude input raw score.

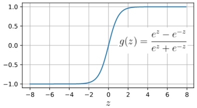
Gradients of activation functions



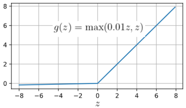
- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{\left(1+e^{-z}\right)^2} = g(z)(1-g(z))$.
- Gradient close to zero (*vanishes*) when input raw score z has large magnitude; as $z \to \infty,\ g(z) \to 1$ and as $z \to -\infty,\ g(z) \to 0$.



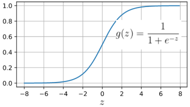
- ReLU (rectified linear unit) activation function.
- $\nabla_z(g(z)) = \begin{cases} 0 & \text{if } z < 0, \\ 1 & \text{if } z > 0. \end{cases}$
- Gradient vanishes for negative input raw score and is undefined at $z = 0$.



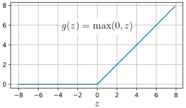
- tanh (hyperbolic tangent) activation function.
- $\nabla_z(g(z)) = 1 - \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)^2 = 1 - g(z)^2$.
- Has similar vanishing gradient behavior like ReLU for large magnitude input raw score.



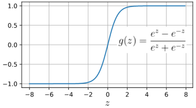
Gradients of activation functions



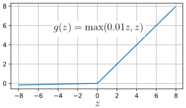
- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{\left(1+e^{-z}\right)^2} = g(z)(1-g(z))$.
- Gradient close to zero (*vanishes*) when input raw score z has large magnitude; as $z \to \infty,\ g(z) \to 1$ and as $z \to -\infty,\ g(z) \to 0$.



- ReLU (rectified linear unit) activation function.
- $\nabla_z(g(z)) = \begin{cases} 0 & \text{if } z < 0, \\ 1 & \text{if } z > 0. \end{cases}$
- Gradient vanishes for negative input raw score and is undefined at $z = 0$.

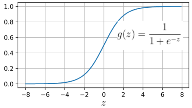


- tanh (hyperbolic tangent) activation function.
- $\nabla_z(g(z)) = 1 - \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)^2 = 1 - g(z)^2$.
- Has similar vanishing gradient behavior like ReLU for large magnitude input raw score.

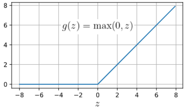


- Leaky ReLU (leaky rectified linear unit) activation function.

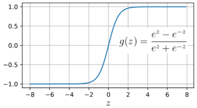
Gradients of activation functions



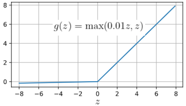
- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{\left(1+e^{-z}\right)^2} = g(z)(1-g(z))$.
- Gradient close to zero (*vanishes*) when input raw score z has large magnitude; as $z \to \infty,\ g(z) \to 1$ and as $z \to -\infty,\ g(z) \to 0$.



- ReLU (rectified linear unit) activation function.
- $\nabla_z(g(z)) = \begin{cases} 0 & \text{if } z < 0, \\ 1 & \text{if } z > 0. \end{cases}$
- Gradient vanishes for negative input raw score and is undefined at $z = 0$.

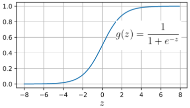


- tanh (hyperbolic tangent) activation function.
- $\nabla_z(g(z)) = 1 - \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)^2 = 1 - g(z)^2$.
- Has similar vanishing gradient behavior like ReLU for large magnitude input raw score.

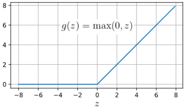


- Leaky ReLU (leaky rectified linear unit) activation function.
- $\nabla_z(g(z)) = \begin{cases} 0.01 & \text{if } z < 0, \\ 1 & \text{if } z > 0. \end{cases}$

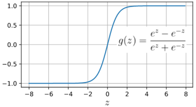
Gradients of activation functions



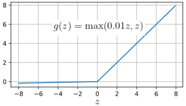
- Sigmoid activation function.
- $\nabla_z(g(z)) = \frac{1}{(1+e^{-z})^2} = g(z)(1-g(z))$.
- Gradient close to zero (*vanishes*) when input raw score z has large magnitude; as $z \to \infty,\ g(z) \to 1$ and as $z \to -\infty,\ g(z) \to 0$.



- ReLU (rectified linear unit) activation function.
- $\nabla_z(g(z)) = \begin{cases} 0 & \text{if } z < 0, \\ 1 & \text{if } z > 0. \end{cases}$
- Gradient vanishes for negative input raw score and is undefined at $z = 0$.



- tanh (hyperbolic tangent) activation function.
- $\nabla_z(g(z)) = 1 - \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)^2 = 1 - g(z)^2$.
- Has similar vanishing gradient behavior like ReLU for large magnitude input raw score.



- Leaky ReLU (leaky rectified linear unit) activation function.
- $\nabla_z(g(z)) = \begin{cases} 0.01 & \text{if } z < 0, \\ 1 & \text{if } z > 0. \end{cases}$
- Gradient does not vanish for negative input raw score but is still undefined at $z = 0$.

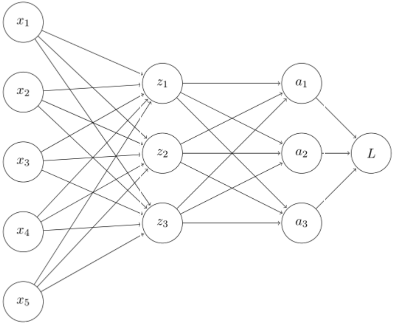
Zero hidden layer neural network (softmax classifier) forward propagation revisited

Zero hidden layer neural network (softmax classifier) forward propagation revisited

Softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels (bias feature 1 ignored for clarity):

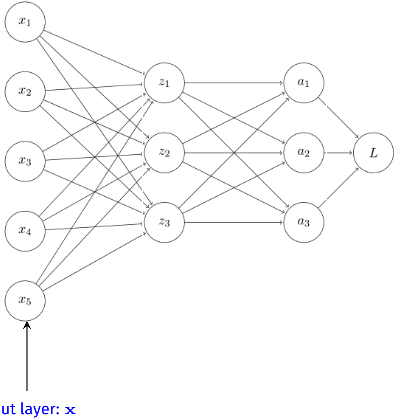
Zero hidden layer neural network (softmax classifier) forward propagation revisited

Softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels (bias feature 1 ignored for clarity):



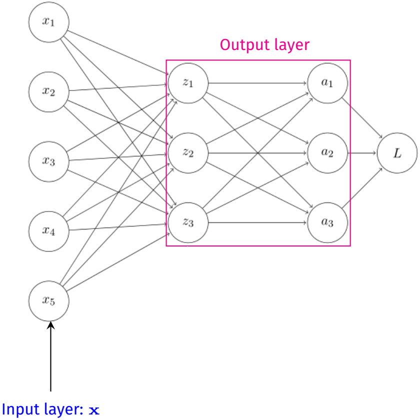
Zero hidden layer neural network (softmax classifier) forward propagation revisited

Softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels (bias feature 1 ignored for clarity):



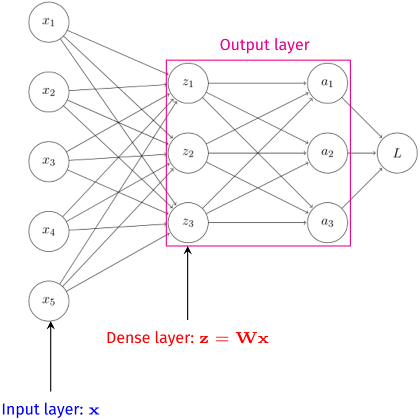
Zero hidden layer neural network (softmax classifier) forward propagation revisited

Softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels (bias feature 1 ignored for clarity):



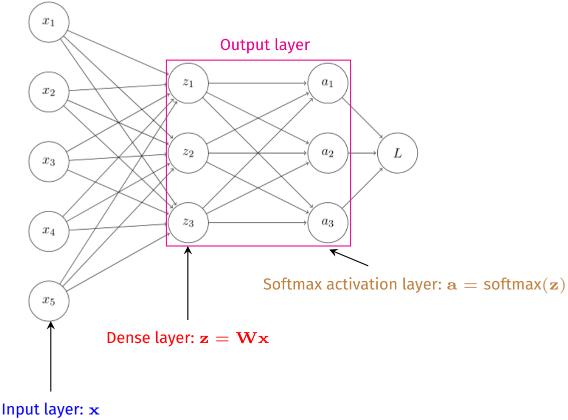
Zero hidden layer neural network (softmax classifier) forward propagation revisited

Softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels (bias feature 1 ignored for clarity):



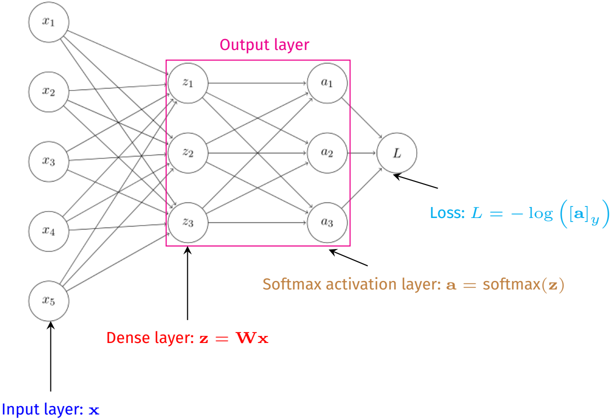
Zero hidden layer neural network (softmax classifier) forward propagation revisited

Softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels (bias feature 1 ignored for clarity):



Zero hidden layer neural network (softmax classifier) forward propagation revisited

Softmax classifier to a sample $\mathbf{x}$ with 5 features and correct output label y from 3 possible output labels (bias feature 1 ignored for clarity):



Categorical cross-entropy (CCE) loss for classification

Categorical cross-entropy (CCE) loss for classification

- The predicted probability vector that the sample belongs to each one of the output categories is given a new name:

Categorical cross-entropy (CCE) loss for classification

- The predicted probability vector that the sample belongs to each one of the output categories is given a new name:

$$\hat{\mathbf{y}} = \mathbf{a}.$$

Categorical cross-entropy (CCE) loss for classification

- The predicted probability vector that the sample belongs to each one of the output categories is given a new name:

$$\hat{\mathbf{y}} = \mathbf{a}.$$

- One-hot encoding the output label:

Categorical cross-entropy (CCE) loss for classification

- The predicted probability vector that the sample belongs to each one of the output categories is given a new name:

$$\hat{\mathbf{y}} = \mathbf{a}.$$

- One-hot encoding the output label: $y \rightarrow \mathbf{y} \left(\text{e.g.}\ 2 \rightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right).$

Categorical cross-entropy (CCE) loss for classification

- The predicted probability vector that the sample belongs to each one of the output categories is given a new name:

$$\hat{\mathbf{y}} = \mathbf{a}.$$

- One-hot encoding the output label: $y \to \mathbf{y} \left(\text{e.g. } 2 \to \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right).$
- This results in the following representation for the softmax loss for the sample which is also referred to as the *categorical cross-entropy* (CCE) loss:

$$L = -\log\left([\mathbf{a}]_y\right) \Rightarrow$$

Categorical cross-entropy (CCE) loss for classification

- The predicted probability vector that the sample belongs to each one of the output categories is given a new name:

$$\hat{\mathbf{y}} = \mathbf{a}.$$

- One-hot encoding the output label: $y \to \mathbf{y} \left(\text{e.g.}\, 2 \to \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right).$
- This results in the following representation for the softmax loss for the sample which is also referred to as the *categorical cross-entropy* (CCE) loss:

$$L = -\log\left([\mathbf{a}]_y\right) \Rightarrow L = L\left(\mathbf{y}, \hat{\mathbf{y}}\right) = \sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right).$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation

The gradient of the loss (of the sample) w.r.t. the weights can be derived using the following computation graph and chain rule:

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation

The gradient of the loss (of the sample) w.r.t. the weights can be derived using the following computation graph and chain rule:

Computation graph:

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation

The gradient of the loss (of the sample) w.r.t. the weights can be derived using the following computation graph and chain rule:

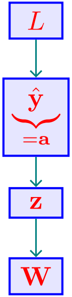
Computation graph:

$$\begin{tikzcd} \boxed{L} \arrow[d] \\ \boxed{\underbrace{\hat{\mathbf{y}}}_{=\mathbf{a}}} \arrow[d] \\ \boxed{\mathbf{z}} \arrow[d] \\ \boxed{\mathbf{W}} \end{tikzcd}$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation

The gradient of the loss (of the sample) w.r.t. the weights can be derived using the following computation graph and chain rule:

Computation graph:



Gradient calculation using chain rule:

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation

The gradient of the loss (of the sample) w.r.t. the weights can be derived using the following computation graph and chain rule:

Computation graph:

$$
\begin{tikzcd}
\boxed{L} \arrow[d] \\
\boxed{\underbrace{\hat{\mathbf{y}}}_{=\mathbf{a}}} \arrow[d] \\
\boxed{\mathbf{z}} \arrow[d] \\
\boxed{\mathbf{W}}
\end{tikzcd}
$$

Gradient calculation using chain rule:

$$\nabla_{\mathbf{w}}(L) =$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation

The gradient of the loss (of the sample) w.r.t. the weights can be derived using the following computation graph and chain rule:

<u>Computation graph:</u>

$$\begin{tikzcd} \boxed{L} \arrow[d] \\ \boxed{\underbrace{\hat{\mathbf{y}}}_{=\mathbf{a}}} \arrow[d] \\ \boxed{\mathbf{z}} \arrow[d] \\ \boxed{\mathbf{W}} \end{tikzcd}$$

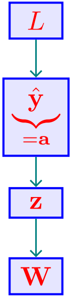
<u>Gradient calculation using chain rule:</u>

$$\nabla_{\mathbf{w}}(L) = \qquad \nabla_{\hat{\mathbf{y}}}(L)$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation

The gradient of the loss (of the sample) w.r.t. the weights can be derived using the following computation graph and chain rule:

<u>Computation graph:</u>



<u>Gradient calculation using chain rule:</u>

$$\nabla_{\mathbf{w}}(L) = \quad \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation

The gradient of the loss (of the sample) w.r.t. the weights can be derived using the following computation graph and chain rule:

Computation graph:

$$
\begin{tikzcd}
\boxed{L} \arrow[d] \\
\boxed{\underbrace{\hat{\mathbf{y}}}_{=\mathbf{a}}} \arrow[d] \\
\boxed{\mathbf{z}} \arrow[d] \\
\boxed{\mathbf{W}}
\end{tikzcd}
$$

Gradient calculation using chain rule:

$$\nabla_{\mathbf{w}}(L) = \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation

The gradient of the loss (of the sample) w.r.t. the weights can be derived using the following computation graph and chain rule:

Computation graph:

$$\begin{tikzcd}
\boxed{L} \arrow[d] \\
\boxed{\underbrace{\hat{\mathbf{y}}}_{=\mathbf{a}}} \arrow[d] \\
\boxed{\mathbf{z}} \arrow[d] \\
\boxed{\mathbf{W}}
\end{tikzcd}$$

Gradient calculation using chain rule:

$$\nabla_{\mathbf{w}}(L) = \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation

The gradient of the loss (of the sample) w.r.t. the weights can be derived using the following computation graph and chain rule:

Computation graph:

$$\begin{array}{c} L \\ \downarrow \\ \underbrace{\hat{\mathbf{y}}}_{=\mathbf{a}} \\ \downarrow \\ \mathbf{z} \\ \downarrow \\ \mathbf{W} \end{array}$$

Gradient calculation using chain rule:

$$\nabla_{\mathbf{w}}(L) = \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

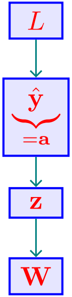
$$= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$\nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right).$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation

The gradient of the loss (of the sample) w.r.t. the weights can be derived using the following computation graph and chain rule:

<u>Computation graph:</u>



<u>Gradient calculation using chain rule:</u>

$$\nabla_{\mathbf{w}}(L) = \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

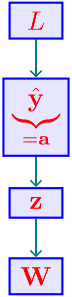
$$= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$\nabla_{\mathbf{z}}(\text{softmax}(\mathbf{z})) \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log(\hat{y}_k)\right).$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation

The gradient of the loss (of the sample) w.r.t. the weights can be derived using the following computation graph and chain rule:

<u>Computation graph:</u>



<u>Gradient calculation using chain rule:</u>

$$\nabla_{\mathbf{w}}(L) = \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$= \nabla_{\mathbf{W}}(\mathbf{W}\mathbf{x}) \times \nabla_{\mathbf{z}}(\text{softmax}(\mathbf{z})) \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log(\hat{y}_k)\right).$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation

The gradient of the loss (of the sample) w.r.t. the weights can be derived using the following computation graph and chain rule:

Computation graph:

$$\begin{tikzcd}[row sep=large]
\boxed{L} \arrow[d] \\
\boxed{\underbrace{\hat{\mathbf{y}}}_{=\mathbf{a}}} \arrow[d] \\
\boxed{\mathbf{z}} \arrow[d] \\
\boxed{\mathbf{W}}
\end{tikzcd}$$

Gradient calculation using chain rule:

$$\begin{aligned}
\nabla_{\mathbf{w}}(L) &= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L) \\
&= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\hat{\mathbf{y}}}(L) \\
&= \nabla_{\mathbf{W}}(\mathbf{W}\mathbf{x}) \times \nabla_{\mathbf{z}}(\text{softmax}(\mathbf{z})) \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log(\hat{y}_k)\right).
\end{aligned}$$

We will calculate the gradient term-by-term backwards.

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

We will now calculate the last gradient $\nabla_{\hat{\mathbf{y}}}(L) = \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)$ in

$$\nabla_{\mathbf{W}}\left(\mathbf{W}\mathbf{x}\right) \times \nabla_{\mathbf{z}}\left(\mathsf{softmax}(\mathbf{z})\right) \times \boxed{\nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)}:$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

We will now calculate the last gradient $\nabla_{\hat{\mathbf{y}}}(L) = \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)$ in

$$\nabla_{\mathbf{W}}\left(\mathbf{W}\mathbf{x}\right) \times \nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right) \times \boxed{\nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)}:$$

$$\nabla_{\hat{\mathbf{y}}}(L) = \begin{bmatrix} \nabla_{\hat{y}_1}(L) \\ \nabla_{\hat{y}_2}(L) \\ \nabla_{\hat{y}_3}(L) \end{bmatrix}$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

We will now calculate the last gradient $\nabla_{\hat{\mathbf{y}}}(L) = \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)$ in

$$\nabla_{\mathbf{W}}\left(\mathbf{Wx}\right) \times \nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right) \times \boxed{\nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)}:$$

$$\nabla_{\hat{\mathbf{y}}}(L) = \begin{bmatrix} \nabla_{\hat{y}_1}(L) \\ \nabla_{\hat{y}_2}(L) \\ \nabla_{\hat{y}_3}(L) \end{bmatrix} = \begin{bmatrix} \nabla_{\hat{y}_1}\left(-y_1 \log\left(\hat{y}_1\right) - y_2 \log\left(\hat{y}_2\right) - y_3 \log\left(\hat{y}_3\right)\right) \\ \nabla_{\hat{y}_2}\left(-y_1 \log\left(\hat{y}_1\right) - y_2 \log\left(\hat{y}_2\right) - y_3 \log\left(\hat{y}_3\right)\right) \\ \nabla_{\hat{y}_3}\left(-y_1 \log\left(\hat{y}_1\right) - y_2 \log\left(\hat{y}_2\right) - y_3 \log\left(\hat{y}_3\right)\right) \end{bmatrix}$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

We will now calculate the last gradient $\nabla_{\hat{\mathbf{y}}}(L) = \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)$ in

$$\nabla_{\mathbf{W}}\left(\mathbf{Wx}\right) \times \nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right) \times \boxed{\nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)}:$$

$$\nabla_{\hat{\mathbf{y}}}(L) = \begin{bmatrix} \nabla_{\hat{y}_1}(L) \\ \nabla_{\hat{y}_2}(L) \\ \nabla_{\hat{y}_3}(L) \end{bmatrix} = \begin{bmatrix} \nabla_{\hat{y}_1}\left(-y_1 \log\left(\hat{y}_1\right) - y_2 \log\left(\hat{y}_2\right) - y_3 \log\left(\hat{y}_3\right)\right) \\ \nabla_{\hat{y}_2}\left(-y_1 \log\left(\hat{y}_1\right) - y_2 \log\left(\hat{y}_2\right) - y_3 \log\left(\hat{y}_3\right)\right) \\ \nabla_{\hat{y}_3}\left(-y_1 \log\left(\hat{y}_1\right) - y_2 \log\left(\hat{y}_2\right) - y_3 \log\left(\hat{y}_3\right)\right) \end{bmatrix} = \begin{bmatrix} -y_1/\hat{y}_1 \\ -y_2/\hat{y}_2 \\ -y_3/\hat{y}_3 \end{bmatrix}.$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

We will now calculate the middle gradient $\nabla_{\mathbf{z}}(\mathbf{a}) = \nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right)$ in $\nabla_{\mathbf{W}}\left(\mathbf{W}\mathbf{x}\right) \times \boxed{\nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right)} \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)$:

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

We will now calculate the middle gradient $\nabla_{\mathbf{z}}(\mathbf{a}) = \nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right)$ in $\nabla_{\mathbf{W}}\left(\mathbf{W}\mathbf{x}\right) \times \boxed{\nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right)} \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)$:

$$\nabla_{\mathbf{z}}(\mathbf{a}) = \nabla_{\mathbf{z}}\left(\begin{bmatrix} \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}} \end{bmatrix}\right)$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

We will now calculate the middle gradient $\nabla_{\mathbf{z}}(\mathbf{a}) = \nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right)$ in $\nabla_{\mathbf{W}}\left(\mathbf{W}\mathbf{x}\right) \times \boxed{\nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right)} \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)$:

$$\nabla_{\mathbf{z}}(\mathbf{a}) = \nabla_{\mathbf{z}}\left(\begin{bmatrix} \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}} \end{bmatrix}\right) = \left[\nabla_{\mathbf{z}}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \quad \nabla_{\mathbf{z}}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \quad \nabla_{\mathbf{z}}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right)\right]$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

We will now calculate the middle gradient $\nabla_{\mathbf{z}}(\mathbf{a}) = \nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right)$ in $\nabla_{\mathbf{W}}\left(\mathbf{W}\mathbf{x}\right) \times \boxed{\nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right)} \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)$:

$$\nabla_{\mathbf{z}}(\mathbf{a}) = \nabla_{\mathbf{z}}\left(\begin{bmatrix} \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}} \end{bmatrix}\right) = \begin{bmatrix} \nabla_{\mathbf{z}}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{\mathbf{z}}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{\mathbf{z}}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \end{bmatrix}$$

$$= \begin{bmatrix} \nabla_{z_1}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_1}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_1}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \\ \nabla_{z_2}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_2}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_2}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \\ \nabla_{z_3}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_3}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_3}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \end{bmatrix}$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

We will now calculate the middle gradient $\nabla_{\mathbf{z}}(\mathbf{a}) = \nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right)$ in

$$\nabla_{\mathbf{W}}\left(\mathbf{W}\mathbf{x}\right) \times \boxed{\nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right)} \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right):$$

$$\nabla_{\mathbf{z}}(\mathbf{a}) = \nabla_{\mathbf{z}}\left(\begin{bmatrix} \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}} \end{bmatrix}\right) = \begin{bmatrix} \nabla_{\mathbf{z}}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{\mathbf{z}}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{\mathbf{z}}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \end{bmatrix}$$

$$= \begin{bmatrix} \nabla_{z_1}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_1}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_1}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \\ \nabla_{z_2}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_2}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_2}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \\ \nabla_{z_3}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_3}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_3}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \end{bmatrix}$$

$$\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}} - \left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right)^2 = a_1 - a_1^2 = a_1(1-a_1)$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

We will now calculate the middle gradient $\nabla_{\mathbf{z}}(\mathbf{a}) = \nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right)$ in $\nabla_{\mathbf{W}}\left(\mathbf{W}\mathbf{x}\right) \times \boxed{\nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right)} \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)$:

$$\nabla_{\mathbf{z}}(\mathbf{a}) = \nabla_{\mathbf{z}}\left(\begin{bmatrix} \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}} \end{bmatrix}\right) = \begin{bmatrix} \nabla_{\mathbf{z}}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{\mathbf{z}}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{\mathbf{z}}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \end{bmatrix}$$

$$= \begin{bmatrix} \nabla_{z_1}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_1}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_1}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \\ \nabla_{z_2}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_2}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_2}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \\ \nabla_{z_3}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_3}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_3}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \end{bmatrix}$$

$$\frac{-e^{z_1}e^{z_2}}{(e^{z_1}+e^{z_2}+e^{z_3})^2} = -a_1 a_2$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

We will now calculate the middle gradient $\nabla_{\mathbf{z}}(\mathbf{a}) = \nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right)$ in $\nabla_{\mathbf{W}}\left(\mathbf{W}\mathbf{x}\right) \times \boxed{\nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right)} \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)$:

$$\nabla_{\mathbf{z}}(\mathbf{a}) = \nabla_{\mathbf{z}}\left(\begin{bmatrix} \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}} \\ \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}} \end{bmatrix}\right) = \begin{bmatrix} \nabla_{\mathbf{z}}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{\mathbf{z}}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{\mathbf{z}}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \end{bmatrix}$$

$$= \begin{bmatrix} \nabla_{z_1}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_1}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_1}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \\ \nabla_{z_2}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_2}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_2}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \\ \nabla_{z_3}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_3}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) & \nabla_{z_3}\left(\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}\right) \end{bmatrix} = \begin{bmatrix} a_1(1-a_1) & -a_2 a_1 & -a_3 a_1 \\ -a_1 a_2 & a_2(1-a_2) & -a_3 a_2 \\ -a_1 a_3 & -a_2 a_3 & a_3(1-a_3) \end{bmatrix}.$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

We will now calculate the last gradient $\nabla_{\mathbf{W}}(\mathbf{W}\mathbf{x})$ in

$$\boxed{\nabla_{\mathbf{W}}\left(\mathbf{W}\mathbf{x}\right)} \times \nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right) \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right)$$ using the fact that $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^{\mathrm{T}} \\ \mathbf{w}_2^{\mathrm{T}} \\ \mathbf{w}_3^{\mathrm{T}} \end{bmatrix}$:

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

Gradient shape = shape of $\mathbf{W}$ $\times$ shape of $\mathbf{Wx}$ $= (3 \times 5) \times 3$ which is a 5×3-matrix repeating 3 times

We will now calculate the last gradient $\nabla_{\mathbf{W}}(\mathbf{Wx})$ in

$$\boxed{\nabla_{\mathbf{W}}(\mathbf{Wx})} \times \nabla_{\mathbf{z}}(\text{softmax}(\mathbf{z})) \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log(\hat{y}_k)\right)$$ using the fact that $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^{\mathrm{T}} \\ \mathbf{w}_2^{\mathrm{T}} \\ \mathbf{w}_3^{\mathrm{T}} \end{bmatrix}$:

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

Gradient shape = shape of $\mathbf{W}$ $\times$ shape of $\mathbf{Wx}$ $= (3\times 5)\times 3$ which is a 5×3-matrix repeating 3 times

We will now calculate the last gradient $\nabla_{\mathbf{W}}(\mathbf{Wx})$ in

$$\boxed{\nabla_{\mathbf{W}}\left(\mathbf{Wx}\right)} \times \nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right) \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right) \text{ using the fact that } \mathbf{W} = \begin{bmatrix}\mathbf{w}_1^{\mathrm{T}}\\ \mathbf{w}_2^{\mathrm{T}}\\ \mathbf{w}_3^{\mathrm{T}}\end{bmatrix}:$$

$$\nabla_{\mathbf{W}}(\mathbf{Wx}) \quad =$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

Gradient shape = shape of $\mathbf{W}$ $\times$ shape of $\mathbf{Wx} = (3 \times 5) \times 3$ which is a 5×3-matrix repeating 3 times

We will now calculate the last gradient $\nabla_{\mathbf{W}}(\mathbf{Wx})$ in

$$\boxed{\nabla_{\mathbf{W}}(\mathbf{Wx})} \times \nabla_{\mathbf{z}}(\text{softmax}(\mathbf{z})) \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log(\hat{y}_k)\right)$$ using the fact that $\mathbf{W} = \begin{bmatrix}\mathbf{w}_1^{\mathrm{T}}\\ \mathbf{w}_2^{\mathrm{T}}\\ \mathbf{w}_3^{\mathrm{T}}\end{bmatrix}$:

$$\nabla_{\mathbf{W}}(\mathbf{Wx}) = \nabla_{\mathbf{W}}\left(\begin{bmatrix}\mathbf{w}_1^{\mathrm{T}}\\ \mathbf{w}_2^{\mathrm{T}}\\ \mathbf{w}_3^{\mathrm{T}}\end{bmatrix}\mathbf{x}\right) =$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

Gradient shape = shape of $\mathbf{W}$ $\times$ shape of $\mathbf{Wx}$ $= (3 \times 5) \times 3$ which is a 5×3-matrix repeating 3 times

We will now calculate the last gradient $\nabla_{\mathbf{W}}(\mathbf{Wx})$ in

$$\boxed{\nabla_{\mathbf{W}}(\mathbf{Wx})} \times \nabla_{\mathbf{z}}(\text{softmax}(\mathbf{z})) \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log(\hat{y}_k)\right)$$ using the fact that $\mathbf{W} = \begin{bmatrix}\mathbf{w}_1^{\mathrm{T}} \\ \mathbf{w}_2^{\mathrm{T}} \\ \mathbf{w}_3^{\mathrm{T}}\end{bmatrix}$:

$$\nabla_{\mathbf{W}}(\mathbf{Wx}) = \nabla_{\mathbf{W}}\left(\begin{bmatrix}\mathbf{w}_1^{\mathrm{T}} \\ \mathbf{w}_2^{\mathrm{T}} \\ \mathbf{w}_3^{\mathrm{T}}\end{bmatrix}\mathbf{x}\right) = \nabla_{\mathbf{W}}\left(\begin{bmatrix}\mathbf{w}_1^{\mathrm{T}}\mathbf{x} \\ \mathbf{w}_2^{\mathrm{T}}\mathbf{x} \\ \mathbf{w}_3^{\mathrm{T}}\mathbf{x}\end{bmatrix}\right) =$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

Gradient shape = shape of $\mathbf{W}$ $\times$ shape of $\mathbf{Wx} = (3 \times 5) \times 3$ which is a 5×3-matrix repeating 3 times

We will now calculate the last gradient $\nabla_{\mathbf{W}}(\mathbf{Wx})$ in

$$\boxed{\nabla_{\mathbf{W}}\left(\mathbf{Wx}\right)} \times \nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right) \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right) \text{ using the fact that } \mathbf{W} = \begin{bmatrix} \mathbf{w}_1^{\mathrm{T}} \\ \mathbf{w}_2^{\mathrm{T}} \\ \mathbf{w}_3^{\mathrm{T}} \end{bmatrix}:$$

$$\nabla_{\mathbf{W}}(\mathbf{Wx}) = \nabla_{\mathbf{W}}\left(\begin{bmatrix} \mathbf{w}_1^{\mathrm{T}} \\ \mathbf{w}_2^{\mathrm{T}} \\ \mathbf{w}_3^{\mathrm{T}} \end{bmatrix}\mathbf{x}\right) = \nabla_{\mathbf{W}}\left(\begin{bmatrix} \mathbf{w}_1^{\mathrm{T}}\mathbf{x} \\ \mathbf{w}_2^{\mathrm{T}}\mathbf{x} \\ \mathbf{w}_3^{\mathrm{T}}\mathbf{x} \end{bmatrix}\right) = \begin{matrix} \begin{bmatrix} \nabla_{\mathbf{w}_1}\left(\mathbf{w}_1^{\mathrm{T}}\mathbf{x}\right) & \nabla_{\mathbf{w}_1}\left(\mathbf{w}_2^{\mathrm{T}}\mathbf{x}\right) & \nabla_{\mathbf{w}_1}\left(\mathbf{w}_3^{\mathrm{T}}\mathbf{x}\right) \end{bmatrix} \\ \begin{bmatrix} \nabla_{\mathbf{w}_2}\left(\mathbf{w}_1^{\mathrm{T}}\mathbf{x}\right) & \nabla_{\mathbf{w}_2}\left(\mathbf{w}_2^{\mathrm{T}}\mathbf{x}\right) & \nabla_{\mathbf{w}_2}\left(\mathbf{w}_3^{\mathrm{T}}\mathbf{x}\right) \end{bmatrix} \\ \begin{bmatrix} \nabla_{\mathbf{w}_3}\left(\mathbf{w}_1^{\mathrm{T}}\mathbf{x}\right) & \nabla_{\mathbf{w}_3}\left(\mathbf{w}_2^{\mathrm{T}}\mathbf{x}\right) & \nabla_{\mathbf{w}_3}\left(\mathbf{w}_3^{\mathrm{T}}\mathbf{x}\right) \end{bmatrix} \end{matrix}$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

Gradient shape = shape of $\mathbf{W}$ $\times$ shape of $\mathbf{Wx} = (3 \times 5) \times 3$ which is a 5×3-matrix repeating 3 times

We will now calculate the last gradient $\nabla_{\mathbf{W}}(\mathbf{Wx})$ in

$$\boxed{\nabla_{\mathbf{W}}\left(\mathbf{Wx}\right)} \times \nabla_{\mathbf{z}}\left(\text{softmax}(\mathbf{z})\right) \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log\left(\hat{y}_k\right)\right) \text{ using the fact that } \mathbf{W} = \begin{bmatrix} \mathbf{w}_1^{\mathrm{T}} \\ \mathbf{w}_2^{\mathrm{T}} \\ \mathbf{w}_3^{\mathrm{T}} \end{bmatrix}:$$

$$\nabla_{\mathbf{W}}(\mathbf{Wx}) = \nabla_{\mathbf{W}}\left(\begin{bmatrix} \mathbf{w}_1^{\mathrm{T}} \\ \mathbf{w}_2^{\mathrm{T}} \\ \mathbf{w}_3^{\mathrm{T}} \end{bmatrix}\mathbf{x}\right) = \nabla_{\mathbf{W}}\left(\begin{bmatrix} \mathbf{w}_1^{\mathrm{T}}\mathbf{x} \\ \mathbf{w}_2^{\mathrm{T}}\mathbf{x} \\ \mathbf{w}_3^{\mathrm{T}}\mathbf{x} \end{bmatrix}\right) = \begin{matrix} \nabla_{\mathbf{w}_1}(\mathbf{w}_1^{\mathrm{T}}\mathbf{x}) & \nabla_{\mathbf{w}_1}(\mathbf{w}_2^{\mathrm{T}}\mathbf{x}) & \nabla_{\mathbf{w}_1}(\mathbf{w}_3^{\mathrm{T}}\mathbf{x}) \\ \nabla_{\mathbf{w}_2}(\mathbf{w}_1^{\mathrm{T}}\mathbf{x}) & \nabla_{\mathbf{w}_2}(\mathbf{w}_2^{\mathrm{T}}\mathbf{x}) & \nabla_{\mathbf{w}_2}(\mathbf{w}_3^{\mathrm{T}}\mathbf{x}) \\ \nabla_{\mathbf{w}_3}(\mathbf{w}_1^{\mathrm{T}}\mathbf{x}) & \nabla_{\mathbf{w}_3}(\mathbf{w}_2^{\mathrm{T}}\mathbf{x}) & \nabla_{\mathbf{w}_3}(\mathbf{w}_3^{\mathrm{T}}\mathbf{x}) \end{matrix}$$

$$\nabla_{\mathbf{w}_i}\left(\mathbf{w}_j^{\mathrm{T}}\mathbf{x}\right) = \begin{cases} \mathbf{x} & \text{if } i = j, \\ \mathbf{0} & \text{if } i \neq j. \end{cases}$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

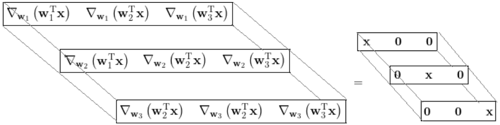
Gradient shape = shape of $\mathbf{W}$ $\times$ shape of $\mathbf{Wx}$ $= (3 \times 5) \times 3$ which is a 5×3-matrix repeating 3 times

We will now calculate the last gradient $\nabla_{\mathbf{W}}(\mathbf{Wx})$ in

$$\boxed{\nabla_{\mathbf{W}}(\mathbf{Wx})} \times \nabla_{\mathbf{z}}(\text{softmax}(\mathbf{z})) \times \nabla_{\hat{\mathbf{y}}}\left(\sum_{k=1}^{3} -y_k \log(\hat{y}_k)\right)$$

using the fact that $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^{\mathrm{T}} \\ \mathbf{w}_2^{\mathrm{T}} \\ \mathbf{w}_3^{\mathrm{T}} \end{bmatrix}$:

$$\nabla_{\mathbf{W}}(\mathbf{Wx}) = \nabla_{\mathbf{W}}\left(\begin{bmatrix} \mathbf{w}_1^{\mathrm{T}} \\ \mathbf{w}_2^{\mathrm{T}} \\ \mathbf{w}_3^{\mathrm{T}} \end{bmatrix}\mathbf{x}\right) = \nabla_{\mathbf{W}}\left(\begin{bmatrix} \mathbf{w}_1^{\mathrm{T}}\mathbf{x} \\ \mathbf{w}_2^{\mathrm{T}}\mathbf{x} \\ \mathbf{w}_3^{\mathrm{T}}\mathbf{x} \end{bmatrix}\right) =$$



$$\nabla_{\mathbf{w}_i}\left(\mathbf{w}_j^{\mathrm{T}}\mathbf{x}\right) = \begin{cases} \mathbf{x} & \text{if } i = j, \\ \mathbf{0} & \text{if } i \neq j. \end{cases}$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

The zero hidden layer (softmax classifier) gradient can be written as:

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

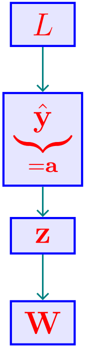
The zero hidden layer (softmax classifier) gradient can be written as:

Computation graph:

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

The zero hidden layer (softmax classifier) gradient can be written as:

Computation graph:



Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

The zero hidden layer (softmax classifier) gradient can be written as:

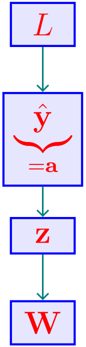
<u>Computation graph:</u> <u>Gradient calculation using chain rule:</u>

$$
\begin{tikzcd}
\boxed{L} \arrow[d] \\
\boxed{\underbrace{\hat{\mathbf{y}}}_{=\mathbf{a}}} \arrow[d] \\
\boxed{\mathbf{z}} \arrow[d] \\
\boxed{\mathbf{W}}
\end{tikzcd}
$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

The zero hidden layer (softmax classifier) gradient can be written as:

Computation graph:



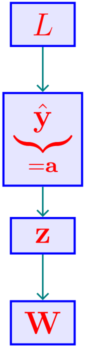
Gradient calculation using chain rule:

$$\nabla_{\mathbf{w}}(L) =$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

The zero hidden layer (softmax classifier) gradient can be written as:

Computation graph:



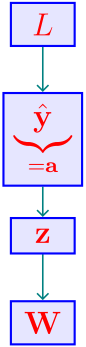
Gradient calculation using chain rule:

$$\nabla_{\mathbf{w}}(L) = \qquad \nabla_{\hat{\mathbf{y}}}(L)$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

The zero hidden layer (softmax classifier) gradient can be written as:

Computation graph:



Gradient calculation using chain rule:

$$\nabla_{\mathbf{W}}(L) = \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

The zero hidden layer (softmax classifier) gradient can be written as:

Computation graph:

$$\begin{tikzcd} \boxed{L} \arrow[d] \\ \boxed{\underbrace{\hat{\mathbf{y}}}_{=\mathbf{a}}} \arrow[d] \\ \boxed{\mathbf{z}} \arrow[d] \\ \boxed{\mathbf{W}} \end{tikzcd}$$

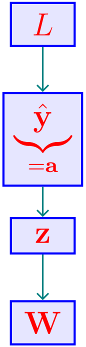
Gradient calculation using chain rule:

$$\nabla_{\mathbf{w}}(L) = \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

The zero hidden layer (softmax classifier) gradient can be written as:

<u>Computation graph:</u>



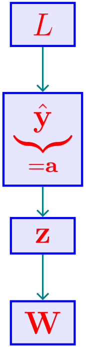
<u>Gradient calculation using chain rule:</u>

$$
\begin{aligned}
\nabla_{\mathbf{w}}(L) &= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L) \\
&= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\hat{\mathbf{y}}}(L)
\end{aligned}
$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

The zero hidden layer (softmax classifier) gradient can be written as:

Computation graph: Gradient calculation using chain rule:



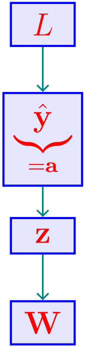
$$\begin{aligned}\nabla_{\mathbf{W}}(L) &= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L) \\ &= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\hat{\mathbf{y}}}(L)\end{aligned}$$

$$\begin{bmatrix} -y_1/\hat{y}_1 \\ -y_2/\hat{y}_2 \\ -y_3/\hat{y}_3 \end{bmatrix}$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

The zero hidden layer (softmax classifier) gradient can be written as:

Computation graph:



Gradient calculation using chain rule:

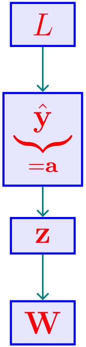
$$
\begin{aligned}
\nabla_{\mathbf{w}}(L) &= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L) \\
&= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\hat{\mathbf{y}}}(L)
\end{aligned}
$$

$$
\times \begin{bmatrix} a_1(1-a_1) & -a_2a_1 & -a_3a_1 \\ -a_1a_2 & a_2(1-a_2) & -a_3a_2 \\ -a_1a_3 & -a_2a_3 & a_3(1-a_3) \end{bmatrix} \times \begin{bmatrix} -y_1/\hat{y}_1 \\ -y_2/\hat{y}_2 \\ -y_3/\hat{y}_3 \end{bmatrix}
$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

The zero hidden layer (softmax classifier) gradient can be written as:

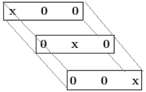
Computation graph:



Gradient calculation using chain rule:

$$\begin{aligned}\nabla_{\mathbf{w}}(L) &= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)\\ &= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\hat{\mathbf{y}}}(L)\end{aligned}$$

=

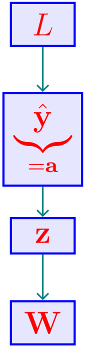


$$\times \begin{bmatrix} a_1(1-a_1) & -a_2a_1 & -a_3a_1 \\ -a_1a_2 & a_2(1-a_2) & -a_3a_2 \\ -a_1a_3 & -a_2a_3 & a_3(1-a_3) \end{bmatrix} \times \begin{bmatrix} -y_1/\hat{y}_1 \\ -y_2/\hat{y}_2 \\ -y_3/\hat{y}_3 \end{bmatrix}$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

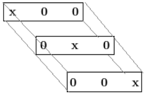
The zero hidden layer (softmax classifier) gradient can be written as:

Computation graph:



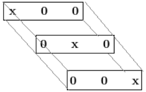
Gradient calculation using chain rule:

$$\nabla_{\mathbf{w}}(L) = \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$
$$= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\hat{\mathbf{y}}}(L)$$



$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} v_1\mathbf{x}^T \\ v_2\mathbf{x}^T \\ v_3\mathbf{x}^T \end{bmatrix} = \mathbf{v}\mathbf{x}^T$$

=

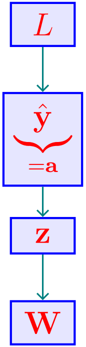


$$\times \begin{bmatrix} a_1(1-a_1) & -a_2a_1 & -a_3a_1 \\ -a_1a_2 & a_2(1-a_2) & -a_3a_2 \\ -a_1a_3 & -a_2a_3 & a_3(1-a_3) \end{bmatrix} \times \begin{bmatrix} -y_1/\hat{y}_1 \\ -y_2/\hat{y}_2 \\ -y_3/\hat{y}_3 \end{bmatrix}$$

Zero hidden layer neural network (softmax classifier) gradient calculation using backward propagation – continued

The zero hidden layer (softmax classifier) gradient can be written as:

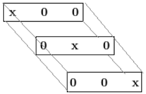
Computation graph:



Gradient calculation using chain rule:

$$
\begin{aligned}
\nabla_{\mathbf{w}}(L) &= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)\\
&= \nabla_{\mathbf{W}}(\mathbf{z}) \times \nabla_{\mathbf{z}}(\mathbf{a}) \times \nabla_{\hat{\mathbf{y}}}(L)
\end{aligned}
$$

=

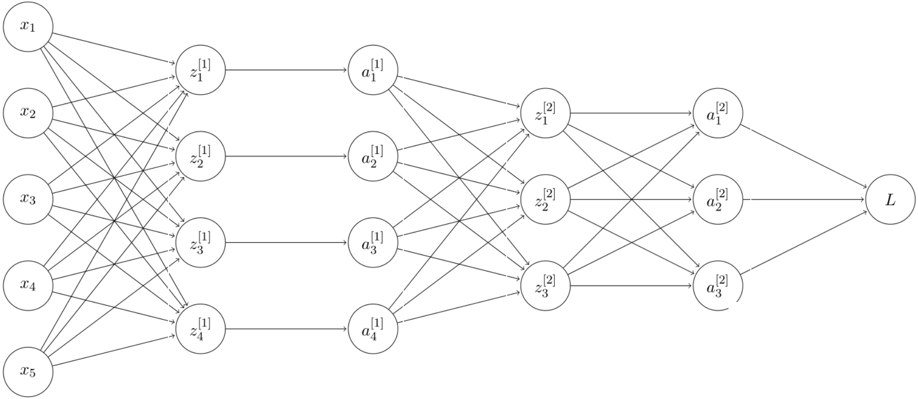


$$
\times \begin{bmatrix} a_1(1-a_1) & -a_2a_1 & -a_3a_1 \\ -a_1a_2 & a_2(1-a_2) & -a_3a_2 \\ -a_1a_3 & -a_2a_3 & a_3(1-a_3) \end{bmatrix} \times \begin{bmatrix} -y_1/\hat{y}_1 \\ -y_2/\hat{y}_2 \\ -y_3/\hat{y}_3 \end{bmatrix}
$$

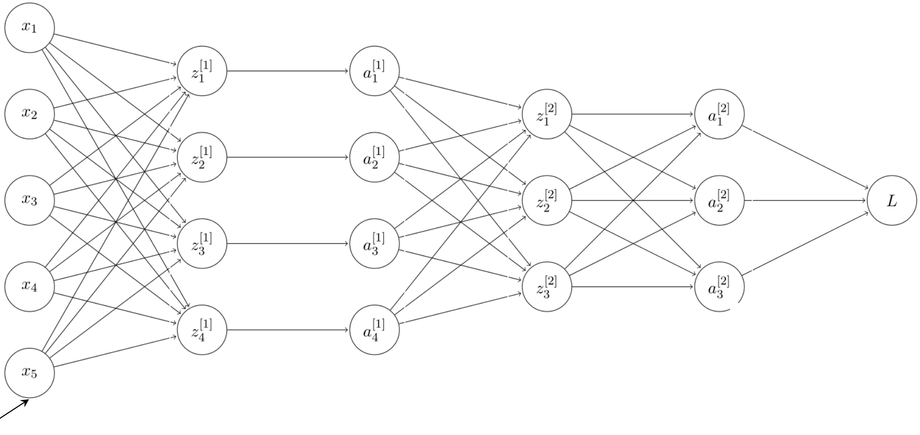
$$
= \begin{bmatrix} a_1(1-a_1) & -a_2a_1 & -a_3a_1 \\ -a_1a_2 & a_2(1-a_2) & -a_3a_2 \\ -a_1a_3 & -a_2a_3 & a_3(1-a_3) \end{bmatrix} \times \begin{bmatrix} -y_1/\hat{y}_1 \\ -y_2/\hat{y}_2 \\ -y_3/\hat{y}_3 \end{bmatrix} \mathbf{x}^{\mathrm{T}}.
$$

Single hidden layer neural network forward propagation

Single hidden layer neural network forward propagation

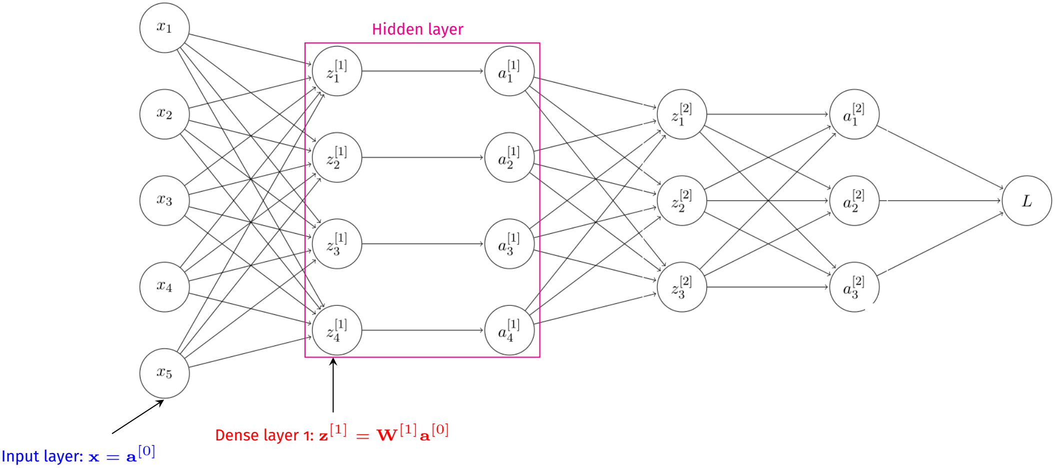


Single hidden layer neural network forward propagation

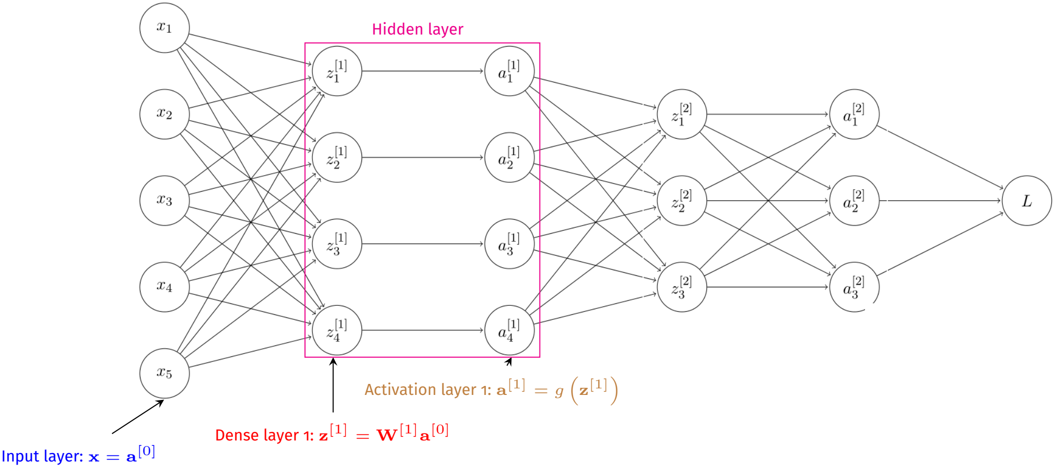


Input layer: $\mathbf{x} = \mathbf{a}^{[0]}$

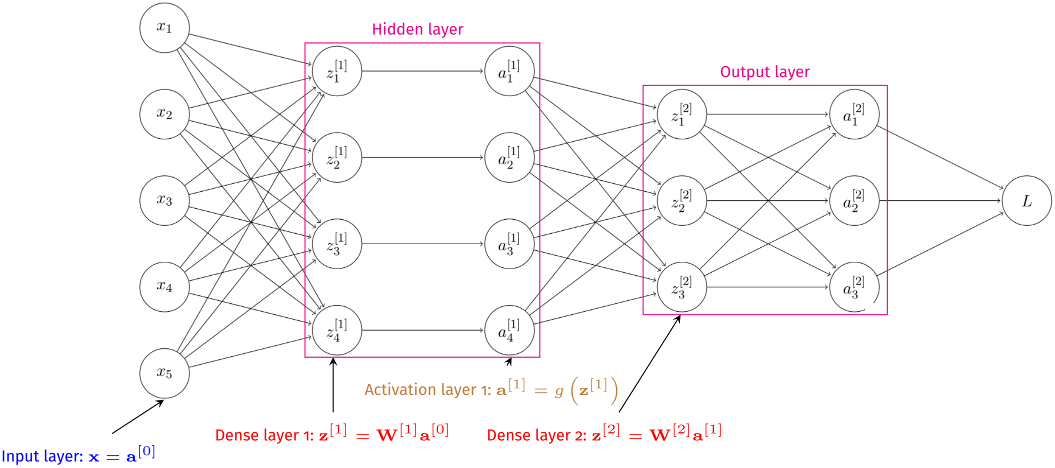
Single hidden layer neural network forward propagation



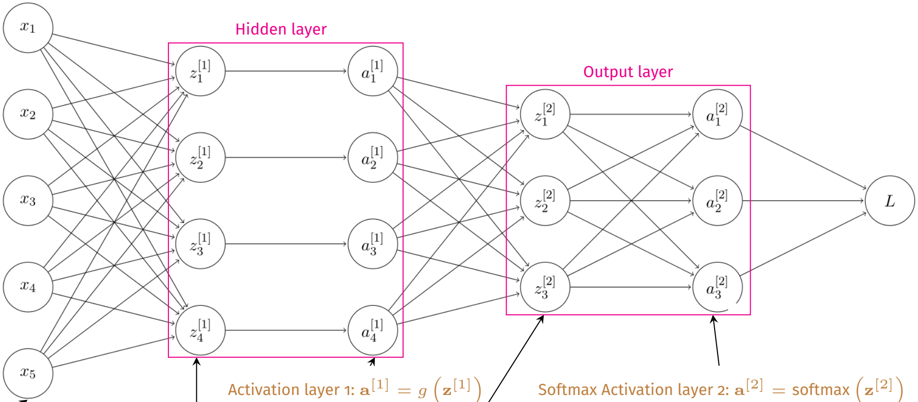
Single hidden layer neural network forward propagation



Single hidden layer neural network forward propagation



Single hidden layer neural network forward propagation



Activation layer 1: $\mathbf{a}^{[1]} = g\left(\mathbf{z}^{[1]}\right)$

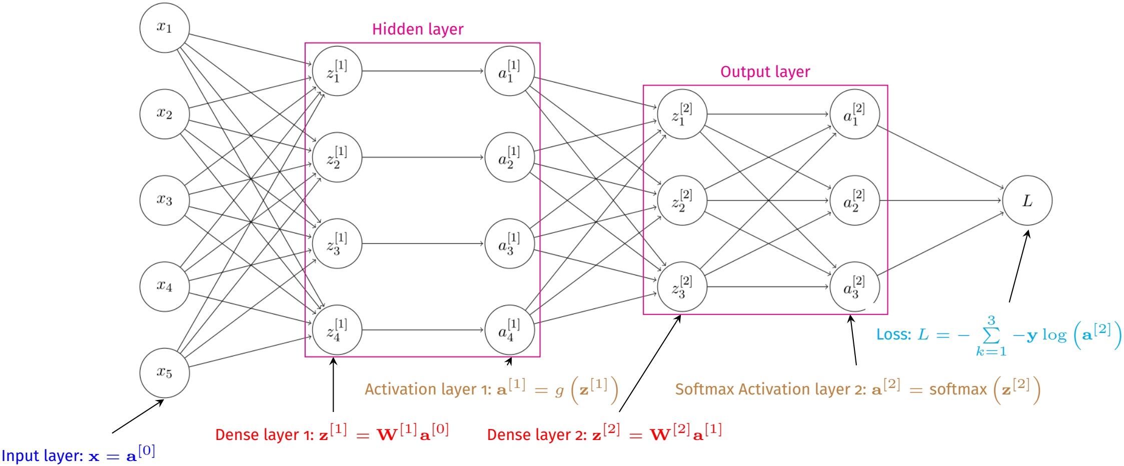
Softmax Activation layer 2: $\mathbf{a}^{[2]} = \text{softmax}\left(\mathbf{z}^{[2]}\right)$

Dense layer 1: $\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{a}^{[0]}$

Dense layer 2: $\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]}$

Input layer: $\mathbf{x} = \mathbf{a}^{[0]}$

Single hidden layer neural network forward propagation



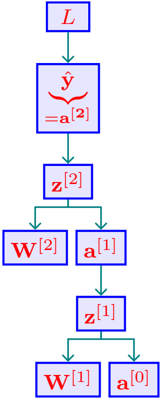
Single hidden layer neural network gradient calculation calculation using backward propagation

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph:

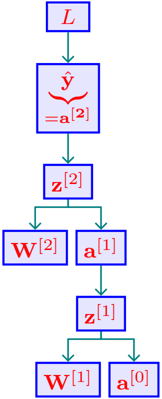
Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph:



Single hidden layer neural network gradient calculation calculation using backward propagation

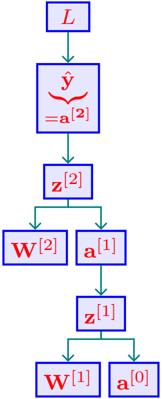
Computation graph:



Gradient calculation using chain rule:

Single hidden layer neural network gradient calculation calculation using backward propagation

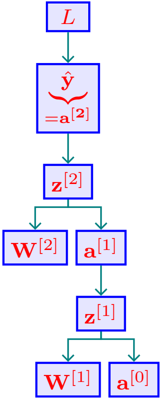
Computation graph: Gradient calculation using chain rule:



$$\nabla_{\mathbf{W}^{[2]}}(L) =$$

Single hidden layer neural network gradient calculation calculation using backward propagation

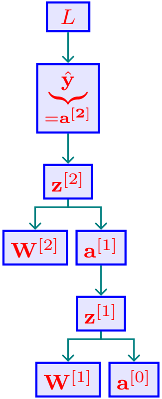
Computation graph: **Gradient calculation using chain rule:**



$$\nabla_{\mathbf{W}^{[2]}}(L) = \qquad \nabla_{\hat{\mathbf{y}}}(L)$$

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph:

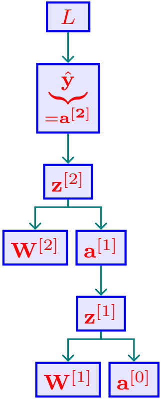


Gradient calculation using chain rule:

$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph:

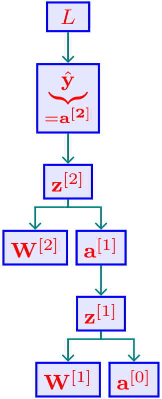


Gradient calculation using chain rule:

$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{W}^{[2]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph:



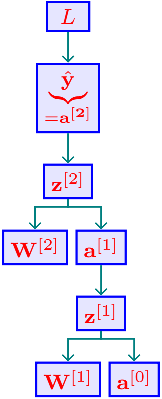
Gradient calculation using chain rule:

$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{W}^{[2]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$\nabla_{\mathbf{W}^{[1]}}(L) =$$

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph: Gradient calculation using chain rule:

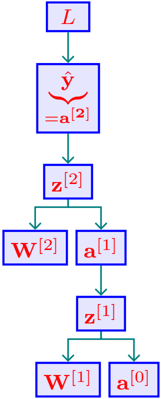


$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{W}^{[2]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$\nabla_{\mathbf{W}^{[1]}}(L) = \qquad \nabla_{\hat{\mathbf{y}}}(L)$$

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph:



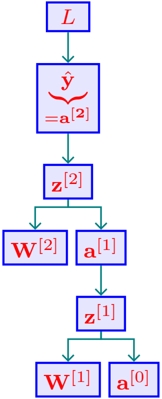
Gradient calculation using chain rule:

$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{W}^{[2]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$\nabla_{\mathbf{W}^{[1]}}(L) = \qquad \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph: Gradient calculation using chain rule:

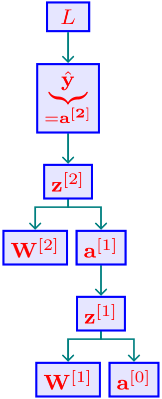


$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{W}^{[2]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$\nabla_{\mathbf{W}^{[1]}}(L) = \qquad \nabla_{\mathbf{a}^{[1]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph: Gradient calculation using chain rule:

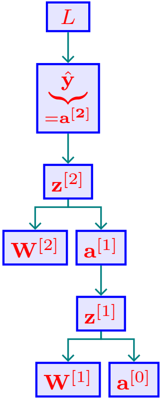


$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{W}^{[2]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$\nabla_{\mathbf{W}^{[1]}}(L) = \nabla_{\mathbf{z}^{[1]}}(\mathbf{a}^{[1]}) \times \nabla_{\mathbf{a}^{[1]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph:



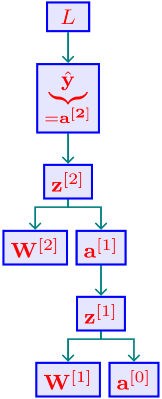
Gradient calculation using chain rule:

$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{W}^{[2]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$\nabla_{\mathbf{W}^{[1]}}(L) = \nabla_{\mathbf{W}^{[1]}}(\mathbf{z}^{[1]}) \times \nabla_{\mathbf{z}^{[1]}}(\mathbf{a}^{[1]}) \times \nabla_{\mathbf{a}^{[1]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph:



Gradient calculation using chain rule:

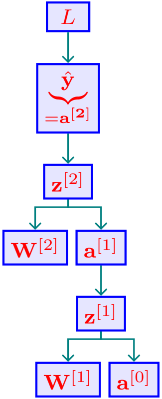
$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{W}^{[2]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$\nabla_{\mathbf{W}^{[1]}}(L) = \nabla_{\mathbf{W}^{[1]}}(\mathbf{z}^{[1]}) \times \nabla_{\mathbf{z}^{[1]}}(\mathbf{a}^{[1]}) \times \nabla_{\mathbf{a}^{[1]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$= \begin{bmatrix} \nabla_{\mathbf{z}^{[1]}}\left(a_1^{[1]}\right) & \nabla_{\mathbf{z}^{[1]}}\left(a_2^{[1]}\right) & \nabla_{\mathbf{z}^{[1]}}\left(a_3^{[1]}\right) & \nabla_{\mathbf{z}^{[1]}}\left(a_4^{[1]}\right) \end{bmatrix}$$

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph:



Gradient calculation using chain rule:

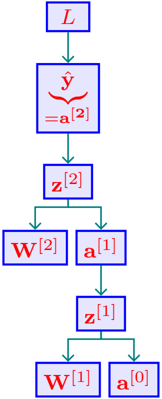
$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{W}^{[2]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$\nabla_{\mathbf{W}^{[1]}}(L) = \nabla_{\mathbf{W}^{[1]}}(\mathbf{z}^{[1]}) \times \nabla_{\mathbf{z}^{[1]}}(\mathbf{a}^{[1]}) \times \nabla_{\mathbf{a}^{[1]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$= \left[\nabla_{\mathbf{z}^{[1]}}\left(g\left(z_1^{[1]}\right)\right) \quad \nabla_{\mathbf{z}^{[1]}}\left(g\left(z_2^{[1]}\right)\right) \quad \nabla_{\mathbf{z}^{[1]}}\left(g\left(z_3^{[1]}\right)\right) \quad \nabla_{\mathbf{z}^{[1]}}\left(g\left(z_4^{[1]}\right)\right)\right]$$

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph:



Gradient calculation using chain rule:

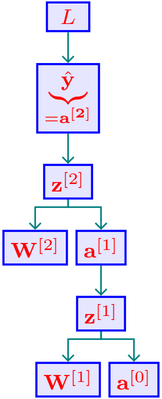
$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{W}^{[2]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$\nabla_{\mathbf{W}^{[1]}}(L) = \nabla_{\mathbf{W}^{[1]}}(\mathbf{z}^{[1]}) \times \nabla_{\mathbf{z}^{[1]}}(\mathbf{a}^{[1]}) \times \nabla_{\mathbf{a}^{[1]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$= \begin{bmatrix} \nabla_{z_1^{[1]}} g\left(z_1^{[1]}\right) & \nabla_{z_1^{[1]}} g\left(z_2^{[1]}\right) & \nabla_{z_1^{[1]}} g\left(z_3^{[1]}\right) & \nabla_{z_1^{[1]}} g\left(z_4^{[1]}\right) \\ \nabla_{z_2^{[1]}} g\left(z_1^{[1]}\right) & \nabla_{z_2^{[1]}} g\left(z_2^{[1]}\right) & \nabla_{z_2^{[1]}} g\left(z_3^{[1]}\right) & \nabla_{z_2^{[1]}} g\left(z_4^{[1]}\right) \\ \nabla_{z_1^{[1]}} g\left(z_3^{[1]}\right) & \nabla_{z_3^{[1]}} g\left(z_2^{[1]}\right) & \nabla_{z_3^{[1]}} g\left(z_3^{[1]}\right) & \nabla_{z_3^{[1]}} g\left(z_4^{[1]}\right) \\ \nabla_{z_4^{[1]}} g\left(z_1^{[1]}\right) & \nabla_{z_4^{[1]}} g\left(z_2^{[1]}\right) & \nabla_{z_4^{[1]}} g\left(z_3^{[1]}\right) & \nabla_{z_4^{[1]}} g\left(z_4^{[1]}\right) \end{bmatrix}$$

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph:



Gradient calculation using chain rule:

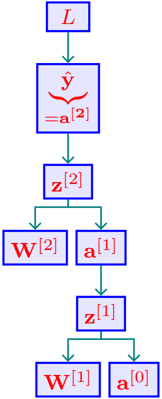
$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{W}^{[2]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$\nabla_{\mathbf{W}^{[1]}}(L) = \nabla_{\mathbf{W}^{[1]}}(\mathbf{z}^{[1]}) \times \nabla_{\mathbf{z}^{[1]}}(\mathbf{a}^{[1]}) \times \nabla_{\mathbf{a}^{[1]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$= \begin{bmatrix} \nabla_z(g(z))|_{z=z_1^{[1]}} & 0 & 0 & 0 \\ 0 & \nabla_z(g(z))|_{z=z_2^{[1]}} & 0 & 0 \\ 0 & 0 & \nabla_z(g(z))|_{z=z_3^{[1]}} & 0 \\ 0 & 0 & 0 & \nabla_z(g(z))|_{z=z_4^{[1]}} \end{bmatrix}$$

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph:



Gradient calculation using chain rule:

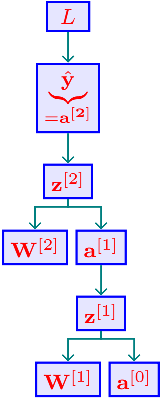
$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{W}^{[2]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$\nabla_{\mathbf{W}^{[1]}}(L) = \nabla_{\mathbf{W}^{[1]}}(\mathbf{z}^{[1]}) \times \nabla_{\mathbf{z}^{[1]}}(\mathbf{a}^{[1]}) \times \nabla_{\mathbf{a}^{[1]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

- $\mathbf{W}^{[2]}$ is updated first as soon as the gradient $\nabla_{\mathbf{W}^{[2]}}(L)$ is available.
- The term $\nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$ is reused for calculating $\nabla_{\mathbf{W}^{[1]}}(L)$.

Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph:

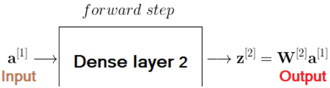


Gradient calculation using chain rule:

$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{W}^{[2]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

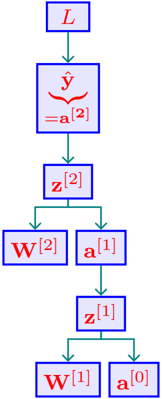
$$\nabla_{\mathbf{W}^{[1]}}(L) = \nabla_{\mathbf{W}^{[1]}}(\mathbf{z}^{[1]}) \times \nabla_{\mathbf{z}^{[1]}}(\mathbf{a}^{[1]}) \times \nabla_{\mathbf{a}^{[1]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

- $\mathbf{W}^{[2]}$ is updated first as soon as the gradient $\nabla_{\mathbf{W}^{[2]}}(L)$ is available.
- The term $\nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$ is reused for calculating $\nabla_{\mathbf{W}^{[1]}}(L)$.



Single hidden layer neural network gradient calculation calculation using backward propagation

Computation graph:

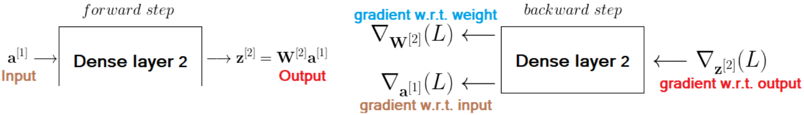


Gradient calculation using chain rule:

$$\nabla_{\mathbf{W}^{[2]}}(L) = \nabla_{\mathbf{W}^{[2]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

$$\nabla_{\mathbf{W}^{[1]}}(L) = \nabla_{\mathbf{W}^{[1]}}(\mathbf{z}^{[1]}) \times \nabla_{\mathbf{z}^{[1]}}(\mathbf{a}^{[1]}) \times \nabla_{\mathbf{a}^{[1]}}(\mathbf{z}^{[2]}) \times \nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$$

- $\mathbf{W}^{[2]}$ is updated first as soon as the gradient $\nabla_{\mathbf{W}^{[2]}}(L)$ is available.
- The term $\nabla_{\mathbf{z}^{[2]}}(\hat{\mathbf{y}}) \times \nabla_{\hat{\mathbf{y}}}(L)$ is reused for calculating $\nabla_{\mathbf{W}^{[1]}}(L)$.



Parameters and hyperparameters

Parameters and hyperparameters

- Model parameters are internal to the model that are learned from the data typically in an iterative manner following random initial guesses.

Parameters and hyperparameters

- Model parameters are internal to the model that are learned from the data typically in an iterative manner following random initial guesses.
- Examples of model parameters are *weights* and *biases* of a neural network.

Parameters and hyperparameters

- Model parameters are internal to the model that are learned from the data typically in an iterative manner following random initial guesses.
- Examples of model parameters are *weights* and *biases* of a neural network.
- Model parameters also help predict the response variable for unseen test data.

Parameters and hyperparameters

- Model parameters are internal to the model that are learned from the data typically in an iterative manner following random initial guesses.
- Examples of model parameters are *weights* and *biases* of a neural network.
- Model parameters also help predict the response variable for unseen test data.
- Model hyperparameters are external to the model and are not learned from data but are rather used to estimate the model parameters.

Parameters and hyperparameters

- Model parameters are internal to the model that are learned from the data typically in an iterative manner following random initial guesses.
- Examples of model parameters are *weights* and *biases* of a neural network.
- Model parameters also help predict the response variable for unseen test data.
- Model hyperparameters are external to the model and are not learned from data but are rather used to estimate the model parameters.
- Examples of model hyperparameters are *number of layers* and *nodes per layer in a neural network*, *learning rate*, *regularization strength* etc.

Parameters and hyperparameters

- Model parameters are internal to the model that are learned from the data typically in an iterative manner following random initial guesses.
- Examples of model parameters are *weights* and *biases* of a neural network.
- Model parameters also help predict the response variable for unseen test data.
- Model hyperparameters are external to the model and are not learned from data but are rather used to estimate the model parameters.
- Examples of model hyperparameters are *number of layers* and *nodes per layer in a neural network*, *learning rate*, *regularization strength* etc.
- Model hyperparameters are typically tuned when constructing models specific to a dataset.