Outline

ConvNets as Deep Neural Networks for Vision

1. Neural Nets
2. Deep Convolutional Neural Networks
The Formal Neuron: 1943  [MP43]

- Basis of Neural Networks
- Input: vector $\mathbf{x} \in \mathbb{R}^m$, i.e. $\mathbf{x} = \{x_i\}_{i \in \{1, 2, \ldots, m\}}$
- Neuron output $\hat{y} \in \mathbb{R}$: scalar
Mapping from \( x \) to \( \hat{y} \):
1. Linear (affine) mapping: \( s = \mathbf{w}^\top \mathbf{x} + b \)
2. Non-linear activation function: \( f: \hat{y} = f(s) \)
The Formal Neuron: Linear Mapping

- Linear (affine) mapping: \( s = \mathbf{w}^\top \mathbf{x} + b = \sum_{i=1}^{m} w_i x_i + b \)
  - \( \mathbf{w} \): normal vector to an hyperplane in \( \mathbb{R}^m \) ⇒ linear boundary
  - \( b \): bias, shift the hyperplane position

2D hyperplane: line

3D hyperplane: plane

\[ w^t x + b = 0 \]
The Formal Neuron: Activation Function

- \( \hat{y} = f(\mathbf{w}^\top \mathbf{x} + b) \), \( f \) activation function
  - Popular \( f \) choices: step, sigmoid, tanh

- Step (Heaviside) function: 
  \[
  H(z) = \begin{cases} 
    1 & \text{if } z \geq 0 \\
    0 & \text{otherwise}
  \end{cases}
  \]
Formal neuron, step activation $H$: $\hat{y} = H(w^T x + b)$
- $\hat{y} = 1$ (activated) $\iff w^T x \geq -b$
- $\hat{y} = 0$ (unactivated) $\iff w^T x < -b$

**Biological Neurons:** output activated
$\iff$ input weighted by synaptic weight $\geq$ threshold
Sigmoid Activation Function

- Neuron output $\hat{y} = f(\mathbf{w}^\top \mathbf{x} + b)$, f activation function
- Sigmoid: $\sigma(z) = (1 + e^{-az})^{-1}$

- $a \uparrow$: more similar to step function (step: $a \to \infty$)
- Sigmoid: linear and saturating regimes
The Formal neuron: Application to Binary Classification

- Binary Classification: label input $x$ as belonging to class 1 or 0
- Neuron output with sigmoid: $\hat{y} = \frac{1}{1 + e^{-a(w^T x + b)}}$
- Sigmoid: probabilistic interpretation $\Rightarrow \hat{y} \sim P(1/x)$
  - Input $x$ classified as 1 if $P(1/x) > 0.5 \iff w^T x + b > 0$
  - Input $x$ classified as 0 if $P(1/x) < 0.5 \iff w^T x + b < 0$
  $\Rightarrow \text{sign}(w^T x + b)$: linear boundary decision in input space!
The Formal neuron: Toy Example for Binary Classification

- 2d example: \( m = 2, \ x = \{ x_1, x_2 \} \in [-5; 5] \times [-5; 5] \)
- Linear mapping: \( w = [1; 1] \) and \( b = -2 \)
- Result of linear mapping: \( s = w^T x + b \)
The Formal neuron: Toy Example for Binary Classification

- 2d example: \( m = 2, \mathbf{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5] \)
- Linear mapping: \( \mathbf{w} = [1; 1] \) and \( b = -2 \)
- Result of linear mapping: \( s = \mathbf{w}^T \mathbf{x} + b \)
- Sigmoid activation function: \( \hat{y} = \left(1 + e^{-a(\mathbf{w}^T \mathbf{x} + b)}\right)^{-1}, \quad a = 10 \)
The Formal neuron: Toy Example for Binary Classification

- 2d example: \( m = 2, \mathbf{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5] \)
- Linear mapping: \( \mathbf{w} = [1; 1] \) and \( b = -2 \)
- Result of linear mapping: \( s = \mathbf{w}^\top \mathbf{x} + b \)
- Sigmoid activation function: \( \hat{y} = \left(1 + e^{-a(\mathbf{w}^\top \mathbf{x} + b)}\right)^{-1}, \quad a = 1 \)
The Formal neuron: Toy Example for Binary Classification

- 2d example: \( m = 2, \mathbf{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5] \)
- Linear mapping: \( \mathbf{w} = [1; 1] \) and \( b = -2 \)
- Result of linear mapping: \( s = \mathbf{w}^\top \mathbf{x} + b \)
- Sigmoid activation function: \( \hat{y} = \left(1 + e^{-a(\mathbf{w}^\top \mathbf{x} + b)}\right)^{-1} \), \( a = 0.1 \)
From Formal Neuron to Neural Networks

- Formal Neuron:
  1. A single scalar output
  2. Linear decision boundary for binary classification

- Single scalar output: limited for several tasks
  - Ex: multi-class classification, e.g. MNIST or CIFAR
Perceptron and Multi-Class Classification

- **Formal Neuron**: limited to binary classification
- **Multi-Class Classification**: use several output neurons instead of a single one! ⇒ **Perceptron**

- Input $x$ in $\mathbb{R}^m$
- Output neuron $\hat{y}_1$ is a formal neuron:
  - Linear (affine) mapping: $s_1 = w_1^T x + b_1$
  - Non-linear activation function: $f$: $\hat{y}_1 = f(s_1)$
- Linear mapping parameters:
  - $w_1 = \{w_{11}, \ldots, w_{m1}\} \in \mathbb{R}^m$
  - $b_1 \in \mathbb{R}$
Perceptron and Multi-Class Classification

- Input $\mathbf{x}$ in $\mathbb{R}^m$
- Output neuron $\hat{y}_k$ is a formal neuron:
  - Linear (affine) mapping: $s_k = \mathbf{w}_k^T \mathbf{x} + b_k$
  - Non-linear activation function: $f$: $\hat{y}_k = f(s_k)$
- Linear mapping parameters:
  - $\mathbf{w}_k = \{w_{1k}, \ldots, w_{mk}\} \in \mathbb{R}^m$
  - $b_k \in \mathbb{R}$
Perceptron and Multi-Class Classification

- Input $x$ in $\mathbb{R}^m$ ($1 \times m$), output $\hat{y}$: concatenation of $K$ formal neurons
- Linear (affine) mapping $\sim$ matrix multiplication: $s = xW + b$
  - $W$ matrix of size $m \times K$ - columns are $w_k$
  - $b$: bias vector - size $1 \times K$
- Element-wise non-linear activation: $\hat{y} = f(s)$
Perceptron and Multi-Class Classification

- **Soft-max Activation:**
  \[ \hat{y}_k = f(s_k) = \frac{e^{s_k}}{\sum_{k'=1}^{K} e^{s_{k'}}} \]

- **Probabilistic interpretation for multi-class classification:**
  - Each output neuron ⇔ class
  - \( \hat{y}_k \sim P(k|x, w) \)

  ⇒ Logistic Regression (LR) Model!
2d Toy Example for Multi-Class Classification

- \( \mathbf{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5] \), \( \hat{\mathbf{y}} \): 3 outputs (classes)

Linear mapping for each class:
\[ s_k = \mathbf{w}_k^\top \mathbf{x} + b_k \]

Soft-max output:
\[ P(k|x, \mathbf{W}) \]
2d Toy Example for Multi-Class Classification

- \( \mathbf{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5] \), \( \hat{y} \): 3 outputs (classes)

Soft-max output: 
\[ P(k/\mathbf{x}, \mathbf{W}) \]

Class Prediction:
\[ k^* = \arg \max_k P(k/\mathbf{x}, \mathbf{W}) \]
Beyond Linear Classification

X-OR Problem

- Logistic Regression (LR): NN with 1 input layer & 1 output layer
- LR: limited to linear decision boundaries
- **X-OR**: NOT 1 and 2 OR NOT 2 AND 1
  - **X-OR**: Non linear decision function
Beyond Linear Classification

- **LR**: limited to linear boundaries
- **Solution**: add a layer!

- Input $x$ in $\mathbb{R}^m$, e.g. $m = 4$
- Output $\hat{y}$ in $\mathbb{R}^K$ ($K$ classes), e.g. $K = 2$
- **Hidden layer $h$ in** $\mathbb{R}^L$
Multi-Layer Perceptron

- **Hidden layer** $h$: $x$ projection to a new space $\mathbb{R}^L$
- Neural Net with $\geq 1$ hidden layer: Multi-Layer Perceptron (MLP)
- $h$: intermediate representations of $x$ for classification $\hat{y}$: $h = f(xW + b)$
- Mapping from $x$ to $\hat{y}$: non-linear boundary! $\Rightarrow$ activation $f$ crucial!
Deep Neural Networks

- Adding more hidden layers: Deep Neural Networks (DNN) ⇒ Basis of Deep Learning
- Each layer $h^i$ projects layer $h^{i-1}$ into a new space
- Gradually learning intermediate representations useful for the task
Conclusion

- Deep Neural Networks: applicable to classification problems with non-linear decision boundaries

- Visualize prediction from fixed model parameters

- Reverse problem: **Supervised Learning**
Outline

Neural Networks

Training Deep Neural Networks
Training Multi-Layer Perceptron (MLP)

- Input $x$, output $y$
- A parametrized ($w$) model $x \Rightarrow y$: $f_w(x_i) = \hat{y}_i$
- Supervised context:
  - Training set $\mathcal{A} = \{(x_i, y_i^*)\}_{i \in \{1,2,\ldots,N\}}$
  - Loss function $\ell(\hat{y}_i, y_i^*)$ for each annotated pair $(x_i, y_i^*)$
  - Goal: Minimizing average loss $\mathcal{L}$ over training set: $\mathcal{L}(w) = \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{y}_i, y_i^*)$
- Assumptions: parameters $w \in \mathbb{R}^d$ continuous, $\mathcal{L}$ differentiable
- Gradient $\nabla_w = \frac{\partial \mathcal{L}}{\partial w}$: steepest direction to decrease loss $\mathcal{L}(w)$
MLP Training

- Gradient descent algorithm:
  - Initialize parameters $\mathbf{w}$
  - Update: $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$
  - Until convergence, e.g. $\|\nabla \mathbf{w}\|^2 \approx 0$
Gradient Descent

Update rule: $w^{(t+1)} = w^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial w}$ \(\eta\) learning rate

- **Convergence ensured**? $\Rightarrow$ provided a "well chosen" learning rate $\eta$

![Graph showing convergence and divergence](image-url)
Gradient Descent

Update rule: \[ w^{(t+1)} = w^{(t)} - \eta \frac{\partial L}{\partial w} \]

- **Global minimum ?**
  ⇒ convex a) vs non convex b) loss \( L(w) \)

![Diagram](image.png)

- a) Convex function
- a) Non convex function
Supervised Learning: Multi-Class Classification

- Logistic Regression for multi-class classification
- $s_i = x_i W + b$
- Soft-Max (SM): $\hat{y}_k \sim P(k|x_i, W, b) = \frac{e^{s_k}}{\sum_{k'=1}^{K} e^{s_{k'}}}$
- Supervised loss function: $\mathcal{L}(W, b) = \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{y}_i, y_i^*)$

1. $y \in \{1; 2; \ldots; K\}$
2. $\hat{y}_i = \arg\max_k P(k|x_i, W, b)$
3. $\ell_{0/1}(\hat{y}_i, y_i^*) = \begin{cases} 1 & \text{if } \hat{y}_i \neq y_i^* \\ 0 & \text{otherwise} \end{cases}$: 0/1 loss
Logistic Regression Training Formulation

- Input $x_i$, ground truth output supervision $y^*_i$
- One hot-encoding for $y^*_i$:
  
  $y^*_{c,i} = \begin{cases} 
  1 & \text{if } c \text{ is the ground truth class for } x_i \\
  0 & \text{otherwise}
  \end{cases}$
Logistic Regression Training Formulation

- Loss function: multi-class Cross-Entropy (CE) $\ell_{CE}$
- $\ell_{CE}$: Kullback-Leiber divergence between $y_i^*$ and $\hat{y}_i$

$$\ell_{CE}(y_i, y_i^*) = KL(y_i^*, \hat{y}_i) = -\sum_{c=1}^{K} y_{c,i}^* \log(\hat{y}_{c,i}) = -\log(\hat{y}_{c^*,i})$$

- △ KL asymmetric: $KL(y_i, y_i^*) \neq KL(y_i^*, y_i)$ △

```
\[
\begin{array}{c|c|c}
   & y_i^* & \hat{y}_i \\
\hline
1.0 & 0.80 & \\
0.0 & 0.15 & \\
0.0 & 0.05 & \\
\end{array}
\]
```

$$KL(y_i, \hat{y}_i) = -\log(\hat{y}_{c^*,i}) = -\log(0.8) \approx 0.22$$
Logistic Regression Training

- \[ \mathcal{L}_{CE}(W, b) = \frac{1}{N} \sum_{i=1}^{N} \ell_{CE}(\hat{y}_i, y^*_i) = -\frac{1}{N} \sum_{i=1}^{N} \log(\hat{y}_{c^*}, i) \]
- \( \ell_{CE} \) smooth convex upper bound of \( \ell_{0/1} \) 
  \( \Rightarrow \) gradient descent optimization
- Gradient descent: \( W(t+1) = W(t) - \eta \frac{\partial \mathcal{L}_{CE}}{\partial W} \) \( (b(t+1) = b(t) - \eta \frac{\partial \mathcal{L}_{CE}}{\partial b}) \)
- **MAIN CHALLENGE:** computing \( \frac{\partial \mathcal{L}_{CE}}{\partial W} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial W} \) ?

  \( \Rightarrow \) **Key Property:** chain rule \( \frac{\partial x}{\partial z} = \frac{\partial x}{\partial y} \frac{\partial y}{\partial z} \)

  \( \Rightarrow \) Backpropagation of gradient error!
Chain Rule

\[
\frac{\partial l}{\partial x} = \frac{\partial l}{\partial y} \frac{\partial y}{\partial x}
\]

- Logistic regression:

\[
\frac{\partial l_{CE}}{\partial W} = \frac{\partial l_{CE}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial W}
\]
Logistic Regression Training: Backpropagation

\[ \frac{\partial l_{CE}}{\partial W} = \frac{\partial l_{CE}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial W}, \quad l_{CE}(\hat{y}_i, y_i^*) = -\log(\hat{y}_{c*,i}) \Rightarrow \text{Update for 1 example:} \]

- \[ \frac{\partial l_{CE}}{\partial \hat{y}_i} = \frac{-1}{\hat{y}_{c*,i}} = \frac{-1}{\hat{y}_i} \odot \delta_{c,c^*} \]
- \[ \frac{\partial l_{CE}}{\partial s_i} = \hat{y}_i - y_i^* = \delta_i^y \]
- \[ \frac{\partial l_{CE}}{\partial W} = x_i^T \delta_i^y \]
Logistic Regression Training: Backpropagation

- Whole dataset: data matrix $\mathbf{X} (N \times m)$, label matrix $\hat{\mathbf{Y}}, \mathbf{Y}^* (N \times K)$

- $\mathcal{L}_{CE}(\mathbf{W}, \mathbf{b}) = -\frac{1}{N} \sum_{i=1}^{N} \log(\hat{y}_{c*,i})$, $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}_{CE}}{\partial \hat{\mathbf{Y}}} \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{S}} \frac{\partial \mathbf{S}}{\partial \mathbf{W}}$

\[
\mathbf{X} \quad \mathbf{W} \quad \mathbf{S} \quad \hat{\mathbf{Y}} \quad \mathcal{L}_{CE}(\hat{\mathbf{Y}}, \mathbf{Y}^*)
\]

- $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{S}} = \hat{\mathbf{Y}} - \mathbf{Y}^* = \Delta^y$

- $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \mathbf{X}^T \Delta^y$
Perceptron Training: Backpropagation

- Perceptron vs Logistic Regression: adding hidden layer (sigmoid)
- **Goal:** Train parameters $W^y$ and $W^h$ (+bias) with Backpropagation

  $\Rightarrow$ computing

  $\frac{\partial L_{CE}}{\partial W^y} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial L_{CE}}{\partial W^y}$ and $\frac{\partial L_{CE}}{\partial W^h} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial L_{CE}}{\partial W^h}$

- Last hidden layer $\sim$ Logistic Regression
- First hidden layer: $\frac{\partial L_{CE}}{\partial W^h} = x_i^T \frac{\partial L_{CE}}{\partial u_i} \Rightarrow$ computing $\frac{\partial L_{CE}}{\partial u_i} = \delta^h_i$
Perceptron Training: Backpropagation

- Computing $\frac{\partial \ell_{CE}}{\partial u_i} = \delta^h_i \Rightarrow$ use chain rule: $\frac{\partial \ell_{CE}}{\partial u_i} = \frac{\partial \ell_{CE}}{\partial v_i} \frac{\partial v_i}{\partial h_i} \frac{\partial h_i}{\partial u_i}$
- ... Leading to: $\frac{\partial \ell_{CE}}{\partial u_i} = \delta^h_i = \delta^y_i \mathbf{W}^y \odot \sigma'(h_i) = \delta^y_i \mathbf{W}^y \odot (h_i \odot (1 - h_i))$
Deep Neural Network Training: Backpropagation

- Multi-Layer Perceptron (MLP): adding more hidden layers
- Backpropagation update ~ Perceptron: assuming $\frac{\partial L}{\partial u_{l+1}} = \Delta^{l+1}$ known
  - $\frac{\partial L}{\partial W^{l+1}} = H_l^T \Delta^{l+1}$
  - Computing $\frac{\partial L}{\partial U_l} = \Delta^l$ ($= \Delta^{l+1}^T W^{l+1} \odot H_l \odot (1 - H_l)$ sigmoid)
  - $\frac{\partial L}{\partial W^l} = H_{l-1}^T \Delta^{h_l}$
Neural Network Training: Optimization Issues

- Classification loss over training set (vectorized $w$, $b$ ignored):

$$
\mathcal{L}_{CE}(w) = \frac{1}{N} \sum_{i=1}^{N} \ell_{CE}(\hat{y}_i, y^*_i) = -\frac{1}{N} \sum_{i=1}^{N} \log(\hat{y}_{c^*,i})
$$

- Gradient descent optimization:

$$
w^{(t+1)} = w^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial w} (w^{(t)}) = w^{(t)} - \eta \nabla_w^{(t)}
$$

- Gradient $\nabla_w^{(t)} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}(\hat{y}_i, y^*_i)}{\partial w} (w^{(t)})$ linearly scales wrt:
  - $w$ dimension
  - Training set size

$\Rightarrow$ Too slow even for moderate dimensionality & dataset size!
Stochastic Gradient Descent

- **Solution**: approximate \( \nabla_w^{(t)} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}(\hat{y}_i, y_i^*)}{\partial w} \left( w^{(t)} \right) \) with subset of examples

  ⇒ **Stochastic Gradient Descent (SGD)**

  - Use a single example (online):
    \[
    \nabla_w^{(t)} \approx \frac{\partial \ell_{CE}(\hat{y}_i, y_i^*)}{\partial w} \left( w^{(t)} \right)
    \]

  - Mini-batch: use \( B < N \) examples:
    \[
    \nabla_w^{(t)} \approx \frac{1}{B} \sum_{i=1}^{B} \frac{\partial \ell_{CE}(\hat{y}_i, y_i^*)}{\partial w} \left( w^{(t)} \right)
    \]
Stochastic Gradient Descent

- **SGD:** approximation of the true Gradient $\nabla w$!
  - Noisy gradient can lead to bad direction, increase loss
  - **BUT:** much more parameter updates: online $\times N$, mini-batch $\times \frac{N}{B}$
  - **Faster convergence**, at the core of Deep Learning for large scale datasets

![Diagram showing full gradient, SGD (online), and SGD (mini-batch)]
Optimization: Learning Rate Decay

- Gradient descent optimization: $w^{(t+1)} = w^{(t)} - \eta \nabla w^{(t)}$
- $\eta$ setup? $\Rightarrow$ open question
- Learning Rate Decay: decrease $\eta$ during training progress
  - Inverse (time-based) decay: $\eta_t = \frac{\eta_0}{1 + r \cdot t}$, $r$ decay rate
  - Exponential decay: $\eta_t = \eta_0 \cdot e^{-\lambda t}$
  - Step Decay $\eta_t = \eta_0 \cdot r^{t/t_u}$ ...

Exponential Decay ($\eta_0 = 0.1$, $\lambda = 0.1s$)  
Step Decay ($\eta_0 = 0.1$, $r = 0.5$, $t_u = 10$)
Generalization and Overfitting

- **Learning**: minimizing classification loss $\mathcal{L}_{CE}$ over training set
  - Training set: sample representing data vs labels distributions
  - **Ultimate goal**: train a prediction function with low prediction error on the true (unknown) data distribution

\[
\mathcal{L}_{train} = 4, \quad \mathcal{L}_{train} = 9 \\
\mathcal{L}_{test} = 15, \quad \mathcal{L}_{test} = 13
\]

⇒ **Optimization ≠ Machine Learning!**
⇒ **Generalization / Overfitting!**
Regularization

- **Regularization**: improving generalization, i.e. test (≠ train) performances
- Structural regularization: add *Prior* $R(w)$ in training objective:

$$\mathcal{L}(w) = \mathcal{L}_{CE}(w) + \alpha R(w)$$

- $L^2$ regularization: weight decay, $R(w) = ||w||^2$
  - Commonly used in neural networks
  - Theoretical justifications, generalization bounds (SVM)
- Other possible $R(w)$: $L^1$ regularization, dropout, etc
Deep for image classification

• $M$ classes
• $M$ output neurons
  • 1 neuron / class

Question: how to connect the image to the MLP?
Outline

ConvNets as Deep Neural Networks for Vision

1. Neural Nets
2. Deep Convolutional Neural Networks