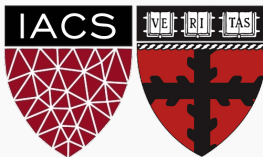# Advanced Section #1: Moving averages, optimization algorithms, understanding dropout and batch normalization

AC 209B: Data Science 2

Javier Zazo   Pavlos Protopapas

Lecture Outline

Moving averages

Optimization algorithms

Tuning the learning rate

Gradient checking

How to address overfitting

Dropout

Batch normalization

Moving averages

Moving averages

- Given a stationary process $x[n]$ and a sequence of observations $x_1, x_2, \ldots, x_n, \ldots$, we want to estimate the average of all values *dynamically*.

- We can use a *moving average* for instant n:

$$\overline{x}_{n+1} = \frac{1}{n} \left(x_1 + x_2 + \ldots + x_n \right)$$

- **To save computations and memory**:

$$\overline{x}_{n+1} = \frac{1}{n} \sum_{i=1}^{n} x_i = \frac{1}{n} \left(x_n + \sum_{i=1}^{n-1} x_i \right) = \frac{1}{n} \left(x_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} x_i \right)$$
$$= \frac{1}{n} \left(x_n + (n-1)\overline{x}_n \right) = \overline{x}_n + \frac{1}{n} \left(x_n - \overline{x}_n \right)$$

- Essentially, for $\alpha_n = 1/n$,

$$\boxed{\overline{x}_{n+1} = \overline{x}_n + \alpha_n \left(x_n - \overline{x}_n \right)}$$

Weighted moving averages

- Previous step size $\alpha_n = 1/n$ is dynamic.

- From **stochastic approximation theory**, the estimate converges to the true value with probability 1, if

$$\sum_{i=1}^{\infty} \alpha_i = \infty \qquad \text{and} \qquad \sum_{i=1}^{\infty} \alpha_i^2 < \infty$$

- $\alpha_n = \frac{1}{n}$ satisfies the previous conditions.

- Constant α does not satisfy the second!!

- This can be useful to track ***non-stationary*** processes.
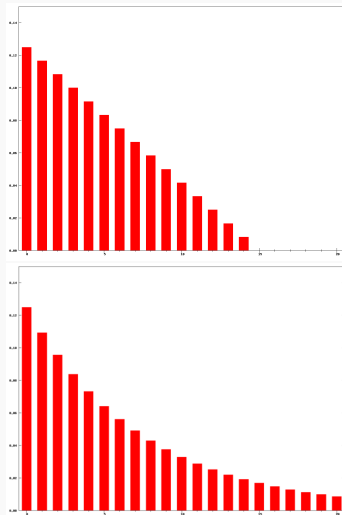
Exponentially weighted moving average

▶ Update rule for constant step size is

$$
\begin{aligned}
\overline{x}_{n+1} &= \overline{x}_n + \alpha\left(x_n - \overline{x}_n\right) \\
&= \alpha x_n + (1-\alpha)\overline{x}_n \\
&= \alpha x_n + (1-\alpha)[\alpha x_{n-1} + (1-\alpha)\overline{x}_{n-1}] \\
&= \alpha x_n + (1-\alpha)\alpha x_{n-1} + (1-\alpha)^2 \overline{x}_{n-1}] \\
&= \alpha x_n + (1-\alpha)\alpha x_{n-1} + (1-\alpha)^2 \alpha x_{n-2} + \ldots + (1-\alpha)^{n-1}\alpha x_1 + (1-\alpha)^n \overline{x}_1 \\
&= \boxed{(1-\alpha)^n \overline{x}_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} x_i}
\end{aligned}
$$

▶ Note that $(1-\alpha)^n + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = 1$.

▶ With infinite terms we get

$$
\lim_{n\to\infty} \overline{x}_n = \lim_{n\to\infty} \frac{x_n + (1-\alpha)x_{n-1} + (1-\alpha)^2 x_{n-2} + (1-\alpha)^3 x_{n-3} + \ldots}{1 + (1-\alpha) + (1-\alpha)^2 + (1-\alpha)^3 + \ldots}
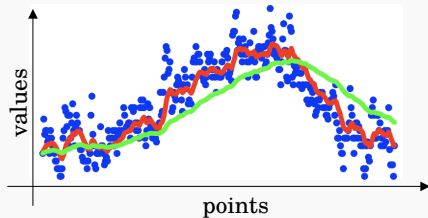$$

Exponentially weighted moving average

▶ Recap update rule, but change $1 - \alpha = \beta$

$$\overline{x}_{n-1} = \beta\overline{x}_{n-1} + (1 - \beta)x_n,$$

▶ β controls the amount of points to consider (variance):

▶ **Rule of thumb:**

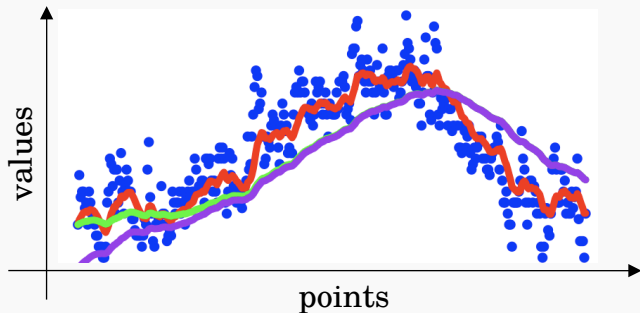$$N = \frac{1 + \beta}{1 - \beta} \text{ amounts to } 86\% \text{ of influence.}$$

– $\beta = 0.9$ corresponds to 19 points.
– $\beta = .98$ corresponds to 99 points (wide window).
– $\beta = 0.5$ corresponds to 3 points (susceptible to outliers).

Bias correction

- ► The rule of thumb works for sufficiently large N.
- ► Otherwise, the first values are biased.
- ► We can correct the variance with:

$$v_t^{\text{corrected}} = \frac{v_t}{1 - \beta^t}.$$

Bias correction II

- ▶ The bias correction can in practice be ignored (Keras does not implement it).
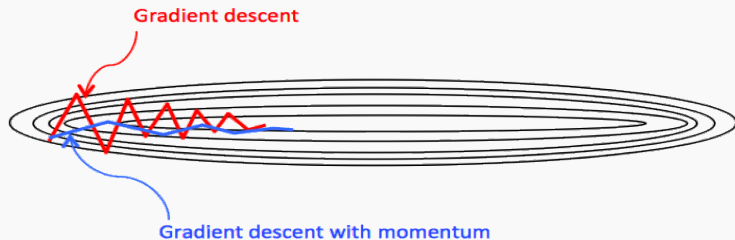- ▶ Origin of bias comes from zero initialization:

$$\overline{x}_{n+1} = \beta^n \underbrace{\overline{x}_1}_{0} + (1 - \beta) \sum_{i=1}^{n} \beta^{n-i} x_i$$

- ▶ Derivation:

$$\mathbb{E}[\overline{x}_{n+1}] = \mathbb{E}\left[(1 - \beta) \sum_{i=1}^{n} \beta^{n-i} x_i\right]$$

$$= \mathbb{E}[x_n](1 - \beta) \sum_{i=1}^{n} \beta^{n-i} + \zeta$$

$$= \mathbb{E}[x_n](1 - \beta^n) + \zeta$$

Optimization algorithms

Gradient descent

- ► Gradient descent will have high variance if the problem is ill-conditioned.
- ► Aim to estimate directions of high variance and reduce their influence.
- ► Descent with momentum, RMSprop or Adam, help reduce the variance and speed up convergence.

Gradient descent with momentum

▶ The algorithm:
  1: On iteration t for W update:
  2:    Compute dW on current mini-batch.
  3:    $v_{dW} = \beta v_{dW} + (1 - \beta)dW$.
  4:    $W = W - \alpha v_{dW}$.

▶ Gradient with momentum performs an exponential moving average over the gradients.

▶ This will reduce the variance and give more stable descent directions.

▶ Bias correction is usually not applied.

RMSprop

▶ The algorithm:

1: On iteration t for W update:

2:     Compute dW on current mini-batch.

3:     $s_{dW} = \beta_2 s_{dW} + (1 - \beta_2)dW^2$.

4:     $W = W - \alpha \frac{dW}{\sqrt{s_{dW}} + \epsilon}$.

▶ $\epsilon = 10^{-8}$ controls numerical stability.

▶ High variance gradients will have larger values $\rightarrow$ the squared averages will be large $\rightarrow$ reduces the step size.

▶ Allows a higher learning rate $\rightarrow$ faster convergence.

Adaptive moment estimation (Adam)

► The algorithm:

1: On iteration t for W update:

2:     Compute dW on current mini-batch.

3:     $v_{dW} = \beta_1 v_{dW} + (1 - \beta_1)dW$.

4: $s_{dW} = \beta_2 s_{dW} + (1 - \beta_2)dW^2$.

5:     $v^{\text{corrected}} = \frac{v_{dW}}{1 - \beta_1^t}$

6: $s^{\text{corrected}} = \frac{s_{dW}}{1 - \beta_2^t}$

7:     $W = W - \alpha \frac{v^{\text{corrected}}}{\sqrt{s_{dW}} + \epsilon}$.

AMSGrad

- ▶ Adam/RMSprop fail to converge on certain convex problems.

- ▶ Reason is that some important descent directions are weakened by high second order estimations.

- ▶ AMSGrad proposes a conservative fix where second order moment estimator can only increase.

- ▶ The algorithm:

  1: On iteration t for W update:

  2:     Compute dW on current mini-batch.

  3:     $v_{dW}^{n+1} = \beta_1 v_{dW}^n + (1 - \beta_1)dW$.

  4:     $s_{dW}^{n+1} = \beta_2 s_{dW}^n + (1 - \beta_2)dW^2$.

  5:     $\hat{s}_{dW}^{n+1} = \max(\hat{s}_{dW}^n, s_{dW}^{n+1})$

  6:     $W = W - \alpha \frac{v^{\text{corrected}}}{\sqrt{\hat{s}_{dW}^{n+1}} + \epsilon}$.
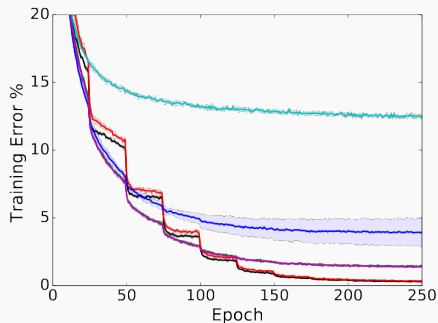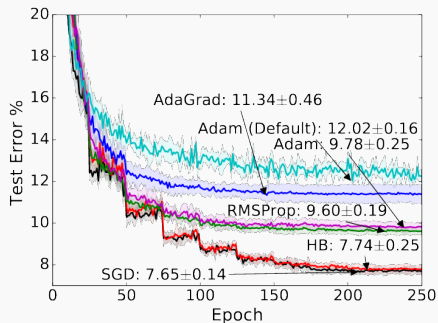
(a) CIFAR-10 (Train) (b) CIFAR-10 (Test)

Tuning the learning rate

Cyclical Learning Rates for Neural Networks

- ▶ Use cyclical learning rates to escape local extreme points.

- ▶ Saddle points are abundant in high dimensions, and convergence becomes very slow. Furthermore, they can help escape sharp local minima (overfitting).

- ▶ Cyclic learning rates raise the learning rate periodically: **short term negative effect and yet achieve a longer term beneficial effect.**

- ▶ Decreasing learning rates may still help reduce error towards the end.
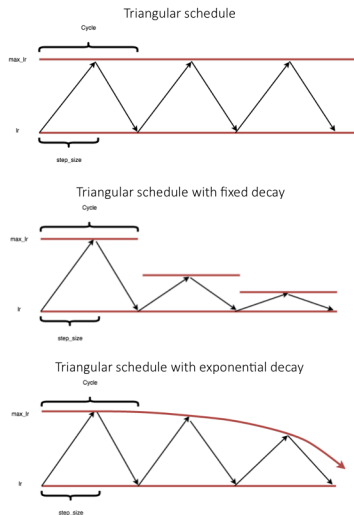
Estimating the learning rate

- ▶ How can we get a good LR estimate?
- ▶ Start with a small LR and increase it on every batch exponentially.
- ▶ Simultaneously, compute loss function on validation set.
- ▶ This also works for finding bounds for cyclic LRs.
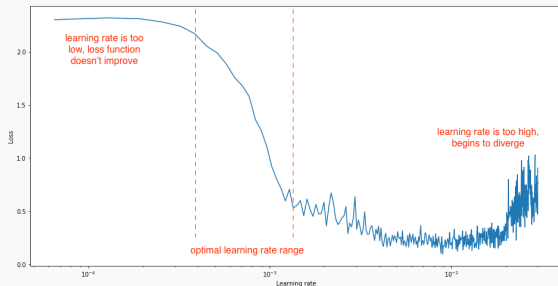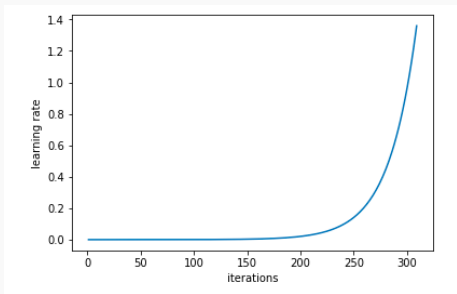
SGD with Warm Restarts

▶ Key idea: restart every T_i epochs. **Record best estimates before restart**.

▶ Restarts are not from scratch, but from last estimate, and learning rate is increased.

$$\alpha_t = \alpha_{\min}^i + \frac{1}{2}(\alpha_{\max}^i - \alpha_{\min}^i)(1 + \cos(\frac{T_c ur}{T_i}\pi))$$

▶ The cycle can be lengthened with time.

▶ $\alpha_{\min}^i$ and $\alpha_{\max}^i$ can be decayed after a cycle.

Cifar10 (L=100,k=24, B=300 epochs)
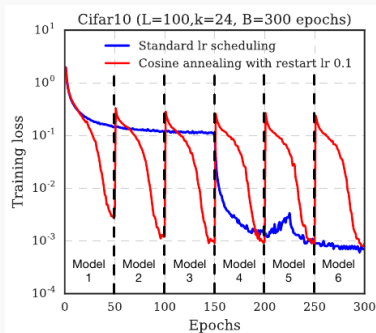
Snapshot ensembles: Train 1, get M for free

► Ensemble networks are much more robust and accurate than individual networks.

► They constitute another type of regularization technique.

► The novelty is to train a single neural network, but obtain M different models.

► The idea is to converge to M different local optima, and save network parameters.
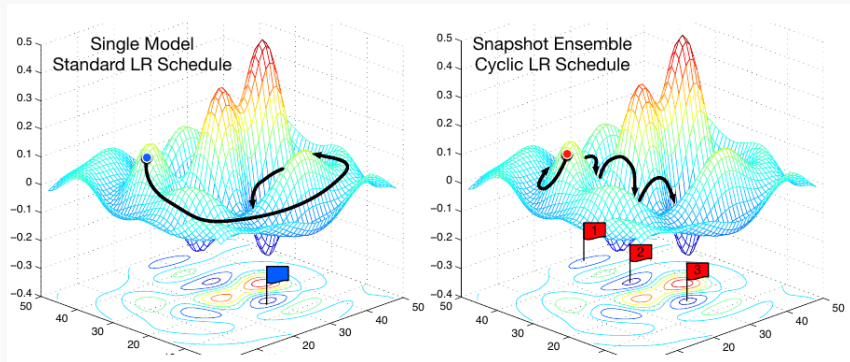
Snapshot ensembles II

- ▶ Different initialization points, or hyperarameter choices may converge to different local minima.

- ▶ Although these local minima may perform similarly in terms of averaged errors, they may not make the same mistakes.

- ▶ Ensemble methods train many NN, and then optimize through majority vote, or averaging of the prediction outputs.

- ▶ The proposal uses a cycling step size procedure (cosine), in which the learning rate is abruptly raised and wait for new convergence.

- ▶ The final ensemble consists of snapshots of the optimization path.

Snapshot ensembles III

	Method	C10	C100	SVHN	Tiny ImageNet
ResNet-110	Single model	5.52	28.02	1.96	46.50
	NoCycle Snapshot Ensemble	5.49	26.97	1.78	43.69
	SingleCycle Ensembles	6.66	24.54	1.74	42.60
	Snapshot Ensemble ($\alpha_0 = 0.1$)	5.73	25.55	**1.63**	40.54
	Snapshot Ensemble ($\alpha_0 = 0.2$)	**5.32**	**24.19**	1.66	**39.40**
Wide-ResNet-32	Single model	5.43	23.55	1.90	39.63
	Dropout	4.68	22.82	1.81	36.58
	NoCycle Snapshot Ensemble	5.18	22.81	1.81	38.64
	SingleCycle Ensembles	5.95	21.38	1.65	35.53
	Snapshot Ensemble ($\alpha_0 = 0.1$)	**4.41**	**21.26**	1.64	35.45
	Snapshot Ensemble ($\alpha_0 = 0.2$)	4.73	21.56	**1.51**	**32.90**
DenseNet-40	Single model	5.24^*	24.42^*	1.77	39.09
	Dropout	6.08	25.79	1.79^*	39.68
	NoCycle Snapshot Ensemble	5.20	24.63	1.80	38.51
	SingleCycle Ensembles	5.43	22.51	1.87	38.00
	Snapshot Ensemble ($\alpha_0 = 0.1$)	4.99	23.34	**1.64**	37.25
	Snapshot Ensemble ($\alpha_0 = 0.2$)	**4.84**	**21.93**	1.73	**36.61**
DenseNet-100	Single model	3.74^*	19.25^*	-	-
	Dropout	3.65	18.77	-	-
	NoCycle Snapshot Ensemble	3.80	19.30	-	-
	SingleCycle Ensembles	4.52	18.38	-	-
	Snapshot Ensemble ($\alpha_0 = 0.1$)	3.57	18.12	-	-
	Snapshot Ensemble ($\alpha_0 = 0.2$)	**3.44**	**17.41**	-	-

Gradient checking

Gradient checking

▶ Useful technique to debug code of manual implementations of neural networks.

▶ Not intended for training of networks, but it can help to identify errors in a backpropagation implementation.

▶ Derivative of a function:

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x + \epsilon) - f(x - \epsilon)}{2\epsilon} \approx \frac{f(x + \epsilon) - f(x - \epsilon)}{2\epsilon}.$$

▶ The approximation error is in the order $O(\epsilon^2)$.

▶ In the multivariate case, the ϵ term affects a single component:

$$\frac{df(\theta)}{d\theta_r} \approx \frac{f(\theta_r^+) - f(\theta_r^-)}{2\epsilon}$$

where $\theta_r^+ = (\theta_1, \ldots, \theta_r + \epsilon, \ldots, \theta_n)$, $\theta_r^- = (\theta_1, \ldots, \theta_r - \epsilon, \ldots, \theta_n)$.

Algorithm for gradient checking

1: Reshape input vector in a column vector θ.
2: **for** each r component **do**
3:     $\theta_{\text{old}} \leftarrow \theta_r$
4:     Calculate $f(\theta_r^+)$ and $f(\theta_r^-)$.
5:     Compute approx. $\frac{df(\theta)}{d\theta_r}$.
6:     Restore $\theta_r \leftarrow \theta_{\text{old}}$
7: **end for**
8: Verify relative error is below some threshold:

$$\xi = \frac{\|d\theta^{\text{approx}} - d\theta\|}{\|d\theta^{\text{approx}}\| + \|d\theta\|}$$

How to address overfitting

Estimators

► Point estimation is the attempt to provide the single "best" prediction of some quantity of interest:

$$\hat{\boldsymbol{\theta}}_m = g(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}).$$

  – $\boldsymbol{\theta}$: true value.
  – $\hat{\boldsymbol{\theta}}_m$: estimator for m samples.

► Frequentist perspective: $\boldsymbol{\theta}$ fixed but unkwown.

► Data is random $\implies \hat{\boldsymbol{\theta}}_m$ is a r.v.

Bias and Variance

▶ Bias: expected deviation from the true value.

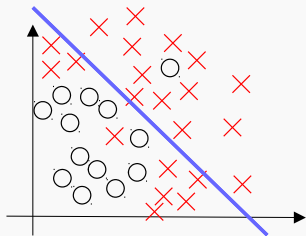▶ Variance: deviation from the expected estimator.

Examples:
- Sample mean: $\hat{\mu}_m = \frac{1}{m} \sum_i \mathbf{x}^{(i)}$
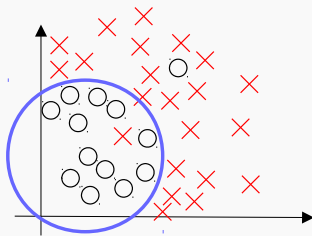- Sample variance $\hat{\sigma}_m^2 = \frac{1}{m} \sum_i (x^{(i)} - \hat{\mu}_m)^2$:

$$\mathbb{E}[\hat{\sigma}_m^2] = \frac{m-1}{m}\sigma^2$$

- Unbiased sample variance: $\tilde{\sigma}_m^2 = \frac{1}{m-1} \sum_i (x^{(i)} - \hat{\mu}_m)^2$

▶ How to choose estimators with different statistics?
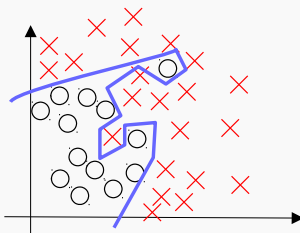- Mean square error (MSE).
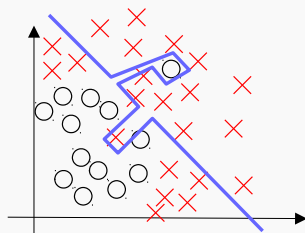- Cross-validation: **empirical**.
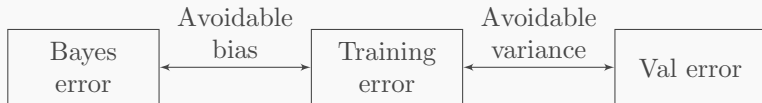
high bias &
underfitting

appropriate

high variance &
overfitting

high bias & variance

Diagnose bias-variance

- ▶ In high dimensions we cannot draw decision curves to inspect bias-variance.
- ▶ We calculate error values to infer the source of errors on the training set, as well as on the val set.
- ▶ To determine bias, we need a base line, such as human level performance.

```
    Avoidable              Avoidable
      bias                  variance
┌─────────┐            ┌──────────┐            ┌───────────┐
│  Bayes  │ ◄────────► │ Training │ ◄────────► │ Val error │
│  error  │            │  error   │            │           │
└─────────┘            └──────────┘            └───────────┘
```

- ▶ Example:

Human level error	$\approx 0\%$			
Training error	0.5%	15%	1%	12%
Val error	1%	16%	11%	20%
	low bias low variance	high bias	high variance	high bias high variance

Orthogonalization

Human
level error

$\uparrow$ Avoidable
$\downarrow$ bias

$\left\{\begin{array}{l}\text{Train a bigger model} \\ \text{Train longer/better optimization alg.} \\ \text{NN architecture/hyperparameter search.}\end{array}\right.$

Training
error

$\uparrow$ Avoidable
$\downarrow$ variance

$\left\{\begin{array}{l}\text{Get more data.} \\ \text{Use regularization (L2, dropout, data aug., etc.)} \\ \text{NN architecture/hyperparameter search.}\end{array}\right.$

Val error
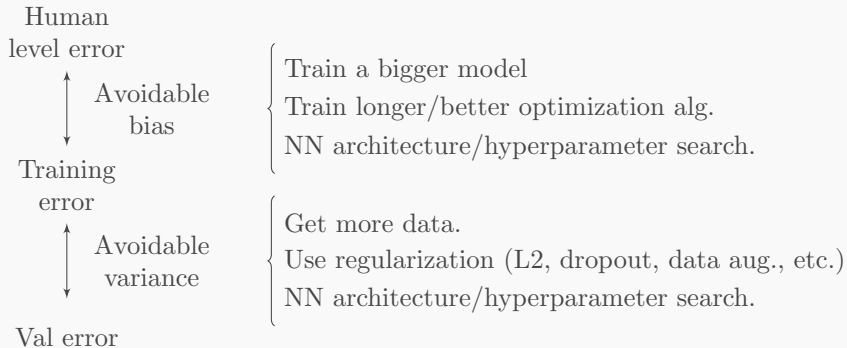
- ▶ Orthogonalization aims to decompose the process to adjust NN performance.
- ▶ It assumes the errors come from different sources and uses a systematic approach to minimize them.
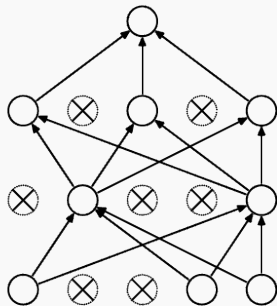- ▶ Early stopping is a popular regularization mechanism, but couples the bias and variance errors.

Dropout

Dropout

- ▶ Regularization technique for deep NN.
- ▶ Employed at training time.
- ▶ Eliminates the output of some units randomly.
- ▶ Can be used in combination with other regularization techniques (such as L2, batch normalization, etc.).
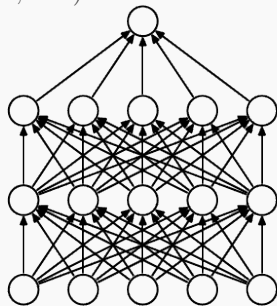

(a) Standard Neural Net (b) After applying dropout.

Motivation and direct implementation

- **Purpose**: prevent the co-adaptation of feature detectors for a set of neurons, and avoid overfitting.
  - It enforces the neurons to develop an individual role on their own given an overall population behavior.
  - Training weights are encouraged to be spread along the NN, because no neuron is permanent.
- **Interpretation**: training examples provide gradients from different, randomly sampled architectures.
- **Direct implementation:**
  - At training time: eliminate the output of some units randomly.
  - At test time: all units are present.

Present with probability p

(a) At training time

Always present
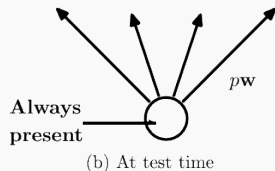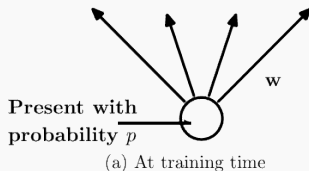
(b) At test time

Inverted dropout

▶ Current implementations use ***inverted dropout***
  – Weighting is performed during training.
  – Does not require re-weighting at test time.

▶ In particular, for layer l,

$$z^{[l]} = \frac{1}{p_l} W^{[l]} D^{[l]} a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g(z^{[l]}),$$

▶ Notation:

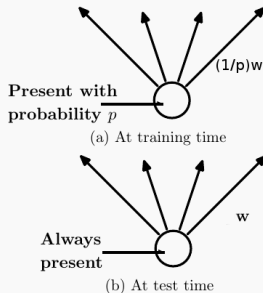p_l : Retention probability.
$D^{[l]}$: Dropout activations.
$a^{[l-1]}$: Output from previous layer.
$W^{[l]}$: Layer weights.
$b^{[l]}$: Offset weights.
$z^{[l]}$: Linear output.
$g(\cdot)$: Nonlinear activation function.

Present with
probability p

(1/p)w

(a) At training time

Always
present

w

(b) At test time

Understanding dropout

We aim to understand dropout as a regularization technique on simplified neural architectures such as:

- ▶ *Linear networks*.
- ▶ *Logistic regression*.
- ▶ *Deep networks*.

These results are are based on the following reference:

Pierre Baldi and Peter J Sadowski, "Understanding dropout,"
in *Advances in Neural Information Processing Systems*, 2013, pp. 2814–2822.

Dropout in linear networks

- **Linear network**: all activations units correspond to the identity function.
- For a single training example we get

$$z^{[l]} = W^{[l]} D^{[l]} z^{[l-1]}.$$

- The expectation over all possible network realizations:

$$\mathbb{E}\{z^{[l]}\} = p_l W^{[l]} z^{[l-1]},$$

- p_l corresponds to the probability of keeping a unit on layer l.

Dynamics of a single linear unit

► Consider the error terms for the averaged ensemble network, and dropout:

$$E^{\text{ens}} = (y^{(i)} - p_l W^{[l]} x^{(i)})^2$$
$$E^{\text{d}} = (y^{(i)} - W^{[l]} D^{[l]} x^{(i)})^2.$$

► We want to minimize these cost functions.

1. Compute the gradients.

2. Take expectation over dropout realizations.

3. Obtain:

$$\mathbb{E}\{E^{\text{d}}\} = E^{\text{ens}} + \sum_{r=1}^{n_1} \frac{1}{2} \operatorname{var}(D^{[l]})(x_r^{(i)})^2 w_r^2$$

► Dropout corresponds to a regularized cost function of the ensemble network.

Dropout in logistic regression

▶ Single logistic unit with n inputs:

$$\sigma(z) = a^{[1]} = \frac{1}{1 + e^{-z}} \text{ and } z = w^T x.$$

▶ The **normalized weighted geometric mean** over al possible network configurations corresponds to a feedforward pass of the averaged weights.

$$\text{NWGM} = \frac{G}{G + G'} = \frac{1}{1 + e^{-\sum_j p w_j x_j}} = \sigma(pz).$$

▶ Definitions:

 – Total number of network configurations: $m = 2^n$.
 – $a_1^{[1]}, \dots, a_m^{[1]}$ possible outcomes.
 – Weighted geometric mean: $G = \prod_i (a_i^{[1]})^{P_i}$.
 – Weighted geometric mean of the complements $G' = \prod_i (1 - a_i^{[1]})^{P_i}$.

Dynamics of a single logistic unit

▶ The result from a single linear unit generalizes to a sigmoidal unit as well.

▶ The expected gradient of the dropout network:

$$\mathbb{E}\left\{\frac{\partial E^{\mathrm{d}}}{\partial w_i}\right\} \approx \frac{\partial E^{\mathrm{ens}}}{\partial w_i} + \lambda \sigma'(pz) x_i^2 \operatorname{var}(p) w_i.$$

▶ The expectation of the dropout gradient corresponds approximately to the gradient of the ensemble network plus a ridge regularization term.

Dropout in Deep Neural Networks

- ▶ Network of sigmoidal units.

- ▶ Output of unit i in layer l: $a_i^{[l]} = \sigma\left(\sum_j W_{ij}^{[l]} a^{[l-1]}\right)$

- ▶ *Normalized weighted geometric mean*:

$$\text{NWGM}(a_i^{[l]}) = \frac{\Pi_N (a_i^{[l]})^{P(N)}}{\Pi_N (1 - a_i^{[l]})^{P(N)} + \Pi_N (a_i^{[l]})^{P(N)}}$$

where N ranges over all possible configuration networks.

- ▶ Averaging properties of dropout:

$$\mathbb{E}\{a_i^{[l]}\} = \sigma\left(\mathbb{E}\left\{\sum_j W_{ij}^{[l]} a_i^{[l-1]}\right\}\right)$$

- ▶ **Take-out message:** the expected dropout gradient corresponds to an approximated ensemble network, regularized by an adaptive weight decay with a propensity for self-consistent variance minimization.

- ▶ **Convergence** can be understood via analysis of stochastic gradient descent.

Batch normalization

Problems of deep networks

- ▶ Adaptive reparametrization, motivated by the difficulty of training very deep models.
- ▶ Parameters from all layers are updated at the same time.
  - composition of many functions can have unexpected results because all functions have been changed simultaneously.
  - learning rate becomes difficult to tune.
- ▶ Consider a linear network with a single neuron per layer and single input.
- ▶ We update $w \leftarrow w - \epsilon g$, where $g = \nabla_w J$:

$$\hat{y} \leftarrow (w^{[1]} - \epsilon g^{[1]})(w^{[2]} - \epsilon g^{[2]}) \dots (w^{[L]} - \epsilon g^{[L]})x.$$

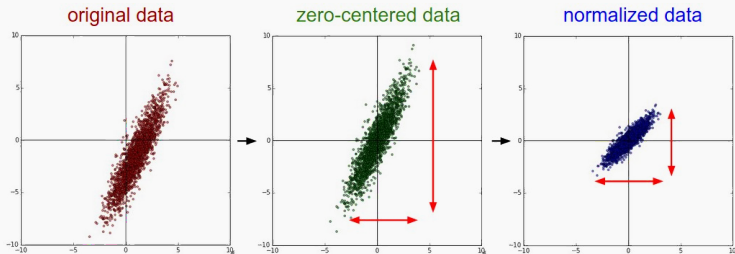- ▶ Previous update has many high order components, that can influence greatly the value of $\hat{y}$.

Input normalization

► The method is inspired by the normalization step normally applied to an input:

$$\widetilde{X}^{\{i\}} = \frac{X^{\{i\}} - \mu}{\sigma + \epsilon}$$

where $\epsilon = 10^{-8}$ is frequently used,

$$\mu = \frac{1}{m} \sum_r x^{\{i\}(r)}, \text{ and } \sigma^2 = \frac{1}{m} \sum_r (x^{\{i\}(r)} - \mu)^2.$$

original data   zero-centered data   normalized data

Batch normalization

▶ Batch normalization extends the concept to other hidden layers.

$$Z_{\mathrm{norm}}^{\{i\}[l]} = \frac{Z^{\{i\}[l]} - \mu^{\{i\}[l]}}{\sigma^{\{i\}[l]} + \epsilon}$$

where

$$\mu^{\{i\}[l]} = \frac{1}{m} \sum_r z^{\{i\}[l](r)}, \text{ and } (\sigma^{\{i\}[l]})^2 = \frac{1}{m} \sum_r (z^{\{i\}[l](r)} - \mu^{\{i\}[l]})^2.$$

▶ i refers to the mini-batch index; m to the number of elements.
   – the normalization depends on the minibatch.

▶ The outcome is rescaled with new parameters:

$$\widetilde{Z}^{\{i\}[l]} = \gamma^{\{i\}[l]} Z_{\mathrm{norm}}^{\{i\}[l]} + \beta^{\{i\}[l]},$$

where $\gamma^{\{i\}[l]}$ and $\beta^{\{i\}[l]}$ are incorporated in the learning process.

Batch normalization

- The scheme has the same expressive capabilities
  - setting $\beta^{\{i\}[l]} = \mu^{\{i\}[l]}$ and $\gamma^{\{i\}[l]} = \sigma^{\{i\}[l]}$.
- The weights from one layer do not affect the statistics (first and second order) of the next layer.
- The offsets $b^{[l]}$ become obsolete.
- **Testing:** a weighted average on all parameters:

$$\gamma_t = \beta\gamma_t + (1-\beta)\gamma^{\{i\}[l]}$$
$$\beta_t = \beta\beta_t + (1-\beta)\beta^{\{i\}[l]}$$
$$\mu_t = \beta\mu_t + (1-\beta)\mu^{\{i\}[l]}$$
$$\sigma_t = \beta\sigma_t + (1-\beta)\sigma^{\{i\}[l]}$$