COMS 6730 **Spring 2025** **Name: Mobina Amrollahi**

CheatSheet

Linear Models

1. **Linear regression** is a linear model, trying to predict a **continuous scaler**.
2. **RSS** in linear regression minimizes the sum of the squared distances between the label of each data point and the predicted value.
3. The weights can be found by **maximizing the likelihood of the data** under the assumption of Gaussian noise, which is equivalent to **minimizing the RSS**.
4. Finding the weights by solving the normal equations might not be possible as $X^T X$ might not exist.
5. If X, the feature matrix, is **full rank**, then the optimal solution for linear regression exists and is unique.
6. three assumptions of linear regression: 1. Data is linear, 2. Data points are independent, 3. The residual follows normal distribution with zero mean.
7. The perceptron learning algorithm guarantees convergence for the dataset that is **linearly separable**, regardless of the initial data or weights.
8. **Logistic regression** is a linear model because the decision boundary is a linear function of the input features.
9. The **monotonicity** of the sigmoid function, ensures the logistic regression produce a linear decision boundary. By changing sigmoid to sine for inastance, logistic regression becomes a non-linear model.

Gradient Descent Algorithm

1. Batch size affect both the **forward** and **backward** propagation.

Convolution

1. Kernel is a small matrix. They're typically small and scan across the image.
2. Feature maps: the result of applying filters to the input.
3. The primary purpose of a convolution layer is to extract spatial features from the input.
4. A convolution operation multiplies kernel and input element-wise, then sums them up.

2.

Batch	**Mini - Batch**	**Stochastic**
• process the entire dataset at once • smoother, more accurate gradient estimates • computationally expensive at practice	• helps escape local minima during optimization • introduces noise into the optimization process • small to medium batch sizes can lead to better generalization	• update weights using one sample at a time

Table 1: Comparison between Different Algorithms

Box Filter	**Sharpening Filter**	**Sobel (Vertical Edge)**	**Sobel (Horizontal Edge)**
$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
• Averages neighboring pixels • Smooths the image • Reduces noise • Removes sharp features	• Enhances edges • Emphasizes differences with neighbors • Increases sharpness • High-pass filter	• Detects vertical edges • Computes horizontal gradients • Sensitive to vertical features • Emphasizes vertical boundaries	• Detects horizontal edges • Computes vertical gradients • Sensitive to horizontal features • Emphasizes horizontal boundaries

Table 2: Comparison of Different Filters and Their Properties

5. Hand-Crafted Kernels:
6. Despite hand-crafted filters, the learned ones are randomly initialized, and then optimized through back-propagation.
7. Convolution uses a set of kernels, each applied to an individual input channel.
8. The main effect of **stride** is that it reduces the output feature map.
9. The main effect of **padding** is to ensure all pixels are equally used.
10. Number of trainable parameters in each convolution layer: $K \times K \times C_{in} \times C_{out}$.

11. Number of multi-add operations or computational cost in each convolution layer: $K \times K \times C_{in} \times C_{out} \times W' \times D'$.

12. **Dilated convolutions** enlarges the receptive field by introducing gaps (holes) in the kernel to cover a larger area. For a dilation "rate" d, $d-1$ spaces are inserted between kernel elements such that $d = 1$ corresponds to a regular convolution.

13. Output dimension: $W' = \frac{W-K+2P-(k-1)(d-1)}{S} + 1$.

14. Receptive Field: the region of the input space that affects a particular unit in the network.

$$\forall \quad l : R_l = R_{l-1} + (k_l - 1) \times J_{l-1} \times d_l$$

The jump J_l describes how far we move in the input when moving one unit in the feature map:

$$J_l = J_{l-1} \times S_l, \quad \text{and} \quad J_0 = 1, \quad R_0 = 1$$

where

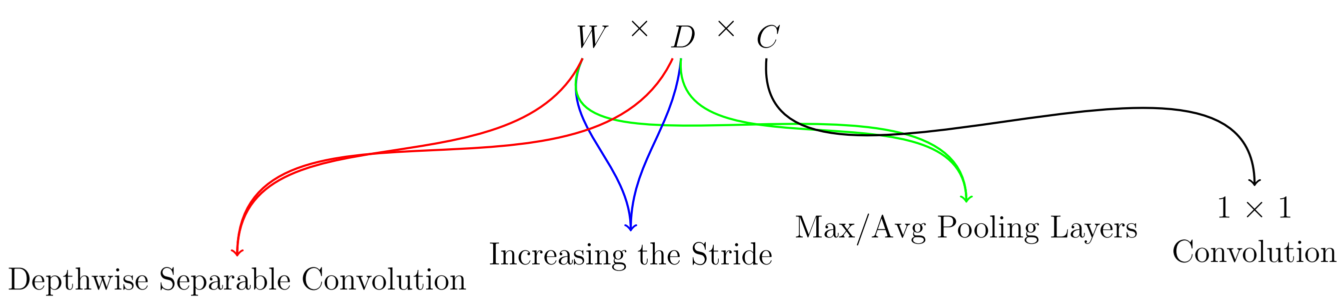
- R_l: receptive field size at layer l
- k_l: kernel size at layer l
- J_{l-1}: jump (effective stride) from the previous layer
- S_l: stride at layer l
- d_l: dilation rate at layer l

In which, you can see for a network with one layer, only **the kernel size** can affect the receptive field. However, once the number of kernels increases, the **stride** and **pooling layers** also affect it.

15. 1×1 Convolution: is used to reduce the number of channels (dimensionality) while introducing non-linearity.

16. **Pooling layers** reduce the output feature maps, while avoid overfitting, and enlarging the receptive field.

17. Solving the high dimension problem:



18. Activation functions are element-wise function that introduce non-linearity. Tanh and sigmoid have gradient vanishing problems, meanwhile ReLU has the problem of **dying neurons**. One attempt to fix the dying neurons problem is to use **leaky ReLU**.

19. **Nearest Neighbors**, **Bi-Linear Interpolation**, **Bed of Nails**, and **Max-Unpooling** are all **deterministic** up sampling techniques, meanwhile transposed convolution is learnable.

20. **Bed of Nails** is followed by a convolution layer that can interpolate or blend the sparse data points to generate meaningful and smooth output.

21. **Max-Unpooling** it's always following a corresponding max-pooling layer.

22. Transposed convolution is often **incorrectly** called **de-convolution**. However de-convolution refers to the inverse operation of standard convolution. The name **transposed** corresponds to multiplying by the transpose of the convolution kernel matrix. Transposed convolutions swap the forward and backward passes of a convolution. **Output size formula**:

$$W_{out} = S \times (W - 1) + K - 2P$$

23. In transposed convolution if $s > 1$ we will put $s - 1$ zeros between input elements and in the boarders.

24. Unlike deep convolutional networks, **attention** mechanisms can capture long-range dependencies in a single layer. They're called global extractors, because each query attends all the keys and values.

25. **Scaled Dot-Product Attention:**

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

   - $Q \in \mathbb{R}^{n \times d}$
   - $K \in \mathbb{R}^{m \times d}$
   - $V \in \mathbb{R}^{m \times p}$
   - d_k: Dimensionality of keys (used for scaling)

26. **Self-Attention with a Single Input Matrix X:**

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

$$\text{Attention}(X) = \text{Softmax}\left(\frac{(XW_Q)(XW_K)^T}{\sqrt{d_k}}\right) (XW_V)$$

   - When using self-attention, X is projected into Q, K, V using trainable weight matrices.
   - This allows each token in X to attend to others, learning dependencies across positions.

27. **Convolution** and **attention** layers can be interpreted as variations of fully-connected layers, since they're all matrix multiplications. However, pooling layers (e.g., max pooling, average pooling) perform non-parametric reduction operations and are not typically expressed as matrix multiplications.

Convolution Neural Network

1. Winners of ImageNet Comparison:

AlexNet	VGG	ResNet	DenseNet
• 8 layers • ReLU Activation Function • High number of kernels at each layer • Different kernel sizes of $11 \times 11, 5 \times 5, 3 \times 3$. • Introduced Dropout • Trained with Data Augmentation • Overlap pooling layer	• 19 Layers • ReLU Activation Function • Fixed pattern of conv + conv + pool at each layer • Fixed kernel size of 3×3. • Max pooling 2×2 with stride = 2 • Same padding everywhere • Has a degradation problem: vanishing gradients or explosion • Prone to Overfit	• 152 Layers • Skip connection: preventing the gradient vanishing • Batch Normalization: preventing the gradient explosion • Each residual network: 3×3 conv + Batch Normalization + ReLU + 3×3 conv + Batch Normalization	• Concatenates the outputs from different layers. • Computes vertical gradients • Uses more memory compare to ResNet

Table 3: Comparison of Different Deep Learning Architectures

2. In **DenseNet**, the l-th layer receives as input the concatenation of all feature maps produced by previous layers:

$$x_l = H_l([x_0, x_1, x_2, ..., x_{l-1}])$$

Where:

- $[x_0, x_1, ..., x_{l-1}]$ represents the concatenation of feature maps along the channel dimension.
- $H_l(\cdot)$ is the composite function of operations: BatchNorm, ReLU, and Convolution.
- If the growth rate is k, the input depth to layer l is:

$$d_{in} = d_0 + k \times (l - 1)$$

3. In **Transfer Learning**:
   (a) Train a network like VGG or ResNet on a large dataset
   (b) Freeze the earlier convolution layers weights
   (c) Replace the fully connected layer with new layer specific to the task
4. Weight Initialization in CNNs:
5. **Batch Normalization**: During training, the mean and variance are computed from each mini-batch. During testing, the moving average of the mean and variance, calculated during training, is used instead.
6. **Batch size** has an impact in batch normalization.
7. During the prepossessing we want the **input data** to be **normalized** and **zero-centered.**

Xavier Initialization	**Kaiming Initialization**	**Random Initialization**
• $W \sim \mathcal{N}\left(0, \frac{2}{n_{in}+n_{out}}\right)$ • Works well with sigmoid and tanh activations.	• $W \sim \mathcal{N}\left(0, \frac{2}{n_{in}}\right)$ • Best for ReLU and Leaky ReLU activations.	• $W \sim \mathcal{N}(0, 10^{-2})$ • Works on small networks • still have problems on deep networks • Weight gradients in deeper layers have variances of nearly zero

Table 4: Weight Initialization

8. **Data augmentation** increases the diversity of the training dataset, forcing the model to generalize better. However, the labels of the augmented data remain the **same**.

9. During training, **Dropout** randomly deactivates neurons with probability p, which lowers the expected output. To maintain consistency between training and testing:

   - **Inverted Dropout (common)**: During training, scale the output of active neurons by $\frac{1}{1-p}$.
   - **Standard Dropout**: During training, no scaling is applied. During testing, scale the activations by $(1-p)$.

10. Since in the drop-out neurons will randomly be set to zero, the number of trainable parameters won't change.

11. Drop-out is only applied to the **hidden** layer, and not the output layer.