CS 4365/5354
Deeper Learning
Exploding/vanishing gradient problem

Recall:
• \( y = \sigma(z) ; z = W_2 \text{relu}(W_1(\text{relu}(W_0 x + b_0) + b_1) + b_2) \)

For learning to be possible, we need small changes in \( x \) to correspond to small changes in \( z \) (similar examples should produce similar network outputs) and large changes in \( x \) to correspond to large changes in \( z \).
• In other words, we want \(|dz/dx| \approx 1|\)
Exploding/vanishing gradient problem

For simplicity, let’s assume relus work in linear part and ignore biases. Then

- $z = W_2 W_1 W_0 x$
- $\frac{dz}{dx} = W_2 W_1 W_0$
- $|\frac{dz}{dx}| = |W_2 W_1 W_0|$
Exploding/vanishing gradient problem

In case of a very deep network, this would be:

- \( \frac{dz}{dx} = W_n W_{n-1} \cdots W_2 W_1 W_0 \)
- If we initialize weights randomly, \(|W_n W_{n-1} \cdots W_2 W_1 W_0|\) is likely to be very large OR very small
- This is known as the exploding/vanishing gradient problem
Exploding/vanishing gradient problem

Solutions:

1. Initialization:
   - Choose the variance of the weights in such a way that:
     
     \[ |W_0 x| \approx |x|, \quad |W_1 W_0 x| \approx |W_0 x|, \quad |W_2 W_1 W_0 x| \approx |W_1 W_0 x|, \]
   - Several ways to do this. Weights can be chosen from uniform or normal distribution
     - Keras initializers: He, Glorot

2. Optimization:
   - Much of the problem is due to a single learning rate for all parameters. This is what conventional SGD does.
   - Modern algorithms with different learning rates for different parameters:
     - Keras optimizers: RMSprop, Adadelta, Adam
How deep can we go?

Combining Adam and He/Glorot initialization networks with up to 30 layers have been trained successfully.

Other ideas: data driven initialization - use first training batch to figure out initial weights

Can we go deeper?
Can we go deeper?

Batch Normalization
Normalize inputs and outputs for every batch
- Recall we want:
  \[ |W_0 x| \approx |x|, \ |W_1 W_0 x| \approx |W_0 x|, \ |W_2 W_1 W_0 x| \approx |W_1 W_0 x| \]
Batch Normalization (BN)

\[ x \xrightarrow{\text{layer}} \hat{x} = \frac{x - \mu}{\sigma} \xrightarrow{\gamma, \beta} y = \gamma \hat{x} + \beta \]

- \( \mu \): mean of \( x \) in mini-batch
- \( \sigma \): std of \( x \) in mini-batch
- \( \gamma \): scale
- \( \beta \): shift
- \( \mu, \sigma \): functions of \( x \), analogous to responses
- \( \gamma, \beta \): parameters to be learned, analogous to weights
Batch Normalization

Allowed networks to be trained with fewer training examples
Allowed deeper networks (up to 60 layers)
Going Deeper

Deep Residual Learning

- Plain net
  
  $x \rightarrow \text{weight layer} \rightarrow \text{relu} \rightarrow \text{weight layer} \rightarrow \text{relu} \rightarrow H(x)$

$H(x)$ is any desired mapping, hope the 2 weight layers fit $H(x)$
Going Deeper

Deep Residual Learning

- Residual net

\[ H(x) = F(x) + x \]

\[ F(x) \]

weight layer

relu

identity \( x \)

weight layer

relu

\( H(x) \) is any desired mapping,

hope the 2 weight layers fit \( H(x) \)

hope the 2 weight layers fit \( F(x) \)

let \( H(x) = F(x) + x \)