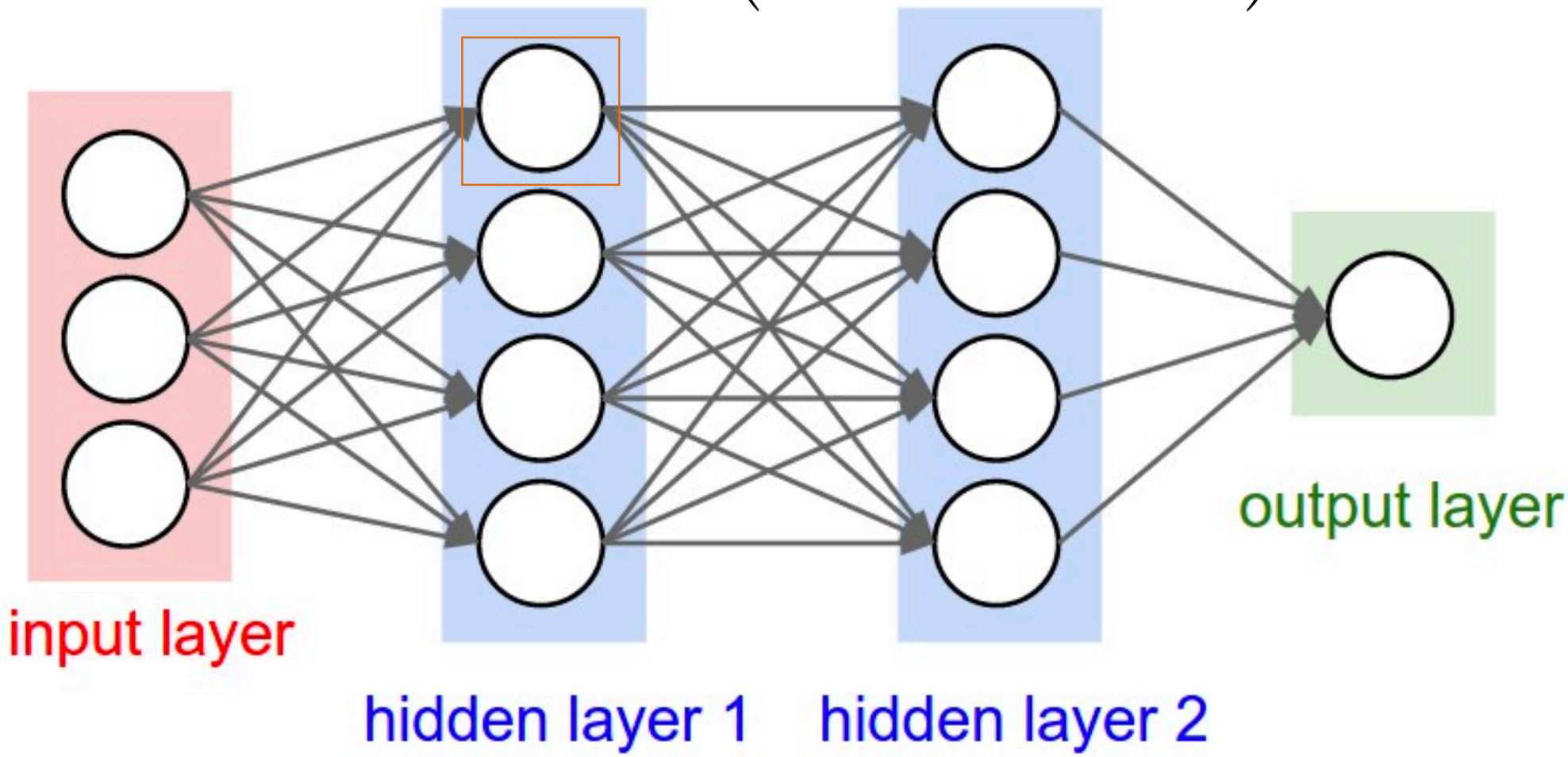
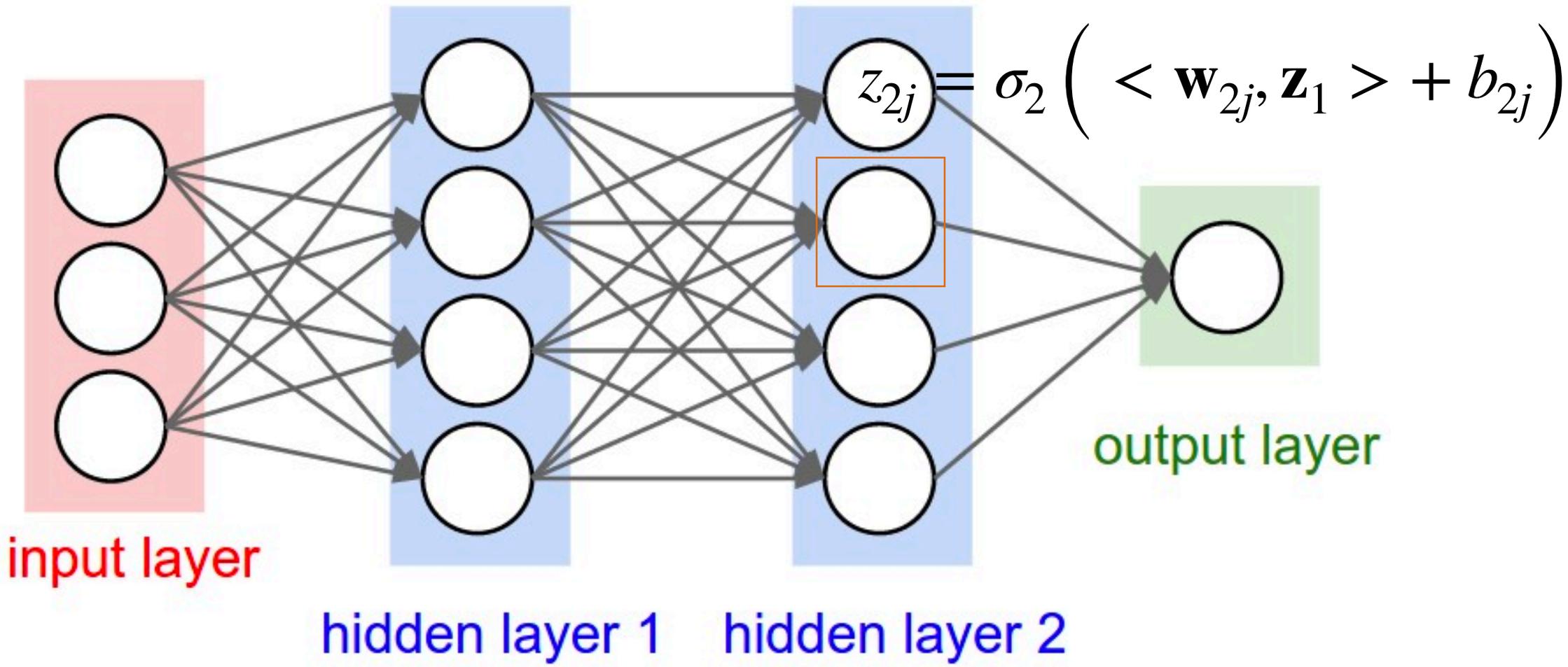
Miscellaneous Tips for deep neural networks

Recap: deep neural networks

$$z_{1j} = \sigma_1 \left(< \mathbf{w}_{1j}, \mathbf{x} > + b_{1j} \right)$$



Recap: deep neural networks



Recap: derivatives of artificial neurons

$$\frac{\partial y}{\partial b} = \sigma' \left(b + \sum_{j=1}^{p} w_j x_j \right)$$

$$\frac{\partial y}{\partial w_j} = \sigma' \left(b + \sum_{j=1}^{p} w_j x_j \right) \cdot x_j$$

$$\frac{\partial y}{\partial x_j} = \sigma' \left(b + \sum_{j=1}^{p} w_j x_j \right) \cdot w_j$$

Recap: deep neural networks

$$\mathbf{z}_1 = \sigma_1\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right)$$
$$\mathbf{z}_2 = \sigma_2\left(\mathbf{W}_2\mathbf{z}_1 + \mathbf{b}_2\right)$$
$$\vdots$$
$$\mathbf{z}_\ell = \sigma_\ell\left(\mathbf{W}_\ell\mathbf{z}_{\ell-1} + \mathbf{b}_\ell\right)$$
$$\hat{y} = f(\mathbf{x}) = \sigma_{\ell+1}\left(\mathbf{W}_{\ell+1}\mathbf{z}_\ell + \mathbf{b}_{\ell+1}\right)$$

The empirical risk of a feed–forward networks becomes

$$R = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, \hat{y}_i\right)$$

Recap: backpropagation

Obtain ∇R automatically by chain rules:

$$\frac{\partial R}{\partial \mathbf{W}_{\ell+1}} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial L\left(y_i, \hat{y}_i\right)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{W}_{\ell+1}},$$

$$\frac{\partial R}{\partial \mathbf{W}_{\ell}} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial L\left(y_i, \hat{y}_i\right)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{z}_{\ell}} \frac{\partial \mathbf{z}_{\ell}}{\mathbf{W}_{\ell}},$$

$$\frac{\partial R}{\partial \mathbf{W}_{\ell-1}} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial L\left(y_i, \hat{y}_i\right)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{z}_{\ell}} \frac{\partial \mathbf{z}_{\ell}}{\partial \mathbf{z}_{\ell-1}} \frac{\partial \mathbf{z}_{\ell-1}}{\partial \mathbf{W}_{\ell-1}},$$

$$\vdots$$

Agenda

- Feature normalization
- Vanishing and exploding gradients
- Weight initialization
- Batch normalization

Feature normalization

- Mathematically, the optimal value of $\mathbf{W}_1\mathbf{x}$ is invariant to the scale of $\mathbf{x}$.

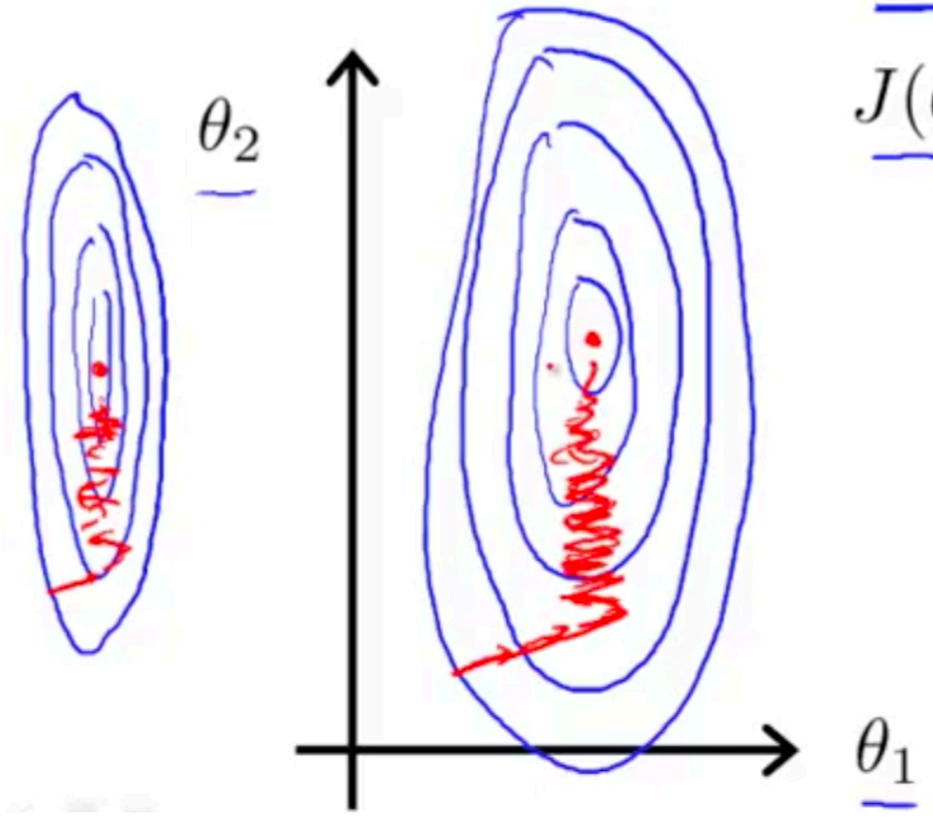
- However, the gradient of $\mathbf{W}_1$ does depend on the scale of $\mathbf{x}$ since

$$\frac{\partial y}{\partial w_j} = \sigma'\left(b + \sum_{j=1}^{p} w_j x_j\right) \cdot x_j$$

Feature normalization

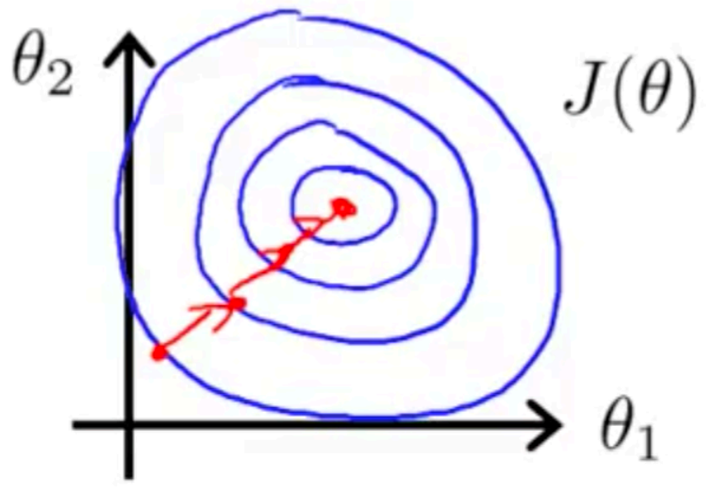
E.g. x_1 = size (0-2000 feet^2)

x_2 = number of bedrooms (1-5)



$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$



https://www.ritchieng.com/multi-variable-linear-regression/

Vanishing gradient

- Recall that

$$\frac{\partial R}{\partial \mathbf{W}_l} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial L\left(y_i, \hat{y}_i\right)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{z}_\ell} \frac{\partial \mathbf{z}_\ell}{\partial \mathbf{z}_{\ell-1}} \cdots \frac{\partial \mathbf{z}_{l+1}}{\partial \mathbf{z}_l} \frac{\partial \mathbf{z}_l}{\partial \mathbf{W}_l},$$

- $\frac{\partial R}{\partial \mathbf{W}_l}$ may vanish if $\frac{\partial \mathbf{z}_{j+1}}{\partial \mathbf{z}_j} \approx 0$ for some j or $\left|\frac{\partial \mathbf{z}_{j+1}}{\partial \mathbf{z}_j}\right|$ are small for lots of j

Exploding gradient

- Similarly, $\frac{\partial R}{\partial \mathbf{W}_l}$ may also explode when $\left|\frac{\partial \mathbf{z}_{j+1}}{\partial \mathbf{z}_j}\right|$ are large for lots of j

- We can prevent exploding gradient by clipping it (gradient clipping)

Why can't we start from $\mathbf{W} = \mathbf{0}$?

- Recall that

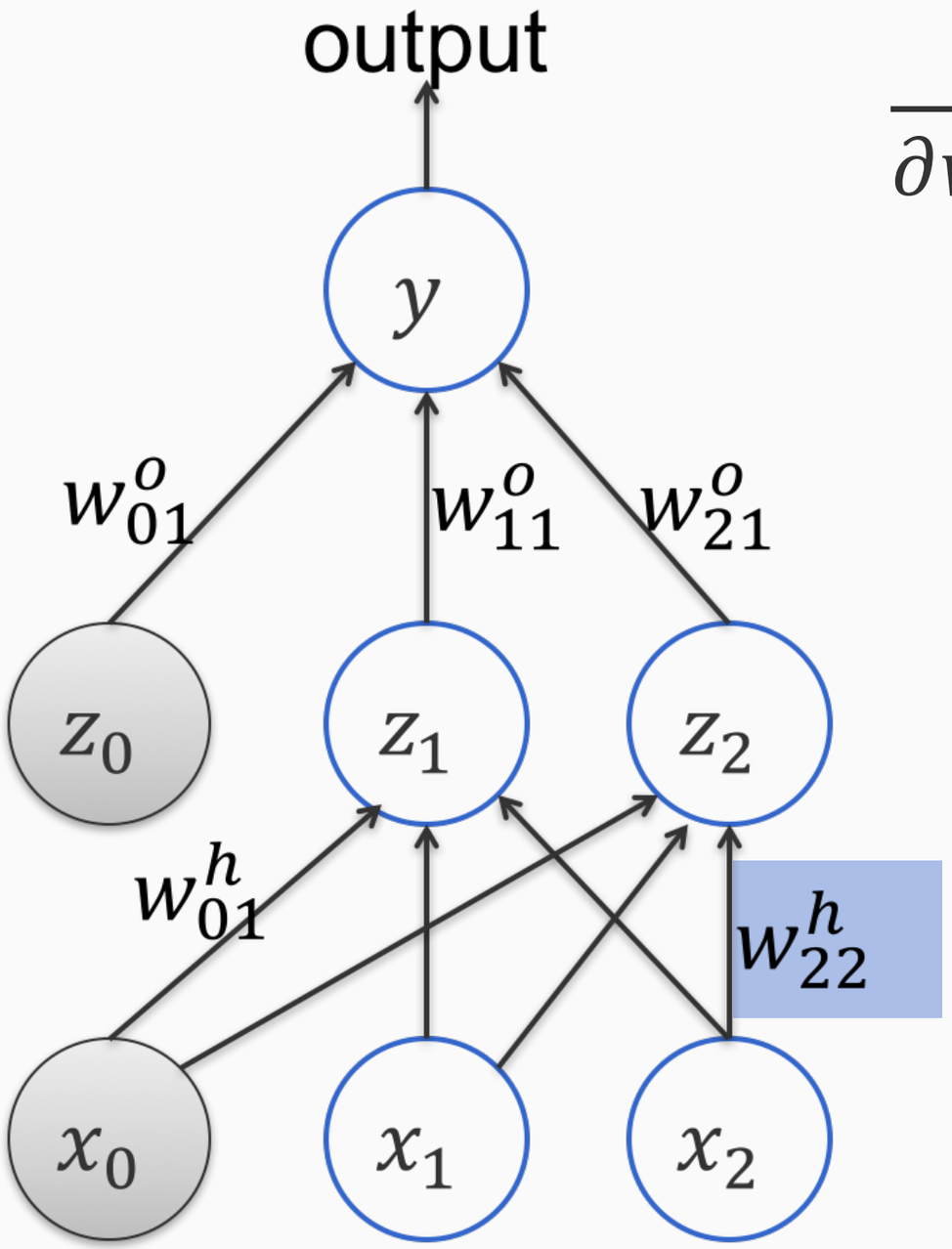
$$\frac{\partial R}{\partial \mathbf{W}_l} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial L\left(y_i, \hat{y}_i\right)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{z}_\ell} \frac{\partial \mathbf{z}_\ell}{\partial \mathbf{z}_{\ell-1}} \cdots \frac{\partial \mathbf{z}_{l+1}}{\partial \mathbf{z}_l} \frac{\partial \mathbf{z}_l}{\partial \mathbf{W}_l},$$

while

$$\frac{\partial \mathbf{z}_{l+1}}{\partial \mathbf{z}_l} = \mathbf{W}_{l+1} \mathbf{f}'\left(\mathbf{b}_{l+1} + \mathbf{W}_{l+1} \mathbf{z}_l\right) = 0$$

when $\mathbf{W}_{l+1} = 0$

Why can't we start from $\mathbf{W} = \mathbf{0}$?



$$\frac{\partial L}{\partial w_{22}^{h}} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{22}^{h}}$$

$$= \frac{\partial L}{\partial y}\frac{\partial}{\partial w_{22}^{h}}\left(w_{01}^{o} + w_{11}^{o}z_1 + w_{21}^{o}z_2\right)$$

$$= \frac{\partial L}{\partial y}\boxed{w_{21}^{o}}\frac{\partial z_2}{\partial w_{22}^{h}}$$

What if we start from $\mathbf{W} = \mathbf{1}$?

- If we initialize all the parameters by a constant, $\mathbf{z}_l$ will be a constant vector for all l

- Moreover, the gradient vectors will always be constant vectors since

$$\frac{\partial \mathbf{z}_{l+1}}{\partial \mathbf{z}_l} = \mathbf{W}_{l+1}\mathbf{f}'_{l+1}\left(\mathbf{b}_{l+1} + \mathbf{W}_{l+1}\mathbf{z}_l\right)$$

Other issues for weight initialization

- Small initialization may lead to vanishing gradients

- Large initialization may lead to exploding gradients

Xavier and He initialization

- Recall that

$$\frac{\partial y}{\partial w_j} = \sigma'\left(b + \sum_{j=1}^{p} w_j x_j\right) \cdot x_j$$

- Idea: find appropriate random initializations such that

$$\mathrm{E}\left(\partial \mathbf{z}_{j+1} / \partial \mathbf{z}_j\right) \approx 1 \quad \text{and} \quad \mathrm{Var}\left(\partial \mathbf{z}_{j+1} / \partial \mathbf{z}_j\right) \approx c$$

for some small c

Xavier and He initialization

- Use Xavier initialization with tanh activations
- Use He initialization with ReLU activations
- Both methods may not work well if the network is really deep

Batch normalization

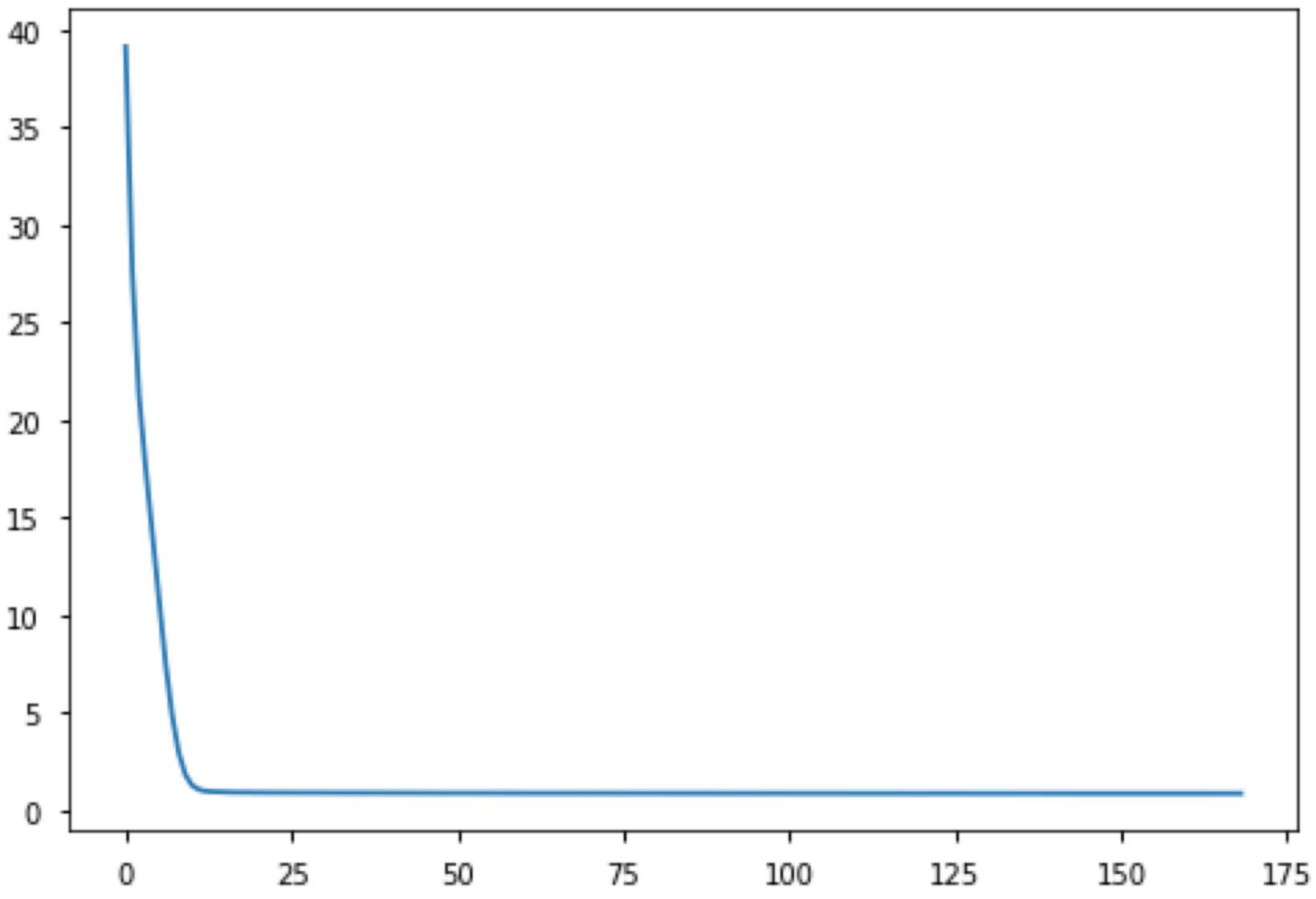
- Normalize $\mathbf{z}_j$ (the outputs of the j–th layer) for each j to avoid vanishing and exploding gradients

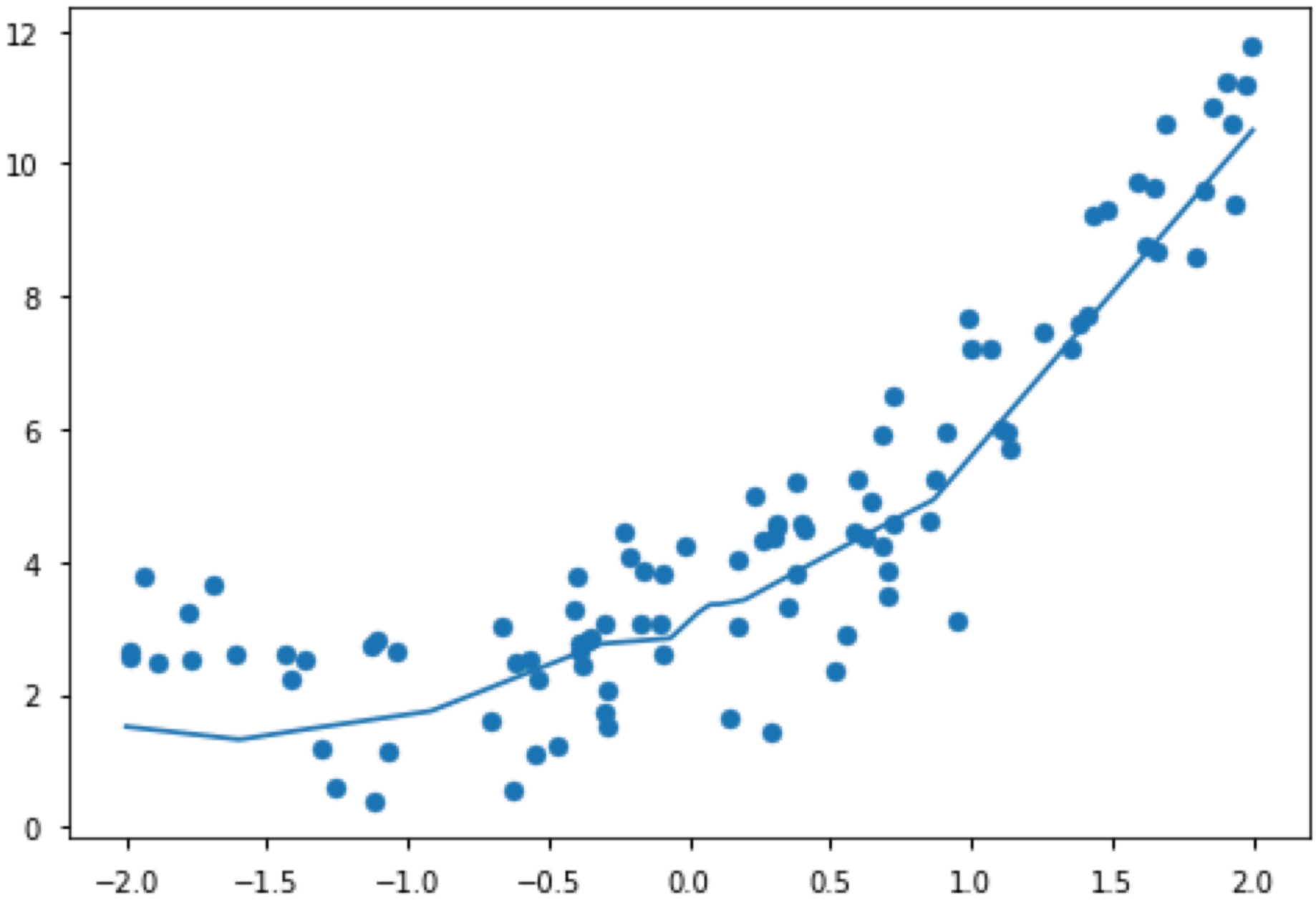
- The normalization is carried out batch–by–batch

- It also provides some regularizations; so don't use it with the other regularizations, e.g., dropout

```
initializer = tf.keras.initializers.HeNormal()
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(1,)))
model.add(layers.Dense(10, activation='relu', kernel_initializer=initializer))
model.add(layers.BatchNormalization())
model.add(layers.Dense(10, activation='relu', kernel_initializer=initializer))
model.add(layers.Dense(1, activation='linear'))
```

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-notebook')
plt.plot(history.history['loss'])
plt.show()
```




12
10
8
6
4
2
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

References

- Xavier initialization
- He initialization
- Batch normalization
- Batch normalization as regularization