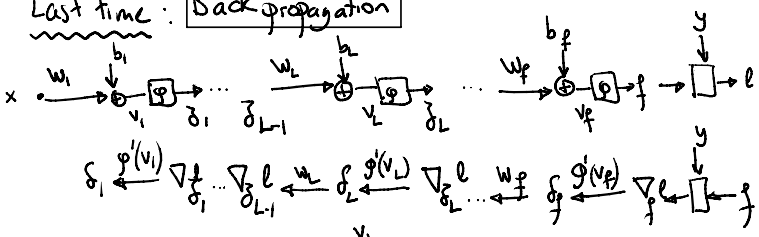
L11

ECE/CS 559 Lecture 11 T 10/1

Last time: **Backpropagation**



Forward: $\ell(z_L) = \ell\big(\varphi(\underbrace{W_L z_{L-1} + b_L}_{v_L})\big)$

Backward: $\nabla_{z_{L-1}} \ell = W_L^T \big(\varphi'(v_L) \circ \nabla_{z_L} \ell\big) = W_L^T \delta_L$ (element wise)

Gradients: $\nabla_{W_L} \ell = \big(\varphi'(v_L) \circ \nabla_{z_L} \ell\big) z^T = \delta_L z^T$, $\nabla_{b_L} \ell = \delta_L$

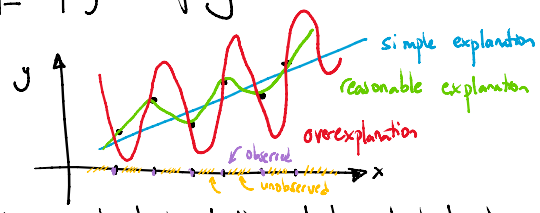
- When learning neural networks, issue is that we train on the observed contexts x. However, we mostly care about performance on new unobserved contexts!
- Performing well in unobserved contexts = **generalization**
  Overfitting prevents generalization (unless it's mild)
- How do we quantify this?

  Training Data $= \{(x,y),\dots\}$ ↳ use this to build NN

  Testing Data $= \{(x',y'),\dots\}$ ↳ use this to assess generalization
- We must be careful!
  We can't keep on using testing data to refine the NN, otherwise we may overfit it too and not generalize to newer data.

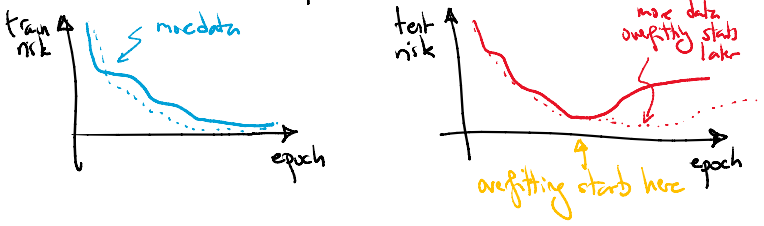
① Overfitting

- Tendency to overexplain the training data.
- Overexplain = when many explanations describe experience equally well, choose one that adds many complicated details and twists (typically what conspiracy theories do)
- Example: polynomial fitting



- Typical property leading to this: high sensitivity to changes in data

- Typical behavior of train vs. test risk:



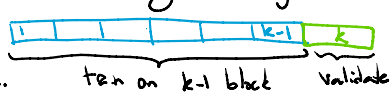
- Observations:
  - Stopping early could help alleviate overfitting
    Why? The NN will not fit the noise.
    How? By staying closer to its initialization (staying "simpler")
  - More training data reduces overfitting, delays its onset.

② Cross-Validation

- Since training risk is not a good enough proxy to testing risk, we could hold out some of the training data for validation:

Training Data $= \{(x,y),\dots\}$ (train on this often) Validation Data $= \{(x'',y''),\dots\}$ (check if okay only sometimes) Testing Data $= \{(x',y'),\dots\}$ (test on this rarely)

- What is validation good for?
  - Choose what epoch to stop at.
  - Choose between possible architectures, learning rates, etc. ("hyper parameters")
  - These choices are "coarser" than choosing the weights & biases
- k-Fold Cross-validation:



Change validation block, repeat. Average.

③ Regularization

- Cross-validation helps see if we overfit. Regularization helps prevent it.
- The word comes from inverse problems in physics:
  - To decide between many possible explanations, we need to favor some.
  - We tend to favor "simple" ones (Occam's Razor) (e.g., low energy)
- E.g., for polynomials and NNs, favor small weights:

  $\min_w R(w)$ for $\|w\|^2 \le C$ $\Leftrightarrow$ $\min_w R(w) + \lambda \|w\|_2^2$ (regularized/penalized risk)
- What does it mean? Prefer smoother functions:
- Why does it work? Can't overfit noise
  Regularized risk is a better proxy for test-risk

  $\|w\|_1 = |w_1| + \dots + |w_n|$ (other choice)
- How does it affect backpropagation? Simple!

  $\nabla_w \big(R(w) + \lambda \|w\|_2^2\big) = \nabla_w R + 2\lambda w$

Note: Any change that helps prefer some w over others is regularization. This introduces what we call inductive bias, which (if right) improves generalization.

L9&10

ECE/CS 559 Lecture 9 & 10 T,Th 9/24, 9/26

Last time: Gradient Descent; Widrow-Hoff LMS Algorithm

$$\min_w R(w) = \sum_{(x,y)\in Data} \ell(y, f(x;w))$$

$$w' \leftarrow w - \eta \nabla R(w) \qquad \nabla_w R = \left[\frac{\partial R}{\partial w_1}, \dots, \frac{\partial R}{\partial w_k}\right]$$

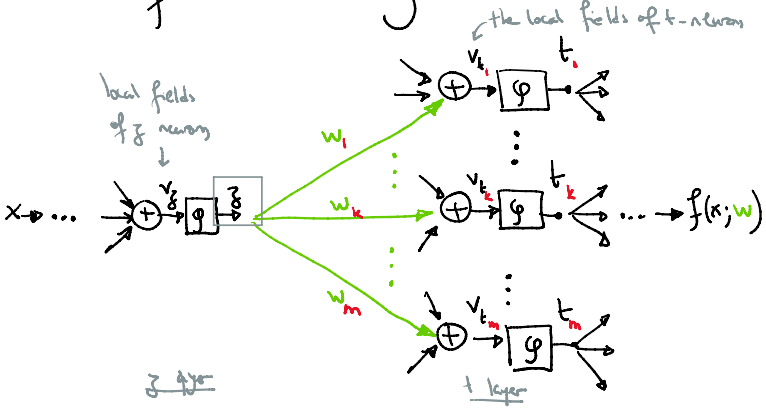
- $\ell(y, f(x;w)) = \| y - f(x;w) \|^2$

$f(x;w) = wx$

$$w' \leftarrow w - \eta \nabla R(w) = w + 2\eta \sum_{(x,y)} (y - wx) x^T \quad \text{(absorb)}$$

Back propagation

- So far we've been calculating gradients with 1 neuron.
- What if we connect many neurons?



- We will apply the chain rule multiple times:

$$\frac{\partial R}{\partial w_k} = \sum_{(x,y)} \left[\frac{\partial \ell}{\partial w_k}\right] \quad (?)$$

need [this]

(?) $$\frac{\partial \ell}{\partial w_k} = \frac{\partial}{\partial w_k} \ell(t_k, \dots)\Big|_{t_k = \varphi(w_k z + \dots)} \quad (v_{t_k} = w_k z + \dots)$$

have this from forward pass

(two chain rules) $$= \left[\frac{\partial \ell}{\partial t_k}\Big|_{t_k=\varphi(v_{t_k})}\right] \cdot \varphi'(v_{t_k}) \cdot z \quad (*)$$

need [this] at every neuron output

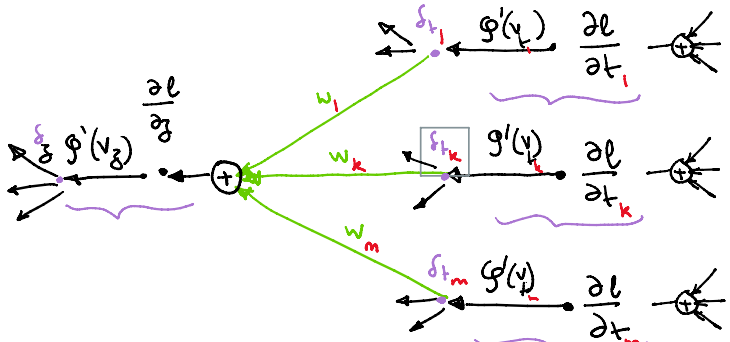
$$\delta_{t_k}$$

(*) $$\frac{\partial \ell}{\partial z}\Big|_{z=\varphi(u)} = \frac{\partial}{\partial z} \ell(t_1, \dots, t_k, \dots, t_m)\Big|_{t_k=\varphi(w_k z + \dots)} \quad (v_{t_k} = w_k z + \dots)$$

(two chain rules) $$= \sum_{k=1}^{m} \frac{\partial \ell}{\partial t_k}\Big|_{t_k=\varphi(v_{t_k})} \cdot \varphi'(v_{t_k}) \cdot w_k$$

$$\delta_{t_k}$$

- To calculate (?) and (*), we perform a forward pass to get all the outputs $(u, z, v_t, t, \dots)$, then we perform a backward pass:



Gradient of w = forward signal z × backward signal δ_t

- Of course, the layer of z has other (say n) neurons. Let's index the equations for these (blue j)

$$\frac{\partial \ell}{\partial w_{kj}} = \delta_{t_k} \cdot z_j \qquad \frac{\partial \ell}{\partial z_j} = \sum_{k=1}^{m} \delta_{t_k} w_{kj} \qquad j = 1, \dots, n$$

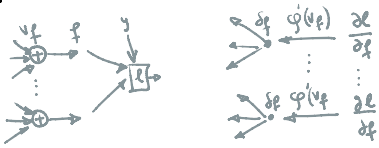
- If we think of δ_t as a m-dim vector, of z as an n-dim vector, of W as a $m \times n$-dim matrix
- We get the linear algebraic backward equation:

$$\nabla_W \ell = \delta_t z^T \qquad \nabla_z \ell = W^T \delta_t \qquad \delta_z = \nabla_z \ell \bullet \varphi'(v_z)$$

outer product; matrix multiplication; elementwise product

- To complete this, we need 2 things: how to start, how to handle bias
- Start at the last layer $\nabla_f \ell$ depends on the choice of ℓ

e.g. if $\ell = \|f - y\|^2$ then $\nabla_f \ell = 2(f - y)$, $\delta_f = \nabla_f \ell \bullet \varphi'(v_f)$

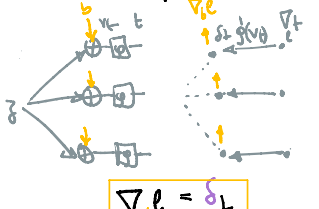


- Biases are like dead-ends on the backward path.

$$\frac{\partial \ell}{\partial b} = \frac{\partial}{\partial b} \ell(t_k, \dots)\Big|_{t_k=\varphi(\dots + b)} \quad (v_{t_k} = \dots + b)$$

$$= \frac{\partial \ell}{\partial t}\Big|_{t_k=\varphi(v_t)} \cdot \varphi'(v_{t_k}) \cdot 1$$

$$\delta_{t_k}$$



Note: Can we forward eqs to get these: $\ell(t) = \ell(\underbrace{\varphi(w_j z + b)}_{t})$ $\Rightarrow$ $\nabla_w \ell = [\underbrace{\nabla_t \ell \bullet \varphi'(v_t)}_{\delta_t}] z^T$, $\nabla_b \ell = [\underbrace{\nabla_t \ell \bullet \varphi'(v_t)}_{\delta_t}] \cdot 1$, $\nabla_z \ell = W^T [\underbrace{\nabla_t \ell \bullet \varphi(v_t)}_{\delta_t}]$

L8

ECE/CS 559 Lecture 8 Th 9/19

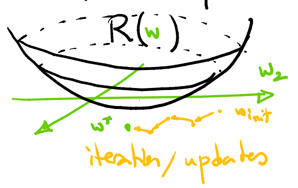
Last time: Linear Regression with Squared Loss

- **Data**: $\{(x,y), \dots\}$ $x \in \mathbb{R}^n$ $y \in \mathbb{R}^m$
- **Predictor**: $f(x; W) = Wx$ $W \in \mathbb{R}^{m \times n}$
- **Goal**: $\min_W R(W) = \sum_{(x,y) \in \text{Data}} \underbrace{\| y - Wx \|^2}_{\ell(y, Wx) \equiv \ell((x,y); W)}$

$m [\hat{W}] \cdot [X]_{\#\text{data}} = m [Y]_{\#\text{data}}$

it's a quadratic equation

$W = Y \, \boxed{X^T (XX^T)^{-1}}$ Pseudo inverse



① **Gradient Descent**

Goal: $\min_W R(W) = \sum_{(x,y) \in \text{Data}} \overbrace{\| y - f(x;W) \|^2}^{\ell(y, f(x;W))}$

Idea: Set the derivative to zero! → reduce step-by-step.

Gradient: $\nabla_W R = \left[\frac{\partial R}{\partial w_1}, \dots, \frac{\partial R}{\partial w_k} \right]$

Each step: Change W by ΔW : $\boxed{W' \leftarrow W + \Delta W}$

$R(W') = R(W + \Delta W) = R(W) + \nabla R(W)^T \Delta W + O(\|\Delta W\|^2)$ (Taylor expansion; ignore)

How should we choose ΔW to get the best reduction?

- To $\min_{\Delta W} \nabla R^T \Delta W$, let $\Delta W \propto -\nabla_W R$ (proportional) : $\boxed{W' \leftarrow W - \eta \nabla R(W)}$

1-D Example: $R(w) = w^4$ $\frac{dR}{dw} = 4w^3$ $\eta = \frac{1}{8}$

$w(0) = 1$

$w(1) \leftarrow w(0) - \nabla R(w(0)) \cdot \eta$
$= 1 - 4 \cdot 1^3 \cdot \frac{1}{8} = \frac{1}{2}$

$w(2) \leftarrow w(1) - \nabla R(w(1)) \cdot \eta$
$= \frac{1}{2} - 4 \left(\frac{1}{2}\right)^3 \cdot \frac{1}{8} = \frac{7}{16}$

- What's happening?

$w(t) \leftarrow w(t-1) - 4\, w(t-1)^3 \eta = [1 - \underbrace{4\eta\, w(t-1)^2}] \, w(t-1)$

- If η is small enough, always diminishes. (<1)
Since bounded from below (by 0) will converge.
- But will it converge to 0? Yes, think about why.
- What if η is too big? Try $\eta = \frac{1}{2}$ with $w(0) = 1$.

② **Widrow-Hoff LMS Algorithm**

- Let's apply gradient descent to linear regression. Recall:
- $\nabla R(W) = -2 \sum_{(x,y) \in \text{Data}} (y - Wx) x^T$ same shape as W

$W \leftarrow W - \eta \nabla R(W) = W + 2\eta \sum_{(x,y)} (y - Wx) x^T$ (absorb)

- When $y \in \mathbb{R}^1$ then $W = w^T$ and $y - w^T x \in \mathbb{R}$
The updates become: $w \leftarrow w + \eta \sum_{(x,y)} (y - w^T x) x$
- We could do this 1 data point at a time:
For each $(x,y) \in$ Data: $w \leftarrow w + \eta (y - w x) x$

Similar to the Perceptron learning algorithm!

③ **Delta Rule**

- Squared loss for single neuron with differentiable $\varphi(\cdot)$:

$R(w) = \sum_{(x,y)} (y - \varphi(w^T x))^2$

$\nabla R(w) = -2 \sum_{(x,y)} (y - \varphi(w^T x)) \varphi'(w^T x) x^T$ chain rule

$w \leftarrow w + 2\eta \sum_{(x,y)} (y - \varphi(w^T x)) \varphi'(w^T x) x$

- 1 data point at a time:

$w \leftarrow w + \eta (y - \varphi(w^T x)) \varphi'(w^T x) x$

Example $\varphi(\cdot)$:

- Sigmoid: $\varphi(v) = \frac{1}{1 + e^{-av}}$ since $1 - \varphi(v) = \frac{e^{-av}}{1 + e^{-av}}$

$\varphi'(v) = \frac{0 - (-a e^{-av})}{(1 + e^{-av})^2} = \frac{a e^{-av}}{(1 + e^{-av})^2} = a \varphi(v) (1 - \varphi(v))$

- ReLU: $\varphi(v) = \begin{cases} v & v \geq 0 \\ 0 & \text{otherwise} \end{cases}$

$\varphi'(v) = \begin{cases} 1 & v \geq 0 \\ 0 & \text{otherwise} \end{cases} = \text{step}(v)$

L7

ECE/CS 559 Lecture 7 9/17

Last time: Perceptron Learning Example — Supervised Learning

- Data: $\{(x,y),\dots\}$ $x \in \mathcal{X}$ (feature space), $y \in \mathcal{Y}$ (label space)
- Task: Predict/imitate y using only x.
- How: Use a predictor — neural network $y_w(x) \equiv f(x;w)$
- Loss: how far y and $f(x;w)$ are: $\ell(y,f) = \mathbb{1}\{y \neq f\}$ or $\|y-f\|^2$
- Risk: (empirical) $R(w) = \frac{1}{|\text{Data}|} \sum_{(x,y)\in\text{Data}} \ell(y, f(x,w))$
- Goal: Minimize (empirical) risk, make few mistakes/errors / stay close to y.

- Types of algorithms:
  - Online: - See (x,y), update w - See (x',y'), update w - See (x'',y''), update w ⋮
  - Batch: - Get Data $= \{(x,y),\dots\}$ • Epoch 1: use Data, update w • Epoch 2: use Data, update w ⋮

① Linear Regression with Squared Loss

- Data: $\{(x,y),\dots\}$ $x \in \mathbb{R}^n$ $y \in \mathbb{R}^m$
- Predictor: $f(x;w) = Wx$ $W \in \mathbb{R}^{m\times n}$ (this is a neuron with $g(v) = v$, identity activation)
- Goal: $\min_W R(W) = \sum_{(x,y)\in\text{Data}} \underbrace{\|y - Wx\|^2}_{\ell(y,Wx)}$

Analytic Solution: Optimality conditions, unconstrained $\Rightarrow \nabla_w R = 0$ (gradient)

- Single data point $\ell = \sum_{k=1}^{m} \left(y_k - \sum_{j=1}^{n} w_{kj} x_j\right)^2$

$$\frac{d\ell}{dw_{kj}} = -2x_j\left(y_k - \sum_{j'=1}^{n} w_{kj'}x_{j'}\right) = -2y_k x_j + \sum_{j'=1}^{n} w_{kj'}(x_j x_{j'})$$

$$\Rightarrow \nabla_W \ell = -2yx^T + Wxx^T$$

- All data points: $R(w) = \sum_{(x,y)\in\text{Data}} \ell(y,Wx) \Rightarrow \nabla_w R = \sum_{(x,y)} \nabla_w \ell$

$$\nabla_W R = \sum_{(x,y)\in\text{Data}} -2yx^T + 2W \sum_{x\in\text{Data}} xx^T$$

$$= -2\underbrace{\begin{bmatrix} | & | & & | \\ y & y & \cdots & y \\ | & | & & | \end{bmatrix}}_{m\times|\text{Data}|} \underbrace{\begin{bmatrix} - & x & - \\ - & x & - \\ & \vdots & \\ - & x & - \end{bmatrix}}_{|\text{Data}|\times n} + 2W \underbrace{\begin{bmatrix} | & | & & | \\ x & x & \cdots & x \\ | & | & & | \end{bmatrix}}_{n\times|\text{Data}|} \begin{bmatrix} - & x & - \\ - & x & - \\ & \vdots & \\ - & x & - \end{bmatrix}$$

$$= -2YX^T + 2WXX^T$$

- $\nabla_W R = 0 \Rightarrow -2YX^T + 2WXX^T = 0$

$$\Rightarrow W = Y\boxed{X^T(XX^T)^{-1}}$$ Moore-Penrose Pseudo inverse

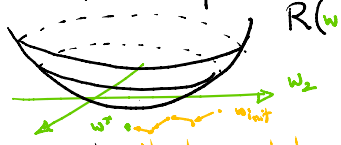
Compare to: $Wx \approx Y$ not invertible if $|\text{Data}| > n$

$$m\begin{bmatrix} & W & \end{bmatrix} n\begin{bmatrix} | & & | \\ x & \cdots & x \\ | & & | \end{bmatrix} \approx m\begin{bmatrix} | & & | \\ y & \cdots & y \\ | & & | \end{bmatrix}$$ $|\text{Data}|$

but it's "like" we inverted it above.

- Where are the updates? One single batch update!

Intuition: it's a quadratic equation



- Other shapes → solve by iterative updates

② Gradient Descent

Goal: $\min_W R(W) = \sum_{(x,y)\in\text{Data}} \underbrace{\|y - f(x;W)\|^2}_{\ell(y,f(x;W))}$

Idea: Set the derivative to zero! reduce step-by-step.

Gradient: $\nabla_w R = \left[\frac{\partial R}{\partial w_1}, \dots, \frac{\partial R}{\partial w_k}\right]$

Each step: Change w by Δw: $w' \leftarrow w + \Delta w$.

$$R(w') = R(w+\Delta w) = R(w) + \nabla_w R(w)^T \Delta w + O(\|\Delta w\|^2)$$

Taylor expansion; ignore

How should we choose Δw to get the best reduction?

- To minimize $\nabla_w R^T \Delta w$, let $\Delta w \propto -\nabla_w R$: $w' \leftarrow w - \eta \nabla R(w)$

proportional

1-D Example: $R(w) = w^4$ $\frac{dR}{dw} = 4w^3$ $\eta = \frac{1}{8}$

$w(0) = 1$

$w(1) \leftarrow w(0) - \nabla R(w(0))\cdot\eta = 1 - 4\cdot 1^3 \cdot \frac{1}{8} = \frac{1}{2}$

$w(2) \leftarrow w(0) - \nabla R(w(0))\cdot\eta = \frac{1}{2} - 4\left(\frac{1}{2}\right)^3 \cdot \frac{1}{8} = \frac{3}{16}$

- What's happening?

$$w(t) \leftarrow w(t-1) - 4w(t-1)^3\eta = \underbrace{[1 - 4\eta w(t-1)^2]}_{<1} w(t-1)$$

- If η is small enough, always diminishes. (<1) Since bounded from below (by 0) will converge.
- But will it converge to 0? Yes, think about why.
- What if η is too big? Try $\eta = \frac{1}{2}$ with $w(0) = 1$.