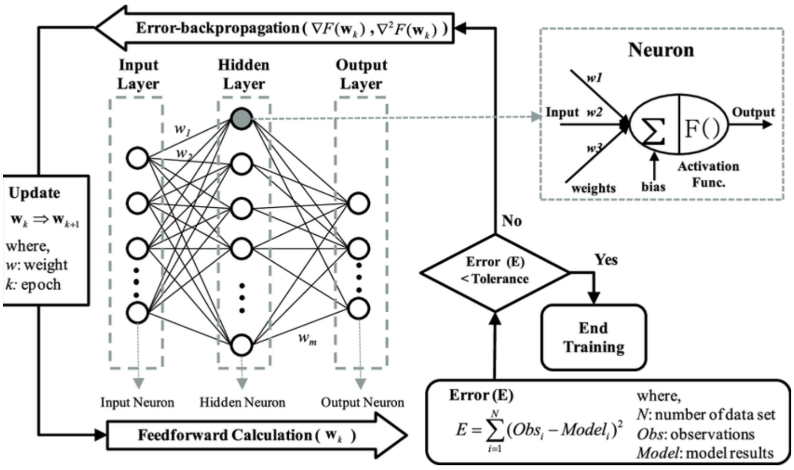
Vanishing Gradient & Exploding Gradient

- One of the problems of Gradient Desent
  - Gradient Desent （minimize loss function）
    - Composition:



      - Feedforword propagation to feed network with training dataset
      - Backpropagation to update the parameters (include weights, bias)
    - Types
      - Batch Gradient Desent
        - All samples of data are taken into consideration when computing gradient
        - Evaluation
          - Pros
            - computational efficiency, cuz updates only once in each epoch
            - the estimates of gradient is more accurate for using all data, getting a stable convergence
          - Cons
            - slower and hard to be fitted for the memory if the dataset is large
      - Stochastic Gradient Desent(SGD)
        - only a single row of data will be used in computation
        - Evaluation
          - Pros
            - faster in converging, cuz the updates happen more frequentaly
            - Considering only one single point, the updates of weights are nosiy, avoiding getting suboptimal local minimum
          - Cons
            - time consuming for the number of iterations is large

                        - the weight updates have noise, maybe not come to the local minimum actually
            - Mini-batch Gradient Desent
                - Divide training datasets into manageable groups and updates each separately
                - Evaluation: 结合SGD and Batch Gradient Desent.Give the balance of computational efficiency and durable.
        - Challenge
            - Difficult to find a global minimum for nonconvex issue
                - Solution: Adam
            - Vanishing and Exploding Gradient
                - Vanishing Gradient:
                    - the gradient keeps updating smaller and smaller, then comes to 0
                    - converge to a non-optimal solution too earlier. i.e.:never converge to the optimum
                - Exploding Gradient:
                    - gradient keeps on getting larger and larger, causing very large weight updates and causes the gradient descent to diverge
- Vanishing Gradient
    - Recognization(both forward/backward directions)
        - **parameters** of high layers change significantly, while that of low layers do not change so much
        - model **weights** become `0` during training
        - model learns slowly
- Exploding Gradient
    - Recognization(both forward/backward directions)
        - model parameters exponential growth
        - model weights become `NaN`
        - model gets a avalance learning
- Solution
    - 1. Proper Weight Initialization
        - If the inital weights are <u>too small or lacking variance</u>, it causes vanishing gradient.
        - Choose different weight initialization scheme for different activation functions
            - Strategies
                - The default is "zeros", but not to set all weights =0;
                - `glorot_uniform` for Tanh, Softmax, Logistic active functions
                - `He` for ReLU and its variants

            - `LeCun` for SELU
            - initalize randomly
            - don't initalize weight too large
    - 2. Using Different Activation Functions
        - if a gradient of an activation function is small, then following chain rule and multiplying the gradient to the input layer may cause vanishing. (like Sigmoid, tanh)
        - Hence, using non-saturating Activiation function is important. (eg: ReLU, Leaky ReLU, Parametric Leaky ReLU, Exponential Linear Unit(ELU), Scaled ELU(SELU))
        - Coding: `layer=keras.layers.Dense(10,activation="selu",kernel_initalizer="lecun_normal")`
    - 3. Using Different/faster Optimizers and Learning Rates
        - Faster Optimizers: Momentum optimization, Nesterov Acceler-ated Gradient, AdaGrad, RMSProp, Adam and Nadam optimization.
    - 4. Batch Normaliztion
        - Coding
            - `model=keras.models.Sequential([`
            - `keras.layers.Flatten(input_shape[28,28]),`
            - `keras.layers.BatchNormalization(),`
            - `keras.layers.Dense(300,activation='elu',kernel_initalizer='he_normal'),`
            - ...
            - Rule: Just add a BatchNormalization layer before or after each hidden layer's activationfunction, and optionally add a BN layer as well as the first layer in your model.
    - 4. Gradient Clipping for Exploding Gradient
        - clip the gradients during backpropagation so that they never exceed some threshold
        - clip every component of gradient (all the partial derivatives of the loss) to range: [-1.0, 1.0]
        - Coding: by setting `clipvalue` or `clipnorm`
            - `optimizer = keras.optimizers.SGD(clipvalue=1.0)`
            - `model.compile(loss="mse", optimizer=optimizer)`
    - 5. Reusing Pretrained Layers(Transfer Learning)
- References

    - How to deal with vanishing and exploding gradients? https://medium.com/geekculture/how-to-deal-with-vanishing-and-exploding-gradients-in-neural-networks-24eb00c80e84
    - The challenge of vanishing/exploding gradients in deep neural networks. https://www.analyticsvidhya.com/blog/2021/06/the-challenge-of-vanishing-exploding-gradients-in-de

ep-neural-networks/

- Gradient Descent Algorithms. https://medium.com/datadriveninvestor/gradient-descent-algorithm-b4c5afb4eb98
- Hands-On Machine Learning with Scikit-Learn and TensorFlow. https://www.knowledgeisle.com/wp-content/uploads/2019/12/2-Aur%C3%A9lien-G%C3%A9ron-Hands-On-Machine-Learning-with-Scikit-Learn-Keras-and-Tensorflow_-Concepts-Tools-and-Techniques-to-Build-Intelligent-Systems-O%E2%80%99Reilly-Media-2019.pdf