Last time

- convolutional layers
- pooling layers

} capturing spatial structures

- smaller parameter counts vs dense layers

Today

- other important types of layers
  - BatchNormalization
  - Dropout

  } → different behavior at train & test time

  - Residual Layer

→ can learn to be removed during training

Batch Normalization

- a "special" layer type which does not follow the linear + nonlinear piece recipe

- is usually applied either: (1) after the linear piece and before the non-linear piece of another layer, or (2) after the non-linear piece of another layer.

$$(1)\ \sigma(\mathrm{BatchNorm}(Z(X)))$$
$$(2)\ \mathrm{BatchNorm}(\sigma(Z(X)))$$

- was introduced as a fix to the gradient explosion / gradient vanishing problem, but is perhaps best thought of as a noising / regularizing layer.

- Idea: center and scale the activations of a layer to prevent the drift of this distribution to extremes during training OR reparameterize the activations of each observation as measuring the relative differences between it and members of the batch.

input: $X = [n, p]$ ← e.g. output of previous layer

n ← batch size

Behavior during training:

(1) Obtain the batch mean $\bar{x}_B$ and batch variance s^2_B feature-wise:

$$\bar{x}_B = \frac{1}{n} \sum_{i=1}^{n} x_i \in \mathbb{R}^p$$

⚠ highly dependent on batch size

$$s^2_B = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x}_B)^2$$

(2) Update the moving average estimators of the population mean and variance of activations:

$$\hat{\mu} = \lambda \hat{\mu} + (1-\lambda)\, \bar{x}_B$$

$$\hat{\sigma}^2 = \lambda \hat{\sigma}^2 + (1-\lambda)\, s^2_B$$

λ is the momentum hyperparameter ($\sim .99$)

(3) Calculate normalized activations:

$$\tilde{x}_i = \left(\frac{x_i - \bar{x}_B}{\sqrt{s^2_B + \varepsilon}} \right)$$

(4) learned scale and shift :

$$z_i = \gamma \tilde{x}_i + \beta$$

where $\gamma \in \mathbb{R}^p$, $\beta \in \mathbb{R}^p$ are trainable parameters, initialized to 1 and 0 respectively

→ allows the network to "undo" the scaling

Behavior during testing :

$$z_i = \gamma \left(\frac{x_i - \hat{u}}{\sqrt{\hat{\sigma}^2 + \varepsilon}} \right) + \beta$$

→ replace batch mean and variance estimates with estimates of population statistics.

→ why moving average? distribution may change over time during training

Related layers

→ LayerNorm

Dropout

- a "special" layer that does not have the usual linear → nonlinear structure
- is usually applied before a Batch Normalization layer e.g. BatchNorm(Dropout($\sigma(Z(X))$))
- Idea: Noise a layer's outputs by "dropping" each neuron (i.e. setting its value to zero) for each observation with a probability d.

Behavior during training:

element-wise multiplication

$$\text{Dropout}_d(X) = \underset{[n \times p]}{X} \cdot \underset{[n \times p]}{M}$$

$$M_{ij} = \mathbb{1}\{U_{ij} > d\}, \quad U_{ij} \sim \text{uniform}(0,1)$$

Behavior during testing:

$$\text{Dropout}_d(X) = X \cdot (1-d)$$

→ want the expected magnitudes of the outputs to be the same during training and testing

Behavior during training (v2):

$$\text{Dropout}_d(X) = X \cdot M / (1-d)$$

Behavior during testing (v2):

$$\text{Dropout}_d(X) = X$$

→ avoids one computation during inference (where latency can be a big issue)

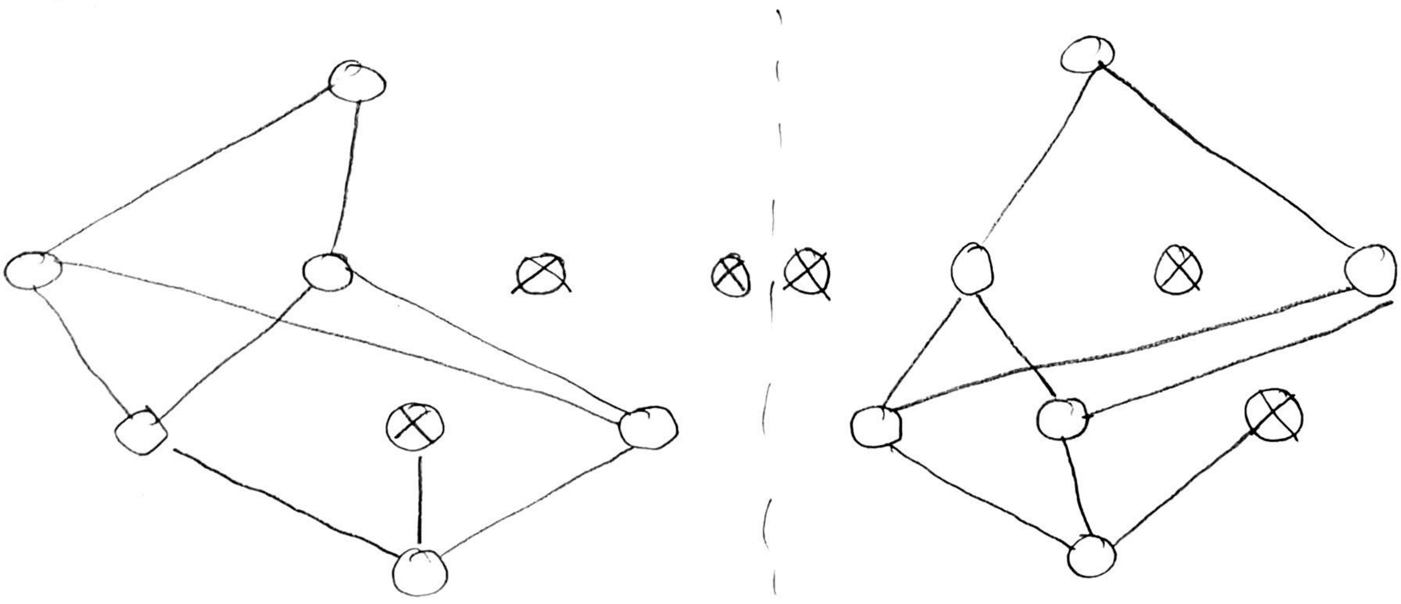
Related: Stochastic Depth (layer drop)
DropConnect (weight drop)

→ Interpretations:

→ prevent over-reliance on any particular feature

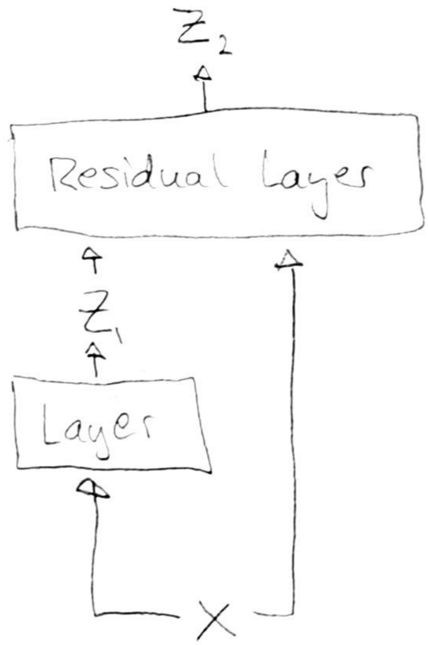
→ encourage utilization of all features

→ learn features that are useful for all sub-networks of the trained model



Residual Layer

→ <u>Idea</u>: Enable a model to choose / learn the optimal number of layers needed during training.



$$Z_2 = Z_1 + X$$ (where $+$ is element-wise addition)

→ <u>Note</u>: Z_1 and X must have the same shape

→ A key innovation which enabled training of very deep (>100 layer) networks, overcoming vanishing / exploding gradient problem.