Neural Networks

Perceptron

1. "A simple model for this neuron is the perceptron"

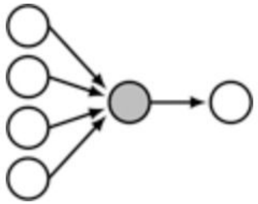

Figure 12.2: Perceptron.

$$z = f(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i + b = \mathbf{w}^\top \mathbf{x} + b \qquad \triangleleft \quad \text{linear layer} \tag{12.1}$$

$$g(z) = \begin{cases} 1, & \text{if} \quad z > 0 \\ 0, & \text{otherwise} \end{cases} \qquad \triangleleft \quad \text{activation function} \tag{12.2}$$

$$y = g(f(\mathbf{x})) \qquad \triangleleft \quad \text{perceptron} \tag{12.3}$$

Perceptron - Learning

1. Minimize classification loss L

$$\mathbf{w}^*, b^* = \arg\min_{\mathbf{w}, b} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\mathbf{w}^\top \mathbf{x}^{(i)} + b, y^{(i)}) \tag{12.4}$$

Multilayer Perceptrons

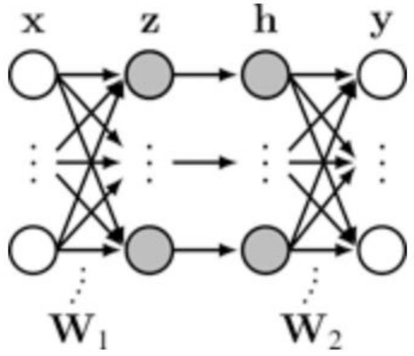

Figure 12.6: Mutilayer perceptron.

1. “How many layers does this net have? Some texts will say two [W1, W2], others three [x, {z, h}, y], others four [x, z, h, y]. We must get comfortable with the ambiguity.”

Multilayer Perceptrons

$$\mathbf{z} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \qquad \triangleleft \quad \text{linear layer} \tag{12.5}$$

$$\mathbf{h} = g(\mathbf{z}) \qquad \triangleleft \quad \text{activation function} \tag{12.6}$$

$$\mathbf{y} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2 \qquad \triangleleft \quad \text{linear layer} \tag{12.7}$$

1. “In general, MLPs can be constructed with any number of layers following this pattern: *linear layer, activation function, linear layer, activation function*, and so on.”

Activations vs Parameters

1. “Both activations and parameters are tensors of variables.”
2. “anything we can do with parameters, we can do with activations instead, and vice versa”
3. “while normally we learn the values of the parameters, we could instead hold the parameters fixed and learn the values of the activations that achieve some objective”

Optimizing Activations

Optimizing an Input to Maximize a Neuron’s Activation

Let’s say we have a neural network $f(x)$, and we want to find an input x that maximizes the activation of a particular neuron in a hidden layer.

1. **Define Input x as a Learnable Variable:**
   - Instead of training model parameters, we start with a random input x and treat it as a learnable tensor.
2. **Compute Activation $a=f(x)$:**
   - Pass x through the network and get activations at the chosen layer.
3. **Define the Objective Function:**
   - Maximize the activation of a specific neuron:

     $$L=-a_{target\ neuron}$$

   - (Negative sign because optimizers typically minimize loss functions.)
4. **Perform Gradient Descent on x:**
   - Compute gradients $\partial L/\partial x$ and update x using an optimizer
5. **Repeat Until Convergence:**
   - Keep updating x until we get an optimized input that maximizes the neuron’s activation.

Optimizing Activations

Real-World Use Cases

1. **Feature Visualization (DeepDream, Activation Maximization)**
   - We optimize input pixels x to activate specific neurons and understand what features the model learns.
2. **Adversarial Attacks**
   - We optimize x to make the model misclassify it while keeping changes imperceptible.

Deep Nets

1. “Deep nets are neural nets that stack the linear-nonlinear motif many times”

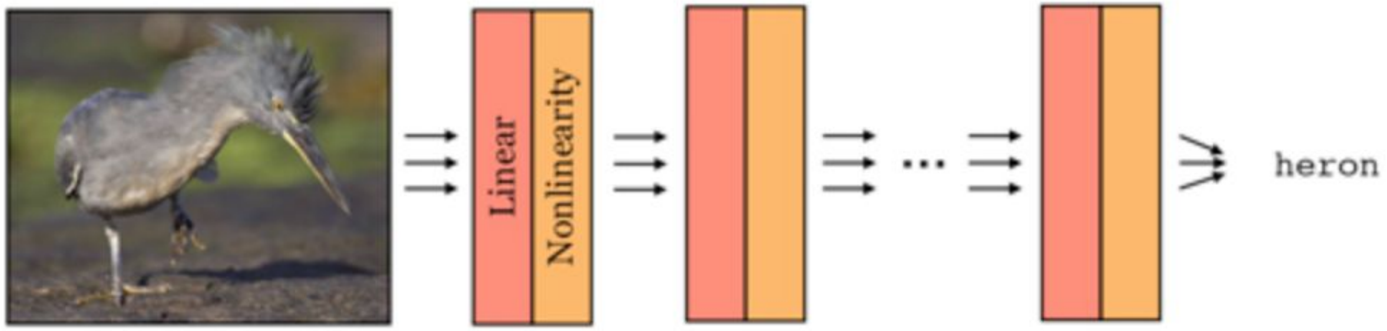

Figure 12.8: Deep nets consist of linear layers interleaved with nonlinearities.

Deep Learning

1. Typically use gradient-based optimization that utilizes backpropagation
2. “However, it is certainly possible to optimize neural nets with other methods, including **zeroth-order optimizers** like evolution strategies”
3. “One intriguing alternative to backpropagation is called Hebbian learning”
4. “Hebbian learning, in contrast, is a bottom-up approach, where neurons wire up just based on the feedforward pattern of activity in the net.”
5. “The canonical learning rule in Hebbian methods is Hebb's rule: **“fire together, wire together.”**
6. “Although this learning rule is not explicitly minimizing a loss function, it has been shown to lead to effective neural representations.”
7. “Hebbian learning is also of interest because it is considered to be more biologically plausible than backpropagation”

Why Neural Networks are Good Architecture?

1. They are high capacity (big enough nets are universal approximators).
2. They are differentiable (the parameters can be optimized via gradient descent).
3. They have good inductive biases (neural architectures reflect real structure in the world).
4. They run efficiently on parallel hardware.
5. They build abstractions.

References

1. Foundations of Computer Vision - Chapter 12