AOS 2 – Deep learning

Lecture 02: Training Neural Networks

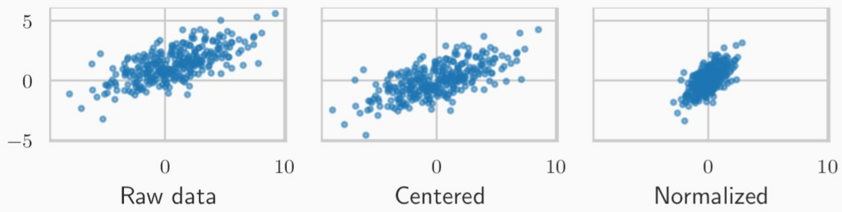
Sylvain Rousseau

Data Preprocessing

Training data $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, $\boldsymbol{x}_i \in \mathbb{R}^n$

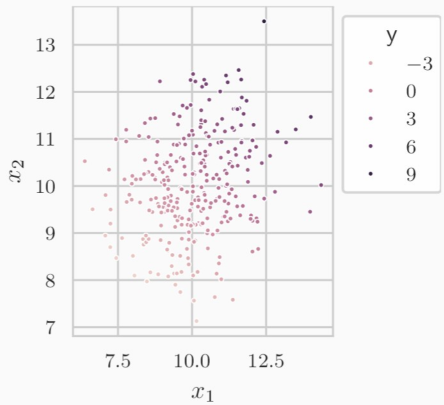
- Centered: $\boldsymbol{x}_i^{\text{centered}} = \boldsymbol{x}_i - \overline{\boldsymbol{x}}$
- Normalized $\boldsymbol{x}_i^{\text{norm}} = \boldsymbol{x}_i^{\text{centered}} / \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^2}$ (entry-wise operations)

Example with $N = 500$ and $n = 2$



Why centering?

- Data generating model that is not centered
  - $Y = \langle X - \boldsymbol{\mu}, \mathbf{u}\rangle + \varepsilon$
  - $X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, $\varepsilon \sim \mathcal{N}(0, \sigma^2)$
- Fit a single neuron on that data (linear regression)
  - Learn $\boldsymbol{w}$ such that $Y \approx \langle X, \boldsymbol{w}\rangle$
- Expected squared error loss
  - $\ell(\boldsymbol{w}) = \mathbb{E}\left(Y - \langle X, \boldsymbol{w}\rangle\right)^2$



Why centering?

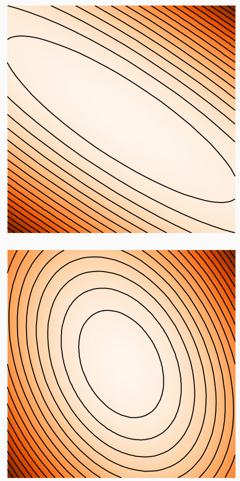
- Shape of quadratic function is controlled by Σ and $\boldsymbol{\mu}$

$$\begin{aligned}\ell &= \mathbb{E}\left(Y - \langle X, \boldsymbol{w}\rangle\right)^2 \\ &= \boldsymbol{w}^T\left(\Sigma + \boldsymbol{\mu}\boldsymbol{\mu}^T\right)\boldsymbol{w} + \text{linear term in } \boldsymbol{w} + \sigma^2\end{aligned}$$

- If we recenter and rescale before learning: $X \leftrightarrow \widehat{D}(X - \widehat{\mu})$

$$\begin{aligned}\ell &= \mathbb{E}\left(Y - \left\langle \widehat{D}(X - \widehat{\mu}), \boldsymbol{w}\right\rangle\right)^2 \\ &= \boldsymbol{w}^T\left(\widehat{D}\Sigma\widehat{D} + \widehat{D}(\boldsymbol{\mu} - \widehat{\mu})(\boldsymbol{\mu} - \widehat{\mu})^T\widehat{D}\right)\boldsymbol{w} \\ &\quad + \text{linear term in } \boldsymbol{w} + \sigma^2\end{aligned}$$

- If $\widehat{\mu}$ estimates $\boldsymbol{\mu}$ and $\widehat{D}$ estimates $(\operatorname{diag}\Sigma)^{-1/2}$



Optimization

- Training samples $\mathcal{T} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$ iid and drawn from $\mathcal{D}$, loss ℓ, neural network $F_{\boldsymbol{\theta}}$
- Empirical risk minimization (ERM)

$$\mathcal{L}_{\text{emp}}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i)$$

- Computing $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\text{emp}}(\boldsymbol{\theta})$ requires computing $\nabla_{\boldsymbol{\theta}}\ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i)$ for all $i = 1, \ldots, N$!
- Does not scale well training set size

Stochastic gradient descent (SGD)

- Empirical loss on full training set

$$\mathcal{L}_{\text{emp}}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i)$$

- Take a training sample $(\boldsymbol{x}_i, y_i)$ randomly from $\mathcal{T}$

$$\mathcal{L}_{(\boldsymbol{x}_i, y_i)}(\boldsymbol{\theta}) = \ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i)$$

- We use $\nabla_{\boldsymbol{\theta}}\ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i)$ instead of $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\text{emp}}(\boldsymbol{\theta})$

Minibatch stochastic gradient descent (SGD)

- Empirical loss on full training set

$$\mathcal{L}_{\text{emp}}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i)$$

- Minibatch loss where $\mathcal{B} \subset \mathcal{T}$ chosen randomly with $|\mathcal{B}| \ll N$

$$\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{B}|}\sum_{(\boldsymbol{x}, y)\in\mathcal{B}} \ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}), y)$$

- We use $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta})$ instead of $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\text{emp}}(\boldsymbol{\theta})$

Why does that work?

- What we really want to minimize is the expected risk

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\boldsymbol{x}, y\sim\mathcal{D}} \ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}), y)$$

- Applying the gradient operator $\nabla_{\boldsymbol{\theta}}$ yields

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x}, y\sim\mathcal{D}} \ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}), y) \\ &= \mathop{\mathbb{E}}_{\boldsymbol{x}, y\sim\mathcal{D}} \nabla_{\boldsymbol{\theta}}\ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}), y)\end{aligned}$$

- So $\nabla_{\boldsymbol{\theta}}\ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}), y)$ is an unbiased estimator of $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta})$
- So every $\nabla_{\boldsymbol{\theta}}\ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i)$ is an unbiased estimate of $\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta})$

Why does that work?

- Easy to get an observation of $\nabla_{\boldsymbol{\theta}} \ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}), y)$
- Unbiased but possibly of high variance
- Combining unbiased estimator to reduce variance

$$\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{B}|} \sum_{(\boldsymbol{x},y)\in\mathcal{B}} \ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}), y)$$

- $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}}$ is an unbiased estimate of $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{D}}$
- $\mathbb{E}_{\mathcal{B}} \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}} = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{D}}$
- Size of minibatch $|\mathcal{B}|$ is a trade-off term between good estimate of the gradient and cheap computations

Practical considerations

- Practically minibatches are not drawn randomly with replacement but randomly without replacement
- Training set is divided into disjoint minibatches $\mathcal{T} = \mathcal{B}_1 \sqcup \cdots \sqcup \mathcal{B}_p$
- Training over all minibatches (one pass of the training set) is called an epoch
- For next epoch we have a different set of minibatches $\mathcal{T} = \mathcal{B}'_1 \sqcup \cdots \sqcup \mathcal{B}'_p$

Toy model

- Data model $Y = \langle \boldsymbol{w}_0, X \rangle + \varepsilon$ with
  - $X \sim \mathcal{N}(0, \Sigma)$
  - $\varepsilon \sim \mathcal{N}(0, \sigma^2)$
- Minimization problem

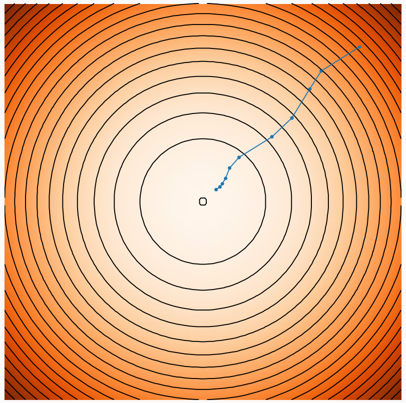
$$\operatorname*{arg\,min}_{\boldsymbol{w}\in\mathbb{R}^p} \mathbb{E}\left(Y - \langle \boldsymbol{w}, X \rangle\right)^2$$

- Expected loss is an ellipsoid centered at $\boldsymbol{w}_0$

$$\mathbb{E}\left(Y - \langle \boldsymbol{w}, X \rangle\right)^2 = (\boldsymbol{w} - \boldsymbol{w}_0)^T \Sigma (\boldsymbol{w} - \boldsymbol{w}_0) + \sigma^2$$

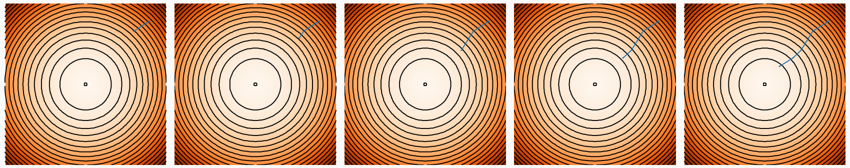
- Empirical loss is close to expected loss

$$\sum_{i=1}^{N} (\boldsymbol{y}_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle)^2 \approx (\boldsymbol{w} - \boldsymbol{w}_0)^T \Sigma (\boldsymbol{w} - \boldsymbol{w}_0) + \sigma^2$$

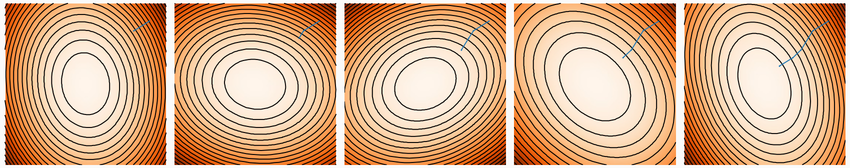


Stochastic gradient descent path

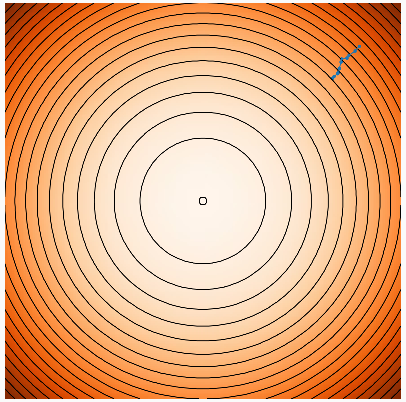
- Gradient descent for first minibatches, theoretical loss is $\mathcal{L}_{\mathcal{D}} = \mathbb{E}\left(Y - \langle \boldsymbol{w}, X \rangle\right)^2$



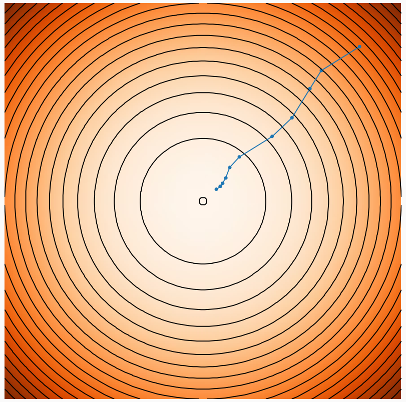
- What SGD is really seeing through minibatches, loss is $\mathcal{L}_{\mathcal{B}} = \dfrac{1}{|\mathcal{B}|} \sum_{(\boldsymbol{x},y)\in\mathcal{B}} (y - \langle \boldsymbol{w}, \boldsymbol{x} \rangle)^2$



Influence of learning rate

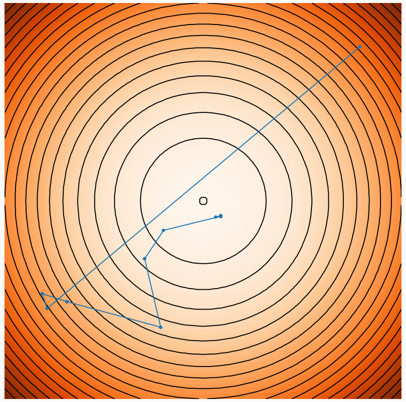


(a) $\eta = 0.01,\ |\mathcal{B}| = 10$

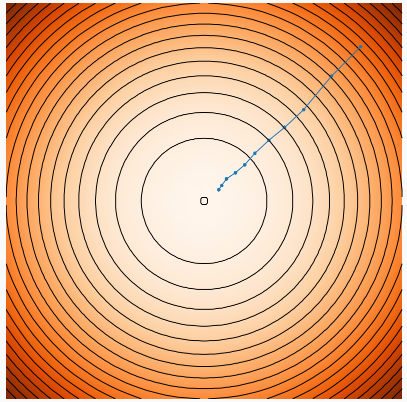


(b) $\eta = 0.1,\ |\mathcal{B}| = 10$

Influence of minibatch size



(a) $\eta = 0.1,\ |\mathcal{B}| = 1$



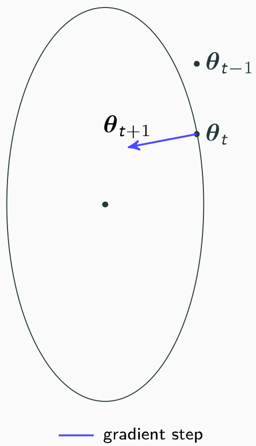
(b) $\eta = 0.1,\ |\mathcal{B}| = 100$

Gradient descent

- Starting point $\boldsymbol{\theta}_0$
- Update rule

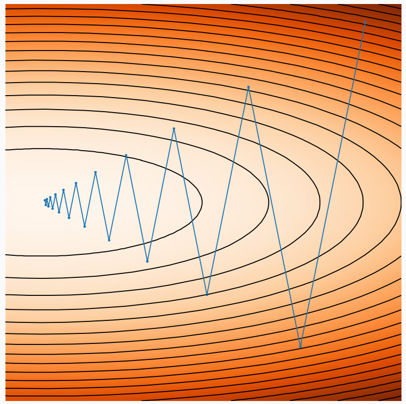
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \mathcal{L}$$

- Steps orthogonal to level lines
- Update only depends on $\mathcal{L}$ at $\boldsymbol{\theta}_t$: no memory!
- Problematic if gradient is noisy



Gradient descent

- If learning rate is high or loss landscape is bumpy SGD (and GD) trajectory might be erratic



(a) $\eta = 0.1$

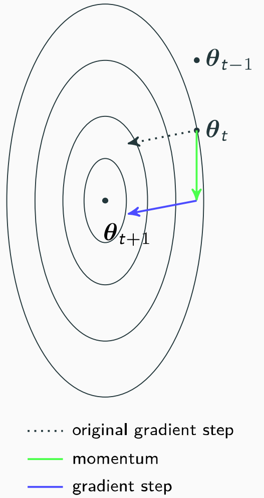
Polyak momentum

- Starting point $\boldsymbol{\theta}_0$
- Update rule

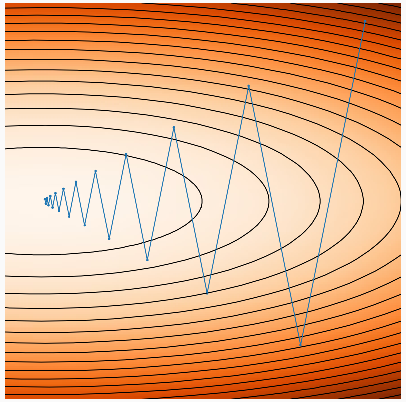
$$\mathbf{v}_{t+1} = \mu \mathbf{v}_t - \eta \nabla_{\boldsymbol{\theta}_t} \mathcal{L}$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{v}_{t+1}$$

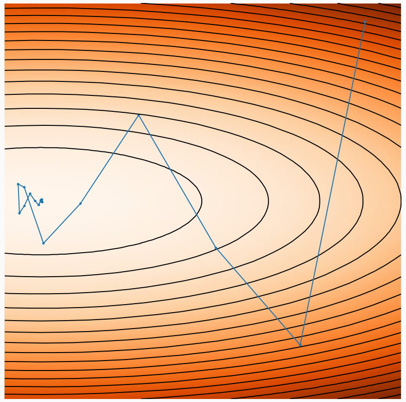
  with $\mathbf{v}_0 = 0$
- Exponentially weighted average on past gradients

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \sum_{i=0}^{t} \mu^i \nabla_{\boldsymbol{\theta}_{t-i}} \mathcal{L}$$



Momentum in action (GD)


(a) $\eta = 0.1$, no momentum. Gradients are abruptly changing


(b) $\eta = 0.1$, momentum $= 0.5$. Gradients change is smoothed out

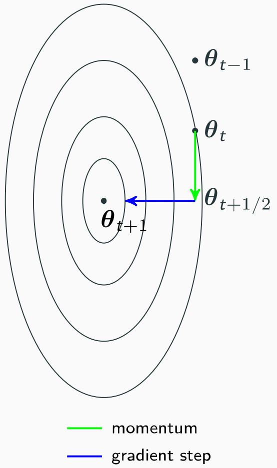
Nesterov accelerated momentum

- Starting point $\boldsymbol{\theta}_0$
- Update rule

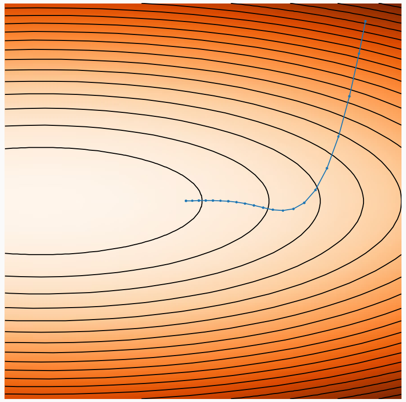
$$\boldsymbol{\theta}_{t+1/2} = \boldsymbol{\theta}_t + \mu \mathbf{v}_t$$
$$\mathbf{v}_{t+1} = \mu \mathbf{v}_t - \eta \nabla_{\boldsymbol{\theta}_{t+1/2}} \mathcal{L}$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{v}_{t+1}$$

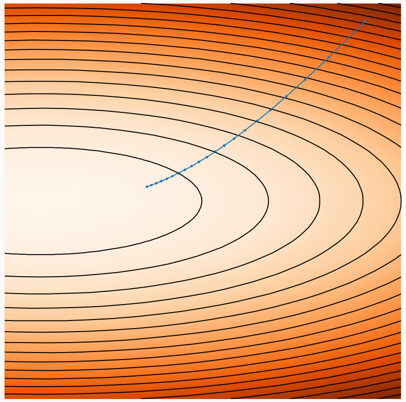
  with $\mathbf{v}_0 = 0$
- Exponentially weighted average on past gradients

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \sum_{i=0}^{t} \mu^i \nabla_{\boldsymbol{\theta}_{t-i+1/2}} \mathcal{L}$$



Problem with SGD (with momentum)


(a) SGD + momentum favors y-axis over x-axis


(b) More direct path toward minimum

Adagrad (Adaptive Gradient) from Duchi, Hazan, and Singer 2011

Normalize gradient coordinate-wise

- Starting point $\boldsymbol{\theta}_0$, $\boldsymbol{g}_0 = 0$
- Update rule

$$\boldsymbol{g}_{t+1} = \boldsymbol{g}_t + \nabla_{\boldsymbol{\theta}_t}\mathcal{L} \odot \nabla_{\boldsymbol{\theta}_t}\mathcal{L} \qquad (\odot \text{ is entrywise product})$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\nabla_{\boldsymbol{\theta}_t}\mathcal{L}}{\sqrt{\boldsymbol{g}_{t+1} + \varepsilon}} \qquad \text{(gradient normalization)}$$

- Add ε for numerical reasons
- Current gradient are rescaled to match average magnitude of all past gradients

RMSprop from Tieleman and Hinton 2012

Adagrad + exponentially weighted average on past gradients

- Starting point $\boldsymbol{\theta}_0$, $\boldsymbol{g}_0 = 0$, $\beta = 0.99$ by default
- Update rule

$$\boldsymbol{g}_{t+1} = \beta \boldsymbol{g}_t + (1-\beta)\nabla_{\boldsymbol{\theta}_t}\mathcal{L} \odot \nabla_{\boldsymbol{\theta}_t}\mathcal{L} \qquad (\odot \text{ is entrywise product})$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\nabla_{\boldsymbol{\theta}_t}\mathcal{L}}{\sqrt{\boldsymbol{g}_{t+1} + \varepsilon}}$$

Adam (Adaptive moment estimation) from Kingma and Ba 2015

RMSprop + momentum + bias correction

- Starting point $\boldsymbol{\theta}_0$, $\mathbf{v}_0 = 0$, $\boldsymbol{g}_0 = 0$, $\beta_1 = 0.9$, $\beta_2 = 0.999$
- Update rule

$$\mathbf{v}_{t+1} = \beta_1 \mathbf{v}_t + (1-\beta_1)\nabla_{\boldsymbol{\theta}_t}\mathcal{L} \qquad \text{(momentum)}$$

$$\boldsymbol{g}_{t+1} = \beta_2 \boldsymbol{g}_t + (1-\beta_2)\nabla_{\boldsymbol{\theta}_t}\mathcal{L} \odot \nabla_{\boldsymbol{\theta}_t}\mathcal{L} \qquad \text{(same as RMSprop)}$$

$$\tilde{\mathbf{v}}_{t+1} = \mathbf{v}_{t+1} / \left(1 - \beta_1^t\right)$$

$$\tilde{\boldsymbol{g}}_{t+1} = \boldsymbol{g}_{t+1} / \left(1 - \beta_2^t\right) \qquad \text{(bias correction)}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\tilde{\mathbf{v}}_{t+1}}{\sqrt{\tilde{\boldsymbol{g}}_{t+1} + \varepsilon}}$$

- Bias correction to compensate $\mathbf{v}_0 = 0$ and $\boldsymbol{g}_0 = 0$
  If $\nabla_{\boldsymbol{\theta}_t}\mathcal{L}$ is constant, we should have $\mathbf{v}_t = \nabla_{\boldsymbol{\theta}_t}\mathcal{L}$ and $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}_t}\mathcal{L} \odot \nabla_{\boldsymbol{\theta}_t}\mathcal{L}$

Weight initialization

Weights and biases need to be initialized to some value

- Weights initialized to zero means no learning at all
- Constant initialization means no diversification
- Biases can be initialized to zero

Random initialization

- Centered
- Constant variance per layer
- Which variance?

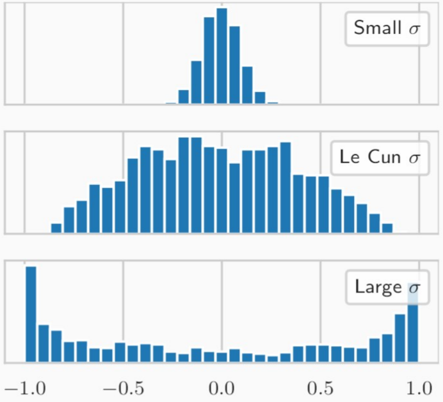
Effect of variance

Neural network

- 7 layers deep
- 1024 wide
- $\tanh$ activation

Distribution of outputs at layer 3

- With $\boldsymbol{w} \sim \mathcal{N}(0, \sigma^2)$
- Saturate or collapse



Le Cun's initialization from LeCun et al. 1998

Make variance of outputs constant across layers: $\operatorname{Var}\left(\boldsymbol{x}_i^{(k)}\right) = \operatorname{Var}\left(\boldsymbol{x}_j^{(k-1)}\right)$

- Suppose $\boldsymbol{x}_i^{(k-1)}$ iid centered, $\boldsymbol{w}_i^{(k)}$ iid centered and $\sigma \simeq \mathrm{Id}$ (tanh) then from $\boldsymbol{x}_i^{(k)} = \sigma\left(\left\langle \boldsymbol{w}_i^{(k)}, \boldsymbol{x}^{(k-1)} \right\rangle + \boldsymbol{b}_i^{(k)}\right)$ we have

$$\begin{aligned}
\operatorname{Var}\left(\boldsymbol{x}_i^{(k)}\right) &= n_{k-1} \operatorname{Var}\left(\boldsymbol{w}_{ij}^{(k)} \boldsymbol{x}_j^{(k-1)}\right) \\
&= n_{k-1} \operatorname{Var}\left(\boldsymbol{w}_{ij}^{(k)}\right) \mathbb{E}\left(\left(\boldsymbol{x}_j^{(k-1)}\right)^2\right) \\
&= n_{k-1}\sigma^2 \operatorname{Var}\left(\boldsymbol{x}_j^{(k-1)}\right)
\end{aligned}$$

- We have $\operatorname{Var}\left(\boldsymbol{w}_{ij}^{(k)}\right) = \dfrac{1}{n_{k-1}}$
- $\boldsymbol{w}_{ij}^{(k)} \sim \mathcal{N}(0, 1/n_{k-1})$

Glorot's initialization from Glorot and Bengio 2010

- From Le Cun's initialization we have $\sigma^2 = 1/n_{k-1}$
- Constant variance on gradients as well

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_i^{(k-1)}} = \sum_{j=1}^{n_k} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_j^{(k)}} \frac{\partial \mathbf{x}_j^{(k)}}{\partial \mathbf{z}_j^{(k)}} \frac{\partial \mathbf{z}_j^{(k)}}{\partial \mathbf{x}_i^{(k-1)}} \simeq \sum_{j=1}^{n_k} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_j^{(k)}} \boldsymbol{w}_{ji}^{(k)} \qquad \left(\frac{\partial \mathbf{x}_j^{(k)}}{\partial \mathbf{z}_j^{(k)}} = \sigma'\left(\mathbf{z}_j^{(k)}\right) \simeq 1\right)$$

- $\operatorname{Var}\left(\dfrac{\partial \mathcal{L}}{\partial \mathbf{x}_i^{(k-1)}}\right) = \operatorname{Var}\left(\dfrac{\partial \mathcal{L}}{\partial \mathbf{x}_j^{(k)}}\right)$ gives $\sigma^2 = 1/n_k$
- Harmonic mean of $1/n_k$ and $1/n_{k-1}$ gives $\sigma^2 = 2/(n_{k-1} + n_k)$

- Normally distributed

$$\boldsymbol{w}_{ij}^{(k)} \sim \mathcal{N}\left(0, \frac{2}{n_{k-1} + n_k}\right)$$

- Uniformly distributed

$$\boldsymbol{w}_{ij}^{(k)} \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{k-1} + n_k}}, \sqrt{\frac{6}{n_{k-1} + n_k}}\right)$$

He's initialization from He et al. 2015

Make variance of preactivation constant across layers $\operatorname{Var}\left(\mathbf{z}_i^{(k)}\right) = \operatorname{Var}\left(\mathbf{z}_j^{(k-1)}\right)$

$$\begin{aligned}
\operatorname{Var}\left(\mathbf{z}_i^{(k)}\right) &= n_{k-1} \operatorname{Var}\left(\boldsymbol{w}_{ij}^{(k)} \boldsymbol{x}_j^{(k-1)}\right) \\
&= n_{k-1} \operatorname{Var}\left(\boldsymbol{w}_{ij}^{(k)}\right) \mathbb{E}\left(\left(\boldsymbol{x}_j^{(k-1)}\right)^2\right) \\
&= n_{k-1} \operatorname{Var}\left(\boldsymbol{w}_{ij}^{(k)}\right) \mathbb{E}\left(\operatorname{ReLU}\left(\mathbf{z}_j^{(k-1)}\right)^2\right) \\
&= n_{k-1}\sigma^2 \frac{1}{2} \operatorname{Var}\left(\mathbf{z}_j^{(k-1)}\right)
\end{aligned}$$

We then have $\sigma^2 = 2/n_{k-1}$

- Normally distributed

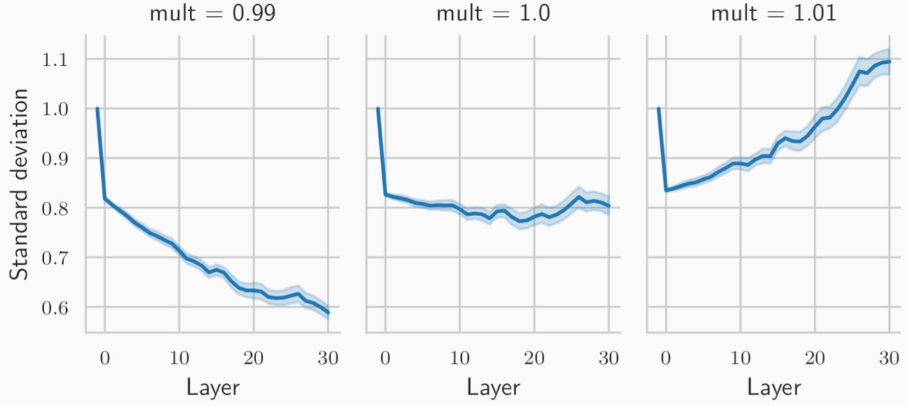
$$\boldsymbol{w}_{ij}^{(k)} \sim \mathcal{N}\left(0, \frac{2}{n_{k-1} + n_k}\right)$$

- Uniformly distributed

$$\boldsymbol{w}_{ij}^{(k)} \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{k-1} + n_k}}, \sqrt{\frac{6}{n_{k-1} + n_k}}\right)$$

Illustration of He's initialization

- mult is an additional ratio wrt He's initialization: $\sigma^2 = \text{mult} \cdot 2/n_{k-1}$



Batch normalization from Ioffe and Szegedy 2015

- Normalize each pre-activation independently from the minibatch statistics

$$\boldsymbol{\mu}^{(k)} = \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{x}^{(k)} \in \mathcal{B}^{(k)}} \mathbf{z}^{(k)}$$

$$\sigma_i^{(k)} = \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{x}^{(k)} \in \mathcal{B}^{(k)}} \left(\mathbf{z}_i^{(k)} - \boldsymbol{\mu}_i^{(k)}\right)^2$$

- For each element in the minibatch, replace $\mathbf{z}_i^{(k)}$ and $\boldsymbol{x}_i^{(k+1)}$ by

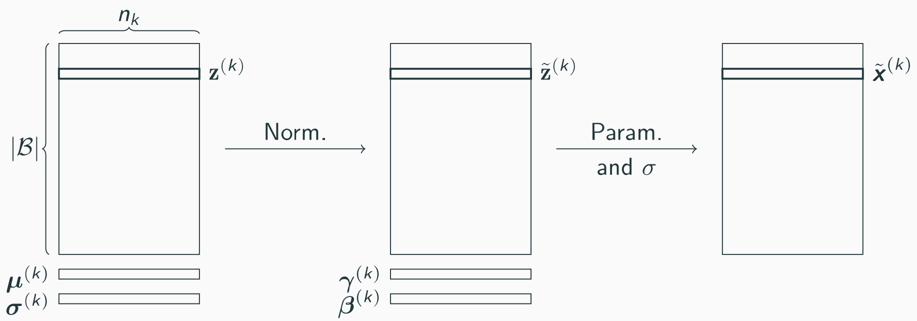
$$\tilde{\mathbf{z}}_i^{(k)} = \frac{\mathbf{z}_i^{(k)} - \boldsymbol{\mu}_i^{(k)}}{\sigma_i^{(k)}} \qquad \tilde{\boldsymbol{x}}_i^{(k)} = \sigma\left(\gamma_i^{(k)} \tilde{\mathbf{z}}_i^{(k)} + \boldsymbol{\beta}_i^{(k)}\right)$$

- $\gamma_i^{(k)}$ and $\boldsymbol{\beta}_i^{(k)}$ are $2n_k$ extra parameters
- The $\boldsymbol{b}_i$'s from $\mathbf{z}_i^{(k)} = \left\langle \boldsymbol{w}_i^{(k)}, \boldsymbol{x}^{(k)} \right\rangle + \boldsymbol{b}_i^{(k)}$ are useless ($\tilde{\mathbf{z}}_i^{(k)}$ does not depend on $\boldsymbol{b}_i$)

Batch normalization

Normalize minibatch just like it is a mini-dataset before preprocessing

- Allow to learn a common rescaling

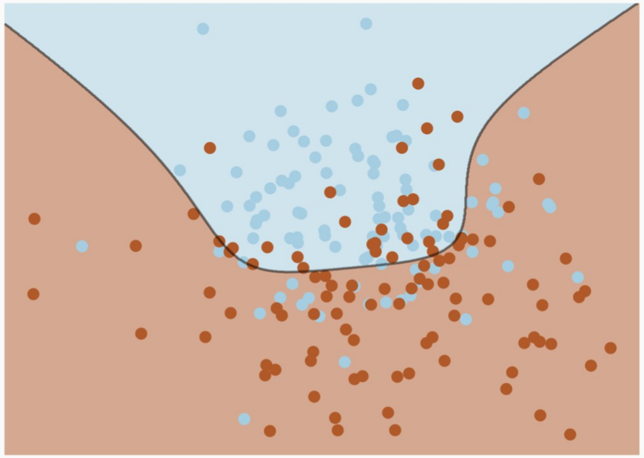


- At test time $\boldsymbol{\mu}^{(k)}$ and $\boldsymbol{\sigma}^{(k)}$ are replaced by estimations from a running average

Regularization

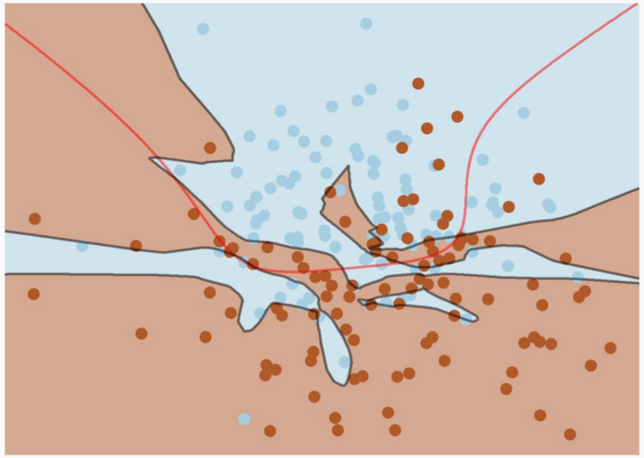
Toy dataset

- 200 samples, 2 classes
- Gaussian mixture model
- Bayes decision boundary



Standard neural network classification

- Neural network
  - 1 hidden layer with 50 units
  - ~ 200 parameters
- SGD algorithm
  - learning rate: 0.1
  - momentum: 0.9



Regularizing the ERM

- Empirical risk minimization (ERM)

$$\arg\min_{\theta \in \Theta} \mathcal{L}_{\mathcal{B}} = \arg\min_{\theta \in \Theta} \frac{1}{|\mathcal{B}|} \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{B}} \ell(F_\Theta(\boldsymbol{x}), \boldsymbol{y})$$

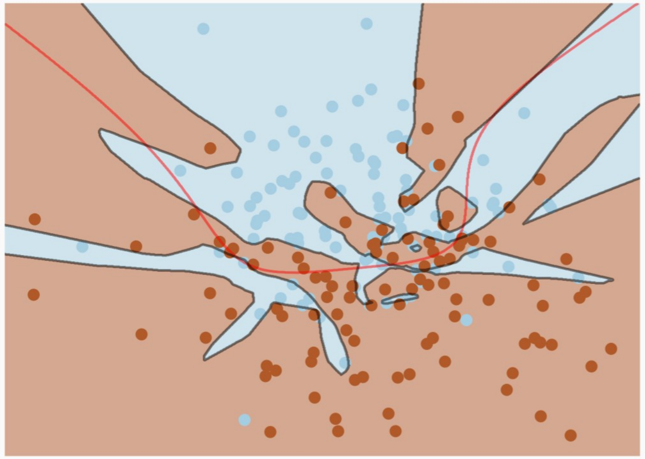
- L_2 penalizing term

$$\arg\min_{\theta \in \Theta} \mathcal{L}_{\mathcal{R}} = \arg\min_{\theta \in \Theta} \frac{1}{|\mathcal{B}|} \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{B}} \ell(F_\Theta(\boldsymbol{x}), \boldsymbol{y}) + \lambda \sum_{k=1}^{K} \left\| W^{(k)} \right\|_F$$

- Biases terms are not regularized
- λ is the tradeoff parameter called *weight decay*

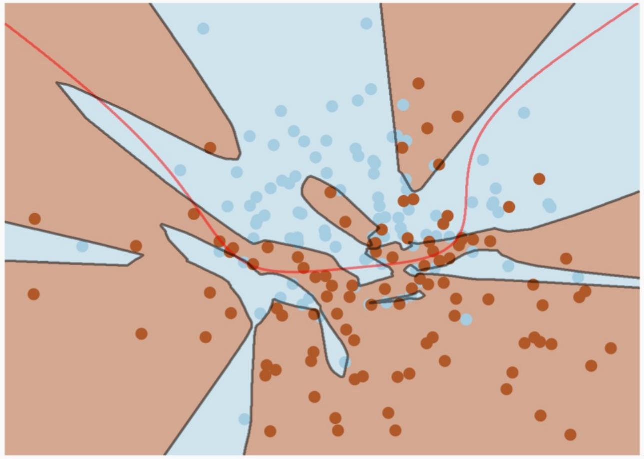
Weight decay: an example

- SGD algorithm
  - learning rate: 0.1
  - momentum: 0.9
- **weight decay:** 10^{-4}



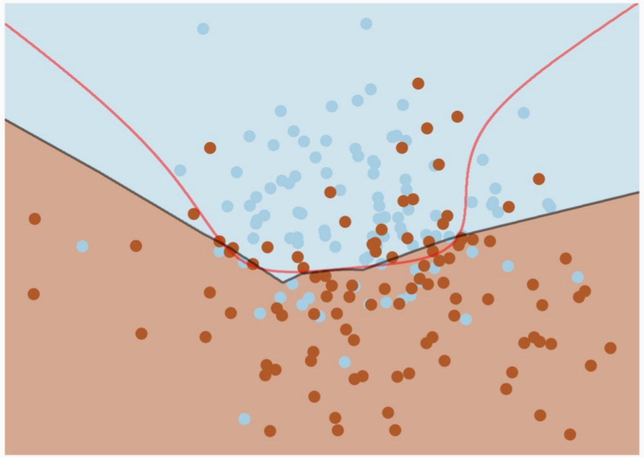
Weight decay: an example

- SGD algorithm
  - learning rate: 0.1
  - momentum: 0.9
- **weight decay:** 10^{-3}



Weight decay: an example

- SGD algorithm
  - learning rate: 0.1
  - momentum: 0.9
- **weight decay:** 10^{-2}



Weight decay: gradient

- One extra term in the loss

$$\mathcal{L}_{\mathcal{R}} = \mathcal{L}_{\mathcal{B}} + \lambda \sum_{k=1}^{K} \left\| W^{(k)} \right\|_F$$

- Gradient is easy to get

$$\frac{\partial \mathcal{L}_{\mathcal{R}}}{\partial \boldsymbol{w}_{ij}^{(k)}} = \frac{\partial \mathcal{L}_{\mathcal{B}}}{\partial \boldsymbol{w}_{ij}^{(k)}} + 2\lambda \boldsymbol{w}_{ij}^{(k)}$$

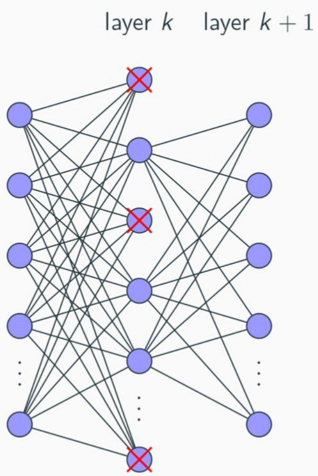
- Gradients w.r.t. $\boldsymbol{b}_i^{(k)}$ are unchanged

Dropout from Srivastava et al. 2014

- Randomly kill nodes in layers during training time

$$\boldsymbol{x}^{(k+1)} = \sigma\left(\left(W^{(k+1)}\right)^T \left(\boldsymbol{x}^{(k)} \odot \mathbf{h}^{(k)}\right) + \boldsymbol{b}^{(k+1)}\right)$$

with $\mathbf{h}^{(k)} \in \{0,1\}^{n_k}$
- Bernoulli distribution $\mathbf{h}^{(k)} \sim \mathcal{B}(h)^{\otimes n_k}$
- h is the dropout rate
- Prevent "co-adaptation" of neurons, encourage redundancy



Dropout at test-time

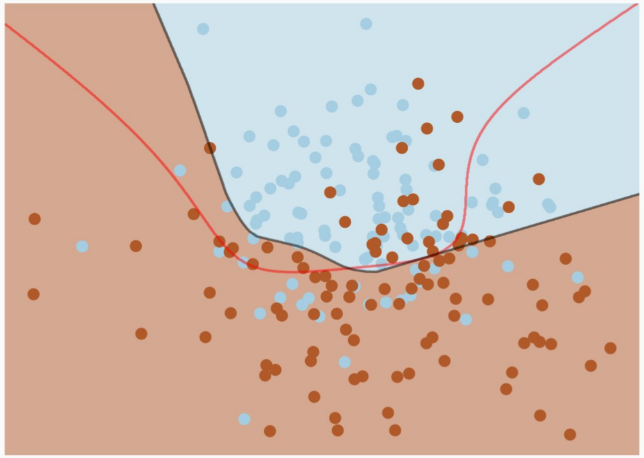
- Output is stochastic
- Replace pre-activation by expected pre-activation

$$\begin{aligned}\mathbb{E}_{\mathbf{h}^{(k)}}\left(\mathbf{z}^{(k+1)}\right) &= \mathbb{E}\left(\left(W^{(k+1)}\right)^T\left(\boldsymbol{x}^{(k)} \odot \mathbf{h}^{(k)}\right) + \boldsymbol{b}^{(k+1)}\right)\\ &= \left(W^{(k+1)}\right)^T\left(\boldsymbol{x}^{(k)} \odot \mathbb{E}\left(\mathbf{h}^{(k)}\right)\right) + \boldsymbol{b}^{(k+1)}\\ &= \left(W^{(k+1)}\right)^T\left(h\boldsymbol{x}^{(k)}\right) + \boldsymbol{b}^{(k+1)}\end{aligned}$$

- At test-time, no dropout but rescale output by h

Dropout: an example

- SGD algorithm
  - learning rate: 0.1
  - momentum: 0.9
- **dropout rate:** 0.7
- Mostly present at
  - Fully connected layers
  - Embeddings
- Usually $h = 0.5$

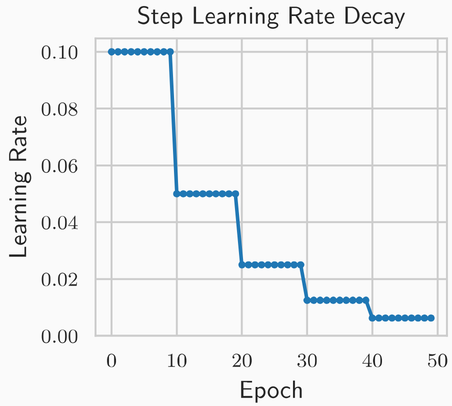


Step learning rate decay

- Milestones: $m_1, \ldots, m_l$
- Learning rate decays at each milestone

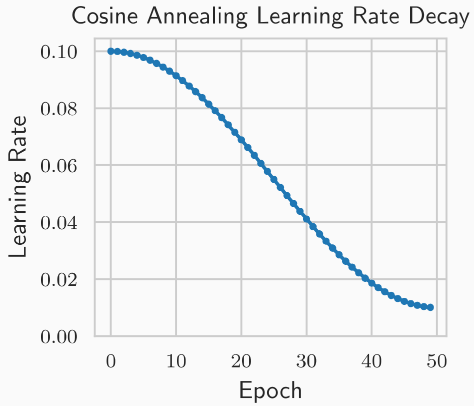
$$\eta_t = \eta \cdot \gamma^{\#\{i,\, m_i \leqslant t\}}$$

- Example
  - Learning rate: $\eta = 0.1$
  - Learning rate decay: $\gamma = 1/2$
  - Milestones: $m_i = 10i$



Cosine Annealing Learning Rate Decay from Loshchilov and Hutter 2017

- Cosine between η_{min} and η_{max}
- Example
  - $\eta_{\text{min}} = 0.01$
  - $\eta_{\text{max}} = 0.1$



Cosine Annealing Learning Rate Decay from Loshchilov and Hutter 2017

- Cosine between η_{min} and η_{max}
- Example
  - $\eta_{\text{min}} = 0.01$
  - $\eta_{\text{max}} = 0.1$
- Cycle length is increasing by a factor