10417/10617
Intermediate Deep Learning:
Fall 2023

Yuanzhi Li / Russ Salakhutdinov

Machine Learning Department

Training neural networks

- Give a machine learning model h(W, x), where W is the parameter, x is the input.
- For MLPs: $h(W, x) = W_{L+1}\, \sigma(W_L \sigma(W_{L-1} \sigma \circ \cdots \circ \sigma(W_1 x + b_1) \; ... + b_{L-1}) + b_L) + b_{L+1}$
- For $W = (W_{L+1}, W_L, \dots, W_1, b_{L+1}, b_L, \dots, b_1)$.
- We train to $\min_W \frac{1}{N} \sum_{i \in [N]} l\left(h\left(W, x^{(i)}\right), y^{(i)}\right) + R(W)$
- We have learned how to compute the gradient of the objective.

Training neural networks

- We train to find the minimizer:
  $\min_W \frac{1}{N} \sum_{i \in [N]} l\left(h\left(W, x^{(i)}\right), y^{(i)}\right) + R(W)$
- We have learned how to compute the gradient of the objective.
- Can we train neural networks now?
- Answer: Yes, but it is going to be hard for the training to work on deep neural networks…

Training neural networks

- Biggest problems training (multi-layer) neural networks.
- $h(W, x) = W_{L+1}\, \sigma(W_L \sigma(W_{L-1} \sigma \circ \cdots \circ \sigma(W_1 x + b_1)\ \ldots + b_{L-1}) + b_L) + b_{L+1}$
- Key observation:
  - If $||W_l||_2 > 2$ for every l, then potentially $h(W, x) > 2^L$
  - If $||W_l||_2 < \frac{1}{2}$ for every l, then potentially $h(W, x) < 2^{-L}$
- The output of the neural network will blow up/shrink to zero unless $||W_l||_2$ is in a narrow “nice range”.

Output explosion/vanishing

One of the key difficulties of training a neural network is:

- The output of the neural network (or intermediate neurons) at a higher layer can easily explode (too large) or vanish (too small).
- The explosion/vanishing happens exponentially (in terms of layers).

This makes training deep neural networks quite difficult.

- We are going to learn several techniques to mitigate it, including normalization and residual links.

Output explosion/vanishing

- At some layer l, how do we maintain that $h_l(x)$ stays in a "healthy range"? Meaning that each coordinate of $h_l(x)$ is typically neither too large nor too small.
- The naïve solution: If σ is a sign function, then each coordinate of $h_l(x)$ is in {-1, 1} (good).
  - But when σ is a sign function, the gradient of the neural network is zero…
    - Recall $g_l = W_{l+1}^{\mathrm{T}} g_{l+1} \otimes \sigma'(z_l)$

Output explosion/vanishing

- We want each coordinate of $h_l(x)$ to be in a good range, like in {-1, 1}
- But we can’t use the sign activation function.
- Solution?
  - Normalization techniques.

Normalization techniques in deep learning

Layer normalization and Bach normalization.

Normalization techniques are what make deep learning training possible.

Layer normalization

- Key idea:
  - We want each coordinate of $h_l(x)$ to be in a good range, like {-1, 1}
  - But this is not doable in a differentiable manner.
- We relax it to be: The norm of $h_l(x)$ is 1.

Layer normalization

- Given a vector z, the layer-normalization layer is defined as:
- $LN(a, b, z) = a \otimes \frac{z}{||z||_2 + \varepsilon} + b$
  - Where a, b are two vectors, they are trainable parameters, z is the input. a is typically initialized at 1, b is initialized at 0. ε is fixed and typically very small, like 10^{-8} or 10^{-6}.
- We also use LN(z) to denote LN(a, b, z) for simplicity (to hide the trainable parameters – They are still there, but we just don't write them in the expression for notation simplicity).

Layer normalization

- $LN(a, b, z) = a \otimes \frac{z}{||z||_2 + \varepsilon} + b$
- In this way, as long as a is not too large/small and b is not too large, the norm of the output of LN(a, b, z) is in a good range for every z.
- $LN(z)$ is a differentiable function of z for every z.

Layer normalization

- As an example, we can use layer normalization in an MLP as:
- $h(W, x) = \ W_{L+1}\, LN \circ \sigma(W_L LN \circ \sigma(W_{L\,-1} LN \circ \sigma \circ \cdots \circ LN \circ \sigma(W_1 x + b_1)\ \ldots + b_{L\,-1}) + b_L) + b_{L+1}$
- In this way, the output norm of each hidden layer is in a good range (not exploding nor vanishing).

Batch normalization

- Layer normalization is great.
  - But still, the norm of the output is good could still lead to some bad configurations.
    - Only one neuron always outputs 1, all the other neurons output 0.
- What if I really want each coordinate of $h_l(x)$ to be in a good range, like {-1, 1}, instead of the norm of the entire layer?
- In this way, we enforce every neuron to be useful.

Batch normalization

- Batch normalization ensures that “every neuron is useful”.
- Given a batch of n inputs $z_1, z_2, \ldots, z_n$ in R,
- Batch normalization operation BN is defined as:
- $BN(z_i) = a \times \frac{z_i - mean(\{z_1,\ldots,z_n\})}{std(\{z_1,\ldots,z_n\}) + \varepsilon} + b$
- Where a, b are trainable real values, ε is fixed.

Batch normalization

- $BN(z_i) = a \times \frac{z_i - mean(\{z_1,\ldots,z_n\})}{std(\{z_1,\ldots,z_n\}) + \varepsilon} + b$
- BN is a differentiable function of each z_i.
- If a = 1 and b = 0, it ensures that the variance of $\{BN(z_1), \ldots, BN(z_n)\}$ is 1 and mean is 0.
  - Almost like the output of $BN(z_i)$ is in {-1, 1}.

Batch normalization

- To use Batch normalization in a neural network:
- For example, if we apply batch-normalization to a neuron n(x)
  - Given a batch of input $x^{(1)}, x^{(2)}, \dots, x^{(n)}$:
  - $BN\left(n\left(x^{(i)}\right)\right) = a\frac{n\left(x^{(i)}\right) - mean\left\{n\left(x^{(j)}\right)\right\}_{j\in[n]}}{std\left(\left\{n\left(x^{(j)}\right)\right\}_{j\in[n]}\right) + \varepsilon} + b$
- So, we can also use:
- $h(W, x) = \ W_{L+1}\, BN \circ \sigma(W_L BN \circ \sigma(W_{L-1} BN \circ \sigma \circ \cdots \circ BN \circ \sigma(W_1 x + b_1) \ \dots + b_{L-1}) + b_L) + b_{L+1}$

Batch normalization versus layer normalization

Batch normalization ensures the output of each neuron has a good variance. While layer normalization only ensures the output of the layer has a good norm. (Batch normalization wins).

Batch normalization requires batch input, which means you can only use batch normalization with a relatively large training batch. So it’s more memory intensive (Batch normalization loses).

Batch normalization is typically used in CNN (Convolution neural networks), layer normalization is typically used in transformers.

Residual Link

Now we make sure the output of each neuron/layer in the neural network is good…

Can we train neural networks now?

Residual Link

Can we train neural networks now?

We can, but it still won’t be good…

Recall what we want a neural network to do.

We want a neural network to perform hierarchical feature learning.

Residual Link

Hierarchical Feature Learning:

For example, to learn advanced calculus.

- We want the first layer of neural network to learn basic number arithmetics.
- The second layer to learn variable arithmetics.
- The third layer to learn matrix arithmetics.
- The fourth layer to learn tensor arithmetics…

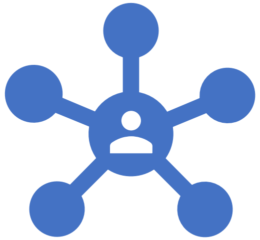
Key observation: Even tensor arithmetics rely on basic number arithmetics!

- The fourth layer relies on the features of the first layer.

Residual Link

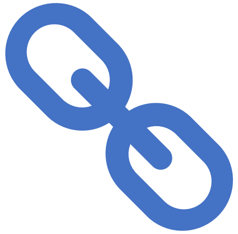
- In hierarchical feature learning,
  - The higher layer often directly relies on the features of the very low layers.
- $h(W, x) = \ W_{L+1}\, \sigma(W_L \sigma(W_{L\,-1} \sigma \circ \cdots \circ \sigma(W_1 x + b_1)\ \ldots + b_{L\,-1}) + b_L) + b_{L+1}$
- Directly accessing the features in very low layers from very high layers is not that easy…
  - There's so much non-linearity in between.

Residual Link



When neural network is performing Hierarchical Feature Learning:

Can ensure the higher layers can directly access the features of the (much) lower layers?

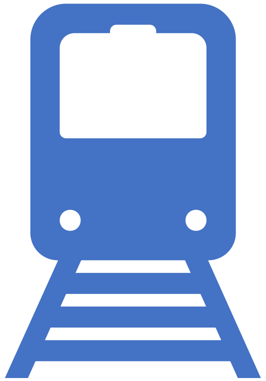


Solution: Residual link.

Residual Link

- Residual link: Replace the basic block of MLP from $\sigma(Wz + b)$
- To $z + V\sigma(Wz + b)$
- Original MLP:
  - $h_l(x) = \sigma(W_l h_{l-1}(x) + b_l)$
- MLPs with Residual link:
  - $h_l(x) = h_{l-1}(x) + V_{l-1}\sigma(W_{l-1}h_{l-1}(x) + b_{l-1})$

One last trick



Now, can we train neural networks???????????????????????????

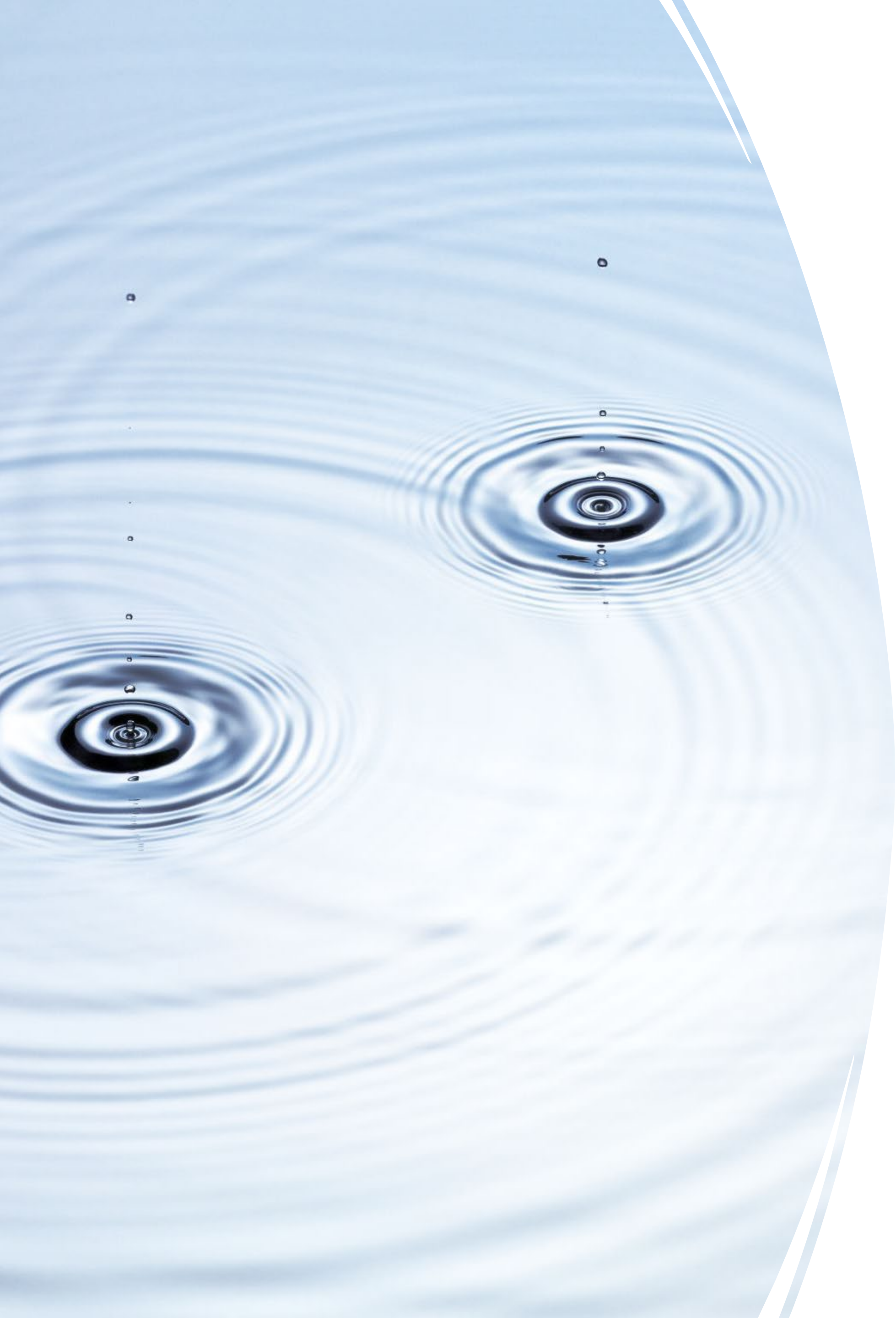
Yes, we finally can, but there’s one additional trick that helps training.

Dropout

- Let us consider an one-hidden-layer MLP $h(x) = \sum_i a_i \sigma(w_i^T x + b_i)$
- Key problem during training: Mode collapsing.
- At anytime during training, whenever $a_i = a_j, w_i = w_j, b_i = b_j$.
  - Then $a_i = a_j, w_i = w_j, b_i = b_j$ forever afterwards during training.
    - If we use gradient based method.
- This is because these two neurons will have the same gradient at any iteration afterwards.

Dropout

- At anytime during training, whenever $a_i = a_j, w_i = w_j, b_i = b_j$.
  - Then $a_i = a_j, w_i = w_j, b_i = b_j$ forever afterwards during training.
- This is because these two neurons will have the same gradient at any iteration afterwards.
- Can we save it?
  - Dropout.



Dropout

- For a vector $z \in R^d$, the dropout layer is defined as:
- Dropout(z) = $z \otimes \tau, where\ \tau \in \{0,1\}^d$ is a random variable, each coordinate is i.i.d.
  - $\Pr[\tau_i = 0] = p$
  - τ is not trainable, but sampled randomly at every training batch.
- We can apply dropout like:
  - MLP $h(x) = \sum_i a_i Dropout(\sigma(w_i^T x + b_i))$
  - $h(x)$ is a randomized function.

Dropout

- We can apply dropout like:
  - MLP $h(x) = \sum_i a_i Dropout(\sigma(w_i^T x + b_i))$
  - $h(x)$ is a randomized function.
- Even if $a_i = a_j, w_i = w_j, b_i = b_j$.
- Their gradient might still be different, since τ_i can be different from τ_j

Dropout Training

- To train using dropout on, for example, $h(x) = \sum_i a_i Dropout(\sigma(w_i^T x + b_i))$
- At every iteration, for each $x^{(j)}$, we randomly sample a $\tau^{(j)}$, and obtain function $h^{(j)}(x^{(j)}) = \sum_i a_i \tau_i^{(j)} \sigma(w_i^T x^{(j)} + b_i)$
- Compute the gradient of W for $L(h^{(j)}(x^{(j)}), y^{(j)})$
- Update using this gradient.