Training Neural Nets

COMPSCI 527 — Computer Vision
Outline

1 The Softmax Simplex
2 Loss and Risk
3 Back-Propagation
4 Stochastic Gradient Descent
5 Regularization
The Softmax Simplex

- Neural-net classifier: \( \hat{y} = h(x) : \mathbb{R}^d \rightarrow Y \)
- The last layer of a neural net used for classification is a soft-max layer
  \( p = \sigma(z) = \frac{\exp(z)}{\sum \exp(z)} \)
- The net is \( p = f(x, w) : X \rightarrow P \)
- The classifier is \( \hat{y} = h(x) = \arg \max p = \arg \max f(x, w) \)
- \( P \) is the set of all nonnegative real-valued vectors \( p \in \mathbb{R}^K \)
  whose entries add up to 1 (with \( K = |Y| \)):
  \[
P \overset{\text{def}}{=} \{ p \in \mathbb{R}^K : p \geq 0 \text{ and } \sum_{i=1}^K p_i = 1 \}.
  \]
The Softmax Simplex

\[ P \stackrel{\text{def}}{=} \{ \mathbf{p} \in \mathbb{R}^K : \mathbf{p} \succeq \mathbf{0} \text{ and } \sum_{i=1}^{K} p_i = 1 \} \]

- Decision regions are polyhedral and convex:
  \[ P_c = \{ p_c \geq p_j \text{ for } j \neq c \} \text{ for } c = 1, \ldots, K \]
- A network transforms images into points in \( P \)
Training is Empirical Risk Minimization

- Define a loss $\ell(y, \hat{y})$. How much do we pay when the true label is $y$ and the network says $\hat{y}$?
- Network: $p = f(x, w)$, then $\hat{y} = \arg\max p$
- Risk is average loss over training set $T = \{(x_1, y_1), \ldots, (x_N, y_N)\}$:
  \[
  L_T(w) = \frac{1}{N} \sum_{n=1}^{N} \ell_n(w) \quad \text{where} \quad \ell_n(w) = \ell(y_n, f(x_n, w))
  \]
- Determine network weights $w$ by minimizing $L_T(w)$
- Use some variant of steepest descent
- We need $\nabla L_T(w)$ and therefore $\nabla \ell_n(w)$
The Cross-Entropy Loss

- Ideal loss would be 0-1 loss \( \ell_{0,1}(y, \hat{y}) \) on classifier output \( \hat{y} \)
- 0-1 loss is constant where it is differentiable!
- Not useful for computing a gradient for risk minimization
- Use cross-entropy loss on the softmax output \( p \) as a proxy loss
  \[ \ell(y, p) = - \log p_y \]
- Differentiable!
- Unbounded loss for total misclassification
Example: $K = 5$ Classes

- Last layer before soft-max has activations $\mathbf{z} \in \mathbb{R}^K$
- Soft-max has output $\mathbf{p} = \sigma(\mathbf{z}) = \frac{\exp(\mathbf{z})}{\sum \exp(\mathbf{z})} \in \mathbb{R}^5$
- Ideally, if the correct class is $y = 4$, we would like output $\mathbf{p}$ to equal $\mathbf{q} = [0, 0, 0, 1, 0]$, the one-hot encoding of $y$
- That is, $q_y = q_4 = 1$ and all other $q_j$ are zero
- $\ell(y, \mathbf{p}) = -\log p_y = -\log p_4$
- $p_y \to 1$ and $\ell(y, \mathbf{p}) \to 0$ when $z_y \to \infty$
- That is, when $\mathbf{p}$ approaches the right corner of the simplex
- $p_y \to 0$ and $\ell(y, \mathbf{p}) \to \infty$ when $z_y \to -\infty$
- That is, when $\mathbf{p}$ is far from the right corner of the simplex
Example, Continued

\[ \ell(y, p) = - \log p_y = - \log \frac{\exp(z_y)}{\sum \exp(z)} = \log \left( \frac{\sum \exp(z)}{\exp(z_y)} \right) - z_y \]

- \( p_y \to 0 \) and \( \ell(y, p) \to \infty \) when \( z_y \to -\infty \)
- \( p_y \to 1 \) and \( \ell(y, p) \to 0 \) when \( z_y \to \infty \)
- Actual plot depends on all values in \( z \)
- This is a “soft hinge loss” in \( z \) (not in \( p \))
Back-Propagation

- We need $\nabla L_T(w)$ and therefore $\nabla \ell_n(w) = \frac{\partial \ell_n}{\partial w}$.
- Computations from $x_n$ to $\ell_n$ form a chain.
- Apply the chain rule.
- Every derivative of $\ell_n$ w.r.t. layers before $k$ goes through $x^{(k)}$.

\[
\frac{\partial \ell_n}{\partial w^{(k)}} = \frac{\partial \ell_n}{\partial x^{(k)}} \frac{\partial x^{(k)}}{\partial w^{(k)}}
\]

Start:
\[
\frac{\partial \ell_n}{\partial x^{(K)}} = \frac{\partial \ell_n}{\partial p}
\]

(recursion!)
Local Jacobians

- Local computations at layer $k$: $\frac{\partial x^{(k)}}{\partial w^{(k)}}$ and $\frac{\partial x^{(k)}}{\partial x^{(k-1)}}$
- Partial derivatives of $f^{(k)}$ with respect to layer weights and input to the layer
- Local Jacobian matrices, can compute by knowing what the layer does
- The start of the process can be computed from knowing the loss function, $\frac{\partial \ell_n}{\partial x^{(K)}} = \frac{\partial \ell}{\partial p}$ (gradient)
- Another local Jacobian
- The rest is going recursively from output to input, one layer at a time, accumulating $\frac{\partial \ell_n}{\partial w^{(k)}}$ into a vector $\frac{\partial \ell_n}{\partial w}$
Back-Propagation Spelled Out for $K = 3$

\[
x_n = x^{(0)}
\]

\[
x^{(1)} = f^{(1)}(x^{(0)})
\]

\[
x^{(2)} = f^{(2)}(x^{(1)})
\]

\[
x^{(3)} = f^{(3)}(x^{(2)}) = p
\]

\[
y_n = \ell_n
\]

\[
\frac{\partial \ell_n}{\partial w^{(1)}} = \frac{\partial \ell_n}{\partial x^{(1)}} \frac{\partial x^{(1)}}{\partial w^{(1)}}
\]

\[
\frac{\partial \ell_n}{\partial w^{(2)}} = \frac{\partial \ell_n}{\partial x^{(2)}} \frac{\partial x^{(2)}}{\partial w^{(2)}}
\]

\[
\frac{\partial \ell_n}{\partial w^{(3)}} = \frac{\partial \ell_n}{\partial x^{(3)}} \frac{\partial x^{(3)}}{\partial w^{(3)}}
\]

(Jacobians in blue are local)
Computing Local Jacobians

\[
\frac{\partial \mathbf{x}^{(k)}}{\partial \mathbf{w}^{(k)}} \quad \text{and} \quad \frac{\partial \mathbf{x}^{(k)}}{\partial \mathbf{x}^{(k-1)}}
\]

- Easier to make a “layer” as simple as possible
- \( \mathbf{z} = V\mathbf{x} + \mathbf{b} \) is one layer (Fully Connected (FC), affine part)
- \( \mathbf{z} = \rho(\mathbf{x}) \) (ReLU) is another layer
- Softmax, max-pooling, convolutional,...
Local Jacobians for a FC Layer

\( z = Vx + b \)

- \( \frac{\partial z}{\partial x} = V \) (easy!)
- What is \( \frac{\partial z}{\partial w} \)? Three subscripts: \( \frac{\partial z_i}{\partial v_{jk}} \). A 3D tensor?
- For a general package, tensors are the way to go
- Conceptually, it may be easier to vectorize everything:

\[
V = \begin{bmatrix}
v_{11} & v_{12} & v_{13} \\
v_{21} & v_{22} & v_{23} \\
\end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}
\]

\[
w = [v_{11}, v_{12}, v_{13}, v_{21}, v_{22}, v_{23}, b_1, b_2]^T
\]

- \( \frac{\partial z}{\partial w} \) is a 2 \( \times \) 8 matrix
- With \( e \) outputs and \( d \) inputs, an \( e \times e(d + 1) \) matrix
Jacobian $\mathbf{w}$ for a FC Layer

\[
\begin{bmatrix}
  z_1 \\
  z_2
\end{bmatrix} =
\begin{bmatrix}
  w_1 & w_2 & w_3 \\
  w_4 & w_5 & w_6
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3
\end{bmatrix} +
\begin{bmatrix}
  w_7 \\
  w_8
\end{bmatrix}
\]

- Don’t be afraid to spell things out:

\[
\begin{align*}
  z_1 &= w_1 x_1 + w_2 x_2 + w_3 x_3 + w_7 \\
  z_2 &= w_4 x_1 + w_5 x_2 + w_6 x_3 + w_8
\end{align*}
\]

\[
\frac{\partial z_1}{\partial \mathbf{w}} =
\begin{bmatrix}
  \frac{\partial z_1}{\partial w_1} & \frac{\partial z_1}{\partial w_2} & \frac{\partial z_1}{\partial w_3} \\
  \frac{\partial z_2}{\partial w_4} & \frac{\partial z_2}{\partial w_5} & \frac{\partial z_2}{\partial w_6} \\
  \frac{\partial z_2}{\partial w_7} & \frac{\partial z_2}{\partial w_8}
\end{bmatrix}
\]

\[
\frac{\partial z_2}{\partial \mathbf{w}} =
\begin{bmatrix}
  \frac{\partial z_1}{\partial w_7} & \frac{\partial z_1}{\partial w_8} \\
  \frac{\partial z_2}{\partial w_7} & \frac{\partial z_2}{\partial w_8}
\end{bmatrix}
\]

- Obvious pattern: Repeat $\mathbf{x}^T$, staggered, $e$ times
- Then append the $e \times e$ identity at the end
Stochastic Gradient Descent

Training

- Local gradients are used in back-propagation
- So we now have $\nabla L_T(w)$
- $\hat{w} = \arg\min L_T(w)$
- $L_T(w)$ is (very) non-convex, so we look for local minima
- $w \in \mathbb{R}^m$ with $m$ very large: No Hessians
- Gradient descent
- Even so, every step calls back-propagation, $N = |T|$ times
- Back-propagation computes $m$ derivatives $\nabla L_T(w)$
- Computational complexity is $\Omega(mN)$ per step
- Even gradient descent is way too expensive!
No Line Search

- Line search is out of the question
- Fix some step multiplier $\alpha$, called the *learning rate*
  \[ w_{t+1} = w_t - \alpha \nabla L_T(w_t) \]
- How to pick $\alpha$? Cross-validation is too expensive
- Tradeoffs:
  - $\alpha$ too small: Slow progress
  - $\alpha$ too big: Jump over minima
- Frequent practice:
  - Start with $\alpha$ relatively large, and monitor $L_T(w)$
  - When $L_T(w)$ levels off, decrease $\alpha$
- Alternative: Fixed decay schedule for $\alpha$
- Better (recent) option: Change $\alpha$ adaptively
  (Adam, 2015)
Manual Adjustment of $\alpha$

- Start with $\alpha$ relatively large, and monitor $L_T(w_t)$
- When $L_T(w_t)$ levels off, decrease $\alpha$
- Typical plots of $L_T(w_t)$ versus iteration index $t$: