11-695: AI Engineering
Customizing NNs II

LTI/SCS

Spring 2020

Outline

1. Choosing Metrics
2. Choosing Optimizers

How to apply a metric

Carnegie Mellon

tf_keras_metric

```
1  model.compile(optimizer=...,
2                loss=tf.keras.losses.SparseCategoricalCrossentropy(),
3                metrics=tf.keras.metrics.Accuracy())
```

- Apply a built-in loss from `tf.keras.metrics`

How to apply a metric

Carnegie Mellon

tf_keras_metric

```
1  model.compile(optimizer=...,
2                loss=tf.keras.losses.SparseCategoricalCrossentropy(),
3                metrics=tf.keras.metrics.Accuracy())
```

- Apply a built-in loss from `tf.keras.metrics`
- Convenient way: apply into `model.compile` function

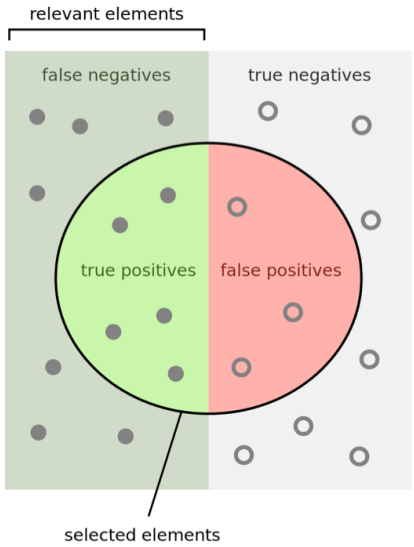
How to apply a metric

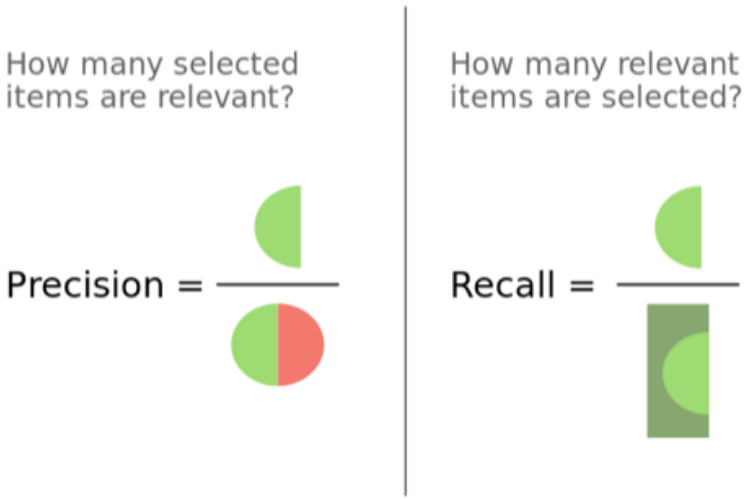
Carnegie Mellon

```
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='train_accuracy')

@tf.function
def train_step(images, labels):
  with tf.GradientTape() as tape:
    predictions = model(images)
    loss = loss_object(labels, predictions)
  gradients = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))

  train_loss(loss)
  train_accuracy(labels, predictions)

for epoch in range(EPOCHS):
  train_loss.reset_states()
  train_accuracy.reset_states()

  for images, labels in train_ds:
    train_step(images, labels)
```

- Other ways: like losses, but for `tf.keras.metrics`
- Also contain the most common losses

Common metrics for binary case[1]





- `TrueNegatives`, `TruePositives`, `FalseNegatives`, `FalsePositives`
- `Precision`: $\frac{\text{TP}}{\text{TP+FP}}$
- `Recall`: $\frac{\text{TP}}{\text{TP+FN}}$
- `BinaryAccuracy`: $\frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$

[1] Those metrics can be generalized for multi-classes as well.

Image credit: `wikipedia`

Binary cases: ROC Curve[2]

Carnegie Mellon

A			B			C		
TP=63	FP=28	91	TP=77	FP=77	154	TP=24	FP=88	112
FN=37	TN=72	109	FN=23	TN=23	46	FN=76	TN=12	88
100	100	200	100	100	200	100	100	200
TPR = 0.63			TPR = 0.77			TPR = 0.24		
FPR = 0.28			FPR = 0.77			FPR = 0.88		
PPV = 0.69			PPV = 0.50			PPV = 0.21		
F1 = 0.66			F1 = 0.61			F1 = 0.23		
ACC = 0.68			ACC = 0.50			ACC = 0.18		

- Consider A, B, and C, which one is the best?

[2] https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Image credit: wikipedia

Binary cases: ROC Curve

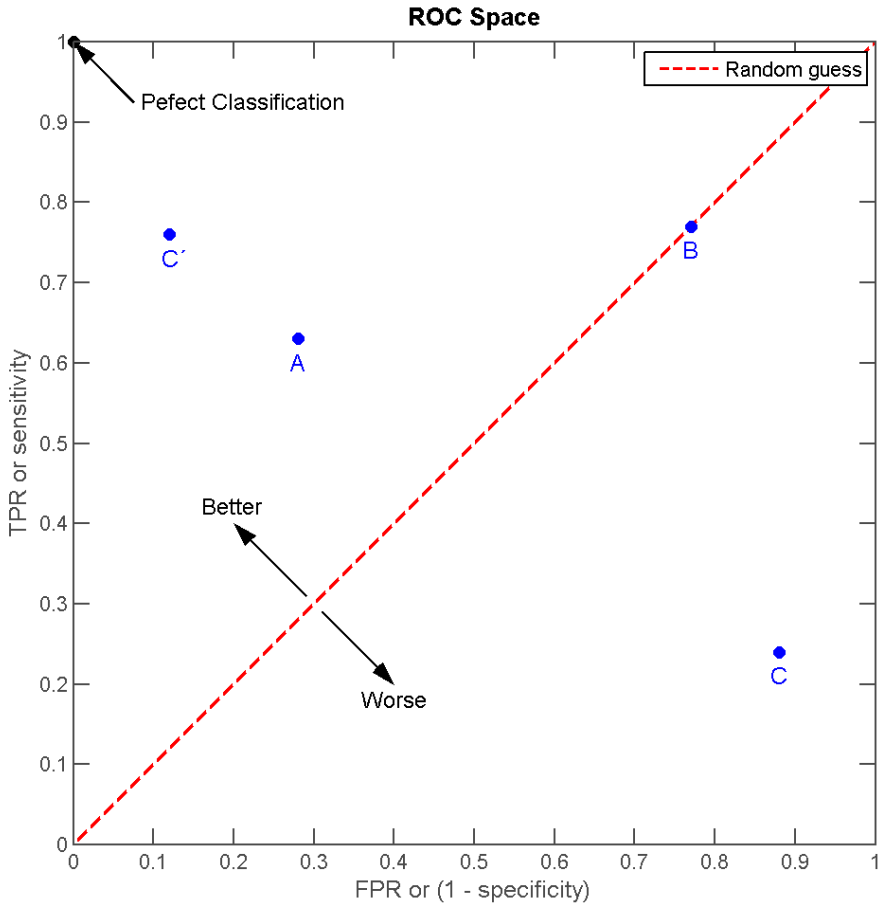
Carnegie Mellon

A	B	C	C′
TP=63 FP=28 91 FN=37 TN=72 109 100 100 200	TP=77 FP=77 154 FN=23 TN=23 46 100 100 200	TP=24 FP=88 112 FN=76 TN=12 88 100 100 200	TP=76 FP=12 88 FN=24 TN=88 112 100 100 200
TPR = 0.63	TPR = 0.77	TPR = 0.24	TPR = 0.76
FPR = 0.28	FPR = 0.77	FPR = 0.88	FPR = 0.12
PPV = 0.69	PPV = 0.50	PPV = 0.21	PPV = 0.86
F1 = 0.66	F1 = 0.61	F1 = 0.23	F1 = 0.81
ACC = 0.68	ACC = 0.50	ACC = 0.18	ACC = 0.82

- Consider A, B, and C, which one is the best?
- How about C', the inverse of C?

Image credit: wikipedia

Binary cases: ROC Curve

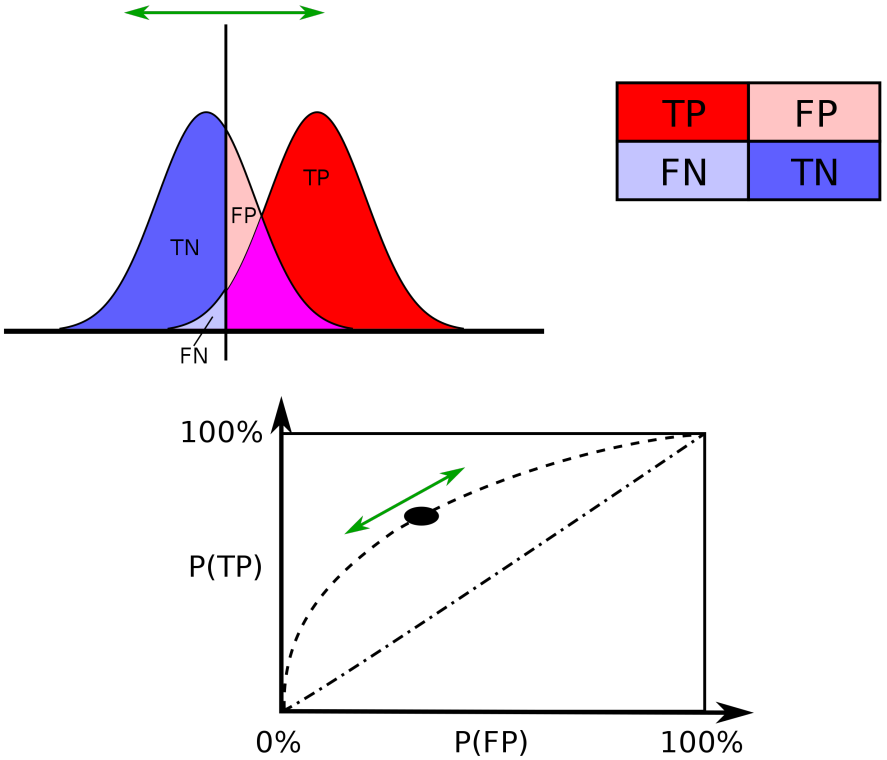


- How about C', the inverse of C?

Image credit: wikipedia

Binary cases: ROC Curve

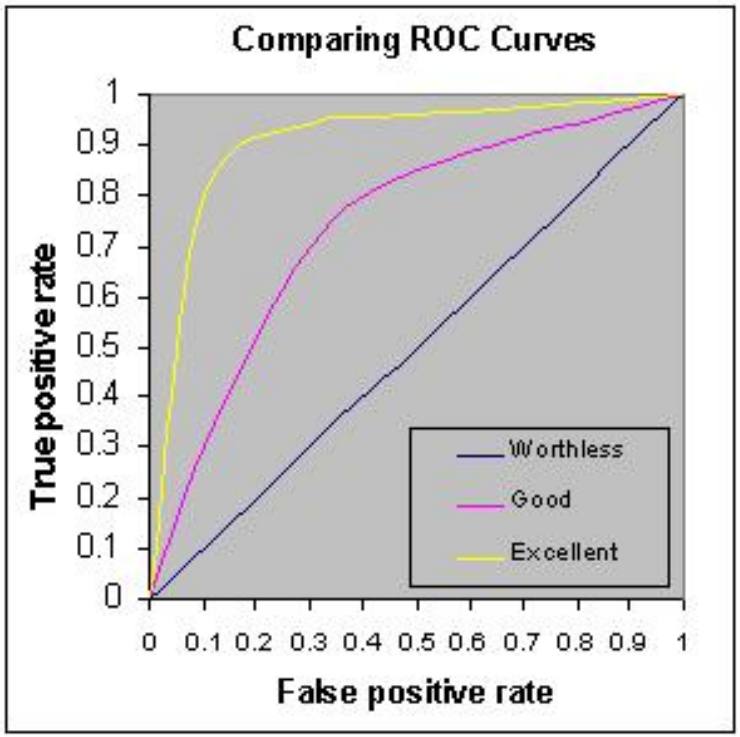
Carnegie Mellon

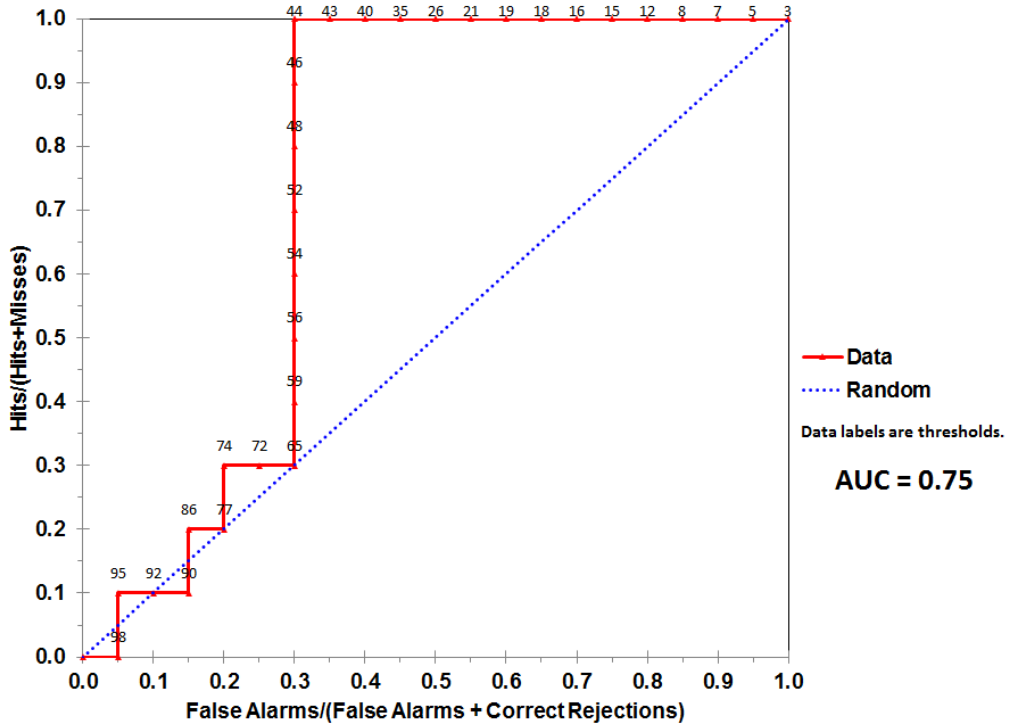


- Tradeoff between TPR $= \frac{\text{TP}}{\text{TP+FN}}$ (`Recall`) and FPR $= \frac{\text{FP}}{\text{FP+TN}}$
- These rates are *stable* metrics with classification errors

Image credit: `wikipedia`

Binary cases: AUC





- `AUC`: Area under (the ROC) curve, ranges $[0, 1]$
  - AUC is the "aggregation" of ROC over all thresholds
  - A single, stable metric combining others, and so is desirable

Image credit: `gim.unmc.edu` and wikipedia

Common metrics for general cases[3]

Carnegie Mellon

- `Accuracy`: across all classes, can be optionally weighted, for scalars
- `CategoricalAccuracy`: general cases, for one-hots labels
- `SparseCategoricalAccuracy`: for scalar labels
- `TopKCategoricalAccuracy`: for $k > 1$
- `SparseTopKCategoricalAccuracy`: for scalar labels

[3] More: `https://www.tensorflow.org/api_docs/python/tf/keras/metrics`

Outline

1 Choosing Metrics

2 Choosing Optimizers

Minibatch SGD

Carnegie Mellon

Minibatch SGD Algorithm

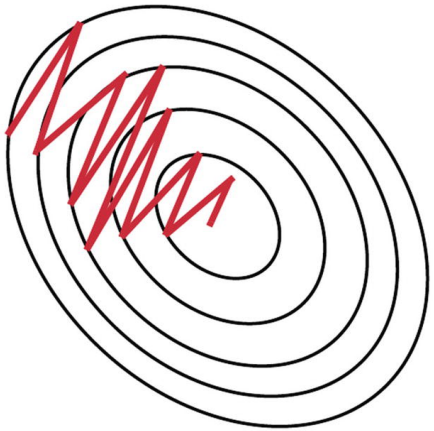
- Initialize $\theta^0 \in \mathbb{R}^D$; $t = 0$
- While not satisfactory do:
  - Randomly B samples from data
  - Calculate average gradients for this batch

$$\nabla^{(t)} = \sum_{i=1}^{B} \nabla_{\theta^{(t)}} \mathcal{L}(\mathbf{f}(\mathbf{x}^{(i)}; \theta^{(t)}), \mathbf{y}^{(i)})$$

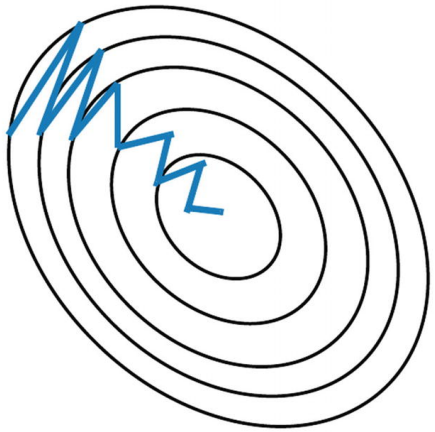
  - Update params: $\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \nabla^{(t)}$
  - $t \leftarrow t + 1$

- The most basic algorithm, but has several drawbacks

Momentum



Stochastic Gradient Descent **withhout** Momentum

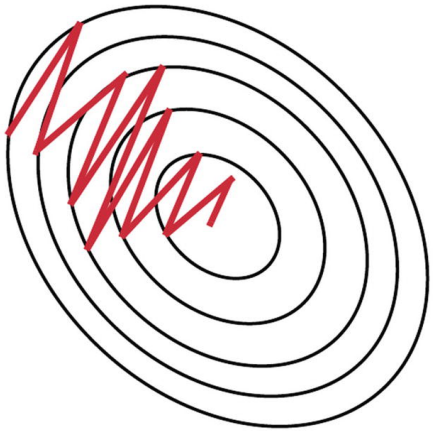


Stochastic Gradient Descent **with** Momentum

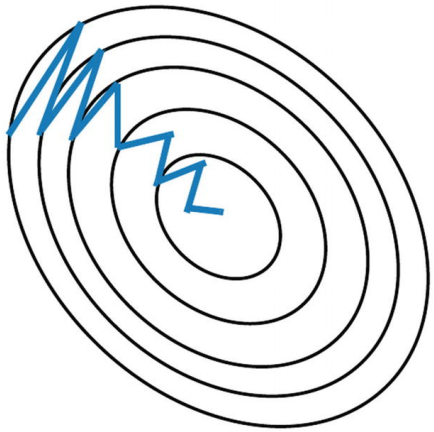
- SGD has some drawbacks
  - Being slow
  - Being a first-order, no idea about curvature (relating to Hessian)
  - High variance

Image credit: Ekaba Bisong

Momentum



Stochastic Gradient Descent **withhout** Momentum

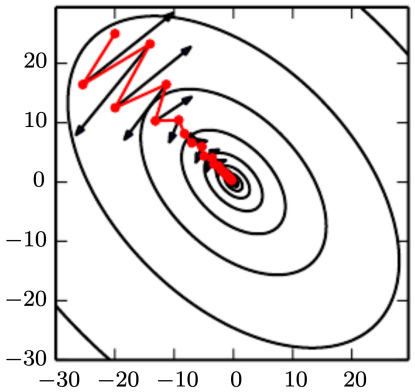


Stochastic Gradient Descent **with** Momentum

- SGD has some drawbacks
  - Being slow
  - Being a first-order, no idea about curvature (relating to Hessian)
  - High variance
- Momentum: keep historical gradients and so reduce "surprises"

Image credit: Ekaba Bisong

(Polyak's) Momentum[4]



- Keep a *exponentially smoothed averages* of gradients ∇_θ

$$\nabla^{(t)} = \sum_{i=1}^{B} \nabla_{\theta^{(t)}} \mathcal{L}(\mathbf{f}(\mathbf{x}^{(i)}; \theta^{(t)}), \mathbf{y}^{(i)}) \tag{1}$$

$$v^{(t+1)} \leftarrow \alpha \cdot v^{(t)} - \eta \nabla^{(t)} \tag{2}$$

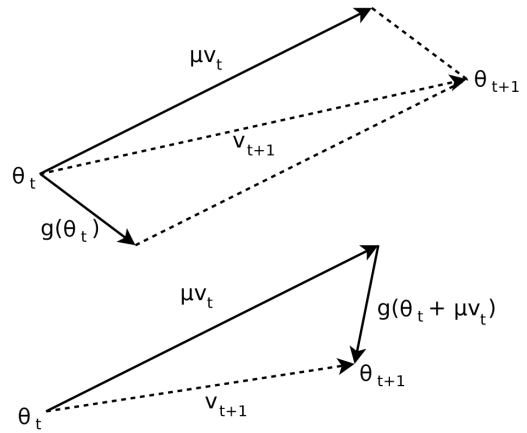
$$\theta^{(t+1)} \leftarrow \theta^{(t)} + v^{(t+1)} \tag{3}$$

- Normally, adapt α by increasing over training time

[4] https://vsokolov.org/courses/750/2018/files/polyak64.pdf

Image credit: Ian Goodfellow *et al.*

(Nesterov's) Momentum[5]



- Gradients in Eq. (1) is instead calculated at "future" weights:

$$\nabla^{(t)} = \sum_{i=1}^{B} \nabla_{\theta^{(t)}} \mathcal{L}(\mathbf{f}(\mathbf{x}^{(i)}; \theta^{(t)} + \alpha v^{(t)}), \mathbf{y}^{(i)}) \tag{4}$$

$$v^{(t+1)} \leftarrow \alpha \cdot v^{(t)} - \eta \nabla^{(t)}$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} + v^{(t+1)}$$

[5] http://mpawankumar.info/teaching/cdt-big-data/nesterov83.pdf

Image credit: Ilya Sutskever *et al.*

AdaGrad[6]

- Update to the learning rate η to get an adaptive params-based rate ψ:

$$\nabla^{(t)} = \sum_{i=1}^{B} \nabla_{\theta^{(t)}} \mathcal{L}(\mathbf{f}(\mathbf{x}^{(i)}; \theta^{(t)}), \mathbf{y}^{(i)}) \in \mathbb{R}^d$$

$$G_i^{(t)} = \sum_{i=1}^{t} \left(\nabla_i^{(t)}\right)^2 \tag{5}$$

$$\psi_i^{(t)} = \frac{\eta}{\sqrt{G_i^{(t)} + \epsilon}} \tag{6}$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \psi^{(t)} \odot \nabla^{(t)}$$

[6] http://jmlr.org/papers/v12/duchi11a.html

AdaGrad[6]

- Update to the learning rate η to get an adaptive params-based rate ψ:

$$\nabla^{(t)} = \sum_{i=1}^{B} \nabla_{\theta^{(t)}} \mathcal{L}(\mathbf{f}(\mathbf{x}^{(i)}; \theta^{(t)}), \mathbf{y}^{(i)}) \in \mathbb{R}^d$$

$$G_i^{(t)} = \sum_{i=1}^{t} \left(\nabla_i^{(t)}\right)^2 \tag{5}$$

$$\psi_i^{(t)} = \frac{\eta}{\sqrt{G_i^{(t)} + \epsilon}} \tag{6}$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \psi^{(t)} \odot \nabla^{(t)}$$

- Every learning rate update in Eq (6), make itself *shrink*,

[6] http://jmlr.org/papers/v12/duchi11a.html

AdaGrad[6]

- Update to the learning rate η to get an adaptive params-based rate ψ:

$$\nabla^{(t)} = \sum_{i=1}^{B} \nabla_{\theta^{(t)}} \mathcal{L}(\mathbf{f}(\mathbf{x}^{(i)}; \theta^{(t)}), \mathbf{y}^{(i)}) \in \mathbb{R}^d$$

$$G_i^{(t)} = \sum_{i=1}^{t} \left(\nabla_i^{(t)}\right)^2 \tag{5}$$

$$\psi_i^{(t)} = \frac{\eta}{\sqrt{G_i^{(t)} + \epsilon}} \tag{6}$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \psi^{(t)} \odot \nabla^{(t)}$$

- Every learning rate update in Eq (6), make itself *shrink*, until *not being good*.

[6] http://jmlr.org/papers/v12/duchi11a.html

RMSProp[7] (Hinton): a fix for AdaGrad

Carnegie Mellon

- Change: applying *momentum* for the accumulated squared grads:

$$\nabla^{(t)} = \sum_{i=1}^{B} \nabla_{\theta^{(t)}} \mathcal{L}(\mathbf{f}(\mathbf{x}^{(i)}; \theta^{(t)}), \mathbf{y}^{(i)}) \in \mathbb{R}^d$$

$$v_i^{(t)} = \alpha\, v_i^{(t-1)} + (1-\alpha)(\nabla_i^{(t)})^2 \tag{7}$$

$$\psi_i^{(t)} = \frac{\eta}{\sqrt{v_i^{(t)} + \epsilon}} \tag{8}$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \psi^{(t)} \odot \nabla^{(t)}$$

- The step size's denominator will not get inflated over time

[7] Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning. 2012

AdaDelta[8]: another fix for AdaGrad

Carnegie Mellon

- Change: applying *momentum* for the accumulated squared grads and *accumulated step size*:

$$\nabla^{(t)} = \sum_{i=1}^{B} \nabla_{\theta^{(t)}} \mathcal{L}(\mathbf{f}(\mathbf{x}^{(i)}; \theta^{(t)}), \mathbf{y}^{(i)}) \in \mathbb{R}^d$$

$$v_i^{(t)} = \alpha\, v_i^{(t-1)} + (1-\alpha)(\nabla_i^{(t)})^2$$

$$w_i^{(t-1)} = \gamma\, \psi_i^{(t-2)} + (1-\gamma)(\psi_i^{(t-1)})^2 \tag{9}$$

$$\psi_i^{(t)} = \frac{\sqrt{w_i^{(t-1)} + \epsilon}}{\sqrt{v_i^{(t)} + \epsilon}} \tag{10}$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \psi^{(t)} \odot \nabla^{(t)} \tag{11}$$

- Learning rate is eliminated

[8] https://arxiv.org/pdf/1212.5701.pdf

Adam[9]: Moments with Momentum

Carnegie Mellon

- Change: applying *momentum* for the accumulated squared grads and *grads*:

$$m_i^{(t)} = \beta_1 \, m_i^{(t-1)} + (1 - \beta_1) \nabla_i^{(t)} \tag{12}$$

$$v_i^{(t)} = \beta_2 \, v_i^{(t-1)} + (1 - \beta_2) (\nabla_i^{(t)})^2 \tag{13}$$

- De-bias the moments and update

$$\widehat{m}^{(t)} = m^{(t)} / (1 - \beta_1^t) \tag{14}$$

$$\widehat{v}^{(t)} = v^{(t)} / (1 - \beta_2^t) \tag{15}$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \frac{\eta}{\sqrt{\widehat{v}^{(t)} + \epsilon}} \odot \widehat{m}^{(t)} \tag{16}$$

[9] https://arxiv.org/pdf/1412.6980.pdf

More about Adam

- AdaMax (same paper): $v_i^{(t)}$ uses l_∞ norm instead of l_2
- Nadam[10]: Adam + Nesterov momentum
- Step size η is invariant to gradient magnitudes
- Sometimes it saturates and is worse than SGD with momentum[11]
- Some proofs are wrong, leading to AMSGrad[12]:

$$\hat{v}_i^{(t)} = \max(\hat{v}_i^{(t-1)}, v_i^{(t)})$$

- Adam with decoupled weight decay[13] (AdamW) shows big improvement
- Still generally the *best* in practice.

[10] https://openreview.net/pdf?id=OM0jvwB8jIp57ZJjtNEZ
[11] https://arxiv.org/pdf/1705.08292.pdf
[12] https://arxiv.org/pdf/1904.09237.pdf
[13] https://arxiv.org/pdf/1711.05101.pdf