Machine Learning

Claire Monteleoni
University of Colorado Boulder

LECTURE 23

Slides adapted from Chenhao Tan, Jordan Boyd-Graber, Chris Ketelsen. Credit also to Andrew Ng.

Today

Quick review

Back propagation
- Chain rule (review)
- Back propagation
- Full algorithm

Extensions and Improvements
- Improve SGD
- Data prepocessing

Outline

Neural networks in a nutshell

- Training data $S_{\text{train}} = \{(\boldsymbol{x}, y)\}$
- Network architecture (model)

$$\hat{y} = f_W(\boldsymbol{x})$$

$$\boldsymbol{W}^l, \boldsymbol{b}^l, l = 1, \dots, L$$

- Loss function (objective function)

$$\mathscr{L}(y, \hat{y})$$

- How do we learn the parameters?

Neural networks in a nutshell

- Training data $S_{\text{train}} = \{(\boldsymbol{x}, y)\}$
- Network architecture (model)

$$\hat{y} = f_w(\boldsymbol{x})$$

$$\boldsymbol{W}^l, \boldsymbol{b}^l, l = 1, \ldots, L$$

- Loss function (objective function)

$$\mathscr{L}(y, \hat{y})$$

- How do we learn the parameters?
  Stochastic gradient descent,

$$\boldsymbol{W}^l \leftarrow \boldsymbol{W}^l - \eta \frac{\partial \mathscr{L}(y, \hat{y})}{\partial \boldsymbol{W}^l}$$

Challenge

- **Challenge**: How do we compute derivatives of the loss function with respect to weights and biases?
- **Solution**: Back Propagation

Outline

Quick review

Back propagation
- Chain rule (review)
- Back propagation
- Full algorithm

Extensions and Improvements
- Improve SGD
- Data prepocessing

The Chain Rule

The chain rule allows us to take derivatives of nested functions.
There are two forms of the Chain Rule

The Chain Rule

The chain rule allows us to take derivatives of nested functions.
There are two forms of the Chain Rule
Baby Chain Rule:

$$\frac{d}{dx} f(g(x)) = f'(g(x))\, g'(x) = \frac{df}{dg}\frac{dg}{dx}$$

The Chain Rule

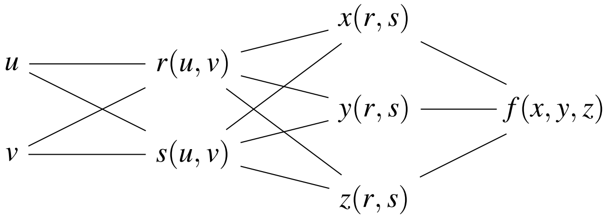
The chain rule allows us to take derivatives of nested functions.
There are two forms of the Chain Rule
Baby Chain Rule:

$$\frac{d}{dx} f(g(x)) = f'(g(x))\; g'(x) = \frac{df}{dg}\frac{dg}{dx}$$

Example: $\frac{d}{dx}\ \sin(x^2) = \cos(x^2)\ 2x$

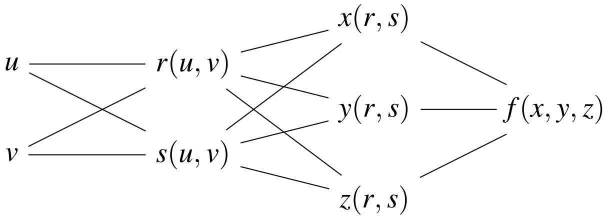
The Chain Rule

Full-Grown Adult Chain Rule:



The Chain Rule

Full-Grown Adult Chain Rule:



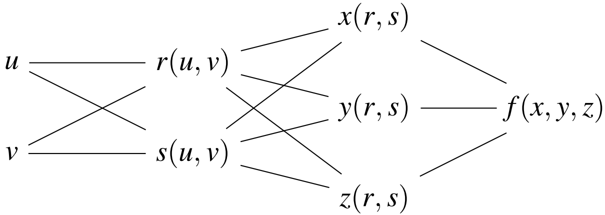
Derivative of $\mathscr{L}$ with respect to x:

$$\frac{\partial f}{\partial x}$$

Similarly, $\frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

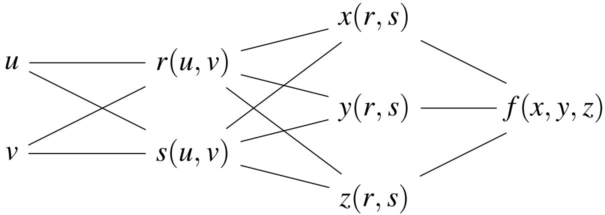
The Chain Rule

What is the derivative of f with respect to r?



The Chain Rule

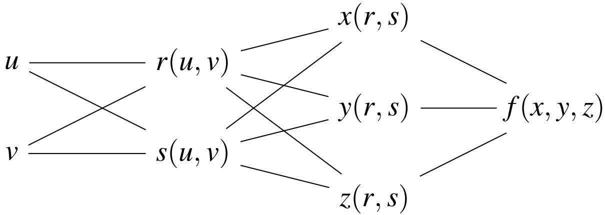
What is the derivative of f with respect to r?



$$\frac{\partial f}{\partial r} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial r} + \frac{\partial f}{\partial z}\frac{\partial z}{\partial r}$$

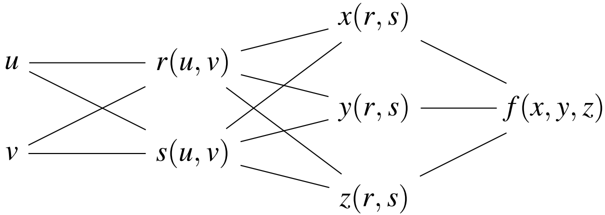
The Chain Rule

What is the derivative of f with respect to s?



The Chain Rule

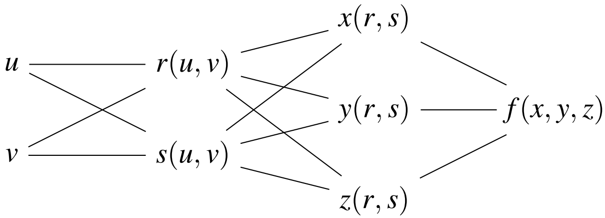
What is the derivative of f with respect to s?



$$\frac{\partial f}{\partial s} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial s} + \frac{\partial f}{\partial z}\frac{\partial z}{\partial s}$$

The Chain Rule

What is the derivative of f with respect to s?

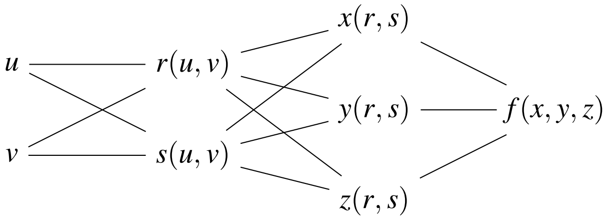


Example: Let $f = xyz$, $x = r$, $y = rs$, and $z = s$. Find $\partial f / \partial s$

$$\frac{\partial f}{\partial s} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial s} + \frac{\partial f}{\partial z}\frac{\partial z}{\partial s}$$

The Chain Rule

What is the derivative of f with respect to s?

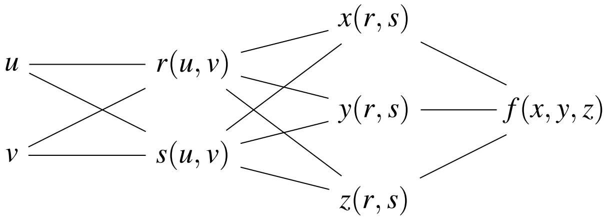


Example: Let $f = xyz$, $x = r$, $y = rs$, and $z = s$. Find $\partial f / \partial s$

$$\frac{\partial f}{\partial s} = yz \cdot 0 + xz \cdot r + xy \cdot 1$$

The Chain Rule

What is the derivative of f with respect to s?

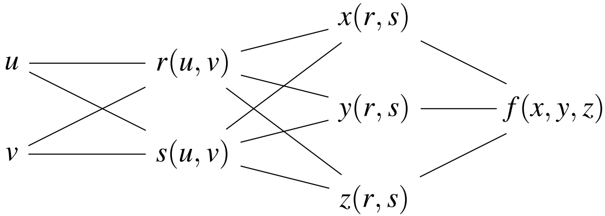


Example: Let $f = xyz$, $x = r$, $y = rs$, and $z = s$. Find $\partial f / \partial s$

$$\frac{\partial f}{\partial s} = rs^2 \cdot 0 + rs \cdot r + r^2 s \cdot 1$$

The Chain Rule

What is the derivative of f with respect to s?

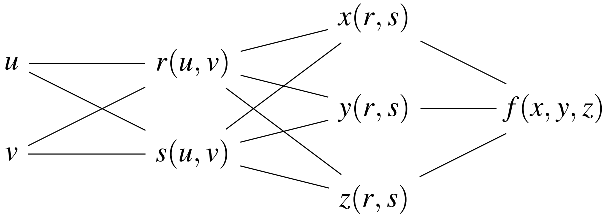


Example: Let $f = xyz$, $x = r$, $y = rs$, and $z = s$. Find $\partial f / \partial s$

$$\frac{\partial f}{\partial s} = 2r^2 s$$

The Chain Rule

What is the derivative of f with respect to s?

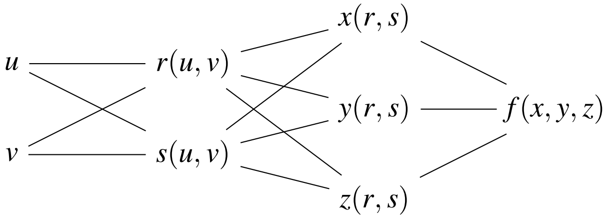


Example: Let $f = xyz$, $x = r$, $y = rs$, and $z = s$. Find $\partial f / \partial s$

$$f(r,s) = r \cdot rs \cdot s = r^2 s^2 \quad \Rightarrow \quad \frac{\partial f}{\partial s} = 2r^2 s \ \checkmark$$

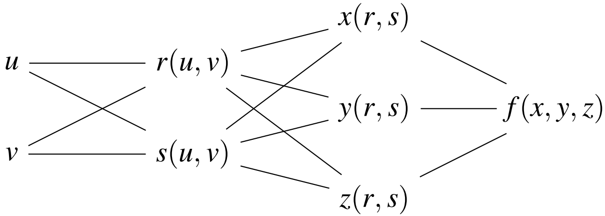
The Chain Rule

What is the derivative of f with respect to u?



The Chain Rule

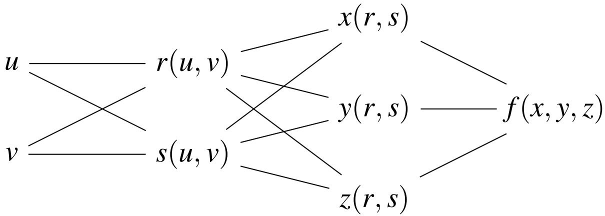
What is the derivative of f with respect to u?



$$\frac{\partial f}{\partial u} = \frac{\partial f}{\partial r}\frac{\partial r}{\partial u} + \frac{\partial f}{\partial s}\frac{\partial s}{\partial u}$$

The Chain Rule

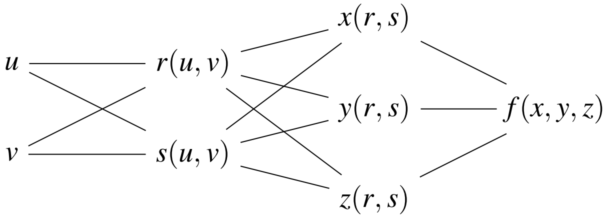
What is the derivative of f with respect to u?



Crux: If you know derivative of objective w.r.t. intermediate value in the chain, can eliminate everything in between.

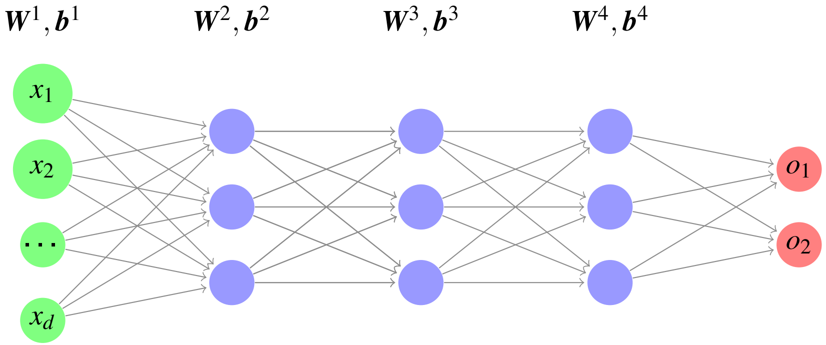
The Chain Rule

What is the derivative of f with respect to u?



Crux: If you know derivative of objective w.r.t. intermediate value in the chain, can eliminate everything in between.
This is the cornerstone of the Back Propagation algorithm.

Back Propagation



Back Propagation

Notation

- The input vector is denoted as $\boldsymbol{a}^0$.
- At each subsequent layer, $l > 0$, we define:

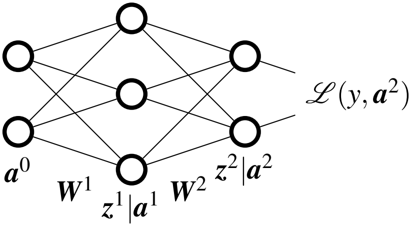
$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

and then we obtain $\boldsymbol{a}^l$ by applying the activation function for that layer, i.e.,

$$a_j^l = g^l(z_j^l)$$

Back Propagation

For the derivation, we'll consider a simple network

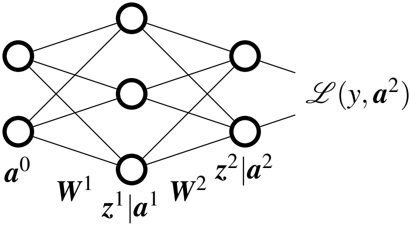


We want to use back propagation to compute partial derivatives of $\mathcal{L}$ w.r.t. the weights and biases

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^2}, \text{for } l = 1, 2$$

Back Propagation

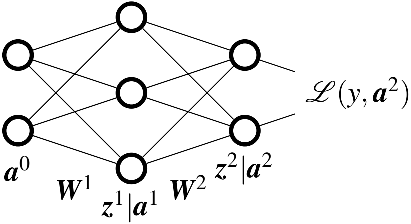
For the derivation, we'll consider a simple network



We need to choose an intermediate term that lives on the nodes, that we can easily compute derivative with respect to
Could choose $\boldsymbol{a}$'s, but we'll choose z's because math is easier

Back Propagation

For the derivation, we'll consider a simple network



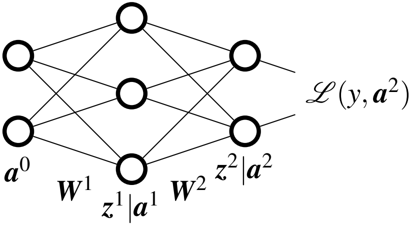
Define the derivative w.r.t. the $\boldsymbol{z}$'s by $\boldsymbol{\delta}$:

$$\delta_j^l = \frac{\partial \mathscr{L}}{\partial z_j^l}$$

Note that $\boldsymbol{\delta}^l$ has the same size as $\boldsymbol{z}^l$ and $\boldsymbol{a}^l$

Back Propagation

For the derivation, we'll consider a simple network



Let's compute δ^L for output layer L:

$$\delta_j^L = \frac{\partial \mathscr{L}}{\partial z_j^L} = \frac{\partial \mathscr{L}}{\partial a_j^L} \frac{d a_j^L}{d z_j^L}$$

Back Propagation

Continuing,

$$\delta_j^L = \frac{\partial \mathscr{L}}{\partial z_j^L} = \frac{\partial \mathscr{L}}{\partial a_j^L} \frac{da_j^L}{dz_j^L}$$

We know that $a_j^L = g(z_j^L)$, so $\frac{da_j^L}{dz_j^L} = g'(z_j^L)$. So we can apply this substitution:

$$\delta_j^L = \frac{\partial \mathscr{L}}{\partial a_j^L} g'(z_j^L)$$

Note: The first term is j^{th} entry of gradient of $\mathscr{L}$ w.r.t. $\boldsymbol{a}^L$, $\nabla_{a_j^L} \mathscr{L}$.

Back Propagation

So we can substitute this in, yielding:

$$\delta_j^L = (\nabla_{a_j^L}\mathscr{L})g'(z_j^L)$$

We can combine all of these into a vector operation

$$\boldsymbol{\delta}^L = \nabla_{\boldsymbol{a}^L}\mathscr{L} \odot g'(\boldsymbol{z}^L)$$

Where $g'(\boldsymbol{z}^L)$ is the activation function applied elementwise to $\boldsymbol{z}^L$.
The symbol $\odot$ indicates element-wise multiplication of vectors.

Back Propagation

So we can substitute this in, yielding:

$$\delta_j^L = (\nabla_{a_j^L} \mathscr{L}) g'(z_j^L)$$

We can combine all of these into a vector operation

$$\boldsymbol{\delta}^L = \nabla_{\boldsymbol{a}^L} \mathscr{L} \odot g'(\boldsymbol{z}^L)$$

Where $g'(\boldsymbol{z}^L)$ is the activation function applied elementwise to $\boldsymbol{z}^L$.
The symbol $\odot$ indicates element-wise multiplication of vectors.
Notice that computing $\boldsymbol{\delta}^L$ requires knowing activations.
This means that before we can compute derivatives for SGD through back propagation, we first run forward propagation through the network.

Back Propagation

Example: Suppose we're in regression setting and choose a sigmoid activation function:

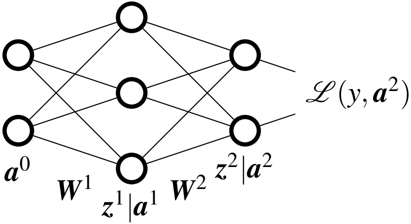
$$\mathscr{L} = \frac{1}{2}\sum_j (y_j - a_j^L)^2 \quad \text{and} \quad \sigma(z) = \mathrm{sigm}(z)$$

$$\frac{\partial \mathscr{L}}{\partial a_j^L} = (a_j^L - y_j), \quad \frac{da_j^L}{dz_j^L} = \sigma'(z_j^L) = \sigma(z_j^L)(1 - \sigma(z_j^L))$$

So $\boldsymbol{\delta}^L = (\boldsymbol{a}^L - y) \odot \sigma(\boldsymbol{z}^L) \odot (1 - \sigma(\boldsymbol{z}^L))$

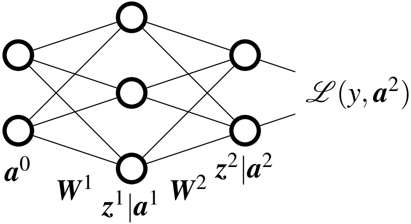
Back Propagation

- So we know how to compute the δ's for the output layer.
- But we still need to compute the partial derivatives w.r.t. to weights and biases.



Back Propagation

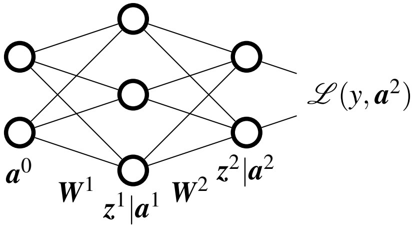
- So we know how to compute the δ's for the output layer.
- But we still need to compute the partial derivatives w.r.t. to weights and biases.



Question: What do you notice?

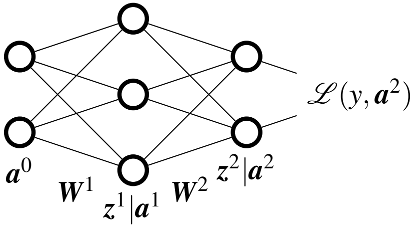
Back Propagation

We want to find derivative $\mathscr{L}$ w.r.t. to weights and biases



Each weight into a node in layer L is only related to a single δ_j^L.

Back Propagation

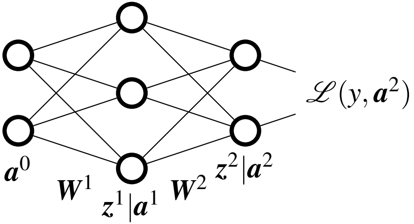


Given the network structure, $$\frac{\partial \mathscr{L}}{\partial w_{jk}^L} = \frac{\partial \mathscr{L}}{\partial z_j^L} \frac{\partial z_j^L}{\partial w_{jk}^L} = \delta_j^L \frac{\partial z_j^L}{\partial w_{jk}^L}$$

Need to compute $\dfrac{\partial z_j^L}{\partial w_{jk}^L}$. Recall $\mathbf{z}^L = W^L \mathbf{a}^{L-1} + \mathbf{b}^L$

$$j^{\text{th}} \text{ entry in vector} \Rightarrow \quad z_j^L = \sum_i w_{ji}^L a_i^{L-1} + b_j^L$$

Back Propagation



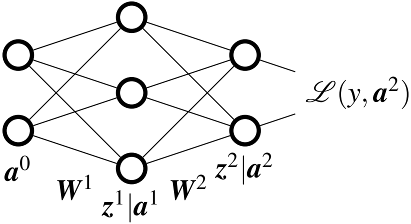
Given the network structure,

$$\frac{\partial \mathscr{L}}{\partial w_{jk}^L} = \frac{\partial \mathscr{L}}{\partial z_j^L} \frac{\partial z_j^L}{\partial w_{jk}^L} = \delta_j^L \frac{\partial z_j^L}{\partial w_{jk}^L}$$

Taking derivative w.r.t. w_{jk}^L gives

$$\Rightarrow \quad \frac{\partial z_j^L}{\partial w_{jk}^L} = a_k^{L-1} \quad \Rightarrow \quad \frac{\partial \mathscr{L}}{\partial w_{jk}^L} = a_k^{L-1} \delta_j^L$$

Back Propagation



So we have $$\frac{\partial \mathscr{L}}{\partial w_{jk}^L} = a_k^{L-1} \delta_j^L$$

Easy expression for derivative w.r.t. every weight leading into layer L.

Back Propagation

Let's make the notation a little more practical.

$$\boldsymbol{W}^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \end{bmatrix}$$

Back Propagation

Let's make the notation a little more practical.

$$\boldsymbol{W}^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \end{bmatrix}$$

$$\frac{\partial \mathscr{L}}{\partial \boldsymbol{W}^2} = \begin{bmatrix} \frac{\partial \mathscr{L}}{\partial w_{11}^2} & \frac{\partial \mathscr{L}}{\partial w_{12}^2} & \frac{\partial \mathscr{L}}{\partial w_{13}^2} \\ \frac{\partial \mathscr{L}}{\partial w_{21}^2} & \frac{\partial \mathscr{L}}{\partial w_{22}^2} & \frac{\partial \mathscr{L}}{\partial w_{23}^2} \end{bmatrix} = \begin{bmatrix} \delta_1^2 a_1^1 & \delta_1^2 a_2^1 & \delta_1^2 a_3^1 \\ \delta_2^2 a_1^1 & \delta_2^2 a_2^1 & \delta_2^2 a_3^1 \end{bmatrix}$$

Now we can write this as an outer-product of $\boldsymbol{\delta}^2$ and $\boldsymbol{a}^1$,

$$\frac{\partial \mathscr{L}}{\partial \boldsymbol{W}^2} = \boldsymbol{\delta}^2 (\boldsymbol{a}^1)^T$$

(Exercise: derive $\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^2}$)

Recap

- For a given training example $\boldsymbol{x}$, perform forward propagation to get $\boldsymbol{z}^l$ and $\boldsymbol{a}^l$ on each layer.
- Then to get the partial derivatives for $\boldsymbol{W}^2$ or $\boldsymbol{W}^L$:
  1. Compute $\boldsymbol{\delta}^L = \nabla_{\boldsymbol{a}^L}\mathscr{L} \odot g'(\boldsymbol{z}^L)$
  2. Compute $\frac{\partial \mathscr{L}}{\partial \boldsymbol{W}^L} = \boldsymbol{\delta}^L(\boldsymbol{a}^{L-1})^T$ and $\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^L} = \boldsymbol{\delta}^L$

Notice: these are very simple expressions for the derivatives with respect to the weights in the final hidden layer.

Recap

- For a given training example $\boldsymbol{x}$, perform forward propagation to get $\boldsymbol{z}^l$ and $\boldsymbol{a}^l$ on each layer.
- Then to get the partial derivatives for $\boldsymbol{W}^2$ or $\boldsymbol{W}^L$:
  1. Compute $\boldsymbol{\delta}^L = \nabla_{\boldsymbol{a}^L}\mathscr{L} \odot g'(\boldsymbol{z}^L)$
  2. Compute $\frac{\partial \mathscr{L}}{\partial \boldsymbol{W}^L} = \boldsymbol{\delta}^L(\boldsymbol{a}^{L-1})^T$ and $\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^L} = \boldsymbol{\delta}^L$

Notice: these are very simple expressions for the derivatives with respect to the weights in the final hidden layer.
Problem: How do we do the other layers?

Recap

- For a given training example $\boldsymbol{x}$, perform forward propagation to get $\boldsymbol{z}^l$ and $\boldsymbol{a}^l$ on each layer.
- Then to get the partial derivatives for $\boldsymbol{W}^2$ or $\boldsymbol{W}^L$:
  1. Compute $\boldsymbol{\delta}^L = \nabla_{\boldsymbol{a}^L} \mathscr{L} \odot g'(\boldsymbol{z}^L)$
  2. Compute $\frac{\partial \mathscr{L}}{\partial \boldsymbol{W}^L} = \boldsymbol{\delta}^L (\boldsymbol{a}^{L-1})^T$ and $\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^L} = \boldsymbol{\delta}^L$

Notice: these are very simple expressions for the derivatives with respect to the weights in the final hidden layer.
Problem: How do we do the other layers?
Since the formulas were so nice once we knew the adjacent $\boldsymbol{\delta}^l$, it would be nice if we could easily compute the $\boldsymbol{\delta}^l$'s on earlier layers.

Back Propagation

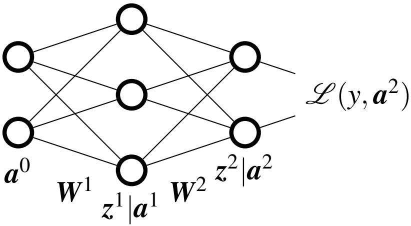
- The relationship between $\mathscr{L}$ and z^1 is really complicated because of multiple passes through the activation functions.

Back Propagation

- The relationship between $\mathscr{L}$ and z^1 is really complicated because of multiple passes through the activation functions.
- Back propagation to the rescue!

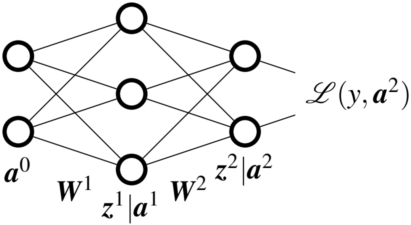
Back Propagation

- The relationship between $\mathscr{L}$ and z^1 is really complicated because of multiple passes through the activation functions.
- Back propagation to the rescue!
- Notice that δ^1 depends on δ^2.



Back Propagation

Notice that δ^1 depends on δ^2.

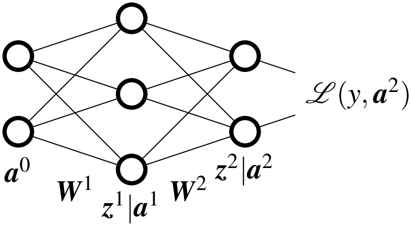


By the (adult) chain rule,

$$\frac{\partial \mathscr{L}}{\partial z_k^{l-1}} = \sum_j \frac{\partial \mathscr{L}}{\partial z_j^l} \frac{\partial z_j^l}{\partial z_k^{l-1}}$$

Back Propagation

Notice that δ^1 depends on δ^2.

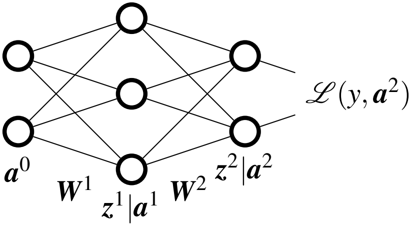


By the (adult) chain rule,

$$\delta_k^{l-1} = \frac{\partial \mathscr{L}}{\partial z_k^{l-1}} = \sum_j \frac{\partial \mathscr{L}}{\partial z_j^l} \frac{\partial z_j^l}{\partial z_k^{l-1}} = \sum_j \delta_j^l \frac{\partial z_j^l}{\partial z_k^{l-1}}$$

Back Propagation

Notice that δ^1 depends on δ^2.



By the (adult) chain rule,

$$\delta_2^1 = \frac{\partial \mathscr{L}}{\partial z_2^1} = \delta_1^2 \frac{\partial z_1^2}{\partial z_2^1} + \delta_2^2 \frac{\partial z_2^2}{\partial z_2^1}$$

Back Propagation

$$\delta_2^1 = \frac{\partial \mathscr{L}}{\partial z_2^1} = \delta_1^2 \frac{\partial z_1^2}{\partial z_2^1} + \delta_2^2 \frac{\partial z_2^2}{\partial z_2^1}$$

Back Propagation

$$\delta_2^1 = \frac{\partial \mathcal{L}}{\partial z_2^1} = \delta_1^2 \frac{\partial z_1^2}{\partial z_2^1} + \delta_2^2 \frac{\partial z_2^2}{\partial z_2^1}$$

Recall that $\boldsymbol{z}^2 = \boldsymbol{W}^2 a^1 + \boldsymbol{b}^2$. So we can write:

$$z_i^2 = w_{i1}^2 a_1^1 + w_{i2}^2 a_2^1 + w_{i3}^2 a_3^1 + b_i^2$$

Taking the derivative $\frac{\partial z_i^2}{\partial z_2^1} = w_{i2}^2 g'(z_2^1)$, and plugging in gives

$$\delta_2^1 = \frac{\partial \mathcal{L}}{\partial z_2^1} = \delta_1^2 w_{12}^2 g'(z_2^1) + \delta_2^2 w_{22}^2 g'(z_2^1)$$

Back Propagation

If we do this for each of the 3 δ_i^1's, something nice happens:
(Exercise: work out δ_1^1 and δ_3^1)

$$\begin{aligned}
\delta_1^1 &= \delta_1^2 w_{11}^2 g'(z_1^1) + \delta_2^2 w_{21}^2 g'(z_1^1) \\
\delta_2^1 &= \delta_1^2 w_{12}^2 g'(z_2^1) + \delta_2^2 w_{22}^2 g'(z_2^1) \\
\delta_3^1 &= \delta_1^2 w_{13}^2 g'(z_3^1) + \delta_2^2 w_{23}^2 g'(z_3^1)
\end{aligned}$$

Back Propagation

If we do this for each of the 3 δ_i^1's, something nice happens:
(Exercise: work out δ_1^1 and δ_3^1)

$$\begin{aligned}
\delta_1^1 &= \delta_1^2 w_{11}^2 g'(z_1^1) + \delta_2^2 w_{21}^2 g'(z_1^1) \\
\delta_2^1 &= \delta_1^2 w_{12}^2 g'(z_2^1) + \delta_2^2 w_{22}^2 g'(z_2^1) \\
\delta_3^1 &= \delta_1^2 w_{13}^2 g'(z_3^1) + \delta_2^2 w_{23}^2 g'(z_3^1)
\end{aligned}$$

Notice that each row of the system gets multiplied by $g'(z_i^1)$, so let's factor those out.

Back Propagation

If we do this for each of the 3 δ_i^2's, something nice happens:

$$\begin{aligned}
\delta_1^1 &= (\delta_1^2 w_{11}^2 + \delta_2^2 w_{21}^2) \cdot g'(z_1^1) \\
\delta_2^1 &= (\delta_1^2 w_{12}^2 + \delta_2^2 w_{22}^2) \cdot g'(z_2^1) \\
\delta_3^1 &= (\delta_1^2 w_{13}^2 + \delta_2^2 w_{23}^2) \cdot g'(z_3^1)
\end{aligned}$$

Back Propagation

If we do this for each of the 3 δ_i^2's, something nice happens:

$$\begin{aligned}
\delta_1^1 &= (\delta_1^2 w_{11}^2 + \delta_2^2 w_{21}^2) \cdot g'(z_1^1) \\
\delta_2^1 &= (\delta_1^2 w_{12}^2 + \delta_2^2 w_{22}^2) \cdot g'(z_2^1) \\
\delta_3^1 &= (\delta_1^2 w_{13}^2 + \delta_2^2 w_{23}^2) \cdot g'(z_3^1)
\end{aligned}$$

Remember $\boldsymbol{\delta}^2 = \begin{bmatrix} \delta_1^2 \\ \delta_2^2 \end{bmatrix}$, $\boldsymbol{W}^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \end{bmatrix}$

Do you see $\boldsymbol{\delta}^2$ and $\boldsymbol{W}^2$ lurking anywhere in the above system?

Back Propagation

If we do this for each of the 3 δ_i^2's, something nice happens:

$$\begin{aligned}
\delta_1^1 &= (\delta_1^2 w_{11}^2 + \delta_2^2 w_{21}^2) \cdot g'(z_1^1) \\
\delta_2^2 &= (\delta_1^2 w_{12}^2 + \delta_2^2 w_{22}^2) \cdot g'(z_2^1) \\
\delta_3^2 &= (\delta_1^2 w_{13}^2 + \delta_2^2 w_{23}^2) \cdot g'(z_3^1)
\end{aligned}$$

Does this help?

$$(\boldsymbol{W}^2)^T = \begin{bmatrix} w_{11}^2 & w_{21}^2 \\ w_{12}^2 & w_{22}^2 \\ w_{13}^2 & w_{23}^2 \end{bmatrix}, \boldsymbol{\delta}^2 = \begin{bmatrix} \delta_1^2 \\ \delta_2^2 \end{bmatrix}.$$

Back Propagation

If we do this for each of the 3 δ_i^2's, something nice happens:

$$\begin{aligned}
\delta_1^1 &= (\delta_1^2 w_{11}^2 + \delta_2^2 w_{21}^2) \cdot g'(z_1^1) \\
\delta_2^1 &= (\delta_1^2 w_{12}^2 + \delta_2^2 w_{22}^2) \cdot g'(z_2^1) \\
\delta_3^1 &= (\delta_1^2 w_{13}^2 + \delta_2^2 w_{23}^2) \cdot g'(z_3^1)
\end{aligned}$$

$$\boldsymbol{\delta}^1 = (\boldsymbol{W}^2)^T \boldsymbol{\delta}^2 \odot g'(\boldsymbol{z}^1)$$

Back Propagation

Great! We can compute $\boldsymbol{\delta}^1$ from $\boldsymbol{\delta}^2$.
Then we can compute derivatives of $\mathscr{L}$ w.r.t. weights $\boldsymbol{W}^1$ and biases $\boldsymbol{b}^1$ exactly the way we did for $\boldsymbol{W}^2$ and biases $\boldsymbol{b}^2$

1. Compute $\boldsymbol{\delta}^1 = (\boldsymbol{W}^2)^T \boldsymbol{\delta}^2 \odot g'(\boldsymbol{z}^1)$
2. Compute $\frac{\partial \mathscr{L}}{\partial \boldsymbol{W}^1} = \boldsymbol{\delta}^1 (\boldsymbol{a}^0)^T$ and $\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^1} = \boldsymbol{\delta}^1$

Back Propagation

Great! We can compute $\boldsymbol{\delta}^1$ from $\boldsymbol{\delta}^2$.
Then we can compute derivatives of $\mathscr{L}$ w.r.t. weights $\boldsymbol{W}^1$ and biases $\boldsymbol{b}^1$ exactly the way we did for $\boldsymbol{W}^2$ and biases $\boldsymbol{b}^2$

1. Compute $\boldsymbol{\delta}^1 = (\boldsymbol{W}^2)^T \boldsymbol{\delta}^2 \odot g'(\boldsymbol{z}^1)$
2. Compute $\frac{\partial \mathscr{L}}{\partial \boldsymbol{W}^1} = \boldsymbol{\delta}^1 (\boldsymbol{a}^0)^T$ and $\frac{\partial \mathscr{L}}{\partial \boldsymbol{b}^1} = \boldsymbol{\delta}^1$

We worked this out for a simple network with one hidden layer.
Nothing we've done assumed anything about the number of layers, so we can apply the same procedure recursively with any number of layers.

Back Propagation

$\delta^L = \nabla_{\boldsymbol{a}^L} \mathscr{L} \odot \sigma'(\boldsymbol{z}^L)$ # Compute δ's on output layer

For $\ell = L, \ldots, 1$

$\frac{\partial \mathscr{L}}{\partial \mathbf{W}^\ell} = \boldsymbol{\delta}^\ell (\boldsymbol{a}^{l-1})^T$ # Compute weight derivatives

$\dfrac{\partial \mathscr{L}}{\partial \boldsymbol{b}^\ell} = \boldsymbol{\delta}^\ell$ # Compute bias derivatives

$\boldsymbol{\delta}^{\ell-1} = \left(W^\ell\right)^T \boldsymbol{\delta}^\ell \odot \sigma'(\mathbf{z}^{\ell-1})$ # Back prop $\boldsymbol{\delta}$'s to previous layer

(After this, ready to do a SGD update on weights/biases)

Training a Feed-Forward Neural Network

Initialize weights and biases.
Loop over each training example in random order:

1. Forward Propagate to get activations on each layer
2. Back Propagate to get derivatives
3. Update weights and biases via Stochastic Gradient Descent
4. Repeat

Training a Feed-Forward Neural Network

Remaining Questions:

1. Can I batch this?
2. When do we stop?
3. How do we initialize weights and biases?

Outline

Extensions and Improvements

Huge literature, currently an extremely active research area
Some areas of improvement:

- Improve stochastic gradient descent
- Unstable gradients
- Data preprocessing
- Weight Initialization
- Model Architecture

Mini-batch SGD

- We update the weights and biases using stochastic gradient descent (SGD), instead of batch gradient descent, as the latter is too expensive.

Mini-batch SGD

- We update the weights and biases using stochastic gradient descent (SGD), instead of batch gradient descent, as the latter is too expensive.
    - That is, instead of computing a gradient on the whole training data set, we compute it on one data point at a time. So we can make progress after training on each point.

Mini-batch SGD

- We update the weights and biases using stochastic gradient descent (SGD), instead of batch gradient descent, as the latter is too expensive.
    - That is, instead of computing a gradient on the whole training data set, we compute it on one data point at a time. So we can make progress after training on each point.
    - SGD has a noisier optimization path than GD, and will never converge to the optimum. This can be partially addressed by choice of learning rate, e.g., a small learning rate, or one that decays with time.

Mini-batch SGD

- However, SGD, by only processing a single point at a time, does not take advantage of speedups available with vectorization.

Mini-batch SGD

- However, SGD, by only processing a single point at a time, does not take advantage of speedups available with vectorization.
- Therefore, a common improvement to SGD is to use mini-batch SGD
  - where the batch-size is larger than one, but smaller than the whole training set size.

Mini-batch SGD

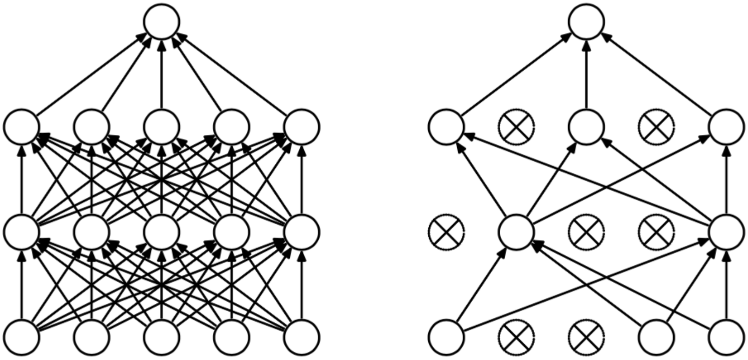
- However, SGD, by only processing a single point at a time, does not take advantage of speedups available with vectorization.
- Therefore, a common improvement to SGD is to use mini-batch SGD
  - where the batch-size is larger than one, but smaller than the whole training set size.
  - This takes advantage of vectorization, and thus provides speedups over SGD.
  - It also reduces the noise in SGD's optimization path somewhat. (One can also use decaying learning rates here).

Mini-batch SGD

- However, SGD, by only processing a single point at a time, does not take advantage of speedups available with vectorization.
- Therefore, a common improvement to SGD is to use mini-batch SGD
  - where the batch-size is larger than one, but smaller than the whole training set size.
  - This takes advantage of vectorization, and thus provides speedups over SGD.
  - It also reduces the noise in SGD's optimization path somewhat. (One can also use decaying learning rates here).
  - Meanwhile, unlike batch GD, you do not need to go through your whole data set before making progress.

Dropout

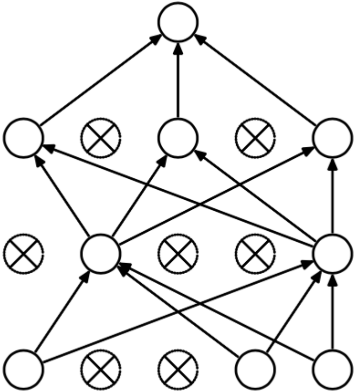
"randomly set some neurons to zero in the forward pass" [Srivastava et al. 2014]



(a) Standard Neural Net

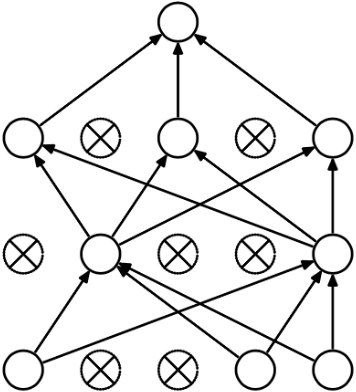
(b) After applying dropout.

Dropout



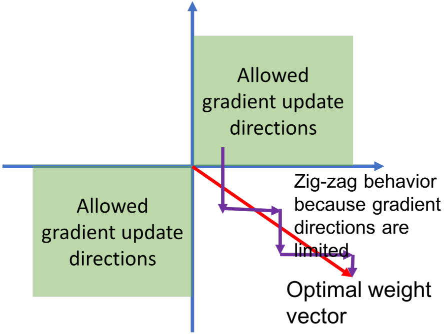
Forces the network to have a redundant representation

Dropout



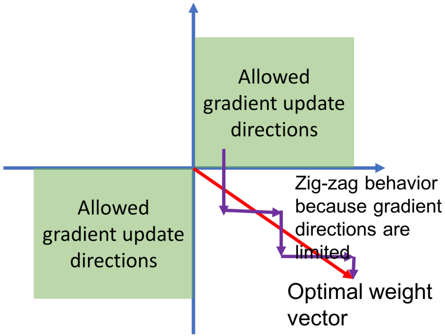
Another interpretation: Dropout is training a large ensemble of models (differing on which neurons are zeroed out)

Why preprocess the input data?



If all inputs x are positive, the gradients on w are either all positive or all negative.

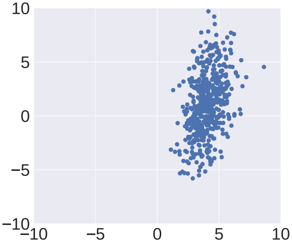
Why preprocess the input data?



If all inputs x are positive, the gradients on w are either all positive or all negative.
Solution: zero-center the inputs!

Data preprocessing

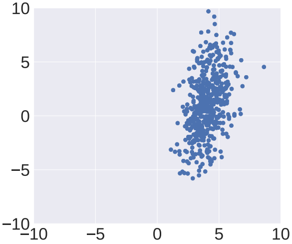
Original data

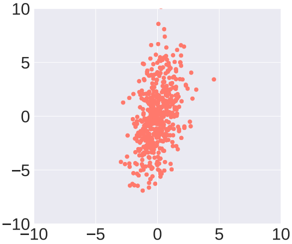


Data preprocessing

Original data

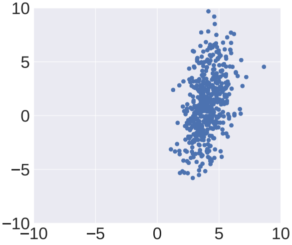
Zero-centered data
$(X - X.mean(axis = 0))$



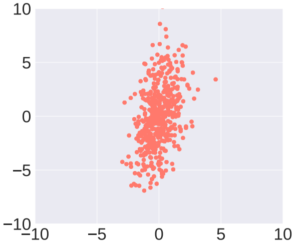


Data preprocessing

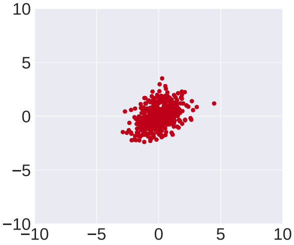
Original data



Zero-centered data
$(X - X.mean(axis = 0))$

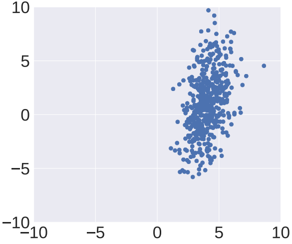


Normalized data
$(X/ = np.std(X, axis = 0))$

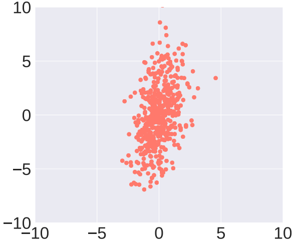


Data preprocessing

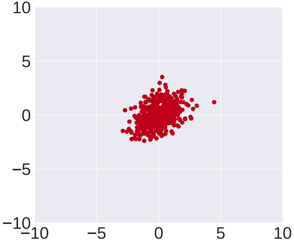
Original data



Zero-centered data
$(X - X.mean(axis = 0))$



Normalized data
$(X/ = np.std(X, axis = 0))$



PCA, whitening

Batch normalization

Why do this only for the input data? [Ioffe and Szegedy, 2015]
Consider a batch of activations at some layer. Make each dimension unit gaussian:

$$\hat{a}^k = \frac{a^k - E[a^k]}{\sqrt{Var[a^k]}}$$

Batch normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- Reduces internal covariant shift
- Reduces the dependence of gradients o the scale of the parameters or their initial values
- Allows higher learning rates and use of saturating nonlinearities
- May reduce the need for dropout?

Batch normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

During training, use batch mean and batch variance; during testing use empirical mean and variance on training data